Introduction to Malware
Outline

- Malware Introduction
- Understanding the malware
  - System monitoring
- Lab1 – Metasploit
- Lab2 – Shellcode/Malware
Lab Environment

1: Target Probing
2: Vulnerability Exploitation
3: Payload
What is Malicious Software?

- Malicious Software (a.k.a. Malware)
  - Software designed to infiltrate or damage a computer system, without the owner's **informed consent**
    -- http://en.wikipedia.org/wiki/Malware
  - Examples:
    - Viruses, worms, Trojan horses, spyware, and other malicious and unwanted software
  - How about adware?

- Malware references the intent of the creator, rather than any particular features
  - DRM v.s. Rootkit (e.g., Sony CD rootkit)
Why should we care?

- Malware remains a top threat

Source: Symantec Internet Security Threat Report
Recent Trends

- Recruiting Vulnerable Nodes ➔ *Attack Network*
  - Zero-day exploits w/o software patches
  - New attack strategies
    - Exploiting vulnerable client-side software, such as IE
    - Propagating malware with RFID tags
  - Fast massive replications

- Providing “Value-Added” Services
  - DDoS, spamming, or other malicious purposes …
  - Sell/rent attack networks for profit
Taxonomy of Malicious Software

- Malicious Programs
  - Needs Host Program
    - Trapdoors
    - Logic Bombs
    - Trojan Horses
    - Viruses
  - Independent
    - Worms
    - Zombies
    - Rootkits

Replicate
Trapdoor

- Secret entry point into a system
  - Specific user identifier or password that circumvents normal security procedures.
- Commonly used by developers
  - Could be included in a compiler.
Logic Bomb

- Embedded in legitimate programs
- Activated when specified conditions met
  - E.g., presence/absence of some file; Particular date/time or particular user
- When triggered, typically damages system
  - Modify/delete files/disks
Trojan Horse

- Program with an expected and hidden effect
  - Appears normal/expected
  - Hidden effect violates security policy

- User tricked into executing Trojan horse
  - Expects (and sees) expected behavior
  - Hidden effect performed with user’s authorization

- **Attacker:**
  ```
cat >/homes/victim/ls <<eof
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls
ls $*
eof
```

- **Victim**
  ```
  ls
  ```
Virus

- Self-replicating code
  - Alters normal code with “infected” version
  - Generally tries to remain undetected

- Operates when infected code executed
  
  If spread condition then
  
  For target files
    if not infected then alter to include virus
  
  Perform malicious action
  
  Execute normal program
Virus Types (I)

- **Boot Sector**
  - Problem: How to ensure virus “carrier” executed?
  - Solution: Place in boot sector of disk
    - Run on any boot
  - Propagate by altering boot disk creation
    - *Less common with few boots off floppies*

- **Executable**
  - Malicious code placed at beginning of legitimate program
  - Runs when application run
Virus Types (I)

- **Macro Virus**
  - Infected “executable” isn’t machine code
    - Relies on something “executed” inside application data
    - Common example: Macros
  - Otherwise similar properties to other viruses
    - Architecture-independent
    - Application-dependent

- **Question:**
  - How about a program that modifies a configuration file, including boot sector?
Virus Types (II)

- **Armored**
  - Encrypt virus
    - Prevents “signature” to detect virus

- **Polymorphic**
  - Change virus code to prevent signature

- **Stealth**
  - Conceal Infection
    - Trap read and disinfect
    - Let execute call infected file
  - Terminate and Stay Resident (TSR)
    - Stays active in memory after application complete
    - Allows infection of previously unknown files
      - Trap calls that execute a program
Worm

- Runs independently
  - Does not require a host program
- Propagates a fully working version of itself to other machines
- Carries a payload performing hidden tasks
  - Backdoors, spam relays, DDoS agents; ...
- Phases
  - Probing ➔ Exploitation ➔ Replication ➔ Payload
Worm Propagation

A Staged View of Worm Infection

1: Target Probing
2: Vulnerability Exploitation
3: Replication
1. Exploits target on port 135/TCP
2. Binds svchost.exe to port 4444/TCP via injected code
3. Connects to target on port 4444/TCP
4. Creates a shell “cmd.exe” and binds it to port 4444/TCP
5. Creates “TFTP Server” on port 69/UDP
6. Sends “TFTP” command to shell
7. Runs TFTP command; “teleports” msblast.exe file
8. Sends “START msblast.exe” command
9. Runs worm on target!
10. Closes connection
11. Shell closes

```
alert ip $EXTERNAL_NET any -> $HOME_NET 135 (msg:"RPC DCOM exploit/ Blaster Worm Attack");
```
Example II: Lion/Linux Worm

1. Checks reachability by connecting to 192.168.0.1 on port 53/TCP.

2. Exploit target on port 53/TCP. 
   - alert tcp $EXTERNAL_NET any -> $HOME_NET 53 (msg:"DNS EXPLOIT named tsig overflow attempt"; flow:to_server,established; content:"|AB CD 09 80 00 00 01 00 00 00 00 01 00 01 20 20 20 20 02 61|"; reference:cve,CVE-2001-0010; ...)

3. Replicate worms on port 27374/TCP.
   - alert tcp $HOME_NET any -> $EXTERNAL_NET 27374 (msg:"MISC Lion worm"; flow:to_server,established; content:"GET "; depth:8; nocase; sid:514; ...)

Lion
192.168.0.1

Target/Bind
192.168.10.11
Zombie

- Secretly takes over another networked computer by exploiting software flows.
- Builds the compromised computers into a zombie network or botnet.
- Uses it to indirectly launch attacks.
  - E.g., DDoS.
Detailed Steps (1)

1. Attacker scans Internet for unsecured systems that can be compromised
Detailed Steps (2)

2. Attacker secretly installs zombie agent programs, turning unsecured computers into zombies.
Detailed Steps (3)

3. Zombie agents `phone home` and connect to a master server.

Attacker

Master Server

Zombies

Internet
Detailed Steps (4)

4. Attacker sends commands to Master Server to launch a DDoS attack against a targeted system.
Detailed Steps (5)

5 Master Server sends signal to zombies to launch attack on targeted system

Attacker

Master Server

Zombies

Internet

Targeted System System
Detailed Steps (6)

6. Targeted system is overwhelmed by zombie requests, denying requests from normal users.

- Attacker
- Master Server
- Zombies
- Request Denied
- Targeted System
- Internet
A rootkit is a set of programs and code that allows a permanent or consistent, undetectable presence on a computer.

**Goals:**
- **Hide malicious resources** (e.g., processes, files, registry keys, open ports, etc.)
- **Provide hidden backdoor access**
ls               Trojaned! Hide files
du               Trojaned! Hide files
ifconfig         Trojaned! Hide sniffing
netstat          Trojaned! Hide connections
chfn             Trojaned! User->r00t
chsh             Trojaned! User->r00t
inetd            Trojaned! Remote access
login            Trojaned! Remote access
passwd           Trojaned! User->r00t
ps               Trojaned! Hide processes
top              Trojaned! Hide processes
rshd             Trojaned! Remote access
syslogd          Trojaned! Hide logs
linsniffer       Packet sniffer!
fix              File fixer!
z2               Zap2 utmp / wtmp / lastlog eraser!
wted             wtmp / utmp editor!
lled             lastlog editor!
bindshell        port / shell type daemon!
tcpd             Trojaned! Hide connections, avoid denies
Rootkit

- Simple rootkits:
  - Modify user programs (ls, ps)
  - Detectable by tools like Tripwire

- Sophisticated rootkits:
  - Modify the kernel itself
  - Hard to detect from userland
Rootkit Classification

Application-level Rootkit (I)
- Trojan login
- Trojan ps
- Trojan ifconfig
- good tripwire

Kernel
- Lrk5, t0rn

Application-level Rootkit (II)
- Evil Program
- good login
- good ps
- good ifconfig
- good tripwire

Kernel
- Hxdef, NTIllusion

Kernel-level RootKit
- Trojan Kernel Module
- Shadow Walker, adore
Rootkit Classification

Under-Kernel RootKit

- good login
- good ps
- good ifconfig
- good tripwire

Kernel

Evil VMM

SubVirt, "Blue Pill"
System Monitoring

- Requirement
  - Deep Inspection
  - Tamper-Resistance

- Two main approaches (state of the art)
  - Internal monitoring (e.g., API hooking, system call logging)
  - External monitoring (e.g., traffic sniffers)
API Hooking

User Application

Kernel32.DLL

NTDLL.DLL

EAX = 0x00000020

call INT 2Eh

Switch Back to USER MODE

Log

USER MODE

KERNEL MODE

NTOSKRNL.EXE
Example API Hooking Log

- iexplore.exe (268) : File::Write (C:\WINDOWS\Downloaded Program Files\ieloader.exe)
- iexplore.exe (268) : Sys::Execute (C:\WINDOWS\Downloaded.exe)
- ieloader.exe (1728) : Reg::SetValue (HKLM\Software\Microsoft\Windows\CurrentVersion\Run, FX)
- iexplore.exe (268) : Net::Connect (205.205.86.51, 80)
- ieloader.exe (1728) : File::Write (C:\WINDOWS\System32\uvbdcgrtjce.dll)
- ieloader.exe (1728) : Sys::Execute (C:\Program.exe)
- iexplore.exe (268) : File::Write (C:\Documents and Settings\HS\Local Settings\Temporary Internet Files \Content.IE5\EJ89IPOV\init[1].js)
- iexplore.exe (1812) : Reg::SetValue (HKCU\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders, Desktop)
- iexplore.exe (1812) : Reg::SetValue (HKCU\Software\Microsoft\Windows\ShellNoRoam\BagMRU, NodeSlots)
- iexplore.exe (1812) : Reg::SetValue (HKCU\Software\Microsoft\Windows\ShellNoRoam\BagMRU, MRUListEx)
- iexplore.exe (268) : File::Write (C:\WINDOWS\Downloaded Program Files\SET4A.tmp)
- iexplore.exe (268) : File::Write (C:\WINDOWS\Downloaded Program Files\ieloader.exe)
- iexplore.exe (268) : Sys::Execute (C:\WINDOWS\Downloaded.exe)
- ieloader.exe (1728) : Reg::SetValue (HKLM\Software\Microsoft\Windows\CurrentVersion\Run, FX)
- iexplore.exe (268) : Net::Connect (205.205.86.51, 80)
- ieloader.exe (1728) : File::Write (C:\WINDOWS\System32\uvbdcgrtjce.dll)
- ieloader.exe (1728) : Sys::Execute (C:\Program.exe)
- iexplore.exe (268) : File::Write (C:\Documents and Settings\HS\Local Settings\Temporary Internet Files \Content.IE5\EJ89IPOV\init[1].js)
System Call Logging

User Space

Kernel Space

System Call Dispatcher

System Call Table

wrap_restart_syscall
wrap_exit
wrap_fork
wrap_read
wrap_write
wrap_ni_syscall

sys_restart_syscall
sys_exit
sys_fork
sys_read
sys_write
sys_ni_syscall

Syscall-wrapping interception

fork("/bin/sh")
Example System Call Log

... 673["sendmail"]: 5_open("/proc/loadavg", 0, 438) = 5
673["sendmail"]: 192_mmap2(0, 4096, 3, 34, 4294967295, 0) = 1073868800
673["sendmail"]: 3_read(5, "0.26 0.10 0.03 2...", 4096) = 25
673["sendmail"]: 6_close(5) = 0
673["sendmail"]: 91_munmap(1073868800, 4096) = 0

... 2568["httpd"]: 102_accept(16, sockaddr[2, cbbdff3a], cbbdff38) = 5
2568["httpd"]: 3_read(5, "\1281\1\0\2\0\24...", 11) = 11
2568["httpd"]: 3_read(5, "\7\0À\5\0\128\3\...", 40) = 40
2568["httpd"]: 4_write(5, "\132@\4\0\1\0\2\...", 1090) = 1090

... 2568["httpd"]: 4_write(5, "\128\19Ê\136\18\...", 21) = 21
2568["httpd"]: 63_dup2(5, 2) = 2
2568["httpd"]: 63_dup2(5, 1) = 1
2568["httpd"]: 63_dup2(5, 0) = 0
2568["httpd"]: 11_execve("/bin//sh", bffff4e8, 00000000)
2568["sh"]: 5_open("/etc/ld.so.prelo...", 0, 8) = -2
2568["sh"]: 5_open("/etc/ld.so.cache", 0, 0) = 6