Malicious Software

CSCI 454/554

Malicious Software

- Virus
- Worm
- Trojan horse
- Logic bomb
- Trapdoor
- Independent

Replicate
**Trapdoors**

- secret entry point into a program
- allows those who know access bypassing usual security procedures
- have been commonly used by developers
- a threat when left in production programs allowing exploited by attackers
- very hard to block in O/S
- requires good s/w development & update

**Logic Bomb**

- one of oldest types of malicious software
- code embedded in legitimate program
- activated when specified conditions met
  - eg presence/absence of some file
  - particular date/time
  - particular user
- when triggered typically damage system
  - modify/delete files/disks
Trojan Horse

- program with hidden side-effects
- which is usually superficially attractive
  - eg game, s/w upgrade etc
- when run performs some additional tasks
  - allows attacker to indirectly gain access they do not have directly
- often used to propagate a virus/worm or install a backdoor
- or simply to destroy data

Zombie (bot)

- program which secretly takes over another networked computer
- then uses it to indirectly launch attacks
- often used to launch distributed denial of service (DDoS) attacks
- exploits known flaws in network systems
Bot Remote Control Facility

- distinguishes a bot from a worm
  - worm propagates itself and activates itself
  - bot is initially controlled from some central facility
- typical means of implementing the remote control facility is on an IRC server
  - bots join a specific channel on this server and treat incoming messages as commands
  - more recent botnets use covert communication channels via protocols such as HTTP
  - distributed control mechanisms use peer-to-peer protocols to avoid a single point of failure

Viruses

- a piece of self-replicating code attached to some other code
  - cf biological virus
- both propagates itself & carries a payload
  - carries code to make copies of itself
  - as well as code to perform some covert task
Virus Operation

- virus phases:
  - dormant – waiting on trigger event
  - propagation – replicating to programs/disks
  - triggering – by event to execute payload
  - execution – of payload
- target at specific machine/OS
  - exploiting features/weaknesses

Types of Viruses

- parasitic virus
- memory-resident virus
- boot sector virus
- stealth
- polymorphic/metamorphic virus
- macro virus
- email virus
Macro Virus

- **macro virus** infects **documents** (data files), not executable files
  - macro code embedded in word processing file
- macro virus is platform independent
- is a major source of new viral infections
- blurs distinction between data and program files making task of detection even harder
- classic trade-off: "ease of use" vs "security"

Email Virus

- spread using email with attachment containing a macro virus
  - e.g Melissa
- triggered when user opens attachment
- or worse even when mail viewed by using scripting features in mail agent
- usually targeted at Microsoft Outlook mail agent & Word/Excel documents
Worms

- replicating but not infecting program
- typically spreads over a network
  - cf Morris Internet Worm in 1988
- using users distributed privileges or by exploiting system vulnerabilities
- widely used by hackers to create zombie PC's, subsequently used for further attacks, esp DoS
- major issue is lack of security of connected systems, esp PC's

Worm Operation

- worm phases like those of viruses:
  - dormant
  - propagation
    - search for other systems to spread
    - establish connection to target remote system
    - replicate self onto remote system
  - triggering
  - execution
Worm Attacks

- **Code Red**
  - exploited buffer overflow in MS IIS to penetrate & spread
  - probes random IPs for systems running IIS
  - 2nd wave infected 360000 servers in 14 hours

- **Code Red 2**
  - had backdoor installed to allow remote control

- **Nimda**
  - MS Outlook, IE, IIS
  - search strategy: island hopping
    - 50% same first two octets
    - 25% same first octet
    - 25% completely random IP

- **Sapphire Worm** *(Slammer, January 2003)* *(UDP-based)*
  - two orders magnitude faster than the Code Red worm
  - Buffer overflow in MS SQL Server

Spread of Sapphire Worm
Mobile Phone Worms

- First discovery was Cabir worm in 2004
  - Then Lasco and CommWarrior in 2005
- Communicate through Bluetooth wireless connections or MMS
- Target is the smartphone
  - can completely disable the phone, delete data on the phone, or force the device to send costly messages
  - CommWarrior replicates by means of Bluetooth to other phones, sends itself as an MMS file to contacts and as an auto reply to incoming text messages

Buffer Overflow

- Most common cause of Internet attacks
  - Over 50% of advisories published by CERT (computer security incident report team) are caused by various buffer overflows
- Buffer is a data storage area inside computer memory (stack or heap)
  - Intended to hold pre-defined amount of data
    - If more data is stuffed into it, it spills into adjacent memory
  - If executable code is supplied as “data”, victim’s machine may be fooled into executing it – we’ll see how
    - Code will self-propagate or give attacker control over machine
Stack Buffers

- Suppose Web server contains this function:

```c
void func(char *str) {
    char buf[126];
    strcpy(buf,str);
}
```

- When this function is invoked, a new frame with local variables is pushed onto the stack:

```
Allocate local buffer
(126 bytes reserved on stack)

Copy argument into local buffer
```

What If Buffer is Overstuffed?

- Memory pointed to by str is copied onto stack:

```c
void func(char *str) {
    char buf[126];
    strcpy(buf,str);
}
```

- If a string longer than 126 bytes is copied into buffer, it will overwrite adjacent stack locations:

```
strcpy does NOT check whether the string at *str contains fewer than 126 characters
```

```
This will be interpreted as return address!
```
Executing Attack Code

- Suppose buffer contains attacker-created string

  - When function exits, code in the buffer will be executed, giving attacker a shell
    - **Root shell** if the victim program is setuid root

Preventing Buffer Overflow

- Use safe programming languages, e.g., Java
  - What about legacy C code?
- Mark stack as non-executable
- Randomize stack location or encrypt return address on stack by XORing with random string
  - Attacker won’t know what address to use in his string
- Static analysis of source code to find overflows
- Run-time checking of array and buffer bounds
  - StackGuard, libsafe, many other tools
- Black-box testing with long strings
**DDoS Attacks**

Source Address Spoofing

- use forged source addresses
  - usually via the raw socket interface on operating systems
  - makes attacking systems harder to identify

- Reflection attack: attacker generates large volumes of packets that have the victim system as the destination address
TCP SYN Spoofing Attack

- Attacker sends SYN with spoofed src (seq = x)
- Server sends SYN-ACK (seq = y, ack = x+1)
- Spoofed Client resend SYN-ACK after timeouts
- Assume failed connection request
- SYN-ACK's to non-existent client discarded

Figure 7.3 TCP SYN Spoofing Attack

Reflection Attacks

- Attacker sends packets to a known service on the intermediary with a spoofed source address of the actual victim system
- When intermediary responds, the response is sent to the target
- “reflects” the attack off the intermediary (reflector)
Reflection Attacks

DoS Attack Defenses

four lines of defense against DDoS attacks

- these attacks cannot be prevented entirely
- high traffic volumes may be legitimate
  - high publicity about a specific site
  - activity on a very popular site
  - described as slashdotted, flash crowd, or flash event
Virus Countermeasures

- viral attacks exploit lack of integrity control on systems
- to defend need to add such controls
- typically by one or more of:
  - prevention - block virus infection mechanism
  - detection - of viruses in infected system
  - reaction - restoring system to clean state

Host-based Behavior-Blocking Software

- integrated with host O/S
- monitors program behavior in real-time
  - eg file access, disk format, executable mods, system settings changes, network access
- for possibly malicious actions
  - if detected can block, terminate, or seek ok
- but malicious code runs before detection
Generations of Anti-Virus Software

first generation: simple scanners
- requires a malware signature to identify the malware
- limited to the detection of known malware

d second generation: heuristic scanners
- uses heuristic rules to search for probable malware instances
- another approach is integrity checking

d third generation: activity traps
- memory-resident programs that identify malware by its actions rather than its structure in an infected program

fourth generation: full-featured protection
- packages consisting of a variety of anti-virus techniques used in conjunction
- include scanning and activity trap components and access control capability

Worm Countermeasures

- perimeter network activity and usage monitoring can form the basis of a worm defense

- worm defense approaches include:
  - signature-based worm scan filtering
  - filter-based worm containment
  - payload-classification-based worm containment
  - threshold random walk (TRW) scan detection
  - rate limiting
  - rate halting
Summary

- have considered:
  - various malicious programs
  - trapdoor, logic bomb, trojan horse, zombie
  - viruses
  - worms and DDoS attacks
  - countermeasures