I’m Jon 👋
@jonhoo on the internet;
formerly MIT, now Rust at AWS
All the software you use matters.

Not just “is the code insecure”, but *could* it be insecure/manipulated.
Supply Chain Attacks are increasing

Sonatype 8th State of the Software Supply Chain (2022)
https://www.sonatype.com/state-of-the-software-supply-chain/
European Union Agency for Cybersecurity (ENISA) was European Network and Information Security Agency. The agency is responsible for providing guidance on cybersecurity threats and challenges. The table of contents from the document includes:

### TABLE OF CONTENTS

2. INTRODUCTION 6
   2.1 BACKGROUND 6
   2.2 PURPOSE OF THIS EXERCISE 6
   2.3 TARGET AUDIENCE 7

3. EMERGING CYBERSECURITY THREATS FOR 2030 8
   3.1 SUPPLY CHAIN COMPROMISE OF SOFTWARE DEPENDENCIES - #1 11
   3.2 ADVANCED DISINFORMATION / INFLUENCE OPERATIONS (IO) CAMPAIGNS - #2 13
   3.3 RISE OF DIGITAL SURVEILLANCE AUTHORITARIANISM / LOSS OF PRIVACY - #3 13
   3.4 HUMAN ERROR AND EXPLOITED LEGACY SYSTEMS WITHIN CYBER-PHYSICAL ECOSYSTEMS - #4 14

According to the EU, top threat in the next 7 years
Stricter supply chain security rules in the EU

Japanese parliament passed an economic security bill Wednesday aimed at guarding technology and reinforcing critical supply chains, while also imposing tighter oversight of Japanese firms working in sensitive sectors or critical infrastructure.

Measures in the legislation, which is primarily aimed at warding off risks from China, will be implemented over two years once it is enacted, according to the bill. It comes after the United States imposed restrictions on technology imports, such as on semiconductors, amid growing tension with Beijing.
Improving the Nation's Cybersecurity

A Presidential Document by the Executive Office of the President on 05/17/2021

Sec. 4. Enhancing Software Supply Chain Security. (a) The security of software used by the Federal Government is vital to the Federal Government’s ability to perform its critical functions. The development of commercial software often lacks transparency, sufficient focus on the ability of the software to resist attack, and adequate controls to prevent tampering by malicious actors. There is a pressing need to implement more rigorous and predictable mechanisms for ensuring that products function securely, and as intended. The security and integrity of “critical software”—software that performs functions critical to trust (such as affording or requiring elevated system privileges or direct access to networking and computing resources)—is a particular concern. Accordingly, the Federal Government must take action to rapidly improve the security and integrity of the software supply chain, with a priority on addressing critical software.

https://www.federalregister.gov/documents/2021/05/17/2021-10460/improving-the-nations-cybersecurity
New supply chain security guidance; no laws (yet)

New ‘supply chain mapping’ guidance

The latest addition to the NCSC’s suite of supply chain guidance is now available.

Supply chain mapping is the process of recording, storing and using information gathered from suppliers who are involved in a company’s supply chain. Building on our existing supply chain guidance, we’re pleased to announce new guidance that focuses explicitly on this process, aimed at procurement specialists, risk managers and cyber security professionals.

Supply chain mapping follows the principles of all good risk management; organisations need to understand the risks inherent in...
When you have a moment:

https://www.sonatype.com/resources/vulnerability-timeline
The scariest part: many aren't even aware

(from Sonatype)
Do you know:

(the answer better be yes)
((but it probably isn’t))

What you are deploying where?
Where it came from?
What’s in it?
Do you know:

(the answer better be yes)
((but it probably isn’t))

What you are deploying where?
Where it came from?
What’s in it?
Questions you should be able to answer:

- What software is currently at each host?
- What software was on host H at time T?
- Why did a deploy happen to host H at time T?
- Where are artifacts of software version V deployed?
- When were artifacts of software version V no longer in use anywhere?
- What configuration did V have on host H at time T?

Some of these are for “where are known risks present”
Some are for “where and when were we vulnerable”
Some are for proactive analytics (e.g., “how many different versions are we using at once”)
Note: “artifacts of software version V”, not “software version V”. We’ll get back to that one.
Every deployment should be logged

- How was the deployment initiated?
- When did the deployment happen?
- What went into the deployment?
- What was deployed to?

This information must be append-only, durable, and kept long term.

The first one is important for cases like CI/CD credentials being leaked (Travis CI, GitHub Actions, etc.)
Append-only because even rollbacks are important. Don’t let attackers hide their tracks.
Securing the deployment logging system is itself tricky!
**Every host matters**

Production hosts

Developer environments

Beta environments

Embedded devices

Customer devices

Other environments (e.g., Lambda, CloudFlare Workers)
Do you know:

(the answer better be yes)

((but it probably isn’t))

What you are deploying where?
Where it came from?
What’s in it?
Can you trace every artifact back to sources you trust?

Not quite a “turtles all the way down” problem, but close.
Verified path from only trust anchors

If you downloaded it:

- Do you trust the entity that built it?
- How do you know that entity actually built it?
- Did that entity verify $$\downarrow \downarrow \downarrow$$ (and how do you know?)

If you built it yourself:

- How did you get the source?
- Is that source what the author intended to publish?
- Do you trust the tools you downloaded the source with?
- Do you trust the tools you verified the source with?
- Do you trust the tools you built the artifact with?
- Do you trust the host you’re building the source on?

Trust anchor: a source you assume, rather than derive, is trustworthy
As an example, Maven Central serves binary JARs, and allows publishing source, but no requirement the two match up.
Note: you can sever this search at many different points. May trust “Microsoft”, and that eliminates chunks of the graph.
Will need to choose authors you trust, mark particular source instances as trusted, or trust tools you run over the source they provide you.
Tained sources are real.

Not quite a “turtles all the way down” problem, but close.
Dependency Confusion (2021)

https://medium.com/@alex.birsan/dependency-confusion-4a5d60fec610
While attempting to hack PayPal with me during the summer of 2020, Justin Gardner (@Rhynorater) shared an interesting bit of Node.js source code found on GitHub.

The code was meant for internal PayPal use, and, in its `package.json` file, appeared to contain a mix of public and private dependencies — public packages from npm, as well as non-public package names, most likely hosted internally by PayPal. These names did not exist on the public npm registry at the time.

```json
"dependencies": {
  "express": "^4.3.0",
  "dustjs-helpers": "~1.6.3",
  "continuation-local-storage": "^3.1.0",
  "plogger": "^@.2",
  "auth-paypal": "~2.0.0",
  "wurfl-paypal": "^1.0.0",
  "analytics-paypal": "~1.0.0"
}
```
It's hard to get this right! Very briefly: widely used tool for IT monitoring (oh the irony) with auto-updates. A (signed) update from SolarWinds included a backdoored DLL. Attackers either access build hosts or got access to signing creds (updates lived on FTP server with bad pw (“solarwinds123”)). How would you detect this?

On the Feasibility of Stealthily Introducing Vulnerabilities in Open-Source Software via Hypocrite Commits

Qiushi Wu and Kangjie Lu
University of Minnesota
{wul00273, kjl}@umn.edu

Abstract—Open source software (OSS) has thrived since the forming of Open Source Initiative in 1998. A prominent example is the Linux kernel, which has been used by numerous major software vendors and empowering billions of devices. The higher availability and lower costs of OSS boost its adoption, while its openness and flexibility enable quicker innovation. More importantly, the OSS development approach is believed to produce more reliable and higher-quality software since it typically has thousands of independent programmers testing and fixing bugs of the software collaboratively.

In this paper, we instead investigate the insecurity of OSS from its openness also encourages thousands of independent programmers to contribute to the OSS itself. Such an approach not only allows higher flexibility in software evolution, but is also believed to enhance software security [21].

A prominent example of such insecurity is the University of Minnesota’s efforts to introduce vulnerabilities into the Linux kernel. This effort was motivated by the need to test newly developed software under realistic conditions. However, this approach has raised concerns about the security implications for the Linux kernel.

University of Minnesota & Linux (2021)

The commits did not ultimately land, but the attack vector is real (and scary).

Credential Leaks (constantly)

Makes it hard to trust that third-party artifacts (or code!) you download is actually from the author.

“But Jon, just sign it” — many repositories don’t even support signing!

Also watch out for outright compromised registries.

PHP git repository compromise (2021)

What did they do? Inject an RCE backdoor into PHP itself. Found a few hours later.

Rogue maintainers (2022)

https://jfrog.com/blog/malware-civil-war-malicious-npm-packages-targeting-malware-authors/ — many masquerading as colors.js! some are _for_ writing malware, but are _also_malicious
https://snyk.io/blog/peacenotwar-malicious-npm-node-ipc-package-vulnerability/ — overwrite all files with ♥ if origin is Russia or Belarus
Linux Mint ISO hack (2016)

Fighting tainted sources is difficult

SigStore to have authors sign what they publish.

The Update Framework (TUF) to check that registries behave.

Mandate 2FA for publishing to mitigate leaked credentials.

Automated continuous monitoring of known risks (like CVEs).

Ultimately, you’re at the mercy of authors...

...so choose the authors you’ll depend on wisely.

There’s more, such as if the author’s publish box is compromised!
Automated code scanning may help, if you have the source…
If you wish to make an apple pie from scratch, you must first invent the universe.

*Carl Sagan*
Do you know:

(the answer better be yes)
((but it probably isn’t))

What you are deploying where?
Where it came from?
What’s in it?

Whether you download or run `make` yourself, how do you know all the things that ended up in the artifact?
Need to know that list so that we know what we’ve deployed!
One artifact, many inputs

Regular dependencies.

Dependencies from the build host.

Downloads during the build.

Vendored or inlined sources.

Bundled binary artifacts.

Any of the above transitively...

Finding all of these is tricky even if you have the source. If you don’t doubly so. Software that tries to do this does exist, although it’s best-effort.
Heuristics will only get you so far.
Software
Bill of Materials

This is a trust exercise too — do you trust that authors included everything? But it’s better than only relying on heuristics/detection.
**BoMs have existed elsewhere for ages**

- Started in car manufacturing, since *everywhere*.
- Helps for:
  - **Design**: which part should go there?
  - **Sales**: what parts do I order?
  - **Manufacturing**: which part goes here?
  - **Repair**: which part broke?
  - **Recall**: is the affected part present?
- Similar benefits for software.
**Provenance (origin info) is useful**

- Security breadcrumbs
  - Tells you if something is at-risk (e.g., via CVE + NVD)
  - May tell you **how** it is at-risk
  - Can also tell you if it is not!
- License and compliance information
- Supply chain funding (in theory)
- Waste identification
- Quality assessment (e.g., maintenance status/EoL)
**Less important to an attacker**

A list of potential weak-points, true.

But in practice, attackers:
- already have decent heuristics and other incomplete channels;
- can probe for weaknesses directly;

The SBOM is **more** incrementally-useful to defenders.
SBOMs are hierarchical lists of contents

I produce one for my software.

It includes a list of records, each one holding:

<table>
<thead>
<tr>
<th>Component name</th>
<th>Version string</th>
<th>Hash</th>
<th>UID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supplier name</td>
<td>Author</td>
<td>Relationship</td>
<td>Relationship assertion</td>
</tr>
</tbody>
</table>

Multiple data formats exist. Two common ones are:

- Software Package Data Exchange (SPDX)
- Software Identification Tagging (SWID)

Why is author and supplier different? Quoth spec: “Until this state of transcendent SBOM utopia is achieved, SBOM authors may want to make non-authoritative claims or assertions about SBOMs for which the authors are not the suppliers.” Relationship is usually “included in”. Can be “self”.

Assertion is “what do I know of these relationships?”, such as: “unknown”, “partial” (I know there at at least these, but there may be more), “known” (I know there are only these), and “root” (I know there are none).


Multiple formats exist; SPDX and SWID are two common ones. SPDX = Software Package Data Exchange; SWID = Software Identification (tagging)
SBOMs can be combined

If you use my software, you can concatenate my SBOM.

Incomplete SBOMs are okay — there’s incremental benefit!

Don’t even need to publish your SBOMs!

SBOMs are not required to be signed, but it’s vital if you want the trust anchor, especially around author == supplier.
Imagine here for example that this was concatenated with an SBOM signed by Carol that asserts Supplier = Author = Carol for Compression Engine with a _different_ hash for same version.

SBOMs also combine horizontally

Doesn’t have to be “included in”:

- “was built by”
- “was present when built”
- “generated by”
- “patched with”
- “read data from”
- etc.

You can keep adding info and improving analysis independently.

Also “runtime/test” dependencies
Do you know:

(The answer better be yes)

What you are deploying where?
Where it came from?
What's in it?

___