Bruno Bossola, CTO
bruno@meterian.io

Supply chain risks in software development

"We are not that smart"

@bbossola

My Self

- Developer 1988+
- XP coach 2000+
- Java Champion since 2005
- CTO/co-founder @meterian.io
- Passionate about security

My Company

- UK company
- Founded in 2018
- SCA solution
- 276k+ vulnerabilities tracked
- 4m+ analysis performed

meterian

www.meterian.io

Agenda

This is an informational session to understand supply chain risks in software development.

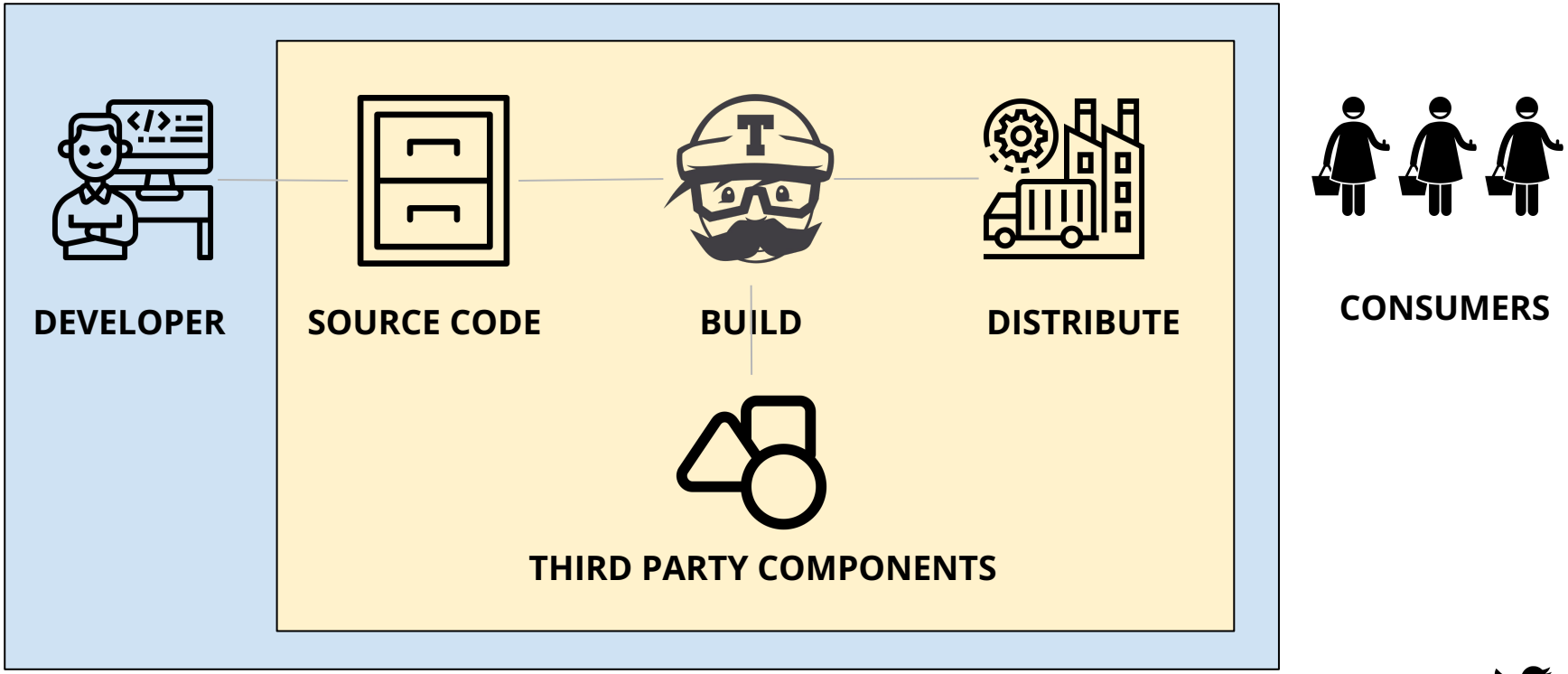
I will tell you about:

- What is the supply chain in software development?
- Common supply chain risks
  - Techniques for managing supply chain risks
  - Best practices for developers
- Is there an industry standard?

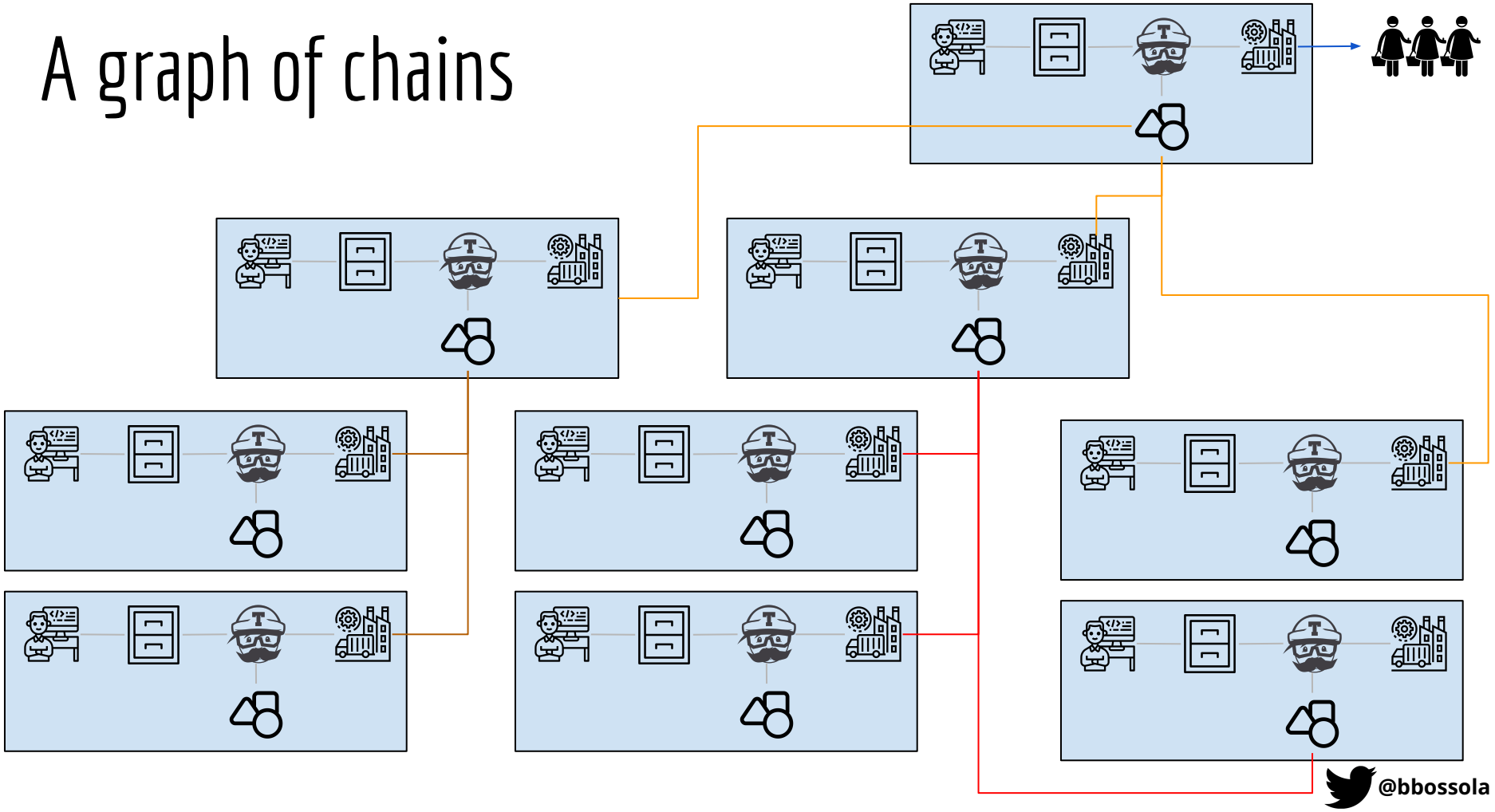
The supply chain in software development

@bbossola

Simplified supply chain

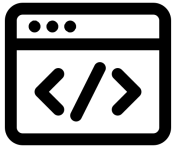


A graph of chains



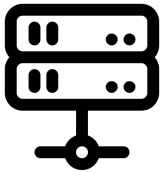
@bbossola

SD supply chain components in details



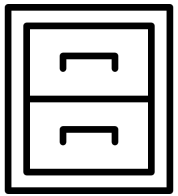
Source code

Build tools

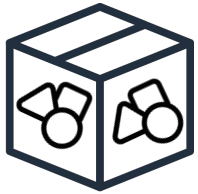


Infrastructure and Hosting

Development tools

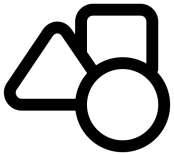


Code repositories



Package repositories

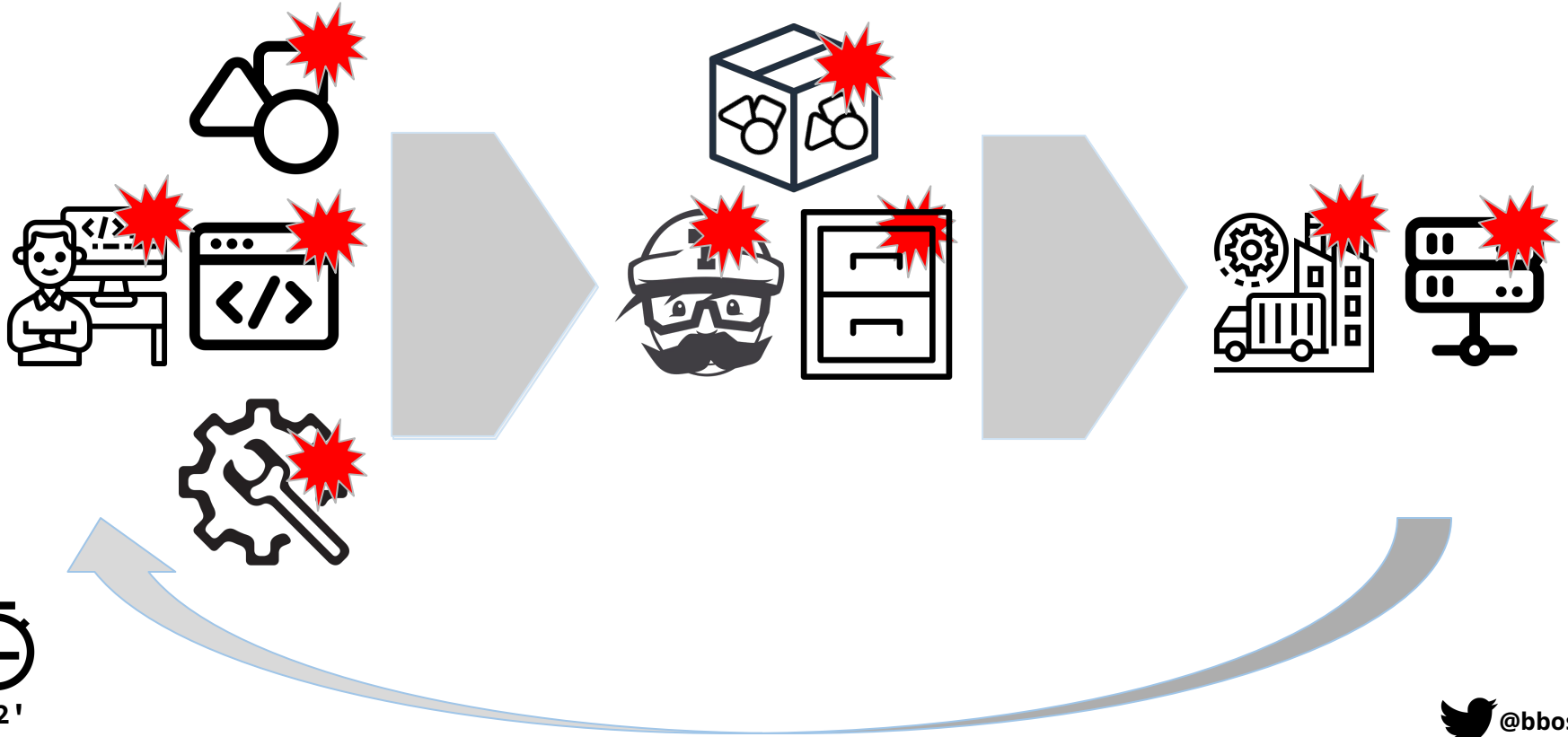
Developers



Third party libraries

Deployment, Distribution, Maintenance

SD supply chain in operation



12'

Software development supply chain risks

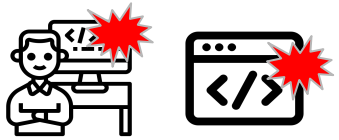
@bbossola

Software development supply chain main risks

- Malicious code injection
- Third-Party components
- Compromised tools
- Lack of vendor security practices *(we won't drill into this one)*

- **650%** surge in OSS supply chain attacks (Sonatype)
- **84%** commercial code bases have OSS vulnerabilities (Synopsys)
- **45%** of orgs will experience supply chain attacks in 2025 (Gartner)

Malicious code injection



Inclusion of malicious code, malware, or backdoors in software components or libraries during the development process.

Solarwinds

September 2020

- Major US software company which provides system management tools for network and infrastructure monitoring
- SolarWinds Orion, an IT monitoring system, has privileged access to IT systems and it's widely deployed on all infrastructure
- Hackers compromising the infrastructure of SolarWinds
- A DLL is trojanized and then digitally signed
- The DLL is then distributed by Solarwinds in a normal update
- After being dormant for two weeks, the trojan activates
- Sends traffic masquerading it as an internal Orion protocol
- Uses sophisticated obfuscation techniques

Solarwinds

September 2020

- The attack affected a wide range of sectors, including government agencies, technology companies, and critical infrastructure providers
- More than 18,000 SolarWinds customers installed the malicious updates, with the malware spreading undetected around the world
- Went undetected for 14 months

Solarwinds

September 2020

Prevention?

- proper security posture via INFOSEC policies
- active monitoring of infrastructure, patching policies and processes
- artifacts fingerprinting
- CI/CD mirroring

GitlLab Backdoor

April 2022

Merged **JH need more complex passwords** `memorycancel-master-patch-…` into `master`

Overview 32 | Commits 1 | Pipelines 6 | **Changes 34**

lib/gitlab/auth/o_auth/user.rb

```
@@ -230,8 +230,8 @@ def user_attributes
230 230            name:                     name.strip.presence || valid_username,
231 231            username:                 valid_username,
232 232            email:                    email,
233     -          password:                 auth_hash.password,
234     -          password_confirmation:    auth_hash.password,
    233 +          password:                 Gitlab::Password.test_default(21),
    234 +          password_confirmation:    Gitlab::Password.test_default(21),
235 235            password_automatically_set: true
236 236          }
237 237        end
```

@bbossola

GitlLab Backdoor

April 2022

- The commit was coming from a side project (JiHu) managed in China that was subsequently merged into the main GitLab codebase
- The vulnerability was disclosed as CVE-2022-1162
- GitLab released patched versions which fix the issue and provided a script for detection
- Password resets were sent across to users

GitlLab Backdoor

April 2022

Prevention?

- secret scanning automation
- rigorous review process
- better distinction between test and source files
- MFA on all developer accounts

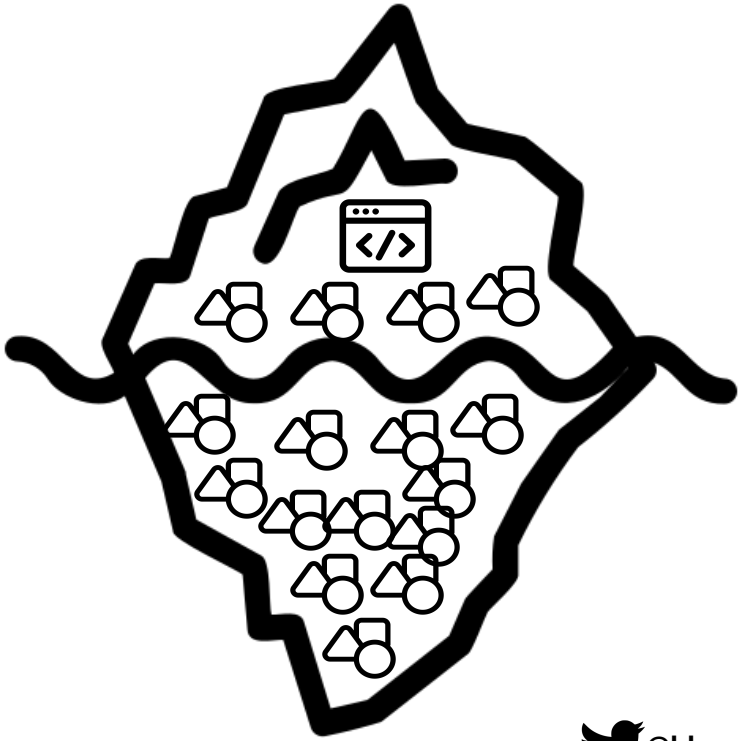
Third-Party components

In modern software development, more than 90% of the code is now composed of third-party components, typically OSS

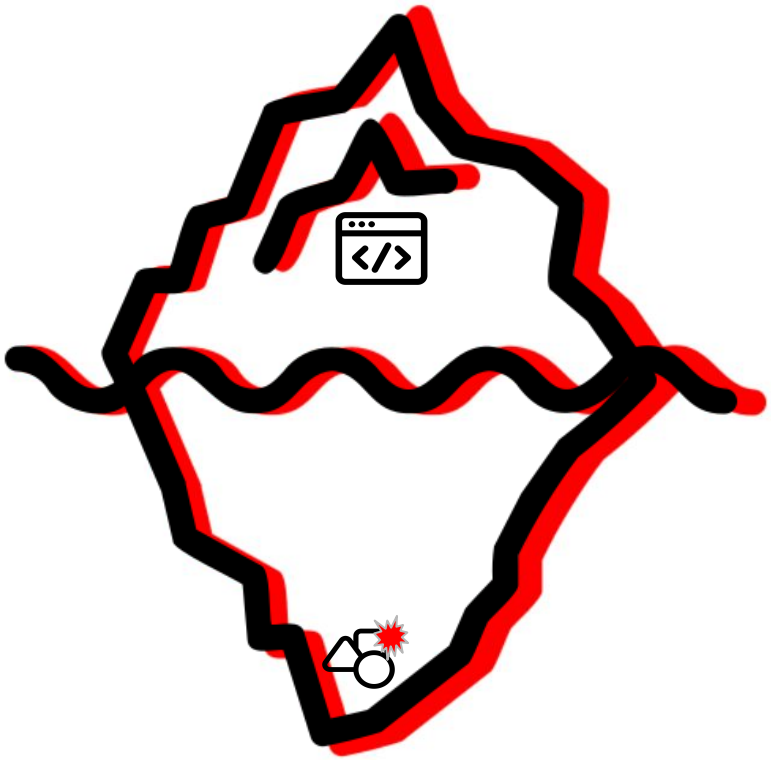
Third-Party components

Your code accounts typically to 10% / 15% of the code packaged into your application.

Most of it it's someone else's code.



Vulnerable components

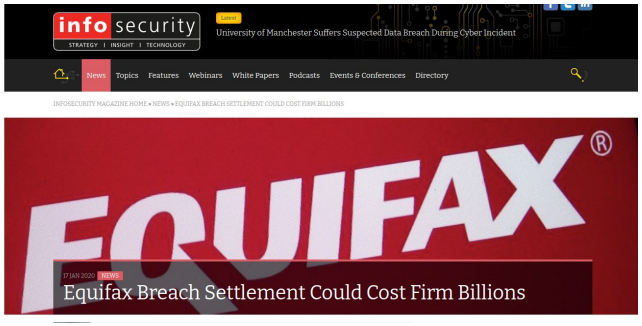


A single vulnerable component can make your packaged application exploitable

- Every day more than 20 new vulnerabilities appear on the scene in OSS components.
- Components source code is public
- Malicious actors know this well

Equifax Megabreach

July 2017



A vulnerability in the Apache Struts library was exploited simply by passing a specially crafted HTTP header (CVE-2017-5638)

- library unpatched for 2 months
- 140M customers affected
- cost equal to 40% of annual revenue
- total cost probably in the billions of dollars
- At least other 7 (known) successful exploits using the same vulnerability

Unmaintained components

Outdated, unpatched or unmaintained components can rot the entire application

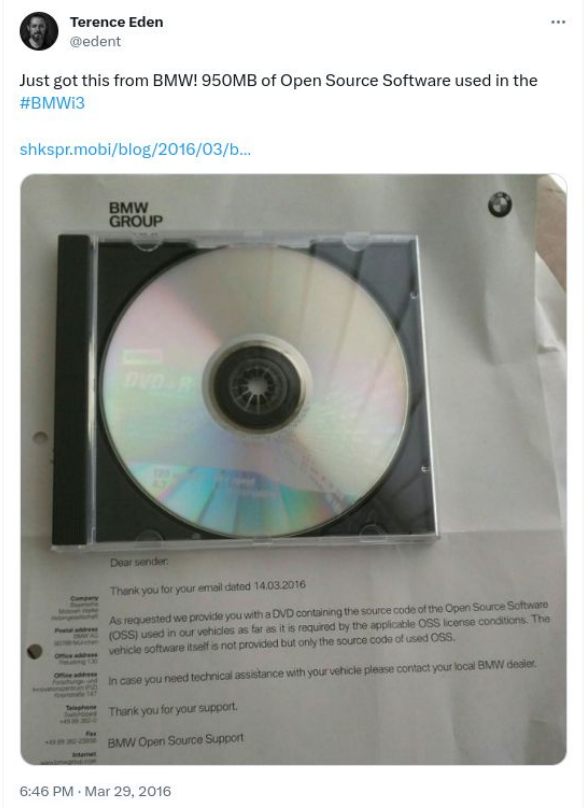
- They may contain vulnerabilities
- They may contain unsolved bugs
- As they are not looked after, they can become very dangerous

Wrongly licensed components

A single CopyLeft licensed components can make the whole application CopyLeft as well.

- it can be a deeply nested component
- your whole application is "infected"
- some license are infectious over the network (i.e. AGPL)
- you may need to release your code to comply with the license

BMW i3 code ...and others!



IBM sent Terence Eden a DVD containing proprietary code used on the i3 model

Terence put the code on GitHub the next day :)

Why? He asked for it, as the code was including GPL licensed components

Progress

Undisclosed settlement

Westinghouse

Filed for Bankruptcy

Panasonic

$100M settlement

@bbossola

Third-Party components

How could you prevent this?

- Implement a robust Software Composition Analysis solution as part of your build process
- Introduce license compliance and evaluate risks
- Validate the provenance of all components

meterian

Compromised tools: IDE

Your IDE may not be safe: malicious extensions and plugins can create havoc.

Visual studio code extensions

Run with the privileges of the user, no sandbox, built on Electron. And it's all written in Node! (more on this later

- some real extensions use an embedded web server to operate
  - in presence of vulnerable code this can be used to access the developer machine
  - multiple real world extensions are open to this exploit
  - reported by Snyk, several plugins were found vulnerable to different exploits

LaTex Workshop
(command injection via Websocket)

Open In Default Browser
(path traversal)

Rainbow Fart
(zip path traversal)

Visual studio code extensions

Impersonation!

- it's possible to impersonate a popular extensions
  - typosquatting is used to appear almost like the real extension
  - displayed name can be an exact match of a highly popular one
  - verification on the marketplace means the author owns the domain
  - famous example by aquasec

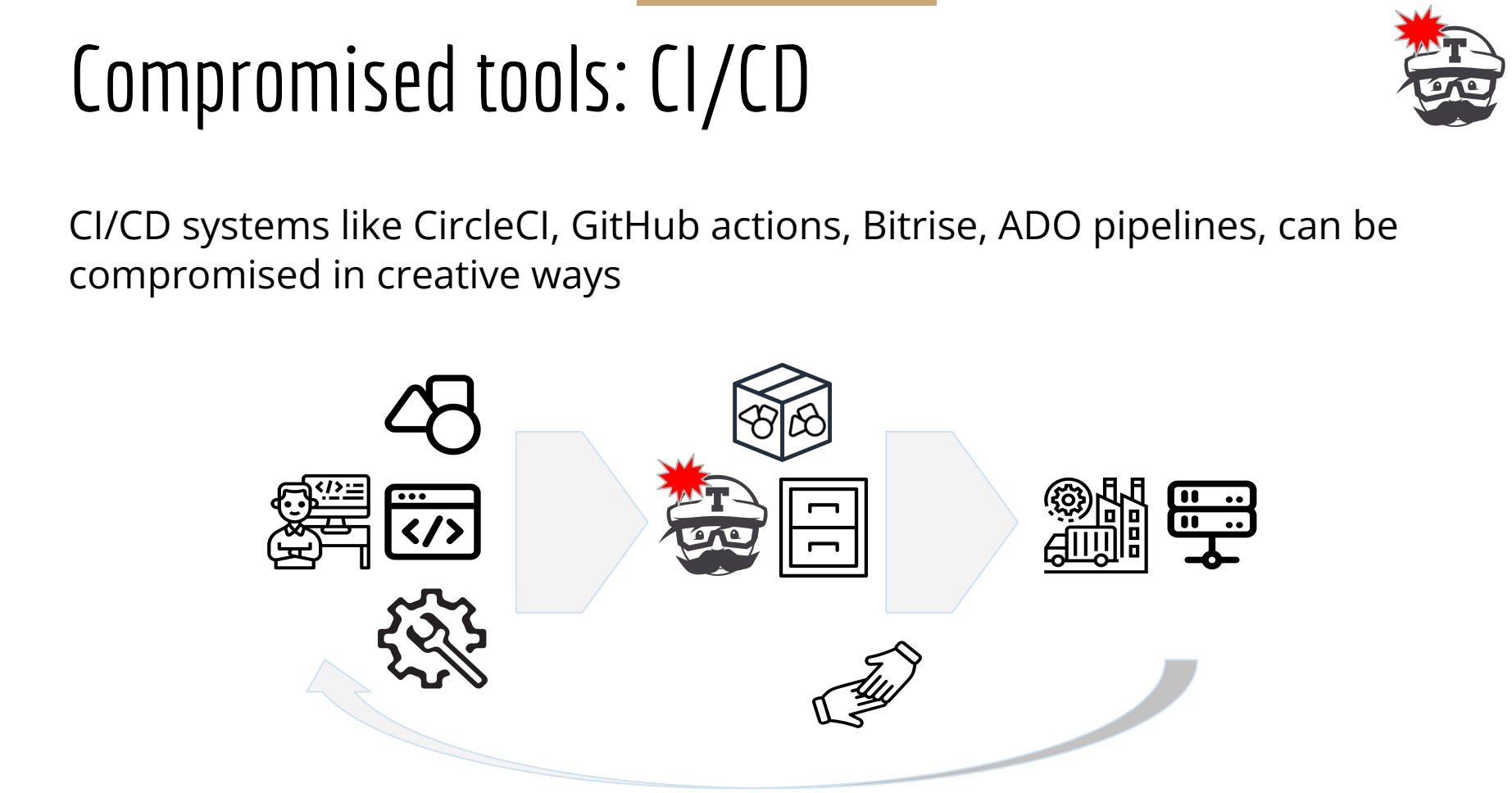
Prettier

Visual studio code extensions

How could you prevent this?

- unfortunately there's no mechanism to vet extensions
- make sure you they come from a reputable source
- always double check what you are installing, be aware of impersonation
- disable auto update of extensions
- implement your own vetting process

Compromised tools: CI/CD



CI/CD systems like CircleCI, GitHub actions, Bitrise, ADO pipelines, can be compromised in creative ways

CodeCov

April 2021

Attackers managed to gain access to the Bash Uploader script and altered it without being caught.

- attackers were able to collect sensitive information undetected
- for two months
- detected only after the difference between the SHA fingerprint of the script present on the website and the one on GitHub was reported

CodeCov

How could you prevent this?

- always validate checksums
- use encrypted secrets management for safe credential storage
- avoid storing secrets in code (git-secrets)
- rotate keys programmatically and establish a triggerable process

Compromised tools: code repository

A source code repository can be compromised, as we see before with GitLab.

GitHub breach

July 2022

Attackers managed to exfiltrate encrypted code signing certificates pertaining to some versions of GitHub Desktop for Mac and Atom apps.

- attackers could produce valid and digitally signed version of GitHub apps
- 10 versions of GitHub desktop were affected, 2 versions of Atom
- Atom was subsequently discontinued (whoopsie)

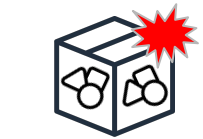
Two Digicert code signing certificates used for Windows and one Apple Developer ID certificate were then set for revocation on February 2, 2023.

Incident

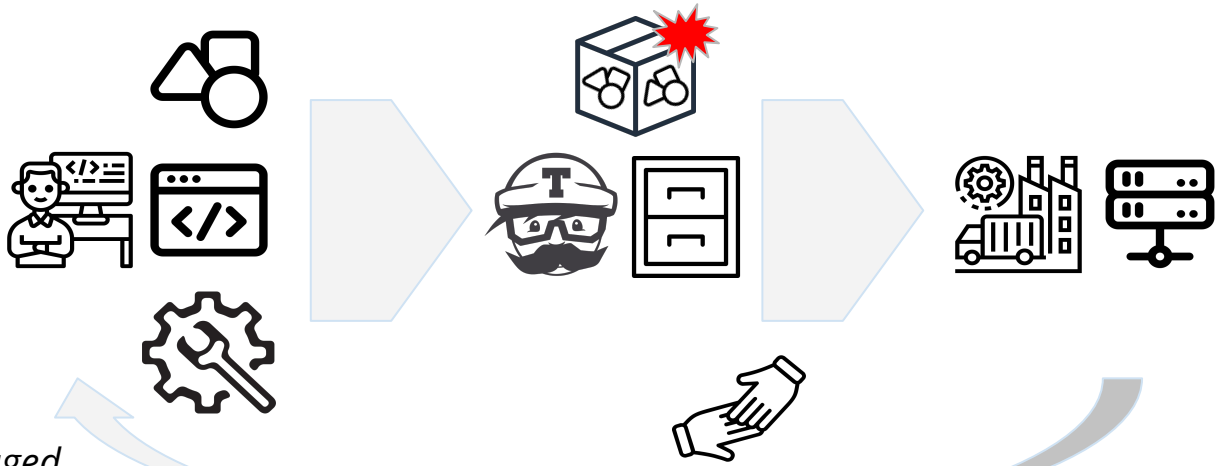
Prevention?

- have a way to revoke your signing certificate
- overall better general security posture
  - a PAT token was stolen from a not-so-well protected machine
  - do not let your PAT go loose
  - especially the ones controlling repositories where you store your signing certificates :)
- add additional validation steps to the pipelines

Compromised tools: package repositories



Package can be compromised on package repositories (npmjs.com, nuget.org, rubygems.org)



Sometimes sabotaged by their own maintainers! ('colors' and 'faker', NPM)

The RubyGems incident

August 2019

Attackers gained unauthorized access to a RubyGems.org account and uploaded a malicious version of the popular "strong_password" gem

- widely used by Ruby developers to enforce password strength
- widely used in Ruby gems (libraries)

Developers who unknowingly installed the compromised gem could inadvertently introduce a security vulnerability into their applications.

The same happened later with another gem, "rest-client", which was altered to fetch malicious code from a pastebin to be executed on the server. This gem was also used by other gems, that had to also patched.

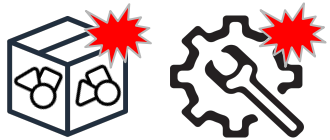
The RubyGems incident

Prevention?

- always use a lockfile
- validate dependencies update
- we need a way to assert the provenance of an artifact
- we need a way to verify that the artifact was built as expected
- possibly some kind signing and validation

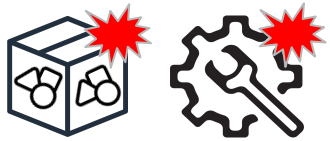
This is currently not available by any package manager, so it's something you would need to implement yourself

Compromised tools: combined attacks



A graph of supply chains supports sophisticated attacks

VS Code event-stream incident

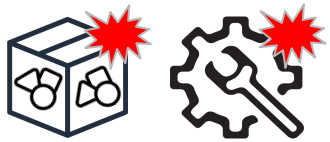


An attacker gained control of an NPM account associated with a popular package named 'event-stream'.

- a new maintainer takes over
- adds a (reasonable) dependency to a new package 'flatmap-stream'
- re-implements the functionality and "forgets" to remove 'flatmap-stream'
- later, adds a bitcoin-wallet stealing to 'flatmap-stream' as a patch version
- in the meantime, 'event-stream' have been distributed to end users via a number of VSCode extensions

A number of wallets are stolen from blissfully unaware developers

VS Code event-stream incident



Prevention?

- always use a lockfile
- validate dependencies updates
  - it's going to be very difficult when the issue is in a transitive dependency

Compromised tools: distribution

Malware can be delivered exploiting software distribution systems

JRE distribution incident (CVE-2017-3272)

JRE (Java Runtime Environment) update system was exploited to distribute malware on the end-users computer

- once installed on a machine the JRE can update itself
- the update system verifies that the received update is signed
- unfortunately, affected versions of the JRE trusted *any* certificate used for the signature (whoopsie!)

This particular exploit required active participation of a user

JRE distribution incident (CVE-2017-3272)

Prevention?

- we need a way to formally validate the provenance of the package
- current proprietary signing systems can be compromised

Prevention?

What we learned so far...

- proper security posture via INFOSEC policies
- active monitoring of infrastructure, patching policies and processes
- artifacts fingerprinting
- CI/CD mirroring
- SCA solution in the pipelines
- public signing processes
- rigorous review process

...plus some other strategies

- Apply least privilege permissions (especially in CI/CD tasks)
- IP address safelisting / Zero Trust Networking
- Assess vendors, train them if required
- MFA everywhere
- Use honeytokens / honeypots

Tools?

git-secrets

Scans commits, commit messages, and merges to prevent adding secrets into your git repositories.

- configured as a commit hook
- can also be used as a on-demand scanner
- costs 0$
- can be customised

QUICK DEMO

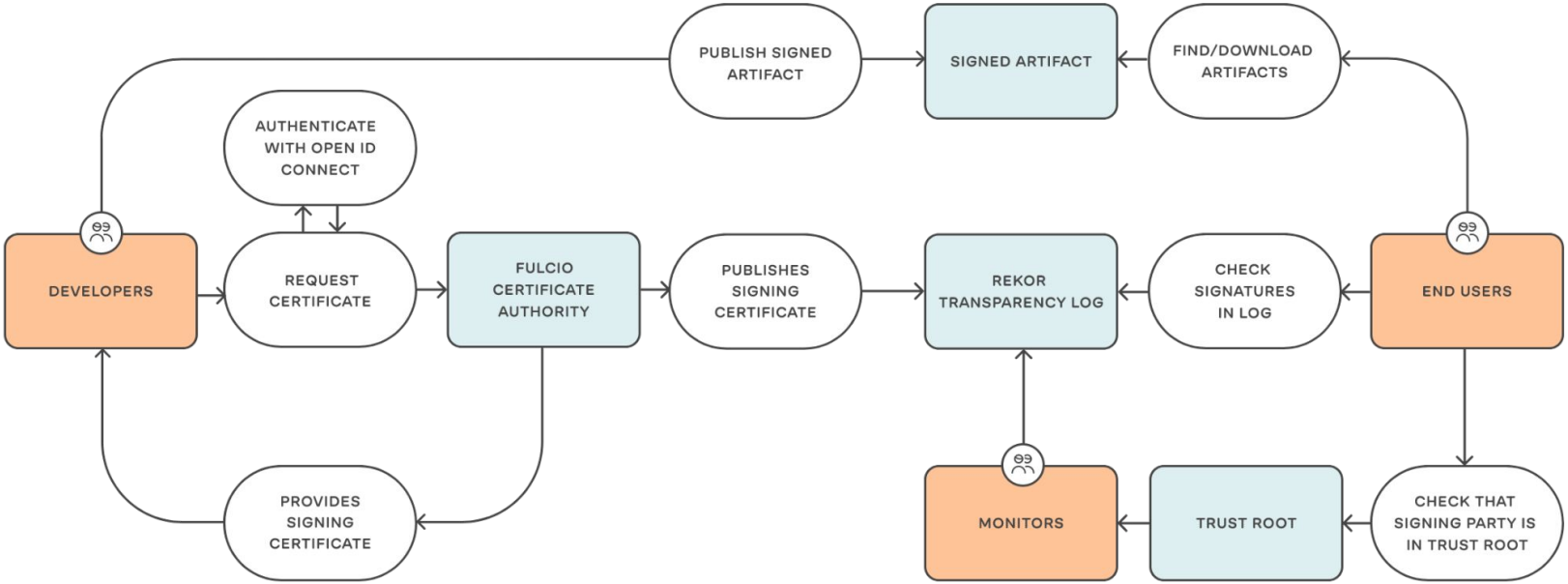
in-toto

A framework to secure the integrity of software supply chains

- tooling is available for many languages (python, go, java, rust)
- allows recording and validation of:
  - what steps were performed
  - by whom
  - in what order
- each step can be recoded, signed and later validated
  - clone
  - changes
  - builds
  - packaging

sigstore

Securely sign software artifacts with signatures stored in a tamper-resistant public log. OpenID can be used to generate ephemeral keys.



@bbossola

Industry standard?

Enter **SLSA**

Supply-chain **L**evels for **S**oftware **A**rtifacts

- a framework for Supply Chain Security
- ensure integrity of software artifacts throughout the supply chain
- inspired by Google internal "Binary Authorization for Borg"
- structured in tracks and levels
  - atm only the build track is released in spec 1.0
- establish the provenance of each component

OpenSSF

OPEN SOURCE SECURITY FOUNDATION

SLSA Principles

- Trust platforms, verify artifacts
- Trust code, not individuals
- Prefer attestations over inferences

SLSA build track

- increasing levels of trustworthiness and completeness in a package artifact's provenance
- enable verification that the artifact was built as expected.
- Each ecosystem or organization defines exactly how this is implemented
  - what provenance format is accepted
  - whether reproducible builds are used
  - how provenance is distributed
  - when verification happens
  - what happens on failure.

This is not fluff, there are actually OSS tools to help :)

SLSA build track levels

Each level provide increasing supply chain security guarantees

Level	Requirements	Focus
L0	None	
L1	Provenance showing how the package was built	Mistakes, documentation
L2	Signed provenance, generated by a hosted build platform	Tampering after the build
L3	Hardened build platform	Tampering during the build

Build L1 - Provenance exists

Package has provenance showing how it was built. Can be used to prevent mistakes but is trivial to bypass or forge.

- the build platform automatically generates provenance
- the software producer follows a consistent build process
- the software producer distributes provenance to consumers

Note that provenance may be incomplete and/or unsigned at L1.

Build L2- Hosted build platform

Builds run on a hosted platform that generates and signs the provenance.

- the build runs on a hosted build platform
- the hosted platform generates and signs the provenance itself
- the authenticity of the provenance can be validated
- the provenance cannot be easily falsified

Build L3 - Hardened builds

Builds run on a hardened build platform that offers strong tamper protection.

- prevent runs from influencing one another
- secret material used to sign the provenance is not accessible to the user-defined build steps
- prevents tampering during the build
- strong confidence that the package was built from the official source and build process.

SLSA

(micro)

DEMO

Anything else in the industry? (that I know of)

TACOS - Trusted Attestation and Compliance for Open Source

- Developed by Tidelift
- A framework for assessing the development practices of open source projects against a set of secure development standards specified by the NIST Secure Software Development Framework (SSDF)
- Defines a machine-readable specification vendors can use as a part of their overall self-attestation paperwork compliance

What about SBOMs?

- a SBOM identifies all the components that are part of an application
- provide transparency about the components of a software
- fashionable since Biden executive order
- available in different and confusing formats :)

When no signed provenance is associated with them, they need to be trusted. SLSA can provide that :) and you can provide SBOMs for every part of the local supply chain

Bruno Bossola, CTO
bruno@meterian.io

Q&A

meterian

www.meterian.io

@bbossola