

CE4062/CZ4062

Computer Security

Tutorial 3: Memory Safety Vulnerabilities

Tianwei Zhang

Q1

Short answers

- (a) What is the root cause of format string vulnerability? What are the possible consequences?
- (b) How to prevent integer overflow vulnerabilities?
- (c) What is the scripting vulnerability?

Solution

Short answers

- (a) What is the root cause of format string vulnerability? What are the possible consequences?

The number of arguments does not match the number of escape sequences in the format string.

The possible consequences include:

- ▶ *Leak unauthorized information from the stack*
- ▶ *Crash the program*
- ▶ *Modify the data in the stack, or hijack the control flow.*

Solution

Short answers

- b) How to prevent integer overflow vulnerabilities?

Check the boundary of integers carefully

- ▶ *Check the sign of integers.*
- ▶ *For arithmetic operations, check the boundary of each operand.*
- ▶ *Try to convert the integers to the type with larger ranges (e.g., big number)*

Solution

Short answers

- c) What is the scripting vulnerability?

The scripting commands are built from predefined code fragments and user input. Then the script is passed to the system for execution

An attacker can hide additional commands in the user input. So the system will execute the malicious command without any awareness.

Q2

Consider the fragment of a C program below. The program has a vulnerability that would allow an attacker to cause the program to disclose the content of the variable “secret” at runtime. We assume that the attacker has no access to the exact implementation of the ‘get_secret()’ function so the attack has to work regardless of how the function ‘get_secret()’ is implemented.

- (a) Explain how the attack mentioned above works. You do not need to produce the exact input to the program that would trigger the attack. It is sufficient to explain the strategy of the attack. Explain why the attack works.
- (b) The vulnerability above can be fixed by modifying just one statement in the program without changing its functionality. Show which statement you should modify and how you would modify it to fix the vulnerability. Show the C code of the proposed solution

```
int main(int argc, char* argv[]) {  
    int uid1 = x12345;  
    int secret = get_secret();  
    int uid2 = x56789;  
    char str[256];  
  
    if (argc < 2)  
        return 1;  
  
    strncpy(str, argv[1], 255);  
    str[255] = '\0';  
    printf("Welcome");  
    printf(str);  
  
    return 0;  
}
```

Solution

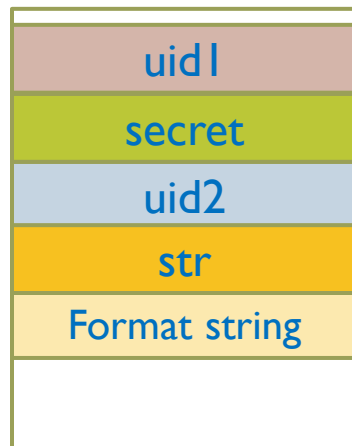
(a) Format string vulnerability

`argv[1]` can contain malicious format specifier when copied to `str`. After `printf(str)`, secret on the stack can be leaked.

```
printf("%08x %08x %08x %08x %08x")
```

(b) Fix the vulnerabilities: change the format of the `printf` function:

```
printf("%s\n", str);
```



```
int main(int argc, char* argv[]) {  
    int uid1 = x12345;  
    int secret = get_secret();  
    int uid2 = x56789;  
    char str[256];  
  
    if (argc < 2)  
        return 1;  
  
    strncpy(str, argv[1], 255);  
    str[255] = '\0';  
    printf("Welcome");  
    printf(str);  
  
    return 0;  
}
```

Q3

You are developing a web service, which accepts the email title **title** and body **body** from users, and forwards them to **fake-addr@ntu.edu.sg**. This is achieved by the following program.


Identify the security problems in this piece of program

```
void send_mail(char* body, char* title) {  
    FILE* mail_stdin;  
    char buf[512];  
    sprintf(buf, "mail -s \"Subject: %s\" fake-addr@ntu.edu.sg", title);  
  
    mail_stdin = popen(buf, "w");  
    fprintf(mail_stdin, body);  
    pclose(mail_stdin);  
}
```


Solution

I. Buffer Overflow

- ▶ When the size of title is larger than 471




```
void send_mail(char* body, char* title) {  
    FILE* mail_stdin;  
    char buf[512];  
    sprintf(buf, "mail -s \"Subject: %s\" fake-addr@ntu.edu.sg", title);  
  
    mail_stdin = popen(buf, "w");  
    fprintf(mail_stdin, body);  
    pclose(mail_stdin);  
}
```

Solution

2. Format string vulnerability

- ▶ The string **body** can contain malicious format specifiers to modify or view the stack



```
void send_mail(char* body, char* title) {  
    FILE* mail_stdin;  
    char buf[512];  
    sprintf(buf, "mail -s \"Subject: %s\" fake-addr@ntu.edu.sg", title);  
  
    mail_stdin = popen(buf, "w");  
    fprintf(mail_stdin, body);  
    pclose(mail_stdin);  
}
```

Solution


3. Scripting vulnerability

- ▶ **title** can contain malicious command:

title = empty" foo@bar.com; rm -rf /; echo "

buf = mail -s "Subject: empty" foo@bar.com; rm -rf /; echo "" fake-addr@ntu.edu.sg

- ▶ All the files on the server will be removed.



```
void send_mail(char* body, char* title) {  
    FILE* mail_stdin;  
    char buf[512];  
    sprintf(buf, "mail -s \"Subject: %s\" fake-addr@ntu.edu.sg", title);  
  
    mail_stdin = popen(buf, "w");  
    fprintf(mail_stdin, body);  
    pclose(mail_stdin);  
}
```

Q4

The following program implements a function in a network socket: `get_two_vars`. It receives two packets, and concatenate the data into a buffer. Use an example to show this program has integer overflow vulnerability. Note the first integer in the received buffer from 'recv' denotes the size of the buffer.

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0)
        return -1;
    if(recv(sock, buf2, sizeof(buf2), 0) < 0)
        return -1;

    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));
    size = size1 + size2;

    if(size > len)
        return -1;

    memcpy(out, buf1, size1);
    memcpy(out + size1, buf2, size2);

    return size;
}
```

Solution

It is possible that **size1**, **size2** are very big, but their sum is smaller than **len**.

- ▶ **size1 = 0x7fffffff**
- ▶ **size2 = 0x7fffffff**
- ▶ **(0x7fffffff + 0x7fffffff = 0xffffffff (-2)).**

```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    int size1, size2;
    int size;

    if(recv(sock, buf1, sizeof(buf1), 0) < 0)
        return -1;
    if(recv(sock, buf2, sizeof(buf2), 0) < 0)
        return -1;

    memcpy(&size1, buf1, sizeof(int));
    memcpy(&size2, buf2, sizeof(int));
    size = size1 + size2; ➡ integer overflow

    if(size > len)
        return -1;

    memcpy(out, buf1, size1);
    memcpy(out + size1, buf2, size2); ➡ buffer overflow

    return size;
}
```