The Darker Sides of Assembly

We've seen it.

Alex Radocea, Andrew Zonenberg
Moments in History

Thompson's Compiler Backdoor
http://cm.bell-labs.com/who/ken/trust.html

“I am a programmer. On my 1040 form, that is what I put down as my occupation. As a programmer, I write programs. I would like to present to you the cutest program I ever wrote. I will do this in three stages and try to bring it together at the end.”
This Script Kitty is more cute, right?

http://www.b3tards.com/v/1f879bbd15d3273880f9/1108.jpg
Moments in History

On November 2, 1988, Robert Morris, Jr., a graduate student in Computer Science at Cornell, wrote an experimental, self-replicating, self-propagating program called a worm and injected it into the Internet. He chose to release it from MIT, to disguise the fact that the worm came from Cornell.

http://groups.csail.mit.edu/mac/classes/6.805/articles/morris-worm.html
gets() payload in fingerd

Shellcode spotlight →
Robert Morris, Jr. worm (Spaf)

```
pushl $68732f  '/sh\0'
pushl $6e69622f  '/bin'
movl sp, r10
pushl $0
pushl $0
pushl r10
pushl $3
movl sp, ap
chmk $3b
```
Spaf does it right

- Best analysis ever on The Morris Worm:

8) The infection attempts proceeded by one of three routes: \texttt{rsh}, \texttt{finger}, or \texttt{sendmail}.

8a) The attack via \texttt{rsh} was done by attempting to spawn a remote shell by invocation of (in order of trial) \texttt{/usr/ucb/rsh}, \texttt{/usr/bin/rsh}, and \texttt{/bin/rsh}. If successful, the host was infected as in steps 1 and 2a, above.

8b) The attack via the \texttt{finger} daemon was somewhat more subtle. A connection was established to the remote \texttt{finger} server daemon and then a specially constructed string of 536 bytes was passed to the daemon, overflowing its input buffer and overwriting parts of the stack. For standard 4 BSD versions running on VAX computers, the overflow resulted in the return stack frame for the \texttt{main} routine being changed so that the return address pointed into the buffer on the stack. The instructions that were written into the stack at that location were:

\begin{verbatim}
pushl $58732f  '/sh\n0'
pushl $5e69522f  '/bin'
movl sp, r10
pushl $0
pushl $0
pushl r10
pushl $3
movl sp, ap
chkk $3b
\end{verbatim}

That is, the code executed when the \texttt{main} routine attempted to return was:

\begin{verbatim}
execve("/bin/sh", 0, 0)
\end{verbatim}

On VAXen, this resulted in the worm connecting to a remote shell via the TCP connection. The worm then proceeded to infect the host as in steps 1 and 2a, above. On Suns, this simply resulted in a core file since the code was not in place to corrupt a Sun version of \texttt{finger} in a similar fashion.

8c) The worm then tried to infect the remote host by establishing a连接 to the SMTP port and mailing an infection, as in step 2b, above.
Outline of Today's Agenda

- Moments in History
- Basic terminology
- Code injection
  - Shellcode
  - Building a virus
    - The ELF format
    - Injection Schemes
- ? Surprise us
**Terminology**

- **Backdoor** — Program allowing remote, covert access
- **Virus** — Parasitic program
- **Worm** — Self-propagating network-enabled program
- **Rootkit** — Tools to covertly maintain high-level system access
- **Malware/Spyware** — Harmful software (popups, password/CC sniffers....)
- **Botnet** — MMORPG – without the RPG
Code injection we care about

- Runtime Arbitrary Code Execution
  - Privileged Processes
  - Signed/Trusted Code Execution Environments
  - Remote programs
- Program File injection
- ???
Runtime Code Injection

- Remember all those crashmes?
- Local code injection
  - Command line arguments, environment, pathname, executable interpreter flags, program data (heap, stack, ...)
- Remote code injection
  - Program data
Writing your first shellcode.

● Goal:
  ● do not fork bomb anything
  ● Print a message to the screen
asm

BITS 32

; nasm -f elf code.asm; ld -o code.bin code.o; ./code.bin
; nasm -f bin code.asm ; ndisasm -u ./code

global _start
_start:
xor eax, eax
mov eax, 4
jmp data

back:
xor ebx, ebx
pop ecx
mov edx, 13
int 0x80

mov eax, 1
int 0x80

data:
call back
db "HI csci4971", 0x0a
demo
Minimization tips

• Data is code is data is code is data is code … (von Neumann arch vs Harvard)
• NUL byte safe?
  • Match constants to register sizes
  • Avoid some instructions
  • Use math to get values with NUL
  • Encoder/Decoder
Minimization Tips (II)

• Size problems?
  • Multi-staged payloads
    – Establish data transfer
    – Receive code
    – Decode it
    – Execute it

• Code crunch:
  • extra credz for shortest, self-contained d/l and execute binary code.
No shellcode necessary

- Ret2libc
  - Solar Designer '97
  - ...

Memory corruption can be hard, but also very easy

- Linux local bugs:
  - Off-by-one on gcc4 main()
    - Truncates frame pointer by one byte
  - Bypass ASLR
    - “patched up”
    - Still missing /proc/pid/stat
Writing a Virus

• Parasitic code
  • Injects into drivers, system code files, executable programs, runtime process memory, …

http://www.flickr.com/photos/quiplash/61424646/in/photostream/
Plan of Action

- Harmless Linux ELF Infector
  - Open a file
  - Expand size
  - Inject code
  - Update offsets
  - Save to filesystem
Useful links

- Cesare's http://vx.netlux.org/lib/static/vdat/tuunix02.htm
- Eresi: http://www.eresi-project.org/
- http://www.vx.netlux.org/lib/vrn00.html
- ...
The ELF Format

- ELF Header
  - Man 5 elf
- Program Headers
  - Runtime
- Section Headers
  - Link time
- Misc
More useful links

Linking View

- ELF header
- Program header table (optional)
- section 1
- ...
- section n
- ...
- ...
- Section header table

Execution View

- ELF header
- Program header table
- Segment 1
- Segment 2
- ...
- ...
- Section header table (optional)

http://users.csc.calpoly.edu/~mhaungs/paper/img7.gif
typedef struct {
    unsigned char e_ident[ EI_NIDENT ];
    uint16_t       e_type;
    uint16_t       e_machine;
    uint32_t       e_version;
    ElfN_Addr     e_entry;
    ElfN_Off      e_phoff;
    ElfN_Off      e_shoff;
    uint32_t      e_flags;
    uint16_t      e_ehsize;
    uint16_t      e_phentsize;
    uint16_t      e_phnum;
    uint16_t      e_shentsize;
    uint16_t      e_shnum;
    uint16_t      e_shstrndx;
} ElfN_Ehdr;
Program Headers

typedef struct {
    uint32_t p_type;
    Elf32_Off p_offset;
    Elf32_Addr p_vaddr;
    Elf32_Addr p_paddr;
    uint32_t p_filesz;
    uint32_t p_memsz;
    uint32_t p_flags;
    uint32_t p_align;
} Elf32_Phdr;

PT_LOAD
PT_INTERP

PF_X  An executable segment.
PF_W  A writable segment.
PF_R  A readable segment.
Using readelf/objdump/etc

• Demo
Some ELF File Infection strategies

- Overwrite existing code
  - Semantic nop injector (bukowski framework)
- Hijack GOT/PLT redirection
- Expand TEXT segment
- Insert new PF_X segment
- Replace Dynamic Interpreter
- Inject malicious shared object file paths
Simple infector

• >>
PHDR Injection

- Add a PF_X segment
- Add code
- Hijack entry point / branch
How do you do it all in asm?

- Need self propagation
- No compiler available (Sorry Ken)
All you need is...

- Open()
- Mmap()
- asm code
Infector demo
ELF Virus Detection

- Tripwire...
- Mismatched Section Headers
- Extra executable segments
- Strange shared libraries/dynamic interpreter
- Unusual entry point
  - Q: Can the entry point be outside of the TEXT segment?
- Linux AVs
- ????