

Secure Coding Practices



CYBER SECURITY

neosec



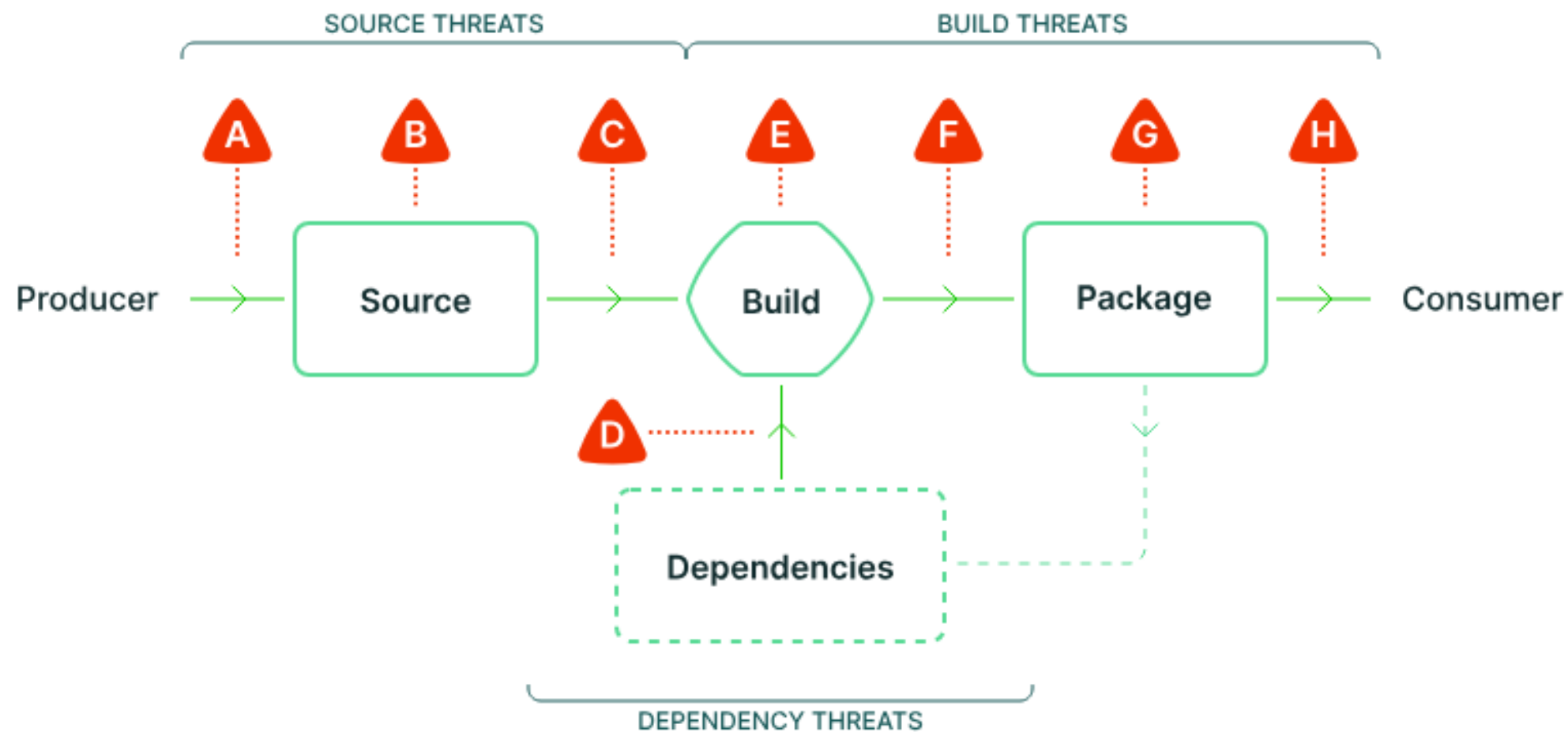
Who is Sven?

Supply Chain Security





Secure Coding



SOURCE THREATS

- A** Submit unauthorized change
- B** Compromise source repo
- C** Build from modified source

DEPENDENCY THREATS

- D** Use compromised dependency

BUILD THREATS

- E** Compromise build process
- F** Upload modified package
- G** Compromise package registry
- H** Use compromised package

Project SLSA - <https://slsa.dev/>

- **Security**
 - **Integrity**
 - **Transparency**
 - **Traceability**
 - **Trust**
- **Software Supply Chain Security**
 - **Trusted Artifacts**
 - **End-to-End Integrity**
 - **Maturity Model (Level 1–4)**
 - **Prevents Supply Chain Attacks**

OWASP - <https://owasp.org/>

- <https://owasp.org/www-project-dependency-track/>
- <https://owasp.org/www-project-top-ten/>
- <https://cyclonedx.org/>
- <https://github.com/CycloneDX/cyclonedx-maven-plugin>

OWASP - <https://owasp.org/>

Input validation

Output validation

Memory Management

Access Control

Cryptographic Practices

Error Handling and Logging

Data Protection

Communication Security

System Configuration

Authentication and Password Management

Database Security

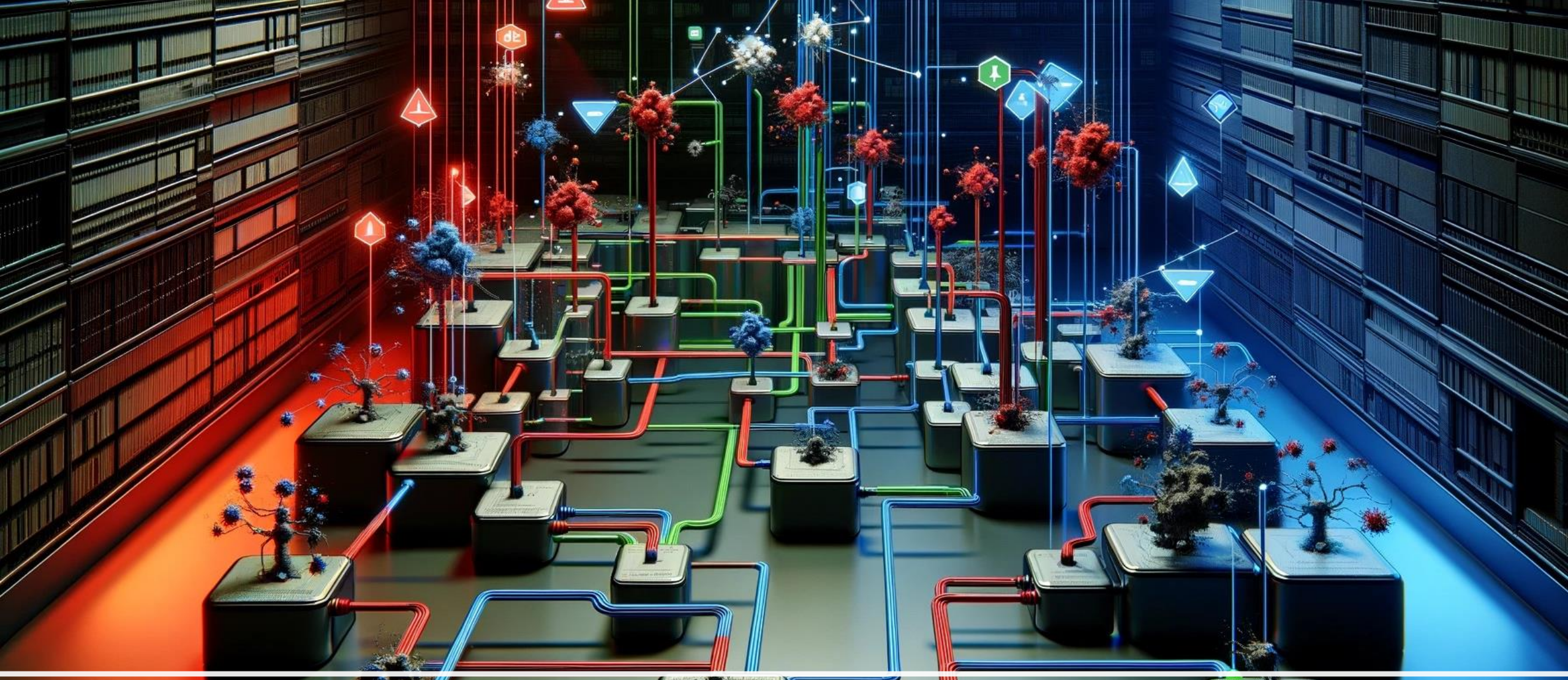
File Management

Session Management

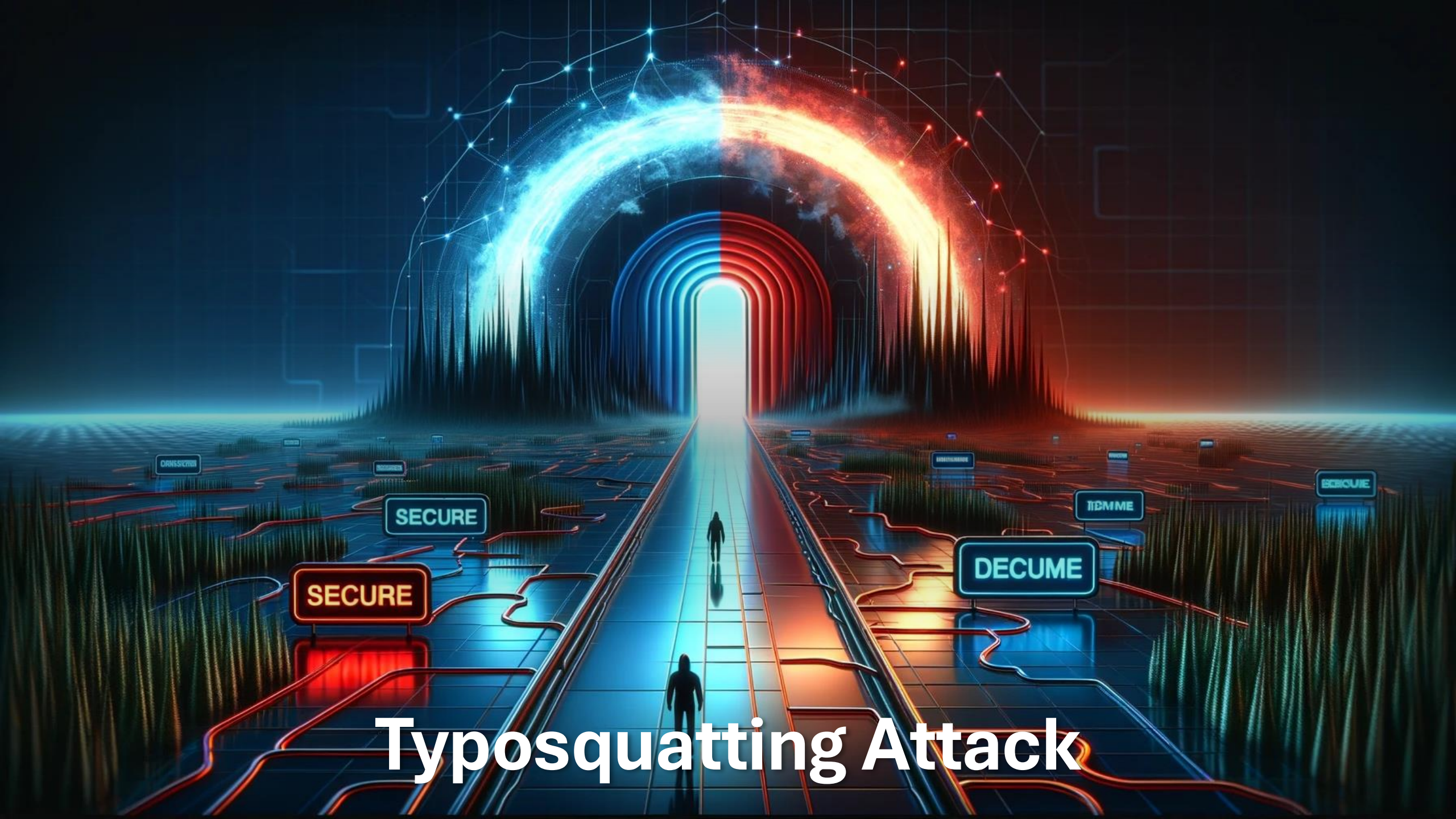
General Coding Practices



Infection Methods



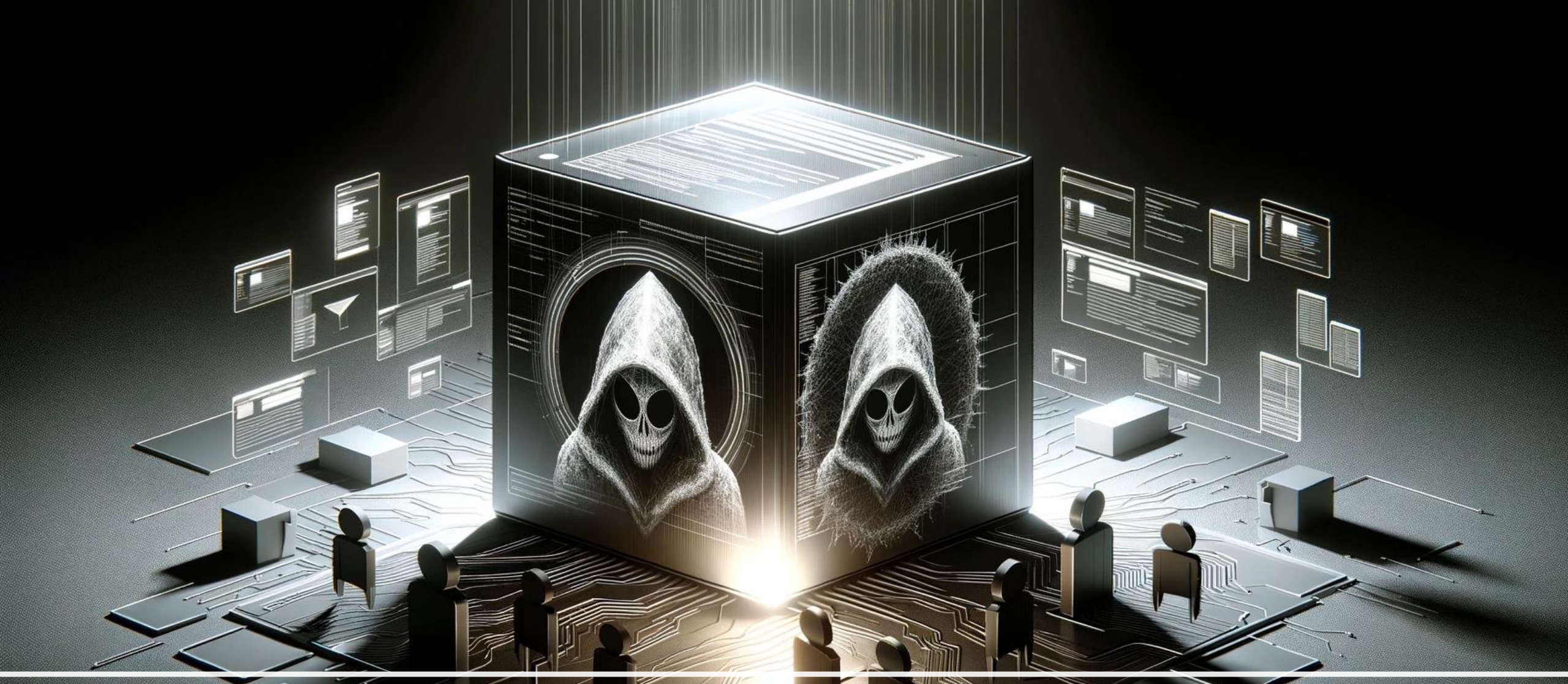
Dependency Confusion Attack



Typosquatting Attack



Masquerading Attack



Trojan Package Attack

Code obfuscation techniques

Off-the-shelf public obfuscators

Pyarmor - <https://github.com/dashingsoft/pyarmor>

obfuscator.io - <https://obfuscator.io/>

Python obfuscation tool - <https://www.development-tools.net/python-obfuscator/>

Custom obfuscation techniques

Mangling variable names

Encrypting strings

Control flow flattening

Invisible backdoors - homoglyphs

Obfuscation techniques

Python obfuscation tool - <https://www.development-tools.net/python-obfuscator/>

Used in “noblesse2” malicious packages we found last year

Encode Python code text with base64

Decode, compile and evaluate the code

```
import base64, codecs
magic = 'cHJpbnQ'
love = 'bVxuyoT'
god = 'xvIHdvc'
destiny = 'zkxVFVc'
joy = '\x72\x6f\x74\x31\x33'
trust = eval('\x6d\x61\x67\x69\x63') + eval('\x63\x6f\x64\x65\x63\x73\x2e\x64\x65\x63\...'
eval(compile(base64.b64decode(eval('\x74\x72\x75\x73\x74')), '<string>', 'exec'))
```

Obfuscation techniques

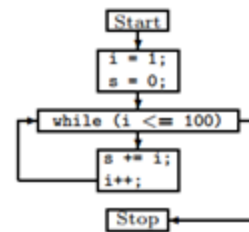
original

```
i = 1;
s = 0;

while (i <= 100) {

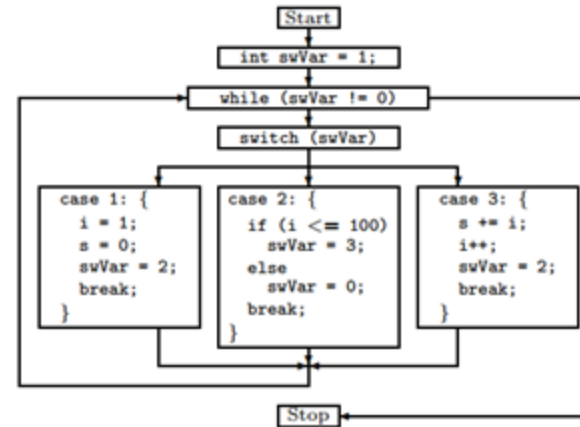
    s += i;
    i++;

}
```



control-flow flattening applied

```
int swVar = 1;
while (swVar != 0) {
    switch (swVar) {
        case 1: {
            i = 1;
            s = 0;
            swVar = 2;
            break;
        }
        case 2: {
            if (i <= 100)
                swVar = 3;
            else
                swVar = 0;
            break;
        }
        case 3: {
            s += i;
            i++;
            swVar = 2;
            break;
        }
    }
}
```



Obfuscation techniques

Unicode homoglyph characters are characters that look like standard ASCII/Latin characters

Can be used by supply-chain attackers to plant "invisible" backdoors

Example: changing a string literal check to make it always succeed / fail

```
void sayHello() {  
    std::cout << "Hello, World!\n";  
}  
  
void sayHello() {  
    std::cout << "Bye, World!\n";  
}
```

Obfuscation techniques

Unicode bidirectional (BiDi) control characters

Normally used to control the flow of text (left-to-right or right-to-left)

Can be used to change the order of the compiled / interpreted source code

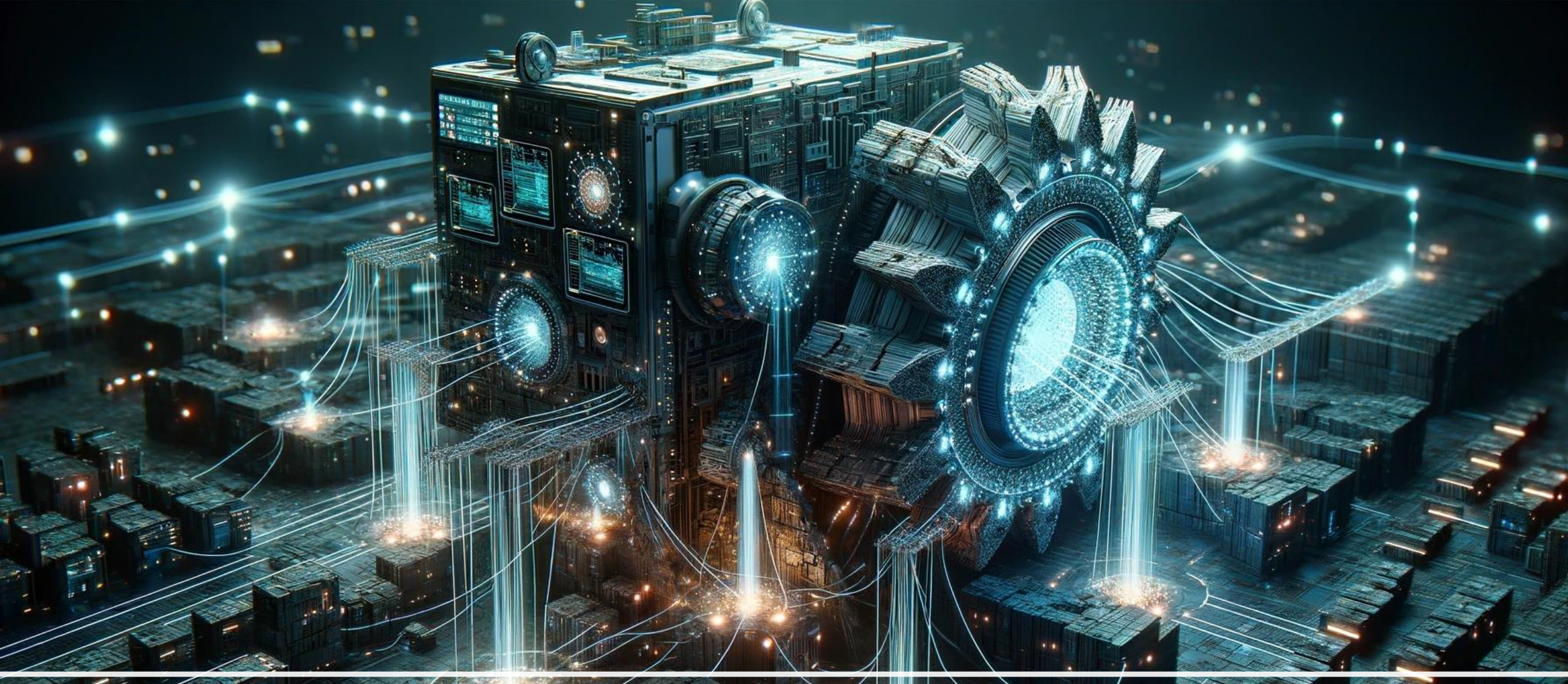
An example - invisibly comment out a condition check

```
int main() {  
    bool isAdmin = false;  
    /* begin admins only */ if (isAdmin) {  
        printf("You are an admin.\n");  
    /* end admins only */ }  
    return 0;  
}
```

```
int main() {  
    bool isAdmin = false;  
    /* begin admins only if (isAdmin) */ {  
        printf("You are an admin.\n");  
    /* end admins only */ }  
    return 0;  
}
```



Security Payloads



Cryptominer

Sensitive Data Stealer

Text Steganography



Image Steganography



Audio Steganography

A futuristic illustration set in space. On the left, a blue, muscular, wireframe-like alien stands holding a microphone, with a glowing blue halo around its head. A bright blue audio waveform extends from the microphone towards the center. In the foreground, a human figure sits in silhouette, facing right, with a glowing pink audio waveform emanating from their ear. A large, glowing blue and green planet (Earth) is in the background, with a pinkish-purple nebula-like structure extending from it. The entire scene is set against a dark blue starry space background.



Video Steganography

SAST



SECURITY
TESTING

IAST
INTERACTIVE
SECURITY
TESTING

INTERACTIVE
APPLICATION
SECURITY
TESTING

INTERACTIVE APPLICATION
SECURITY TESTING

Real time monitoring
while code testing

And Responses

REAL-TIME
MONITORING
SECURITY

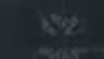
SQJA Injection
while Responses

Hybrid Approach
Combining static and dynamic
security testing

Deep Inspections
and Responses
Hybrid Time Security
Combining static security feedback

IAST

IAST



RASP



The background of the image is a dark, atmospheric library with tall wooden bookshelves filled with books. In the center, a complex network of glowing blue and white nodes connected by thin lines is visible. Several stacks of books are placed on the floor, each with a glowing blue light emanating from it, suggesting a digital or magical energy. The overall scene is dimly lit, with the primary light sources being the glowing nodes and the light from the book stacks.

Common Weakness Enumeration

<https://cwe.mitre.org/>

Common Weakness Enumeration

**Common Weakness Enumeration**
A community-developed list of SW & HW weaknesses that can become vulnerabilities

**New to CWE?**
[Start here!](#)

ID Lookup:

[Home](#) | [About ▼](#) | [Learn ▼](#) | [Access Content ▼](#) | [Community ▼](#) | [Search ▼](#)



Knowing the weaknesses that result in vulnerabilities means software developers, hardware designers, and security architects can eliminate them before deployment, when it is much easier and cheaper to do so

Learn About CWE

Overview – Learn what CWE is and how to use the information available on this website

[Basics](#)
[FAQs](#)
[Glossary](#)

Root Cause Mapping – Learn about identifying the underlying cause(s) of a vulnerability

[Guidance](#)
[Quick Tips](#)
[Examples](#)

Access Content

[All Weaknesses \(944 total\)](#)
[Top-N Lists](#)

Search CWE

ENHANCED BY 

View CWEs by

[Software Development](#)

[Hardware Design](#)

[All Weaknesses](#)

[Other Select Options](#)

Contribute

[Contribute CWE Content](#)
[Participate in Working Groups](#)

Latest News and Updates

News [CWE Version 4.18 Now Available!](#)

News ["2025 CWE™ Most Important Hardware Weaknesses" Now Available](#)

Podcast [Mapping CVEs to CWEs Is Main Topic of "We Speak CVE" Podcast](#)

News [Videos of Three CWE-Focused Sessions at VulnCon 2025 Now Available](#)

Podcast ["Root Cause Mapping and the CWE Top 25"](#)

News ["2024 CWE Top 10 KEV Weaknesses" List Now Available](#)

<https://cwe.mitre.org/>

Common Weakness Enumeration

699 - Software Development

-   API / Function Errors - (1228)
-   Audit / Logging Errors - (1210)
-   Authentication Errors - (1211)
-   Authorization Errors - (1212)
-   Bad Coding Practices - (1006)
-   Behavioral Problems - (438)
-   Business Logic Errors - (840)
-   Communication Channel Errors - (417)
-   Complexity Issues - (1226)
-   Concurrency Issues - (557)
-   Credentials Management Errors - (255)
-   Cryptographic Issues - (310)
-   Key Management Errors - (320)
-   Data Integrity Issues - (1214)
-   Data Processing Errors - (19)
-   Data Neutralization Issues - (137)
-   Documentation Issues - (1225)
-   File Handling Issues - (1219)
-   Encapsulation Issues - (1227)
-   Error Conditions, Return Values, Status Codes - (389)
-   Expression Issues - (569)
-   Handler Errors - (429)
-   Information Management Errors - (199)
-   Initialization and Cleanup Errors - (452)
-   Data Validation Issues - (1215)
-   Lockout Mechanism Errors - (1216)
-   Memory Buffer Errors - (1218)
-   Numeric Errors - (189)
-   Permission Issues - (275)
-   Pointer Issues - (465)
-   Privilege Issues - (265)
-   Random Number Issues - (1213)
-   Resource Locking Problems - (411)
-   Resource Management Errors - (399)
-   Signal Errors - (387)
-   State Issues - (371)
-   String Errors - (133)
-   Type Errors - (136)
-   User Interface Security Issues - (355)
-   User Session Errors - (1217)

<https://cwe.mitre.org/>

Common Weakness Enumeration

-   Documentation Issues - (1225)
-   File Handling Issues - (1219)
 -  Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal') - (22)
 -  Improper Resolution of Path Equivalence - (41)
 -  Improper Link Resolution Before File Access ('Link Following') - (59)
 -  Improper Handling of File Names that Identify Virtual Resources - (66)
 -  Creation of Temporary File With Insecure Permissions - (378)
 -  Creation of Temporary File in Directory with Insecure Permissions - (379)
 -  Untrusted Search Path - (426)
 -  Uncontrolled Search Path Element - (427)
 -  Unquoted Search Path or Element - (428)
-   Encapsulation Issues - (1227)

<https://cwe.mitre.org/>

Common Weakness Enumeration

CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')

Weakness ID: 22
Vulnerability Mapping: ALLOWED
Abstraction: Base

View customized information:

Conceptual

Operational

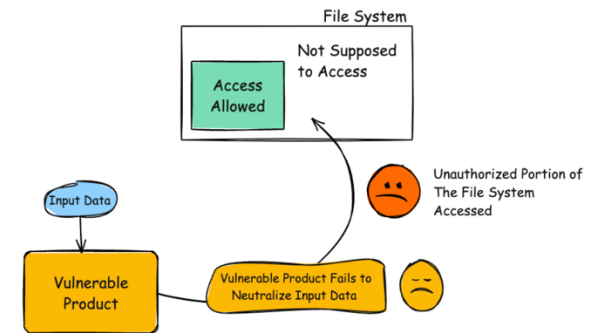
Mapping
Friendly

Complete

Custom

▼ Description

The product uses external input to construct a pathname that is intended to identify a file or directory that is located underneath a restricted parent directory, but the product does not properly neutralize special elements within the pathname that can cause the pathname to resolve to a location that is outside of the restricted directory.



▼ Extended Description

Many file operations are intended to take place within a restricted directory. By using special elements such as "." and "/" separators, attackers can escape outside of the restricted location to access files or directories that are elsewhere on the system. One of the most common special elements is the "../" sequence, which in most modern operating systems is interpreted as the parent directory of the current location. This is referred to as relative path traversal. Path traversal also covers the use of absolute pathnames such as "/usr/local/bin" to access unexpected files. This is referred to as absolute path traversal.



Common Vulnerability Scoring System

www.first.org/cvss/

Exploit Prediction Scoring System

A detailed illustration of a wizard council in a grand, dark cathedral. The council members, wearing blue robes and pointed hats, are seated in a circle on ornate chairs. They are gathered around a large, circular stone floor marked with intricate magical symbols and runes. In the center of the circle, a glowing blue orb with a white symbol is suspended in the air. Above the council, numerous colorful, glowing spheres of various sizes and patterns float in the air, connected by faint lines. The cathedral's architecture features high, arched windows and columns, with a large, ornate stained-glass window at the top. The overall atmosphere is mysterious and magical.

www.first.org/epss/



Penetration Testing

Fuzzing



TDD and Security





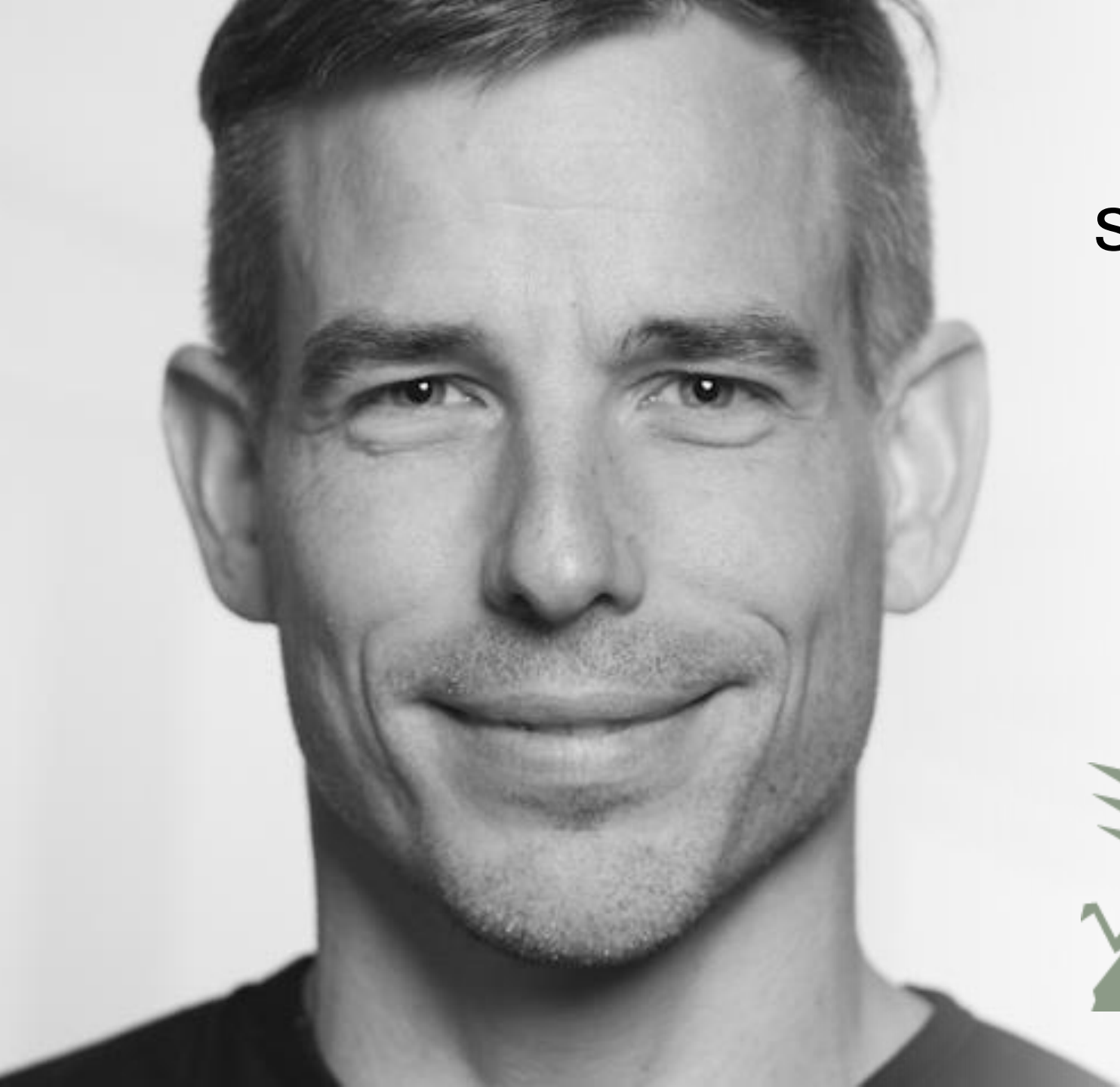
TEST COVERAGE



DEPENDENCIES



VULNERABILITIES



sven@neosec-it.com



CYBER SECURITY

neosec