Program Security and Vulnerabilities

Week 2
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.
- **When `func` returns**
  - The local variables are popped from the stack
  - The old value of the stack frame pointer (sfp) is recovered
  - The return address is retrieved
  - The stack frame is popped
  - Execution continues from return address (calling function)
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.

Stack grows this way:

```
buf          overflow          str
```

This will be interpreted as return address!
Suppose buffer contains attacker-created string
  For example, *str contains a string received from the network as input to some network service daemon

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

- Canary words
- Bounds checking
- Tagging
Canary words
- Known values placed between a buffer and control data on the stack
- When the buffer overflows, the first data to be corrupted will be the canary
- Failed verification of the canary data: overflow alert!
Bounds Checking

- Compiler based technique
- For each allocated memory block
  - Add run-time bounds information
  - Checks all pointers against bounds at run-time
Tagging

- Tag the type of each piece of data in memory
  - Used for type checking (e.g., prevent multiplication of int and string)
- Mark data buffers as non-executable
  - Prevent them from storing executable code
In this lecture

- Nonmalicious Program Errors
- Buffer Overflow
- SQL Injection Attack
- Incomplete Mediation
- Time-of-Check to Time-of-Use Errors
- Malicious Code
SQL in Web Pages

- SQL can be used to display data on a web page
- Web users can input their own search values
- Dynamically change SQL statements to provide the user with selected data:
  - Example (Server side code):
    - `txtUserId = getRequestString("UserId");`
    - `txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId + ";`  
    - `txtSQL` is a select statement
    - Fetch data from “Users” database for “txtUserId” to web the page
SQL Injection Attack

- Technique where malicious users can inject SQL commands into an SQL statement, via web page input.
- Injected SQL commands can alter SQL statement and compromise the security of a web application.
SQL Injection Attack Type 1

- 1=1 is always true
- txtUserId = getRequestString("UserId");
  txtSQL = "SELECT UserId, Name, Password FROM Users WHERE UserId = " + txtUserId + ";
- Malicious user can enter smart (but wrong) input as txtUserId

  UserId:
  105 or 1=1

- Server code:
  - SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1
  - Valid: will return all rows from the table “Users”
SQL Injection Attack Type 2

- SQL Injection Based on ""="" is Always True
- Server code:

```java
uName = getRequestString("UserName");
uPass = getRequestString("UserPass");

sql = "SELECT * FROM Users WHERE Name ="" + uName + "" AND Pass ="" + uPass + ""
```

User Name:

Password:
SQL Injection Attack Type 2 (cont’d)

- SQL Injection Based on ""="" is Always True
- Server code:
  
  ```
  uName = getRequestString("UserName");
  uPass = getRequestString("UserPass");
  sql = "SELECT * FROM Users WHERE Name ='' + uName + 
       '' AND Pass ='' + uPass + "";
  ```

- Attacker can insert " or "=" into the name and password box
- Server code becomes

  SELECT * FROM Users WHERE Name ='' or ''='' AND Pass ='' or ''=''
SQL Injection Attack Type 3

- SQL Injection Based on Batched SQL Statements
- Batched SQL statements: separated by semicolon
  - SELECT * FROM Users; DROP TABLE Suppliers
  - Return all rows in the Users table, then delete the table called Suppliers
SQL Injection Attack Type 3 (cont’d)

- SQL Injection Based on Batched SQL Statements
- Server code

```
    txtUserId = getRequestString("UserId");
    txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

    User id:
    105; DROP TABLE Suppliers
```

- Server code becomes

```
    SELECT * FROM Users WHERE UserId = 105; DROP TABLE Suppliers
```
In this lecture

- Nonmalicious Program Errors
- Buffer Overflow
- SQL Injection Attack
- Incomplete Mediation
- Time-of-Check to Time-of-Use Errors
- Malicious Code
Incomplete Mediation

- What if par2 is
  - 1800Jan01 (outside of range)
  - 2000Feb30 (non-existent)
  - 2048Min32 (undefined)
  - 1Aardvark2Many ?!?
Incomplete Mediation (cont’d)

- How to fix such errors?
  - Have client side code to verify input correctness
  - Restrict choices to only possible ones, e.g., drop-down menus ...
- Still vulnerable!
  - The results of the verification are accessible in the URL
  - The (malicious) user can access and modify fields
  - Only then send to the server
  - The server cannot tell if URL came directly from the user browser or from malicious user
In this lecture

- Nonmalicious Program Errors
- Buffer Overflow
- Incomplete Mediation
- Time-of-Check to Time-of-Use Errors (TOCTTOU)
- Malicious Code
TOCTTOU Errors

- Concurrency issue
  - Successive instructions may not execute serially
  - Other processes may be given control

- Access control
  - Studied later in the course
  - Only users with *rights* can access objects

- **TOCTTOU**: control is given to other process *between* access control check and access operation
TOCTTOU Example

int openfile(char *path) {
    struct stat s;
    if (stat(path, &s) < 0) {
        return -1;
    }
    if (!S_ISREG(s.st_mode)) {
        error("only allowed to regular files");
        return -1;
    }
    return open(path, O_RDONLY);
}

- Path to file
- Extract file meta-data
- Between check and open attacker can change path
  - Initial path is regular file
  - Later path is not
- Adversary by-passes security

No symlink, directory, special file
TOCTTOU: How an Attack Works

- openfile is being run within the kernel (at the OS)
- At the user space level, there is a program P
  - Controlled by adversary
  - Program P defines path variable
- Program P also launches two threads T1 and T2
  - T1 and T2 share the path variable
  - If T2 changes path, T1 also sees the change
- T1 runs openfile where path is set to a file
- T2 sets path to a directory
TOCTTOU Prevention

1. Ensure critical parameters are not exposed during pre-emption
   - `openfile` “owns” path

2. Ensure serial integrity
   - `openfile` is atomic
   - No pre-emption during its execution

3. Validate critical parameters
   - Compute checksum of `path` before pre-emption
   - Compare to checksum of `path` after ...
Combination of Flaws

- Can be used together
- **Example: Attacker can**
  - Use buffer overflow to disrupt code execution
  - Use TOCTTOU to add a new user to system
  - Use incomplete mediation to achieve privileged status
  - ...

In this lecture

- Nonmalicious Program Errors
- Buffer Overflow
- Incomplete Mediation
- Time-of-Check to Time-of-Use Errors (TOCTTOU)
- Malicious Code
What is Malicious Code

- Malicious Code (Malware) is a set of instructions that run on your computer and make your system do something that an attacker wants it to do.
- We previously studied program vulnerabilities
- Malicious code: exploits them
What Is It Good For?

- Steal personal information
- Delete files
- Click fraud
  - Pay-per-click online advertising
- Steal software serial numbers
- Use your computer as relay for other attacks
- Cryptocurrency mining
Types of Malicious Code

- Virus
  - Transient
  - Resident
- Trojan horse
- Logic/time bomb
- Worm
- Etc ...
What is a Virus?

- Program that can infect other programs by modifying them to include a, possibly evolved, version of itself

- Fred Cohen 1983
Some Virus Types

- Polymorphic: uses a polymorphic engine to mutate while keeping the original algorithm intact (packer)
- Methamorphic: Change after each infection
Trojan

- Class of malware that appears to perform a desirable function but in fact performs undisclosed malicious functions that allow unauthorized access to the victim computer [Source: Wikipedia]

- Trojans are not viruses since they do not replicate, but Trojan horse programs can be just as destructive.
Logic Bomb

- Type of Trojan Horse that executes when specific conditions occur:
  - Change in a file
  - Particular series of keystrokes
  - Specific time or date.
**Worm**

- *Reproducing programs that run independently and travel across network connections*
- **Virus vs. Worm - Reproduction Differences:**
  - A virus is dependent upon a host file or boot sector, and the transfer of files between machines to spread
  - A worm can run completely independently and spread of its own will through network connections
What is Spyware/Adware

- Collects small pieces of information about users without their knowledge.
- The presence of spyware is typically hidden from the user, and can be difficult to detect.
- Typically, secretly installed on the user's PC.
- Can be installed by the owner of a shared, corporate, or public computer on purpose in order to secretly monitor other users.

[Source: Wikipedia]
We discuss vulnerabilities and attacks
- Most vulnerabilities have been fixed
- Some attacks may still cause harm
- Do not try these at home or anyplace else

Purpose of this class
- Learn to prevent malicious attacks
- Use knowledge for good purposes
Law Enforcement

- David Smith
  - *Melissa virus*
  - 5 years in prison, $150K fine
- Ehud Tenenbaum (“The Analyzer”)
  - *Broke into US DoD computers*
  - 6 month service, suspended prison, $18K fine
- Dmitry Sklyarov
  - *Broke Adobe ebooks*
  - Prosecuted under DMCA
Summary

- We covered
  - Program flaws and Vulnerabilities
  - Buffer Overflow
  - Incomplete Mediation
  - TOCTTOU
  - Introduction to malicious code