Security Hardened Kernels for Linux Servers

Masters Thesis by Sowgandh Sunil Gadi
Thesis Director: Dr. Prabhaker Mateti

Outline

- Problem: Server security
- Thesis contribution
- Prevention of buffer overflow on IA-32 based Linux
- Prevention of known exploits
- Pruning the kernel
- Additions to the kernel
- Hardened kernels for servers
- Conclusion
- Demo

Server Security

- Servers are the main targets of cyber attacks
  - Cost, time and human resources
- Servers should deploy specialized kernels
  - Better performance and **security**
  - Attacker with root privileges should not be able to do much damage. Even root should not be able to change certain things once they are setup
- Prevention measures
  - Application level
  - Kernel level

Year	2000	2001	2002	2003
Incidents	21756	52658	82094	137529

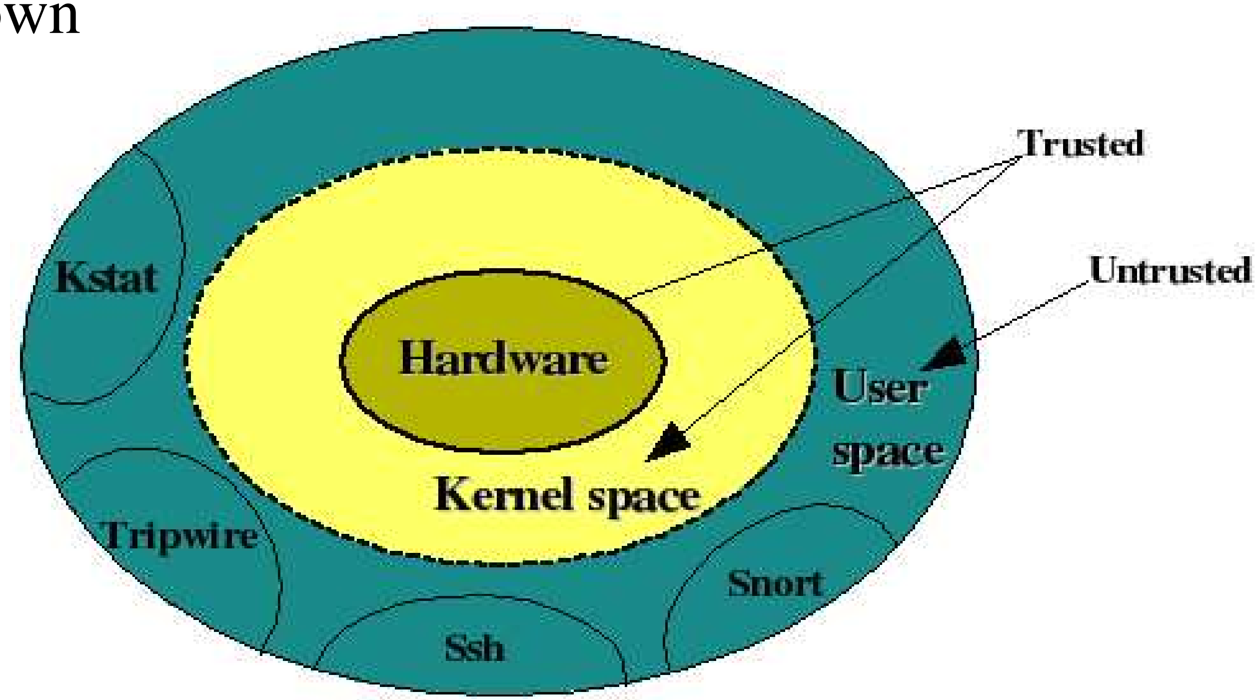
Application Level Security

- Cannot reduce the powers of a root user
- Cannot fight against an attacker with root privileges
- A bug in one application may lead to whole system compromise
- Can easily be backdoored
- Code auditing of millions of lines of code is slow, expensive and cannot be fully automated
  - Buffer overflow attack is known for more than 10 years

Kernel Level Security

A large number of exploits can be prevented by

- Redesigning
- Additions
- Pruning down



Thesis Contribution

- Ready to be deployed security hardened kernels
- Tech docs fully explaining how the security enhancements work
- Techniques of pruning a kernel both at build time and at run time
- Additions of subsystems that fortify a kernel
- New system calls that help the above

Thesis Contribution: Four kernels

- Our main goal is to develop security hardened kernels for server systems
- We built specialized kernels ready-to-be deployed
  - Anonymous FTP server
  - Web server
  - Mail server
  - File server

Thesis Contribution: Unified Patch

- A unified source code patch against Linux kernel 2.4.23 which provides several security enhancements

- Focused on i386: stable platform for Linux development, familiarity and availability of equipment

- Prevents known exploits
  - Chroot jail breaking
  - Temporary file race conditions
  - File descriptor leakage
  - Arbitrary file execution
  - LKM rootkits
  - `/dev/kmem` rootkits

Thesis Contribution: Pruning of Kernels

- Disabling selected System calls
- Disabling selected Capabilities
- Disabling selected Memory devices
- Freezing ext2 file system attributes
- Freezing Network and routing table configuration

Thesis Contribution: Additions of subsystems

- Kernel Logger
- Kernel Integrity Checker
- Trusted Path Mapping

Thesis Contribution: New System Calls

1. Freeze_syscalls
2. cap_elim
3. freeze_network
4. Kic
5. Klogger
6. tpm
7. no_overwrite_ftp

Buffer Overflow Patches

- We reviewed, in detail, five independent patches which prevent buffer overflow attacks
  - OWL (May 2003)
  - Segmented-PAX (May 2003)
  - KNOX (August 2003)
  - RSX (May 2003)
  - Paged-PAX (May 2003)
- We show that OWL and RSX are ineffective
- We brought to attention that Linux on IA-32 does not use segmentation wisely
- We provide performance impact details

Thesis Contribution: Tech Docs

- Open source developers rarely provide documentation
- No technical explanations of
  - Prevention techniques
  - Limitations of patches
  - Side effects of patches
- We fill this gap. The thesis contains technical documentation explaining the inner working of all our patches

Contribution of Technical Justifications

- Existing patches we examined
- Design and implementation of patches we introduced
- Root causes of exploits
- Exploitable features with examples
- Prevention techniques and their limitations

Background

- IA-32
  - Segmentation and Paging
  - Translation lookaside buffers
  - Pagefault exception
  - General Protection error
- Linux
  - Memory mapping of processes
  - Kernel memory layout
  - ELF binary format
  - Capabilities
  - System call table

IA-32 Segmentation

- Running image of a process is a collection of segments
- Depending on needs of a segment containing code, data, stack, or heap of a program, the OS is expected to assign different protection features, such as read-only, read-plus-write-but-no-execute
- GDT and LDT contains the descriptors of the segments

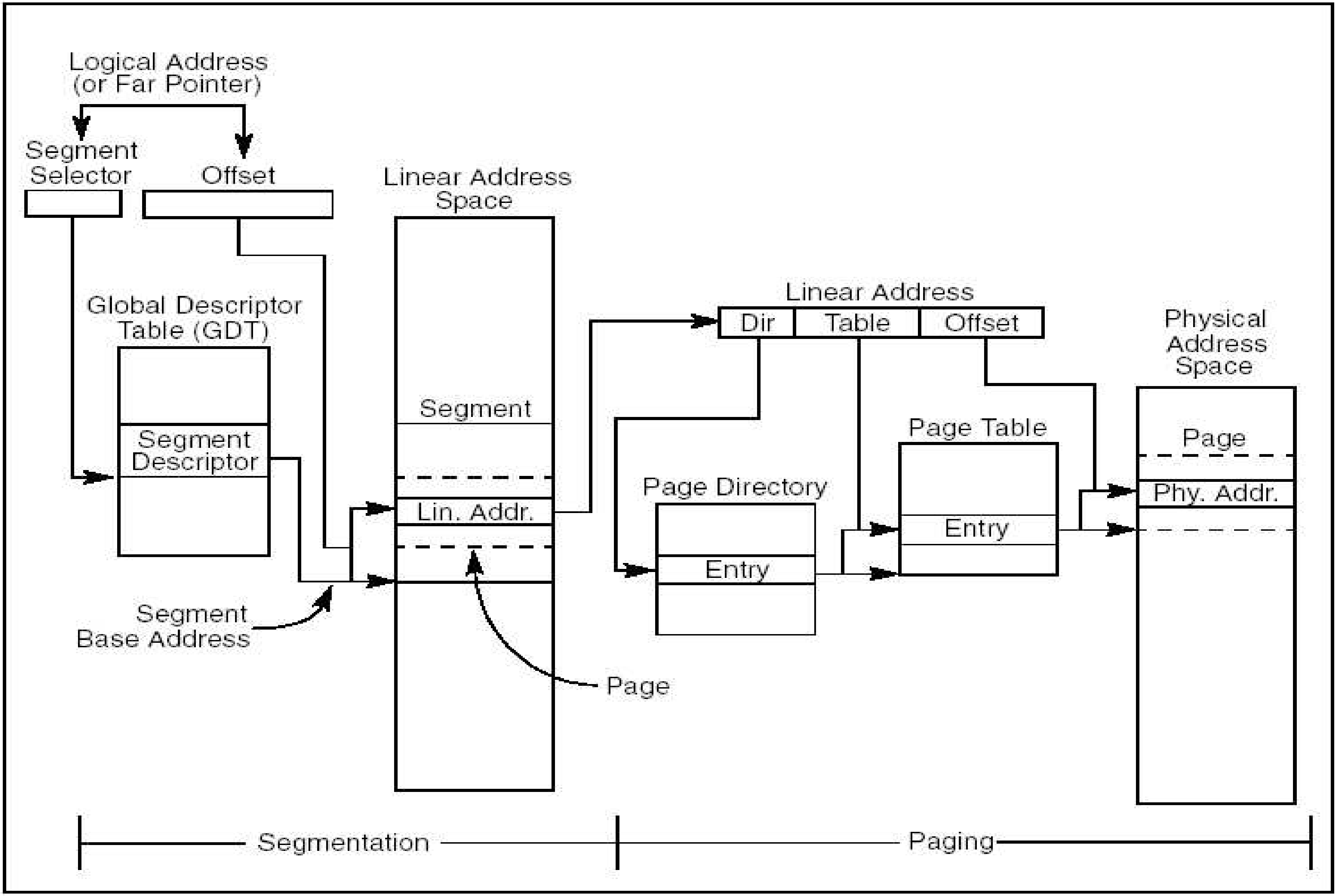
IA-32 Segmentation

- Types of data segment
  - Read only
  - Read/write
- Types of code segment
  - Execute only
  - Execute/read
- Basic Flat Model
  - Hides segmentation mechanism
  - All segments have same base address 0 and segment size 4 GB
  - This model is used in all major operating systems running on IA-32 e.g., Linux, Windows NT/2000/XP, OpenBSD

IA-32 Paging

- Maps pages in linear address space to frames in physical memory
- The entries of page directories and page tables have the same structure
- Each entry includes the fields:
  - User/supervisor flag
  - Read/write flag
- Readable implies Executable; Writable implies Readable
- No explicit flag controlling whether a page contains executable code

Segmentation and Paging



Translation Lookaside Buffers

- Most recently used page-table entries (PTEs) and page-directory entries (PDEs) are stored on on-chip caches called Translation Lookaside Buffers
- P6 family and Pentium processors have separate TLBs for data and instruction caches (DTLB and ITLB)
- Most paging is performed using the contents of the TLBs
- Whenever a PTE or PDE is changed the OS must immediately invalidate the corresponding entry in TLB so that it can be updated next time it is referenced

Page Fault Exception

- A page fault may occur for following reasons
  - When the page is not present in the memory
  - When process attempts to write to a read only page
  - When process does not have sufficient privileges to access the page

- Page fault handler
  - Can recover from page-not-present situation
  - It can also recover from a write attempt to a read only page
  - But privilege violation is not correctable

Error Code for Page Fault

Page Fault Handler can access Error Code and CR2 register in handling the exception.

- Error Code

P	0	The fault was caused by a non-present page.
	1	The fault was caused by a page-level protection violation.
W/R	0	The access causing the fault was a read.
	1	The access causing the fault was a write.
U/S	0	The access causing the fault originated when the processor was executing in supervisor mode.
	1	The access causing the fault originated when the processor was executing in user mode.

- CR2 register contents = the 32-bit address that generated the page fault.

General Protection Error

Processor detects around 30 different kinds of violations by raising a general protection error. They include

- Exceeding the segment limit
- Reading from an execute-only segment
- Exceeding the segment limit when referencing a descriptor table

Segmentation in Linux

- Linux uses Basic Flat Model of segmentation
- All the processes use Global Descriptor Table (GDT)
- Virtual address = Linear address
- Protection between operating system and application code and data is provided by page-level protection mechanism

Segmentation in Linux

GDT of Linux

Segment	Base	Limit	Mode	rwx
Kernel code	0	0xffffffff	Kernel	r-x
Kernel data	0	0xffffffff	Kernel	rw-
User code	0	0xffffffff	User	r-x
User data	0	0xffffffff	User	rw-

Memory Maps of Processes

/proc/*/maps of /bin/bash

```
address space     perm offset   dev   inode      pathname
08048000-080d0000 r-xp 00000000 03:01 217766     /bin/bash
080d0000-080d7000 rw-p 00087000 03:01 217766     /bin/bash
080d7000-08132000 rwxp 00000000 00:00 0
40000000-40015000 r-xp 00000000 03:01 215881     /lib/ld-2.2.4.so
40015000-40016000 rw-p 00014000 03:01 215881     /lib/ld-2.2.4.so
40016000-40017000 rw-p 00000000 00:00 0
...
40034000-40169000 r-xp 00000000 03:01 217078     /lib/libc-2.2.4.so
40169000-4016e000 rw-p 00134000 03:01 217078     /lib/libc-2.2.4.so
4016e000-40172000 rw-p 00000000 00:00 0
...
bfffa000-c0000000 rwxp ffffb000 00:00 0
```

Memory Maps of Processes

/proc/1/maps of /sbin/init

```
08048000-0804f000 r-xp 00000000 03:01 29220 /sbin/init
0804f000-08051000 rw-p 00006000 03:01 29220 /sbin/init
08051000-08055000 rwxp 00000000 00:00 0
40000000-40015000 r-xp 00000000 03:01 215881 /lib/ld-2.2.4.so
40015000-40016000 rw-p 00014000 03:01 215881 /lib/ld-2.2.4.so
40016000-40017000 rw-p 00000000 00:00 0
4002c000-40161000 r-xp 00000000 03:01 217078 /lib/libc-2.2.4.so
40161000-40166000 rw-p 00134000 03:01 217078 /lib/libc-2.2.4.so
40166000-4016a000 rw-p 00000000 00:00 0
bfffe000-c0000000 rwxp fffff000 00:00 0
```

ELF Binary Format

ELF segments of `/sbin/init`

```
Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align

  PHDR           0x000034 0x08048034 0x08048034 0x000c0 0x000c0 R E 0x4
  INTERP         0x0000f4 0x080480f4 0x080480f4 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x06e2f 0x06e2f R E 0x1000
  LOAD           0x006e40 0x0804fe40 0x0804fe40 0x004d8 0x006b4 RW  0x1000
  DYNAMIC        0x007248 0x08050248 0x08050248 0x000d0 0x000d0 RW  0x4
  NOTE           0x000108 0x08048108 0x08048108 0x00020 0x00020 R   0x4
```

System Call Table

- System call table is a data structure containing the addresses of system call routines
- *n*th entry contains the service routine address of the system call having number *n*
- 270 entries in Linux kernel 2.4.23
  - Only 224 are implemented
  - The rest are obsolete, or yet to be implemented

Linux Capabilities

- A capability is a credential for a process which asserts that the process is allowed to perform a specific operation or a class of operations
  - e.g., `cap_sys_mod` for inserting and deleting modules
- Different from traditional "Superuser versus normal user"
- No support from file system
  - Root process has all the capabilities
  - Normal user process has no capabilities
- There are 29 capabilities in Linux kernel 2.4.23
- System calls: `capget, capset`

Prevention of Buffer Overflow Attacks on IA-32 Based Linux

- What is buffer overflow?
- Prevention techniques
- Kernel patches
  - OWL
  - Segmented-PAX
  - KNOX
  - RSX
  - Paged-PAX

Buffer Overflow Attack

- By exploiting a buffer overflow error in a root-privileged program, the return address or a function pointer is overwritten with that of shell-code

```
void main(int argc, char *argv) {
   char buffer[512];
   if(argc > 1)
       strcpy(buffer,argv[1]);
}
```

- Most common attack of the decade

Stack after `ret` is overwritten

```
bottom of  DDDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF     top of
memory     89ABCDEF0123456789AB  CDEF  0123  4567  89AB  CDEF     memory
           buffer                sfp   ret   a     b     c

<------   [JJSSSSSSSSSSSSSSCCss][ssss][0xD8][0x01][0x02][0x03]
           ^|^             ^|             |
           |||_____________||_____________| (1)
       (2)  ||_____________||
             |______________| (3)
top of                                                            bottom of
stack                                                                 stack
```

Buffer Overflow Attack

WRIGHT STATE UNIVERSITY

- Stack overflow
  - A local buffer on stack is overflowed with executable instructions and return address is overwritten to point to the buffer itself
- Heap overflow
  - A heap overflow in dynamically allocated memory
- Function pointer overwrite
  - Overflow buffer to point the return address or a function pointer to a function in `libc`, usually `system()`

Buffer Overflow Prevention

- Compile-time prevention techniques
  - Static checking at compile-time e.g., Splint compiler
- Execution-time prevention techniques
  - Application level
    - StackGuard, Libsafe
  - Kernel level
    - Make all non-code pages non-executable using segmentation, paging or virtual memory techniques

Secure Kernel Modifications

- Using segmentation
  - OWL – Solar Designer, Open Wall Linux Secure kernel patch
  - Segmented-PAX – PAX Team, Page execution
  - KNOX – Purczynski
  - RSX – Starzetz, Runtime address Space extender

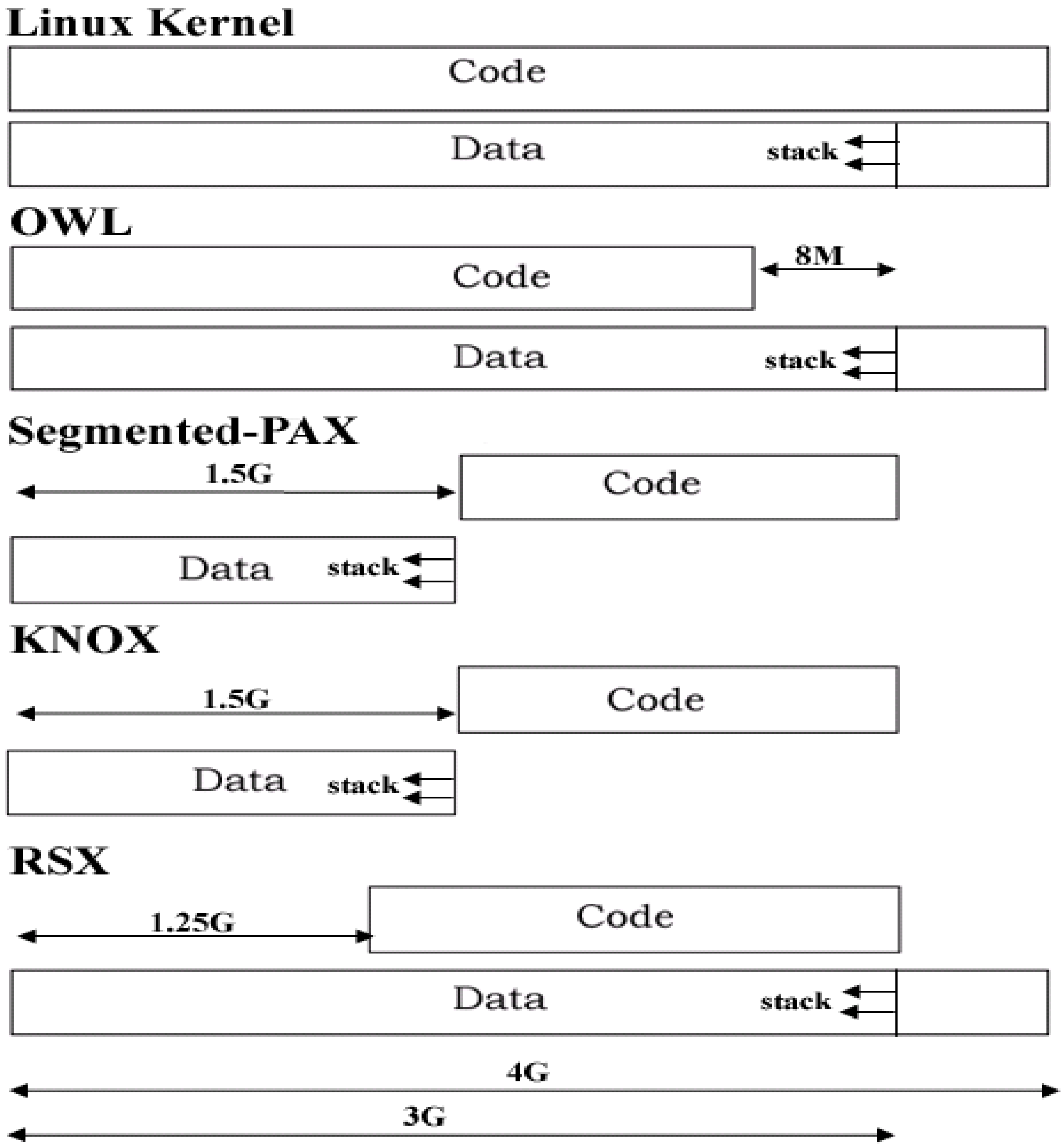
- Using paging and virtual memory techniques
  - Paged-PAX – PAX Team, Page execution

Secure Kernel Modifications (cont.)

Main idea of segmentation based modifications

- Make user code and data segments disjoint by adjusting the GDT and LDT tables
- Corresponding changes are made in functions handling `mmap()`, `munmap()`, `mremap()`, `mprotect()` and `mlock()`

Code and Data Segments of Patched Kernels



OWL

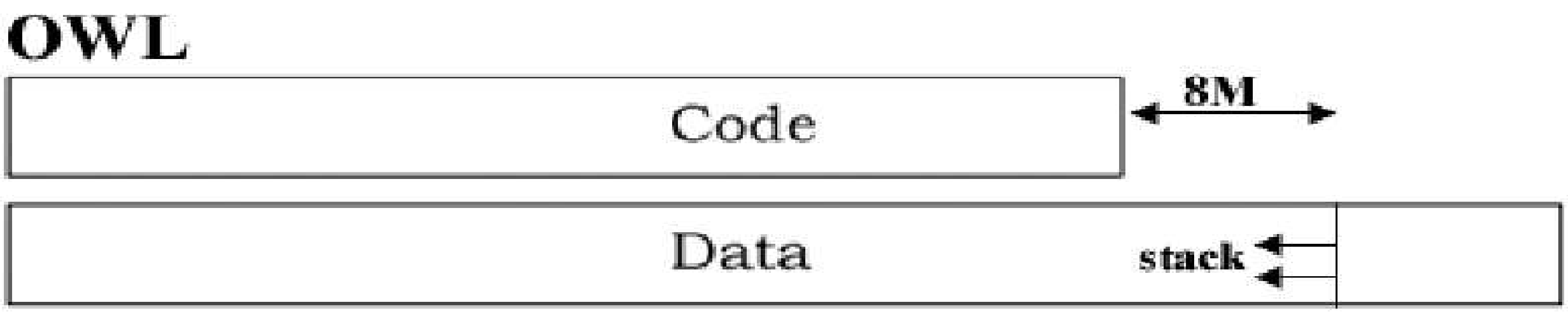
- The limit of the user segment is decreased so that certain portion of stack would not overlap with the code segment
- GDT of OWL patched Linux

Segment	Base	Limit	Mode	rwx
User code	0	0xbf7fffff	User	r-x
User data	0	0xffffffff	User	rw-

- OWL can prevent stack execution only. Heap execution cannot be prevented.
- An attempt to execute an instruction located on the first 8 MB size of stack will have an address outside the code segment and general protection error occurs

Breaking OWL

- Any user can increase the max stack size for his processes using system call `setrlimit` and if the stack increases above 8 MB it overlaps with code segment
- So instructions located after 8 MB can be executed



Segmented-PAX

- The user code and data segments are made completely disjoint
- For every text region in data segment there is a corresponding anonymous region in code segment
- Anonymous regions in code segment and text regions in data segment are backed by the same physical memory frames

Segment	Base	Limit	Mode	rwx
User code	0x60000000	0x5fffffff	User	r-x
User data	0	0x5fffffff	User	rw-

PAX bash maps

```
08048000-080d0000 r-xp 00000000 03:01 217766 /bin/bash
080d0000-080d7000 rw-p 00087000 03:01 217766 /bin/bash
080d7000-08132000 rw-p 00000000 00:00 0
20000000-20015000 r-xp 00000000 03:01 215881 /lib/ld-2.2.4.so
20015000-20016000 rw-p 00014000 03:01 215881 /lib/ld-2.2.4.so
20016000-20017000 rw-p 00000000 00:00 0
20017000-20019000 r-xp 00000000 03:01 155278 /usr/.../ISO8859-1.so
20019000-2001a000 rw-p 00001000 03:01 155278 /usr/.../ISO8859-1.so
2001a000-2001b000 r--p 00000000 03:01 68765  /usr/.../LC_NUMERIC
2001b000-20021000 r--p 00000000 03:01 68855  /usr/.../LC_COLLATE
20021000-20022000 r--p 00000000 03:01 68715  .../SYS_LC_MESSAGES
2002c000-2002f000 r-xp 00000000 03:01 215878 .../libtermcap.so.2.0.8
2002f000-20030000 rw-p 00002000 03:01 215878 .../libtermcap.so.2.0.8
20030000-20032000 r-xp 00000000 03:01 217082 /lib/libdl-2.2.4.so
20032000-20034000 rw-p 00001000 03:01 217082 /lib/libdl-2.2.4.so
20034000-20169000 r-xp 00000000 03:01 217078 /lib/libc-2.2.4.so
20169000-2016e000 rw-p 00134000 03:01 217078 /lib/libc-2.2.4.so
2016e000-20172000 rw-p 00000000 00:00 0
20172000-2017b000 r-xp 00000000 03:01 217103 .../libnss_....so
2017b000-2017d000 rw-p 00008000 03:01 217103 .../libnss_....so
2017d000-201a8000 r--p 00000000 03:01 68856  /usr/.../LC_CTYPE
5fffa000-60000000 rw-p ffffb000 00:00 0
68048000-680d0000 r-xp 00000000 00:00 0
80000000-80015000 r-xp 00000000 00:00 0
80017000-80019000 r-xp 00000000 00:00 0
80019000-8001a000 ---p 00002000 00:00 0
8002c000-8002f000 r-xp 00000000 00:00 0
8002f000-80030000 ---p 00003000 00:00 0
80030000-80032000 r-xp 00000000 00:00 0
80032000-80034000 ---p 00002000 00:00 0
80034000-80169000 r-xp 00000000 00:00 0
80169000-80172000 ---p 00135000 00:00 0
80172000-8017b000 r-xp 00000000 00:00 0
8017b000-8017d000 ---p 00009000 00:00 0
```

Segmented-PAX

Disadvantages

- The total size of virtual memory areas for a process is limited to 1.5 GB
- Performance Loss
  - While creating and initializing text memory regions
  - Handling page faults occurred in code segment
  - GDTR is reloaded for every context switch

KNOX

- User code and data segments are made completely disjoint
- Memory region mapping is same as in standard kernel
- For every text region mapped in data segment, page tables are setup for the corresponding addresses in code segment
- The page tables of text regions in data segment and those in code segment are backed up by same page frames
- The process memory descriptor is never aware of the address locations accessed in code segment

Segment	Base	Limit	Mode	rwx
User code	0x60000000	0x5fffffff	User	--x
User data	0	0x5fffffff	User	rw-

RSX

- RSX is a Loadable Kernel Module
- RSX shifts the base address of the code segment from 0 to 0x50000000
- Data segment range is unchanged
- Every text region is mapped both in data and code segment
- Unlike Segmented-PAX, text regions in code segment and data segment are not backed up by same physical frames

Segment	Base	Limit	Mode	rwx
User code	0	0xffffffff	User	r-x
User data	0	0xffffffff	User	rw-
RSX User code	0x50000000	0x6fffffff	User	r-x

RSX bash maps

```
08048000-080d0000 r-xp 00000000 03:01 217766     /bin/bash
080d0000-080d7000 rw-p 00087000 03:01 217766     /bin/bash
080d7000-0812d000 rwxp 00000000 00:00 0
40000000-40015000 r-xp 00000000 03:01 215881     /lib/ld-2.2.4.so
40015000-40016000 rw-p 00014000 03:01 215881     /lib/ld-2.2.4.so
40016000-40017000 rw-p 00000000 00:00 0
40017000-40019000 r-xp 00000000 03:01 155278     /usr/lib/gconv/ISO8859-1.so
40019000-4001a000 rw-p 00001000 03:01 155278     /usr/lib/gconv/ISO8859-1.so
4001a000-4001b000 r--p 00000000 03:01 68765      /usr/share/locale/en_US/LC_NUMERIC
4001b000-40021000 r--p 00000000 03:01 68855      /usr/share/locale/ISO-8859-1/LC_COLLATE
40021000-40022000 r--p 00000000 03:01 68715      /usr/share/locale/en_US/LC_MESSAGES/SYS_LC
4002c000-4002f000 r-xp 00000000 03:01 215878     /lib/libtermcap.so.2.0.8
4002f000-40030000 rw-p 00002000 03:01 215878     /lib/libtermcap.so.2.0.8
40030000-40032000 r-xp 00000000 03:01 217082     /lib/libdl-2.2.4.so
40032000-40034000 rw-p 00001000 03:01 217082     /lib/libdl-2.2.4.so
40034000-40169000 r-xp 00000000 03:01 217078     /lib/libc-2.2.4.so
40169000-4016e000 rw-p 00134000 03:01 217078     /lib/libc-2.2.4.so
4016e000-40172000 rw-p 00000000 00:00 0
40172000-4017b000 r-xp 00000000 03:01 217103     /lib/libnss_files-2.2.4.so
4017b000-4017d000 rw-p 00008000 03:01 217103     /lib/libnss_files-2.2.4.so
4017d000-401a8000 r--p 00000000 03:01 68856      /usr/share/locale/ISO-8859-1/LC_CTYPE
58048000-580d0000 r-xp 00000000 03:01 217766     /bin/bash
90000000-90015000 r-xp 00000000 03:01 215881     /lib/ld-2.2.4.so
90017000-9001a000 r-xp 00000000 03:01 155278     /usr/lib/gconv/ISO8859-1.so
9002c000-90030000 r-xp 00000000 03:01 215878     /lib/libtermcap.so.2.0.8
90030000-90034000 r-xp 00000000 03:01 217082     /lib/libdl-2.2.4.so
90034000-90172000 r-xp 00000000 03:01 217078     /lib/libc-2.2.4.so
90172000-9017d000 r-xp 00000000 03:01 217103     /lib/libnss_files-2.2.4.so
bfffa000-c0000000 rwxp ffffb000 00:00 0
```

RSX

How does RSX prevent attacks?

- Virtual address is not equal to linear address
- Stack Execution: If attacker tries to execute instructions on stack the General Protection Error occurs
- Heap Execution: The heap and BSS execution are detected in page fault handler

RSX Disadvantages

- Total size of virtual memory areas of the process is limited to 0x50000000 - 0xc0000000. Virtual address space is wasted.
- More physical frames are required by each process
- Performance Loss
  - RSX reloads CS register for each `exec()`
  - While creating and initializing text regions

Breaking RSX

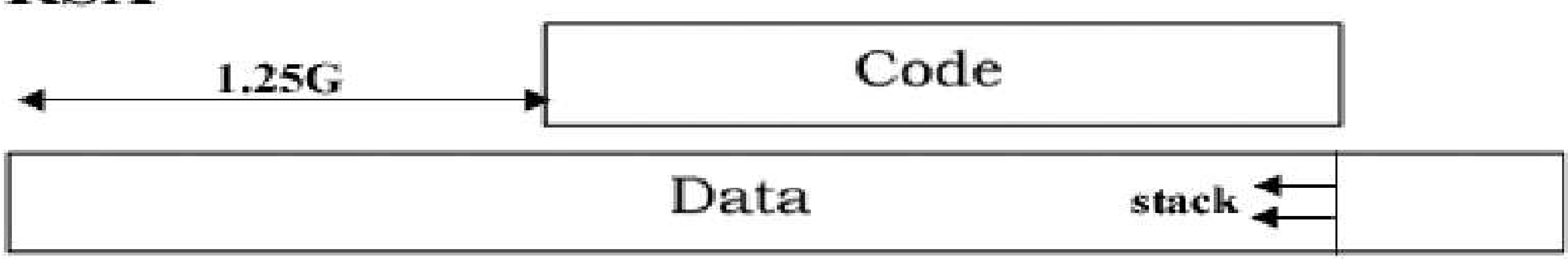
In the “shellcode”

- While overwriting the return address subtract base address of code segment
- While pushing the arguments of `execve`, add base address of code segment

```
bottom of   DDDDDDDDEEEEEEEEEEEE  EEEE  FFFF  FFFF  FFFF  FFFF     top of
memory      89ABCDEF0123456789AB  CDEF  0123  4567  89AB  CDEF     memory
            buffer                sfp   ret   a     b     c

<------    [JJSSSSSSSSSSSSSSCCss][ssss][0xD8][0x01][0x02][0x03]
            ^|^             ^|            |
            |||_____________||____________|  (1)
        (2)  ||_____________||
              |______________|  (3)
top of                                                          bottom of
stack                                                               stack
```

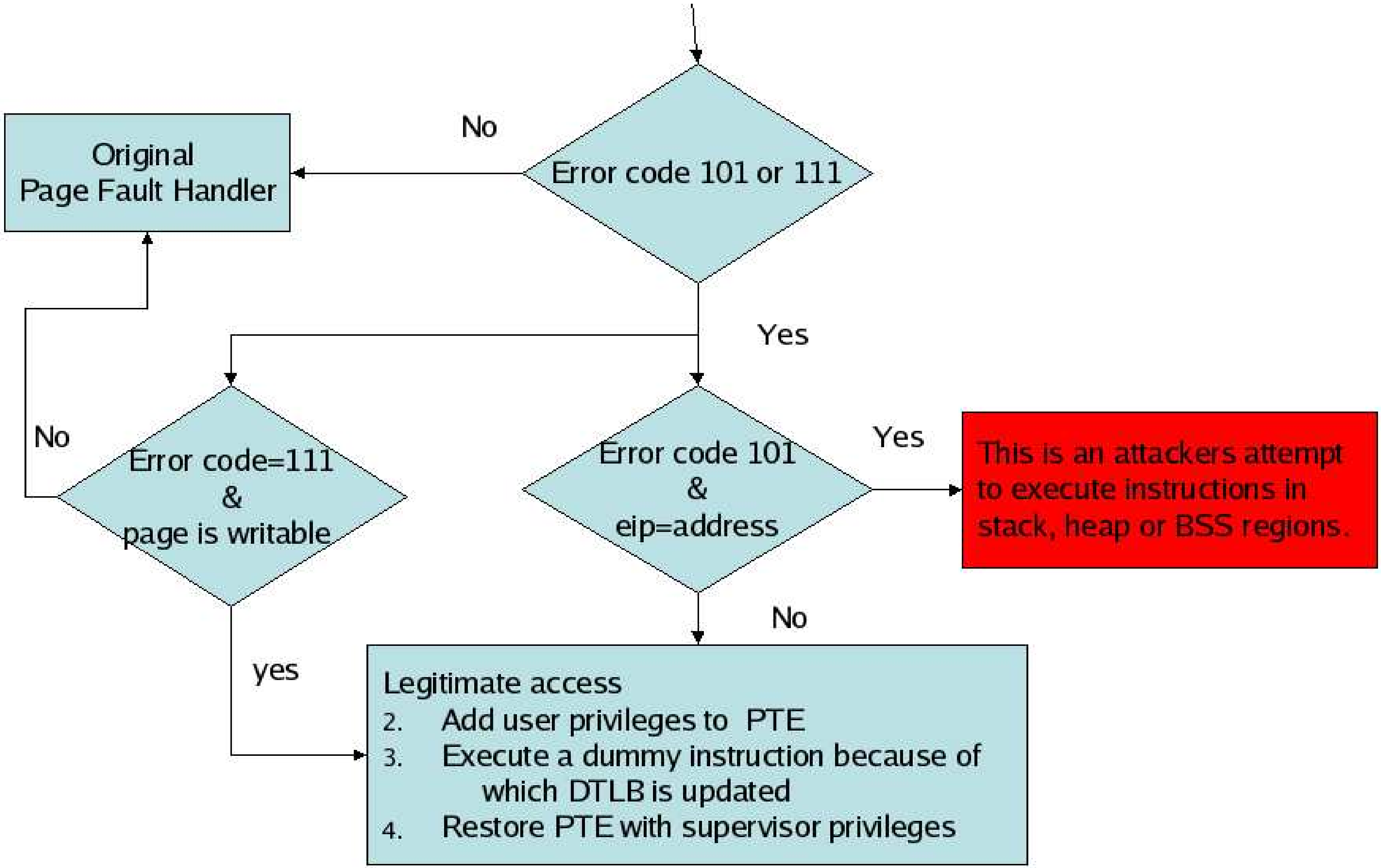
RSX



Paged-PAX

- No changes to GDT
- PAX pagefault handler monitors every address location of data regions
- PAX deliberately sets the page table entries for data regions of user process with supervisor privileges. So when process, in user mode, access them page fault occurs
- PAX extends the page fault handler to handle this

PAX Page Fault handler



Paged-PAX Performance

- PAX generates page faults for every access to a unique address in stack, heap and BSS if the page table entry of the address is not in DTLB
- Because of PAX generated page faults, performance suffers seriously

Pagefaults with Paged-PAX

argv[1]	user	sys	pfpatched	pfstd	pfpax
1	0.00	0.00	686	354	332
2	0.00	0.00	942	354	588
3	0.01	0.01	1200	354	846
257	0.02	0.05	66478	354	66124
100000	5.71	17.86	25600786	351	25600435

Paxtest.c

```
int main (int argc, char *argv[])
{
   char *buf;
   int i, j, limit = 100000;
   if (argc == 2) limit = atoi(argv[1]);
   buf = (char *) malloc(4096 * 257);
   for (j = 0; j < limit; j++)
   {
       for (i = 0; i < 257; i++)
        buf[i * 4096] = 'a';
   }
   return (0);
 }
```

Micro benchmark Results

- Lmbench benchmark results

Kernel	fork+exit	fork+exec+exit	fork+sh
Standard 2.4.23	142.4571	724.5	3632
OWL 2.4.23	144.1111	726.3750	3604.5
Paged-PAX 2.4.23	194.8846	802.5714	3969.5
Segm-PAX 2.4.23	203.0385	949.5	4157.5
Standard 2.4.5	141.0270	680.6250	4924
RSX 2.4.5	163.125	783.5714	5126
Standard 2.2.20	112.6531	603.8889	17820
KNOX 2.2.20	124.2273	667.6250	17801

Times in microseconds

Kernel	mmapx	mmapw	pfx	pfw
Standard 2.4.23	12	12	2	1
OWL 2.4.23	5.664	5.872	2	1
PAX-Paged 2.4.23	13	13	2	2
PAX-Segm 2.4.23	23	23	3	3
Standard 2.4.5	27	27	2	2
RSX 2.4.5	35	33	2	2
Standard 2.2.20	8.344	8.423	451	1
KNOX 2.2.20	8.251	8.357	452	1

Times in microseconds

Prevention of Buffer Overflow

- Proper use of segmentation prevents a large class of buffer overflow attacks
  - Code and data segments should be completely disjoint
- Paging based patch – more performance loss
- Segmentation based patches
  - Total virtual memory is reduced
  - Performance loss while mapping regions and page fault handling
- Open source code listings of programs would not be enough. Proper documentation of patch code is required.
- We provide an independent audit & quality analysis of kernel modifications – the authors did not do it

Why Did Linux Designers Choose Basic Flat Model?

- Loading segment registers requires several memory cycles
- System calls implemented via `INT` instructions, applicable only when using Basic Flat Model, are faster

Prevention of Other Exploits

- Chroot Jail Breaking
- Temp File Race Condition
- File Descriptor Leakage
- Local Denial of Service Attacks
- Kernel Rootkits

Chroot Jail

- System call `chroot` changes root directory of a process
- Absolute path of a file is resolved with respect to the new root directory
- Services like anonumous FTP server are run in a chroot jail
- Chroot jail restricts only file system access

Chroot Break

WRIGHT STATE UNIVERSITY

- By exploiting weakness of following system calls
  - `chdir, fchdir, chroot`
  - These system calls does not make sure that CWD directory lies within root directory
  - `chdir` just checks if `(root == cwd)`
  - No `chdir("/")` on `chroot`
- Using `mknod` system call an attacker can corrupt file system
- Using IPC mechanisms processes inside jail can interact with processes outside the jail
- Privileged system calls such as `mount, capset, stime`

Chroot Break (cont.)

Steps involved in breaking chroot jail

```
1.mkdir(“waterbuffalo”)
2.fd=open(“.”)
3.chroot(“wb”)
4.fchdir(fd)
5.Chdir(“..”) ............... 4095 times
6.Chroot(“.”)
7.execl(“bin/sh”,”sh”,NULL)
```

Securing Chroot Jail

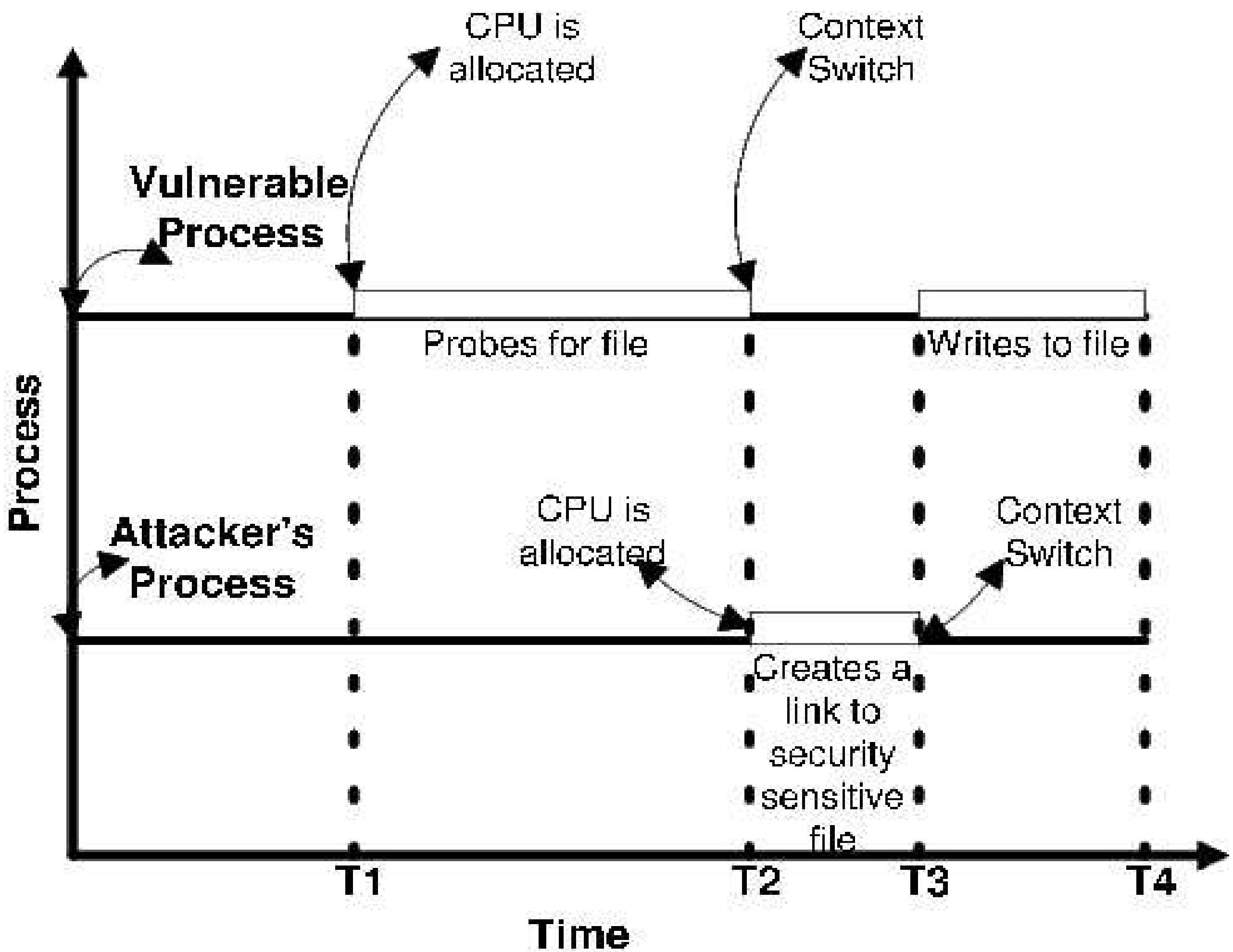
We adopt Grsecurity's secure chroot jail implementation

- No `chroot` inside chroot jail
- Enforce `chdir("/")` on chroot
- No `fchdir` to outside the root directory
- No signals to processes outside chroot jail
- No attaching shared memory outside of chroot jail
- No connecting to abstract UNIX domain sockets outside of chroot jail
- No `mknod` system call inside chroot jail

Temp File Race Condition

- What is a temp file race condition?
  - A privileged process initially probes for state of a file and takes subsequent action based on the results of the probe. If these two actions are not together atomic, an attacker can race between the actions and exploit it.
- Types of attacks
  - File creation race condition
  - File swap race condition

Race Condition (cont.)



Prevention of Race Conditions

- Proper use of `open` system call with `O_EXCL`
- Using system calls which take file descriptor instead of system calls which take file path name
  - `fchdir,fchmod,fchown,flchown,fstat`

Versus

  - `chdir,chmod,chmod,lchown,stat`

OWL /`tmp` links restrictions

- Soft Link: In a directory with sticky bit set, the process cannot follow a soft link unless the link is owned by the user or the owner of the link is the owner of the directory.

- Hard Link: A process can create a hard link to a file only when the file is owned by the user or the user has permissions to read and write the file.

File Descriptor Leakage

- What is File Descriptor Leakage?
  - `execve` does NOT close currently open file descriptors unless `close-on-exec` flag is set.
  - Sloppy developers forget to close files before calling `execve`
  - Attackers often take control of such a vulnerable process and access or modify the contents of the file left open
- Solution
  - Our hardened kernels close all the files on `execve` irrespective of `close-on-exec`. Some applications may break.

Resource Limits

- Often scripts of standard distributions are loosely configured that do not properly restrict resource usage
- A normal user with high amount of resource allocation can start local denial of service attacks
  - Fork bomb
  - Open file descriptor attack

Solution

- Resource limits can be set at kernel compile-time
  - Max number of processes of any normal user
  - Max number of file descriptors of any normal user process

Kernel Rootkits

Known ways of on-the-fly kernel modifications

- Loadable Kernel Modules
- Memory Devices

Prevention

- No LKM support
- Read-only memory devices

Pruning the Kernel

- System Calls
- Capabilities
- NIC and Routing Table Configuration
- Linux Kernel Module support
- Memory Devices: `/dev/kmem`, `/dev/mem`
- Ext file system attributes

System Calls

- Many system calls are not required for a specific type of server
  - A subset of system calls are never used
  - A subset of system calls are used only during system initialization
  - A subset of system calls are used only while initializing the services
- Attackers often exploit the unneeded system calls e.g., `ptrace`

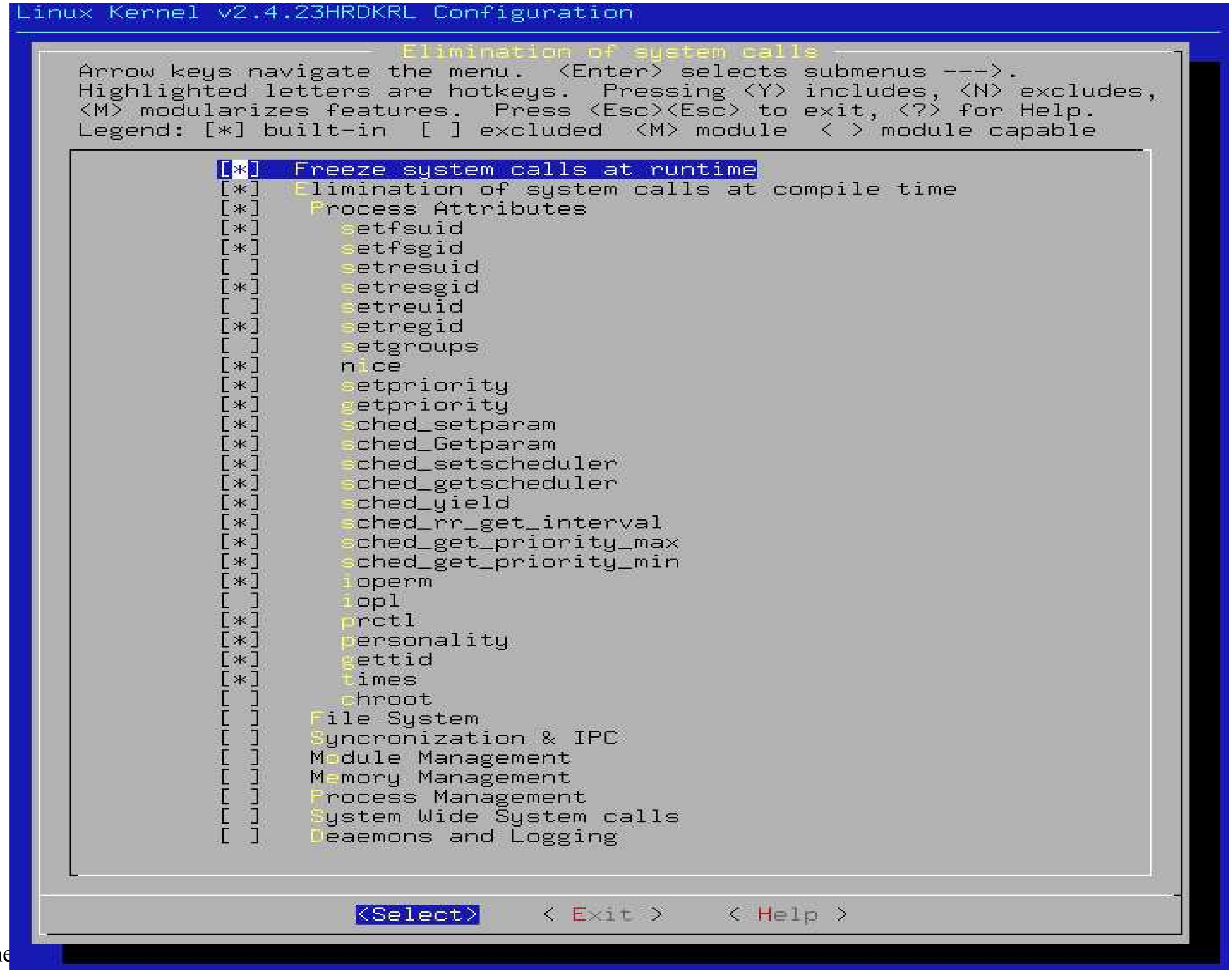
System Call Elimination

- Compile-time elimination We classified system calls into categories
  - Process Attributes
  - File System
  - Module Management
  - Memory Management
  - Inter Process Communication
  - Process Management
  - System Wide System calls
  - Daemons and Services

System Call Elimination

- Run-time freezing A new system call is introduced that
  - Takes the number of the system call to be frozen as an arg X
  - Redirects the system call X to `sys_ni_syscall` which returns error no `–ENOSYS`
  - Requires the capability `CAP_SYS_ADMIN`
  - Can freeze itself

Kconfig Menu of System Calls Elimination



Capabilities

- Eliminate capabilities at compile-time
  - `kconfig` menu of capability elimination
- Eliminate capabilities at run-time
  - A new system "`capelim`" is introduced
  - Removes the capability from capability bounding set
  - Requires capability `CAP_SYS_ADMIN`

NIC and Routing Table Configuration

- Once NIC and kernel's routing table are setup no changes are required
  - Attacker can force NIC into promiscuous mode and hide it from monitoring utilities
- Freeze at run-time
  - Freeze network card configuration
  - Freeze routing table setup
- Freeze after network and routing table are configured and before services are started
- A new system call is introduced
  - Invalidates NIC, routing table options of `ioctl` system call
  - Requires `CAP_SYS_ADMIN` capability

Loadable Kernel Module

- What is LKM?
  - A module is an object file whose code is linked to the kernel at run-time
  - The module is executed in kernel mode and in the context of the current process
  - The modules contain code which implements file systems, device drivers, executable formats etc
- Easier way of installing rootkits

LKM Rootkits

- Weaknesses of LKM
  - No secure authentication
  - Any process with capability `CAP_SYS_MOD` can insert module
  - LKM can modify any part of kernel's memory including text
  - LKM can hide itself
- Common techniques of LKM rootkits
  - System call redirection
  - Modify first few bytes of a system call
  - Modify data structures such as IDT table

Prevention of LKM Rootkits

- Eliminate LKM support at compile-time
  - Build all the modules into the kernel
- Freeze LKM support at run-time
  - Freeze capability `CAP_SYS_MOD`
  - Freeze system calls related to module management
    - `Init_module`
    - `create_module`
    - `delete_module`
    - `query_module`
    - `get_kernel_syms`

Memory Devices

- Linux Memory Devices
  - `/dev/kmem`: Kernel's memory
  - `/dev/mem`: Physical memory
  - `/dev/port`: I/O port
- Requires capability `CAP_SYS_RAWIO`
- Allow read and write access to any part of kernel's memory including text
- Rootkits installed through memory devices are very hard to detect

Prevention of `/dev/kmem` Rootkits

- Elimination of memory devices
- Read-only memory devices: Eliminate
  - `kmem_write`
  - `kmem_map`

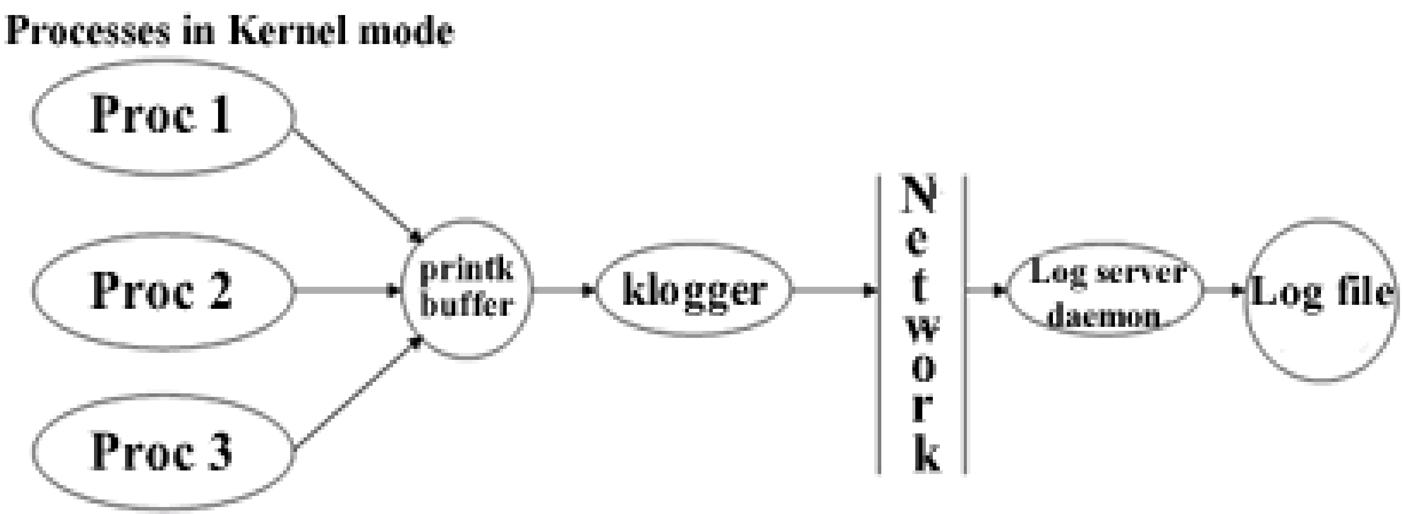
Security Hardening Additions to the Kernel

- Kernel Logger
- Kernel Integrity Checker
- Trusted Path Mapping
- Read-only File System

Kernel Logging As-is

- Kernel writes logs to a circular buffer called `printk` buffer
- `klogd` clears `printk` buffer through `syslog`
- `klogd` writes logs to a file on locally mounted file system
- `klogd` is a user process
- Root user has complete control of `klogd`
- Any process with capability `CAP_SYS_ADMIN` can read and clear `printk` buffer through `syslog`
- Any user process can read `printk` buffer

Our Kernel Logger: klogger



Our Kernel Logger Design

- Klogger contains
  - A kernel thread
  - Circular buffer `printk`
- When `printk` buffer is non-empty
  - The kernel thread locks the buffer
  - Reads and clears the buffer and sends logs to a remote log server
  - Releases the lock on the buffer
  - Relinquishes CPU

Klogger Design (cont.)

- The kernel thread goes to sleep while `printk` buffer is empty
- When connection to log server is lost
  - Klogger relinquishes the CPU and joins the run queue
  - Try again for connection
- Klogger is started by
  - `init` kernel thread
  - Uses the new `klogger` system call
- Klogger is stopped when `reboot` system call is called before power down of devices

Klogger Design (cont.)

- The scheduling policy is `sched_other`
  - Dynamic priority is assigned, no static priority
  - Real-time processes are not affected
- IP address and port number of remote log server are specified at kernel compile-time, not changeable at run-time.

Advantages of Klogger

- No user can control klogger
- The logs are stored in a remote server
- Starts before `init` becomes a user process and exits only when `reboot` system call is called
- No process except klogger can clear logs in `printk` buffer
- No denial of service can happen due to connection loss or log flooding
- Negligible performance loss

Kernel Integrity Checker (KIC)

- What is KIC?
  - To detect run-time kernel modifications done to kernel's text through LKM, memory devices, or some **other as yet unknown methods**
  - This can be extended to detect modifications done to data which is expected to remain unchanged
- Current Detection Tools KSTAT, Samhain
  - The detecting processes are user processes
  - Requires `System.map` and `/dev/kmem`
  - Requires system calls `query_module, get_kernel_syms`
  - Can detect only system call related modifications

KIC Design

- A kernel thread
- MD5 database
  - The MD5 checksum of text region is computed and stored in MD5 database
  - MD5 database is in dynamically allocated kernel's memory
- The kernel thread wakes up every `n` ticks, computes MD5 checksum and compares with that in MD5 database
- KIC is started by
  - `init` kernel thread
  - A new system call `kic`

Advantages of KIC

- Does not depend on `/dev/kmem` and `System.map`
- No process can control KIC
- Configurable only at kernel compile-time
- Can detect modifications to any part of kernel's text
- Neglible performance overhead
- Starts before `init` becomes a user process and exits only when `reboot` is called

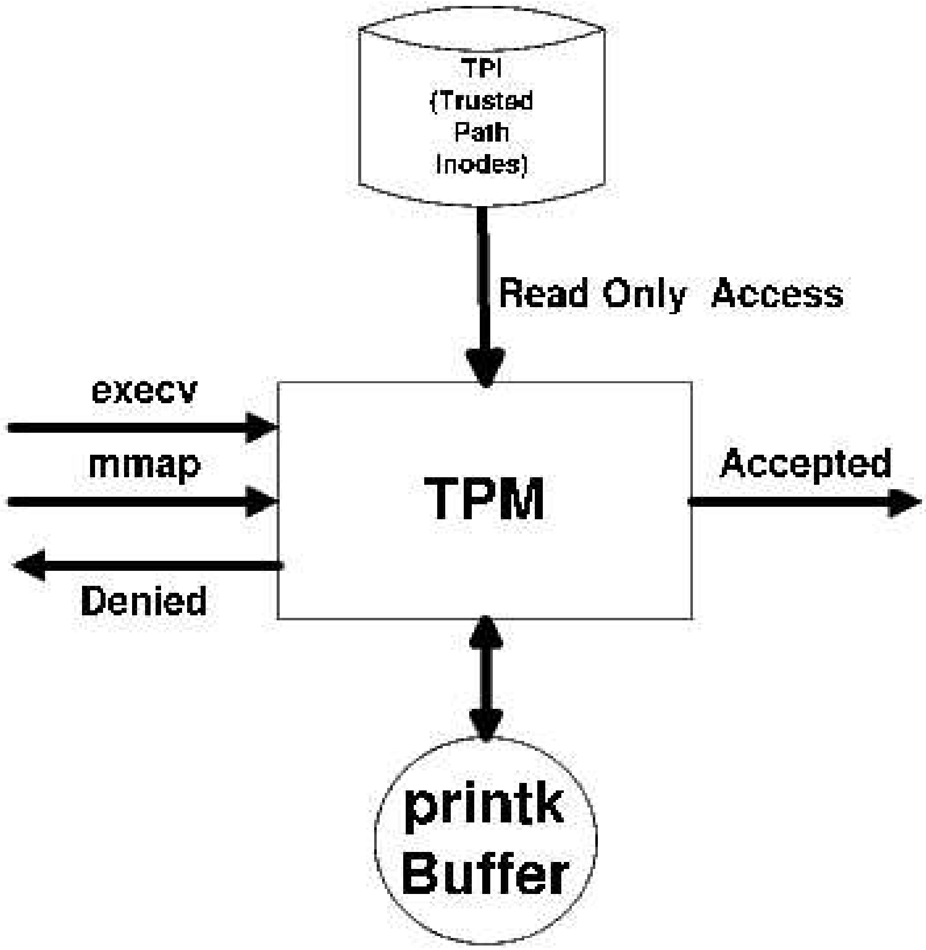
Trusted Path Mapping

- To prevent arbitrary file execution
- What is Trusted Path Execution?
  - File execution is restricted to trusted path directories
  - A Trusted path is one where the parent directory is owned by root and is neither group nor others writable
  - Grsecurity implements TPE
- What is Trusted Path Mapping?
  - Memory Mapping (read,write,execute) is restricted to files in trusted path directories
  - Trusted path directories are specified by administrator at kernel compile-time

Trusted Path Mapping (cont.)

- Even root user cannot override TPM
- System calls intercepted: `execve`, `mmap`
- TPM consists of : TPM monitor, Trusted Path I-node database
- `init` kernel thread lookup the file system and writes i-node details of trusted path directories to TPI database
- TPM is started by
  - `init` kernel thread
  - The new `tpm` system call

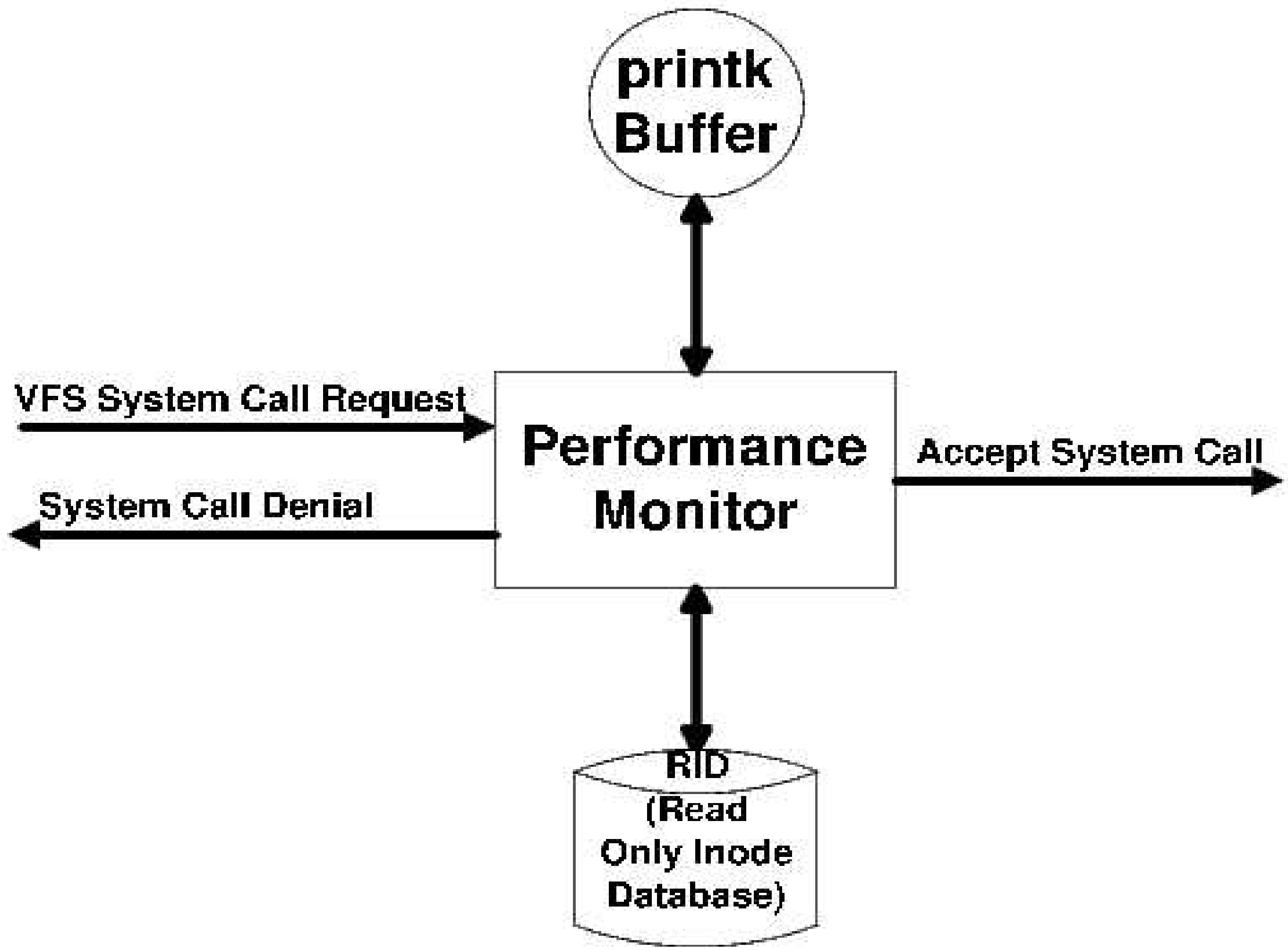
Trusted Path Mapping (cont.)



Read-Only FS

- A file system as a whole can be made read-only. But individual files cannot be made read-only.
- Even with a read-only mount, using raw devices, data can be corrupted
- Our design of read-only file system is based on interception of VFS system calls
- We consider that a file is read-only *only* when
  - The content of file cannot be modified
  - Attributes of the file (access times, ownership, permissions) cannot be modified
  - The file cannot be renamed
  - The file cannot be mapped with `MAP_SHARED`

Read-only FS (cont.)



Read-only FS (cont.)

- System calls intercepted
  - `open, mknod, create, mkdir, rmdir, link, unlink, write, writev, pwrite, truncate, ftruncate and sendfile`
  - `chmod, fchmod, lchown, fchown, chown and utime`
  - `rename`
  - `mmap and mprotect`
- No writes to block devices

Ext2 File System Attributes

- Extra attributes of ext file system
  - `EXT2_IMMUTABLE_FL`: "Immutable" file
  - `EXT2_APPEND_FL`: Writes to file may only append
  - `EXT2_NOATIME_FL`: Do not update *atime*
- To make individual files read-only
  - Set the above attributes in off-line mode
  - And freeze ext file system attributes at compile-time of kernel

Hardened Kernels for Servers

- Anonymous FTP server
- Web server
- Mail server
- File server

Kconfig menu of HRDKRL

```
Linux Kernel v2.4.23HRDKRL Configuration

              Hardened Kernels For Linux Servers
    Arrow keys navigate the menu.  <Enter> selects submenus --->.
    Highlighted letters are hotkeys.  Pressing <Y> includes, <N> excludes,
    <M> modularizes features.  Press <Esc><Esc> to exit, <?> for Help.
    Legend: [*] built-in  [ ] excluded  <M> module  < > module capable

      --- Masters Thesis of SSGadi under Dr.PMateti
      [*]   Chroot Jail Restrictions
      [*]      Deny access to abstract AF_UNIX sockets out of chroot
      [*]      Deny shmat() out of chroot
      [*]      Deny double chroot
      [*]   Temporary File Race conditions Prevention
      [*]      Softlinks Protection
      [*]      Hardlinks Protection
      [*]   Freeze EXT2 file system attributes
      [*]   Close files on execve
      [*]   Trusted Path Mapping
                Enter Trusted directories: "/bin,/sbin,/usr,/lib,/etc"
      [*]      Start TPM while booting before init
      [ ]      Start TPM through a system call
      [*]   Linux Kernel Logger
               IP address of remote log server: "192.168.17.55"
      (8090)      Port of remote log server
      [*]      Start the kernel logger while booting before init
      [ ]      Start the kernel logger through a system call
      [*]   Linux Kernel Integrity Checker
      (100)       Timeout of KIC in ticks
      [*]      Start the KIC while booting before init
      [ ]      Start the KIC through a system call
      [*]   Memory Devices Elimination
      [ ]      /dev/kmem (NEW)
      [ ]      /dev/mem (NEW)
      [ ]      /dev/port (NEW)
      [*]   Freeze Network Configuration
      [ ]      Freeze routing operations (NEW)
      [ ]      Freeze interface operations (NEW)
      [ ]   Configure the resource requests of process
       Elimination of system calls  --->
       Elimination of capabilities  --->
      [*]   No Overwrite in FTP directory(For FTP servers only)
                Enter anonymous FTP directory: "/var/ftp" (NEW)
      [ ]      Start this while booting before init (NEW)
      [*]      Start this through a system call (NEW)

                  <Select>    < Exit >    < Help >
```

Protecting Anonymous FTP Directory

- Problem: Two different “put” requests with same file name may result in one overwriting other
- Solution:
  - Creating a file and opening it for writing should happen in **one** system call
  - While open, no process can write to a file except the one that created it
  - Once the file is closed, no process can to write to it, including the one which created it
  - No process should be able to rename a file
  - No process should be able to remove a file

Protecting Anon. FTP Directory (cont.)

- The absolute path name of the FTP directory should be specified at kernel-compile time
- The FTP protection can be started by the `init` kernel thread
- New system call `no_overwrite_ftp`

New System Calls

1. `freeze_syscalls`
2. `cap_elim`
3. `freeze_network`
4. `kic`
5. `klogger`
6. `tpm`
7. `no_overwrite_ftp`

- **4-7** would freeze themselves once they are called.
- The others should be frozen by a root-owned process.

System Calls Eliminated at Compile-time

System Calls	FTP	Web	Mail	File
setresuid		x		x
chroot		x		x
sendfile			x	x
ftruncate	x	x		x
sync		x		x
fsync		x		
fdatasync	x	x	x	
rename	x	x		
rmdir	x	x		x
mkdir	x	x		x
statfs		x	x	x
mknod	x	x		x
nfsservctl	x	x	x	

System Calls Frozen at Run-time

System Calls	FTP	Web	Mail	File
link	x	x		x
capset		x	x	x
setrlimit	x	x		x
flock	x	x		

Capabilities Eliminated at Compile-time

Capabilities	FTP	Web	Mail	File
CAP_SYS_CHROOT		x		x
CAP_MKNOD	x	x		x

Size of `vmlinux`

Kernel	Size of vmlinux (bytes)	Size of System.map (bytes)
FTP server	3244300	481219
Web server	3235868	481019
Mail server	3235880	481019
File server	3471100	504864

Conclusion

- Our kernels are the result of
  - Serious pruning of kernel
  - Several additions to the kernel
- The patch was built for Linux kernel version 2.4.23
- Reconfiguration should be done in off-line mode
- Our kernels run on stock Mandrake 9.1 distribution running on Dell Precision 210 systems

Future Work

- We did not address TCP/IP/ICMP based attacks
- Focused on the i386 platform. Adapt to other architectures, especially for IA-64
- Support for access control models e.g., MAC, RC, AC
- Further pruning down of services
- Cryptographically signed LKM support

Acknowledgments

- Dr. Prabhaker Mateti
- Dr. Mateen Rizki and Dr. Bin Wang
- Sai Krishna .D and Karthik .M
- Sudhir .D

Questions

?

DEMO

- Chroot jail breaking
- LKM based rootkits
- `/dev/kmem` exploits
- Trusted path management
- A local denial of service attack
- Kernel integrity checker