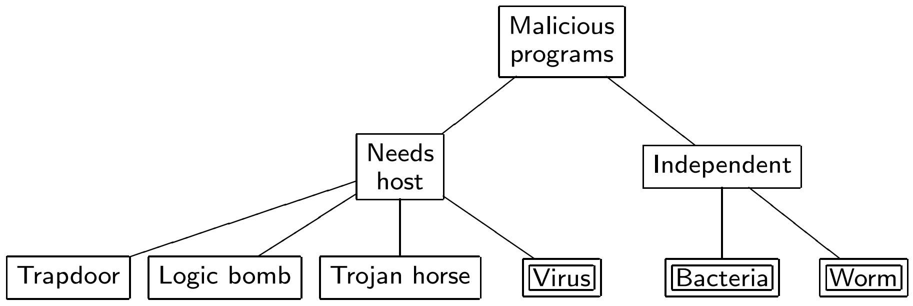
Malware, Part 2

CSC 348·648

Wake Forest University
Department of Computer Science

Spring 2013

Malicious Program Categories



- Can categorize based on independence
  - **Host program** - fragments of programs that cannot exist alone
  - **Independent** - self-contained programs that can be scheduled and run by the operating system
- Can also differentiate based on program replication
  - Some programs may produces copies of itself

Another Backdoor

- Buffer overflow in BIND, root on Lockheed Martin's DNS server
  - Install password sniffer
  - Sniffer logs stored in directory called `/var/adm/␣...` (yes, a space)
- Excite@Home employees connect via dialup
  - Attacker installs remote access trojans on their machines
  - Sniffs IP addresses of interesting targets
  - Commercial remote-access software, modified to be *invisible*

Chernobyl Virus

- CIH (Chernobyl or Spacefiller) is a Windoze virus written in 1998
  - Destructive malware, it overwrites most of hard drive
  - Will try to overwrite the Flash BIOS, if successful the computer will be unable to boot and the chip must be reprogrammed
- Found in multiple places: cracked version of Windoze, Wing Commander game, Ahamay firmware for CD-R drives, demo version of Noisivitca game, MBI computers (*what up now Nostaw?*)
- Use several technique to hide
  - Size of the infected files does not grow (*virus code is around 1 KB*)
  - Virus jumped from processor ring 3 to ring 0 to use system calls
  - Dead code insertion (spacefiller), block reordering, etc...

Some Example Chernobil Instructions

Machine	Assembler
`5B 00 00 00 00`	`pop ebx`
`8D 4B 42`	`lea ecx, [ebx + 42h]`
`51`	`push ecx`
`50`	`push eax`
`50`	`push eax`
`0F 01 4C 24 FE`	`sidt [esp - 02h]`
`5B`	`pop ebx`
`83 C3 1C`	`add ebx, 1Ch`
`FA`	`cli`
`8B 2B`	`mov ebp, [ebx]`

```
5B 00 00 00 00 8D 4B 42 51 50 50 0F 01 4C 24 FE 5B
83 C3 1C FA 8B 2B
```

Chernobool Dead Code

Machine	Assembler
`5B 00 00 00 00`	`pop ebx`
`8D 4B 42`	`lea ecx, [ebx + 42h]`
`51`	`push ecx`
`50`	`push eax`
`90`	`nop`
`50`	`push eax`
`40`	`inc eax`
`0F 01 4C 24 FE`	`sidt [esp - 02h]`
`48`	`dec eax`
`5B`	`pop ebx`
`83 C3 1C`	`add ebx, 1Ch`
`FA`	`cli`
`8B 2B`	`mov ebp, [ebx]`

```
5B 00 00 00 00 8D 4B 42 51 50 90 50 40 0F 01 4C 24
FE 48 5B 83 C3 1C FA 8B 2B
```

Cheerynoble Instruction Reordering

Machine	Assembler
`5B 00 00 00 00`	`pop ebx`
`EB 09`	`jmp <S1>`
`S2:`	
`50`	`push eax`
`0F 01 4C 24 FE`	`sidt [esp - 02h]`
`5B`	`pop ebx`
`EB 07`	`jmp <S3>`
`S1:`	
`8D 4B 42`	`lea ecx, [ebx + 42h]`
`51`	`push ecx`
`50`	`push eax`
`EB F0`	`jmp <S2>`
`S3:`	
`83 C3 1C`	`add ebx, 1Ch`
`FA`	`cli`
`8B 2B`	`mov ebp, [ebx]`

```
5B 00 00 00 00 EB 09 50 0F 01 4C 24 FE 5B EB 07 8D
4B 42 51 50 EB F0 83 C3 1C FA 8B 2B
```

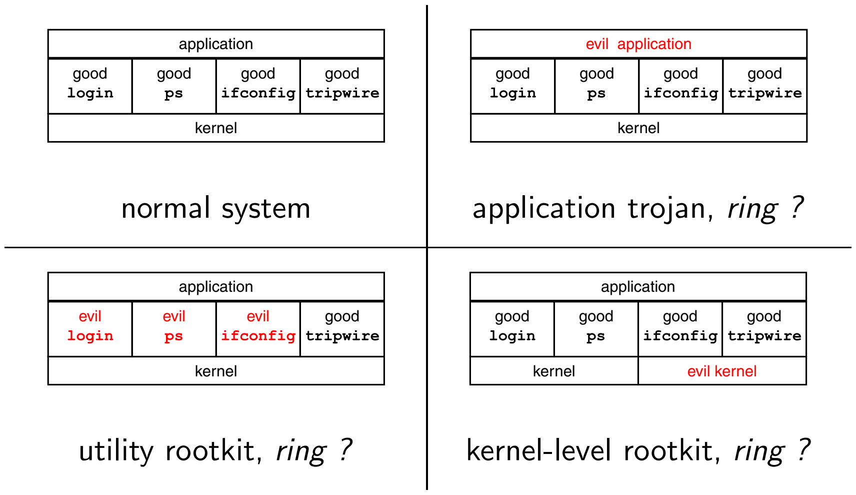
Rootkits

- Rootkit is a set of Trojan system binaries
  - Important characteristic is stealthiness (hides from host's owner)
  - Binaries allow attacker to circumvent authentication and authorization
- Typical infection path (*attack vector*)
  - Use stolen password or dictionary attack to log in
  - Use buffer overflow in `rdist`, `sendmail`, `loadmodule`, `rpc.yp updated`, `lpr`, or `passwd` to gain `root` access
  - Download rootkit, unpack, compile and install
  - Might include network and/or keyboard sniffer(s)
- Rootkits can target firmware, a hypervisor, kernel, or user-mode applications

Rootkit Taxonomy (or an attempt...)

The analogy will be: *"Suppose Chukar is trying to kill someone by poisoning their soup (or sweet berry wine)... now back to rootkits"*

- Traditional rootkits replace system utilities
  - Can be detected using hashes
  - Soup analogy: *attacker places poisonous vegetables in your soup (like tomatoes)... you can taste a small amount of each vegetable and tell if it is poisonous*
- Kernel-level rootkits replace portions of the kernel
  - System utilities are the same, just *bad* information is sent to them from the kernel
  - Soup analogy: *attacker places poison vegetables in your soup and replaces your tongue... hey, tomatoes are awesome...*



"Regardless of the whether they [rootkits] are implemented in a shared library or as a device driver, contemporary rootkits do their work by hooking and rewriting selected services that provide information about the current state of the system (i.e. processes, files, and network objects)" –Meyers and Youndt, "An Introduction to Hardware-Assisted Virtual Machine (HVM) Rootkits"

Bootkits

- Bootkit (boot + rootkit) replaces legitimate bootloader with another
  - Starting a boot, malware can exist in protected mode

    So is this a virus? Stoned virus did this...

- Bootkits can been used to...
  - Stoned bootkit can obtain keys if the laptop is fully encrypted

    > *"During startup, the BIOS first calls the bootkit, which in turn starts the TrueCrypt boot loader. Kleissner [Stoned Bootkit author] says that he neither modified any hooks, nor the boot loader, itself to bypass the TrueCrypt encryption mechanism ... bootkit uses a double forward to redirect I/O interrupt 13h, which allows it to insert itself between the Windows calls and TrueCrypt ... Once the operating system has been loaded, Stoned can get to work and install malware, such as a banking trojan, in the system"*

  - Subvert Windoze 7 64-bit kernel-mode driver signing
- *Only defense, prevent physical access*

Hypervisor Rootkits

- Consider a Type II hypervisor to manage Virtual Machines (VMs)

  > *"Type 2 (or hosted) hypervisors run within a conventional operating system environment. With the hypervisor layer as a distinct second software level, guest operating systems run at the third level above the hardware."*

- This type of rootkit installs and *hosts* the target OS as a VM
  - Can intercept information sent to/from the virtual target OS
  - Does not require any modification to the host OS

    So what?

- Using the ring model for processes, the hypervisor runs in ring -1

  > *"HVM rootkits, theoretically at least, are not vulnerable to any action the OS can take since the rootkits runs in a more privileged state than the OS."*

  - Bootstrap the hypervisor and start the host OS as a VM
  - Processors (Intel Vanderpool and AMD Pacifica) assist virtualization, increasing speed and access to hardware

Hiding Rookit's Presence

- Create a hidden directory
  - `/dev/.lib` or `/usr/src/.cantSeeThis`
  - Often use invisible characters in directory name (why?)
- Install hacked binaries for system programs
  - For example `netstat`, `ps`, `ls`, `du`, `login` ...
    Why?
- Modified binaries have same checksum as originals

  "A widely used cracker program named `fix` will take a snapshot of the system binary to be replaced. When the trojaned or modified binary is moved into place, the `fix` program mimics all three timestamps (atime, ctime, and mtime) and CRC checksum of the original program. A carefully constructed ... binary will also have the same length."

The Power of Redirection

- Most kernel-level rootkits perform *execution redirection*
  - Intercept a call to run a utility, then call another *modified* application instead
- For example, consider `login`
  - Rootkit installs `evilLogin` in `/bin`
  - When the user attempts a `login` the `evilLogin` program is executed instead
  - The original `login` is still there (MD5 check passes), but it is never really executed
- Most kernel-level rootkits support *file hiding*
  - If a user lists the files in the directory, the corrupt files are not displayed

Function Hooking

- Replace pointer to function with an address of malicious code

  "A basic hook redirects execution flow by changing function start or a function pointer but there is no single way to hook a routine"

  Pointer hooking

  – Modify the pointer in OS Global Offset Table, where function addresses are stored

  Detour, **proxy**, or **inline** hooking

  – Insert a jump in first few bytes of a legitimate function

  ```
  FARJMP 0x7fffffffe0a8
  ```

  where `0x7fffffffe0a8` is the address of the malicious function

  How can this be detected?

Sony BMG Rootkit

- 2005 Sony published CD with RDM software
  - Extended Copy Protection (XCP) played music and installed rootkit
  - Rootkit limited the user's access to the CD
- XCP actually did the following
  - Installed a device driver with kernel level privileges
  - Driver hid DRM files and processes that prevent *illegal* coping
  - Kernel code would hide any file, folder, or process that had a name starting with "`$sys$`."

    OK, DRM and EULA debate aside, what is the problem?
- *Is this the only example?* read about Semantic and `NProtect`

Rootkit Detection

- Sad way (*AKA DigitalBear-style*)
  - DigitalBear runs out of physical disk space because of sniffer logs
  - Sadly, logs are invisible because `du` and `ls` have been hacked
- Manual confirmation
  - Reinstall clean `ps` and see what processes are running
  - Use LiveCD and mount drive
  - Use alternatives for common utilities *For example?*
  - Try rootkit type of commands
- Automatic detection
  - Rootkit often do not alter the data structures normally used by `netstat`, `ps`, `ls`, `du`, `ifconfig`
  - Host-based intrusion detection find rootkit, *if not p0wned...*

Worms

- A worm is a *self-replicating* program
  - Does not require another program to exist or replicate
  - Typically does not require human-action to execute

    So what?
- Four stages of the worm *life-cycle*
  1. **Target selection**, find a potential host (target)
  2. **Exploitation**, compromise the target using a vulnerability
  3. **Infection**, replicate (copy) itself to the target
  4. **Propagation**, find a potential host (target)

     The difference between target selection and propagation?

Worm Life-Cycle Examples

- A victim of the Lion worm would see
  1. **Target selection**, connection attempt on port 53
  2. **Exploitation**, a BIND exploit on port 53
  3. **Infection**, outbound connection to a web-site to download the worm and install a backdoor *Why not load from attacker?*
  4. **Propagation**, series of scans for port 53
- A victim of the Ramen worm would see
  1. Connection attempts on ports 21, 111, and 515
  2. Exploit `wu-ftp` and/or `rpc.statd`
  3. Download worm and install backdoor
  4. Scan for ports 21, 111, and 515

Famous Worms

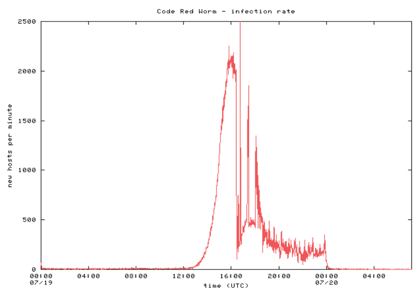
Name	**Date**	**Distinction**
Morris	11/88	Used multiple vulnerabilities, looked local
ADM	5/98	Random scanning for addresses
Ramen	1/01	Exploited three vulnerabilities
L10n	3/01	Rootkit worm
Cheese	6/01	Vigilante worm, that smoked your stash
Code Red	7/01	First significant Windows worm, memory resident and delicious
Walk	8/01	Recompiled at the host
Nimda	9/01	Windows worm, C2S, C2C, S2S, ...
Scalper	6/02	11 days after announcement, P2P
Slammer	1/03	Only requires single UDP packet, messed w/ yo GF/BF
Witty	5/04	Attacks ISS security software (not funny)
Zotob	8/05	"covered live on CNN television, as the network's own computers got infected"
Brepibot	6/06	Uses the Sony XCP *features*... actually a trojan
Stuxnet	9/10	Targets SCADA using USB devices for propagation (*hippynet?*)

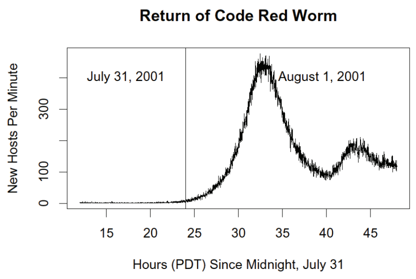
Code Red

- Initial version released July 13, 2001
  - Sends its code as an HTTP request
  - HTTP request exploits buffer overflow

```
GET /default.ida?NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN%u9090
%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090%u6858%ucbd3%u7801%u9090
%u9090%u8190%u00c3%u0003%u8b00%u531b%u53ff%u0078%u0000%u00=aHTTP/1.0
```

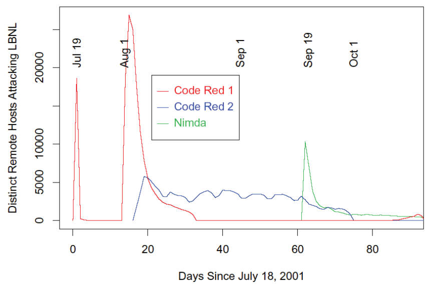
  - Malicious code exists in memory
- When executed it...
  - Checks is worm already exists (`c:/Notworm`), if it exists the worm goes into infinite sleep state
  - Creates new threads, depending on the date the threads...





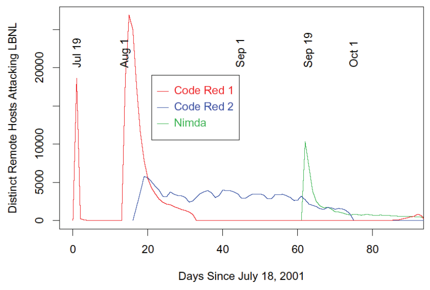
- Initial release
  - 1st through 20th, 99 threads created for random IP search and spread
  - 20th through the end of the month, flood `www.whitehouse.gov`
  - Failure of random seed generator resulted in linear growth
- *Only requires one instance to start it again...* (August 1st above)

Code Red Part 2



- Released August 4, 2001
  - Commented in code as "`Code Red 2`", but entirely new code base
- Payload is a rootkit, can withstand reboots
  - Contains a bug, crashes NT, only works on Windows 2000
  - Performed *localized scanning* and kills Code Red 1
  - Safety valve added, program terminates October 1, 2001

Nimda



- Released September 18, 2001, multiple forms to spread
  - Attack IIS servers via infected clients and emails copies
- Worm creates a type of *ecosystem*
  - Copies itself on network shares and modify web pages with exploit
  - Scanned for Code Red 2 backdoor (*whoze messin w/ yo GF now?*)
- Leaped across firewalls (*yes, physically leaped*)

Nimda Target Acquisition

- Nimda uses probabilities for generating target IP addresss
  - 50% of the time the first 16 bits of the address fixed, remaining least significant bits random
  - 25% of the time the first 8 bits of the address fixed, remaining least significant bits random
  - 25% of the time the entire address was random
- Localizes network propagation and finds local hosts

  Why? What is the advantage?

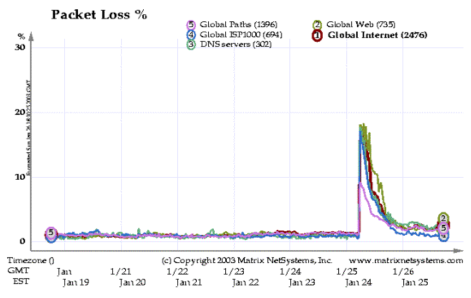
Slammer (Sapphire) Worm

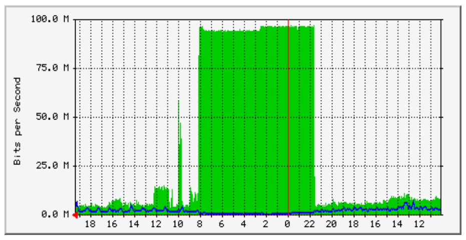
- January 2003: UDP worm exploiting overflow in Microsaft SQL Server
  - Overflow was already known and patched by Microsoft
  - But not everybody installed the patch
- Entire code fits into a single 404-byte UDP packet
  - Normal buffer overflow
  - Worm binary followed by overflow pointer back to itself

> *"The worm is so small that it does not contain code to write itself to disk, so it only stays in memory, and it is easy to remove. For example, Symantec provides a free removal utility (see external link below), or it can even be removed by restarting SQL Server (although the machine would likely be immediately reinfected)."*

- Random scanning used to propagate
  - Randomly generates IP addresses and sends itself to port 1434
  - MS-SQL listens at port 1434

Slammer Impact





- Approximately scan rate 55,000,000 addresses per second
- Initial infection was **doubling** in 8.5 seconds
  - *Doubling time of Code Red was only 37 minutes*
- Saturated carrying capacity of the Internet in 10 minutes
  - 75,000 SQL servers compromised
  - Broken pseudo-random number generator used for IP address generation had no impact...

- No malicious payload, but it stopped several critical infrastructures
  - Bank of America ATM network Entire cell phone network in South Korea Five root DNS servers...

Why was Slammer so successful?

Target Selection

- A worm must find a target before it can infect
  - Several different methods for finding potential targets
- **Scanning**, testing a set of addresses, two simple approaches
  - Sequential scanning tries *blocks* of addresses
  - Random scanning tries addresses outside of blocks
- **Pre-generated lists**, testing predefined addresses (*hit-list*)
  - Allows the existence of *flash-worms*
- **Externally generated lists**, testing predefined addresses
  - List is maintained by a separate server (*metaserver*) or source
  - For example, use Google to discover web-servers
- **Contagion**, use normal communications
  - Parasitically use communications initiated by the user

- **Internal target list**, testing predefined *local* addresses
  - Many hosts keep certain addresses local (`/etc/hosts` file)
  - Such target lists can be used to create *topological worms* which discover the topology of the network
  - Warhol or Flash worms generally use target lists

    So what?
- **Passive**, worm does not seek new hosts
  - Wait for a legitimate connection to the current host
  - Has no anomalous traffic patterns during target selection phase, so very difficult to find

Nimda Target Acquisition

- Nimda uses probabilities for generating target IP addresses
  - 50% of the time the first 16 bits of the address fixed, remaining least significant bits random
  - 25% of the time the first 8 bits of the address fixed, remaining least significant bits random
  - 25% of the time the entire address was random
- Localizes network propagation and finds local hosts

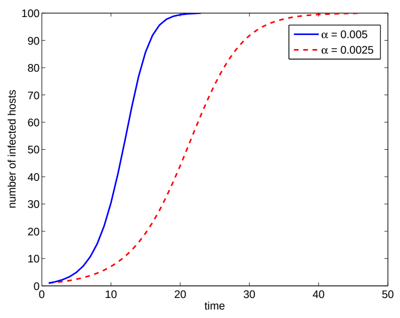
  Why? What is the advantage?

Infection Rate

- Consider a discrete-time epidemic model

$$I_t = (1 + \alpha)I_{t-1} - \beta I_{t-1}^2$$

  - Where I_t is the infected population at time t, α infection rate per infected host, β is the pairwise rate of infection
- Assume all hosts are vulnerable and a simple uniform scan worm



  - The curve depicts *slow start*, *fast spread*, and *slow finish* phases

Worm Distribution

- *How does the worm propagate to the victim?*
  - Again, several different approaches
- **Self carried**, worm actively transmits code
- **Second channel**, worm uses a different channel to transmit code
  - Blaster required a secondary channel to send the code (`TFTP`)
- **Embedded**, worm transmits code during **normal** communication
  - Appending to or replacing normal messages
  - As a result, *propagation does not appear anomalous*
  - Only useful if target selection is also stealthy

Using Search Engines

- Random address searching can be detected
  - Typically associated with malware, attempts to propagate
- Possible to use search engine to find vulnerable servers
  - Used by MyDoom.0 and Santy
  - Santy exploited phpBB and used Google to ...

    "exploits bug in the phpBB bulletin system that allows an adversary to run arbitrary code on the web server. To find vulnerable servers to infect, it uses Google to search for URLs that contain the string viewtopic.php"
- *Is this a more intelligent way of propagating the worm?*

  "Although, the number of infected machines stayed in the low thousands during the outbreak, the actual query traffic was larger as infected web servers were often well connected and ended up running multiple instances of the worm for each vulnerable virtual host."

Solutions for the First Three Stages

- Detect target selection, notice port scans across IP addresses
  - Test for x events across a y-size time interval
  - Notice one source IP scanning different destination IP

    When does this type of detection not work?

- Detecting or preventing exploits
  - Keep systems updated with latest patches
  - Most worms in existence rely on *known* vulnerabilities

    So patches solve this problem? What about IDS signatures?

- Detecting infection
  - Anti-virus and tripwire products can detect known worms
  - Use honeypot to detect **new** worms