CSC429 – Computer Security

LECTURE 5
SOFTWARE SECURITY

Mohammed H. Almeshekah, PhD
meshekah@ksu.edu.sa

Software Security

Input Validation

Source of Input

- In a program:
  - Command line arguments
  - Environment variables
  - Function calls from other modules
  - Configuration files
  - Network packets

- In a Web Application:
  - Web form input
  - Scripting languages with string input

Weak Input Validation

- What are some things that the attacker may try to achieve?
  - Crash programs.
  - Execute arbitrary code:
    - setuid or setgid programs
  - Obtain sensitive information.

Validating Input

- How to validate input:
  - Do NOT allow bad inputs (blacklisting)
    - How can you be exhaustive?!
  - Only allow good input (whitelisting):
    - Can you enumerate all possible ALLOWED input?!

Command Line Input – Example

- What can go wrong in this program:

```
void main(int argc, char ** argv) {
 char buf[1024];
 sprintf(buf,”cat %s”,argv[1]);
 system (buf);
}
```

Input Validation – Environment Variables

- Users can set the environment variables to anything

- Examples:
    - LD_LIBRARY_PATH
    - PATH
    - IFS

A Simple Attack – 1

- Assume you have a setuid program that loads dynamic libraries.
- UNIX searches the environment variable LD_LIBRARY_PATH for libraries.
- A user can set LD_LIBRARY_PATH to /tmp/attack and places his own copy of the libraries here.
- Most modern C runtime libraries have fixed this by not using the LD_LIBRARY_PATH variable when the EUID is not the same as the UID or the EGID is not the same as the GID.

A Simple Attack – 2

- A setuid program has a system call:
  - system(ls);
- The user sets his PATH to be . (current directory) and places a program ls in this directory.
- The user can then execute arbitrary code as the setuid program.
- Solution:
  - Reset the PATH variable to be a standard form (i.e., "/bin:/usr/bin").

A Simple Attack – 3

- The user can reset the IFS variable
  - IFS is the characters that the system considers as white space, Or
  - add “s” to the IFS

- system(ls) becomes system(l)
  - Place a function l in the directory.

Software Security

String Formatting

String Formatting Example

- Take a look at this code example:

```
int  func(char *user)  {
     fprintf(stdout, user);
  }
```

- What if:
  - `User = “%s%s%s%s%s%s%s”;`
  - Most likely the program will crash.
  - May be will print memory context!

- Corrected:

```
int  func(char *user)  {
     fprintf( stdout, “%s”, user);
  }
```

Vulnerable Functions

- Any function using a format string.

- Printing:
  - printf, fprintf, sprintf, …
  - vprintf, vfprintf, vsprintf, …

- Logging:
  - syslog, err, etc.

Software Security

Integer Overflow

Integer Overflow

- Integer overflow:
  - an arithmetic operation attempts to create a numeric value that is larger than can be represented within the available storage space.

- Will the two functions below have the same output?

```
void func1(){
    short x = 30000;
    short y = 30000;
    printf(“%d\n”, x+y);
}
```

```
void func2(){
    short x = 30000;
    short y = 30000;
    short z = x + y;
    printf(“%d\n”, z);
}
```

C Data Types

short int	16bits	[-32,768; 32,767]
unsigned short int	16bits	[0; 65,535]
int	32bits	[-2,147,483,648; 2,147,483,647]
signed char	8bits	[-128; 127]
unsigned char	8 bits	[0; 255]

Why Does Integer Overflow Matter?

- When:
  - Allocating spaces using calculation.
  - Calculating indexes into arrays
  - Checking whether an overflow could occur

Integer Overflow – Example 1

```
int main(int argc, char *argv[]) {
     unsigned short s;
     int i;
     char buf[80];
     if (argc < 3){ return -1; }
     i = atoi(argv[1]);
     s = i;
     if(s >= 80)  { printf(“No you don't!\n");
                  return -1; }
     printf("s = %d\n", s);
     memcpy(buf, argv[2], i);
     buf[i] = '\0‘;
     printf("%s\n", buf); return 0;
}
```

Integer Overflow – Example 2

```
int ConcatBuffers(char *buf1, char *buf2,
    size_t  len1, size_t len2)
{
    char buf[0xFF];
    if ((len1 + len2) > 0xFF) return -1;
    memcpy(buf, buf1, len1);
    memcpy(buf+len1, buf2, len2);
    return 0;
}
```

Integer Overflow – Example 3

```
// 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;
}
```

Next Lecture

- Malicious Programs.

- Readings for next lecture:
  - Anderson’s Book – section 21.3