Introdução à Engenharia Reversa de Aplicações Maliciosas em Ambientes Linux

Marcus Botacin Lucas Galante Otávio Silva Paulo de Geus

XIX SBSEG

2019

Roteiro

Roteiro

Atividades Práticas

Repositório

```
git clone https://github.com/marcusbotacin/
Malware.Reverse.Intro
```

Quais aplicações apresentaremos neste curso?

- `binutils`
- *ht editor*
- `GDB`
- `iptables`

Quais técnicas apresentaremos neste curso?

- *Linking*
- *Disassembly*
- *Tracing*
- *Packing*
- *Patching*
- *Debugging*

Este curso não é sobre:

- Lista exaustiva de técnicas de anti-análise
- Programação avançada em *kernel*
- *Dissectors* de tráfego de rede

O que eu preciso saber?

- **Sistemas Operacionais**:
  - *Kernel*
  - *userland*
  - *syscalls*
- **Compilação**:
  - GCC
  - Geração de código objeto
  - Compilação de bibliotecas
- ***Assembly***:
  - *Branches*

Roteiro

Estrutura

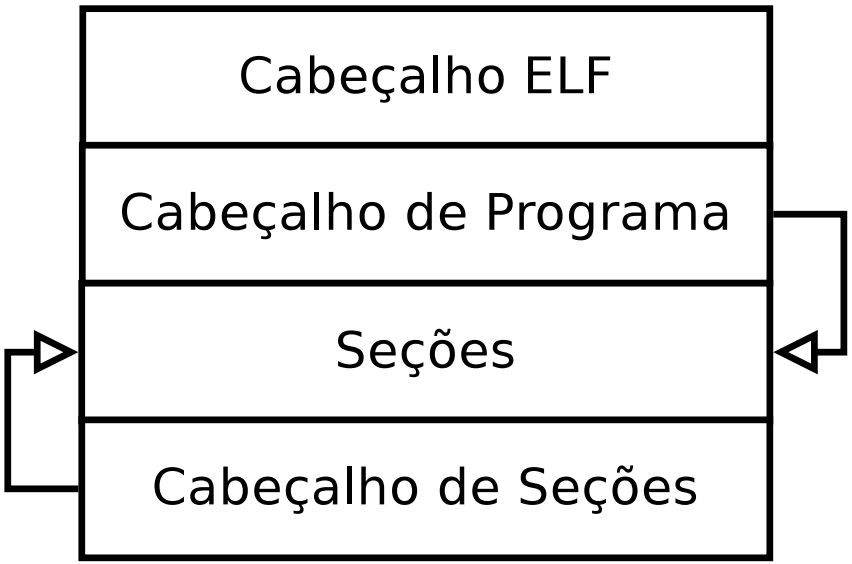

Figura: **Arquivo ELF**. Estrutura Básica.

Cabeçalho ELF

```
1  typedef struct {
2          unsigned char e_ident[EI_NIDENT];
3          uint16_t      e_type;
4          uint16_t      e_machine;
5          uint32_t      e_version;
6          ElfN_Addr     e_entry;
7          ElfN_Off      e_phoff;
8          ElfN_Off      e_shoff;
9          uint32_t      e_flags;
10         uint16_t      e_ehsize;
11         uint16_t      e_phentsize;
12         ...
13     } ElfN_Ehdr;
```

Código 1: Definição do cabeçalho de um arquivo ELF.

Cabeçalho de Programa ELF

```
1  typedef struct {
2          uint32_t   p_type;
3          uint32_t   p_flags;
4          Elf64_Off  p_offset;
5          Elf64_Addr p_vaddr;
6          Elf64_Addr p_paddr;
7          uint64_t   p_filesz;
8          uint64_t   p_memsz;
9          uint64_t   p_align;
10     } Elf64_Phdr;
```

Código 2: Definição do cabeçalho de programa de arquivo ELF.

Cabeçalho de Seção ELF

```
1  typedef struct {
2          uint32_t   sh_name;
3          uint32_t   sh_type;
4          uint64_t   sh_flags;
5          Elf64_Addr sh_addr;
6          Elf64_Off  sh_offset;
7          uint64_t   sh_size;
8          uint32_t   sh_link;
9          uint32_t   sh_info;
10         uint64_t   sh_addralign;
11         uint64_t   sh_entsize;
12     } Elf64_Shdr;
```

Código 3: Definição do cabeçalho de seção de um arquivo ELF.

Seções de um binário ELF

```
1   [Nr] Nome
2   [ 1] .interp              [ 2] .note.ABI-tag
3   [ 3] .note.gnu.build-i    [ 4] .gnu.hash
4   [ 5] .dynsym              [ 6] .dynstr
5   [ 7] .gnu.version         [ 8] .gnu.version_r
6   [ 9] .rela.dyn            [10] .rela.plt
7   [11] .init                [12] .plt
8   [13] .plt.got             [14] .text
9   [15] .fini                [16] .rodata
10  [17] .eh_frame_hdr        [18] .eh_frame
11  [19] .init_array          [20] .fini_array
12  [21] .jcr                 [22] .dynamic
13  [23] .got                 [24] .got.plt
14  [25] .data                [26] .bss
15  [27] .comment             [28] .shstrtab
16  [29] .symtab              [30] .strtab
```

Código 4: Exibição das seções de um arquivo ELF.

Principais Seções de um binário ELF

- **.data**: Variáveis inicializadas.
- **.rodata**: Variáveis inicializadas com permissão apenas de leitura.
- **.text**: Instruções de máquina.
- **.*dyn***: Espaço reservado para ponteiros de alocações dinâmicas.
- **.*sym***: Espaço reservado para ponteiros de códigos ligados dinamicamente.

Manipulação de binário ELF

```
1 def process_file(filename):
2     with open(filename, 'rb') as f:
3         elffile = ELFFile(f)
4         for section in elffile.iter_sections():
5             print(section.name)
```

Código 5: Enumeração das seções de um arquivo ELF através da solução pyelftools.

Roteiro

Identificando Arquivos

```
1 >$ file hello.c
2 hello.c: ASCII text
3 >$ file Documents/
4 Documents/: directory
5 >$ file wticg.pdf
6 wticg.pdf: PDF document, version 1.4
```

Código 6: Comando file em diferentes arquivos.

Identificando Arquivos Binários

```
1 >$ file dynamic-malicious.bin
2 dynamic-malicious.bin: ELF 64-bit LSB executable,
      x86-64, version 1 (SYSV),
3 dynamically linked, interpreter /lib64/l, for GNU/
      Linux 2.6.32,
4 BuildID[sha1]=5
      e42df0d86c9df096eefb5fd99a703c479957fc8, not
      stripped
```

Código 7: Comando file em um arquivo ELF com ligação dinâmica

Identificando Arquivos Binários

```
1 >$ file static-malicious.bin
2 static-malicious.bin: ELF 64-bit LSB executable, x86
      -64, version 1 (GNU/Linux),
3 statically linked, for GNU/Linux 2.6.32,
4 BuildID[sha1]=
      a20285090944c95cc21aac376a87144b37657496, not
      stripped
```

Código 8: Comando file em arquivo ELF compilado com ligação estática

Identificando Bibliotecas Ligadas

```
1 >$ ldd dynamic-malicious.bin
2   linux-vdso.so.1 =>  (0x00007ffe987e6000)
3   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0
        x00007f77a58b3000)
4   /lib64/ld-linux-x86-64.so.2 (0x00007f77a5c7d000)
```

Código 9: Comando ldd em arquivo ELF com ligação dinâmica

```
1 >$ ldd static-malicious.bin
2     not a dynamic executable
```

Código 10: Comando ldd em arquivo ELF com ligação estática

Identificando Funções Ligadas

```
1  >$ objdump -T
       VirusShare_4e0ee9f1571107a015e63925626b562d
2  DYNAMIC SYMBOL TABLE:
3  080484d8 DF *UND*  0000003b  GLIBC_2.0   inet_addr
4  08048538 DF *UND*  000000e2  GLIBC_2.0   exit
5  08048548 DF *UND*  0000003e  GLIBC_2.0   atoi
6  08048558 DF *UND*  00000039  GLIBC_2.0   send
7  08048568 DF *UND*  0000000e  GLIBC_2.0   htons
8  08048578 DF *UND*  00000054  GLIBC_2.0   memset
9  08048588 DF *UND*  00000039  GLIBC_2.0   connect
10 08048598 DF *UND*  00000039  GLIBC_2.0   recv
11 080485a8 DF *UND*  00000034  GLIBC_2.0   sprintf
12 080485b8 DF *UND*  00000039  GLIBC_2.0   socket
```

Código 11: Saída do comando objdump exibindo a tabela de símbolos de um exemplar de malware indicando o uso de recursos de rede

Identificando Funções Ligadas

```
1 >$ objdump -T static-malicious.bin
2 static-malicious.bin:     file format elf64-x86-64
3 objdump: static-malicious.bin: not a dynamic object
4 DYNAMIC SYMBOL TABLE:
5 no symbols
```

Código 12: Saída do comando objdump indicando a ausência da tabela de símbolos para um binário ELF ligado estaticamente

Disassembly

```
1  >$ objdump -d
       VirusShare_4e0ee9f1571107a015e63925626b562d
2  0804876f <main>:
3  8048ced: call   8048528 <gethostbyname@plt>
4  8048cf2: add    $0x10,%esp
5  8048cf5: mov    %eax,-0x8a1c(%ebp)
6  8048cfb: cmpl   $0x0,-0x8a1c(%ebp)
7  8048d02: jne    8048d1e <main+0x5af>
8  8048d04: sub    $0xc,%esp
9  8048d0c: call   80484b8 <perror@plt>
10 8048d11: add    $0x10,%esp
11 8048d14: sub    $0xc,%esp
12 8048d19: call   8048538 <exit@plt>
13 8048d1e: sub    $0x4,%esp
14 8048d27: call   80485b8 <socket@plt>
```

Código 13: Saída do comando objdump ilustrando o dissassembly de um exemplar malicioso que faz uso de recursos de rede

Inspecionando *Strings*

```
25 64 2e 25 64 00 00 00   |191.206.%d.%d...|
25 64 2e 25 64 00 00 00   |187.118.%d.%d...|
25 64 2e 25 64 00 00 00   |187.116.%d.%d...|
25 64 2e 25 64 00 00 00   |179.224.%d.%d...|
25 64 2e 25 64 00 00 00   |179.166.%d.%d...|
1b 5b 33 31 6d 50 68 6f   |admin....[31mPho|
65 64 20 1b 5b 33 32 6d   |ne Cracked .[32m|
25 73 20 7c 20 1b 5b 33   |-> .[37m%s | .[3|
6d 65 20 1b 5b 33 32 6d   |1mUsername .[32m|
61 64 6d 69 6e 20 7c 20   |-> .[37madmin | |
73 77 6f 72 64 20 1b 5b   |.[31mPassword .[|
33 37 6d 61 64 6d 69 6e   |32m-> .[37madmin|
73 75 0d 0a 00 00 00 00   |.[0m....su......|
32 33 0d 0a 00 00 00 00   |oelinux123......|
```

Figura: Comando hexdump -C em exemplar malicioso contendo *strings* suspeitas.

Inspecionando *Strings*

```
>$ strings
    a9a780c66ec18861e4881430202f62d6ceaba3187626d48ab24152

39.34.%d.%d                 59.103.%d.%d
191.12.%d.%d                186.117.%d.%d
179.170.%d.%d               191.206.%d.%d
187.118.%d.%d               179.224.%d.%d
admin                       [31mPhone Cracked
[31mUsername                [31mPassword
[36mDevice Cracked          GETLOCALIP
My IP: %s                   BOTKILL
Killing Bots                NETIS ON | OFF
[TELNET] SCANNER ON:%s      ICMP
HTTP                        STOP
/proc/net/route             /etc/resolv.conf
```

Código 14: Saída do comando strings em exemplar malicioso indicando diversas strings suspeitas

Strings Ofuscadas

```
1 >$ strings upx-file.bin
2 $Info: This file is packed with the UPX executable
      packer http://upx.sf.net $
3 dl%)
4 UPPH
5 q}Nsf
6 x‘TJq&H
7 UPX!
```

Código 15: Saída do comando strings em arquivo empacotado pela solução UPX

Empacotamento

```
1  >$ upx original.bin -o upx-file.bin
2                          Ultimate Packer for
                               eXecutables
3                          Copyright (C) 1996 - 2013
4  UPX 3.91        Markus Oberhumer, Laszlo Molnar &
       John Reiser   Sep 30th 2013
5
6            File size         Ratio      Format
                 Name
7       --------------------   ------   -----------
            -----------
8        610309 ->     270256   44.28%  linux/elf386   1-
            upx
9
10 Packed 1 file.
```

Código 16: Empacotamento de um binário com ligação estática pela solução UPX

Empacotamento

```
1 b9c93c4cf9f0848f03834614c6f91759e4e068c0
      original.bin
2 3a5d19d3372f4d8e05599da542bfa161a14a4a32  upx-
      file.bin
```

Código 17: Diferença no sha1sum dos binários original e empacotado

Empacotamento

```
1 >$ upx -d upx-file.bin
2                        Ultimate Packer for eXecutables
3                           Copyright (C) 1996 - 2013
4 UPX 3.91        Markus Oberhumer, Laszlo Molnar & John Reiser   Sep 30th 2013
5         File size         Ratio      Format      Name
6    --------------------   ------   -----------   -----------
7     610311 <-     270256   44.28%  netbsd/elf386  1-upx
8 Unpacked 1 file.
```

Código 18: Desempacotamento de um binário empacotado via UPX

Roteiro

Fluxos de Execução Protegidos

```
1 int password()
2   scanf("%[^\n]s",string);
3   if(strcmp(string,PASS)==0)
4     malicious();
5   else
6     exit(0);
```

Código 19: Exemplo de código de um exemplar que exibe seu comportamento malicioso apenas quando a string de inicialização é corretamente definida.

Fluxos de Execução Protegidos

```
400692      call        wrapper_601030_400520
400697      test        eax, eax
400699      jnz         loc_4006a5
40069b      mov         edi, strz_Oh_Yeah__40075a
4006a0      call        wrapper_601018_4004f0
4006a5
```

Figura: ***Disassembly* da função verificadora.** A função verificadora é invocada (linha 1) e seu valor de retorno é testado (linha 2) de modo a determinar se a execução deve proceder (linha 4) ou não (linha 3).

Modificação do Binário Protegido

```
400692      call        wrapper_601030_400520
400697      test        eax, eax
400699      nop
40069a      nop
40069b      mov         edi, strz_Oh_Yeah__40075a
4006a0      call        wrapper_601018_4004f0
```

Figura: ***Disassembly* da função verificadora.** A função verificadora pode ser modificada de modo que a rotina de verificação seja eliminada.

Verificações de Integridade

```
1  int main()
2      crc32 = Crc32_ComputeBuf(0,passwd,33);
3      if(crc32!=MYCRC)
4          exit(0);
5      passwd(pass);
```

Código 20: Exemplo de código de verificação de CRC32 para proteger a função verificadora de strings.

Verificações de Integridade

```
4007fd      call        Crc32_ComputeBuf
400802      mov         [rbp-    ], rax
400809      cmp         qword ptr [rbp-    ],
400814      jnz         loc_400820
400816      mov         edi,
40081b      call        wrapper_602050_400680
```

Figura: ***Disassembly*** **da região ao redor da função verificadora..**
Rotina de verificação do CRC32 é invocada (linha 1) e resulta no término da execução (linha 4) caso a função original tenha sido alterada.

Roteiro

Traçando Chamadas de Sistema

```
1 [pid 11047] open("/etc/passwd", O_WRONLY|O_CREAT|
      O_APPEND, 0666) = -1 EACCES (Permission denied)
```

Código 21: Exemplo de registro de chamada de sistema via strace de uma tentativa de acesso a credenciais do usuário (/etc/shadow).

Traçando Chamadas de Sistema

```
1  >$ strace ./malware.bin
2  execve("./malware.bin", ["./malware.bin"], [/* 25
       vars */]) = 0
3  access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
4  access("/etc/ld.so.preload", R_OK)      = -1 ENOENT
5  ...
6  write(2, "Invalid␣parameters!\n", 20Invalid
       parameters!)    = 20
7  write(1, "DNS␣Flooder␣v1.3\nUsage:␣./malwar"..., 244
       DNS Flooder v1.3
8  Usage: ./malware.bin <target IP/hostname> <port to
       hit> <reflection file (format: IP DOMAIN>> <
       number threads to use> <time (optional)>
9  exit_group(-1)                          = ?
10 +++ exited with 255 +++
```

Código 22: Traço de chamadas de sistema de exemplar malicioso exibindo rotinas de carregamento e de finalização

Traçando Carregamento de Bibliotecas

```
1 strace 0c4f1bf21f9433...d29f
2 execve("./malware.bin", ["./malware.bin"], [/* 25
      vars */]) = 0
3 access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
4 access("/etc/ld.so.preload", R_OK)      = -1 ENOENT
5 open("/usr/lib/x86_64-linux-gnu/libstdc++.so.6",
      O_RDONLY|O_CLOEXEC) = 3
6 open("/lib/x86_64-linux-gnu/libm.so.6", O_RDONLY|
      O_CLOEXEC) = 3
```

Código 23: Traço de chamadas de sistema de binário carregando diversas bibliotecas

Traçando Processos Filhos

```
1 >$ strace -f d73a9a8dbd...7587d
2 execve("./malware.bin", ["./malware.bin"], [/* 25
      vars */]) = 0
3 access("/etc/ld.so.nohwcap", F_OK)      = -1 ENOENT
4 ...
5 clone(child_stack=0, flags=CLONE_CHILD_CLEARTID|
      CLONE_CHILD_SETTID|SIGCHLD, child_tidptr=0
      x7fbc2bce09d0) = 11045
6 wait4(-1, strace: Process 11045 attached
7  <unfinished ...>
8 [pid 11045] ptrace(PTRACE_TRACEME, 0, NULL, NULL) =
      -1 EPERM
9 [pid 11045] execve("./malware.bin", ["./malware.bin"
      , "2", "3", "4"], [/* 25 vars */]) = 0
```

Código 24: Traço de chamadas de sistema de exemplar malicioso com chamada fork e evasão de execução por parte do processo filho

Delay de Execução

```
1 >$ strace
      a4332ab4b8f8e2f04fef7c40c103ab570f42011ba41b3caaa03029

2 [pid 11046] waitpid(11084, [{WIFEXITED(s) &&
      WEXITSTATUS(s) == 0}], 0) = 11084
3 [pid 11046] munmap(0xf6f0a000, 4096)      = 0
4 [pid 11046] ioctl(1, SIOCGIFFLAGS, {ifr_name="ens3",
       ifr_flags=IFF_UP|IFF_BROADCAST|IFF_RUNNING|
      IFF_MULTICAST}) = 0
5 [pid 11046] gettimeofday({312344432895491,
      -1781962174260832 0}, NULL) = 0
6 [pid 11046] nanosleep({429496729600000000,
      57775638072372071 2}, <unfinished ...>
```

Código 25: Traço de chamadas de sistema de exemplar malicioso que realiza uma chamada sleep com tempo suficientemente longo para causar o término da execução em muitos sistemas de análise

Detectando o *tracer*

```
1 int main(){
2   if(ptrace(PTRACE_TRACEME)==-1)
3       exit(0);
4   malicious()
```

Código 26: Evasão de análise por strace através da autoverificação da presença do framework ptrace.

Detectando o *tracer*

```
1 >$ strace ./ptrace
2 ...(ommitted initialization system calls)
3 ptrace(PTRACE_TRACEME, 18446744073049584056, 0
      x7ffdd8a9b1c8, NULL) = -1 EPERM (Operation not
      permitted)
4 exit_group(0)                                 = ?
5 +++ exited with 0 +++
```

Código 27: Traço de chamadas de sistema de arquivo com evasão de análise via chamada ptrace

Traçando Chamadas de Função

```
>$ ltrace b0148c40....ecf9f1
[pid 11043] __libc_start_main(0x400ab2, 1, 0
    x7ffc53304f98, 0x400d00 <unfinished ...>
[pid 11043] printf("␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    "..., "/usr/bin/passwd") = 308
[pid 11043] puts("DirtyCow␣root␣privilege␣escalati"
    ...) = 35
[pid 11043] printf("Backing␣up␣%s␣to␣/tmp/bak\n", "/
    usr/bin/passwd") = 39
[pid 11043] asprintf(0x7ffc53304ea0, 0x400f1e, 0
    x6020c0, 0x7ffffe5) = 27
[pid 11043] system("cp␣/usr/bin/passwd␣/tmp/bak" <no
     return ...>
...
```

Código 28: Traço de chamadas de funções de exemplar malicioso executando um exploit para a vulnerabilidade DirtyCow

Traçando Chamadas de Função

```
1 >$ ltrace ./static-malicious.bin
2 Couldn't find .dynsym or .dynstr in "/proc/25332/exe
      "
```

Código 29: Traço de chamadas de funções reportando falha ao analisar um binário estaticamente ligada via ltrace

Programando *Tracers*

```
1 long ptrace(enum __ptrace_request request, pid_t pid
      , void *addr, void *data);
```

Código 30: Protótipo para a chamada ptrace

Traçando Instruções

```
1 >$ itrace f5d5557...ae0337d4
2 RIP: 0x400c63 | /home/SBSeg/itrace 0x400000 0x403000
3 RIP: 0x400c6a | /home/SBSeg/itrace 0x400000 0x403000
4 RIP: 0x7f446f862c17 | /lib/x86_64-linux-gnu/ld-2
      .23.so 0x7f446f852000 0x7f446f878000
5 RIP: 0x7f446f862c33 | /lib/x86_64-linux-gnu/ld-2
      .23.so 0x7f446f852000 0x7f446f878000
6 RIP: 0x401d84 | /home/SBSeg/itrace 0x400000 0x403000
7 RIP: 0x401d8c | /home/SBSeg/itrace 0x400000 0x403000
8 RIP: 0x7f446f862c35 | /lib/x86_64-linux-gnu/ld-2
      .23.so 0x7f446f852000 0x7f446f878000
9 RIP: 0x7f446f862c38 | /lib/x86_64-linux-gnu/ld-2
      .23.so 0x7f446f852000 0x7f446f878000
```

Código 31: Traço de execução de um exemplar malicioso a nível de instruções. Pode-se identificar os pontos exatos de troca de contexto entre a execução do código do binário e da biblioteca ligada.

Roteiro

GDB *Disassembly*

```
1  (gdb) disassemble main
2  Dump of assembler code for function main:
3     0x0000000000400646 <+0>:     push   %rbp
4     0x0000000000400647 <+1>:     mov    %rsp,%rbp
5  => 0x000000000040064a <+4>:     sub    $0x110,%rsp
6     0x0000000000400651 <+11>:    mov    %fs:0x28,%rax
7     0x000000000040065a <+20>:    mov    %rax,-0x8(%
          rbp)
8     0x000000000040065e <+24>:    xor    %eax,%eax
9     0x0000000000400660 <+26>:    mov    $0x400744,%
          edi
10    0x0000000000400665 <+31>:    callq  0x4004f0 <
          puts@plt>
```

Código 32: Disassembly realizado pelo GDB.

Pontos de Parada

```
1 (gdb) b main
2 Ponto de parada 1 at 0x40064a
3 (gdb) r
4 Starting program: /home/sbseg/malware
5 Breakpoint 1, 0x000000000040064a in main ()
```

Código 33: Definição de Breakpoints no GDB.

Valores em Registradores

```
1  (gdb) info registers
2  rax 0x400646                rbx     0x0
3  rcx 0x0                     rdx     0x7fffffffdd78
4  rsi 0x7fffffffdd68          rdi     0x1
5  rbp 0x7fffffffdc80          rsp     0x7fffffffdc80
6  r8  0x400730                r9      0x7ffff7de7ac0
7  r10 0x846                   r11     0x7ffff7a2d740
8  r12 0x400550                r13     0x7fffffffdd60
9  r14 0x0                     r15     0x0
10 rip 0x40064a <main+4>       eflags  0x246 [ PF ZF IF ]
```

Código 34: Obtenção dos valores atuais nos registradores do programa em execução através do GDB.

Execução Passo a Passo

```
1  (gdb) display/i $pc
2  Breakpoint 1, 0x000000000040064a in main ()
3  1: x/i $pc
4  => 0x40064a <main+4>:    sub    $0x110,%rsp
5  (gdb) stepi
6  0x0000000000400651 in main ()
7  1: x/i $pc
8  => 0x400651 <main+11>:   mov    %fs:0x28,%rax
9  (gdb)
10 0x000000000040065a in main ()
11 1: x/i $pc
12 => 0x40065a <main+20>:   mov    %rax,-0x8(%rbp)
13 (gdb)
14 0x000000000040065e in main ()
15 1: x/i $pc
16 => 0x40065e <main+24>:   xor    %eax,%eax
```

Código 35: Avanço de execução passo a passo.

Modificação do Fluxo de Execução

```
1 (gdb) set $eflags|=0x40
2 (gdb) info registers eflags
3 eflags         0x246     [ PF ZF IF ]
4 (gdb) set $eflags|=~0x40
5 (gdb) info registers eflags
6 eflags         0x54fd7   [ CF PF AF ZF SF TF IF DF OF
      NT RF AC ]
```

Código 36: Alteração do registrador de flags via GDB.

Binary Patching

```
1  (gdb) disassemble main
2  Dump of assembler code for function passwd:
3     0x0000000000400692 <+76>:    callq  0x400520 <
       strcmp@plt>
4     0x0000000000400697 <+81>:    test   %eax,%eax
5     0x0000000000400699 <+83>:    jne    0x4006a5 <
       main+95>
6  (gdb) set {int}0x0000000000400699 = 0x90
7  (gdb) set {int}0x000000000040069a = 0x90
8  Dump of assembler code for function passwd:
9     0x0000000000400692 <+76>:    callq  0x400520 <
       strcmp@plt>
10    0x0000000000400697 <+81>:    test   %eax,%eax
11    0x0000000000400699 <+83>:    nop
12    0x000000000040069a <+84>:    nop
```

Código 37: Binary Patching via GDB.

Funções Ocultas

```
1 (gdb) call passwd("MYPASS")
2 $1 = -7
3 (gdb) call passwd("The␣Pass")
4 Oh Yeah!
5 $2 = 9
```

Código 38: Invocação de função independente do contexto através do GDB.

LD_PRELOAD

```
1 LD_PRELOAD=./library.so ./binary.bin
```

Código 39: Exemplo de chamada LD_PRELOAD na qual a biblioteca substitui funções originalmente referenciadas pelo binário

Contornando Anti-Análise

```
1 unsigned int sleep(unsigned int seconds){
2   return 0;
3 }
```

Código 40: Implementação alternativa da função sleep para contornar a evasão de análise via longos delays

Funções Trampolim (1/2)

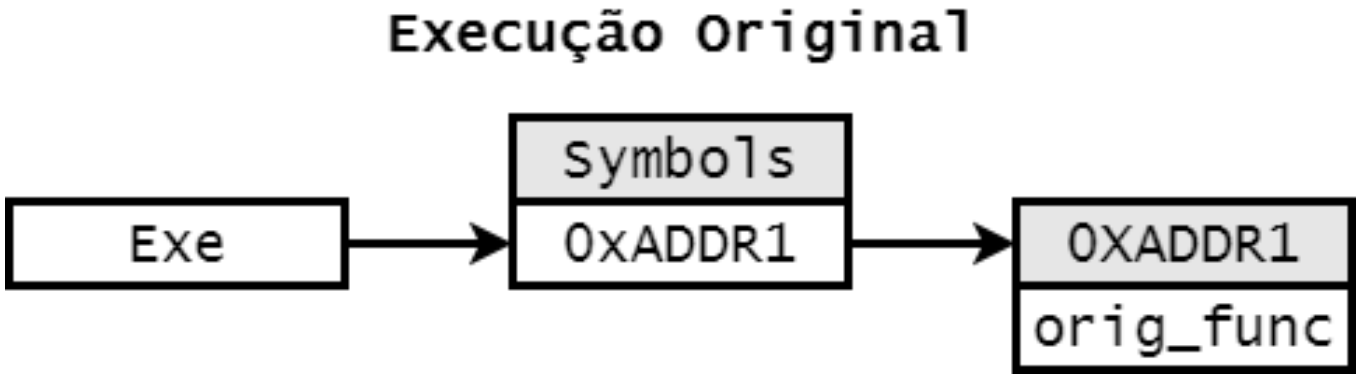

Figura: ***Logging* com LD_PRELOAD.** Fluxo original de execução através da invocação da chamada de função originalmente definida na tabela de símbolos.

Funções Trampolim (2/2)

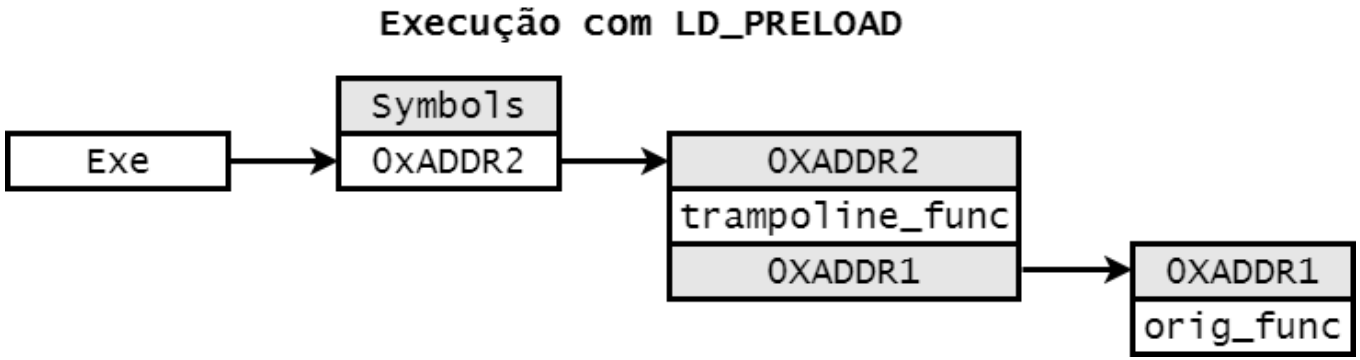

Figura: ***Logging* com LD_PRELOAD.** Fluxo de execução alterado após a modificação da tabela de símbolos por *LD_PRELOAD* para a invocação de uma função trampolim.

Implementando um *logger*

```
1  typedef int (*orig_puts_f_type)(const char *str);
2  int puts(const char *str){
3      FILE *fd = fopen("log","a+");
4      orig_fprintf_f_type orig_fprintf;
5      orig_fprintf = (orig_fprintf_f_type)dlsym(
           RTLD_NEXT,"fprintf");
6      orig_fprintf(fd,str);
7      fclose(fd);
8      orig_puts_f_type orig_puts;
9      orig_puts = (orig_puts_f_type)dlsym(RTLD_NEXT,"
           puts");
10     return orig_puts(str);
11 }
```

Código 41: Código da função trampolim injetada via LD_PRELOAD para registrar a invocação da função. Após o registro, deve-se invocar a função original para garantir a realização da operação

Rootkits

```
1  typedef struct dirent* (*orig_readdir_type)(DIR *
       dirp);
2    struct dirent *readdir(DIR *dirp){
3    struct dirent* valueOfReturn;
4    orig_readdir_type orig_readdir;
5    orig_readdir = (orig_readdir_type)dlsym(RTLD_NEXT,
         "readdir");
6    valueOfReturn = orig_readdir(dirp);
7    if(strcmp(HIDDEN,valueOfReturn->d_name) == 0)
8      return NULL;
9    return valueOfReturn;
10 }
```

Código 42: Exemplo de rootkit capaz de impedir a listagem de um arquivo.

Detectando Injeções de Código

```
1  int main()
2  {
3      if(getenv("LD_PRELOAD"))
4          printf("LD_PRELOAD␣detected␣through␣getenv()
               \n");
5      else
6          printf("Environment␣is␣clean\n");
7      if(open("/etc/ld.so.preload", O_RDONLY) > 0)
8          printf("/etc/ld.so.preload␣detected␣through␣
               open()\n");
9      else
10         printf("/etc/ld.so.preload␣is␣not␣present\n"
               );
11     return 0;
12 }
```

Código 43: Detecção da injeção de código via LD_PRELOAD

Roteiro

Monitoração de Sistemas de Arquivo

```
1 CREATE event: /home/sbseg/myfile.1
2 DELETE event: /home/sbseg/myfile
3 CREATE event: /home/sbseg/myfile2.1
4 DELETE event: /home/sbseg/myfile2
5 CREATE event: /home/sbseg/myfile3.1
6 DELETE event: /home/sbseg/myfile3
```

Código 44: Monitoração do diretório sbseg/ via inotify durante a execução do ransomware Erebus.

Regras da solução Audit

```
1 auditctl -w /etc/passwd -p r
2 auditctl -w /etc/shadow -p r
```

Código 45: Definição de regra do Audit para monitoração da leitura dos arquivos passwd e shadow.

Audit na Prática

```
1 type=PATH msg=audit(1565115273.377:158): item=0 name
    ="/etc/passwd" inode=2105354 dev=08:01 mode
    =0100644 ouid=0 ogid=0 rdev=00:00 nametype=
    NORMAL cap_fp=0000000000000000
2 type=PATH msg=audit(1565115273.377:159): item=0 name
    ="/etc/shadow" inode=2105378 dev=08:01 mode
    =0100644 ouid=0 ogid=0 rdev=00:00 nametype=
    NORMAL cap_fp=0000000000000000
3 type=SYSCALL msg=audit(1565115273.377:159): arch=
    c000003e syscall=2 success=yes exit=5 a0=7
    f0a44c46c50 a1=80000 a2=1b6 a3=80000 items=1
    ppid=2604 pid=4308 auid=4294967295 uid=1000 gid
    =1000 euid=0 suid=0 fsuid=0 egid=1000 sgid=1000
    fsgid=1000 tty=pts2 ses=4294967295 comm="su" exe
    ="/home/sbseg/dcow" key=(null)
```

Código 46: Log do Audit coletado durante um ataque DirtyCow.

Roteiro

Módulos de *Kernel*

```
1  #include <linux/init.h>
2  #include <linux/module.h>
3  #include <linux/kernel.h>
4  MODULE_LICENSE("GPL");
5
6  static int __init lkm_init()
7  {
8      printk(KERN_INFO "Kernel!\n");
9      return 0;
10 }
11 static void __exit lkm_exit()
12 {
13     printk(KERN_INFO "Tchau!\n");
14 }
15
16 module_init(lkm_init);
17 module_exit(lkm_exit);
```

Syscall Hooking

1. Localizar, na memória do *kernel*, a tabela de *syscalls*.
2. Localizar o deslocamento (*offset*) do apontador da tabela de *syscalls* para a função que executa a *syscall* originalmente presente no *kernel* e que será substituída.
3. Atribuir permissão de escrita ao segmento de memória que contém a tabela.
4. Substituir o apontador da *syscall* de interesse originalmente presente na tabela de *syscall* pelo apontador para a nova *syscall* contendo a função de interposição.
5. Remover a permissão de escrita da tabela.

Syscall Hooking

```
1  unsigned long ptr, *p;
2  static long (*sys_close) (unsigned int fd)=NULL;
3  sys_close=(void *)kallsyms_lookup_name("sys_close");
4  if (!sys_close){
5      return NULL;
6  }
7  for (ptr = (unsigned long)sys_close;
8  ptr < (unsigned long)&loops_per_jiffy;
9  ptr += sizeof(void *)){
10         p = (unsigned long *)ptr;
11         if (p[__NR_close] == (unsigned long)
               sys_close){
12             return (unsigned long **)p;
```

Código 47: Função find_sys_call_table() utilizada para identificar o endereço da tabela de chamadas de sistema.

Syscall Hooking

```
1  int ret; unsigned long cr0;
2  syscall_table = (void **)find_sys_call_table();
3  if (!syscall_table){
4      printk(KERN_DEBUG "[HOOK] Trying sys_call_table
           symbol\n");
5      syscall_table=(void **)kallsyms_lookup_name("
           sys_call_table");
6      if (!syscall_table){
7          printk(KERN_DEBUG "[HOOK] Cannot find the
               sys_call_table address\n");
8          return -EINVAL;
9      }
10 }
```

Código 48: Função __init do módulo de kernel que implementa o hook da syscall sys_uname.

Syscall Hooking

```
1  cr0 = read_cr0();
2  write_cr0(cr0 & (~CR0_WP));
3  do_set_memory_rw = (void *)kallsyms_lookup_name("
       set_memory_rw");
4  do_set_memory_ro = (void *)kallsyms_lookup_name("
       set_memory_ro");
5
6  if (do_set_memory_rw == NULL)
7  {
8       printk(KERN_DEBUG "[HOOK] Symbol not found: '
            set_memory_rw'\n");
9       return -EINVAL;
10 }
```

Código 49: Função __init do módulo de kernel que implementa o hook da syscall sys_uname.

Syscall Hooking

```
1  ret = do_set_memory_rw(PAGE_ALIGN((unsigned long)
       syscall_table),1);
2  if(ret){
3      printk(KERN_DEBUG
4      "[HOOK] Cannot set the memory to rw (%d) at addr
           0x%16lx\n",
5      ret, PAGE_ALIGN((unsigned long)syscall_table));
6      return -EINVAL;
7  }else{
8   printk(KERN_DEBUG "[HOOK] Syscall Table page set to
        rw\n");
9  }
10 orig_sys_uname = syscall_table[__NR_uname];
11 syscall_table[__NR_uname] = hook_sys_uname;
12 write_cr0(cr0);
```

Código 50: Função __init do módulo de kernel que implementa o hook da syscall sys_uname.

Syscall Hooking

```
1  asmlinkage long hook_sys_uname(struct new_utsname
       __user *name){
2      char msg[5]="Hook\0";
3      struct new_utsname tmp;
4      orig_sys_uname(name);
5      if(!copy_from_user(&tmp,name,sizeof(tmp))){
6          memcpy(tmp.sysname,msg,5);
7          if(copy_to_user(name,&tmp,sizeof(tmp)))
8            printk(KERN_DEBUG "[HOOK] Can't write to
                 user-buffer!\n");
9      }else{
10       printk(KERN_DEBUG "[HOOK] Can't copy user-
             buffer :(\n");
11     return 0;
12 }
```

Código 51: Função de interposição.

Syscall Hooking

```
1  uname -s; rmmod hook; dmesg; uname -s
2  [ 8001.652808] [HOOK] Symbol sys_close not found
3  [ 8001.652808] [HOOK] Trying sys_call_table symbol
4  [ 8001.656808] [HOOK] System call table at 0
       xffffffff8e6001e0
5  [ 8001.660797] [HOOK] Syscall Table page set to rw
6  [ 8001.660797] [HOOK] sys_uname: 0xffffffff8da93460
       - hook_sys_uname: 0xffffffffc0860000
7  [ 8003.016080] [HOOK] Inside hook_sys_uname
8  [ 8003.016080] [HOOK] Can't write to user-buffer!
9  [ 8073.156170] [HOOK] Inside hook_sys_uname
10 [ 8073.156170] [HOOK] uname->sysname: Linux
11 [ 8073.291740] [HOOK] released module
12 Linux
```

Código 52: Exemplo de saída do dmesg durante o carregamento do LKM implementando o hook de syscall.

Roteiro

O filtro de pacotes iptables

```
1 iptables -A INPUT -j ACCEPT
2 iptables -A FORWARD -j ACCEPT
3 iptables -A OUTPUT -j DROP
```

Código 53: Regras básicas iptables. O tráfego de saída pode ser bloqueado de modo a evitar que infecções se propaguem a partir da sandbox.

Filtragem de Protocolos

```
1 iptables -A INPUT -p tcp -j ACCEPT
2 iptables -A INPUT -p udp -j DROP
```

Código 54: Exemplos de políticas iptables. Regras podem ser definidas para cada protocolo.

Registro de Atividades

```
1 iptables -A INPUT -j LOG
2 iptables -A FORWARD -j LOG
3 iptables -A OUTPUT -j LOG
```

Código 55: Estabelecimento de políticas de log utilizando-se iptables para fins de monitoração.

Investigação na Prática

```
1 May 13 13:21:49 lab kernel: [ 3610.320968] IN= OUT=
    ens3 SRC=192.168.122.5 DST=91.189.89.196
2 May 13 13:21:49 lab kernel: [ 3610.321356] IN= OUT=
    ens3 SRC=192.168.122.5 DST=91.189.89.197
3 May 13 13:21:49 lab kernel: [ 3610.321503] IN= OUT=
    ens3 SRC=192.168.122.5 DST=91.189.89.198
4 May 13 13:21:49 lab kernel: [ 3610.321633] IN= OUT=
    ens3 SRC=192.168.122.5 DST=91.189.89.199
```

Código 56: Log de rede exemplificando a atuação de um malware do tipo scanner.

Roteiro

Sumário

- Soluções de análise na plataforma `Linux`.
- Comportamentos maliciosos e Evasão de análise.
- Desenvolvimento de módulos de *kernel*.

Dúvidas & Sugestões ?

Contato

mfbotacin@inf.ufpr.br
galante@lasca.ic.unicamp.br
otavios@lasca.ic.unicamp.br
paulo@lasca.ic.unicamp.br