Please check your attendance using Blackboard!

Lecture 5 – Various Software Vulnerabilities

[COSE451] Software Security

Instructor: Seunghoon Woo

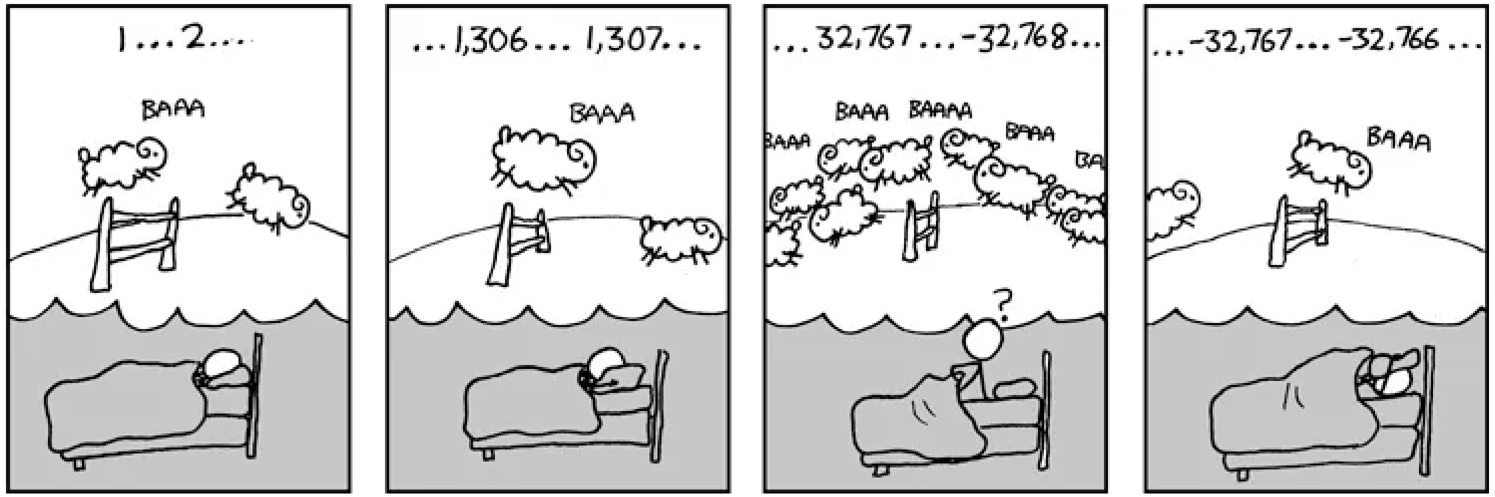
Spring 2024

Overview

- **Various Software Vulnerabilities**
  - Integer overflow/underflow
  - Command injection
  - Path traversal

Integer overflow/underflow

- **Integer overflow/underflow**
  - A value exceeding the maximum (minimum) value for an integer data type results in a sudden change to a very small (large) value



https://xkcd.com/571/

Integer overflow/underflow

- **Integer overflow/underflow**

Data structure	Range
char	0 ~ 65535 ('\u0000' ~ '\uffff')
byte (1 byte)	-128 ~ 127 (-2^7 ~ 2^7 -1)
short (2 bytes)	-32,768 ~ 32,767 (-2^{15} ~ 2^{15} - 1)
int (4 bytes)	-2,147,483,648 ~ 2,147,483,647 (-2^{31} ~ 2^{31} -1)
long (8 bytes)	-9,223,372,036,854,774,808 ~ 9,223,372,038,854,775,807 (-2^{63} ~ 2^{63} -1)
float (4 bytes)	1.4E-45 ~ 3.4E38 (1.4 * 10^{-45} ~ 3.4 * 10^{-38})
double (8 bytes)	4.9E-324 ~ 1.8E308 (4.9 * 10^{-324}~ 1.8 * 10^{308})

Integer overflow/underflow

- **Integer overflow/underflow**
  - Signed int: 4 bytes (32 bits)
    - `MIN: 0000 0000 0000 0000 0000 0000 0000 0000`
    - `MAX: 1111 1111 1111 1111 1111 1111 1111 1111`

Integer overflow/underflow

- **Integer overflow/underflow**
  - Signed int: 4 bytes (32 bits) - **WRONG**
    - `MIN: 0000 0000 0000 0000 0000 0000 0000 0000`
    - `MAX: 1111 1111 1111 1111 1111 1111 1111 1111`
    - The first bit represent the sign (0: positive, 1: negative)

Integer overflow/underflow

- **Integer overflow/underflow**
  - Signed int: 4 bytes (32 bits)

```
- MIN: 0000 0000 0000 0000 0000 0000 0000 0000
- MAX: 0111 1111 1111 1111 1111 1111 1111 1111
          = 2^31 - 1 = 2,147,483,647
```

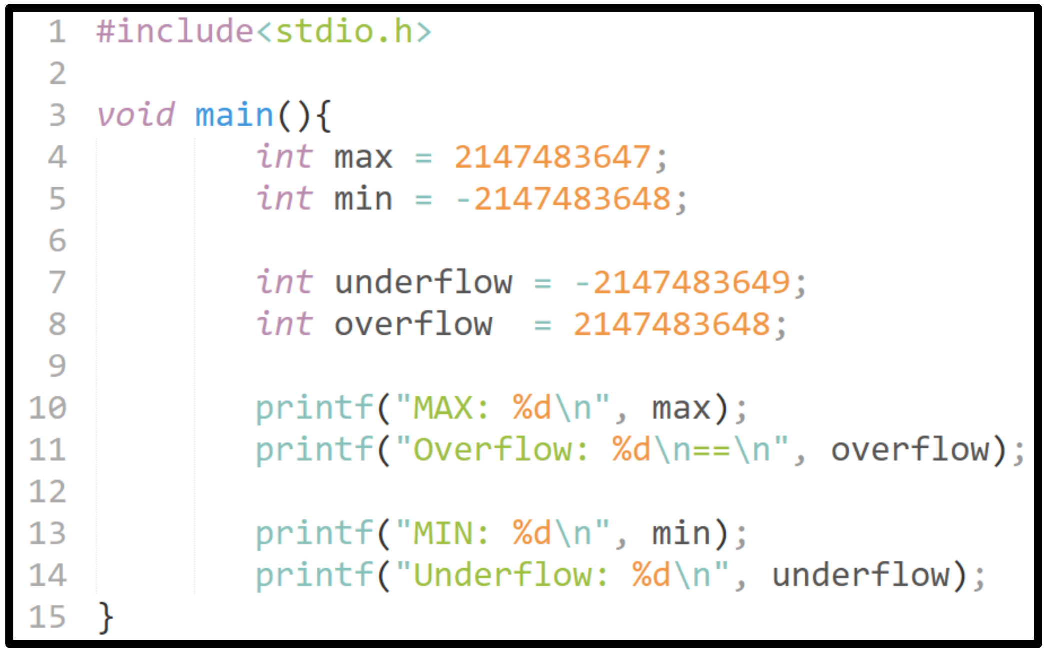
Integer overflow/underflow

- **Integer overflow/underflow**
  - Signed int: 4 bytes (32 bits)

```
- MIN: 0000 0000 0000 0000 0000 0000 0000 0000
- MAX: 0111 1111 1111 1111 1111 1111 1111 1111
          = 2^31 - 1 = 2,147,483,647
- MAX+1: 1000 0000 0000 0000 0000 0000 0000 0000
            = -2^31 = -2,147,483,648
```

Integer overflow/underflow

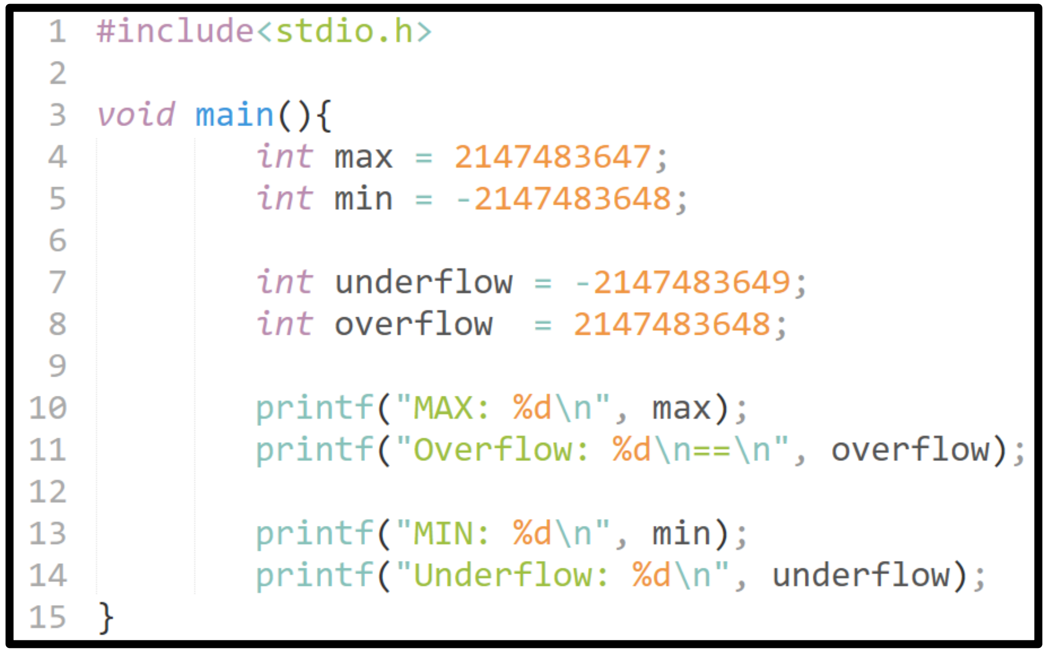
- **Integer overflow/underflow**
  - Example



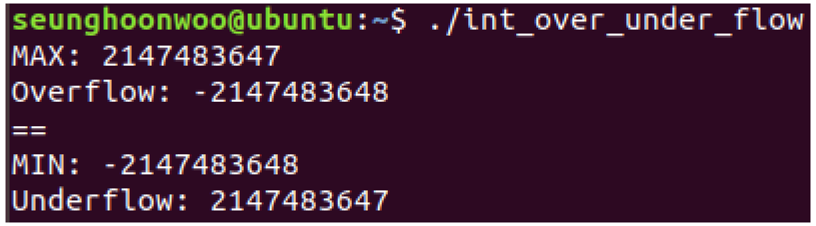
```c
#include<stdio.h>

void main(){
        int max = 2147483647;
        int min = -2147483648;

        int underflow = -2147483649;
        int overflow  = 2147483648;

        printf("MAX: %d\n", max);
        printf("Overflow: %d\n==\n", overflow);

        printf("MIN: %d\n", min);
        printf("Underflow: %d\n", underflow);
}
```

Integer overflow/underflow

- **Integer overflow/underflow**
  - Example



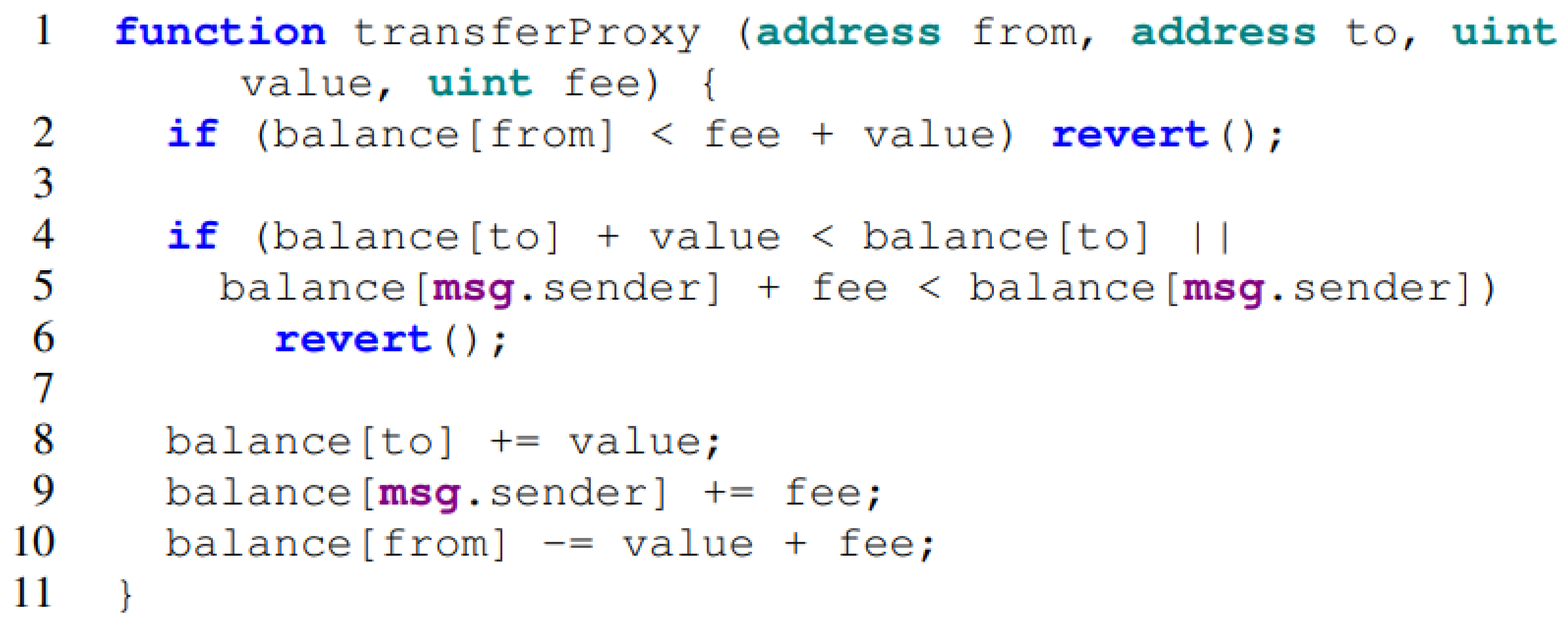
```c
#include<stdio.h>

void main(){
        int max = 2147483647;
        int min = -2147483648;

        int underflow = -2147483649;
        int overflow  = 2147483648;

        printf("MAX: %d\n", max);
        printf("Overflow: %d\n==\n", overflow);

        printf("MIN: %d\n", min);
        printf("Underflow: %d\n", underflow);
}
```



```
seunghoonwoo@ubuntu:~$ ./int_over_under_flow
MAX: 2147483647
Overflow: -2147483648
==
MIN: -2147483648
Underflow: 2147483647
```

Integer overflow/underflow

- **Integer overflow/underflow**
  - A vulnerability discovered in a popular smart contract



```
function transferProxy (address from, address to, uint
    value, uint fee) {
  if (balance[from] < fee + value) revert();

  if (balance[to] + value < balance[to] ||
    balance[msg.sender] + fee < balance[msg.sender])
      revert();

  balance[to] += value;
  balance[msg.sender] += fee;
  balance[from] -= value + fee;
}
```

So, Sunbeom, et al. VeriSmart: A highly precise safety verifier for Ethereum smart contracts. In: *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020. p. 1678-1694.

Integer overflow/underflow

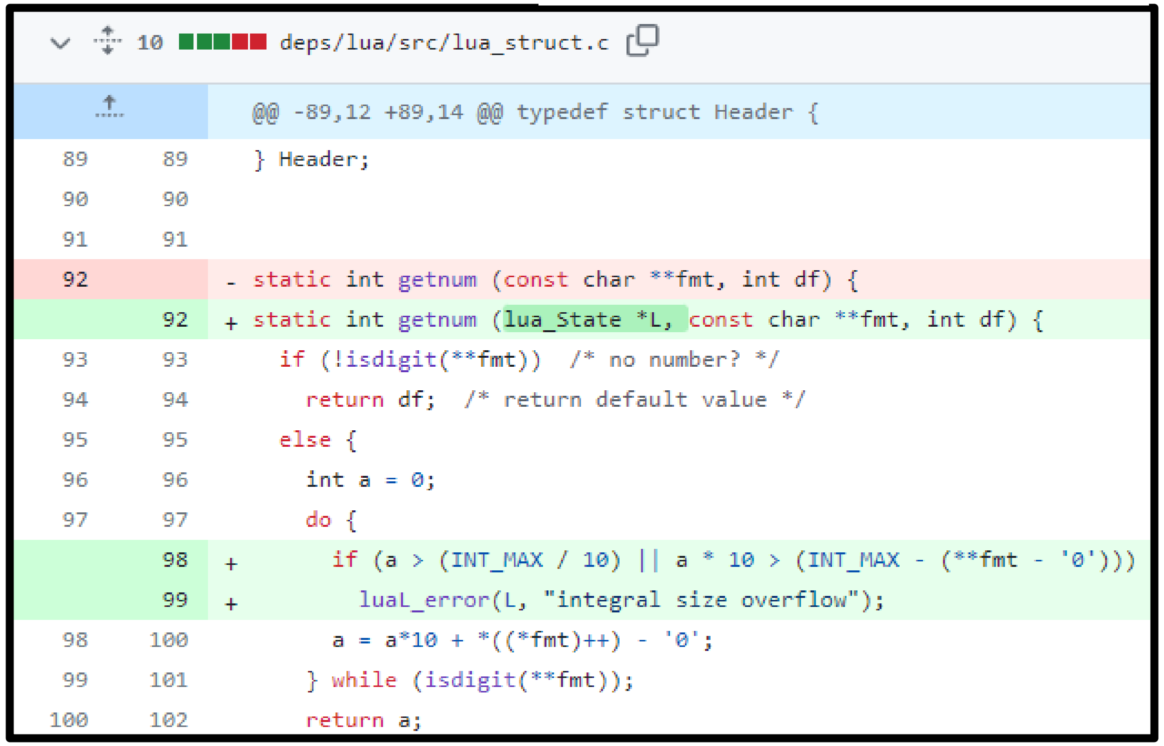
- **Integer overflow/underflow**
  - Related CWEs
    - CWE-190: Integer Overflow or Wraparound (14)
    - CWE-191: Integer Underflow (Wrap or Wraparound)
    - CWE-197: Numeric Truncation Error
    - CWE-369: Divide by Zero
    - CWE-680: Integer Overflow to Buffer Overflow

* Highlighted numbers: the rankings of the top 25 most dangerous Common Weakness Enumeration (CWE) entries in 2023

Integer overflow/underflow

- **Integer overflow/underflow**
  - Real-world example: CVE-2020-14147 in Redis
    - An in-memory database software
    - One of the most popular C software on GitHub (rank 4 as of Feb. 2024)

WOOSEUNGHOON committed on Feb 10, 2020



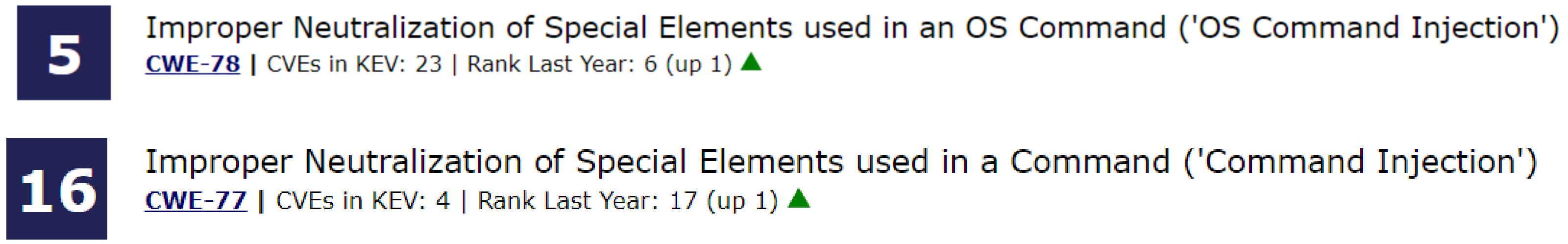
```
10 deps/lua/src/lua_struct.c
@@ -89,12 +89,14 @@ typedef struct Header {
 } Header;


- static int getnum (const char **fmt, int df) {
+ static int getnum (lua_State *L, const char **fmt, int df) {
   if (!isdigit(**fmt))  /* no number? */
     return df;  /* return default value */
   else {
     int a = 0;
     do {
+      if (a > (INT_MAX / 10) || a * 10 > (INT_MAX - (**fmt - '0')))
+        luaL_error(L, "integral size overflow");
       a = a*10 + *((*fmt)++) - '0';
     } while (isdigit(**fmt));
     return a;
```

Integer overflow/underflow

- **Integer overflow/underflow**
  - How can we prevent integer overflow/underflow attacks?
    1. Correct input validation (important)
    2. Using GCC compile options
       - E.g., GCC –ftrapv
         - Generating traps for signed overflow on addition, subtraction, multiplication operations

Command injection

- **Sometimes it is more convenient to utilize pre-existing software rather than writing code from scratch for certain functionalities**
  - E.g., if we want to print the contents of a file, use the system's cat function
- **C/C++ utilizes the system function**
  - E.g., system("cat /etc/passwd")

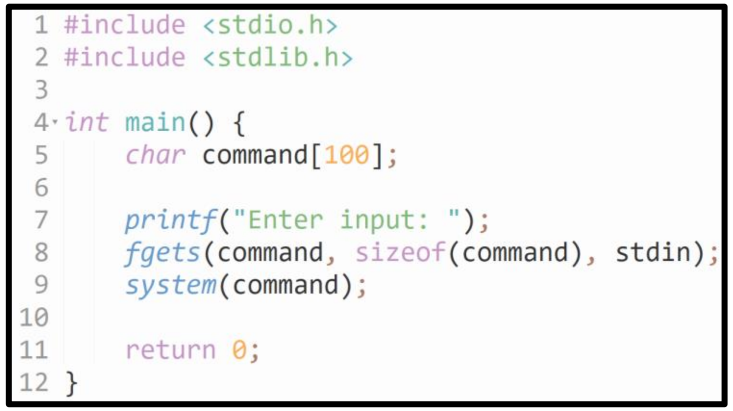


5 Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')
CWE-78 | CVEs in KEV: 23 | Rank Last Year: 6 (up 1) ▲

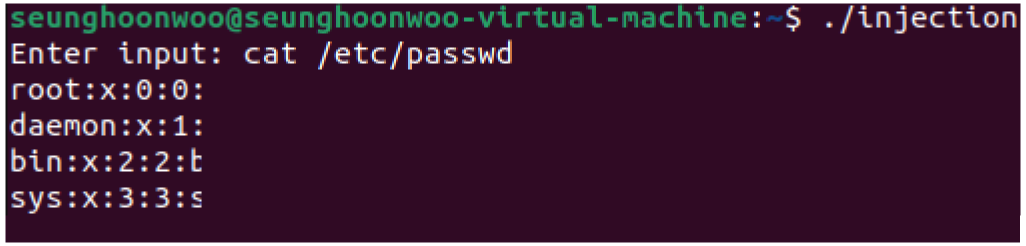
16 Improper Neutralization of Special Elements used in a Command ('Command Injection')
CWE-77 | CVEs in KEV: 4 | Rank Last Year: 17 (up 1) ▲

Command injection

- **Using system functions**
  - Pros
    - Easily utilize the software already installed (e.g., cat)
  - Cons
    - The arguments of the function are passed as shell commands: this can yield critical attacks



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

int main() {
    char command[100];

    printf("Enter input: ");
    fgets(command, sizeof(command), stdin);
    system(command);

    return 0;
}
```



```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./injection
Enter input: cat /etc/passwd
root:x:0:0:
daemon:x:1:
bin:x:2:2:t
sys:x:3:3:s
```

Command injection

- **Injection**
  - Injecting malicious data into a program to execute it as system commands, code, database queries, etc.
- **Command injection**
  - Executing user input as system commands
  - Arbitrary commands may be executed if user input is not properly validated

* Command1 | Command 2: The output of command1 is passed as input to command2, connecting and executing both commands.

Command injection

Meta characters	Description	Example
$	Shell environment variables	`$ echo $SHELL` `/bin/bash`
&&	Executing the next command after the previous command execution	`$ echo hello && echo bye` `hello` `bye`
;	Command separators	`$ echo hello ; echo bye` `hello` `bye`
|	Command piping*	`$ ls | grep injection` `injection.c`
*	Wildcard (Used for string pattern matching)	`$ echo .*` `. .. .local .profile`
`	Command substitution	`$ date` `2024. 03. 26. (Tue) 23:19:15 KST` ``$ current_date=`date` `` `$ echo "The current date is $current_date."` `The current date is 2024. 03. 26. (Tue) 23:19:17 KST.`

* Command1 | Command 2: The output of command1 is passed as input to command2, connecting and executing both commands.

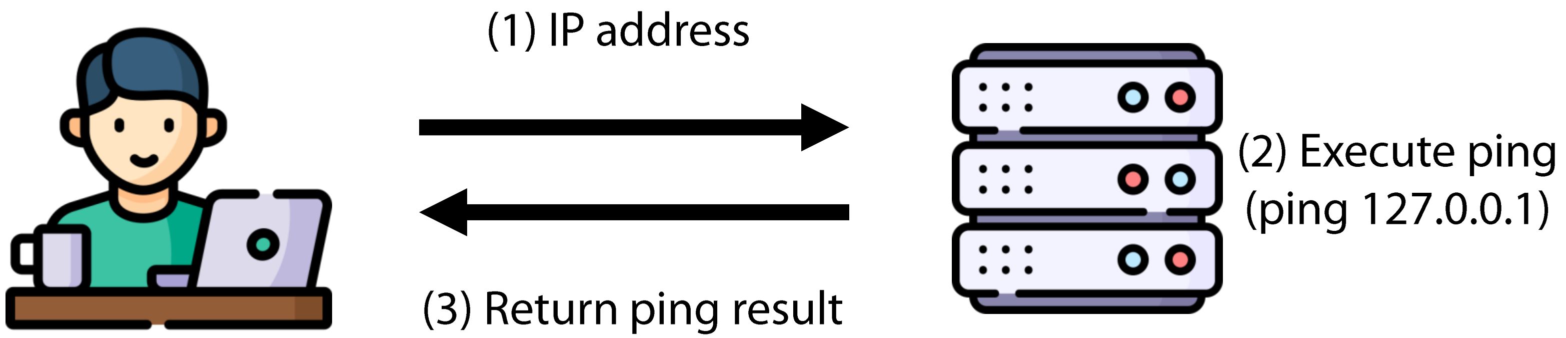
Command injection

Meta characters	Description	Example
$	Shell environment variables	$ echo $SHELL /bin/bash
&&	Executing the next command after the previous command execution	$ echo hello && echo bye hello bye
;	Command separators	$ echo hello ; echo bye hello bye
|	Command piping*	$ ls | grep injection injection.c
*	Wildcard (Used for string pattern matching)	$ echo .* . .. .local .profile

`	Command substitution	$ date
		2024. 03. 26. (Tue) 23:19:15 KST
		$ current_date=`date`
		$ echo "The current date is $current_date."
		The current date is 2024. 03. 26. (Tue) 23:19:17 KST.

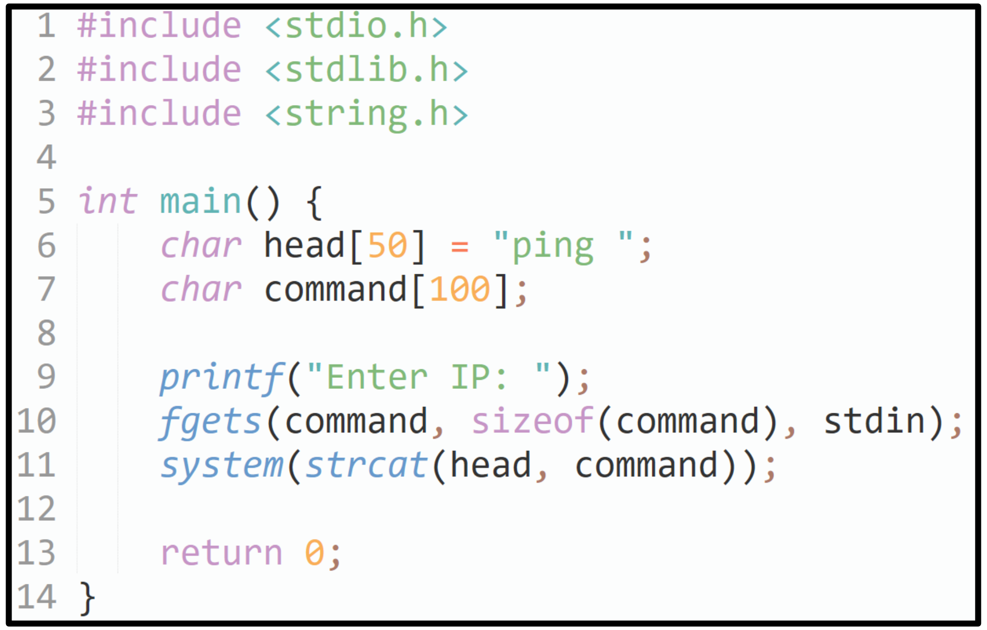
Command injection

- **Example**
  - A server that executes a ping command and returns the results
    - A user enters an IP address



Command injection

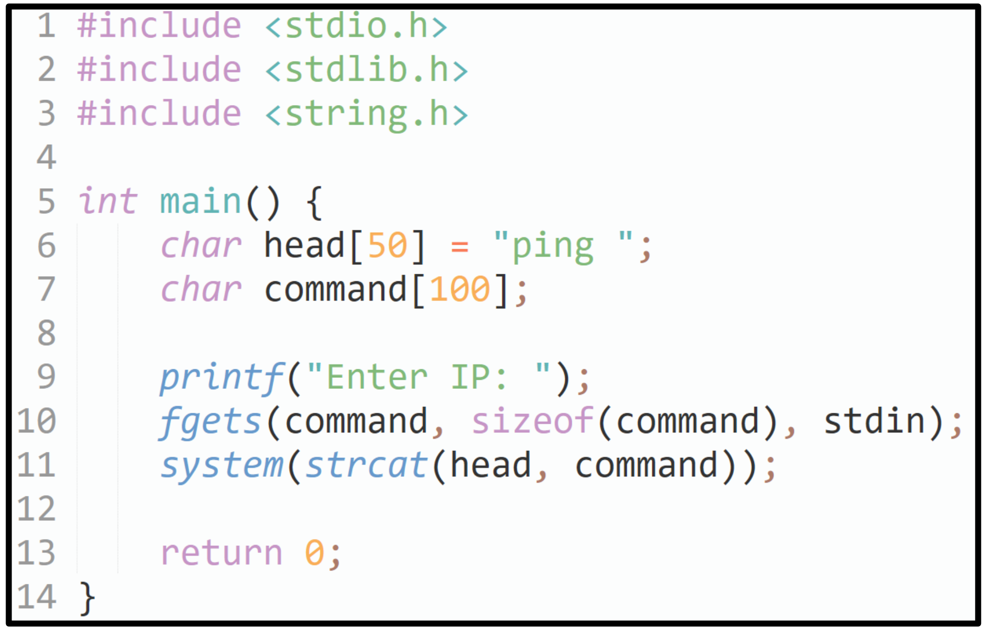
- **Example**
  - A server that executes a ping command and returns the results
    - A user enters an IP address



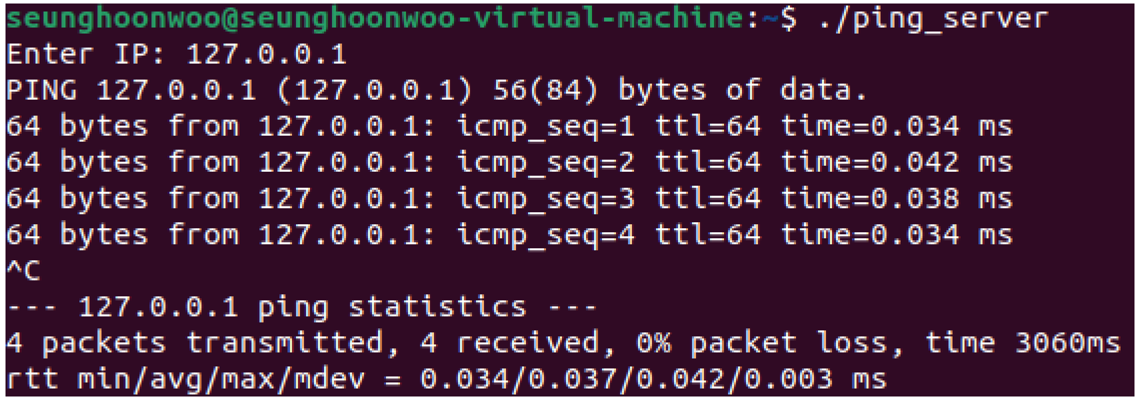
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char head[50] = "ping ";
    char command[100];

    printf("Enter IP: ");
    fgets(command, sizeof(command), stdin);
    system(strcat(head, command));

    return 0;
}
```

Command injection

- **Example**
  - A server that executes a ping command and returns the results
    - A user enters an IP address



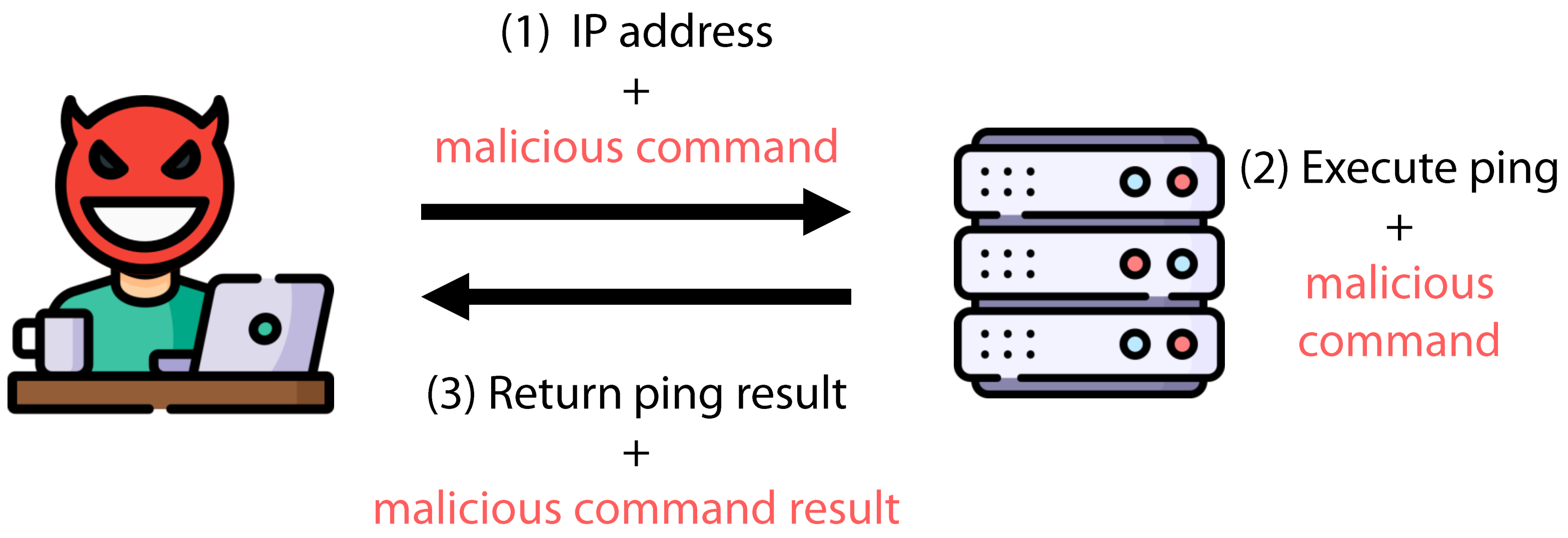
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    char head[50] = "ping ";
    char command[100];

    printf("Enter IP: ");
    fgets(command, sizeof(command), stdin);
    system(strcat(head, command));

    return 0;
}
```



```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./ping_server
Enter IP: 127.0.0.1
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.034 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.042 ms
64 bytes from 127.0.0.1: icmp_seq=3 ttl=64 time=0.038 ms
64 bytes from 127.0.0.1: icmp_seq=4 ttl=64 time=0.034 ms
^C
--- 127.0.0.1 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3060ms
rtt min/avg/max/mdev = 0.034/0.037/0.042/0.003 ms
```

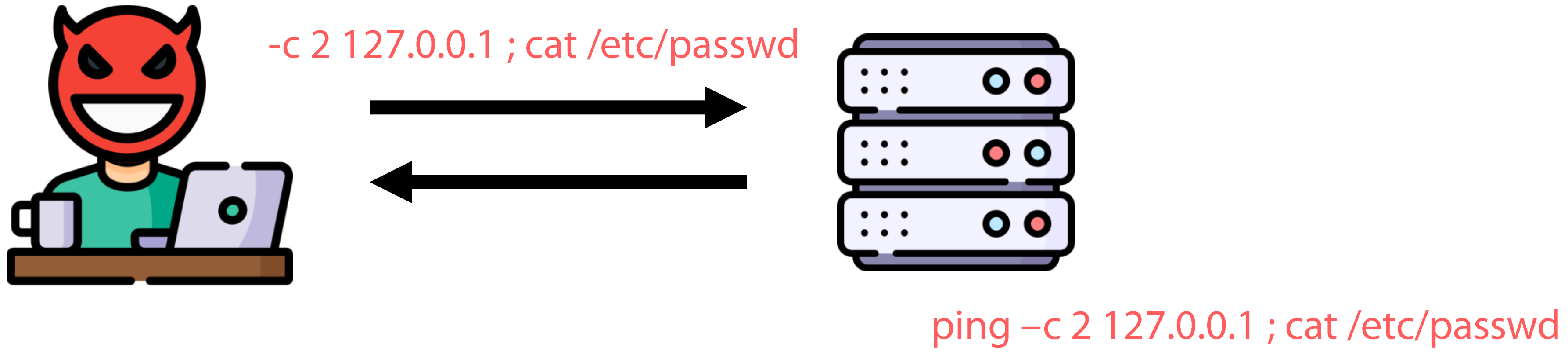
Command injection

- **Example**
  - Command injection



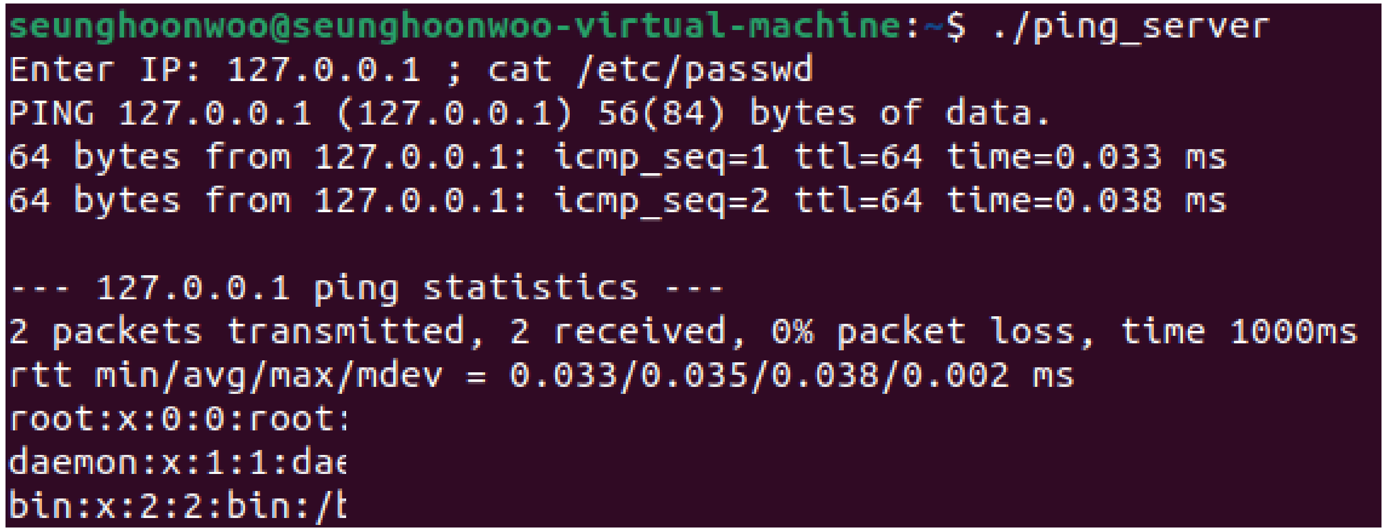
Command injection

- **Example**
  - Command injection


-c 2 127.0.0.1 ; cat /etc/passwd
ping –c 2 127.0.0.1 ; cat /etc/passwd

Command injection

- **Example**
  - Command injection



```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./ping_server
Enter IP: 127.0.0.1 ; cat /etc/passwd
PING 127.0.0.1 (127.0.0.1) 56(84) bytes of data.
64 bytes from 127.0.0.1: icmp_seq=1 ttl=64 time=0.033 ms
64 bytes from 127.0.0.1: icmp_seq=2 ttl=64 time=0.038 ms

--- 127.0.0.1 ping statistics ---
2 packets transmitted, 2 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.033/0.035/0.038/0.002 ms
root:x:0:0:root:
daemon:x:1:1:dae
bin:x:2:2:bin:/b
```

Command injection

- **How can we prevent command injection attacks?**
    1. Correct input validation (important)
    2. Limits system calls
    3. Principle of Least Privilege
        - Running a program with root privileges should be avoided as much as possible
        - Necessary permissions should only be granted when required

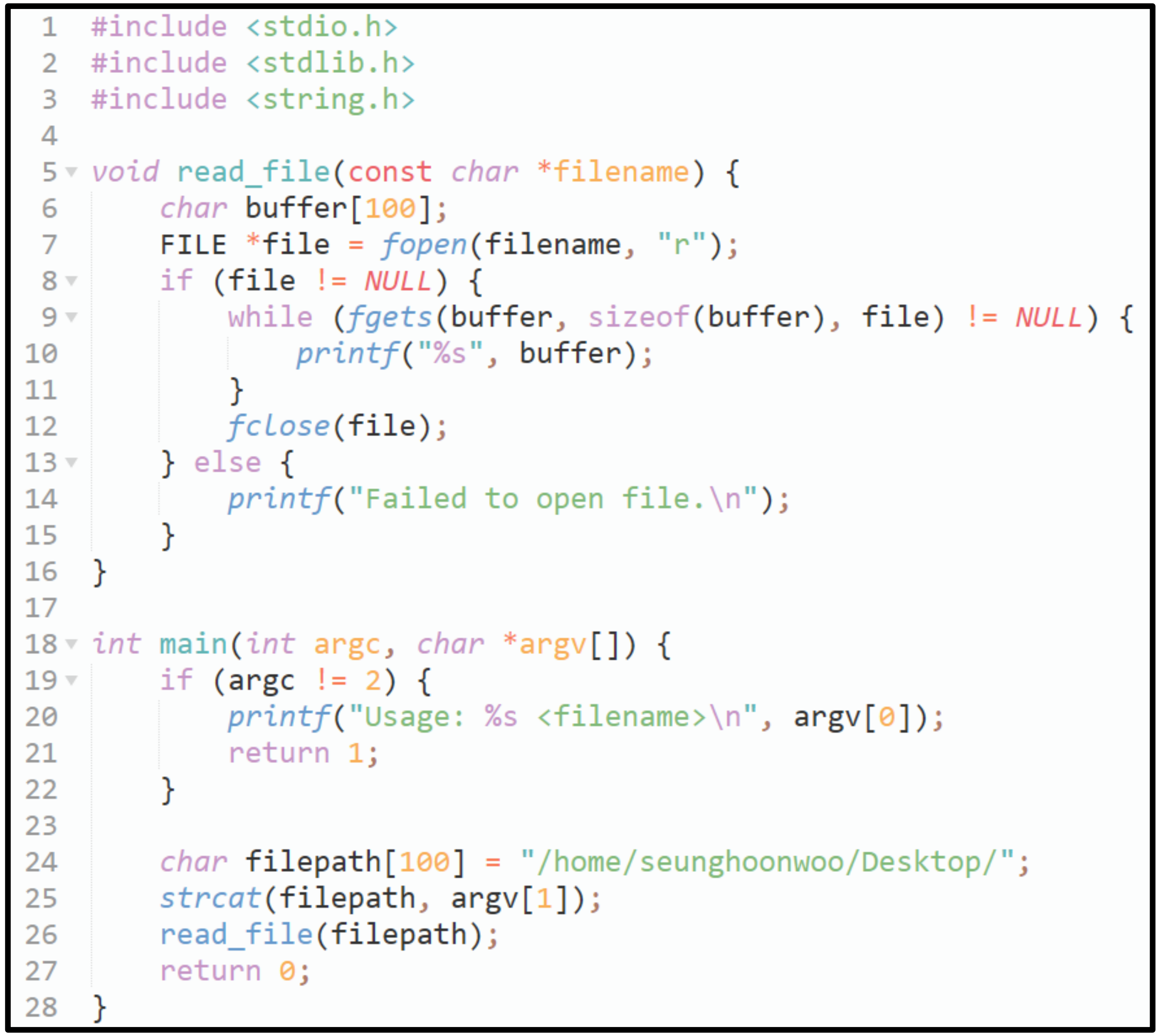
Path traversal

- **We can access the file system to read data from any file or write data to a file**
- **When exposing a service that accesses the local file system,**
  - Restrictions must be placed on the file paths that can be accessed

Path traversal

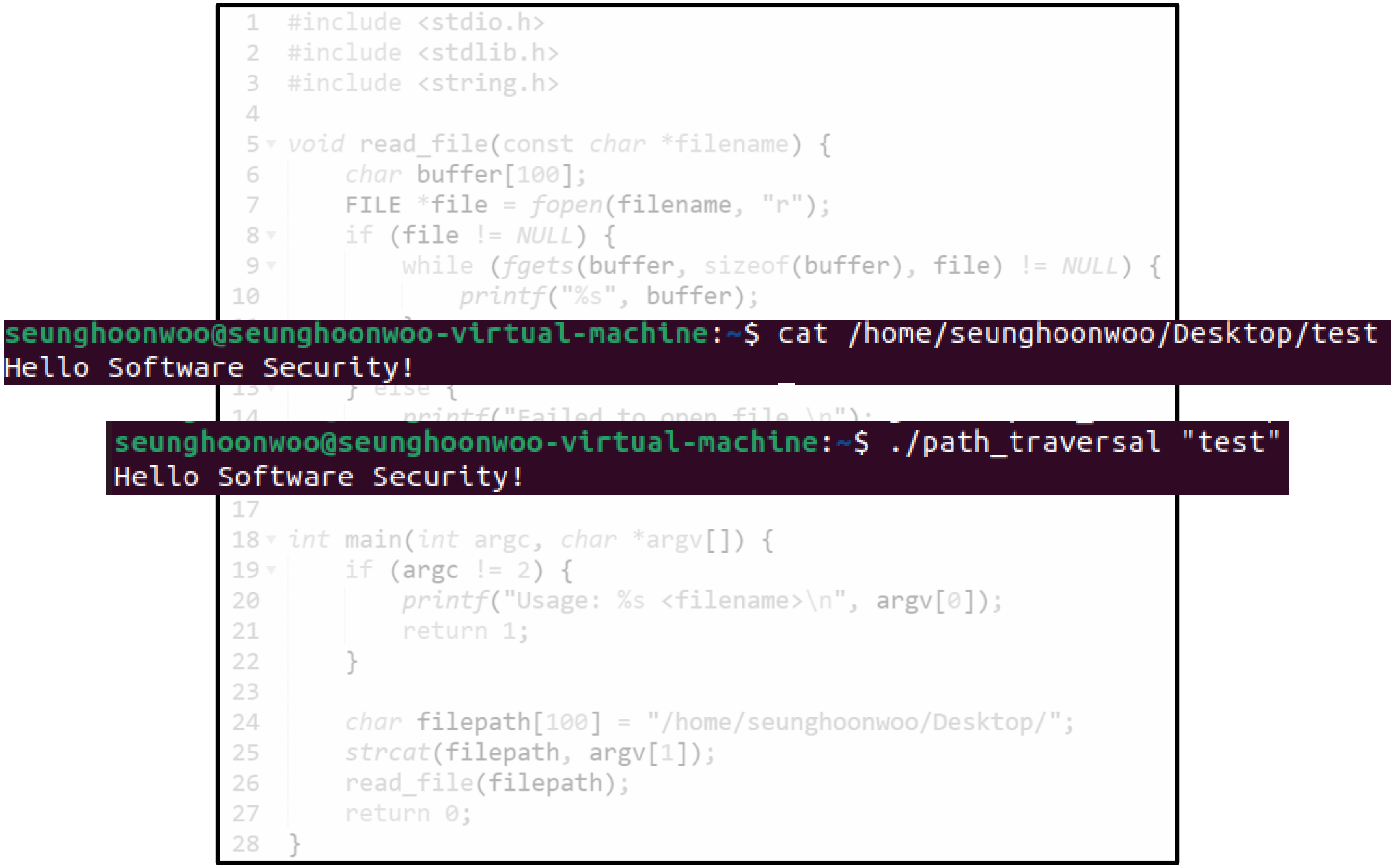
- **Two types of paths**
  - Absolute path
    - Connect all directory names from the root directory ('/') to the file
    - An absolute path is unique to that file
    - E.g., /home/seunghoonwoo/Desktop/target
  - Relative path
    - Path to another file relative to the current directory
    - E.g., ./target, ../target, ../../../target

Path traversal



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

void read_file(const char *filename) {
    char buffer[100];
    FILE *file = fopen(filename, "r");
    if (file != NULL) {
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
            printf("%s", buffer);
        }
        fclose(file);
    } else {
        printf("Failed to open file.\n");
    }
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    char filepath[100] = "/home/seunghoonwoo/Desktop/";
    strcat(filepath, argv[1]);
    read_file(filepath);
    return 0;
}
```

Path traversal

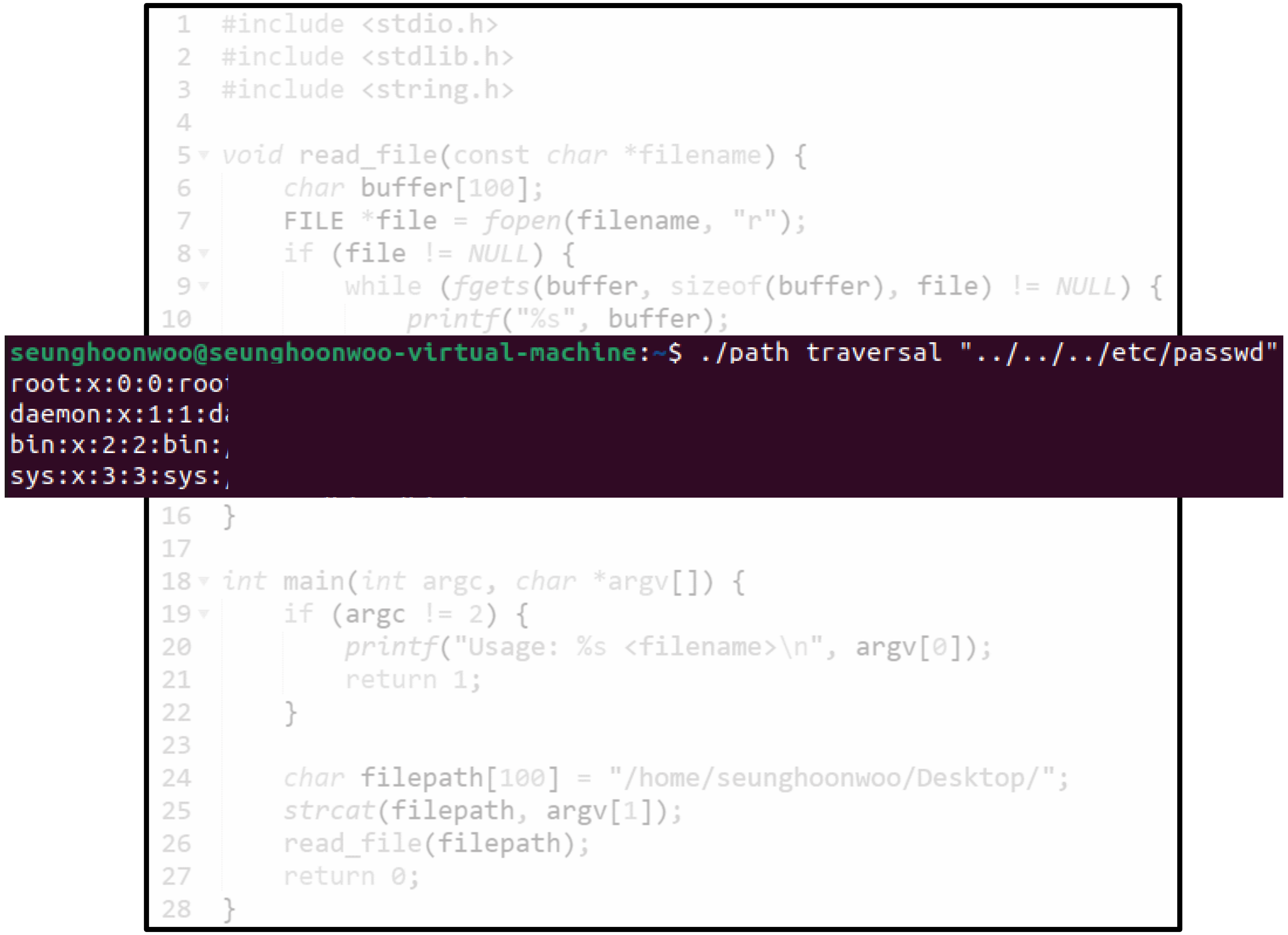


```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

void read_file(const char *filename) {
    char buffer[100];
    FILE *file = fopen(filename, "r");
    if (file != NULL) {
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
            printf("%s", buffer);
    } else {
        printf("Failed to open file.\n");

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    char filepath[100] = "/home/seunghoonwoo/Desktop/";
    strcat(filepath, argv[1]);
    read_file(filepath);
    return 0;
}
```

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ cat /home/seunghoonwoo/Desktop/test
Hello Software Security!
```

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./path_traversal "test"
Hello Software Security!
```

Path traversal



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

void read_file(const char *filename) {
    char buffer[100];
    FILE *file = fopen(filename, "r");
    if (file != NULL) {
        while (fgets(buffer, sizeof(buffer), file) != NULL) {
            printf("%s", buffer);
```

```
seunghoonwoo@seunghoonwoo-virtual-machine:~$ ./path traversal "../../../etc/passwd"
root:x:0:0:roo
daemon:x:1:1:d
bin:x:2:2:bin:
sys:x:3:3:sys:
```

```c
}

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }

    char filepath[100] = "/home/seunghoonwoo/Desktop/";
    strcat(filepath, argv[1]);
    read_file(filepath);
    return 0;
}
```

So what do security experts do?

From this slide, it won't appear on the exam

White (hat) hacker

- Also known as ethical hackers
- Use their skills to improve cybersecurity
- Typically work for organizations, governments, or security firms to identify vulnerabilities in systems, networks, and applications

Black (hat) hacker

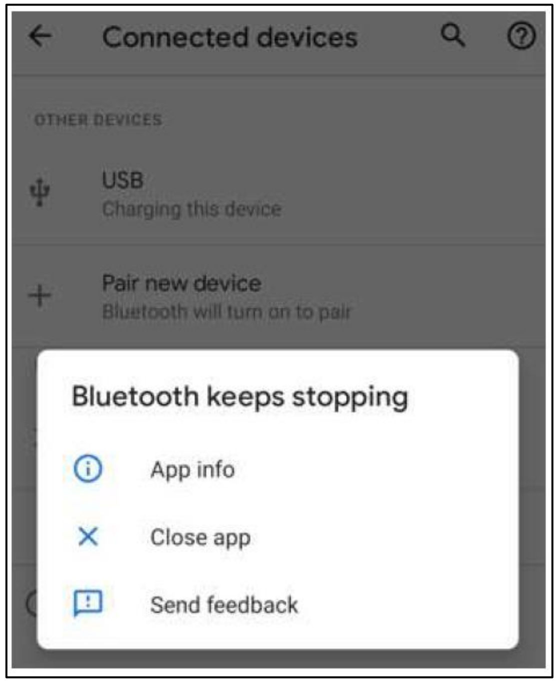
- Hacking for malicious purposes
- Intent to steal data, disrupt systems, or cause harm
- Exploit vulnerabilities in networks, websites, or software for personal gain, financial profit, or simply to cause chaos

So what do security experts do?

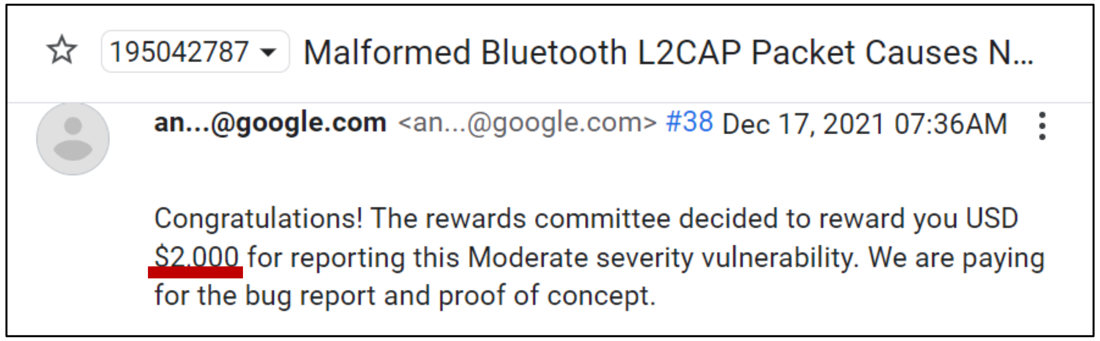
- **Bug bounty program**
  - Offers rewards for discovering security vulnerabilities in an organization's or company's systems or software (e.g., Google, Samsung, Naver, etc.)
  - Security experts with diverse expertise participate
  - Providing monetary rewards

So what do security experts do?

- **Bug bounty program**



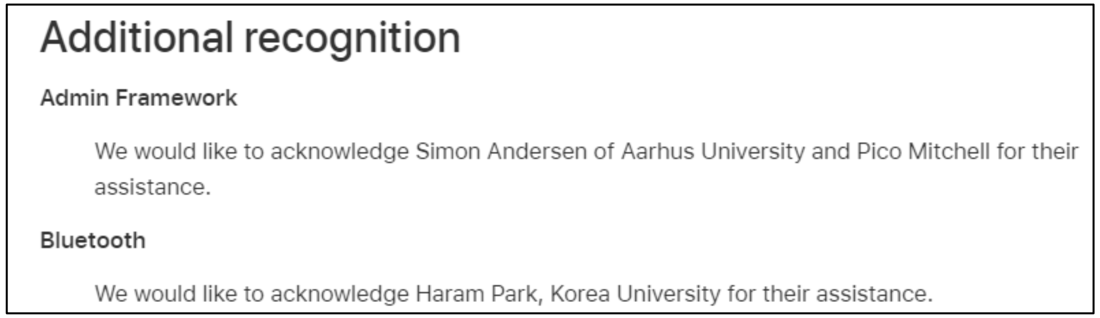
Bluetooth keeps stopping



195042787 Malformed Bluetooth L2CAP Packet Causes N...

an...@google.com <an...@google.com> #38 Dec 17, 2021 07:36AM

Congratulations! The rewards committee decided to reward you USD $2,000 for reporting this Moderate severity vulnerability. We are paying for the bug report and proof of concept.



Additional recognition

Admin Framework

We would like to acknowledge Simon Andersen of Aarhus University and Pico Mitchell for their assistance.

Bluetooth

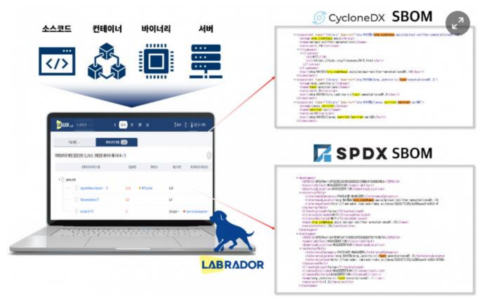
We would like to acknowledge Haram Park, Korea University for their assistance.

(tvOS, watchOS, iOS, iPadOS, macOS Monterey)

So what do security experts do?

- **Research on security**
  - Detecting vulnerabilities (in an automated way)
    - Fuzzing, static analysis, etc.
- **Establishing security policies**
- **Implementing security solutions**
  - E.g., Labrador labs (https://labradorlabs.ai/)
- **Penetration testing**
  - Ethical Hacking

래브라도랩스, 글로벌 1위 수술용 로봇 제조사에 SBOM 솔루션 공급



<데이터넷, 2023. 07. 27>
https://www.datanet.co.kr/news/articleView.html?idxno=185711

So what do security experts do?

- **Software Security & Privacy Lab**
  - https://ssp.korea.ac.kr/
  - Conducting various research on software security
    - Vulnerability discovery
    - Supply chain security
    - Open-source software security
    - Software composition analysis

오픈소스 소프트웨어 취약점 탐지 기술 V1SCAN 개발

박남수 기자 | 승인 2023.09.25 15:10 | 댓글 0

이희조-우승훈 교수팀
USENIX Security 2023에서 발표

우승훈 교수가 발표를 하고 있다

<정보통신신문, 2023. 09. 25
https://www.koit.co.kr/news/articleView.html?idxno=117106

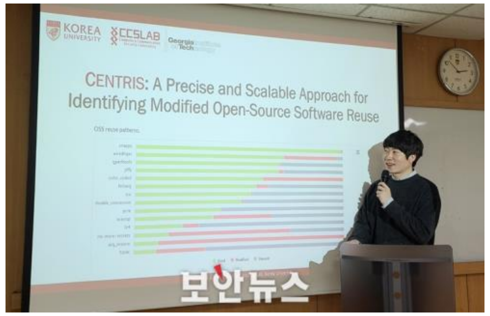
SW 취약점 '최초 근원지' 자동 탐색 기술 등장

| 고려대 이희조 교수팀 개발…'브이제로파인더'로 CVE 오기 96개 발견

브이제로파인더를 개발한 고려대 연구팀

<ZDNET Korea, 2021. 09. 08>
https://zdnet.co.kr/view/?no=20210908121238

오픈소스 커뮤니티 노리는 공급망 공격, 국내 연구팀 기술로 차단한다


KOREA UNIVERSITY
CCSLAB
Georgia Tech
CENTRIS: A Precise and Scalable Approach for Identifying Modified Open-Source Software Reuse
보안뉴스

<보안뉴스, 2021. 03. 25>
https://www.boannews.com/media/view.asp?idx=95883

전자신문 구독

이희조 고려대 교수팀 , 취약 코드 탐지 MOVERY 기술 개발

Korea University College of Informatics
CSSA
CCSLAB
MOVERY: A Precise Approach for Modified Vulnerable Code Clone Discovery from Modified Open-Source Software Components
Seunghoon Woo, Hyunji Hong, Eunjin Choi, Heejo Lee
Korea University
USENIX security 2022

<전자신문, 2022. 09. 26>
https://n.news.naver.com/mnews/article/030/0003046842?sid=105

So what do security experts do?

- **Misunderstanding..**
  - Security-related jobs have low salaries

So what do security experts do?

- **Misunderstanding..**
  - Security-related jobs have low salaries
    - Security graduate school colleagues I worked with
      - **Hxuxdxi 4+ colleagues**
      - **Sxmxuxg 3+ colleagues**
      - **Lx 2+ colleagues**
      - **Nxvxr 2+ colleagues**
      - **Sx hxnxx 1 colleague**
      - **Kxkxo bank 1 colleague**

열심히 하면 똑같이 대기업 갑니다..

Next Lecture

- **Practical exercises (4/1, 4/3)**
  - Actual vulnerability trigger practices
    - Main: stack and heap buffer overflow
    - Other attacks
  - You need to prepare
    - Laptop
    - Environment setup (refer to PPT on Blackboard)