Information Security

Van K Nguyen - HUT

Program (Software) Security

MALICIOUS PROGRAMS

- **Malware: software designed to infiltrate** or damage a computer system without the owner's informed consent
- **Spyware: software designed to intercept** or take partial control over the user's interaction with the computer, without the user's informed consent
  - secretly monitors the user's behavior
  - collect various types of personal information

Trapdoor/backdoor

- Secret entry point into a system
  - Special login into system (circumvents normal security procedures.)
- Presents a security risk
- Can be for good purpose as for Troubleshooting or maintenance
- Can be bad in wrong hand - Malicious intent

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 covert effect besides the expected
  - Appears normal/expected
  - Covert effect violates security policy
- User tricked into executing a trojan horse
  - Look normal but behind the scene, covert effect performed with user's authorization

Virus

- Self-replicating code
  - Like replicating Trojan horse
  - 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

- Problem: How to ensure virus "carrier" executed?
  - Place in boot sector of disk OR in executales which are likely to be used
- Boot Sector
  - Run on any boot
  - Propagate by altering boot disk creation
- Executable
  - Malicious code placed at beginning of legitimate program
  - Runs when application run
  - Application then runs normally

Virus Types

- Terminate but Stay Resident (TSRs)
  - Stays active in memory after application completes
  - 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

Macro Virus

- Infected “executable” isn’t machine code
  - Relies on something “executed” inside application data ➔ Macros
- Properties specific to these viruses
  - Architecture-independent
  - Application-dependent

Worms

- Runs independently
  - Does not require a host program
  - Propagates a fully working version of itself to other machines
- Carrie 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, $8.75 billion

Statistics: Computer Economics Inc., Carlsbad, California

Morris Worm

- Released November 1988
  - Program spread through Digital, Sun workstations
  - Exploited Unix security vulnerabilities
- 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

- 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

Morris Worm: Spread Mechanisms

- Sendmail
  - Exploit debug option in sendmail to allow shell access
- Fingerd
  - Exploit a buffer overflow in the gets 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
    - places 40-line C program in a temporary file
    - Compiles and executes this program
      - Opens socket to machine that sent script
      - Retrieves worm main program, compiles it and runs

Finger

- *An utility that allows users to obtain information about other users.*
  - the full name or login name of a user
    - whether or not a user is currently logged in,
  - telephone numbers, maybe, and other info
- *fingerd: a daemon, or background* process, to service remote requests using the finger protocol
- The bug exploited to break *fingerd: overrunning the buffer for* input
  - G*ets, a* standard C library, *takes input to a buffer without doing any bounds* checking

fingerd

- 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

Buffer Overflow

- Buffer overflow occurs when a program or process tries to store more data in a buffer than the buffer can hold
- Very dangerous because the extra information may:
  - Affect user's data
  - Affect user's code
  - Affect system's data
  - Affect system's code

Why Does Buffer Overflow Happen?

- No check on boundaries
  - Programming languages give user too much control
  - Programming languages have unsafe functions
  - Users do not write safe code
- C and C++, are more vulnerable because they provide no built-in protection against accessing or overwriting data in any part of memory

Why Buffer Overflow Matter

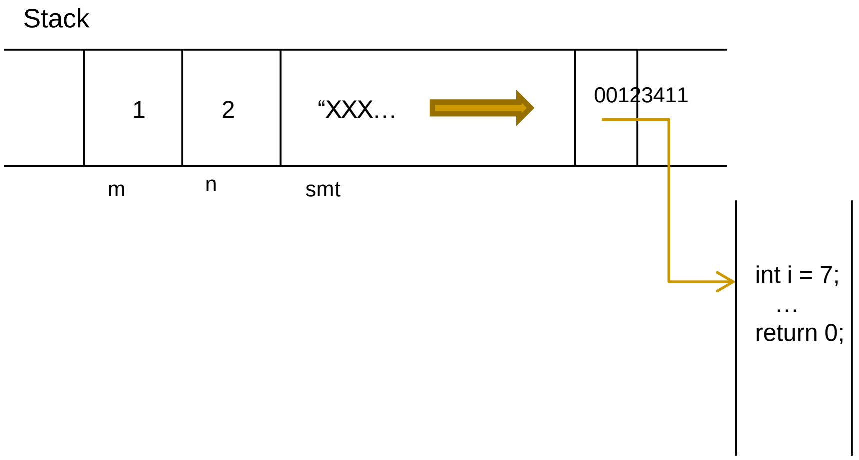
- Overwrites:
  - other buffers
  - variables
  - program flow data
- Results in:
  - erratic program behavior
  - a memory access exception
  - program termination
  - incorrect results
  - **breach of system security**

Basic Example

- A program has defined two data items which are adjacent in memory
  - an 8-byte-long string buffer, A, and a two-byte integer, B.
  - Initially, A contains nothing but zero bytes, and B contains the number 3
- Now, the program attempts to store the character string "excessive" in the A buffer, followed by a zero byte to mark the end of the string
  - By not checking the length of the string, it overwrites the value of B

Stack-based exploitation

- A malicious user may exploit stack-based buffer overflows to manipulate the program in one of several ways:
  - By overwriting a local variable that is near the buffer in memory on the stack to change the behaviour of the program which may benefit the attacker.
  - By overwriting the return address in a stack frame. Once the function returns, execution will resume at the return address as specified by the attacker, usually a user input filled buffer.
  - By overwriting a function pointer,[1] or exception handler, which is subsequently executed.


Stack
1
2
“XXX…
00123411
m
n
smt
int i = 7;
…
return 0;

WEB SECURITY

SQL injection

- **SQL injection** is a code injection technique that exploits a security vulnerability occurring in the database layer of an application.
- The vulnerability is present when user input is either incorrectly filtered for string literal escape characters embedded in SQL statements or user input is not strongly typed and thereby unexpectedly executed.
- It is an instance of a more general class of vulnerabilities that can occur whenever one programming or scripting language is embedded inside another.
- SQL injection attacks are also known as SQL insertion attacks.

Example

- Consider: SELECT * FROM users WHERE name = 'a' OR 't'='t';
- Set **username** as: a' or 't'='t
- Then get: SELECT * FROM users WHERE name = 'a' OR 't'='t';

Another example

- Use:

a';DROP TABLE users; SELECT * FROM data WHERE 't' = 't

So:

SELECT * FROM users WHERE name = 'a';DROP TABLE users; SELECT * FROM DATA WHERE 't' = 't';

Cross Site Scripting (XSS)

- Recall the basics
  - scripts embedded in web pages run in browsers
  - scripts can access cookies
    - get private information
  - and manipulate DOM objects
    - controls what users see
  - scripts controlled by the same-origin policy
- Why would XSS occur
  - Web applications often take user inputs and use them as part of webpage

Why XSS

- Name originated from the fact that a malicious web site could load another web site into another frame or window, then use Javascript to read/write data on the other web site
- The definition changed to mean the injection of HTML/Javascript into a web page

Example: Exploiting Social Network

- 1. Bad guy posts a message
- 2. When good guy reads the message, bad guy steals the cookie that contains information about authentication