NETWORK & MULTIMEDIA LAB

SECURE PROGRAMMING

Spring 2021

Outline

- What is a Secure Programming
- What is a Robust Program
- Fundamental Causes of Insecure Programs
  - Bad Implementations
  - Wrong Assumptions and Trusts
- Secure Coding Standards
  - CWE
  - CERT (C/C++)
  - MISRA (C/C++)
- Security Design Principles
  - 9 principles
- Hunting Vulnerabilities
  - Find Assumptions and Trusts
  - Find Threats
- Case Study
  - CVE-2021-3156:
    Heap-Based Buffer Overflow in Sudo
- Summary

What is a Secure Program

- A program without security vulnerabilities

- A security vulnerability is a weakness which can be exploited to perform unauthorized action (violate security policy)

- Security policies specify the resources that a process/user/system can access:
  - allowed to access a particular directory
  - not allowed to access any other files

What is a Robust Program

- A program that
  - Does not crash
    - Handles bad input and internal errors gracefully
    - On failure, provides information to aid in recovery or analysis
  - Does what it is supposed to

FUNDAMENTAL CAUSES OF INSECURE PROGRAMS

Bad Implementation

Wrong Assumptions and Trusts

Fundamental Causes of Insecure Programs

- Bad Implementation
  - 沒做檢查: Integer/Buffer/Heap overflow/underflow, XSS
  - 存取控制: Race condition, Use after free, Double free
  - 錯用函數: Format string attack, Broken cryptography

- Wrong Assumptions and Trusts
  - User input data
  - Data integrity
  - Authentication
  - Environment variables
  - Registry data
  - Reliability

相信 All is Well，不做檢查

Wrong Assumptions and Trusts

- What am I assuming/trusting?
  - OS, services, libraries, input, output, sensors, devices …
  - When you reuse/refactor code, use external libraries/modules/services, you inherit all their bugs and assumptions.
- What happens if my assumption/trust is wrong?
  - Can I detect it?
  - Should I continue?
- How to make assumption wrong?
  - When you move your program into another environment those assumptions may no longer hold.
  - Platform-dependent & Portability

SECURE CODING STANDARDS

Rules and guidelines used to prevent security vulnerabilities

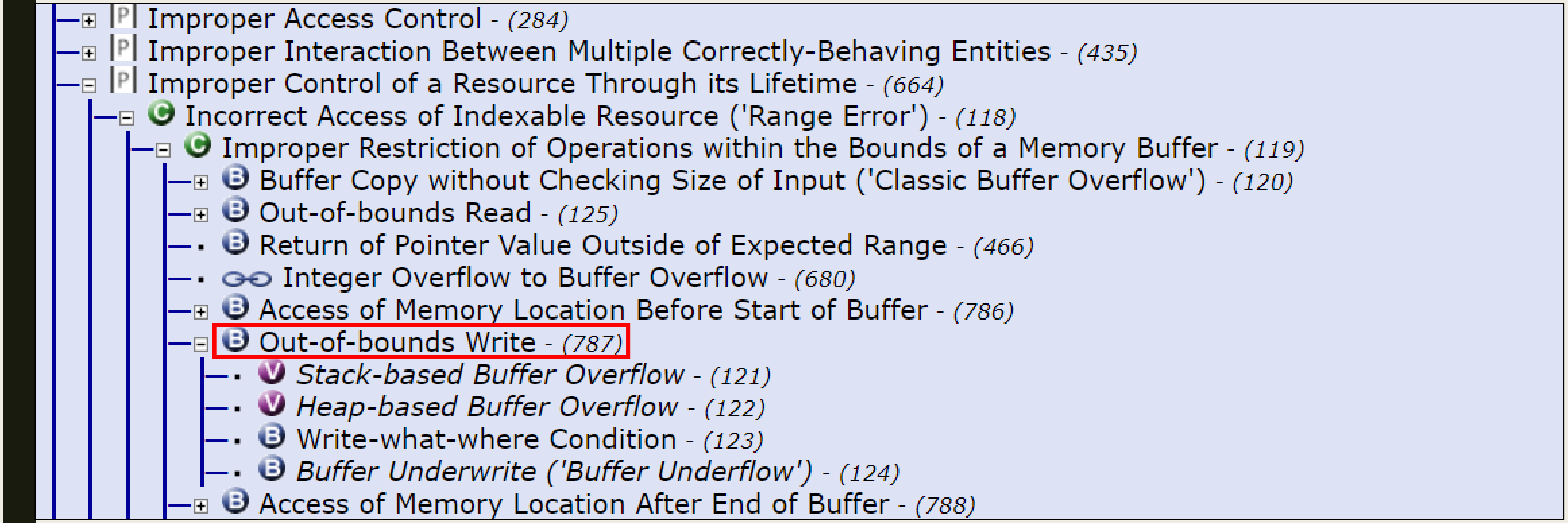
Secure Coding Standards

Rules and guidelines used to prevent security vulnerabilities

- Common Secure Coding Standards:
  - CWE
  - CERT (C/C++)
  - MISRA (C/C++)

CWE (Common Weakness Enumeration)

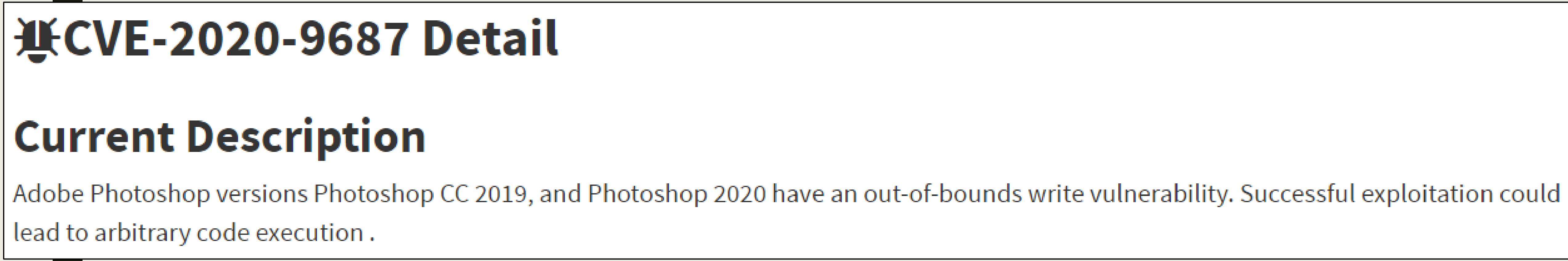
- 由 MITRE 所定義出來的弱點分類



- ⊞ P Improper Access Control - *(284)*
- ⊞ P Improper Interaction Between Multiple Correctly-Behaving Entities - *(435)*
- ⊟ P Improper Control of a Resource Through its Lifetime - *(664)*
  - ⊟ C Incorrect Access of Indexable Resource ('Range Error') - *(118)*
    - ⊟ C Improper Restriction of Operations within the Bounds of a Memory Buffer - *(119)*
      - ⊞ B Buffer Copy without Checking Size of Input ('Classic Buffer Overflow') - *(120)*
      - ⊞ B Out-of-bounds Read - *(125)*
      - B Return of Pointer Value Outside of Expected Range - *(466)*
      - Integer Overflow to Buffer Overflow - *(680)*
      - ⊞ B Access of Memory Location Before Start of Buffer - *(786)*
      - ⊟ B Out-of-bounds Write - *(787)*
        - V *Stack-based Buffer Overflow* - *(121)*
        - V *Heap-based Buffer Overflow* - *(122)*
        - B Write-what-where Condition - *(123)*
        - B *Buffer Underwrite ('Buffer Underflow')* - *(124)*
      - ⊞ B Access of Memory Location After End of Buffer - *(788)*

CVE (Common Vulnerabilities and Exposures)

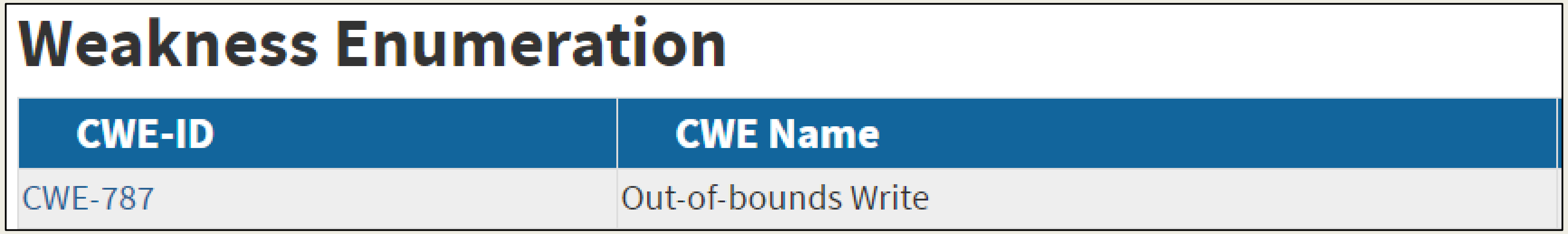
- 由 MITRE 維護的漏洞資料庫



CVE-2020-9687 Detail

Current Description

Adobe Photoshop versions Photoshop CC 2019, and Photoshop 2020 have an out-of-bounds write vulnerability. Successful exploitation could lead to arbitrary code execution .

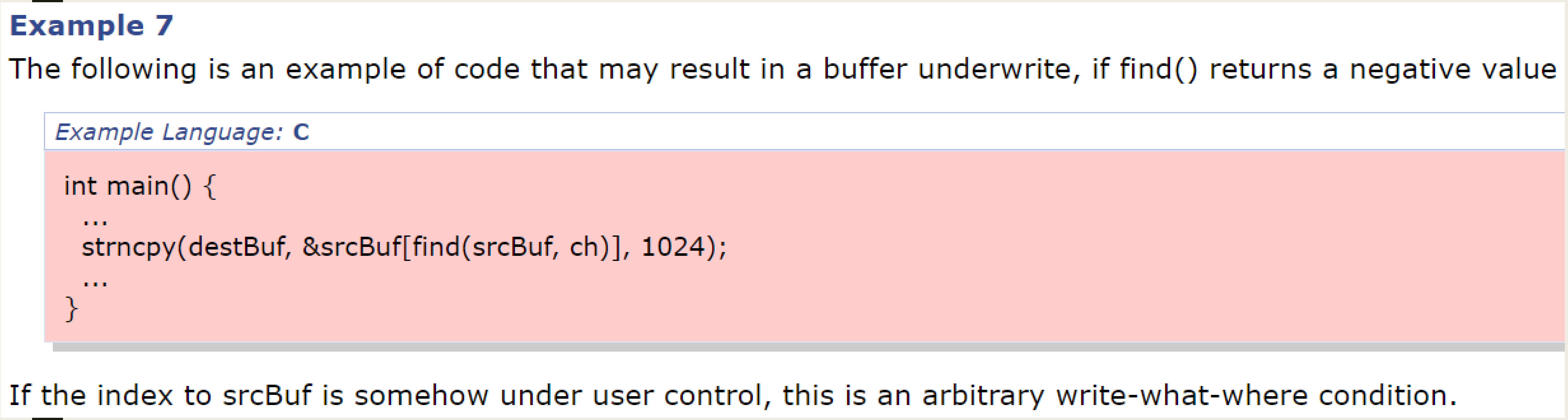


Weakness Enumeration

CWE-ID	CWE Name
CWE-787	Out-of-bounds Write

CWE (Common Weakness Enumeration)

- CWE-787: Out-of-bounds Write



Example 7

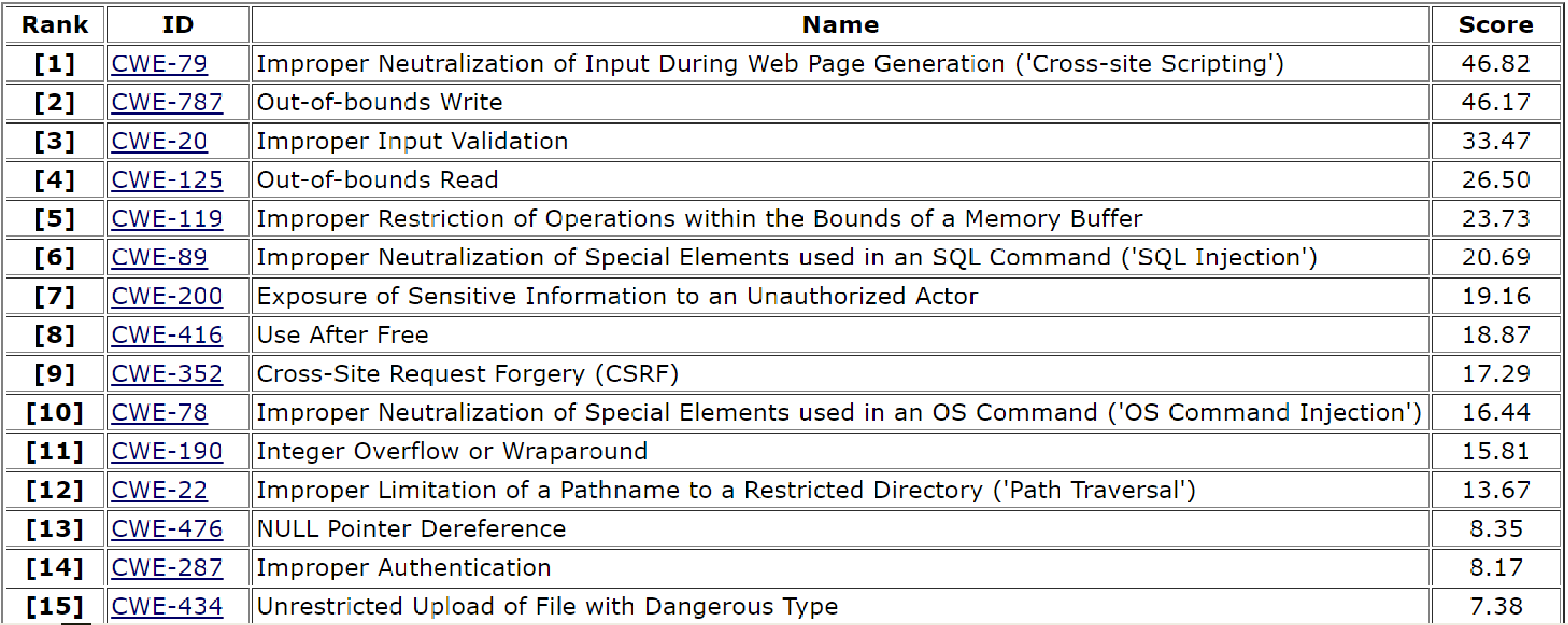
The following is an example of code that may result in a buffer underwrite, if find() returns a negative value

Example Language: **C**

```
int main() {
  ...
  strncpy(destBuf, &srcBuf[find(srcBuf, ch)], 1024);
  ...
}
```

If the index to srcBuf is somehow under user control, this is an arbitrary write-what-where condition.

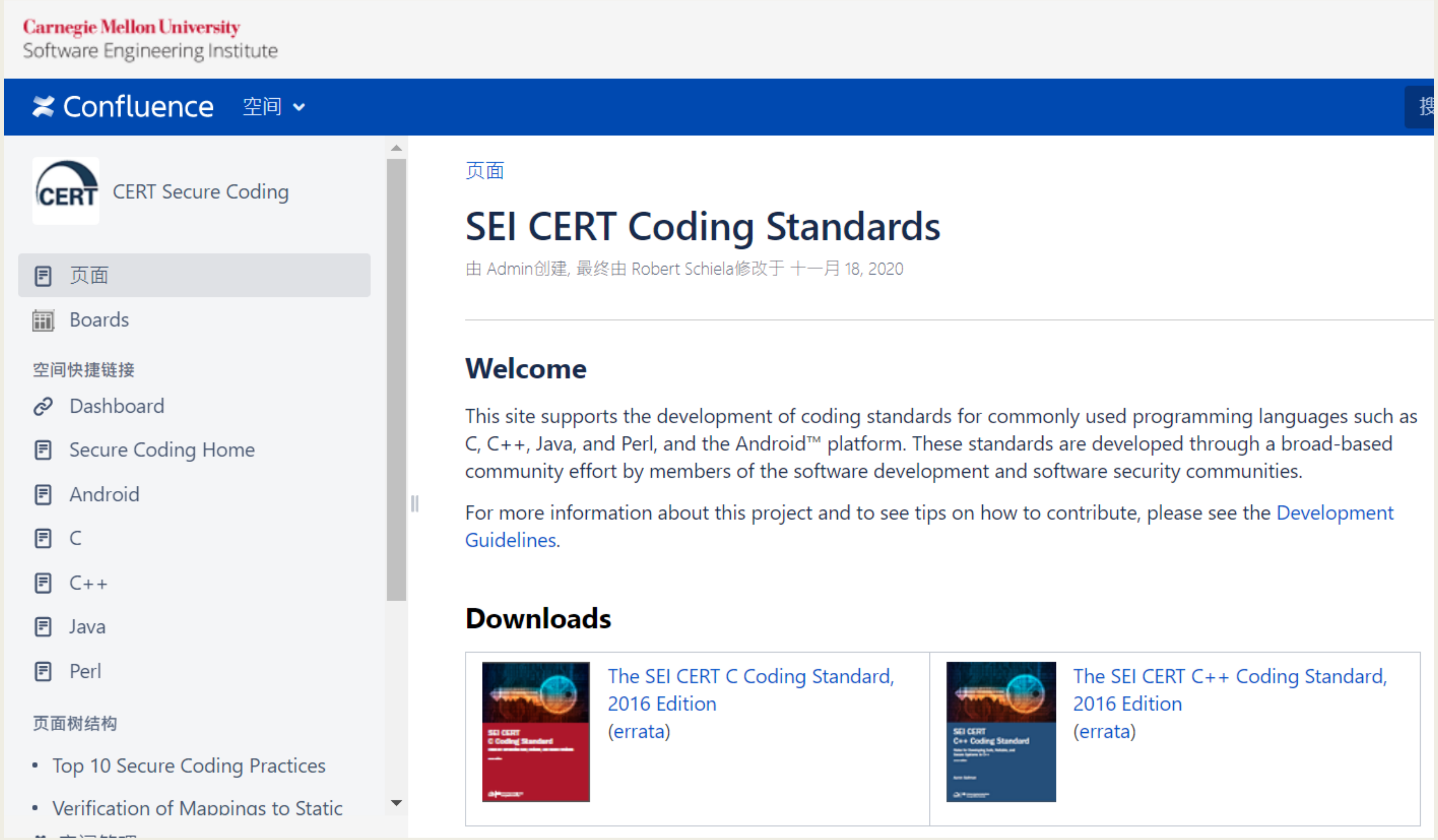
2020 CWE Top 25



Rank	ID	Name	Score
[1]	CWE-79	Improper Neutralization of Input During Web Page Generation ('Cross-site Scripting')	46.82
[2]	CWE-787	Out-of-bounds Write	46.17
[3]	CWE-20	Improper Input Validation	33.47
[4]	CWE-125	Out-of-bounds Read	26.50
[5]	CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	23.73
[6]	CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	20.69
[7]	CWE-200	Exposure of Sensitive Information to an Unauthorized Actor	19.16
[8]	CWE-416	Use After Free	18.87
[9]	CWE-352	Cross-Site Request Forgery (CSRF)	17.29
[10]	CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	16.44
[11]	CWE-190	Integer Overflow or Wraparound	15.81
[12]	CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	13.67
[13]	CWE-476	NULL Pointer Dereference	8.35
[14]	CWE-287	Improper Authentication	8.17
[15]	CWE-434	Unrestricted Upload of File with Dangerous Type	7.38

https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html

CERT (Computer Emergency Response Team)



Carnegie Mellon University
Software Engineering Institute

页面

SEI CERT Coding Standards

由 Admin创建, 最终由 Robert Schiela修改于 十一月 18, 2020

Welcome

This site supports the development of coding standards for commonly used programming languages such as C, C++, Java, and Perl, and the Android™ platform. These standards are developed through a broad-based community effort by members of the software development and software security communities.

For more information about this project and to see tips on how to contribute, please see the Development Guidelines.

Downloads

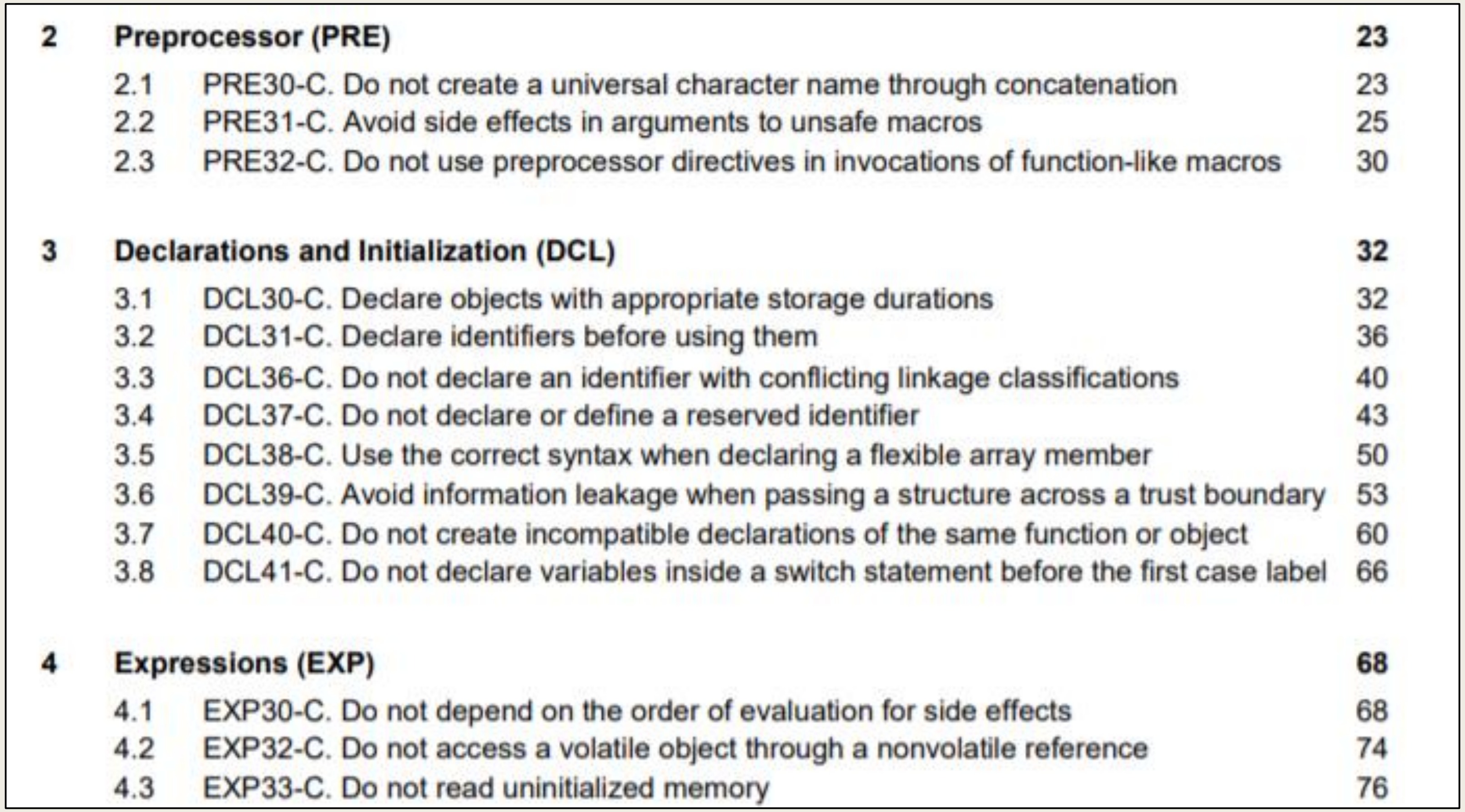
The SEI CERT C Coding Standard, 2016 Edition
(errata)

The SEI CERT C++ Coding Standard, 2016 Edition
(errata)

https://wiki.sei.cmu.edu/confluence/display/seccode/SEI+CERT+Coding+Standards

SEI CERT C Coding Standard (2016 Edition)

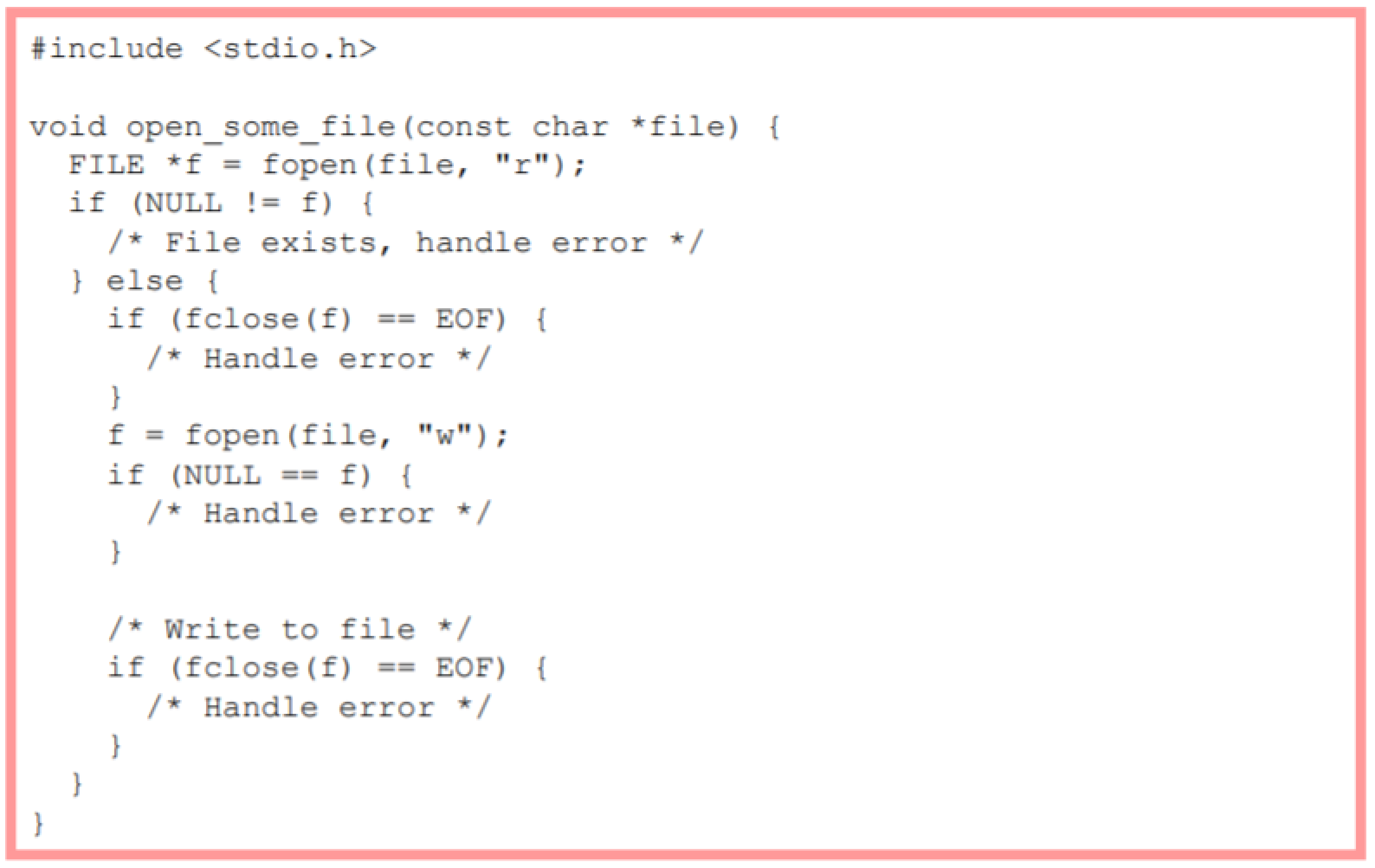
1. 預處理 (PRE)
2. 聲明和初始化 (DCL)
3. 表達式 (EXP)
4. 整數 (INT)
5. 浮點數 (FLP)
6. 數組 (ARR)
7. 字符(數組)和字符串 (STR)
8. 內存管理 (MEM)
9. 輸入輸出 (FIO)
10. 環境 (ENV)
11. 信號 (SIG)
12. 錯誤處理 (ERR)
13. 並行性 (CON)
14. 雜項 (MSC)



SEI CERT C Coding Standard (2016 Edition)

FIO45-C. Avoid TOCTOU race conditions while accessing files

- Time-of-check, time-of-use (TOCTOU)

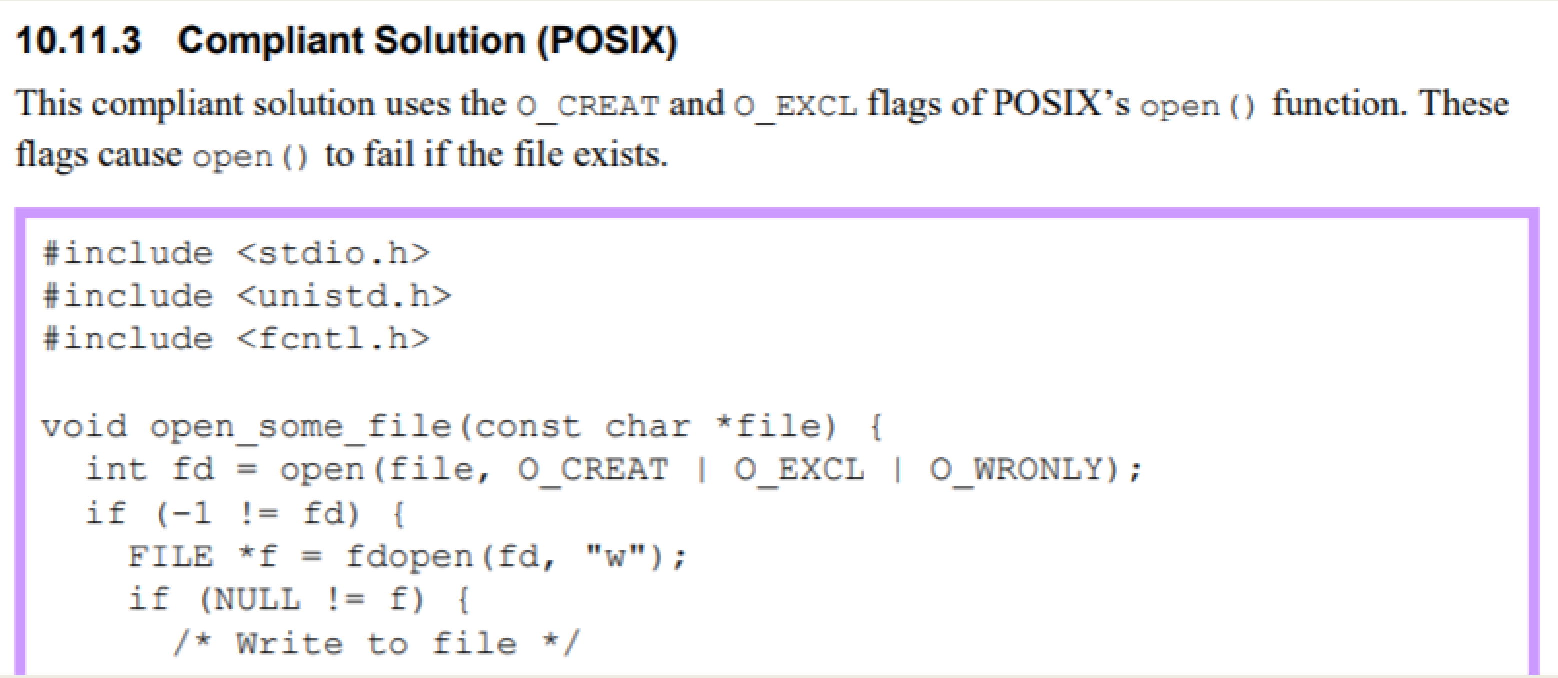


```
#include <stdio.h>

void open_some_file(const char *file) {
  FILE *f = fopen(file, "r");
  if (NULL != f) {
    /* File exists, handle error */
  } else {
    if (fclose(f) == EOF) {
      /* Handle error */
    }
    f = fopen(file, "w");
    if (NULL == f) {
      /* Handle error */
    }

    /* Write to file */
    if (fclose(f) == EOF) {
      /* Handle error */
    }
  }
}
```

SEI CERT C Coding Standard (2016 Edition)

FIO45-C. Avoid TOCTOU race conditions while accessing files

- Time-of-check, time-of-use (TOCTOU)



10.11.3 Compliant Solution (POSIX)

This compliant solution uses the `O_CREAT` and `O_EXCL` flags of POSIX's `open()` function. These flags cause `open()` to fail if the file exists.

```
#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>

void open_some_file(const char *file) {
  int fd = open(file, O_CREAT | O_EXCL | O_WRONLY);
  if (-1 != fd) {
    FILE *f = fdopen(fd, "w");
    if (NULL != f) {
      /* Write to file */
```

Macro: int O_CREAT

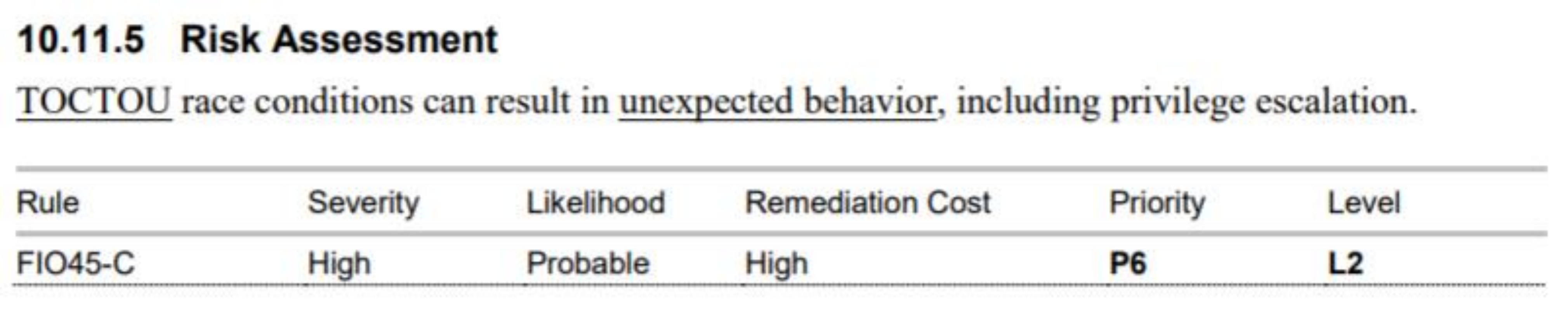
- If set, the file will be created if it doesn't already exist.

Macro: int O_EXCL

- If both O_CREAT and O_EXCL are set, then open fails if the specified file already exists. This is guaranteed to never clobber an existing file.

SEI CERT C Coding Standard (2016 Edition)

FIO45-C. Avoid TOCTOU race conditions while accessing files



10.11.5 Risk Assessment

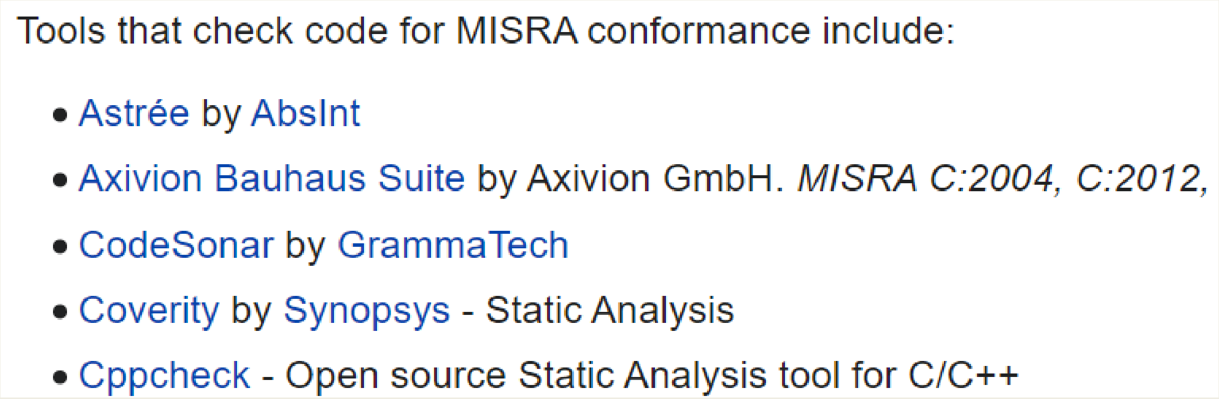
TOCTOU race conditions can result in unexpected behavior, including privilege escalation.

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
FIO45-C	High	Probable	High	**P6**	**L2**

MISRA (Motor Industry Software Reliability Association，汽車工業軟體可靠性協會)

MISRA 提出的 C/C++ 語言開發標準:

- MISRA C:1998
- MISRA C:2004
- MISRA C++:2008
- MISRA C:2012
- MISRA Compliance:2016
- MISRA Compliance:2020



Tools that check code for MISRA conformance include:

- Astrée by AbsInt
- Axivion Bauhaus Suite by Axivion GmbH. *MISRA C:2004, C:2012,*
- CodeSonar by GrammaTech
- Coverity by Synopsys - Static Analysis
- Cppcheck - Open source Static Analysis tool for C/C++

[Tools that check code for MISRA conformance]

SECURITY DESIGN PRINCIPLES

9 條安全設計原則

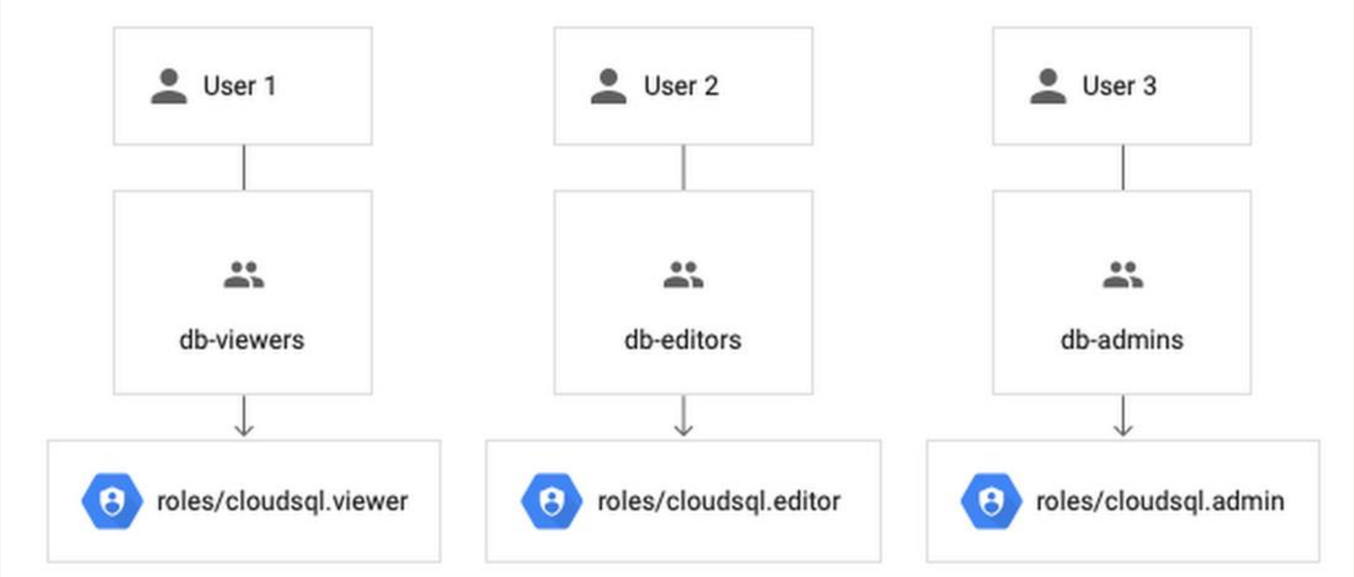
Security Design Principles

1. Least Privilege
2. Establish Secure Defaults
3. Keep it simple
4. Complete Mediation
5. Open Design
6. Defense in Depth
7. Least Common Mechanism
8. Least Astonishment
9. Minimize Attack Surface

1. Least Privilege

A subject should be given the minimum set of privileges required to perform its task

- Privileges should be time based
  - Rights added as needed, discarded after use
- Assign roles to groups, not individuals
  - Place users into logical groups is a safer and more maintainable option



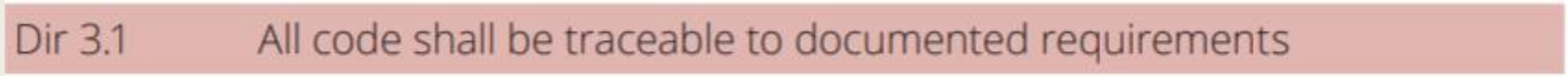
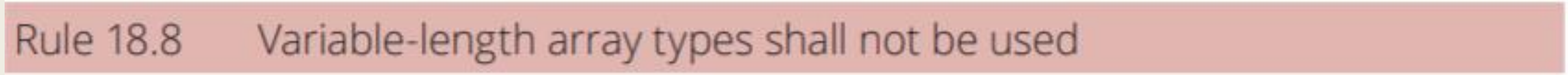
2. Establish Secure Defaults

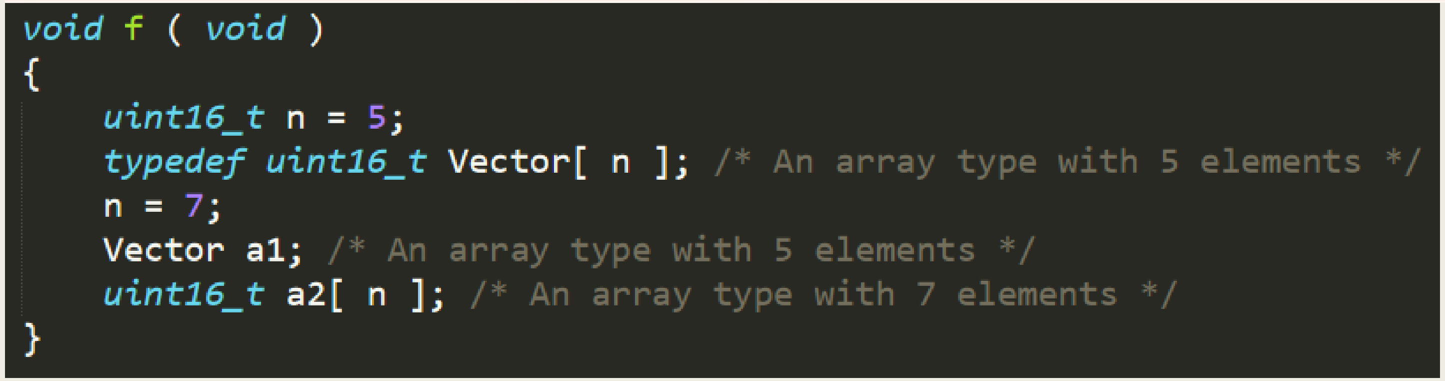
The application must be secure by default

- Secure default password
- Security features should be set to a high-security level by default
- Subjects do not have access to any resources by default
  - Default Deny
  - Whitelisting instead of Blacklisting

3. Keep it simple and clear

Keep it as simple/clear as possible

- Less thing can go wrong
- When errors occur, they are easier to understand and fix
- MISRA C : 
  Dir 3.1 All code shall be traceable to documented requirements
- MISRA C : 
  Rule 18.8 Variable-length array types shall not be used



```
void f ( void )
{
    uint16_t n = 5;
    typedef uint16_t Vector[ n ]; /* An array type with 5 elements */
    n = 7;
    Vector a1; /* An array type with 5 elements */
    uint16_t a2[ n ]; /* An array type with 7 elements */
}
```

- A security mechanism should be easy to use
  - Ease of installation, configuration

4. Complete Mediation

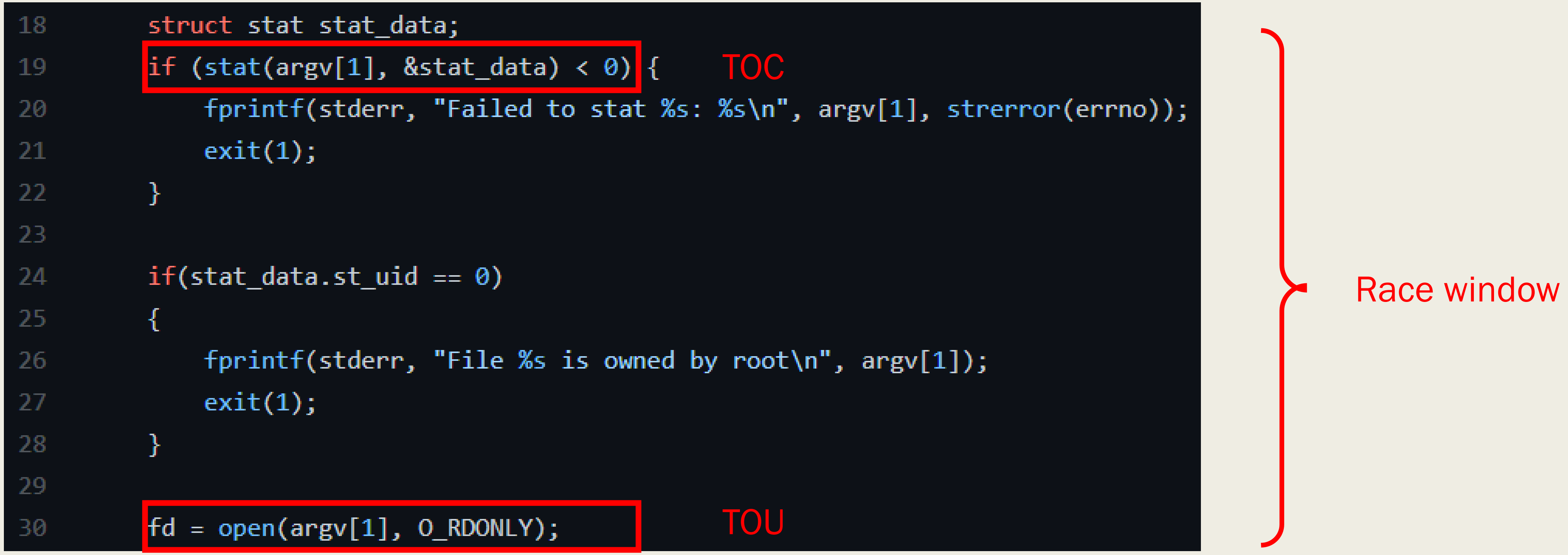
Every access to every resource must be validated for authorization

- Time-of-check, time-of-use (TOCTOU)
  - File Path Race Condition
  - DNS Rebinding
  - UNIX: access checked on open, not checked thereafter, if permissions change after, may get unauthorized access

4. Complete Mediation

File Path Race Condition

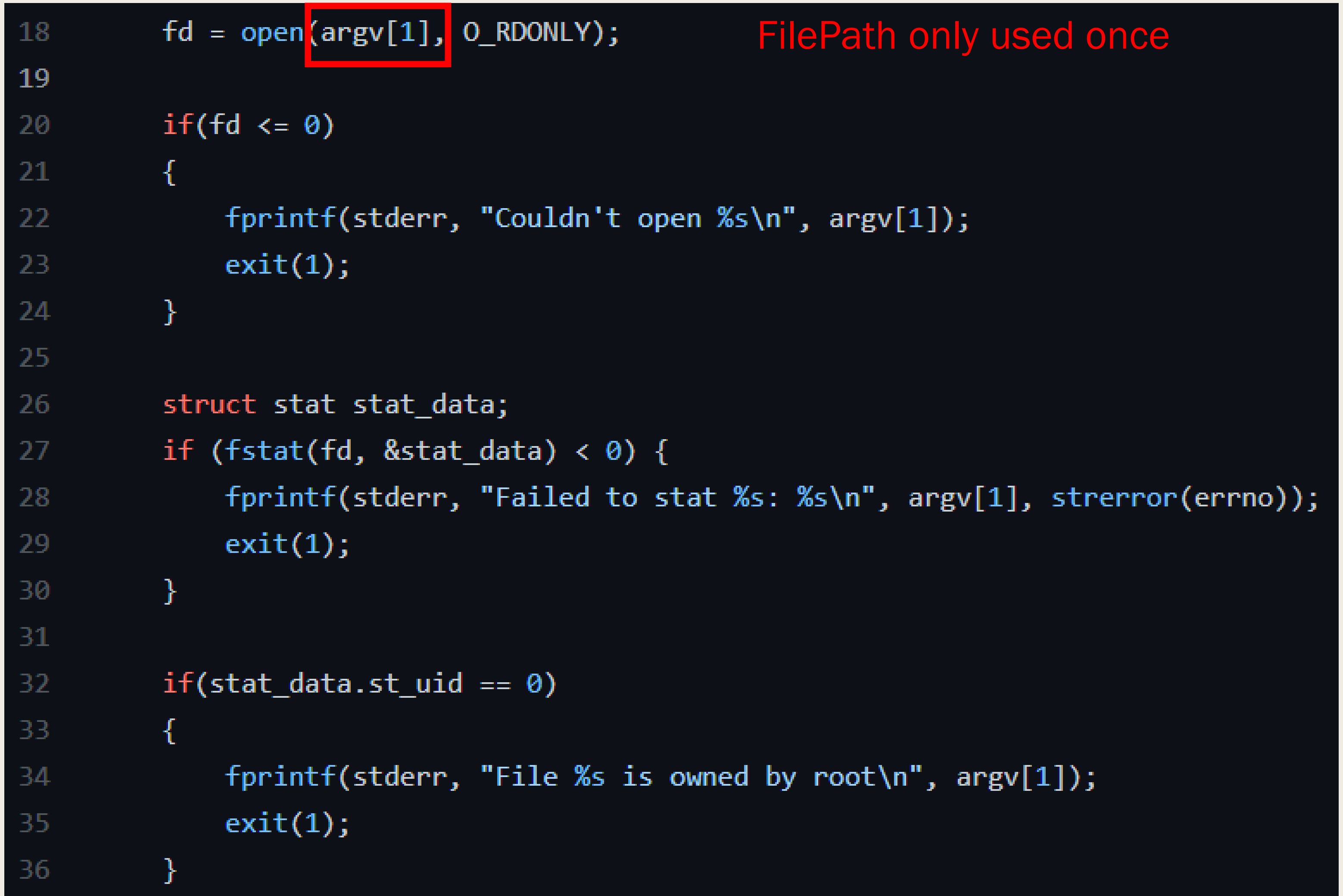
- Read the file if it is not owned by root:



```
    struct stat stat_data;
    if (stat(argv[1], &stat_data) < 0) {
        fprintf(stderr, "Failed to stat %s: %s\n", argv[1], strerror(errno));
        exit(1);
    }

    if(stat_data.st_uid == 0)
    {
        fprintf(stderr, "File %s is owned by root\n", argv[1]);
        exit(1);
    }

    fd = open(argv[1], O_RDONLY);
```

4. Complete Mediation

File Path Race Condition



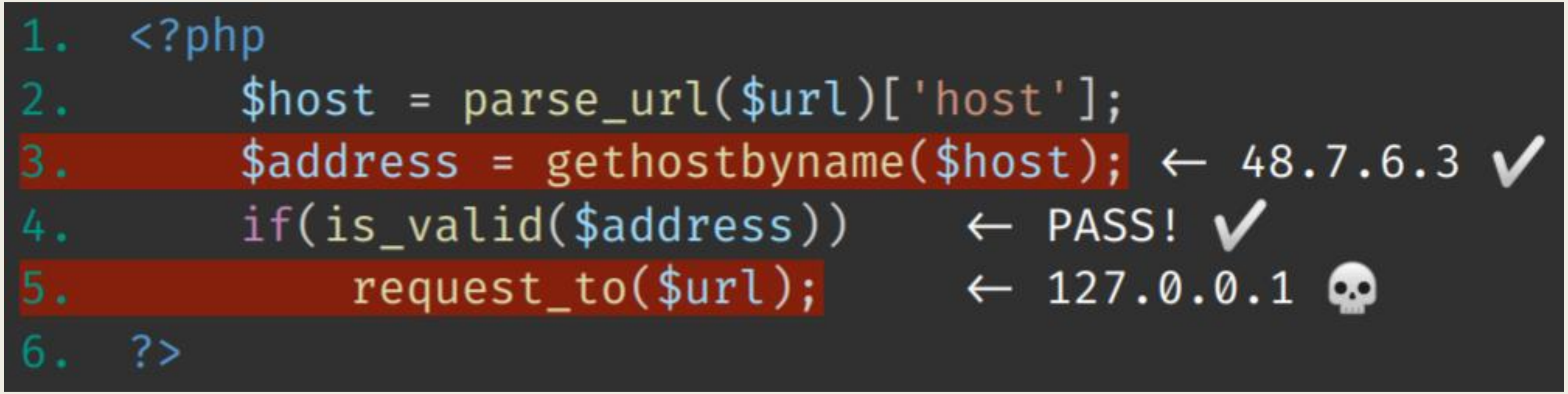
```
    fd = open(argv[1], O_RDONLY);

    if(fd <= 0)
    {
        fprintf(stderr, "Couldn't open %s\n", argv[1]);
        exit(1);
    }

    struct stat stat_data;
    if (fstat(fd, &stat_data) < 0) {
        fprintf(stderr, "Failed to stat %s: %s\n", argv[1], strerror(errno));
        exit(1);
    }

    if(stat_data.st_uid == 0)
    {
        fprintf(stderr, "File %s is owned by root\n", argv[1]);
        exit(1);
    }
```

FilePath only used once

Reference:
File Path Race Condition & How To Prevent It

4. Complete Mediation

DNS Rebinding

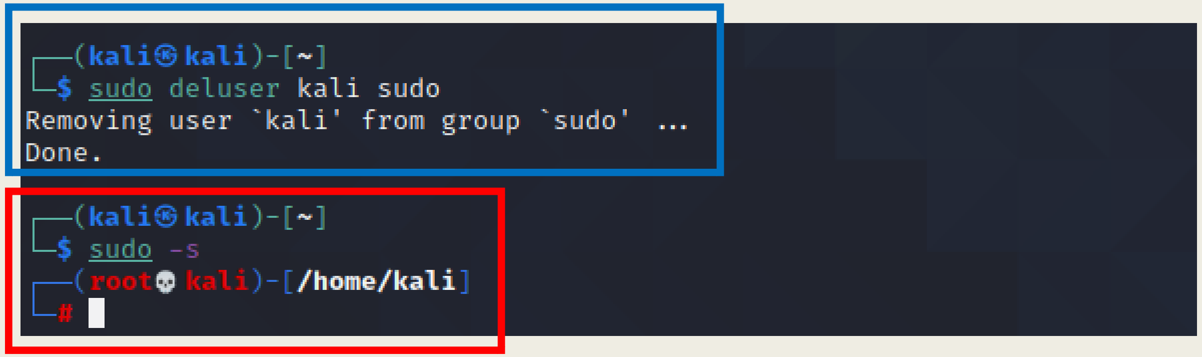


```
<?php
    $host = parse_url($url)['host'];
    $address = gethostbyname($host); ← 48.7.6.3 ✓
    if(is_valid($address))    ← PASS! ✓
        request_to($url);     ← 127.0.0.1 💀
?>
```

4. Complete Mediation

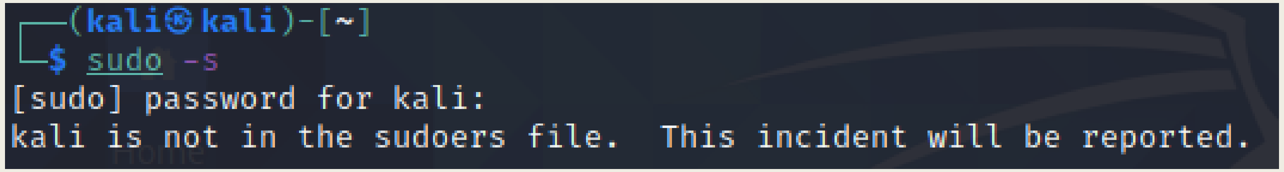
UNIX

- If permissions change after, may get unauthorized access



```
┌──(kali㉿kali)-[~]
└─$ sudo deluser kali sudo
Removing user `kali' from group `sudo' ...
Done.

┌──(kali㉿kali)-[~]
└─$ sudo -s
┌──(root💀kali)-[/home/kali]
└─# 
```

- Start a new terminal can still get unauthorized access
- The permission will be checked after re-login



```
┌──(kali㉿kali)-[~]
└─$ sudo -s
[sudo] password for kali:
kali is not in the sudoers file.  This incident will be reported.
```

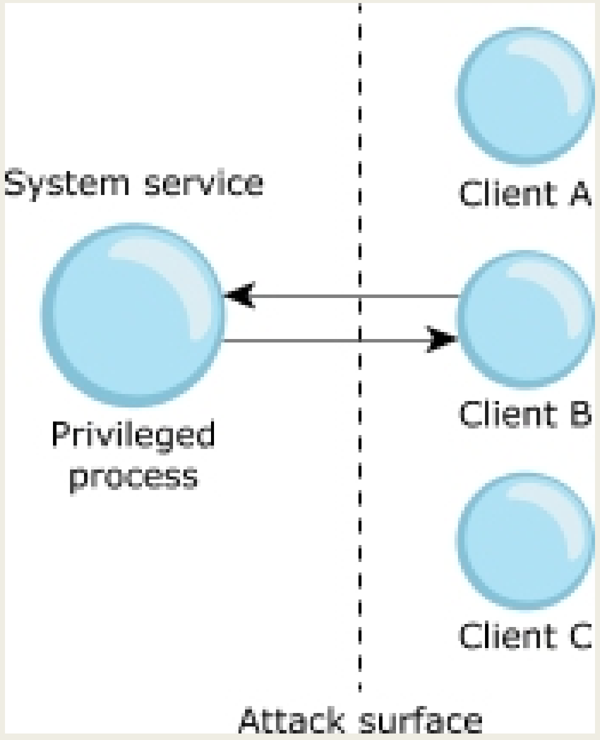
5. Defense in Depth

- Layering defensive mechanisms in a system to reduce
  - the chance of attacks
  - the damage caused by attacks

- Requires multiple conditions to grant privilege/access
  - Separation of Privilege
  - Separation of duty
  - Multi-factor authentication
  - Secrets Management

Separation of Privilege

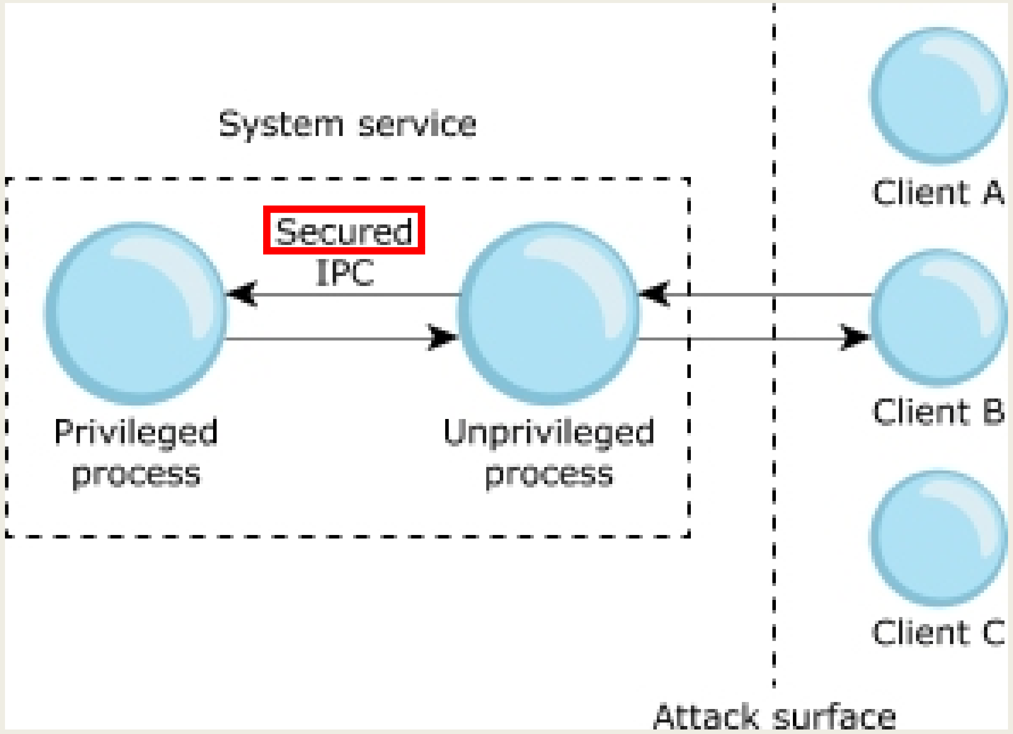
In many applications, services, and drivers, a part of the program will require some amount of elevated privileges to carry out its job.



System service with no privilege separation.

Separation of Privilege

In many applications, services, and drivers, a part of the program will require some amount of elevated privileges to carry out its job.



System service with privilege separation.
IPC (Inter-Process Communication)

Unprivileged process can be compromised, privileged process still need to validate the input to prevent SSRF (Server-side request forgery).

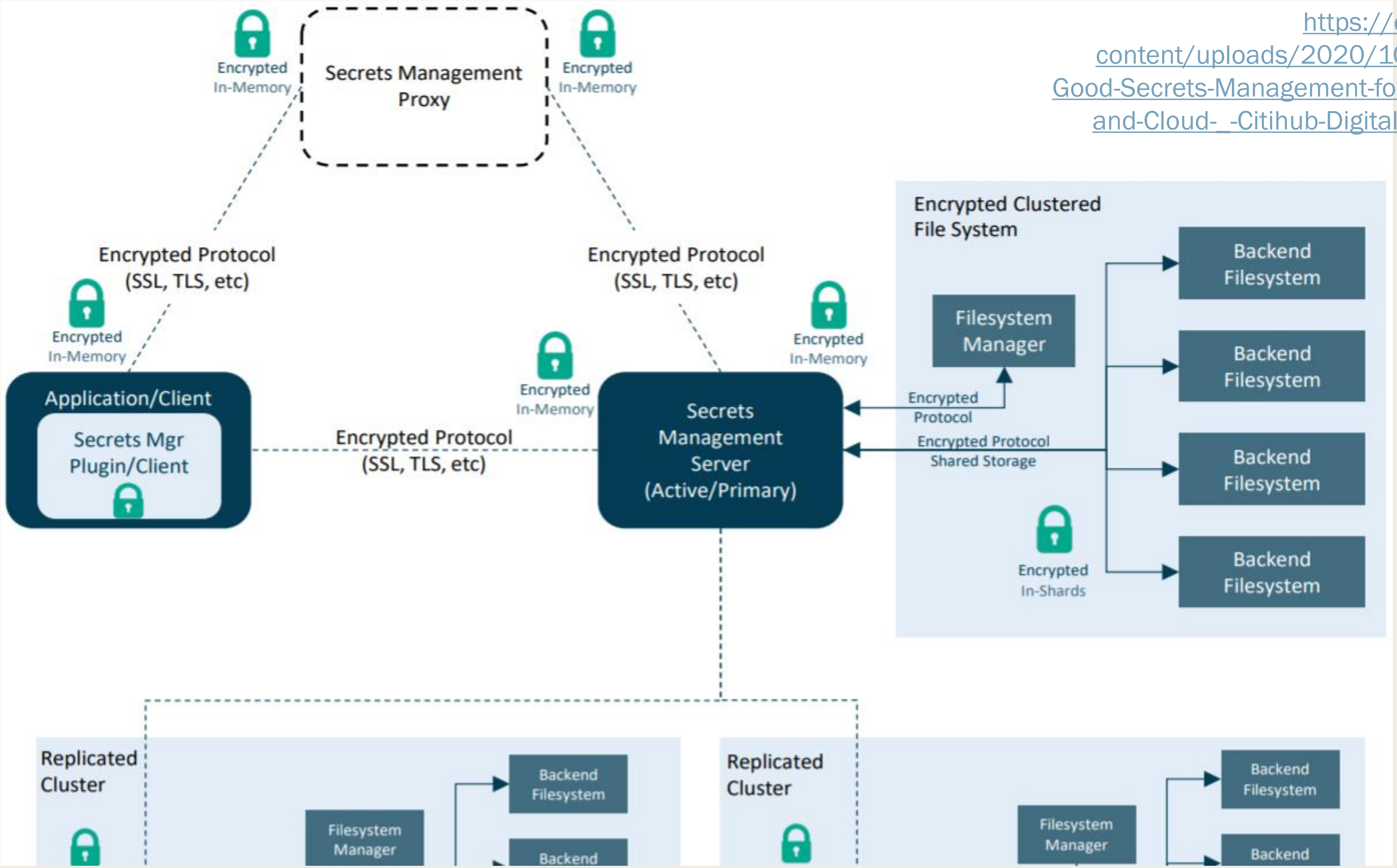
RCE: Attacker get unprivileged account.
SSRF: Will fail if the attack payload is successfully filtered.

Secrets Management

- Secrets may include:
  - API keys
  - Encryption keys
  - Passwords
  - Database credentials
  - Sensitive configuration settings (email address, usernames, debug flags, etc.)
- Secrets may be stored in:
  - Application file system
  - Application database
  - Environment variables
  - Source code management system
  - Secrets management system

Secrets Management System

CitihubDigital

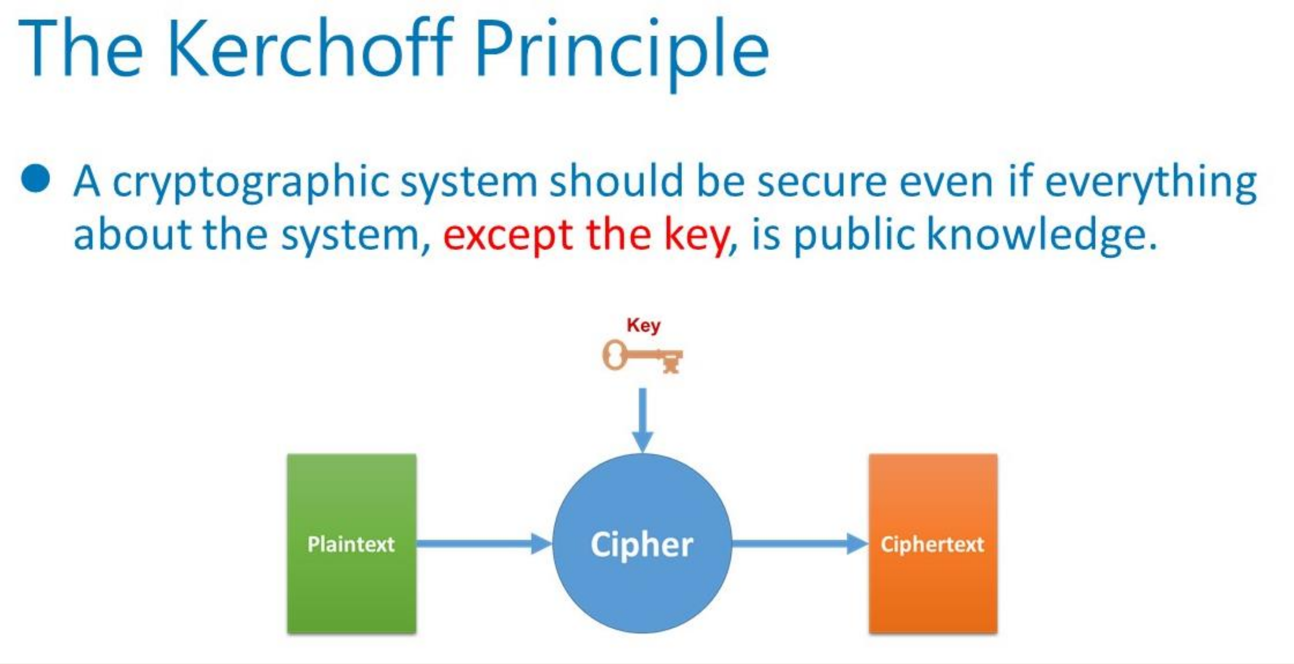


https://citihub.com/wp-content/uploads/2020/10/The-Secret-to-Good-Secrets-Management-for-the-Enterprise-and-Cloud-_-Citihub-Digital-_-Oct-2020.pdf

6. Open Design

- "Security through obscurity" is not secure (also violate Economy of Mechanism)
  - The more complex something is to understand, the harder it is to attack ?
- Security should not depend on secrecy of design or implementation

SECURITY THROUGH OBSCURITY
SECURITY THROUGH OBSCURITY
SECURITY THROUGH OBSCURITY
~#



The Kerchoff Principle

- A cryptographic system should be secure even if everything about the system, except the key, is public knowledge.

Key
Plaintext
Cipher
Ciphertext

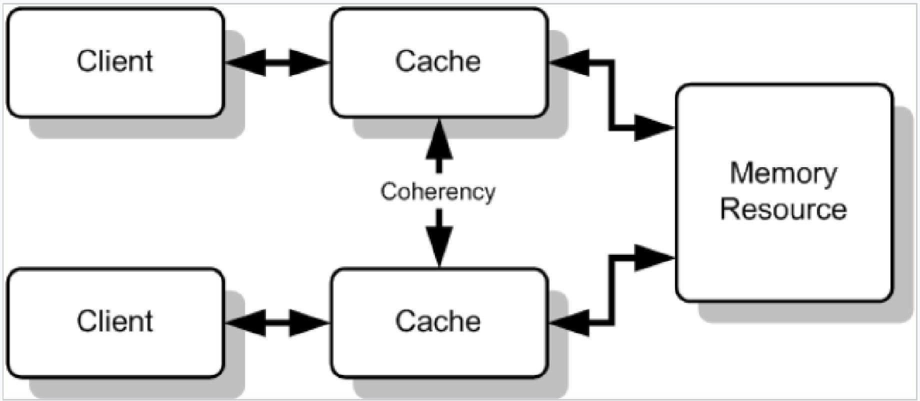
Sometimes misunderstood as that source code should be public

7. Least Common Mechanism

Shared resources should be minimized as much as possible

- Shared files
- Shared memory

- CVE-2017-5753, CVE-2017-5754
  - allow unauthorized disclosure of information to an attacker with local user access via a side-channel analysis of the data cache.

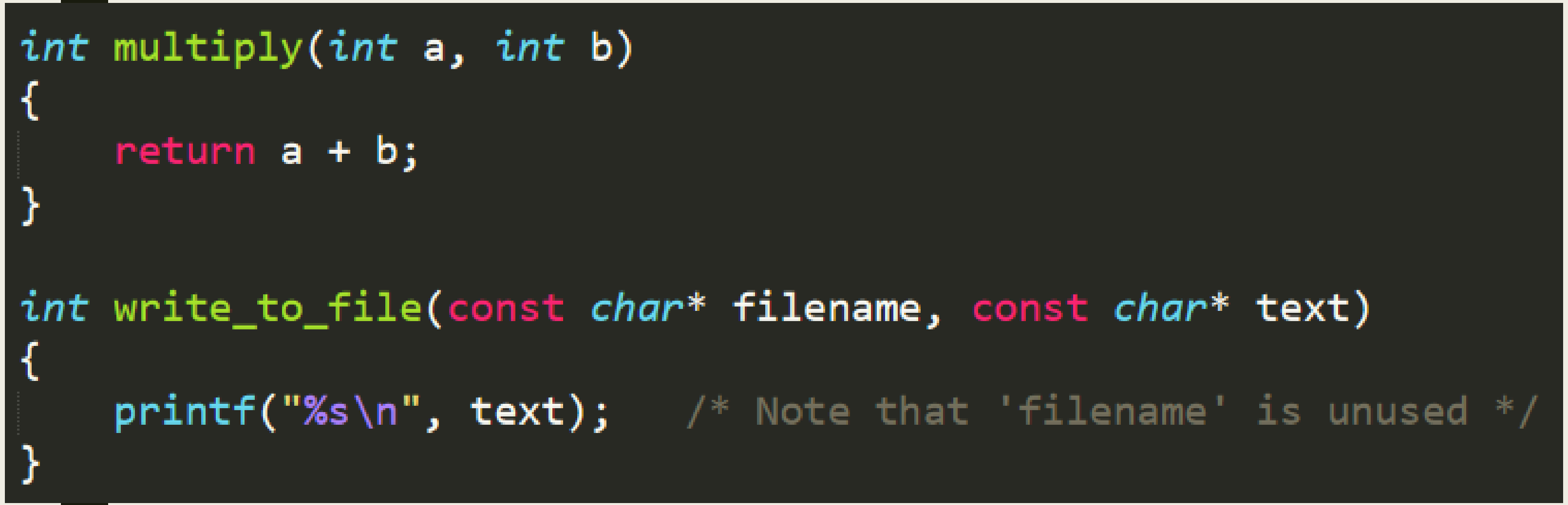


An illustration showing multiple caches of some memory, which acts as a shared resource

8. Least Astonishment

The behavior should not astonish or surprise users

- The result of performing some operation should be
  obvious, consistent, and predictable,
  based upon the name of the operation and other clues (comments/document).
- Good Coding Style



```
int multiply(int a, int b)
{
    return a + b;
}

int write_to_file(const char* filename, const char* text)
{
    printf("%s\n", text);    /* Note that 'filename' is unused */
}
```

The art of naming variables

```
whatsYourName = ['apple', 'banana', 'cherry']
```

The art of naming variables

Arrays

```
– const fruit = ['apple', 'banana', 'cherry'];            // bad - Is it an object?
– const fruitArr = ['apple', 'banana', 'cherry'];         // okay
– const fruits = ['apple', 'banana', 'cherry'];           // good - pluralizing makes sense
– const fruitNames = ['apple', 'banana', 'cherry'];       // great - "names" implies strings
```

The art of naming variables

Booleans

- Booleans can hold only 2 values, true or false. Given this, using prefixes like “is”, “has”, and “can” will help the reader infer the type of the variable.

- Bad examples
  - const open = true;
  - const write = true;
  - const fruit = true;

- Good examples
  - const isOpen = true;
  - const canWrite = true;
  - const hasFruit = true;

The art of naming variables

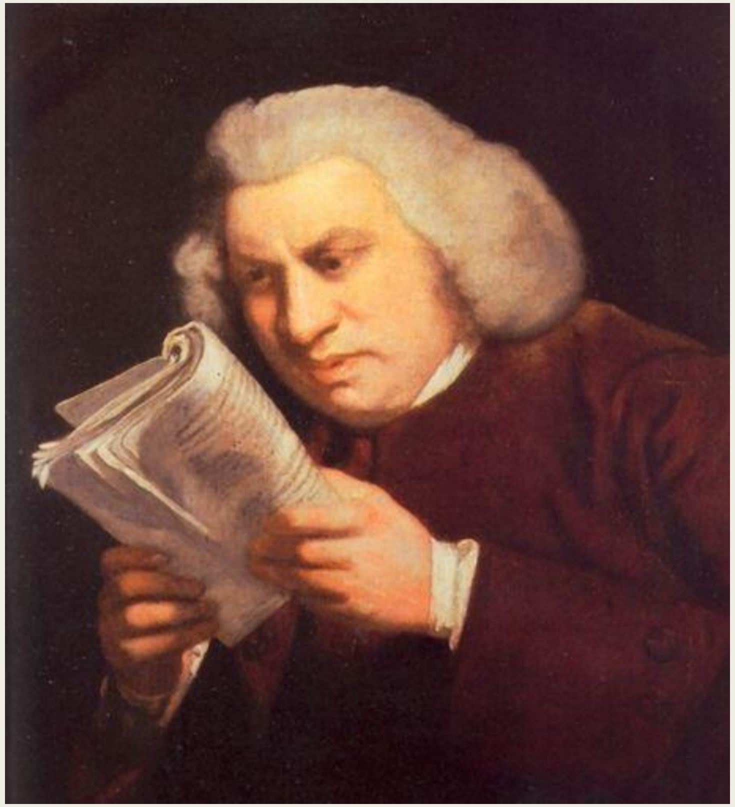
Functions

- Functions should be named using a verb, and a noun.
- A good format to follow is actionResource. For example, getUser.

- Bad examples
  - userData(userId)
  - userDataFunc(userId)
  - totalOfItems(items)

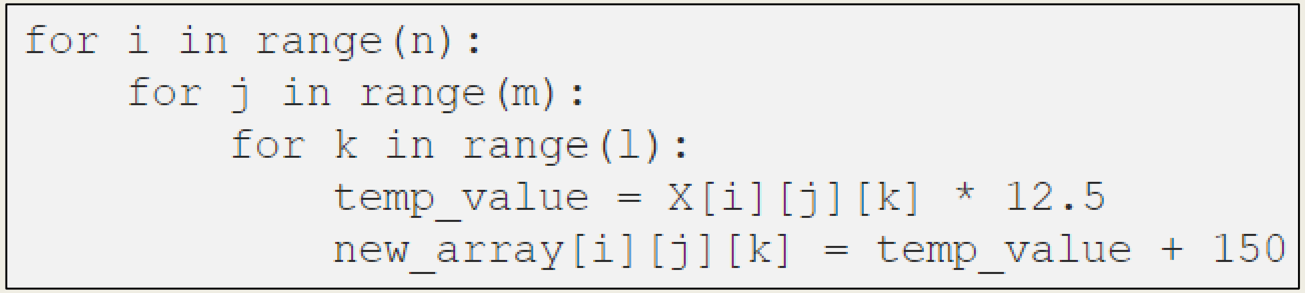
- Good examples
  - getUser(userId);
  - calculateTotal(items);

The art of naming variables



Loop Indexes

- Bad examples



```
for i in range(n):
    for j in range(m):
        for k in range(l):
            temp_value = X[i][j][k] * 12.5
            new_array[i][j][k] = temp_value + 150
```

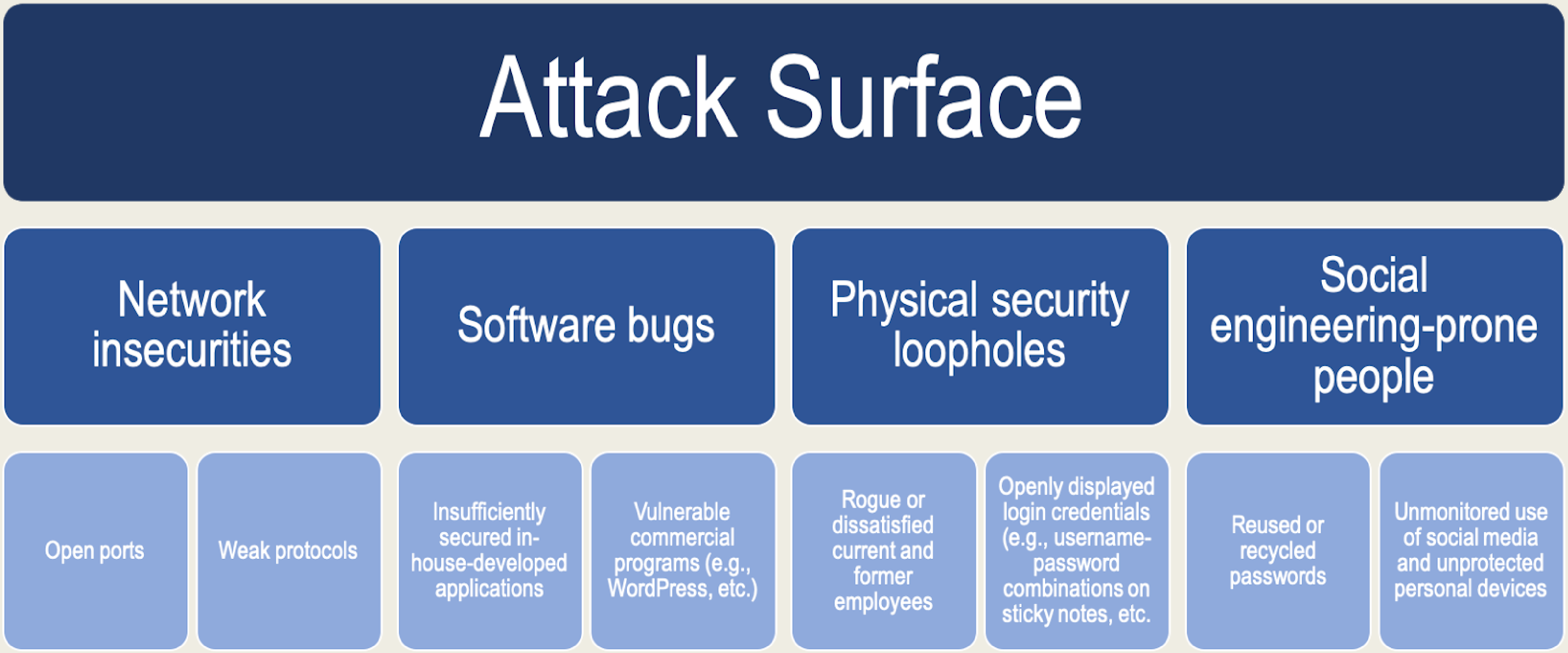
- Good examples
  - for row_index in range(row_count):
  - for building_index in range(building_count):
  - const newFruitNames = fruitNames.map(fruitName => { return doSomething(fruitName); });

The art of naming variables

Names to Avoid

- Non-instinctive names
  - let n = 'use name instead’ // Avoid Single Letter Names
  - let cra = 'no clue what this is’ // Avoid Acronyms
  - let cat = 'cat or category??’ // Avoid Abbreviations
  - let foo = 'what is foo??’ // Avoid Meaningless Names
- Common names
  - temp, tmp
  - var
  - results
  - key, value

9. Minimize Attack Surface



Attack Surface

Network insecurities
- Open ports
- Weak protocols

Software bugs
- Insufficiently secured in-house-developed applications
- Vulnerable commercial programs (e.g., WordPress, etc.)

Physical security loopholes
- Rogue or dissatisfied current and former employees
- Openly displayed login credentials (e.g., username-password combinations on sticky notes, etc.

Social engineering-prone people
- Reused or recycled passwords
- Unmonitored use of social media and unprotected personal devices

HUNTING VULNERABILITIES

Find Assumptions and Trusts (知己)

Find Threats (知彼)

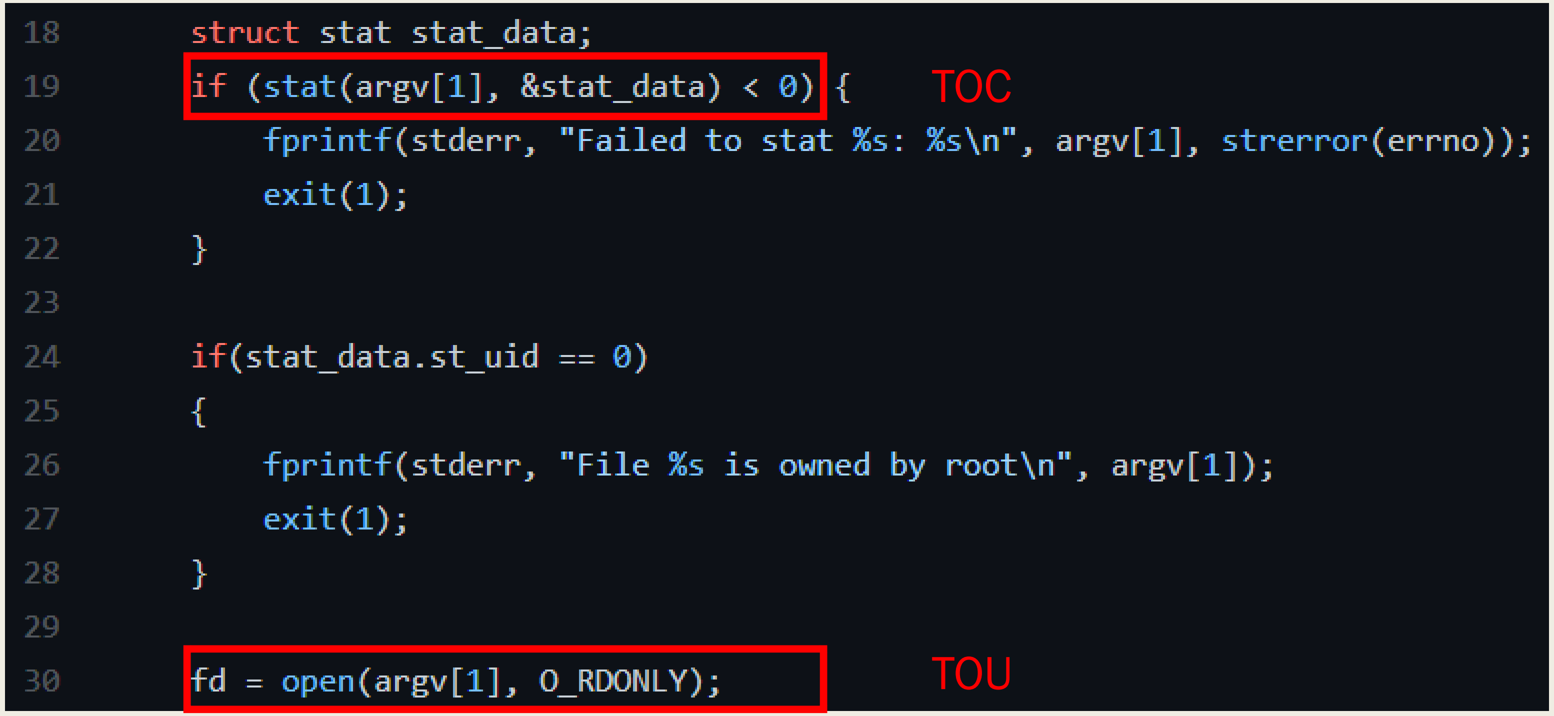
Hunting Vulnerabilities

- Find Assumptions and Trusts (知己)
  - Programs that assume atomicity of some functions
  - Programs that assume they are loaded as compiled
  - Programs that assume caller has cleaned up signals, open files
  - Programs that trust input to be well-formed
  - Programs that trust environment

- Find Threats (知彼)
  - All kinds of input should be treated as threat

Find Assumptions and Trusts

Aware of implicit assumptions



```c
    struct stat stat_data;
    if (stat(argv[1], &stat_data) < 0) {
        fprintf(stderr, "Failed to stat %s: %s\n", argv[1], strerror(errno));
        exit(1);
    }

    if(stat_data.st_uid == 0)
    {
        fprintf(stderr, "File %s is owned by root\n", argv[1]);
        exit(1);
    }

    fd = open(argv[1], O_RDONLY);
```

TOC

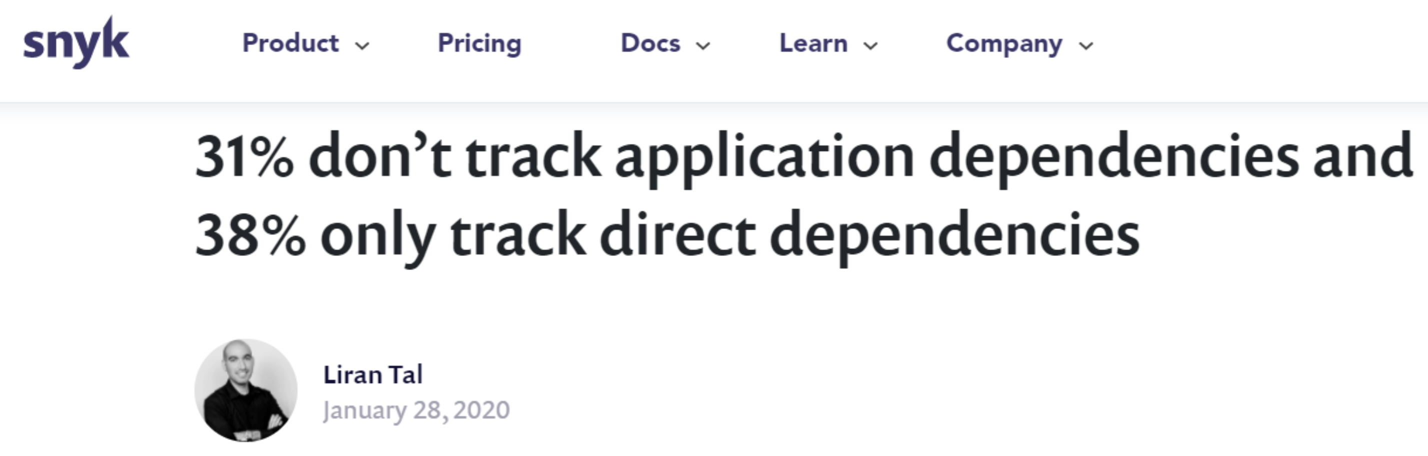
TOU

Implicitly assume argv[1] refer to the same file

Find Assumptions and Trusts

Dependency

- When you use third-party dependency, you
  - Inherit its assumptions
  - Inherit its vulnerabilities
- Check dependencies and update them constantly



31% don't track application dependencies and 38% only track direct dependencies

Liran Tal
January 28, 2020

https://snyk.io/blog/31-percent-dont-track-application-dependencies-and-38-only-track-direct-dependencies/

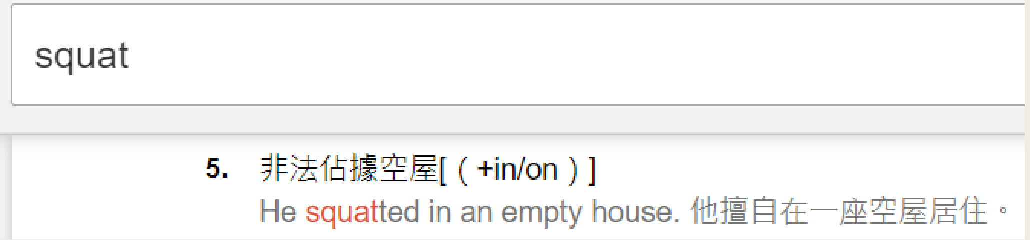
Dependency Confusion

- Which package is installed?
  - pip install package_pikachu
  - npm install package_pikachu

- For pip:
  1. Checks whether library exists on the specified (internal) package index
  2. Checks whether library exists on the public package index (PyPI)
  3. Installs whichever version is found. If the package exists on both, it defaults to installing from the source with the higher version number.
  - Therefore, uploading a package named library 9000.0.0 to PyPI would result in the dependency being hijacked

[Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies]

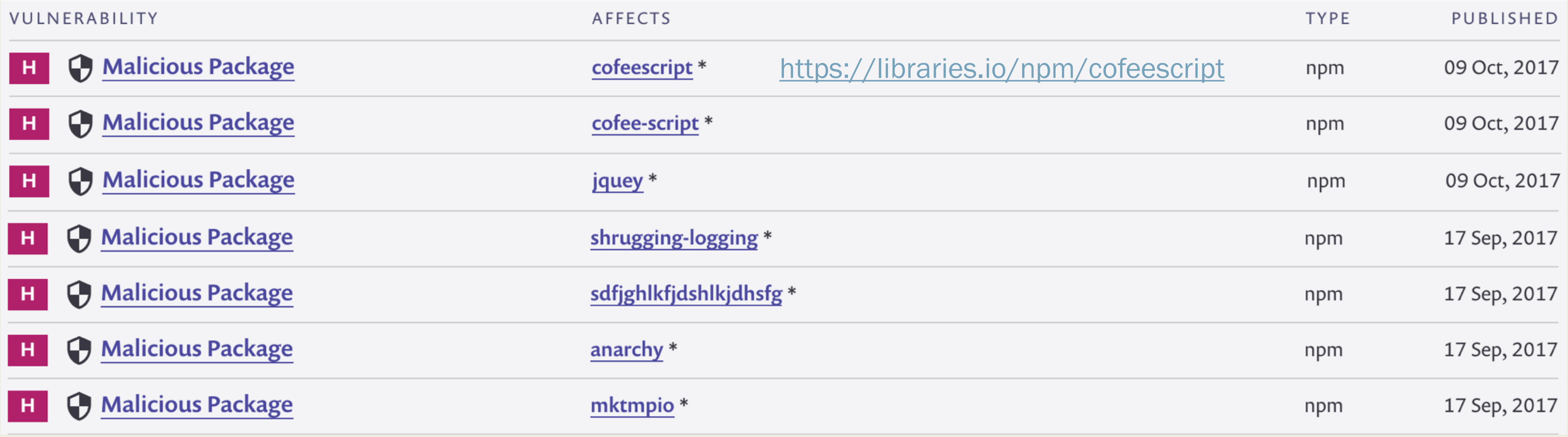
Typosquatting



squat

5. 非法佔據空屋[(+in/on)]
He squatted in an empty house. 他擅自在一座空屋居住。

- Pushing malicious packages to a registry with the hope of tricking users into installing them

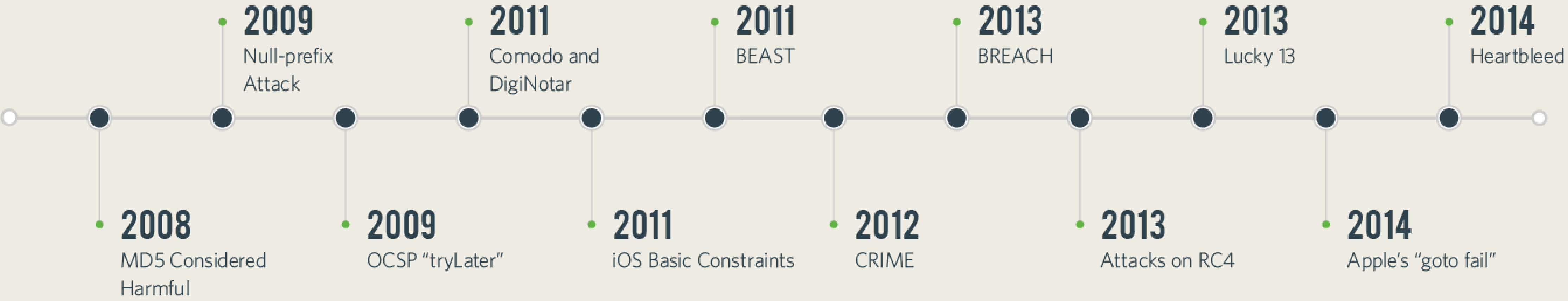


VULNERABILITY	AFFECTS	TYPE	PUBLISHED
H Malicious Package	cofeescript *	npm	09 Oct, 2017
H Malicious Package	cofee-script *	npm	09 Oct, 2017
H Malicious Package	jquey *	npm	09 Oct, 2017
H Malicious Package	shrugging-logging *	npm	17 Sep, 2017
H Malicious Package	sdfjghlkfjdshlkjdhsfg *	npm	17 Sep, 2017
H Malicious Package	anarchy *	npm	17 Sep, 2017
H Malicious Package	mktmpio *	npm	17 Sep, 2017

https://libraries.io/npm/cofeescript

Famous example: Heartbleed (CVE-2014-0160)

- Vulnerability was introduced to OpenSSL library in December 2011
- "Flaws" of SSL before 2014:



2008
MD5 Considered Harmful

2009
Null-prefix Attack

2009
OCSP "tryLater"

2011
Comodo and DigiNotar

2011
iOS Basic Constraints

2011
BEAST

2012
CRIME

2013
BREACH

2013
Attacks on RC4

2013
Lucky 13

2014
Apple's "goto fail"

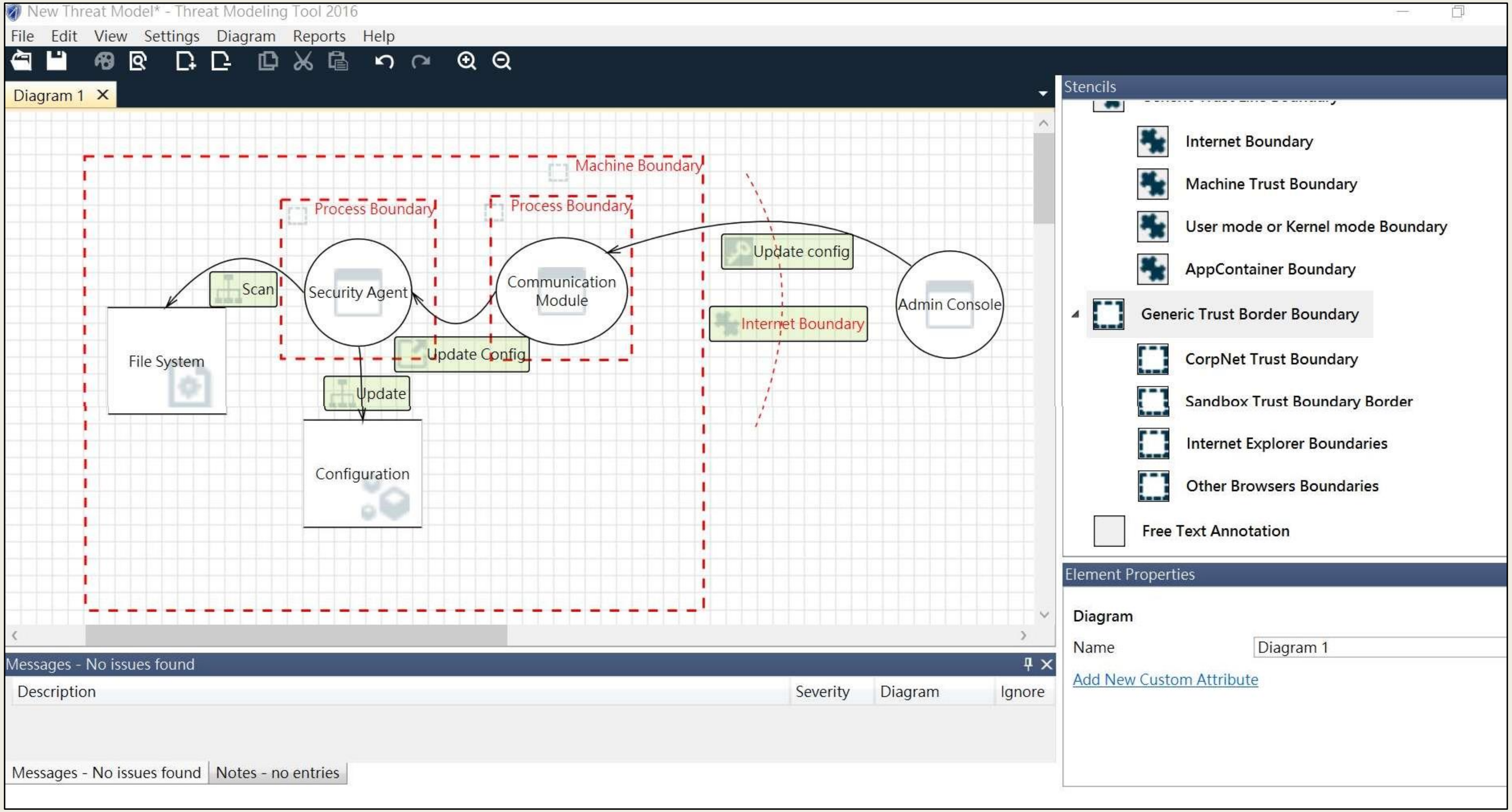
2014
Heartbleed

https://duo.com/blog/heartbleed-defense-in-depth-part-2

Find Threats

- Threat Modeling
  - Identify attack surface during the design phase
  - Minimize Attack Surface
  - Focused Defense

Threat Modeling Tool



Diagram

Name: Diagram 1

Messages - No issues found

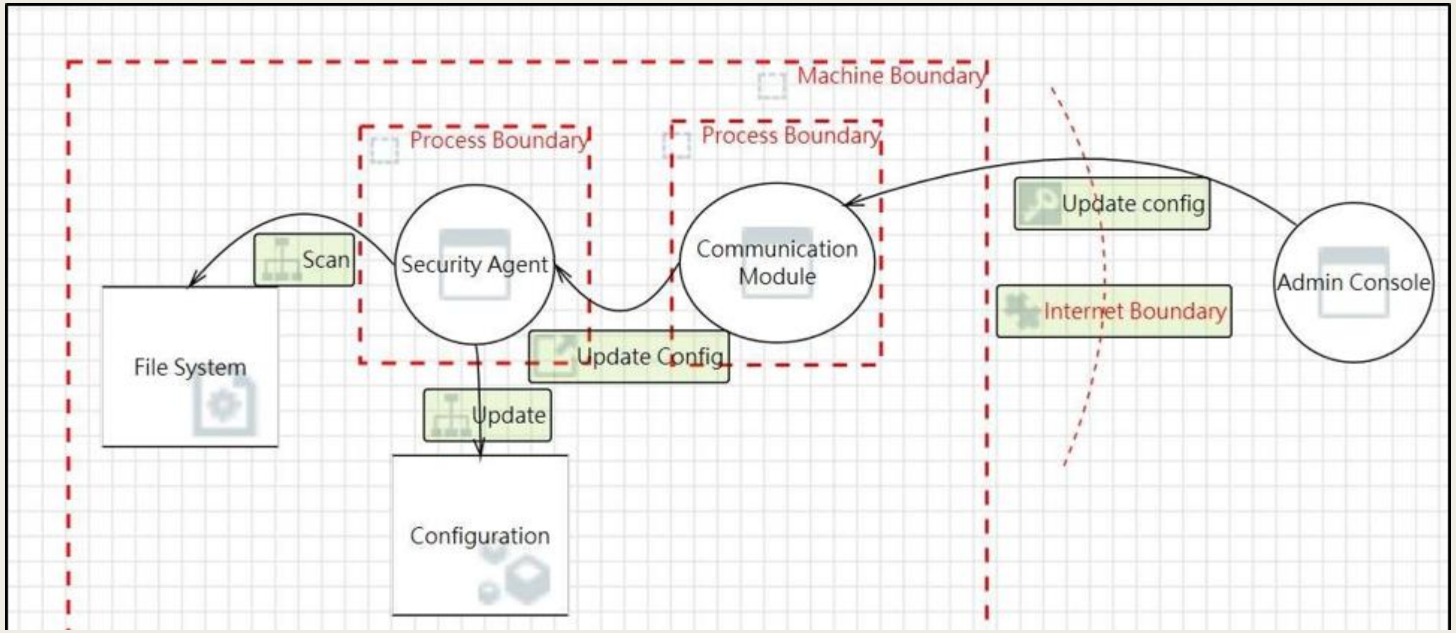
Description	Severity	Diagram	Ignore

https://www.microsoft.com/en-us/securityengineering/sdl/threatmodeling

Data Flow Diagram (DFD)

Graphically represent the flow of data in an information system

- Include processes, data stores, data flows, trust boundaries
- Enumerate assumptions, dependencies
- Diagram per scenario may be helpful
- Update diagrams as product changes



DFD Elements

External Entity

- People
- Other Systems

Process

- EXEs
- DLLs
- Component
- Services

Data Flow

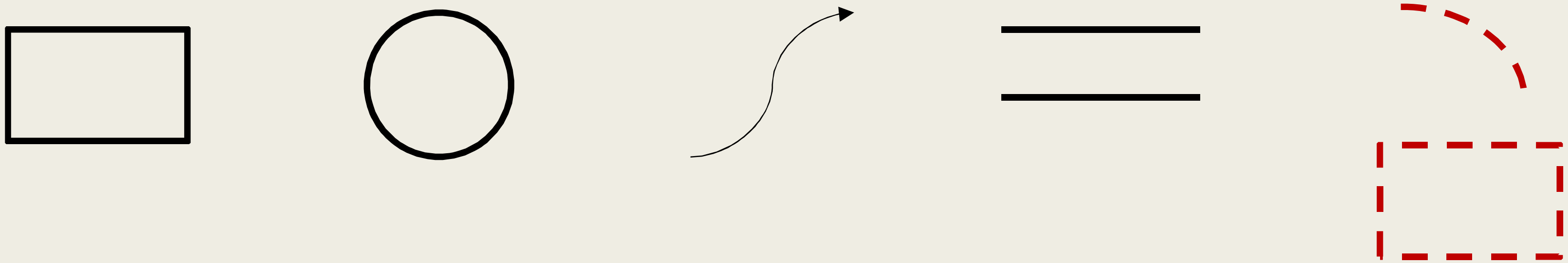
- Function call
- Network traffic
- Remote Procedure Call

Data Store

- Database
- File
- Registry
- Queue

Trust Boundary

- Process boundary
- Machine boundary
- VM
- Network



DFD Trust Boundary

Trust boundaries indicate where trust levels change

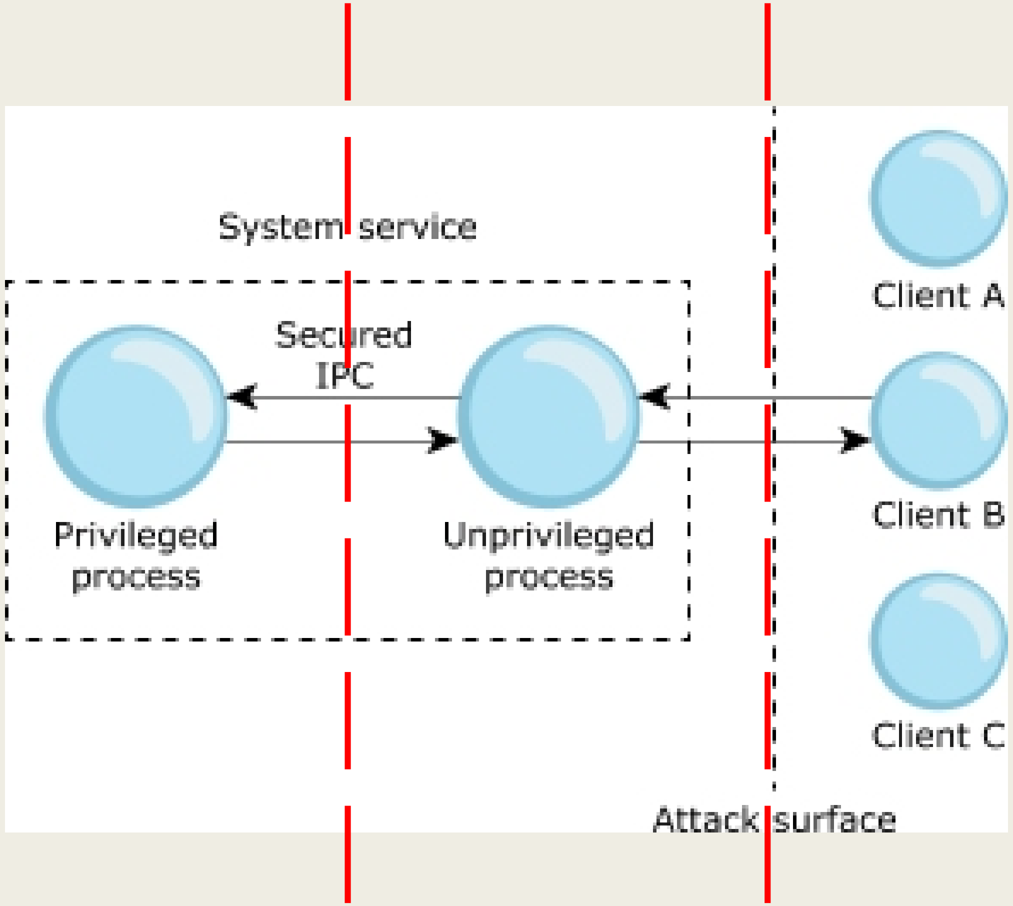
  - Machine Boundary
  - Process Boundary
  - Privilege Boundary
  - Internet Boundary

- Data need to be validated/sanitized after crossing the boundary
- Processes talking across a network always have a trust boundary
  - Encrypting network traffic doesn’t address tampering or spoofing

DFD Trust Boundary

Privilege Boundary

Privileged process:
Input validation is needed
to prevent SSRF.



Internet Boundary (Untrusted data controlled by users)

DFD Sensitive Data

It is advised to identify sensitive data in the DFDs

- Customer data privacy
  - Identifying what privacy data are stored by the product
  - Need protection for GDPR (General Data Protection Regulation) compliance
- Product specific sensitive data
  - E.g. configuration to control whether a security feature is disabled
  - E.g. AES key, private key
  - E.g. customer ID, user name and password
- Focused on defending these sensitive data

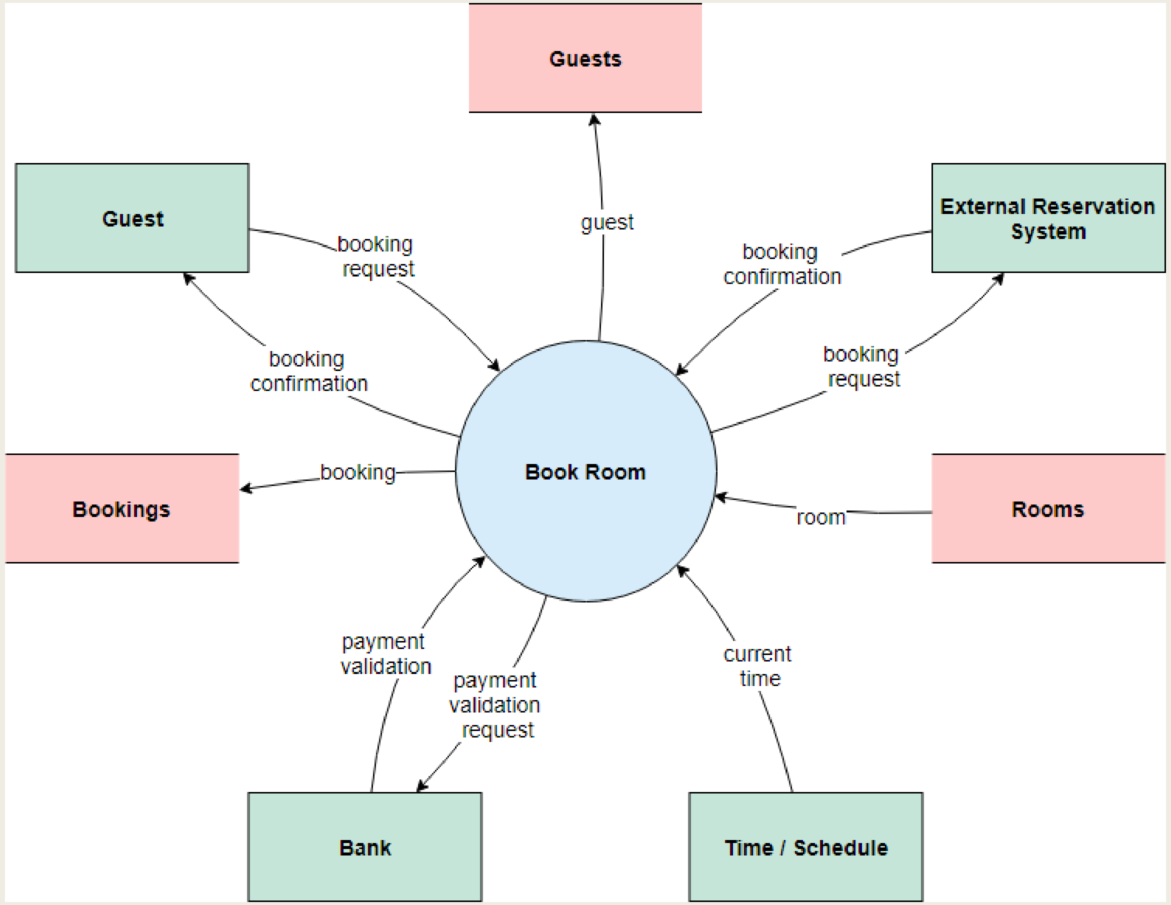
DFD Layers

- Level 0 DFD (System Context Diagram)
  - Highest level; only one process / product / system
  - It identifies the data flows between the system and external entities.
  - A context diagram is typically included in a requirements document.
- Level 1 DFD
  - High level; single feature / scenario
- Level 2 DFD
  - Low level; detailed sub-components of features
- Level 3 DFD
  - More detailed

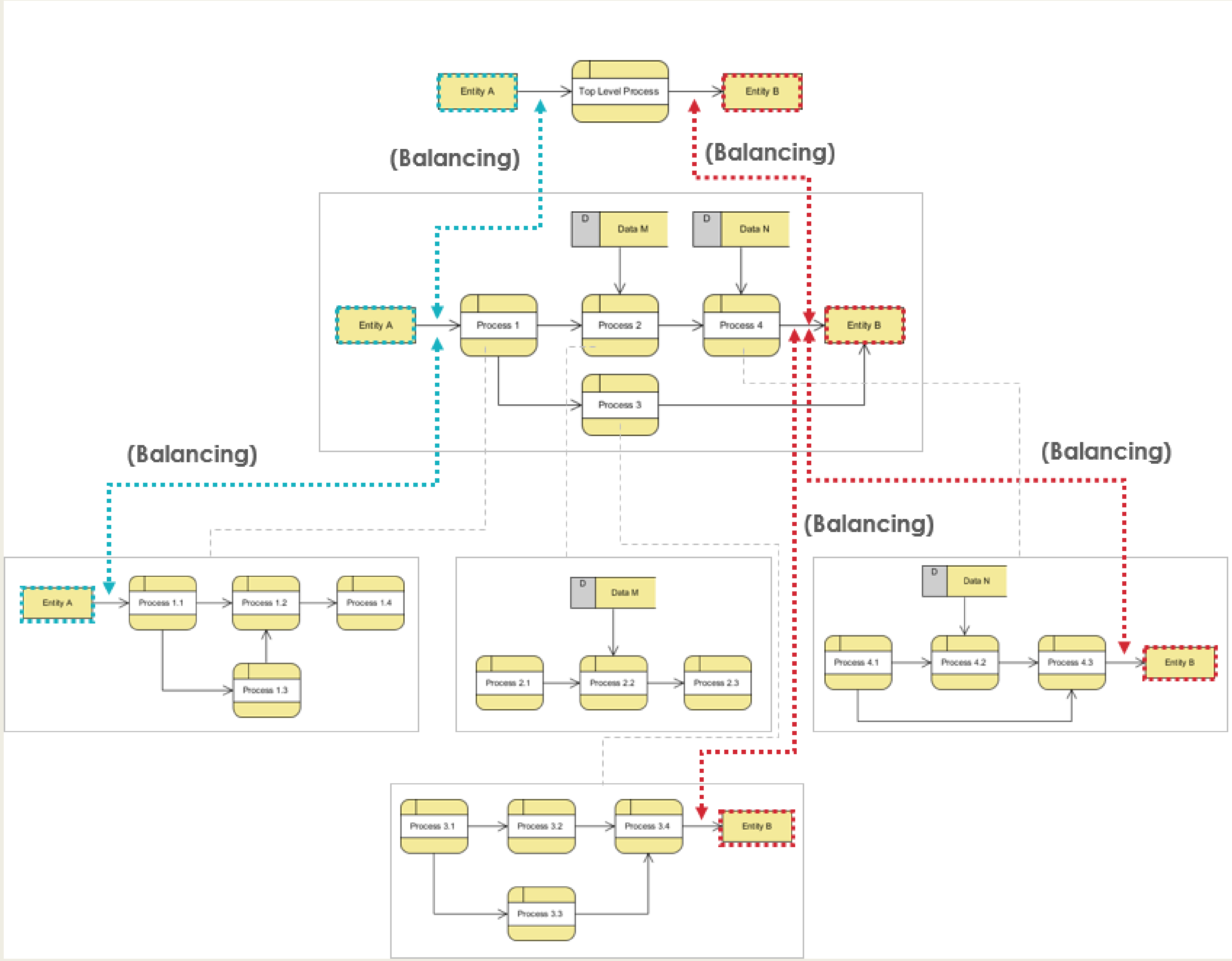
Level 0 DFD (System Context Diagram)

The entire software system is shown as a single process, with no details of its interior structure, surrounded by all its external entities, interacting systems, and environments.

- Help you define interfaces
- Interfaces should be stable



DFD Top-Down Decomposition



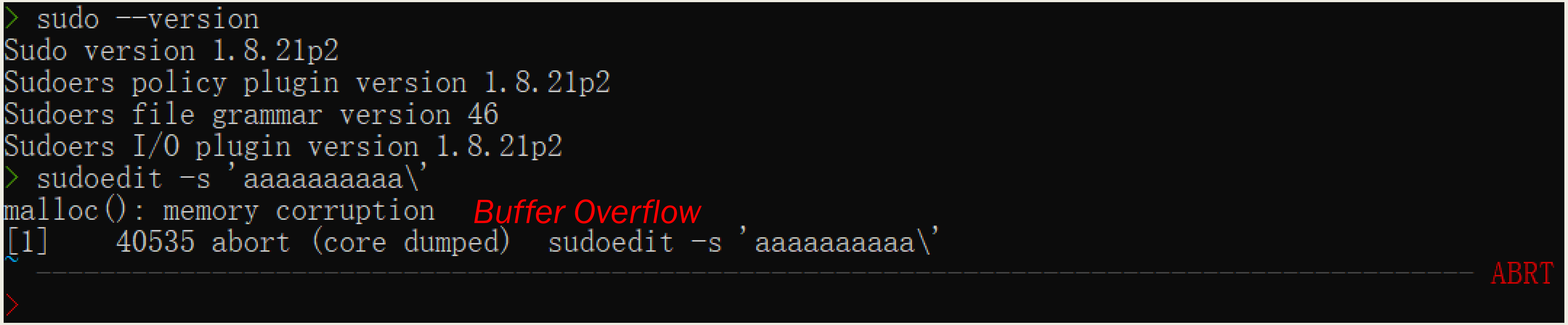
Leveling
Level 0 (Context)
Level 1
Level 2

CVE-2021-3156

Heap-Based Buffer Overflow in Sudo

CVE-2021-3156:
Heap-Based Buffer Overflow in Sudo

- This vulnerability has been hiding in plain sight for nearly 10 years.
- It was introduced in July 2011 (commit 8255ed69).
- Discovered by Qualys through code review.

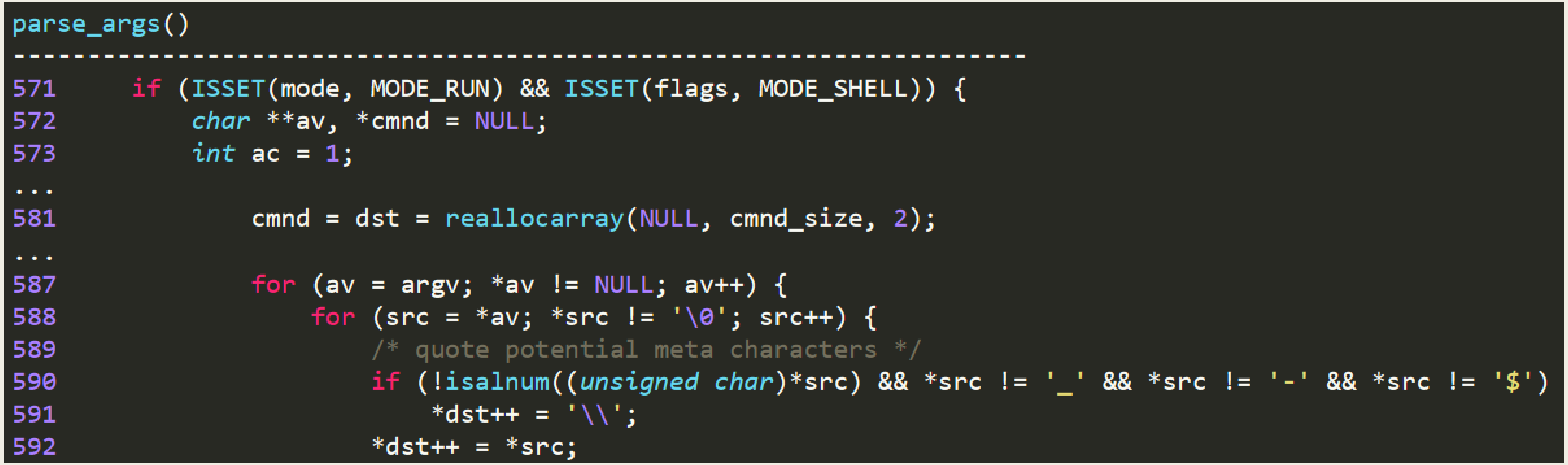


```
> sudo --version
Sudo version 1.8.21p2
Sudoers policy plugin version 1.8.21p2
Sudoers file grammar version 46
Sudoers I/O plugin version 1.8.21p2
> sudoedit -s 'aaaaaaaaaa\'
malloc(): memory corruption
[1]    40535 abort (core dumped)  sudoedit -s 'aaaaaaaaaa\'
~ ------------------------------------------------------------------------------------------ ABRT
>
```

Buffer Overflow

CVE-2021-3156:
Heap-Based Buffer Overflow in Sudo

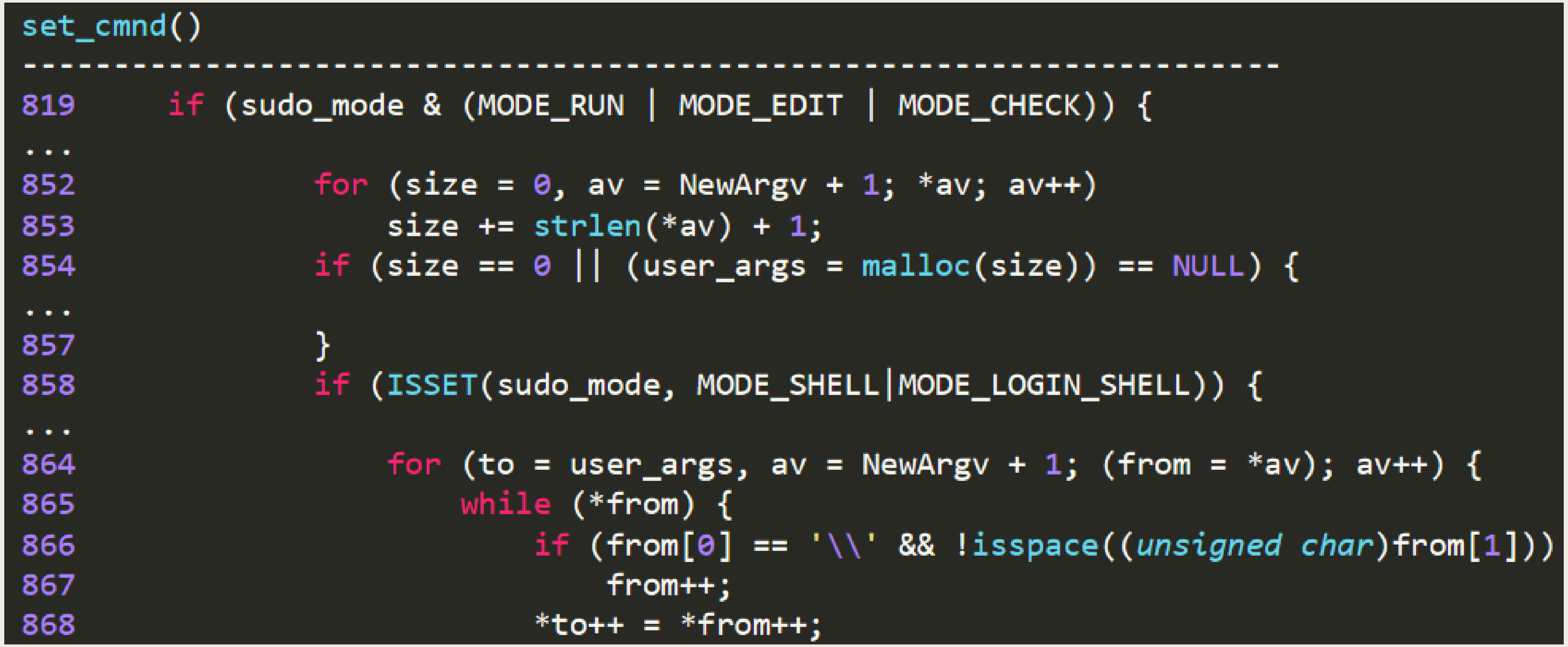
- parse_args()
  - escaped all meta-characters, including backslashes



```
parse_args()
---------------------------------------------------------------------
    if (ISSET(mode, MODE_RUN) && ISSET(flags, MODE_SHELL)) {
        char **av, *cmnd = NULL;
        int ac = 1;
...
            cmnd = dst = reallocarray(NULL, cmnd_size, 2);
...
            for (av = argv; *av != NULL; av++) {
                for (src = *av; *src != '\0'; src++) {
                    /* quote potential meta characters */
                    if (!isalnum((unsigned char)*src) && *src != '_' && *src != '-' && *src != '$')
                        *dst++ = '\\';
                    *dst++ = *src;
```

CVE-2021-3156:
Heap-Based Buffer Overflow in Sudo

- set_cmnd()
  - is vulnerable to a heap-based buffer overflow
  - however, no command-line argument can end with a single backslash character: if MODE_SHELL or MODE_LOGIN_SHELL is set



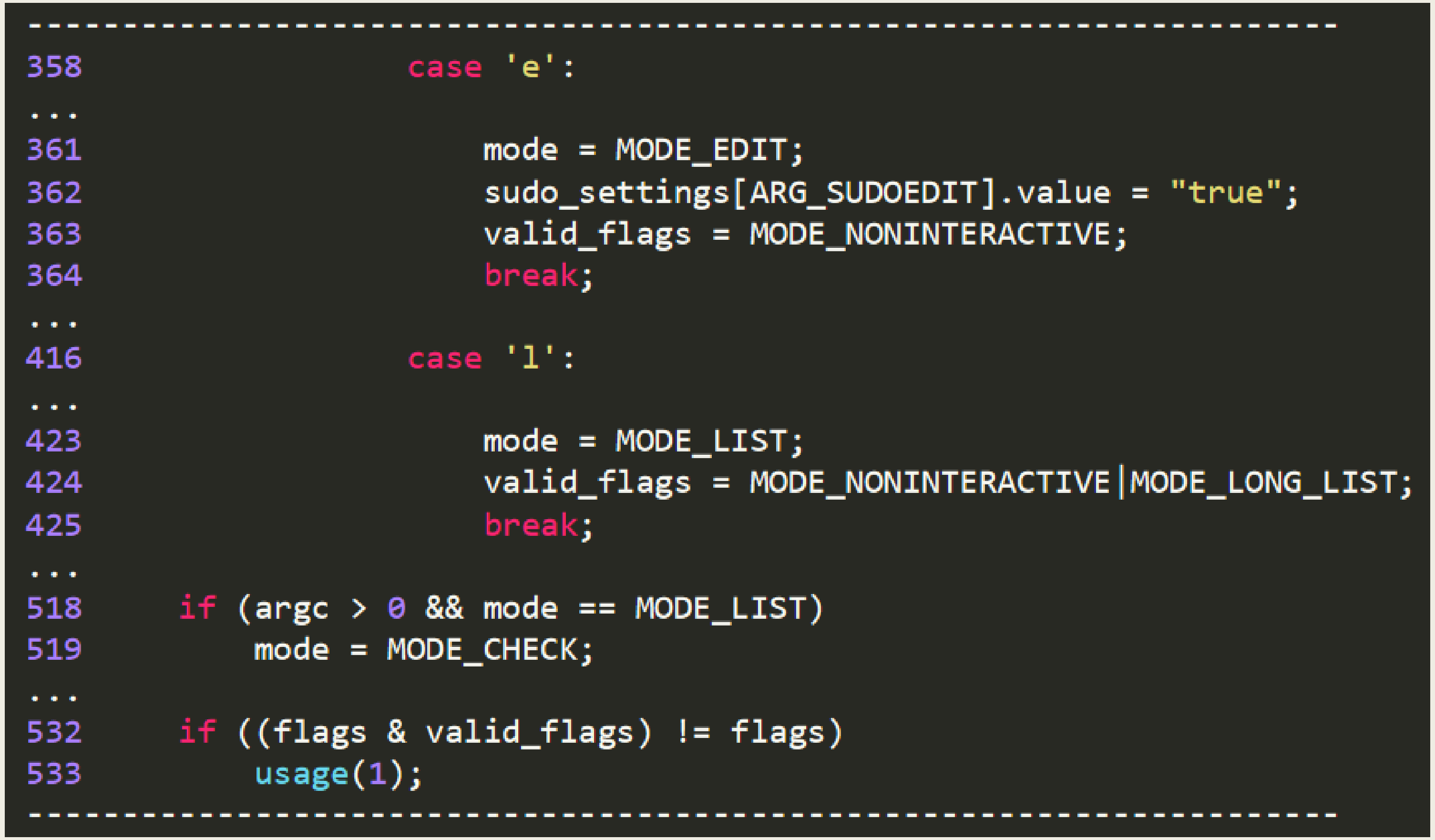
```
set_cmnd()
--------------------------------------------------------
    if (sudo_mode & (MODE_RUN | MODE_EDIT | MODE_CHECK)) {
...
        for (size = 0, av = NewArgv + 1; *av; av++)
          size += strlen(*av) + 1;
        if (size == 0 || (user_args = malloc(size)) == NULL) {
...
        }
        if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
...
          for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
            while (*from) {
              if (from[0] == '\\' && !isspace((unsigned char)from[1]))
                from++;
              *to++ = *from++;
```

CVE-2021-3156: Heap-Based Buffer Overflow in Sudo

Sudo

- -s option
  - MODE_SHELL
- -i option
  - MODE_SHELL
  - MODE_LOGIN_SHELL

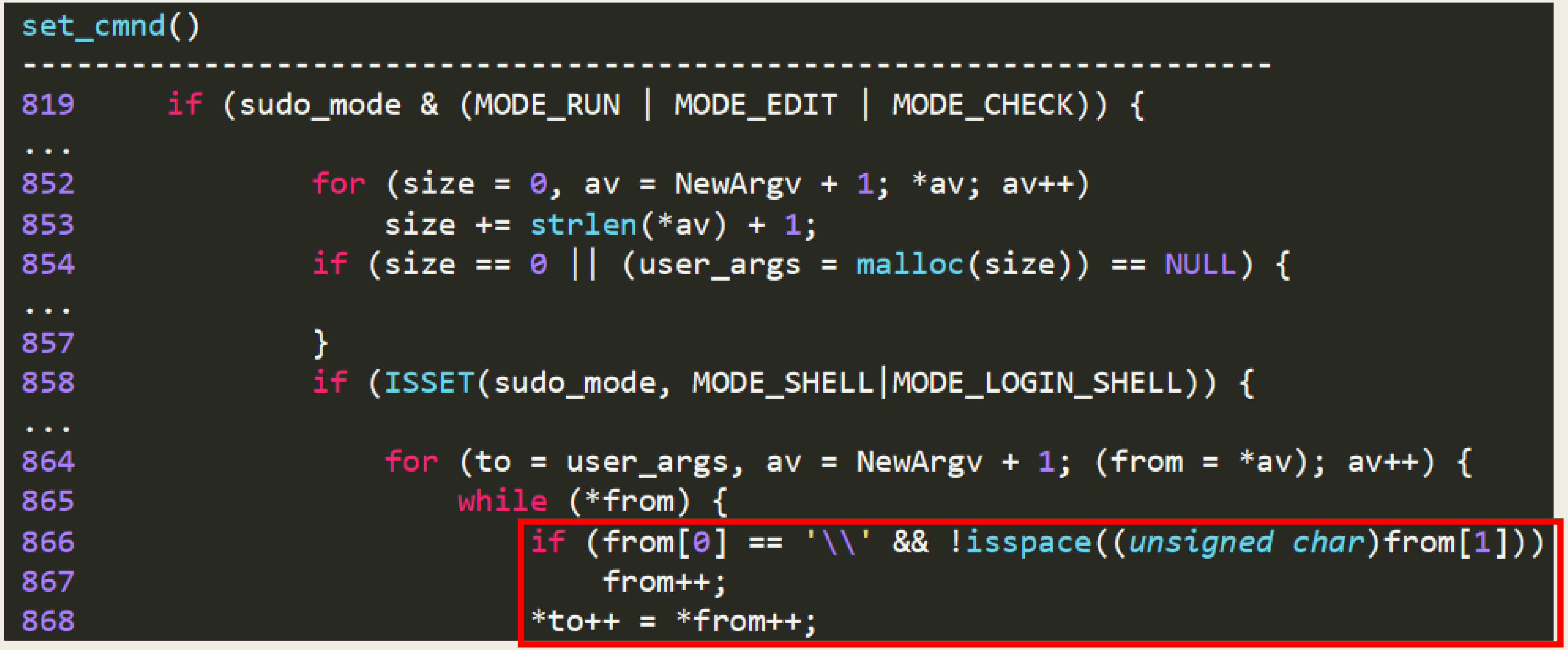
- Can we set MODE_SHELL and either MODE_EDIT or MODE_CHECK (to reach the vulnerable code) but not the default MODE_RUN (to avoid the escape code)?



```
------------------------------------------------------------------
358       case 'e':
...
361         mode = MODE_EDIT;
362         sudo_settings[ARG_SUDOEDIT].value = "true";
363         valid_flags = MODE_NONINTERACTIVE;
364         break;
...
416       case 'l':
...
423         mode = MODE_LIST;
424         valid_flags = MODE_NONINTERACTIVE|MODE_LONG_LIST;
425         break;
...
518 if (argc > 0 && mode == MODE_LIST)
519   mode = MODE_CHECK;
...
532 if ((flags & valid_flags) != flags)
533   usage(1);
------------------------------------------------------------------
```

Line 363, 424 removes MODE_SHELL from the “valid_flags”

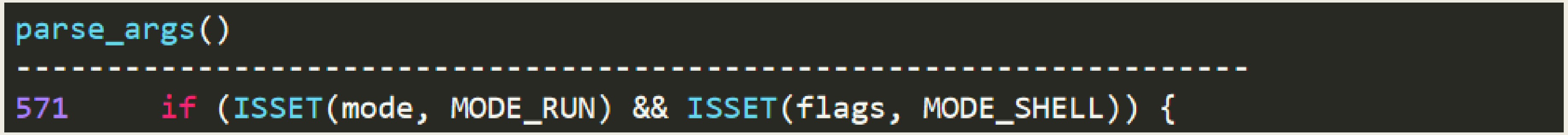
CVE-2021-3156:
Heap-Based Buffer Overflow in Sudo



```
set_cmnd()
--------------------------------------------------------------
if (sudo_mode & (MODE_RUN | MODE_EDIT | MODE_CHECK)) {
...
        for (size = 0, av = NewArgv + 1; *av; av++)
            size += strlen(*av) + 1;
        if (size == 0 || (user_args = malloc(size)) == NULL) {
...
        }
        if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
...
            for (to = user_args, av = NewArgv + 1; (from = *av); av++) {
                while (*from) {
                    if (from[0] == '\\' && !isspace((unsigned char)from[1]))
                        from++;
                    *to++ = *from++;
```

Buffer Overflow

Assumption: Data is sanitized by parse_args() → 不明顯，只有開發者自己知道，可讀性差 → 可維護性差

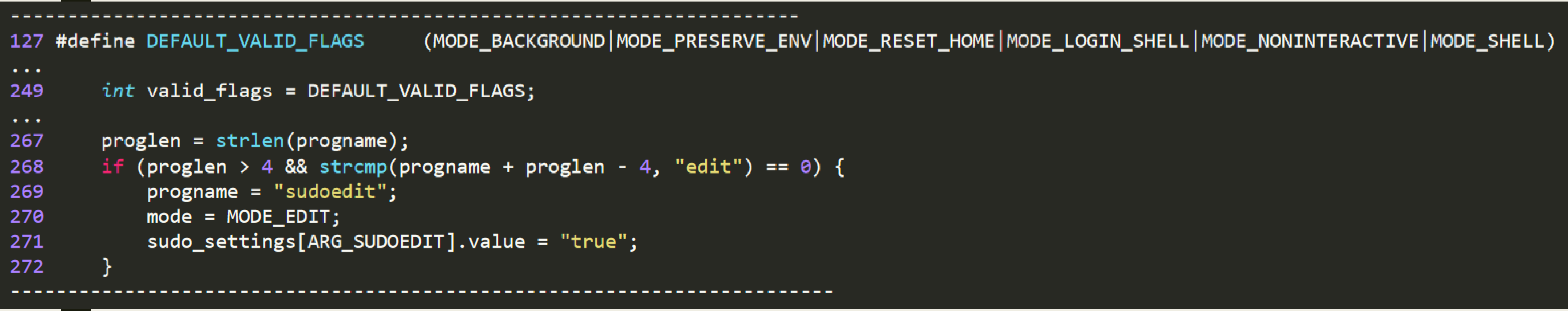


```
parse_args()
------------------------------------------------------------
if (ISSET(mode, MODE_RUN) && ISSET(flags, MODE_SHELL)) {
```

CVE-2021-3156:
Heap-Based Buffer Overflow in Sudo

But we found a loophole: if we execute Sudo as “sudoedit” instead of “sudo”,

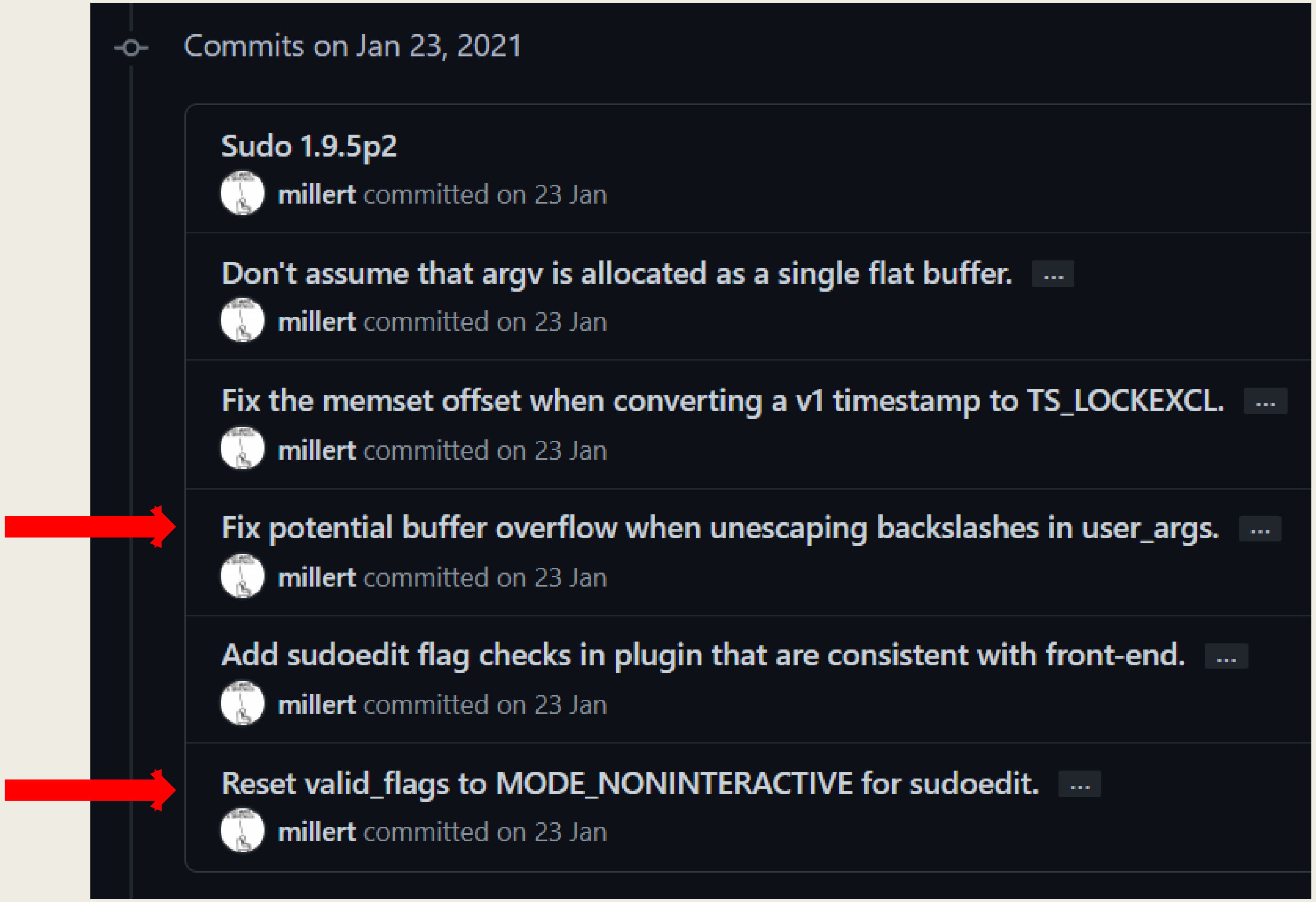
- parse_args() automatically sets MODE_EDIT (line 270)
- but does not reset “valid_flags”
- and the “valid_flags” include MODE_SHELL by default (lines 127 and 249)



```
---------------------------------------------------------------------
127 #define DEFAULT_VALID_FLAGS     (MODE_BACKGROUND|MODE_PRESERVE_ENV|MODE_RESET_HOME|MODE_LOGIN_SHELL|MODE_NONINTERACTIVE|MODE_SHELL)
...
249     int valid_flags = DEFAULT_VALID_FLAGS;
...
267     proglen = strlen(progname);
268     if (proglen > 4 && strcmp(progname + proglen - 4, "edit") == 0) {
269         progname = "sudoedit";
270         mode = MODE_EDIT;
271         sudo_settings[ARG_SUDOEDIT].value = "true";
272     }
------------------------------------------------------------------------
```

CVE-2021-3156:
Heap-Based Buffer Overflow in Sudo

Patches: https://github.com/sudo-project/sudo/commits/main?after=06cb6459c10e3c2d46f229237662d6cfe354d4b5+349&branch=main



Commits on Jan 23, 2021

Sudo 1.9.5p2
millert committed on 23 Jan

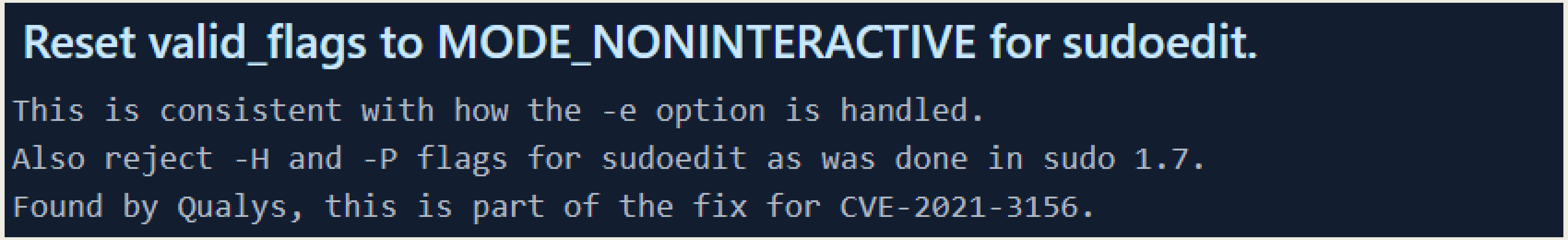
Don't assume that argv is allocated as a single flat buffer. ...
millert committed on 23 Jan

Fix the memset offset when converting a v1 timestamp to TS_LOCKEXCL. ...
millert committed on 23 Jan

Fix potential buffer overflow when unescaping backslashes in user_args. ...
millert committed on 23 Jan

Add sudoedit flag checks in plugin that are consistent with front-end. ...
millert committed on 23 Jan

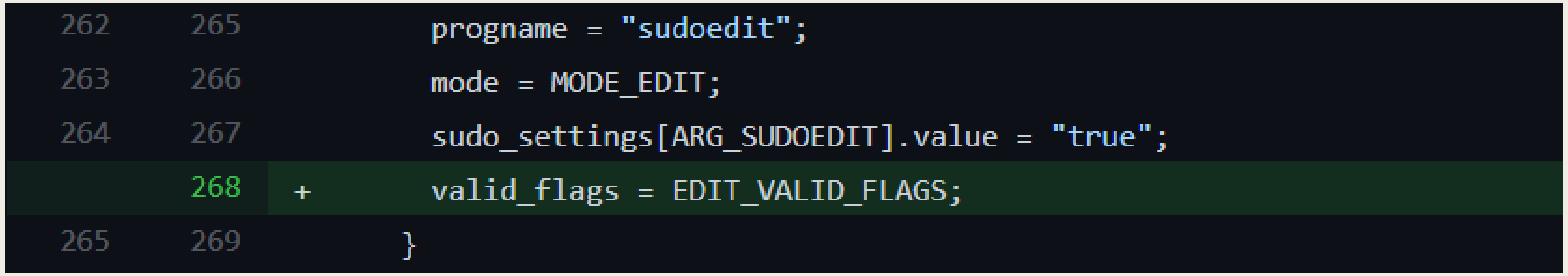
Reset valid_flags to MODE_NONINTERACTIVE for sudoedit. ...
millert committed on 23 Jan



Reset valid_flags to MODE_NONINTERACTIVE for sudoedit.

```
This is consistent with how the -e option is handled.
Also reject -H and -P flags for sudoedit as was done in sudo 1.7.
Found by Qualys, this is part of the fix for CVE-2021-3156.
```

parse_args()

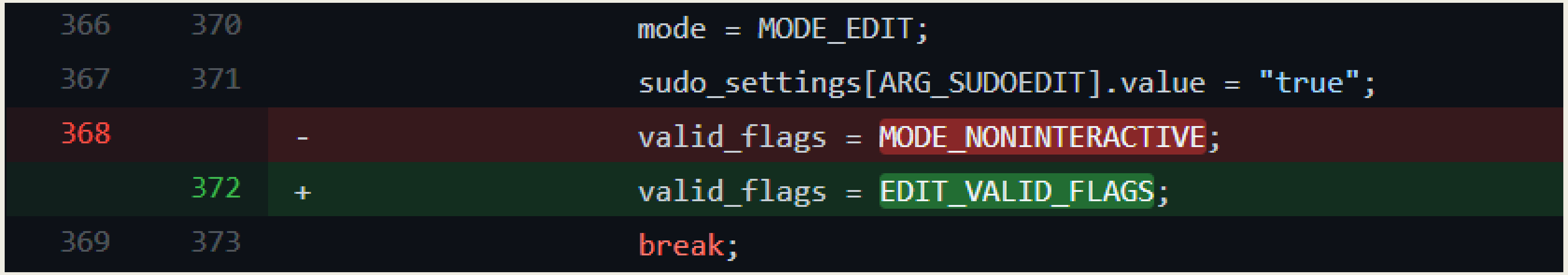


```
                progname = "sudoedit";
                mode = MODE_EDIT;
                sudo_settings[ARG_SUDOEDIT].value = "true";
+               valid_flags = EDIT_VALID_FLAGS;
            }
```

看似是漏洞的成因

但其實有 2 個地方可以防止漏洞發生(下下頁)

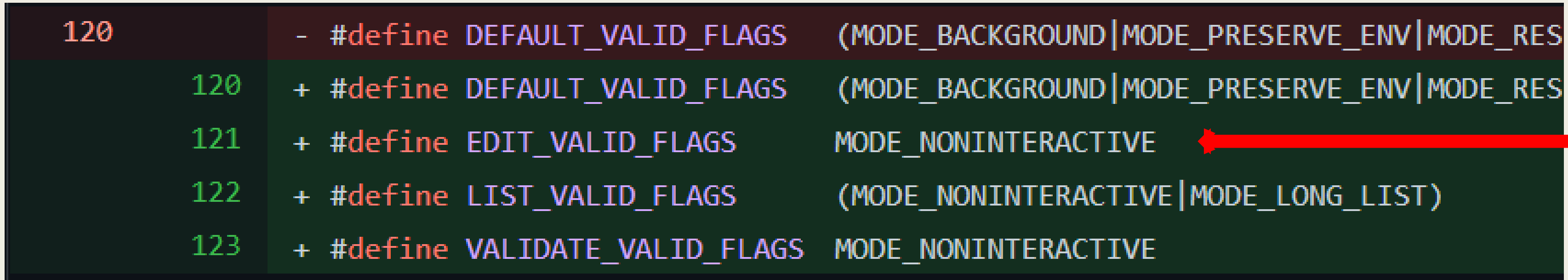
parse_args()



```
                        mode = MODE_EDIT;
                        sudo_settings[ARG_SUDOEDIT].value = "true";
-                       valid_flags = MODE_NONINTERACTIVE;
+                       valid_flags = EDIT_VALID_FLAGS;
                        break;
```

啥碗糕

原來是 valid flag

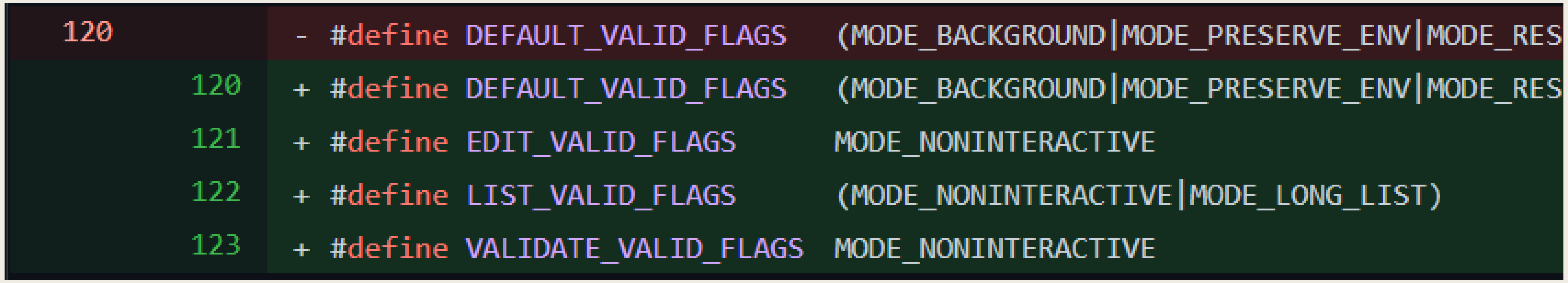


```
- #define DEFAULT_VALID_FLAGS   (MODE_BACKGROUND|MODE_PRESERVE_ENV|MODE_RES
+ #define DEFAULT_VALID_FLAGS   (MODE_BACKGROUND|MODE_PRESERVE_ENV|MODE_RES
+ #define EDIT_VALID_FLAGS      MODE_NONINTERACTIVE
+ #define LIST_VALID_FLAGS      (MODE_NONINTERACTIVE|MODE_LONG_LIST)
+ #define VALIDATE_VALID_FLAGS  MODE_NONINTERACTIVE
```

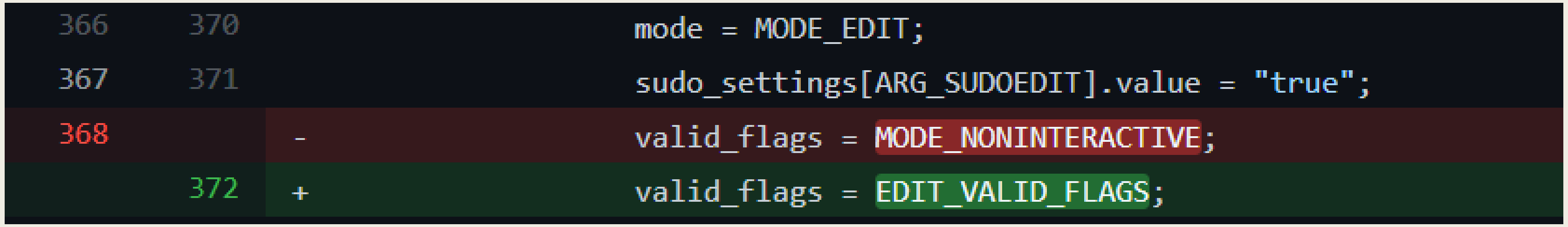
引入解釋變量

提升可讀性

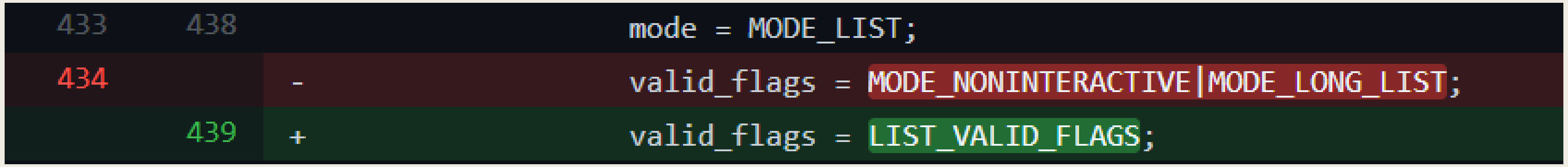
加了 3 個解釋變量



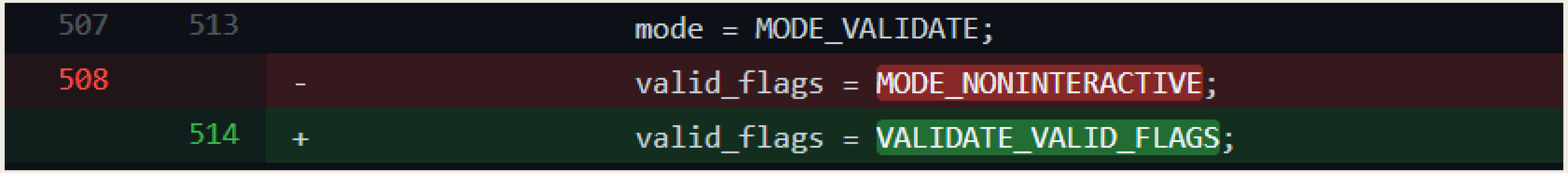
```
- #define DEFAULT_VALID_FLAGS   (MODE_BACKGROUND|MODE_PRESERVE_ENV|MODE_RES
+ #define DEFAULT_VALID_FLAGS   (MODE_BACKGROUND|MODE_PRESERVE_ENV|MODE_RES
+ #define EDIT_VALID_FLAGS      MODE_NONINTERACTIVE
+ #define LIST_VALID_FLAGS      (MODE_NONINTERACTIVE|MODE_LONG_LIST)
+ #define VALIDATE_VALID_FLAGS  MODE_NONINTERACTIVE
```



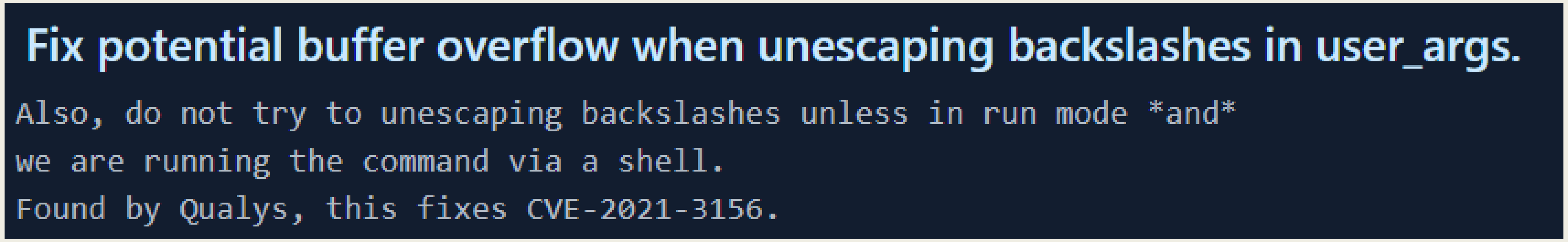
```
                   mode = MODE_EDIT;
                   sudo_settings[ARG_SUDOEDIT].value = "true";
-                  valid_flags = MODE_NONINTERACTIVE;
+                  valid_flags = EDIT_VALID_FLAGS;
```



```
                   mode = MODE_LIST;
-                  valid_flags = MODE_NONINTERACTIVE|MODE_LONG_LIST;
+                  valid_flags = LIST_VALID_FLAGS;
```



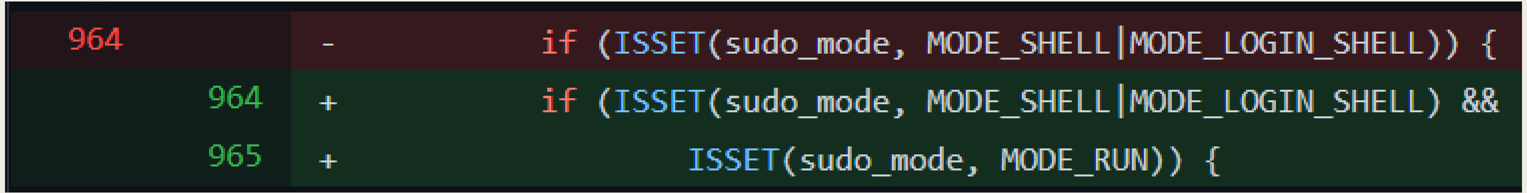
```
                   mode = MODE_VALIDATE;
-                  valid_flags = MODE_NONINTERACTIVE;
+                  valid_flags = VALIDATE_VALID_FLAGS;
```



Fix potential buffer overflow when unescaping backslashes in user_args.

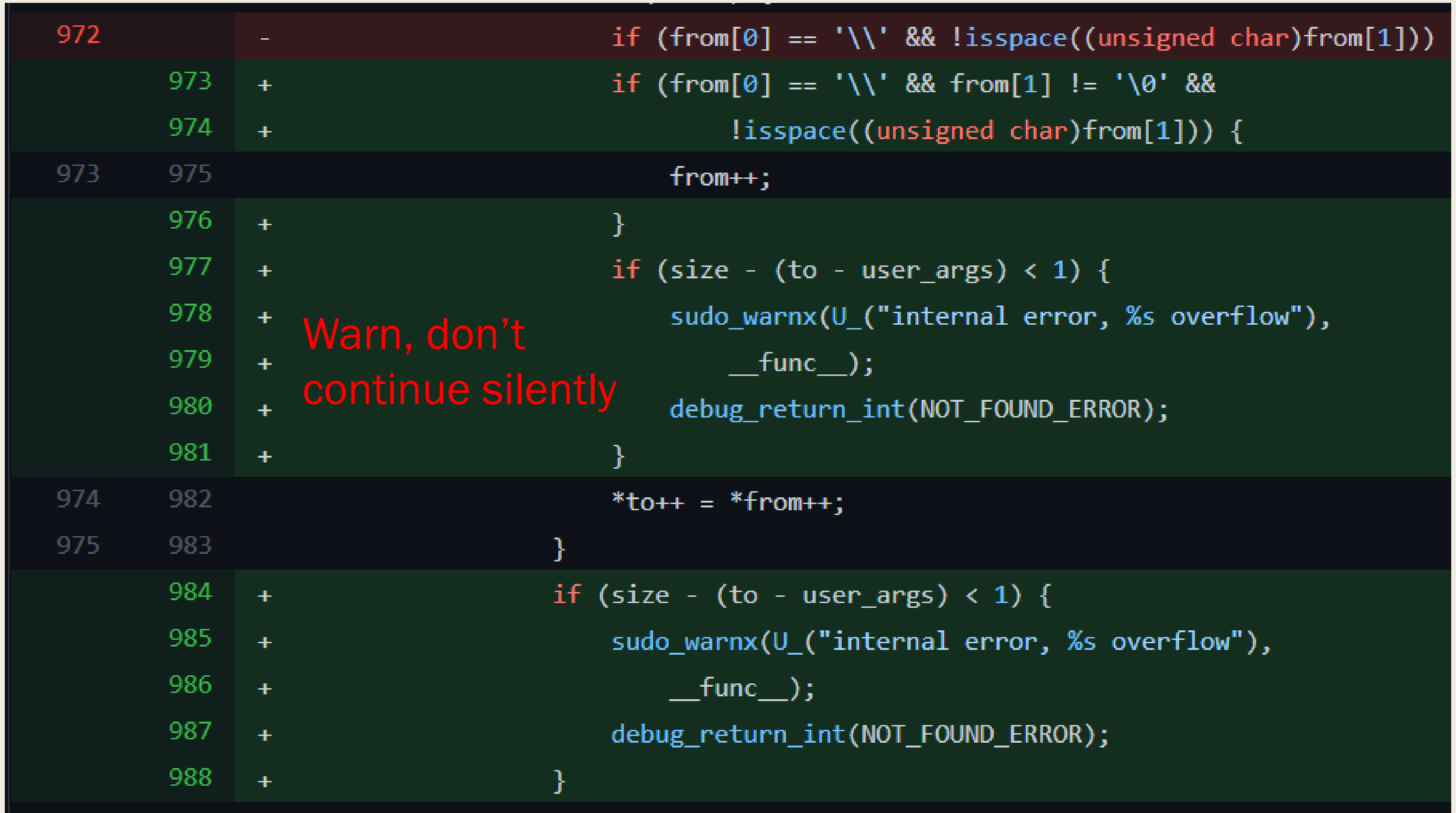
```
Also, do not try to unescaping backslashes unless in run mode *and*
we are running the command via a shell.
Found by Qualys, this fixes CVE-2021-3156.
```

set_cmnd()



```
-        if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL)) {
+        if (ISSET(sudo_mode, MODE_SHELL|MODE_LOGIN_SHELL) &&
+            ISSET(sudo_mode, MODE_RUN)) {
```

set_cmnd()



```
-            if (from[0] == '\\' && !isspace((unsigned char)from[1]))
+            if (from[0] == '\\' && from[1] != '\0' &&
+                    !isspace((unsigned char)from[1])) {
                 from++;
+            }
+            if (size - (to - user_args) < 1) {
+                sudo_warnx(U_("internal error, %s overflow"),
+                    __func__);
+                debug_return_int(NOT_FOUND_ERROR);
+            }
             *to++ = *from++;
         }
+        if (size - (to - user_args) < 1) {
+            sudo_warnx(U_("internal error, %s overflow"),
+                __func__);
+            debug_return_int(NOT_FOUND_ERROR);
+        }
```

Warn, don't continue silently

Don't assume when MODE_SHELL is set, MODE_RUN is also set.

Check it again.

State it explicitly.

Don't assume the buffer is well-formed.

另解: 用 isParsed FLAG 來記錄是否執行過 parse_args()?

假設只存在於開發者腦中，就算成立，未來也很可能被改爛

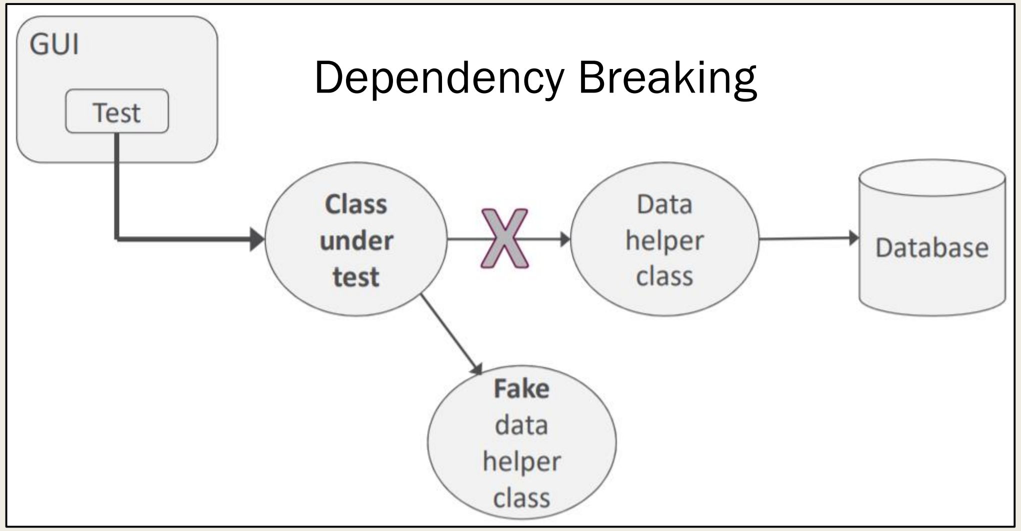
Summary - How to write Secure Code

- No bad implementations
- Find Assumptions and Trusts
  - Take care of Dependencies
  - Aware of implicit Assumptions
- Threats modeling
  - DFD with Trust Boundary
- Security Design Principles
- Secure Coding Standards
  - Use security testing tools
- Good Coding Style

Summary - Good Coding Style

- 高可讀性，Make your code explain itself
  - 清楚的命名語意，e.g., fruitNames, getUser(), row_index, 解釋變量
  - 避免隱含的假設，e.g., TOCTOU, CVE-2021-3156
  - 清楚的程式流程、合理的設計，e.g., class, method, data structure

- 高可維護性，未來擴充或修改功能是否方便
  - 低耦合度 → 高可測試性
  - 使用設計模式 (Design Pattern)



HW

- Survey a CVE (仿照 CVE-2021-3156)
  - 漏洞的根本原因 (Buffer Overflow)
  - 為何沒考慮到這個漏洞 (Flag 設錯，後續的邏輯又依賴這個 Flag)
  - Patch 改了哪些地方 (設定正確的 Flag 、引入解釋變量，set_cmnd() 確保 Flag 是正確的、確保 buffer 是合法的)
  - 其他的 Patch 方法 or Reflection (加一個 isParsed FLAG)

- 上傳 PDF