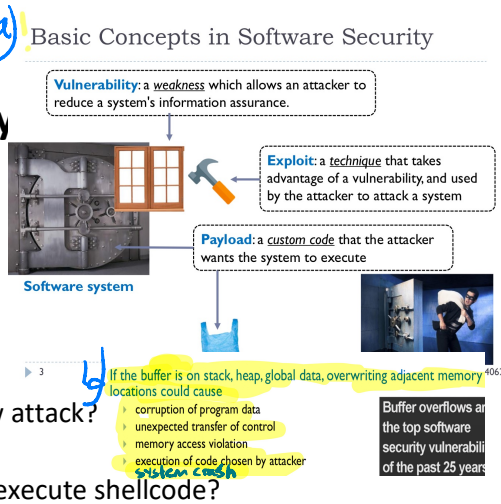
SC3010 TJ9

CE4062/CZ4062 Computer Security

Tutorial 2 – Buffer Overflow



1. Answer the following questions
   a. What do vulnerability, exploit, and payload refer to?

   b. What could be the potential consequences of a buffer overflow attack?

   c. What are the steps to utilize a buffer overflow vulnerability to execute shellcode?

1. Convert the shellcode from C to assembly code, and then binary
2. Store the binary code in a buffer, which is allocated on the victim stack
3. Use the buffer overflow vulnerability to overwrite the return address with the address of the binary shellcode.

2. In the following program in Figure Q2, the function **get_plural** returns the plural of any input string. Assume the attacker can send arbitrary input to the main function. Please identify the possible buffer overflow vulnerabilities in this problem.

```
void get_plural(char* single, char* plural) {
  char* buf;
  plural[0] = '\0';
  int n = strlen(single);
  if (n == 0) return;
  buf = malloc(n+3);
  char last = single[n-1];
  if (last == 's')
    strcpy(buf, single);
  else if (last = 'h')
    sprintf(buf, "%ses", single);
  else
    sprintf(buf, "%ss", single);
  strcpy(plural, buf);
  free(buf)
}
void main(int argc, char* argv[]) {
  char plural_form[256];
  get_plural(argv[1], plural_form);
  printf("The plural of %s is %s\n", argv[1], plural_form);
}
```

Figure Q2

3. The following program in Figure Q3 is designed to generate a random number. It takes a password as input, but always fails to generate a random number. Luckily, this program is vulnerable to a buffer overflow attack. Our goal is to leverage this advantage to generate a random number. Please figure out a password that can achieve this.

4. A developer writes the following program in Figure Q4 for user authentication for his system. However, this program is vulnerable to buffer overflow attacks. Please give some examples of malicious input that an attacker can use to bypass the authentication.

```
int check_authentication(char *pwd) {
  int auth_flag = 0;
  char Password[] = "qwertyu";
  char buffer[8];
  strcpy(buffer, pwd);
  if (strncmp(buffer, Password, 8) == 0)
    auth_flag = 1;
  return auth_flag;
}
int main(int argc, char* argv[]) {
  if(check_authentication(argv[1]))
    printf("Access Granted\n");
  else{
    printf("Access Denied\n");
  }
  return 0;
}
```

The attacker can leverage the `strcpy` to overflow the stack and bypass the authentication

- Overwrite the Password: **pwd** = "abcdefgh" + "abcdefgh"
- Overwrite the **auth_flag: pwd** = "xxxxxxxx" + "xxxxxxxx" + "abcd" -> the corresponding integer is 0x61626364

```
char CheckPassword() {
  char good = 'N';
  char Password[100];
  gets(Password);
  return good;
}
int main(int argc, char* argv[]) {
  printf("Enter your password:");
  if(CheckPassword() == 'Y')
    printf("Your random number is %d\n", rand()%100);
  else{
    printf("You don't have the permission to get a random number");
    exit(-1);
  }
  return 0;
}
```

Figure Q3

```
int check_authentication(char *pwd) {
  int auth_flag = 0;
  char Password[] = "qwertyu";
  char buffer[8];
  strcpy(buffer, pwd);
  if (strncmp(buffer, Password, 8) == 0)
    auth_flag = 1;
  return auth_flag;
}
int main(int argc, char* argv[]) {
  if(check_authentication(argv[1]))
    printf("Access Granted\n");
  else{
    printf("Access Denied\n");
  }
  return 0;
}
```

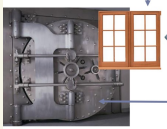
Figure Q4

TUT 2

1) a)

Basic Concepts in Software Security

Vulnerability: a *weakness* which allows an attacker to reduce a system's information assurance.



Exploit: a *technique* that takes advantage of a vulnerability, and used by the attacker to attack a system

Payload: a *custom code* that the attacker wants the system to execute

Software system

3 Buffer Overflow CE4062/CZ4062

b)

Buffer Overflow

Definition

- More input are placed into a buffer than the capacity allocated, overwriting other information

If the buffer is on stack, heap, global data, overwriting adjacent memory locations could cause

- corruption of program data
- unexpected transfer of control
- memory access violation
- execution of code chosen by attacker

- system crash

A very common attack mechanism

- 1988 Morris Worm
- 2001 Code Red
- 2003 Slammer
- 2004 Sasser
- …

Buffer overflows are the top software security vulnerability of the past 25 years

ON MAR 11, 13 • BY CHRIS BUBINAS • WITH 2 COMMENTS

In a report analyzing the entire CVE and NVD databases, which date back to 1988, Sourcefire senior research engineer Yves Younan found that vulnerabilities have generally decreased over the past couple of years before rising again in 2012. Younan

c)

snort answers

c) What are the steps to utilize a buffer overflow vulnerability to execute shellcode?

1. *Convert the shellcode from C to assembly code, and then binary*
2. *Store the binary code in a buffer, which is allocated on the victim stack*
3. *Use the buffer overflow vulnerability to overwrite the return address with the address of the binary shellcode.*

2)

```
void get_plural(char* single, char* plural) {
  char* buf;
  plural[0] = '\0';
  int n = strlen(single);
  if (n == 0) return;
  buf = malloc(n+3);
  char last = single[n-1];
  if (last == 's')
    strcpy(buf, single);
  else if (last = 'h')
    sprintf(buf, "%ses", single);
  else
    sprintf(buf, "%ss", single);
  strcpy(plural, buf);
  free(buf)
}
void main(int argc, char* argv[]) {
  char plural_form[256];
  get_plural(argv[1], plural_form);
  printf("The plural of %s is %s\n", argv[1], plural_form);
}
```

`sprintf`: Similar as `printf` except that write the string to the first argument.

buf size > single → no buffer overflow

if last size < plural size

3)

```
char CheckPassword() {
  char good = 'N';
  char Password[100];
  gets(Password);
  return good;
}
int main(int argc, char* argv[]) {
  printf("Enter your password:");
  if(CheckPassword() == 'Y')
    printf("Your random number is %d\n", rand()%100);
  else{
    printf("You don't have the permission to get a random number");
    exit(-1);
  }
  return 0;
}
```

101 chars end with 'Y'

info fgets more secure than gets

4)

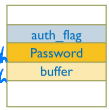
```
int check_authentication(char *pwd) {
  int auth_flag = 0;
  char Password[] = "qwertyu";
  char buffer[8];
  strcpy(buffer, pwd);
  if (strncmp(buffer, Password, 8) == 0)
    auth_flag = 1;
  return auth_flag;
}
int main(int argc, char* argv[]) {
  if(check_authentication(argv[1]))
    printf("Access Granted\n");
  else{
    printf("Access Denied\n");
  }
  return 0;
}
```

Password

The attacker can leverage the `strcpy` to overflow the stack and bypass the authentication

- Overwrite the Password: `pwd` = "abcdefgh" + "abcdefgh"
- Overwrite the `auth_flag`: `pwd` = "xxxxxxxx" + "xxxxxxxx" + "abcd" -> the corresponding integer is 0x61626364

strcpy(buffer, pwd);



abcdefgh abcdefgh

non zero value to overwrite auth_flag

TUT2

1) a Vulnerability = a weakness that give attacker opportunity to reduce system info assurance
exploit = a technique attacker use to take advantage of vulnerability
payload = malicious code attacker want system to execute

b · ~~System~~ corruption of program data
· shift of control
· memory access violation
· system crash
· execution of code by attacker

c) 1) Convert shell code to assembly code then binary
2) Store binary code in buffer which allocate on stack
3) Use buffer overflow vulnerability to overwrite return address to address of binary shellcode

2)

```c
void get_plural(char* single, char* plural) {
  char* buf;
  plural[0] = '\0';
  int n = strlen(single);
  if (n == 0) return;
  buf = malloc(n+3);
  char last = single[n-1];
  if (last == 's')
    strcpy(buf, single);
  else if (last = 'h')
    sprintf(buf, "%ses", single);
  else
    sprintf(buf, "%ss", single);
  strcpy(plural, buf);
  free(buf)
}
void main(int argc, char* argv[]) {
  char plural_form[256];
  get_plural(argv[1], plural_form);
  printf("The plural of %s is %s\n", argv[1], plural_form);
}
```

strcpy(buf, single) → buf size (n+3) > single (n) → no buffer overflow

strcpy(plural, buf): plural any size — if size of plural > n+3 ↓ buffer overflo

3)

3. The following program in Figure Q3 is designed to generate a random number. It takes a password as input, but always fails to generate a random number. Luckily, this program is vulnerable to a buffer overflow attack. Our goal is to leverage this advantage to generate a random number. Please figure out a password that can achieve this.

stack ↓ good ↑ str grow

password 1001 char ending with y

4)

auth_flag	4 byte	17 end with 1
(7 char + null) password	8	8 byte
buffer	8	8 byte

CE4062/CZ4062 Computer Security

Tutorial 3 – Software Security (2)

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.*

1. Answer the following questions
   a. What is the root cause of format string vulnerability? What are the possible consequences?

   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)*

   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.

2. Consider the fragment of a C program in Figure Q2. 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

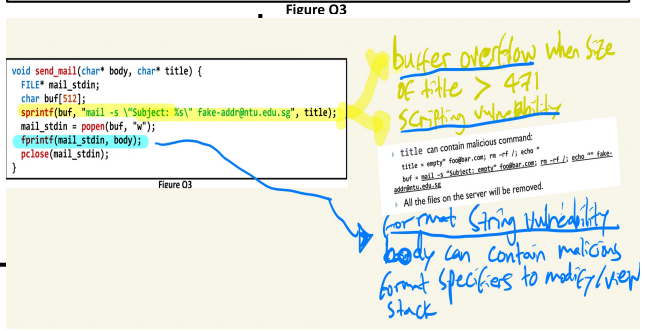
```c
int main(int argc, char* argv[]) {
  int uid1 = x12345;
  int secret = get_secret();
  int uid2 = x56789;
  char str[256];
  if (argc < 2)
    return 1;
  strncpy(str, argv[1], 255);
  str[255] = '\0';
  printf("Welcome");
  printf(str);

  return 0;
}
```

Figure Q2

```c
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);
}
```

Figure Q3



3. 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 in Figure Q3. Identify the security problems in this piece of program

4. The following program in Figure Q4 implements a function in a network socket: '`get_two_vars`'. It receives two packets, and concatenates 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 buff

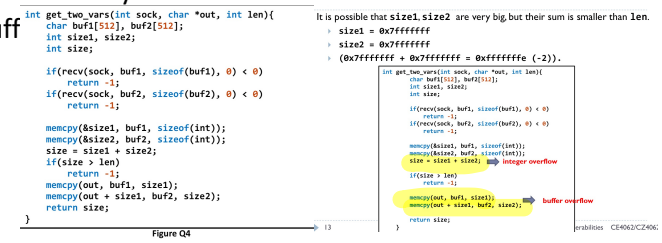


Figure Q4

```
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);
}
```

Figure Q3

```
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;
}
```

Figure Q4

TUT3

1) a)

A Main Source of Security Problems

Escape sequences are essentially instructions.

- Attack works by injecting escape sequences into format strings.

A vulnerable program

- Attacker controls both escape sequences and arguments in `user_input`.
- The number of arguments should match the number of escape sequences in the format string.
- Mismatch can cause vulnerabilities
- C compiler does not (is not able to) check the mismatch

```
#include <stdio.h>
#include <string.h>

int main(int argc, char* argv[]) {
  char user_input[100];
  scanf("%s", user_input);
  printf(user_input);
}
```

C Compiler doesn't check mismatch of number of arguments with number of escape sequences in format string leaving attack possible by injecting escape sequence

b)

Example 4: Signed and Unsigned Vulnerability

Another bad conversion between signed and unsigned integers

```
int func(char *data, int len) {
    char *buf = (char *)malloc(64);

    if (len > 64)
        return 0;

    memcpy(buf, data, len);
}
```

Vulnerability:

- `int` is signed, while `memcpy` can only accept unsigned parameter
- `memcpy` will convert `len` from signed integer to unsigned integer
- When `len=-1`, it will be converted to 0xFFFFFFFF, causing buffer overflow.

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)*

c)

Scripting Vulnerabilities

Scripting languages

- Construct commands (scripts) from predefined code fragments and user input at runtime
- Script is then passed to another software component where it is executed.
- It is viewed as a domain-specific language for a particular environment.
- It is referred to as very high-level programming languages
- Example:
  - Bash, PowerShell, Perl, PHP, Python, Tcl, Safe-Tcl, JavaScript

Vulnerabilities

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

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.

2)

```
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

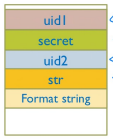
(a) Format string vulnerability

`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;
}
```

3)

```
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);
}
```

Figure Q3

buffer overflow when size of title > 471 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.

format String vulnerability body can contain malicious format specifiers to modify/view stack

4)

```
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;
}
```

Figure Q4

int overflow

buffer overflow

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

- `size1 = 0x7fffffff`
- `size2 = 0x7fffffff`
- `(0x7fffffff + 0x7fffffff = 0xfffffffe (-2)).`

TUT3

1) a. -printf doesn't check number of argument causing number of argument doesn't match with number of escape sequences
   - information leaking
   - program crash
   - modify data in the stack/hijacks control flow

   b. Check boundary of int carefully
   Design, operation, Convert to larger ranges

   c. Vulnerability that exist in high level programming where attacker hide additional commands in in user input for system to execute without awareness
   attacker add scripting commands on top of predefined code fragment then the script is parsed for execution

2) a) format string Vulnerability Can be copied inside buf through strncpy(str,argv[1],255); where argv[1] can be "%08x %08x" for secret or "%s%s%s" to trigger program stop

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

3) 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 in Figure Q3. 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);
}
```

Figure Q3

buffer overflow + scripting vuln

format string vuln

4) The following program in Figure Q4 implements a function in a network socket: 'get_two_vars'. It receives two packets, and concatenates the data into a buffer. Use an example to show this program has integer overflow vulnerability. Note the first inteeer in the received buffer from 'recv' denotes the size of the buff

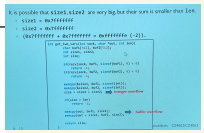
```
int get_two_vars(int sock, char *out, int len){
    char buf1[512], buf2[512];
    int size1, size2;
    int size;

    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;
}
```

→ integer
→ buffer

Figure Q4



CE4062/CZ4062 Computer Security

Tutorial 4 – Software Security (3)

1. Answer the following questions
   a. Why are non-executable stack and heap not enough to defeat buffer overflow attacks?

   b. Distinguish three types of fuzzing techniques?

   c. What is a code reuse attack?

2. For string copy operation, strncpy is regarded as "safer" than strcpy. However, improper use of strncpy can also incur vulnerabilities. Please describe the problem in the following program, and what consequences it will cause.

```c
#include <stdio.h>
#include <string.h>
int main() {
  char src[] = "geeksforgeeks";

  char dest[8];
  strncpy(dest, src, 8);
  int len = strlen(dest);

  printf("Copied string: %s\n", dest);
  printf("Length of destination string: %d\n", len);

  return 0;
}
```

Figure Q2

3. StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.
   a. In some implementations, the canary value is a 64-bit integer that is randomly generated each time the program runs. Explain why this prevents the basic form of buffer-overflow attacks
   b. What is a security drawback to choosing the canary value at compile time instead of at run time?
   c. If the value must be fixed, what will be a good choice?
   d. List an attack which can defeat the StackGuard.

4. One possible solution to defeat buffer overflow attacks is to set the stack memory as Non-executable (NX). This is usually achieved by the OS and the paging hardware. However, imagine that a machine does not support the non-executable feature, then we can implement this functionality at the software level. The compiler can allocate each stack frame in a separate page, and associate a software-manipulated NX bit at the bottom of this page, controlling if this page (stack frame) is non-executable. The structure of a stack frame is shown in the Figure below.

Since the NX bit is at the bottom of the memory page, the buffers inside this stack frame is not able to overwrite this bit to make it executable. Describe a buffer overflow attack that can still overwrite NX bits.

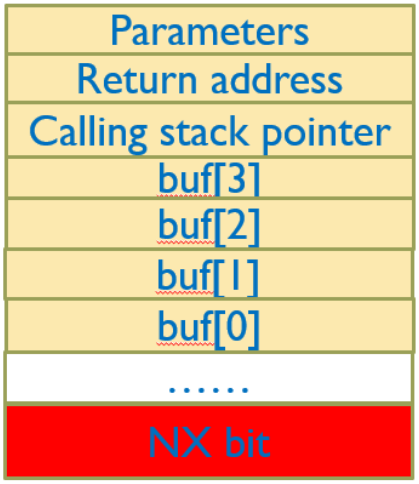


Figure Q4

TUT 4

1)a) Non-executable stack can still be access it depend on the s.tc of stack

b)

Fuzzing

Key Idea:

- Find software bugs in a program by feeding it random, corrupted, or unexpected data
- Random inputs will explore a large part of the state space
- Some unintended states are observable as crashes
- Any crash is a bug, but only some bugs are exploitable

A lot of software testing tools based on fuzzing

- **AFL:** https://github.com/google/AFL
- **FOT:** https://sites.google.com/view/fot-the-fuzzer
- **Peach:** https://wiki.mozilla.org/Security/Fuzzing/Peach
- **Springfield:** https://blogs.microsoft.com/ai/microsoft-previews-project-springfield-cloud-based-bug-detector/

Mutation-based Fuzzing

Steps:

- Collect a corpus of inputs that explores as many states as possible
- Perturb inputs randomly, possibly guided by heuristics
  - Modify: bit flips, integer increments
  - Substitute: small integers, large integers, negative integers
- Run the program on the inputs and check for crashes

```
numwrites = random.randrange(math.ceil((float(len(buf)) / fuzzfactor))) + 1
for j in range(numwrites):
    rn = random.randrange(len(buf))
    buf[rn] = "%c"%(byte)
```

Pros

- Simple to set up;
- Use off-the-shelf software for many programs.

Cons

- Results depend on the quality of the initial corpus;
- coverage may be shallow.

Generation-based Fuzzing

Steps:

- Convert a specification of the input format (RFC, etc.) into a generative procedure
- Generate test cases according to the procedure and introduce random perturbations
- Run the program on the inputs and check for crashes

Pros

- Get higher coverage by leveraging knowledge of the input format;

Cons

- Requires lots of effort to set up;
- Domain-specific;

Coverage-guided Fuzzing

Steps:

- Using traditional fuzzing strategies to create new test cases by mutating the input
- Test the program and measure the code coverage.
- Using the code coverage as a feedback to craft input for uncovered code

Pros

- Good at finding new program states;
- Combine well with other fuzzing techniques;

Cons

- Cannot find all types of bugs

c)

Code Reuse attack

- Reuse the code in the program itself without injecting the code

attacker can compromise the control flow to jump to existing library for attacks

2)

```c
#include <stdio.h>
#include <string.h>
int main() {
  char src[] = "geeksforgeeks";

  char dest[8];
  strncpy(dest, src, 8);
  int len = strlen(dest);

  printf("Copied string: %s\n", dest);
  printf("Length of destination string: %d\n", len);

  return 0;
}
```

3)

TUT 4

1) a) return address can be over written to any code ald loaded

b) Fuzzing tech: testing system for comp bug by injecting unexpected input that explore states, unexpected state will be bug, but not all bug are exploitable

- Mutation based fuzzing
  random input generator
  Pro - simple to implement
  - common to use
  Con - result depend on seed
  - shallow coverage

- Generation based fuzzing
  random input follow certain input format
  pro - higher coverage
  con - domain specific
  harder to set up

- Coverage based fuzzing
  use ~~combine~~ traditional fuzzing techniques
  test program n measure coverage
  use feedback to construct next input for uncovered code
  pro - combine well with other fuzzing
  - good @ finding new state
  con - cannot find all bug

c) attack where attacker doesn't inject malicious code but keep reusing the code in program which will replace return address with dangerous address that will be execute when funct return

2) size of src is bigger than strncpy does not automatically set last element to '\0', this could cause confusion of data

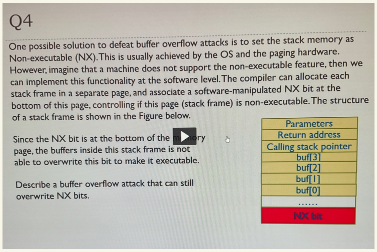
3) a) the system will add guard value in the stack during compile, execute the code and if the guard value = secret canary value proceed else the progam will exit

b) use brute force

c) using terminator canary, attacker cannot copy beyond terminator

d) - denian an attack
- overwrite function pointer

4)



Q4

One possible solution to defeat buffer overflow attacks is to set the stack memory as Non-executable (NX). This is usually achieved by the OS and the paging hardware. However, imagine that a machine does not support the non-executable feature, then we can implement this functionality at the software level. The compiler can allocate each stack frame in a separate page, and associate a software-manipulated NX bit at the bottom of this page, controlling if this page (stack frame) is non-executable. The structure of a stack frame is shown in the Figure below.

Since the NX bit is at the bottom of the memory page, the buffers inside this stack frame is not able to overwrite this bit to make it executable.

Describe a buffer overflow attack that can still overwrite NX bits.

CE4062/CZ4062 Computer Security

Tutorial 5 – Operating System Security (1)

1. Answer the following questions
    a. Give an example of how a rootkit can compromise the system after obtaining the root privilege.

    b. Briefly describe three stages employed by the OS.

    c. What is the controlled invocation? What is its potential danger?

2. Consider a computer system with three users: Alice, Bob, and Cindy. Alice owns the file *alicerc*, and Bob and Cindy can read it. Cindy can read and write the file *bobrc*, which Bob owns, but Alice can only read it. Only Cindy can read and write the file *cindyrc*, which she owns. Assume that the owner of each of these files can execute it.
    a. Create the corresponding access control matrix.

    b. Cindy gives Alice permission to read *cindyrc*, and Bob removes Alice's ability to read *bobrc*. Show the new access control matrix

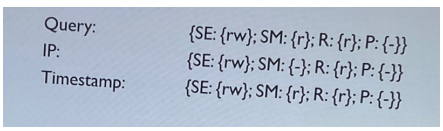
    c. Show the capabilities associated with Alice.

3. Let's consider the scenario in Q2 again. Assume this is the Unix system. The users Bob and Cindy are in the same group, while Alice is in a different group.
    a. For the original access control matrix in Q2(a), please write the permission for the files *alicerc*, *bobrc* and *cindyrc*.

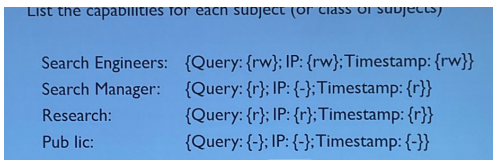
    b. To adjust the permissions in Q2(b), please write the corresponding commands for *cindyc* and *bobrc*, respectively.

4. A group of researchers is working on analyzing web search results from a major Internet search provider. At the Internet company, a group of search engineers collects and updates databases of: search queries, IP (internet protocol) addresses where the queries came from and timestamps for the queries made by online users. A search manager is in charge of the group of engineers, and can read the query and timestamps database, but not the IP address database -- due to privacy concerns. The researchers are able to access the databases with read-only privileges. The general public should not have access to the database for privacy reasons .
    a. Complete the access control matrix by filling in the access permissions for the different objects shown. Each entry can be either read, write, read/write, or '-' (for no access).

    b. List the ACLs for each object (or class of objects)



    c. List the capabilities for each subject (or class of subjects).



TUT3

1) a) it can highjack System Call table by replacing old syscall_open used to display all running process. Rootkit redirect it to new address ~~and~~ call new syscall_open

b) - identification and authentication → check user identity (fingerprint) to check with stored identity to see if it match
- Access Control → check access is allowed for the subject through access controll table
- logging or auditing → record all protection oriented activities, catching things that shouldn't to understand what happen in

c) by enabling SUID, user executing privileged program can inherit permission from program owner

2) a)

	alicerc	bobrc	cindyrc
alice	e	R	
bob	R	e	
cindy	R	Rw	Rwne

c)

	alicerc	bobrc	cindyrc
alice	e		R

b)

	alicerc	bobrc	cindyrc
alice	e		R
bob	R	e	
cindy	R	Rw	Rwne

3) a)

	alicerc	bobrc	cindyrc
alice	e	R	
bob	R	e	
cindy	R	Rw	Rwne

b)

a

alicerc - --x --- r--
bobrc - --x rw- r--
cindyrc - rwx --- ---

b

- --x --- r--
- --x rw- r--
- rwx

4)

	SP	IP	T
researcher	r	r	r
search engineers	rw	rw	rw
public	-	/	—
search manager	r	—	r

CE4062/CZ4062 Computer Security

Tutorial 6 – Operating System Security (2)

1. Answer the following questions
    a. Describe what is the confinement strategy, and why it can be used for malware testing and analysis.

    b. List the security functionalities offered by the TPM.

    c. Describe the lifecycle of an SGX enclave application.

2. Early Intel processors (e.g., the 8086) did not provide hardware support for dual-mode operation (i.e., support for a separate user mode and kernel mode). If a system is implemented on such processors to support the multi-programming scenario, describe one confidentiality, integrity and availability threat respectively in this system, due to the lack of hardware support.

3. Translation Lookaside Buffer (TLB) is a small hardware component that caches the recent translations of virtual memory to physical memory. It can help accelerate the memory access of programs. When a program wants to access the data with the specific virtual memory address, the system will check if there is an entry of this address in the TLB, and if the program has the access permission to this address. If both checks pass, then the corresponding physical address will be generated, and the access is allowed. Otherwise, a hardware interrupt will be triggered. Figure Q3 shows the mechanism of the TLB. Note that the TLB can be updated only when the CPU is in the kernel mode.
    a. The TLB can be regarded as one type of hardware-based reference monitor. Please list the requirements for a reference monitor.

    b. Analyze if the TLB can satisfy these three requirements.

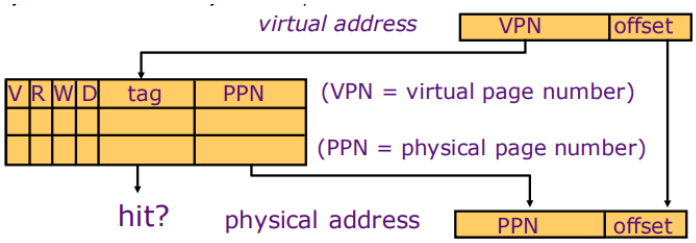


Figure Q3

4. Trusted Computing Base (TCB) is an important concept in computer security. It refers to the set of components (e.g., hardware, software, firmware, etc.) that must be trusted in order to guarantee the security of the entire system. Well protection of the TCB can defend the system against the threats from outside the TCB.
    a. As a system designer, do we expect to have a larger TCB or smaller TCB? Why?

b. Consider a conventional cloud computing scenario, where you launch a virtual machine in a cloud service provider (e.g., Amazon). Figure Q4 shows the system architecture of a cloud server running your virtual machines together with other users' virtual machines. Please specify which components are included in the TCB, and what entities and components are considered untrusted.

c. Assume the cloud provider adopts the TEE solution – AMD SEV processors to protect the users' virtual machines. In this case, specify which components are included in the TCB. Discuss how SEV processors can protect the virtual machines from untrusted components outside of the TCB.

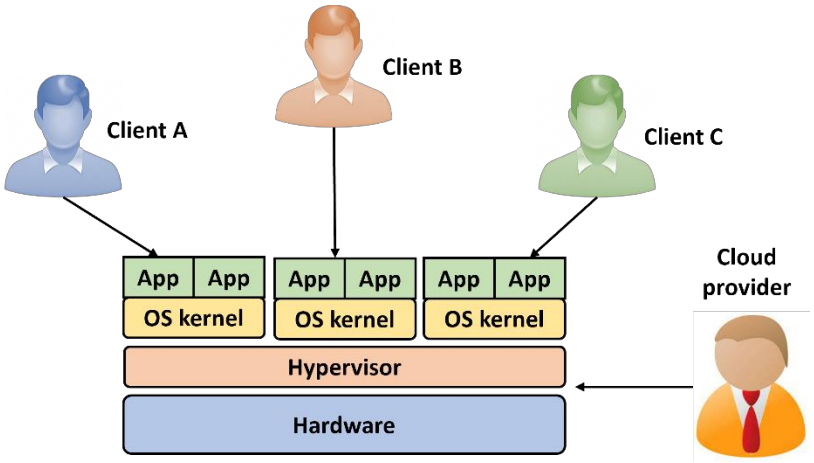
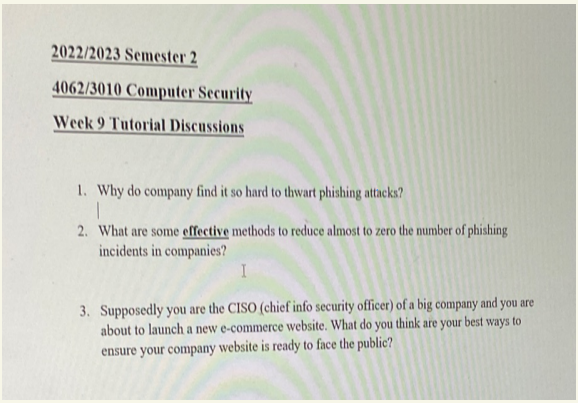


Figure Q4

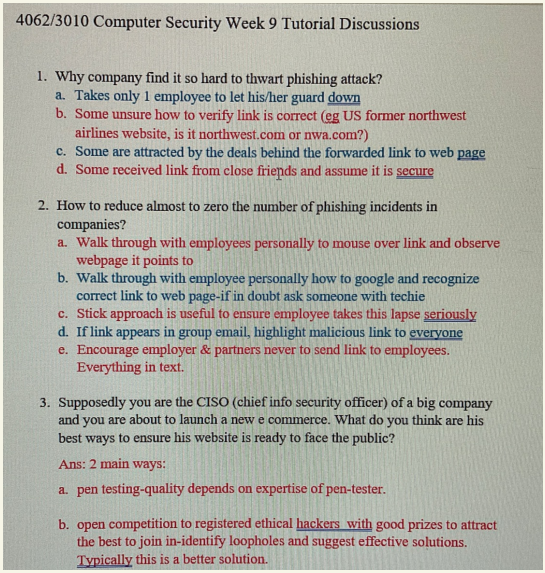


2022/2023 Semester 2

4062/3010 Computer Security

Week 9 Tutorial Discussions

1. Why do company find it so hard to thwart phishing attacks?
2. What are some **effective** methods to reduce almost to zero the number of phishing incidents in companies?
3. Supposedly you are the CISO (chief info security officer) of a big company and you are about to launch a new e-commerce website. What do you think are your best ways to ensure your company website is ready to face the public?



4062/3010 Computer Security Week 9 Tutorial Discussions

1. Why company find it so hard to thwart phishing attack?
   a. Takes only 1 employee to let his/her guard down
   b. Some unsure how to verify link is correct (eg US former northwest airlines website, is it northwest.com or nwa.com?)
   c. Some are attracted by the deals behind the forwarded link to web page
   d. Some received link from close friends and assume it is secure
2. How to reduce almost to zero the number of phishing incidents in companies?
   a. Walk through with employees personally to mouse over link and observe webpage it points to
   b. Walk through with employee personally how to google and recognize correct link to web page-if in doubt ask someone with techie
   c. Stick approach is useful to ensure employee takes this lapse seriously
   d. If link appears in group email, highlight malicious link to everyone
   e. Encourage employer & partners never to send link to employees. Everything in text.
3. Supposedly you are the CISO (chief info security officer) of a big company and you are about to launch a new e commerce. What do you think are his best ways to ensure his website is ready to face the public?

   Ans: 2 main ways:

   a. pen testing-quality depends on expertise of pen-tester.

   b. open competition to registered ethical hackers with good prizes to attract the best to join in-identify loopholes and suggest effective solutions. Typically this is a better solution.

1) few factors that make it hard to thwart phising attk
- takes only 1 employee to let his/her guard down
- some unsure how to verify link is correct
- some attracted by deals behind forwarded link to web page
- some link from acquintance n assume it is safe

2) walk through with employees personally how to google and recognize correct link
- Stick approach is useful to ensure employee takes this lapse seriously
- highlight malicious link to other
- encourage employer & partner never to read link n in text

3) - pen testing quality depends on expertise of pen-tester
- Open comp to registered ethical hackers with good prizes to attract the best to join in-identify loopholes n suggest effective solution.

AY2022/2023 Semester 2- CZ3010 Questions

WK-10

1. In the wise man story, at the 20th square, he already had a bag of rice (abt 2^20 grains). How many bags of rice did he earn as his reward?

2. (a) Show that the complexity of monoalphabetic substitution (26!) is roughly 2^88.
   (b)How do you make a monoalphabetic substitution cipher harder to break?

3. A 3GHz PC can crack approximately 2^34 work in 1 day. Calculate the time taken (in years) to crack monoalphabetic substitution by brute force using 1 PC. What about cracking time of 1 billion PCs of same specs?

4. Why do long keywords, shorter message implies stronger Vigenere cipher?

5. General Douglas sends the message ATTACK to his soldiers using a one time pad {GZAMCQ} through email. Suppose attacker sniffed out such a ciphertext. Explain why he/she is not able to decrypt this cipher with 100% certainty, assuming attacker knows it's from a one-time pad.

6. (a) Why must pad be random?
   (b) Why must pad be not reused again? (asking for a quantitative reasoning)

7. Johnny English want to make his OTP encryption even harder for attackers. He decides to encrypt twice using 2 different OTPs. Is his method more secure than the usual one?

8. NSA has intercepted a Vignere ciphertext: {Y W W L F F D A Q B H L W B G V G R G S N Z D V U}, and Ethan Hunt has obtained the OTP- CODE. **Decrypt this ciphertext.**

plaintext: W I T H D R A W O N E H U N D R E D D O L L A R S
keyword: C O D E C O D E C O D E C O D E C O D E C O D E C
ciphertext: Y W W L F F D A Q B H L W B G V G R G S N Z D V U

9. Use ONLY your mind to create a sequence of 64 random bits, in blocks of 8 bits. Then use any RNG (from OS RNG etc) to generate such a sequence. Then you compare the difference. How many 00000 do you expect to find?

10. In early IPOD days, some listeners complained hearing the same song within 2 hours although they have 400 songs on their ipod. Assuming 4 min songs on average. Question: Is the IPOD shuffling random?

11. Suppose 2 random number generates a 8-bit string 11010110 in one portion of the string & another generates 00000000 at another portion. Is the first more random than 00000000?

Display S

1) 8×8 = 64
(4-20) = 44
$2^{20} \cdot 2^{44}$
3 bleg
2^{44} bags

2a) 26! = 2^Y
$\log(26!) = Y \log_2 X$
$Y = \frac{X}{\log_2} = 88.38$
2b)

AY2022/2023 Semester 2- CZ3010 (Solutions)

1. In the wise man story, at the 20th square, he already had a bag of rice (abt 2^20 grains). How many bags of rice did he earn as his reward?
   Ans:
   Collect all grains up to 20th square & put it into 1 bag on the 20th square.
   Total number of bags (2^20 grains-abt 1 million grains!)
   1 + 2^1 + 2^2 + ... + 2^44 + 2^45, so roughly 2^46 bags of rice!!! Kingdom bankrupt!

2. (a) Show that the complexity of monoalphabetic substitution (26!) is roughly 2^88.
   (b)How do you make a monoalphabetic substitution cipher harder to break?
   Ans:
   First compute log(26!) = log1 + log2 + ... + log26 (eg use Excel to add)
   Call this number X.
   Write 26! = 2^Y.
   Hence log(26!) = Y log2 = X
   Hence Y = X/log2 = 88.38

3. A 3GHz PC can crack approximately 2^34 work in 1 day. Calculate the time taken (in years) to crack monoalphabetic substitution by brute force using 1 PC. What about cracking time of 1 billion PCs of same specs?
   Ans:
   Using 1 PC, Cracking time for 2^88 is 2^88/2^34 days = 2^54 days
   = 2^54/365 years = 5x 10^13 years!
   Using 1 billion PCs, takes, 5x 10^13/10^9 = 50,000 years!
   So even NSA and big intel cant crack 88-bit!

4. Why do long keywords, shorter message implies stronger Vigenere cipher?
   Ans:
   Long keywords lead to lesser observable pattern in ciphertext.
   Also more frequency tables needed (length n keyword means n diff freq subst tables)

   Shorter msg means stats analysis not accurate

5. General Douglas sends the message ATTACK to his soldiers using a one time pad {GZAMCQ} through email. Suppose attacker sniffed out such a ciphertext. Explain why he/she is not able to decrypt this cipher with 100% certainty, assuming attacker knows it's from a one-time pad.
   Ans:
   Adversary will never be sure if plaintext msg is other meaningful 6 letter word such as DEFEND, STATIC etc

6. (a) Why must pad be random?
   Ans: real random pad will ensure perfect secrecy, as each bit has equal chance of 0.5 to appear. Totally unpredictable.
   (b) Why must pad be not reused again? (asking for a quantitative reasoning)
   Ans:
   If pad K is reused.
   C1 = P1 XOR K
   C2 = P2 XOR K (same K)
   Then C1 XOR C2 = P1 XOR P2 XOR K XOR K
   So C1 XOR C2 = P1 XOR P2, K totally disappear!

7. Johnny English want to make his OTP encryption even harder for attackers. He decides to encrypt twice using 2 different OTPs. Is his method more secure than the usual one?
   Ans:
   P XOR K1 XOR K2
   = P XOR (K1 XOR K2)
   (K1 XOR K2) is just another random string of equal length as K1 and K2.
   This string is just another same length random string. So no extra security if we encrypt with 2 different one time pad.

8. NSA has intercepted a Vignere ciphertext: {Y W W L F F D A Q B H L W B G V G R G S N Z D V U}, and Ethan Hunt has obtained the OTP- CODE. **Decrypt this ciphertext.**

plaintext: W I T H D R A W O N E H U N D R E D D O L L A R S
keyword: C O D E C O D E C O D E C O D E C O D E C O D E C
ciphertext: Y W W L F F D A Q B H L W B G V G R G S N Z D V U

9. Use ONLY your mind to create a sequence of 64 random bits, in blocks of 8 bits. Then use any RNG (from OS RNG etc) to generate such a sequence. Then you compare the difference. How many 00000 do you expect to find?
   Ans:
   P(00000) single run) = (1/2)^5
   In a run of 64 entries,
   This string can begin on1st, and last possible occurrence is 60th position.
   So 60 possibilities.
   So expected number of 5 0s = 60* (1/2)^5 = almost 2

2022/2023 3010 Computer Security Semester 2

Week 11 Tutorial

Authentication/Password/Hash/Block & Stream cipher/RNG

1. Is it always a bad idea to write down your password? Is there a way to do it more safely?

2. (a) Verify the computational complexity of these 3 entries: 10 character lower case alphanumeric, 9 character alphanumeric, 8 character printable keyboard characters in slide 16 of the Password Slides. What do you notice?

   (b) The fastest Software Password crackers has cracking speed of up to 16 million passwords per sec (take it as 2^24) on a 3GHz PC. Estimate the time taken to compute cracking time of
   (i) 10 numeric characters passwords
   (ii) 8 character lower case alphanumeric passwords
   (iii) 8 character alphanumeric passwords
   (iv) 8 character all 95 printable characters passwords

$36^{10} = 51.69$
$62^9 = 2^k \quad k = 53.19$
$95^8 = 2^k \quad k = 52.558$

$10^{10} = 2^k \quad k = 51.69$
$36^8 = 2^k \quad k =$
$62^8 = 2^k \quad k = 47.63$
$95^8 = 2^k$

3. Suppose that a certain email system uses hash of time when document is encrypted & emailed together. Example: time 20220203073000 means (3 February 2022 7:30:00)

- Alice encryption software works as follow:
- It uses 256-bit AES-encryption.
- Key used is 256-bit key, namely SHA256(Date & Time of email).
- When Alice use this encrypted email document and want to send to Bob, system looks at the time, then perform SHA256(Date & Time of email) and use this as 256-bit key for AES.
- Bob downloaded document into his pc. One day his pc is hacked and this document is in hands of hacker. Assume that hacker knows encryption algorithm and hash function used but not the date and time when document is sent.
- (a) Explore if hacker has a chance to read this document.
- (b) What are the key lessons we can learn from this implementation of strong algorithm AES?

4. Why do we want to use slow hash for password hashing?

5. 5.1 Define a toy hash function $h : (Z_2)^7 \to (Z_2)^4$ by the rule $h(x) = xA$ whe all operations are modulo 2 and

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

Find all preimages of $(0, 1, 0, 1)$.

6. Why do we need 256-bit hash length instead of 128-bit length to pair with AES-128?

7. Explain why a long codeword such as hippopotamus is not secure for use in Vigenere cipher

8. Keystream of {0,1} generated by pseudo-random number generators will be periodic. Secure keystream must necessary have long period. Show that even with extremely long period, some keystreams may not be suitable for use to generate encryption keys.

9. Explain why for CTR mode in block cipher encryption, your counter values must all be distinct to be secure.

10. If the hash reflected on the web page of download coincides with your computed hash, does it always mean the files have not been tampered with, assuming the hash used is a real secured hash?

Authentication/Password/Crytpo Tutorial

1. Is it always a bad idea to write down your password? Is there a way to do it more safely?

Ans:

Not always a bad idea.

Write down all your difficult pswds on a paper.

Then lock it in a safe or locked office drawer.

Hackers will not be able to access your office, and open your safe or locked drawer.

But if he can install keyloggers into your pc thru your carelessness, then your secure passwords will be captured by it and render your long secure pswds USELESS.

2. (a) Verify the computational complexity of these 3 entries: 10 character lower case alphanumeric, 9 character alphanumeric, 8 character printable keyboard characters in slide 16 of the Password Slides. What do you notice?

Ans:

$\rightarrow c$ $\downarrow n$	26 (lowercase)	36 (lowercase alphanumeric)	62 (mixed case alphanumeric)	95 (keyboard characters)
5	23.5	25.9	29.8	32.9
6	28.2	31.0	35.7	39.4
7	32.9	36.2	41.7	46.0
8	37.6	41.4	47.6	52.6
9	42.3	46.5	53.6	59.1
10	47.0	51.7	59.5	65.7

I will just work on 9 char alphanumeric. U can try the rest, imitating me.
# of possible 9 char alphanumeric
= (62)^9 = 2^k, solve k.
Take log on both sides,
9 lg62 = k lg2,
So k = 9lg62/lg2 = 53.6, as expected (see table above)

(b) The fastest Software Password crackers has cracking speed of up to 16 million passwords per sec (take it as 2^24) on a 3GHz PC. Estimate the time taken to compute cracking time of
(i) 10 numeric characters passwords
(ii) 8 character lower case alphanumeric passwords
(iii) 8 character alphanumeric passwords
(iv) 8 character all 95 printable characters passwords

Ans:
doing computations for 8 char alphanumeric = 2^47.6
U can try the rest, imitating me.
Since cracking speed of pswd crackers is 2^24 pswd per sec,
Time needed to crack 8 char alphanumeric
= 2^47.6/2^24 sec
= 2^23.6 sec
= (2^23.6)/60 mins
= (2^23.6)/3600 hours
= 3532 hours!
= 147 days, almost 5 months of cracking time on 3 GHz PC!
If attacker uses 50 3GHz pc, it will ONLY take 3 days!
For this reason, we use at least 9 char printable PSWDS.

Please dont forget that attackers can use assembly of PCs to crack your password hash.
Please also remember that attackers can use hardware password crackers (abt 100-1000 times faster than 1 PC) to crack your password hash.
Lessons learnt: Always use a much longer pswd; at least 9 chars (95) printable chars

3. Suppose that a certain email system uses hash of time when document is encrypted & emailed together. Example: time 20220203073000 means (3 February 2022 7:30:00)

- Alice encryption software works as follow:
- It uses 256-bit AES-encryption.
- Key used is 256-bit key, namely SHA256(Date & Time of email).
- When Alice use this encrypted email document and want to send to Bob, system looks at the time, then perform SHA256(Date & Time of email) and use this as 256-bit key for AES.
- Bob downloaded document into his pc. One day his pc is hacked and this document is in hands of hacker. Assume that hacker knows encryption algorithm and hash function used but not the date and time when document is sent.
- (a) Explore if hacker has a chance to read this document.
- Ans: Lets make some assumptions, say his document was written some time in 2022. Since the format is measured up to seconds, we only need to compute how many seconds there are in 2022. That will be the number of possible AES keys. Check this number is very small compared to 2^256.

- (b) What are the key lessons we can learn from this implementation of strong algorithm AES? Ans: Strong algorithm alone is not enough. Keys generated must be of as high entropy as possible and cryptographically random.

4. Why do we want to use slow hash for password hashing?
Ans: Slow hash is to slow down attacker's bruteforce or dictionary attack

5.1 Define a toy hash function $h : (\mathbb{Z}_2)^7 \rightarrow (\mathbb{Z}_2)^4$ by the rule $h(x) = xA$ where all operations are modulo 2 and

$$A = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

$(x_1 x_2 x_3 x_4 x_5 x_6 x_7) A = (0,1,0,1)$

$x_1 + x_2 + x_3 + x_4 = 0$
$x_2 + x_3 + x_4 + x_5 = 1$
$x_3 + x_4 + x_5 + x_6 = 0$
$x_4 + x_5 + x_6 + x_7 = 1$

Find all preimages of $(0,1,0,1)$.

Ans:

Let x = (x1,x2,...,x7). Solve xA = (0,1,0,1).

Write the system down as a system of 4 equations with 7 unknowns. System is already in echelon form.

Use guaussian elimination, can express x4 in terms of 3 free variables x5,x6,x7 – 8 possible choices.

Then similary express, x3,x2,x1 in terms of x5,x6,x7. Draw a table.

Answer: This is equivalent to solving the following system of equations:

$$\begin{aligned} x_1 + x_2 + x_3 + x_4 &= 0 \\ x_2 + x_3 + x_4 + x_5 &= 1 \\ x_3 + x_4 + x_5 + x_6 &= 0 \\ x_4 + x_5 + x_6 + x_7 &= 1 \end{aligned}$$

There are eight solutions for $(x_1, \ldots, x_7)$: $(1,1,1,1,0,0,0)$, $(1,1,0,0,0,0,1)$, $(1,0,1,0,0,1,0)$, $(1,0,0,1,0,1,1)$, $(0,1,1,0,1,0,0)$, $(0,1,0,1,1,0,1)$, $(0,0,1,1,1,1,0)$, $(0,0,0,0,1,1,1)$.

6. Why do we need 256-bit hash length instead of 128-bit length to pair with AES-128?
Ans:

Same same 2 for one

Hash space = 2^256
By birthday paradox, hash will repeat approx. after sqrt(2^256) = 2^128.
So cracking 256-bit secure hash takes 128-bit complexity, same as AES128.

7. Explain why a long codeword such as hippopotamus is not secure for use in Vignere cipher. Ans: Dictionary word.
Attacker just need to bruteforce each dictionary word as a possible keyword and do decryption. Since all operations are simple additions, even if dictionary has 10 million words, the bruteforced plaintext can be bruteforced out within minutes since PCs can crack millions of such cases within seconds, at most minutes.
To flush out the correct plaintext, use some frequently occurred words such as "THE" and search for this keyword. It shud appear often in the correct plaintext.

8. Keystream of {0,1} generated by pseudo-random number generators will be periodic. Secure keystream must necessary have long period. Show that even with extremely long period, some keystreams may not be suitable for use to generate encryption keys.
Ans:
Let N be as large as you want.
Consider keystream 11...100...0 (N 1s followed by N 0s, and repeat.
Clearly long period, but totally predictable, so bad keystream.
So long period alone NOT ENOUGH to guarantee secure keystream.

9. Explain why for CTR mode in block cipher encryption, your counter values must all be distinct to be secure.
Ans: If CTRi & CTRj is the same, then the XORing of the corresponding plaintext is EQUAL to the XORing of the associated ciphertext! Keys have disappeared because of XORing! Analogous to one time pad re-use.

10. If the hash reflected on the web page of download coincides with your computed hash, does it always mean the files have not been tampered with, assuming the hash used is a real secured hash?
Ans:
The only thing that can go wrong in such an instance:
If webpage has been hacked, the hacker can simply replace the software by malware, and then replace the original hash checksum with the new malware hash checksum.
Thus very impt for administrators to regularly check software and checksum have not been tampered with.

CE4062/CZ4062
Computer Security

Tutorial 2: Buffer Overflow

Tianwei Zhang

Q1

Short answers

(a) What do vulnerability, exploit, and payload refer to?

(b) What could be the potential consequences of a buffer overflow attack?

(c) What are the steps to utilize a buffer overflow vulnerability to execute shellcode?

Solution

Short answers

(a) What do vulnerability, exploit, and payload refer to?

Vulnerability: the weakness of a program that reduces its information assurance

Exploit: the technique the attacker takes to compromise the target system

Payload: the code the attacker wants the system to run.

Solution

Short answers

b) What could be the potential consequences of a buffer overflow attack?

Corrupt the data

Control flow hijacking

System crash

……

Solution

Short answers

c) What are the steps to utilize a buffer overflow vulnerability to execute shellcode?

1. *Convert the shellcode from C to assembly code, and then binary*
2. *Store the binary code in a buffer, which is allocated on the victim stack*
3. *Use the buffer overflow vulnerability to overwrite the return address with the address of the binary shellcode.*

Q2

In the following program, the function `get_plural` returns the plural of any input string. Assume the attacker can send arbitrary input to the `main` function. Please identify the possible buffer overflow vulnerabilities in this problem.

```
void get_plural(char* single, char* plural) {
  char* buf;
  plural[0] = '\0';
  int n = strlen(single);
  if (n == 0) return;

  buf = malloc(n+3);
  char last = single[n-1];
  if (last == 's')
    strcpy(buf, single);
  else if (last = 'h')
    sprintf(buf, "%ses", single);
  else
    sprintf(buf, "%ss", single);

  strcpy(plural, buf);
  free(buf)
}

void main(int argc, char* argv[]) {
  char plural_form[256];
  get_plural(argv[1], plural_form);
  printf("The plural of %s is %s\n", argv[1], plural_form);
}
```

Solution

`sprintf`: Similar as `printf` except that write the string to the first argument.

```
void get_plural(char* single, char* plural) {
  char* buf;
  plural[0] = '\0';
  int n = strlen(single);
  if (n == 0) return;

  buf = malloc(n+3);
  char last = single[n-1];
  if (last == 's')
    strcpy(buf, single);
  else if (last = 'h')
    sprintf(buf, "%ses", single);
  else
    sprintf(buf, "%ss", single);

  strcpy(plural, buf);
  free(buf)
}

void main(int argc, char* argv[]) {
  char plural_form[256];
  get_plural(argv[1], plural_form);
  printf("The plural of %s is %s\n", argv[1], plural_form);
}
```

buffer overflow

Q3

The following program is designed to generate a random number. It takes a password as input, but always fails to generate a random number. Luckily, this program is vulnerable to a buffer overflow attack. Our goal is to leverage this advantage to generate a random number. Please figure out a password that can achieve this.

```
char CheckPassword() {
  char good = 'N';
  char Password[100];
  gets(Password);

  return good;
}

int main(int argc, char* argv[]) {
  printf("Enter your password:");
  if(CheckPassword() == 'Y')
    printf("Your random number is %d\n", rand()%100);
  else{
    printf("You don't have the permission to get a random number");
    exit(-1);
  }
  return 0;
}
```

Solution

`gets`: can lead buffer overflow.

- Provide an input with size of 101, and end with 'Y' to overwrite good.

```
char CheckPassword() {
  char good = 'N';
  char Password[100];
  gets(Password);        ⇒ buffer overflow

  return good;
}

int main(int argc, char* argv[]) {
  printf("Enter your password:");
  if(CheckPassword() == 'Y')
    printf("Your random number is %d\n", rand()%100);
  else{
    printf("You don't have the permission to get a random number");
    exit(-1);
  }
  return 0;
}
```

Q4

A developer writes the following program for user authentication for his system. However, this program is vulnerable to buffer overflow attacks. Please give some examples of malicious input that an attacker can use to bypass the authentication.

```
int check_authentication(char *pwd) {
  int auth_flag = 0;
  char Password[] = "qwertyu";
  char buffer[8];

  strcpy(buffer, pwd);
  if (strncmp(buffer, Password, 8) == 0)
    auth_flag = 1;
  return auth_flag;
}

int main(int argc, char* argv[]) {
  if(check_authentication(argv[1]))
    printf("Access Granted\n");
  else{
    printf("Access Denied\n");
  }
  return 0;
}
```

Solution

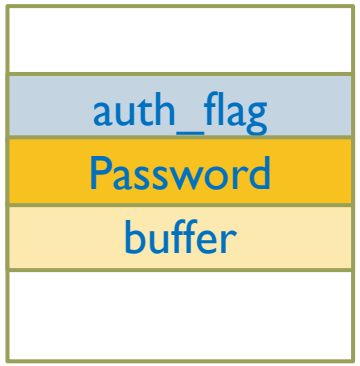
The attacker can leverage the **strcpy** to overflow the stack and bypass the authentication

- Overwrite the Password: **pwd = "abcdefgh" + "abcdefgh"**
- Overwrite the **auth_flag: pwd = "xxxxxxxx" + "xxxxxxxx" + "abcd"** -> the corresponding integer is 0x61626364

```
int check_authentication(char *pwd) {
  int auth_flag = 0;
  char Password[] = "qwertyu";
  char buffer[8];

  strcpy(buffer, pwd);   buffer overflow
  if (strncmp(buffer, Password, 8) == 0)
    auth_flag = 1;
  return auth_flag;
}

int main(int argc, char* argv[]) {
  if(check_authentication(argv[1]))
    printf("Access Granted\n");
  else{
    printf("Access Denied\n");
  }
  return 0;
}
```



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

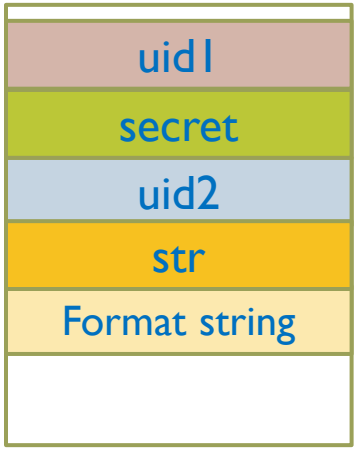
(a) Format string vulnerability

`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

1. 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 = 0xfffffffe (-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;

        if(size > len)
            return -1;

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

        return size;
    }
```

integer overflow

buffer overflow

CE4062/CZ4062
Computer Security

Tutorial 4: Software Vulnerability Defenses

Tianwei Zhang

Q1

Short answers

(a) Why are non-executable stack and heap not enough to defeat buffer overflow attacks?

(b) Distinguish three types of fuzzing techniques?

(c) What is a code reuse attack?

Solution

Short answers

(a) Why are non-executable stack and heap not enough to defeat buffer overflow attacks?

The return address can be overwritten to return to any code already loaded. For instance, the attacker may be able to cause a return into the libc execve() function with "/bin/sh" as an argument.

Solution

Short answers

b) Distinguish three types of fuzzing techniques?

- *Mutation-based fuzzing: randomly perturb the input in a heuristic way to test whether the system will crash*
- *Generation-based fuzzing: generate test cases based on the specification of the input format to test*
- *Coverage-based fuzzing: craft test cases based on the feedback of code coverage*

Solution

Short answers

c) What is a code reuse attack?

Instead of injecting the malicious code into the memory, the attacker can compromise the control flow to jump to the existing library for attacks.

Q2

For string copy operation, `strncpy` is regarded as "safer" than `strcpy`. However, improper use of `strncpy` can also incur vulnerabilities. Please describe the problem in the following program, and what consequences it will cause.

```c
#include <stdio.h>
#include <string.h>

int main() {
  char src[] = "geeksforgeeks";

  char dest[8];
  strncpy(dest, src, 8);
  int len = strlen(dest);

  printf("Copied string: %s\n", dest);
  printf("Length of destination string: %d\n", len);

  return 0;
}
```

Solution

Does not automatically add the NULL value to dest if n is less than the length of string src. So it is safer to always add NULL after strncpy.

- In this program, dest does not have a terminator at the end. Then the function `strlen` will keep counting, which can cause segmentation fault.

```c
#include <stdio.h>
#include <string.h>

int main() {
  char src[] = "geeksforgeeks";

  char dest[8];
  strncpy(dest, src, 8);
  int len = strlen(dest);

  printf("Copied string: %s\n", dest);
  printf("Length of destination string: %d\n", len);

  return 0;
}
```

Q3

StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.

(a) In some implementations, the canary value is a 64-bit integer that is randomly generated each time the program runs. Explain why this prevents the basic form of buffer-overflow attacks

(b) What is a security drawback to choosing the canary value at compile time instead of at run time?

(c) If the value must be fixed, what will be a good choice?

(d) List an attack which can defeat the StackGuard.

Solution

StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.

a) In some implementations, the canary value is a 64-bit integer that is randomly generated each time the program runs. Explain why this prevents the basic form of buffer-overflow attacks

This guarantees the attacker cannot guess canary correctly every time, and cannot corrupt the stack without changing the canary values.

Solution

StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.

b) What is a security drawback to choosing the canary value at compile time instead of at run time?

After compiling, the canary value will be fixed. The attacker may use brute-force attack to guess the correct value, and then perform the buffer overflow attack with that value.

Solution

StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.

c) If the value must be fixed, what will be a good choice?

Using terminator canary: {\0, newline, linefeed, EOF}. The attacker cannot copy strings beyond the terminator.

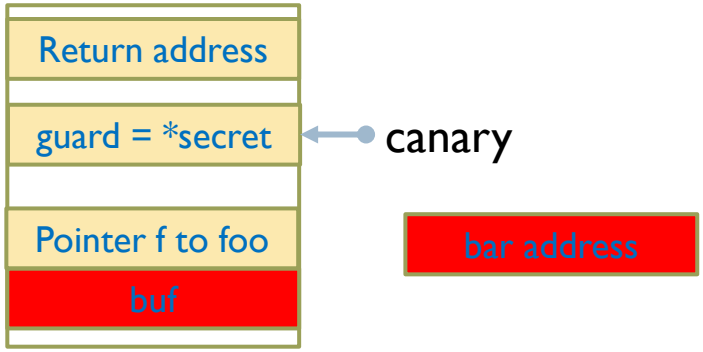
Solution

StackGuard is a mechanism for defending C programs against stack-based buffer overflows. It detects memory corruption using a canary, a known value stored in each function's stack frame immediately before the return address.

d) List an attack which can defeat the StackGuard.

The attacker can overwrite the function pointer instead of the return address

```
void foo () {...}
void bar () {...}
int main() {
    void (*f) () = &foo;
    char buf [16];
    gets(buf);
    f();
}
```

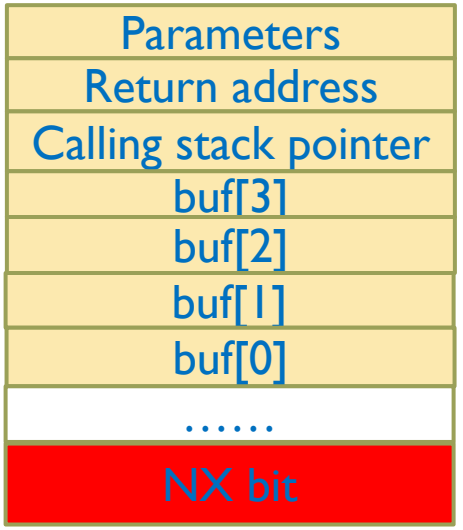


Q4

One possible solution to defeat buffer overflow attacks is to set the stack memory as Non-executable (NX). This is usually achieved by the OS and the paging hardware. However, imagine that a machine does not support the non-executable feature, then we can implement this functionality at the software level. The compiler can allocate each stack frame in a separate page, and associate a software-manipulated NX bit at the bottom of this page, controlling if this page (stack frame) is non-executable. The structure of a stack frame is shown in the Figure below.

Since the NX bit is at the bottom of the memory page, the buffers inside this stack frame is not able to overwrite this bit to make it executable.

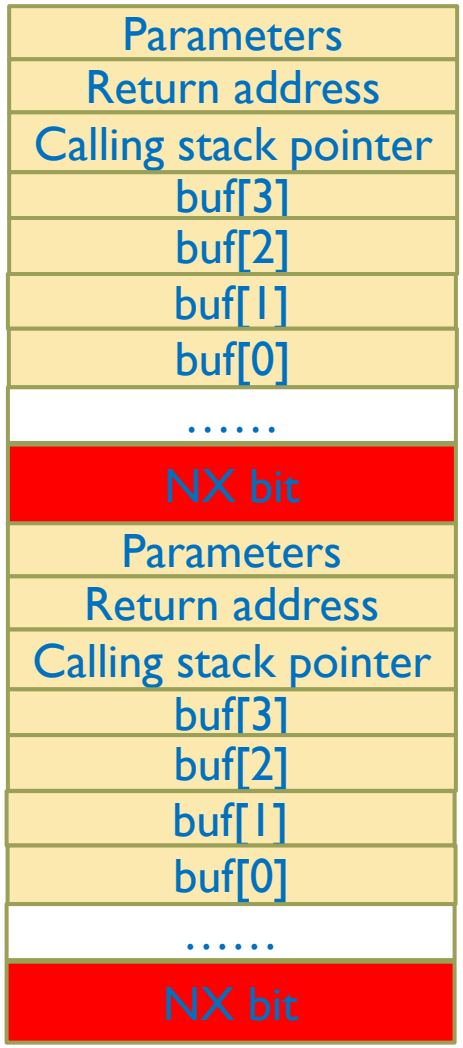
Describe a buffer overflow attack that can still overwrite NX bits.



Solution: Overflow from callee frame

Although the buffers inside the frame cannot overwrite the NX bit. But it can call another function, whose buffer can be used by the attacker to overwrite the NX bit. Then the caller's frame will become executable.

```
foo( ) {
  bar( );
}
bar( ) {
  char buf[4];
}
```



CE4062/CZ4062
Computer Security

Tutorial 5: Operating System Security

Tianwei Zhang

Q1

Short answers

(a) Give an example of how a rootkit can compromise the system after obtaining the root privilege.

(b) Briefly describe three stages employed by the OS

(c) What is the controlled invocation? What is its potential danger?

Solution

Short answers

(a) Give an example of how a rootkit can compromise the system after obtaining the root privilege.

A rootkit can compromise different kernel structures to achieve malicious behaviors. For instance,

- *It can change a function pointer in the system call table to make it point to a malicious function.*
- *It can directly change the system call function, making it jump to a malicious function.*
- *It can change a function pointer in the interrupt descriptor table to make it point to a malicious function.*

Solution

Short answers

b) Briefly describe three stages employed by the OS

- *Identification & Authentication: authenticate if a user attempting to enter the system is correct or not.*
- *Access Control: when a subject (process, user, …) wants to access an object (file, network socket, …), check if such access is allowed.*
- *Logging & auditing: record all protection-orientated activities, important to understand what happened, why, and catching things that shouldn't*

Solution

Short answers

c) What is the controlled invocation? What is its potential danger?

A user executes a privileged program by inheriting the permission of the program's owner. This is usually achieved by enabling SUID

As the user has the program owner's privileges when running a SUID program, the program should only do what the owner intended. By tricking a SUID program owned by root to do unintended things, an attacker can act as the root.

Q2

Consider a computer system with three users: Alice, Bob, and Cindy. Alice owns the file *alicerc*, and Bob and Cindy can read it. Cindy can read and write the file *bobrc*, which Bob owns, but Alice can only read it. Only Cindy can read and write the file *cindyrc*, which she owns. Assume that the owner of each of these files can execute it.

(a) Create the corresponding access control matrix.

(b) Cindy gives Alice permission to read *cindyrc*, and Bob removes Alice's ability to read *bobrc*. Show the new access control matrix

(c) Show the capabilities associated with Alice.

Solution

Consider a computer system with three users: Alice, Bob, and Cindy. Alice owns the file *alicerc*, and Bob and Cindy can read it. Cindy can read and write the file *bobrc*, which Bob owns, but Alice can only read it. Only Cindy can read and write the file *cindyrc*, which she owns. Assume that the owner of each of these files can execute it.

a. Create the corresponding access control matrix.

	alicerc	bobrc	cindyrc
Alice	Execute	Read	
Bob	Read	Execute	
Cindy	Read	Read Write	Read Write Execute

Solution

Consider a computer system with three users: Alice, Bob, and Cindy. Alice owns the file *alicerc*, and Bob and Cindy can read it. Cindy can read and write the file *bobrc*, which Bob owns, but Alice can only read it. Only Cindy can read and write the file *cindyrc*, which she owns. Assume that the owner of each of these files can execute it.

b. Cindy gives Alice permission to read cindyrc, and Bob removes Alice's ability to read bobrc. Show the new access control matrix.

	alicerc	bobrc	cindyrc
Alice	Execute	Read	Read
Bob	Read	Execute	
Cindy	Read	Read Write	Read Write Execute

Solution

Consider a computer system with three users: Alice, Bob, and Cindy. Alice owns the file *alicerc*, and Bob and Cindy can read it. Cindy can read and write the file *bobrc*, which Bob owns, but Alice can only read it. Only Cindy can read and write the file *cindyrc*, which she owns. Assume that the owner of each of these files can execute it.

c. Show the capabilities associated with Alice.

	alicerc	bobrc	cindyrc
Alice	Execute		Read
Bob	Read	Execute	
Cindy	Read	Read Write	Read Write Execute

{alicerc: {execute}; bobrc: {}; cindyrc: {read}}

Q3

Let's consider the scenario in Q2 again. Assume this is the Unix system. The users Bob and Cindy are in the same group, while Alice is in a different group.

(a) For the original access control matrix in Q2(a), please write the permission for the files *alicerc*, *bobrc* and *cindyrc*.

(b) To adjust the permissions in Q2(b), please write the corresponding commands for *cindyc* and *bobrc*, respectively.

Solution

Let's consider the scenario in Q2 again. Assume this is the Unix system. The users Bob and Cindy are in the same group, while Alice is in a different group.

a. For the original access control matrix in Q2(a), please write the permission for the files *alicerc*, *bobrc* and *cindyrc*.

alicerc:	--x---r--	104
bobrc:	--xrw-r--	164
cindyrc:	rwx------	700

	alicerc	bobrc	cindyrc
Alice	Execute	Read	
Bob	Read	Execute	
Cindy	Read	Read Write	Read Write Execute

Solution

Let's consider the scenario in Q2 again. Assume this is the Unix system. The users Bob and Cindy are in the same group, while Alice is in a different group.

b. To adjust the permissions in Q2(b), please write the corresponding commands for *cindyc* and *bobrc*, respectively.

alicerc:	--x---r--		
bobrc:	--xrw----	chmod 160 bobrc	chmod o-r bobrc
cindyrc:	rwx---r--	chmod 704 cindyrc	chmod o+r cindyrc

	alicerc	bobrc	cindyrc
Alice	Execute	Read	
Bob	Read	Execute	
Cindy	Read	Read Write	Read Write Execute

Q4

A group of researchers is working on analyzing web search results from a major Internet search provider. At the Internet company, a group of search engineers collects and updates databases of: search queries, IP (internet protocol) addresses where the queries came from and timestamps for the queries made by online users. A search manager is in charge of the group of engineers, and can read the query and timestamps database, but not the IP address database -- due to privacy concerns. The researchers are able to access the databases with read-only privileges. The general public should not have access to the database for privacy reasons.

(a) Complete the access control matrix by filling in the access permissions for the different objects shown. Each entry can be either read, write, read/write, or '-' (for no access).

(b) List the ACLs for each object (or class of objects)

(c) List the capabilities for each subject (or class of subjects)

Solution

A group of researchers is working on analyzing web search results from a major Internet search provider. At the Internet company, a group of search engineers collects and updates databases of: search queries, IP (internet protocol) addresses where the queries came from and timestamps for the queries made by online users. A search manager is in charge of the group of engineers, and can read the query and timestamps database, but not the IP address database -- due to privacy concerns. The researchers are able to access the databases with read-only privileges. The general public should not have access to the database for privacy reasons.

a. Complete the access control matrix by filling in the access permissions for the different objects shown. Each entry can be either read, write, read/write, or '-' (for no access).

	Query	IP	Timestamp
Search Engineers	rw	rw	rw
Search Manager	r	-	r
Researchers	r	r	r
Public	-	-	-

Solution

b. List the ACLs for each object (or class of objects)

Query: {SE: {rw}; SM: {r}; R: {r}; P: {-}}
IP: {SE: {rw}; SM: {-}; R: {r}; P: {-}}
Timestamp: {SE: {rw}; SM: {r}; R: {r}; P: {-}}

	Query	IP	Timestamp
Search Engineers	rw	rw	rw
Search Manager	r	-	r
Researchers	r	r	r
Public	-	-	-

Solution

c. List the capabilities for each subject (or class of subjects)

Search Engineers: {Query: {rw}; IP: {rw}; Timestamp: {rw}}
Search Manager: {Query: {r}; IP: {-}; Timestamp: {r}}
Research: {Query: {r}; IP: {r}; Timestamp: {r}}
Pub lic: {Query: {-}; IP: {-}; Timestamp: {-}}

	Query	IP	Timestamp
Search Engineers	rw	rw	rw
Search Manager	r	-	r
Researchers	r	r	r
Public	-	-	-

CE4062/CZ4062
Computer Security

Tutorial 6: Operating System Protection

Tianwei Zhang

Q1

Short answers

(a) Describe what is the confinement strategy, and why it can be used for malware testing and analysis.

(b) List the security functionalities offered by the TPM.

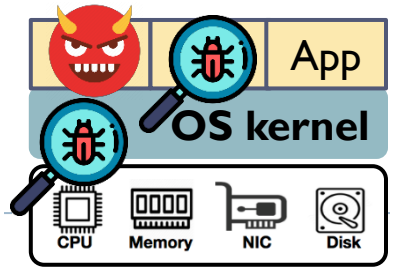
(c) Describe the lifecycle of an SGX enclave application

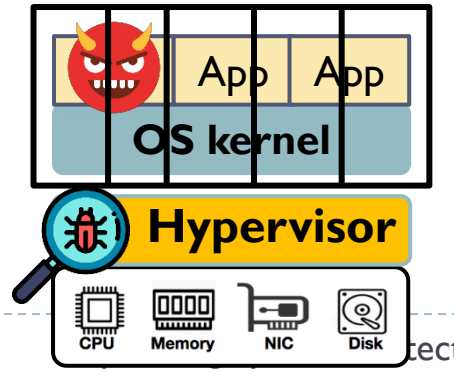
Solution

Short answers

(a) Describe what is the confinement strategy, and why it can be used for malware testing and analysis.

- *Confinement: isolate some components in the system and restrict its impact on other components.*
- *One important application: malware analysis.*
  - *When launching a malware inside the system and analyze its behaviors, the malware can be too strong to compromise the analyzer and the entire system. We can set up an isolated environment for the malware and analyze it from the outside.*
  - *One possible solution: deploy malware inside a virtual machine and analyze it from the hypervisor*





Solution

Short answers

b) List the security functionalities offered by the TPM

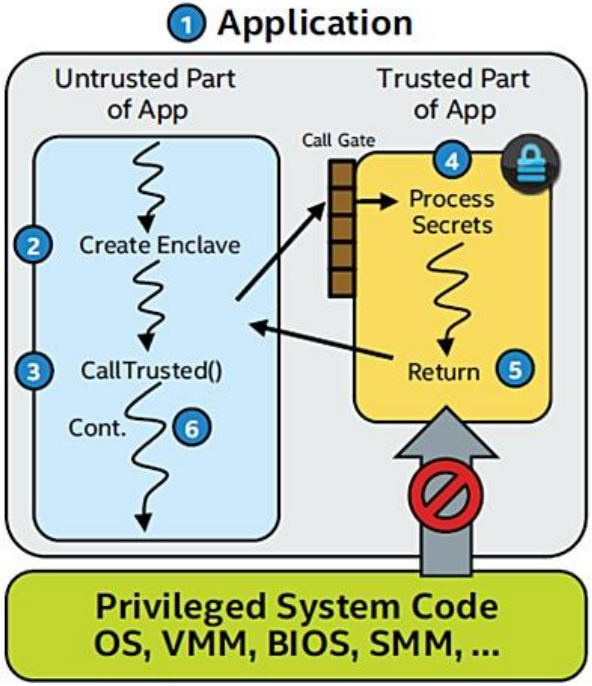
- *Platform integrity: building a chain of trust. Establish a secure boot process from the TPM, and continue until the OS has fully booted and applications are running.*
- *Data encryption: encrypt the data with the key in the hardware. TPM can provide platform authentication before data encryption*
- *Remote attestation: provide unforgeable evidence about the security of its software to the remote client.*

Solution

Short answers

c) Describe the lifecycle of an SGX enclave application.

1. *The target application is divided into two parts: trusted and untrusted.*
2. *The untrusted part creates an enclave, and loads the code/data into this enclave.*
3. *During the execution, when the application needs to run the trusted code, the untrusted part calls an API to enter the enclave.*
4. *The trusted code is executed in the enclave.*
5. *After the trusted code is finished, it exits the enclave and returns to the untrusted code.*
6. *The untrusted code can continue the execution, and repeat steps 3-5 when trusted code needs to be executed again.*



Q2

Early Intel processors (e.g., the 8086) did not provide hardware support for dual-mode operation (i.e., support for a separate user mode and kernel mode). If a system is implemented on such processors to support the multi-programming scenario, describe one confidentiality, integrity and availability threat respectively in this system, due to the lack of hardware support.

Solution

Early Intel processors (e.g., the 8086) did not provide hardware support for dual-mode operation (i.e., support for a separate user mode and kernel mode). If a system is implemented on such processors to support the multi-programming scenario, describe one confidentiality, integrity and availability threat respectively in this system, due to the lack of hardware support.

If there is no hardware support for different modes, then every component (user programs, kernel functions and services) has the same privilege. Then a malicious program can easily affect other processes, services and then the entire system. For instance:

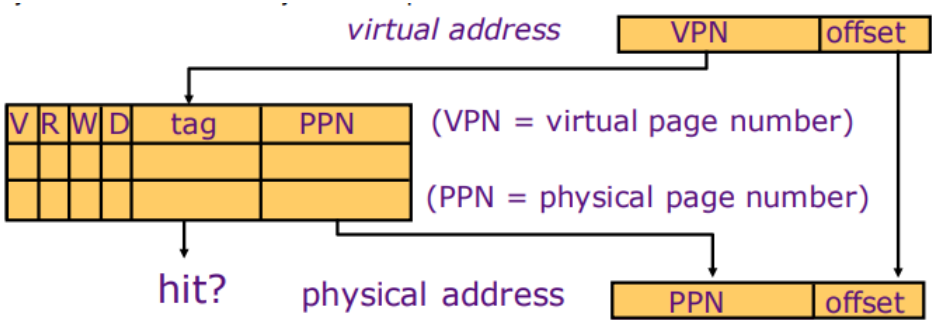
- *Confidentiality threat: any malicious program can read the memory data of other processes and kernel as there is no memory access control restrictions.*
- *Integrity threat: any malicious program can modify the code of other processes and kernel as there is no memory access control restrictions.*
- *Availability threat: any malicious program can disable the interrupts, and avoid getting re-scheduled. Then it will occupy the CPU permanently while other processes can never be scheduled.*

Q3

Translation Lookaside Buffer (TLB) is a small hardware component that caches the recent translations of virtual memory to physical memory. It can help accelerate the memory access of programs. When a program wants to access the data with the specific virtual memory address, the system will check if there is an entry of this address in the TLB, and if the program has the access permission to this address. If both checks pass, then the corresponding physical address will be generated, and the access is allowed. Otherwise, a hardware interrupt will be triggered. Figure Q3 shows the mechanism of the TLB. Note that the TLB can be updated only when the CPU is in the kernel mode.

(a) The TLB can be regarded as one type of hardware-based reference monitor. Please list the requirements for a reference monitor.

(b) Analyze if the TLB can satisfy these three requirements.



Solution

a. The TLB can be regarded as one type of hardware-based reference monitor. Please list the requirements for a reference monitor.

Reference monitor: a security mechanism that monitors and mediates requests from the protected targets at runtime

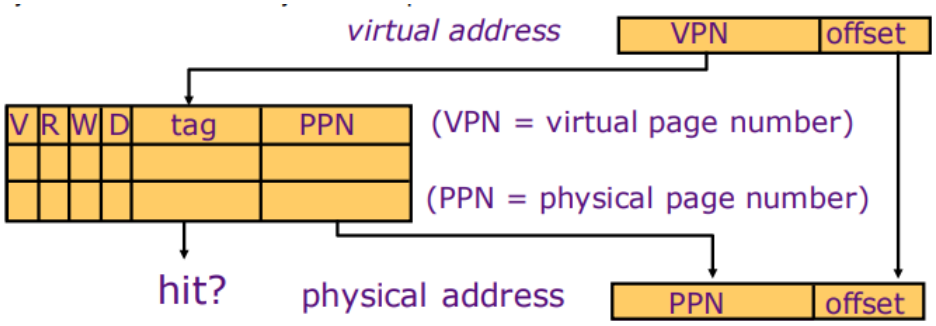
- *Enforce some security policies, e.g., confinement*
- *When any security policy is violated, RM can deny the request*

Requirements of RM:

- *Function requirement: the reference validation mechanism must always be invoked, i.e., it can observe all the requests and deny any malicious ones*
- *Security requirement: the reference validation mechanism must be tamper-proof*
- *Assurance requirement: the reference validation mechanism must be small enough to be analyzed and tested.*

Solution

a. Analyze if the TLB can satisfy these three requirements.



TLB can satisfy all the three requirements of RM:

- *Function requirement: all the memory accesses from all the program must go through the TLB. When the TLB denies this memory request, the program is not able to access the data*
- *Security requirement: any user-level program cannot change any entries in the TLB. Only the kernel has the privilege to do so*
- *Assurance requirement: the TLB is relatively a small hardware unit and is intensively verified. Hardware is usually more trusted compared to software components.*

Q4

Trusted Computing Base (TCB) is an important concept in computer security. It refers to the set of components (e.g., hardware, software, firmware, etc.) that must be trusted in order to guarantee the security of the entire system. Well protection of the TCB can defend the system against the threats from outside the TCB.

(a) As a system designer, do we expect to have a larger TCB or smaller TCB? Why?

(b) Consider a conventional cloud computing scenario, where you launch a virtual machine in a cloud service provider (e.g., Amazon). Figure Q4 shows the system architecture of a cloud server running your virtual machines together with other users' virtual machines. Please specify which components are included in the TCB, and what entities and components are considered untrusted.

(c) Assume the cloud provider adopts the TEE solution – AMD SEV processors to protect the users' virtual machines. In this case, specify which components are included in the TCB. Discuss how SEV processors can protect the virtual machines from untrusted components outside of the TCB.

Solution

a. As a system designer, do we expect to have a larger TCB or smaller TCB? Why?

Smaller TCB is preferred.

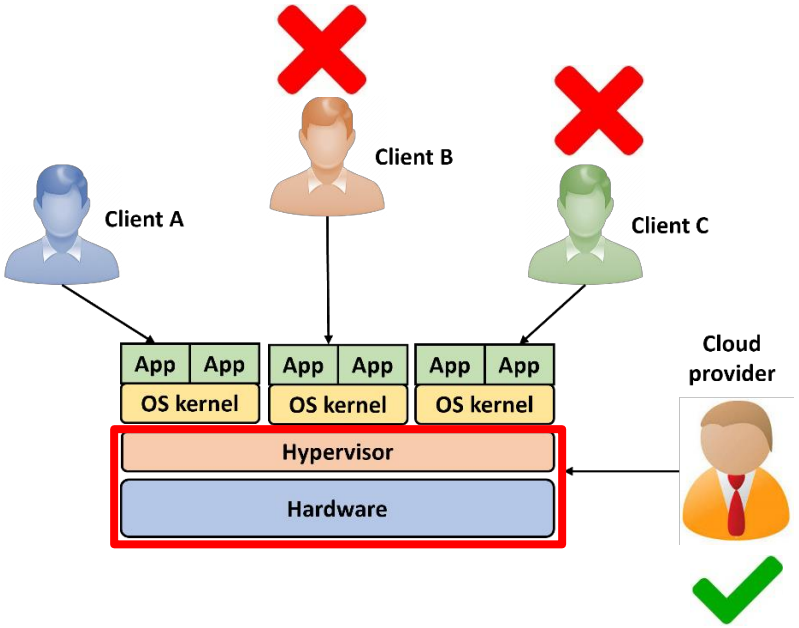
The components inside the TCB must guarantee to be trusted. When designing a system, it is more challenging to make more, and larger components trusted. An ideal secure system is to have a minimal TCB, which can defeat any threats from any components outside the TCB.

Solution

b. Consider a conventional cloud computing scenario, where you launch a virtual machine in a cloud service provider (e.g., Amazon). Figure Q4 shows the system architecture of a cloud server running your virtual machines together with other users' virtual machines. Please specify which components are included in the TCB, and what entities and components are considered untrusted.

TCB: hardware and hypervisor in each cloud server. The cloud provider must be trusted.

The virtual machines from other clients can be untrusted. Even they contain malware, the hypervisor can provide isolation, and prevent the malware from compromising other virtual machines.



Solution

c. Assume the cloud provider adopts the TEE solution – AMD SEV processors to protect the users' virtual machines. In this case, specify which components are included in the TCB. Discuss how SEV processors can protect the virtual machines from untrusted components outside of the TCB.

TCB: only the hardware with the SEV feature in each cloud server. The hypervisor can be malicious, and the cloud provider does not need to be trusted.

The virtual machines from other clients can also be untrusted. The hardware provides protection and isolation from the hypervisor and other virtual machines.

- *Virtual memory encryption*
- *Remote attestation.*