COSC 458 - 647
Application Software Security

Final Review

Topics

- Overflow
    - Buffer overflow: Stack and heap-based buffer overflow
    - Integer overflow
- Assembly language
- SQL Injection
- Format string vulnerability
- Process and File racing
- XSS
- XSRF
- Firewall
- Most important: Code analysis + Situation awareness

General schema

- What the vulnerability is.
- How it happens
- How many types of that vulnerability
- Spot a simple vulnerability in a piece of code.
- Counter measure
    - Conventional methods
    - Your very own method
- **Study your midterm exam + quizzes**

Example questions

Integer Overflow Vulnerabilities Example

```
const   long MAX_LEN = 20K;
char    buf[MAX_LEN];
short   len = strlen(input);

if (len < MAX_LEN) {
 strcpy(buf, input);
}
```

Can a buffer overflow attack occur? If so, how long does input needs to be?

Another Example

The function is supposed to return false when (x + y) overflows unsigned short. Does the function do it correctly?

```
bool  isValidAddition(unsigned short x,
    unsigned short y) {
    if (x + y < x)
        return false;
    return true;
}
```

Buffer overflow

- A proposal for preventing stack buffer overflow attacks is based on making a backup copy of the return address when a function starts.
- The backup copy is written to a shadow stack located at some random location L on the heap.
- In the function epilog, just before the function is about to return, the backup copy of the return address is compared to the return address on the stack and if they differ the program exits. Otherwise, the return instruction is executed normally.
  - (a) Explain why this mechanism can make it harder to mount a stack buffer overflow attack.
  - (b) Give sample code that is vulnerable to a stack buffer overflow even if this mechanism is used.

Race condition

Consider the following code snippet:

```
if (!stat("./file.dat", buf)) return;   // Abort if file exists.
sleep(10);                              // Sleep for 10 seconds.
fp = fopen("./file.dat", "w" );
fprintf(fp, "Hello world" );
close(fp);
```

Suppose this code is running as a setuid root program.

(a) Give an example of how this code can lead to unexpected behavior that could cause a security problem. (try using symbolic links)

Suppose the sleep(10) is removed from the code above. Could the problem you identified in part (a) still occur? Explain. How would you fix the code to prevent the problem from part (a)?

Is the source code vulnerable to XSS?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
        <title> Order Form </title>
        <meta name = "GENERATOR" content = "Quanta Plus">
        <meta http - equiv = "Content-Type" content = "text/html; charset=iso-8859-1">
</head>
<body>
        <h3> Order Form </h3>
        <p> Thank you for placing an order with us today. </p>
        < ? php
        $sid = $_POST['sid'];
echo "<p>Your confirmation number for this order is $sid.</p>";
echo "<hr><p>To pay, we need to ....</p>";
? >
</body>
</html>
```