le cnam

Software Supply Chain Security for the Cloud

Dependency threats

Jacopo Bufalino (CNAM)

Introduction

Introduction

Where we left

- ➜ New code has been developed
- ➜ Merged to a main branch
- ➜ ... CI pipelines ...
- ➜ Produced an artifact that is ready to be distributed

How about the dependencies?

Step back: what goes inside an artifact?

If we build a binary

- ➜ Source code
- ➜ Package dependencies

If we build a container

- ➜ Source code
- ➜ Package dependencies
- ➜ Operating system dependencies

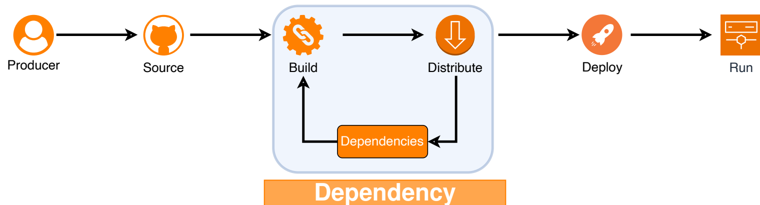
If we build a cloud app

- ➜ Source code
- ➜ Package dependencies
- ➜ Operating system dependencies
- ➜ Other containers

Dependency threats

Dependency threats refer to risks of third-party software used or imported in a project.

Dependencies are a vital part of the software lifecycle as they allow to reuse components and speedup the development process



Dependency threats

Package selection: Typosquatting

Typosquatting is the practice of registering websites, packages and domain names that resemble well-known brands or companies with malicious intent.

Examples

- ➜ Malaware delivery
- ➜ Credentials harvesting
- ➜ Brand impersonation (fraud)
- ➜ Data exfiltration

Typosquatting: an example

amazon.com

Examples of typosquatting

- amaz0n.com
- amzon.com
- amazon.com (Cyrillic ”a” (U+0430))
- amazon.com (Greek ”o” (U+03BF))

Browserify attack (2021)

- Malicious group of attacker created a clone package `web-browserify`
- Malaware designed to target specific individual developers
- Security researchers found out about the package early so not many computers were affected

Mitigations

- ➜ Double check the packages that are installed
- ➜ Copy package names from well-known resources
- ➜ Try not to write down package names by yourself

Package availability: `mimemagic` (2021)

- mimemagic is an OSS component used to understant the MIME type of a file
- It bundled data derived from a database licensed under GPL
- The tool itself was bundled under MIT (less restrictive)
- Package mainainer deleted all published versions for compliance
- Millions of dependant packages were not compiling anymore

Solution: remove GPL licensed software and rebuild (some of) the versions

Package availability: `terraform` (2023)

- Terraform is a tool for Infrastructure as Code (IaC) provisioning
- Used by several of organizations
- Authors noticed third-party companies were using terraform for commercial purposes
- Changed the license to BSD
- Unclear to which extent the tool was usable
- Several customers affected

Solution: OSS community created a fork of the original tool, now called `open-tofu`

Vulnerable third-party software

Software development relies on dependencies. When one of them is vulnerable, it can create a cascading effect on all the packages that use it.

The event-stream case

`event-stream` was a very popular javascript application

- ➜ New developers offers to maintain the project
- ➜ Add a malicious piece of code in the library
- ➜ All users dowloading the version had the malicious code
- ➜ The attack was targeted to a specific organization

Mitigations

- Use package manager
- Audit dependencies for security

Package managers

Package manager

A package manager is a tool that automates:

- ➜ installing
- ➜ updating
- ➜ configuring
- ➜ removing

software packages on a system.

Package manager: fetching packages

Package retrieval is done using:

- official repositories (e.g, `pypi`,`npmjs`)
- third-party indexes (mirrors)
- Directly from git
- From local paths

Package mangers use files to store dependency information.

Two types

- Project manifest (requirements.txt)
- Lockfiles (requirements.lock)

Package manager security

Not all package managers offer the same level of security. Ideally we would like to have the following:

- Reproducible installs
  - guarantee the same package versions across installs
- Integrity verification
  - Verify package contents match a known checksum
- Authenticity of source
  - Use HTTPs to download packages
- Strict installation
  - Only download source code, no extra build while on the host
- Provenance
  - Information on how package was generated
- Build vs dev dependencies

Package manager security: 2

Packages are build using `releases`. They can be stored in VCS or external buckets. A common issue with the deployment process is that git tags are not immutable so they can be overwritten.

requirements.txt

`requirements.txt` is a project manifest for the python programming language. It stores the dependency information in different formats

```
requests
numpy>=1.21,<2.0
urllib3~=1.26.0
Django==3.2.18
Jinja2==2.11.2 \
    --hash=sha256:89aab2...
```

requirements.txt

```
requests
numpy>=1.21,<2.0
urllib3~=1.26.0
Django==3.2.18
Jinja2==2.11.2 \
    --hash=sha256:89aab2...
```

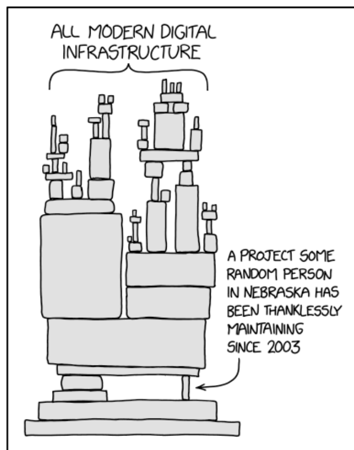
Which guarantees does each of the methods above give us?

Dependencies

Package dependencies create intricate graphs in which each dependency depends on several other packages.
If a security incident happens in one of them, all dependant packages will be affected.

Software ages like milk, not wine

Open source dependencies



Source: xkcd

Almost all software dependencies are open-source and maintained by a few (mostly unpaid) developers.

Dependency audit

It is important to audit dependencies at every stage of the supply chain

- ➜ Specific tools analyze the content of the package manager files
- ➜ Scan the folders where the packages are installed
- ➜ Compare them against vulnerability databases
- ➜ Report vulnerabilities, severity and updates (if available)

The importance of dependency pinning

Without specifying a package used, vulnerability scanning becomes less effective

```
requests
numpy>=1.21,<2.0
urllib3~=1.26.0
```

- ➜ Packages installed locally may not be vulnerable
- ➜ The ones installed in CI the tools install a vulnerable version

The importance of dependency pinning

By using dependency pinning we can also aim for reproducible builds

➜ Building the same artifact produces identical results.

Software composition analysis (SCA)

Not only programming language dependencies

Dependency audit gives us information about dependencies in a software product.

How about other properties / dependencies?

- ➜ Licenses
- ➜ OS
- ➜ OS packages
- ➜ Installed binaries
- ➜ Third party applications / containers

Software composition analysis (SCA)

Software composition analysis (SCA) is an automated process that identifies components in repositories and software artifacts.

How it works

- Scan filesystem, container, repository
- Compare the content agsint a list of ‘well-known’ files and paths
- Eventually open such files and read the content (e.g., read LICENSE file from project to get the license information)

SCA capabilities

SCA answers the following questions:

- ➜ What are the dependencies of my software product?
- ➜ Which files are used/generated by my dependencies?
- ➜ Which licenses does my software use?
  - ► Am I allowed to use this dependency for commercial purposes?
  - ► Do my dependency breach agreements with my clients?
- ➜ Where do my dependencies come from?

Where to deploy SCA software?

- Build:
  - CI pipelines enforce dependency policies
- Deployment:
  - Container/image scanning

Software Bill of Materials (SBOM)

From SCA to SBOM

The output of SCA is a list of components, libraries and licenses of a software product which is called Software Bill of Materials (SBOM).

- ➜ A formal, machine-readable inventory of software components
- ➜ Lists:
  - Libraries and dependencies
  - Versions
  - Suppliers
  - Licensing information

SBOM Core Data Elements

- Component name
- Version
- Unique identifiers
- Dependency relationships
- Supplier / author
- Files
- License
- Hashes (integrity verification)

Package identifiers

Package URL (pURL)

- ➜ `pkg:type/namespace/name@version?qualifiers#subpath`
- ➜ `pkg:maven/org.apache.commons/commons-lang3@3.12.0`

Common Platform Enumeration (CPE)

- ➜ `cpe:2.3:part:vendor:product:version:...`
- ➜ `cpe:2.3:a:apache:log4j:2.14.1:*:*:*:*:*:*:*`

SBOM Main Standards

- SPDX (Software Package Data Exchange)
  - Maintained by Linux Foundation
  - Strong in license metadata
- CycloneDX
  - OWASP-driven
  - Security-focused (vulnerabilities, services)

SBOM Generation

Typically uses the SCA approach

- ➜ Generated during:
  - ▶ Build time
  - ▶ Post-build
- ➜ Integrated into CI/CD pipelines

Example SBOM

```
syft python:3.10 --output json > sbom.json
```


```json
{
  "artifacts": [
    {
      "id": "adbd50b08f319423",
      "name": "Simple Launcher",
      "version": "1.1.0.14",
      "type": "dotnet",
      "foundBy": "dotnet-portable-executable-cataloger",
      "locations": [
        {
          "path": "/usr/local/lib/python3.10/site-packages/pip/_vendor/distlib/t32.exe",
          "layerID": "sha256:d776b3e6e5bfd1a4f40e00a887ef5ef1ac59006a4cf293354ca49036c959267d",
          "accessPath": "/usr/local/lib/python3.10/site-packages/pip/_vendor/distlib/t32.exe",
          "annotations": {
            "evidence": "primary"
          }
        }
      ],
      "licenses": [],
      "language": "dotnet",
      "cpes": [
        "cpe:2.3:a:Simple_Launcher:Simple_Launcher:1.1.0.14:*:*:*:*:*:*:*"
      ],
      "purl": "pkg:nuget/Simple%20Launcher@1.1.0.14",
      "metadataType": "dotnet-portable-executable-entry",
      "metadata": {
        "assemblyVersion": "",
        "legalCopyright": "Copyright (C) Simple Launcher User",
        "internalName": "t32.exe",
        "companyName": "Simple Launcher User",
        "productName": "Simple Launcher",
        "productVersion": "1.1.0.14"
      }
    },
```


Path of the file used for finding the software

Layer ID where the file is

Package name under the **CPE** notation

Package name under the package-URL (**PURL**) notation

Metadata information on the package

Figure: Example of SBOM generated by SCA software

SBOM and Vulnerability Management

- ➜ SBOMs generate per-package unique identifier
- ➜ SBOMs of the same artifacts do not change
- ➜ Identifiers can be used to check for vulnerabilities

SBOM and Vulnerability Management

- Map SBOM components to vulnerability databases
- Continuous monitoring for new CVEs
- Enables:
  - Impact analysis
  - Prioritized patching

Key idea

SBOM is the same but vulnerabilities change.
We take the SBOM and every x days we run a CI job to check for vulnerabilities.

Vulnerability lookup

There is usually no direct mapping between package identifiers and vulnerability databases.

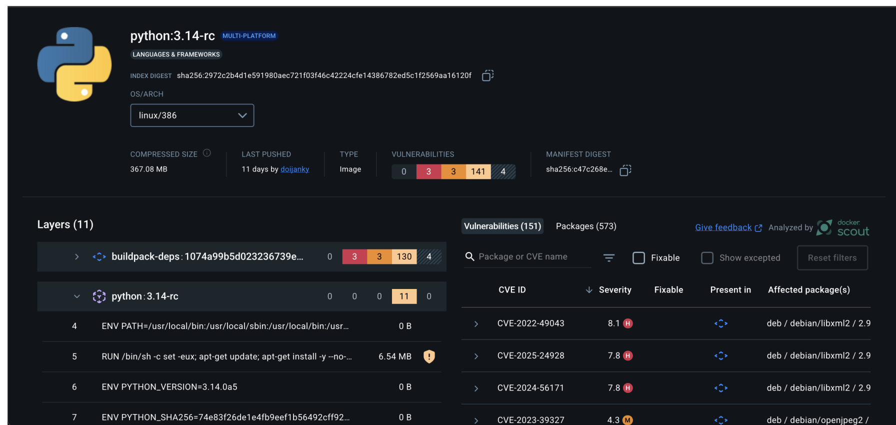
Security tools:

- ➜ Take package identifiers and create a query to vulnerability databases
- ➜ Combine queries into a single DB
- ➜ The DB is used for faster lookup

Many SCA tools are also able to search for CVEs

SBOM integrations

SBOM generation and vulnerability lookup is currently a featuree of many package and container registries.



Vulnerability report from DockerHub

Regulations and legal frameworks

The need for regulation

- ➜ Increasing frequency of supply chain attacks (e.g., SolarWinds, Log4Shell)
- ➜ Complex dependency graphs (open source + third-party components)
- ➜ Lack of transparency in software composition
- ➜ Need to protect consumers

Regulatory Landscape

- United States:
  - Executive Order 14028 (2021)
  - NIST Secure Software Development Framework (SSDF)
- European Union:
  - Cyber Resilience Act (CRA)

Common objective

- Improve software integrity, transparency, and accountability

Executive Order 14028

- Issued May 2021 to strengthen U.S. cybersecurity posture
- Focus on federal systems and procurement
- Companies working with fedeeral government need to show how they handle the software supply chain

EO 14028: Core Supply Chain Requirements

- ➜ Secure Software Development Practices
- ➜ Mandatory transparency of components
- ➜ Suppliers must attest to secure development practices
- ➜ Vulnerability Disclosure Programs

EU Cyber Resilience Act (CRA) Overview

- EU-wide regulation covering software and hardware products
- Applies to manufacturers, importers, and distributors
- Introduces mandatory cybersecurity requirements across lifecycle
- Establishes legal accountability for insecure products

CRA: Core Supply Chain Requirements

- ➜ Secure-by-Design and Secure-by-Default
- ➜ Mandatory Vulnerability Management
- ➜ Mandatory transparency of components
- ➜ Mandatory Incident Reporting

Core of the regulations

- ➜ Secure development lifecycle
- ➜ Vulnerability Disclosure
- ➜ Software Bill of Materials (SBOM)

Secret scanning

Secret Scanning

Process of detecting sensitive credentials embedded in code or artifacts. Targets

- API keys, tokens, passwords
- Private keys (SSH, TLS)
- Cloud credentials (AWS, Azure, GCP)

Applied across:

- Source code repositories
- CI/CD pipelines
- Container images and logs

The issue of secrets

- Secrets are frequently hardcoded or accidentally committed
- Public repository leaks are actively exploited
- Common root cause in major breaches

Secrets exposure often bypasses traditional vulnerability controls

Common SCA or SAST tools do not look for secrets

Which types of secrets do we look for?

- Static credentials:
  - Hardcoded passwords
  - Database connection strings
- Token-based secrets:
  - API keys (e.g., GitHub, AWS)
  - OAuth tokens
- Cryptographic material:
  - Private keys
  - Certificates

Detection Techniques

- ➜ Pattern matching (regex signatures)
- ➜ Entropy analysis (randomness detection)
- ➜ Dictionary based (look at secret names)
- ➜ Hybrid (combination of one or more techniques)

Trade-off: Precision vs recall (false positives vs missed leaks)

Secret Scanning in the SSC

- Pre-commit:
  - Local hooks prevent leaks before commit
- CI/CD:
  - Automated scanning during builds
- Repository:
  - Continuous scanning of code history
- Runtime:
  - Detection in logs, configs, and memory dumps

Secret Rotation

Process of periodically replacing cryptographic secrets and credentials.

Secret rotation is important to mitigate the effect of credential theft

Rotation Architecture

- Centralized secret management system (e.g., Vault, cloud KMS)
- Identity-based authentication (OIDC, workload identity)

Requirement: eliminate manual credential handling

Summary and Takeaways

- ➜ Dependency management is a key component of the SSC
- ➜ Software Bill of Materials (SBOM) is the key document
- ➜ Software Composition Analysis (SCA) is a way to generate SBOM
- ➜ SBOM is becoming mandatory with the EU CRA
- ➜ SBOM can be used to look for vulnerable components
- ➜ Secret scanning is also important and often overlooked
  - ▶ Many vulnerabilities start with leaked secrets!