Alpha-Omega

2024 Grantee Outcomes by Investment

AIRFLOW

Airflow Beach Cleaning project was a joint work between Alpha-Omega Fund, Apache Airflow PMC members — with involvement of the Apache Software Foundation and Python Software Foundation.

The idea behind the project was to explore innovative ways how we can address Open Source Software Supply Chain Security challenges, especially in the wake of approaching regulations — such as Cyber Resilience Act in Europe. Software Supply Chain is one of the most difficult areas of security, and while there are a number of attempts to address it at scale — via a number of tools, services, processes and industry standards such as SBOM, VEX and similar, the challenge of addressing supply chain in highly distributed, not easily regulated reality of vast number of open-source projects remains a difficult problem to solve.

While most of the approaches focus on providing tools and analysing the components of modern software by looking at the project properties and providing some metrics, reports, there are not many ideas what to do with those metrics and how to make them useful at scale. This is particularly challenging in the environment where many open-source projects are completely independent and not part of any structured organisations, where many of those projects are run by small teams or individuals, and where most of the relations in the industry are based on community values, willingful cooperation and trust — where the "spirit" of the cooperation and relationships is more important than business outcome of those.

Airflow Beach cleaning attempts to explore a way how the various metrics, datapoints and signals can be turned into actions, while not losing the community spirit and building on the strength of the open-source community — human relationships and collaboration.

The idea of the project is to incentivize maintainers of the open-source projects to look at their dependencies not only from the perspective of "how can I use them" but also "how can I help my dependencies — and by transition the whole OSS supply chain — to improve their security and maintain it". The idea is that every smaller and bigger project has a number of dependencies, sometimes small, sometimes big. Some of those projects — mostly those that have many dependencies are important-enough for commercial users that they can fund security initiatives run by maintainers of those bigger projects — to contribute back and to help their dependencies with their security challenges.

Apache Airflow "Beach" (or how we named it) — consists of 700+ dependencies — direct and transitive ones. Airflow is one of the projects that can not only attract funding from their stake-holders and security community, but also has a thriving community of contributors, maintainers, PMC members who might spend some of their time on reaching out to their dependencies and

helping them to improve their approach to security — implement best practices such as Trusted Publishing, implement Security Policies and establish security response processes, introduce protections and preventions against tampering with their sources — such as commit signing, branch protection, introduce practices such as reproducible builds, make their CI process resilient to attacks and dangers of tampering with build systems.

While exploring what is available there we came to a number of conclusions and learning that allowed us to shape the project as we were proceeding with it. This is a highly experimental project and when we started it, we had a very vague idea of what such a personal and relationship approach could look like and how it could be turned into concrete actions. However during the process we've learned a lot from the initial interactions and discussions, also coming from cooperation with partners such as "Open Refactory" that run analysis on solved/unsolved/potential security issues on all Airflow dependencies.

The project has three main "pillars":

- Performing the analysis, communicating with the maintainers and helping them to improve security
- Sharing the lessons learned and ideas coming from the initial analysis via participation in industry events, giving talks, organising birds of a feather sessions and eventually possibly workshops where we will help maintainers to understand what they can do
- Building tooling and blueprints so that the learnings and approach can be applied to other ecosystems and to the wider Python ecosystem — including tooling for the Python Software Foundation that can be provided to more maintainers to run their own "Beach Cleaning".

Pillar 1: Analysis and interaction

- Analysis of current, potential and past CVEs handled — while useful and helpful — can be done by multiple tools, but what is crucial is how to make open-source maintainers aware of them and responding to them in the way that can be sustainable. Single audit and report is great, but the important is how we turn it into regular effort and engagement of the maintainers at a scale.
- Human factor counts, automated reports and tools reporting issues are often ignored and not considered as worth spending maintainers time on. Adding the human factor in, knowing that whoever is on the other side also spends their time and energy on relationships and reports, matters a lot to a successful engagement of the maintainers.
- Reaction and engagement of the open-source projects is much better when they are interacting with maintainers of other open-source projects — especially those which are more popular and which are directly depending on their own projects.

- Many of the maintainers do not need money directly. They are more interested in having attention, feedback and contributions from others, rather than being paid for what they are doing directly. In some cases they are worried about the future of the Open-Source approach and we need to educate, collaborate and cooperate with them to help with their worries
- We can likely scale the approach by going to direct dependencies first and finding other maintainers who have their dependencies to move it lower the dependency tree — they might need similar (smaller) funding which can be trickled down the chain and it's likely that by following it with a number of projects, we can get quite a good coverage of education, focus, process improvements across big "beaches" in the industry — often overlapping and resulting in significant proportion of the ecosystem covered.
- As a result we can build a network of distributed — but connected — open-source maintainers who can be advocates and promoters of the collaborative, community driven way of improving security across significant part of the ecosystem

The following things have been delivered by the Airflow team and cooperating partners during the August — December stage of the project:

- Thanks to Open Refactory efforts, we identified, reported and followed up 16 potential security issues that have since been responded to by the maintainers — often after follow-ups from the Airflow maintainers. This has validated the assumption that human factor and relationships with other maintainers makes a difference
- Airflow team created automated analysis and reports that help to analyse Airflow dependencies with various signals and data (using Airflow SBOMs, OpenPSF scorecards, Github statistics and a number of other signals) — producing a "Meta-Health Audit" that allowed to prepare next steps
- The reports have been used to identify a subset of projects (16) that should be the first batch of projects to address improvements in their security processes. Those projects have been contacted and active cooperation is ongoing in improving certain areas:
    - Trusted Publishing
    - Protecting branches
    - Signing commits
    - Having Security Policies and response process
    - Protecting build / CI / release process against tampering (including reproducible builds)
- Currently 7 out of the 16 contacted project maintainers responded and the majority of the maintainers are happy to accept improvements or implement them.
- There are so far several types of reactions observed:

  - Rejecting the need and scope of improvements
  - Happy to cooperate and accept improvements
  - Expressing interests in working together and scaling the efforts
  - Expressing concerns about future of the OSS with the regulations coming
- Work is ongoing to turn the relationships/connections into concrete improvements and changes
- In some cases the interaction will lead to assessing that dependencies should be either "forked" or "foregone" — i.e. replaced with other dependencies where we find interaction leading to conclusion that the dependency is risky to stay
- Also work is ongoing to identify potential ways how money can be used to improve security of some of the projects — including discussions with security researchers, running audits or running individual projects of switching dependencies.

Pillar 2: Raising awareness and sharing lessons learned

The initial findings from the early stage of the project have been shared with the community and feedback has been gathered. The following events hosted us speaking about the approach:

- Airflow Summit — 10-12 September, San Francisco — where the project was presented in the keynote and "Birds of a Feather" happened with few interested members of the community: Recording available
- Community Over Code (Flagship Apache Software Foundation conference) — 7-19 October, Denver, there was a security-themed talk at the Data Engineering track — and Birds of Feather session — with ~ 30 interested community members (including ASF infrastructure and ASF Board members) took active part in
- Further talks are planned at FOSDEM in Brussels 1-2 Feb 2025 (approved already at SBOM track). FOSS backstage in Berlin, March 10/11th (approved), Pycon US (pending submission)
- Possible further talks (pending submission): VulnCon (Raleigh IS) April 7-10, 2025, APPSEC Barcelona — May 26-28,, Berlin Buzzwords — Berlin, June 15-17, 2025

We have two draft blog post prepared — pending publishing:

- Security lessons from interacting with our direct Supply Chain dependencies.
- Supply Chain Security as a human problem (at scale).

Pillar 3: Building reusable tooling and blueprints for others

The 3rd pillar is building on top of the finding and analysis done in Pillar 1 and 2, once they reach enough scale and we will be able to see emerging patterns and extract useful commonalities. Currently the work is on-going in several areas:

- The tooling developed for Airflow is currently bound with Airflow code, but it can be extracted out, and provided to other maintainers — its based on publicly available data and tools (such as GitHub and Open PSD scorecards), uses shared Google Spreadsheets to keep track of the projects and produce automated reports and it can be adapted to be used by anyone
- The learnings and initial idea about 'Beach Cleaning" is now also being done in "Jenkins" project in the Java/Groovy ecosystem
- Tooling is being developed (**Airflow Publish Action** — that aims to produce a reusable set of actions that can be used by Apache Software Foundation projects to plug-in Trusted Publishing in the regular release process of Apache Software Foundation — many of Airflow dependencies are Python based Apache Software Foundation projects, so sharing the tooling and deploying it for other projects in the Foundation is part of the "Beach Cleaning" exercise.
- We plan to turn the whole idea into a blueprint that others will be able to follow, promote it and turn it into a "standard" to follow in all open-source ecosystems.

RUST

Since the first stable release of the Rust programming language in 2015, the Rust ecosystem has grown tremendously, thanks to the tireless work of its maintainers, contributors, and advocates. Every day, Rust's many advantages are becoming more evident to developers and organizations alike.

But like any programming language, the meteoric rise of Rust also introduces complexities and risks. When the user base of any programming language grows, it becomes more attractive to malicious actors. As any programming language ecosystem expands with more libraries, packages, and frameworks, the surface area for attacks increases. Rust is no different.

As a steward of the Rust programming language, the Rust Foundation has a responsibility to provide a range of resources to the growing Rust community. This responsibility means we must empower contributors to participate in the Rust Project in a secure and scalable manner, eliminate security burdens for Rust maintainers, and educate the public about security within the Rust ecosystem.

In September 2022, the Rust Foundation announced its commitment to fulfilling these responsibilities through the Security Initiative, which we were able to create thanks to a generous initial investment of $460k from Alpha-Omega.

In 2023, Alpha-Omega's support of the Security Initiative enabled the Foundation to hire a team of full-time security engineering professionals, establish a plan of focus for the initiative, begin the development of Rust ecosystem threat models, create tools for discovering and analyzing vulnerabilities and start working through security-related technical debt.

The Foundation, through its security initiative, wanted to build upon the successes of the previous year in 2024. We focused on supply chain security, enhanced vulnerability detection, and continued development of tools to protect the open-source community from potential threats. The success of our focus manifested itself in a number of accomplishments:

- The completion of all four Rust security threat models and taking action to address those threats. This includes threat models for the crates ecosystem, Rust infrastructure, crates.io and the Rust Project.
- Publishing the TUF RFC, which proposes the adoption and implementation of The Update Framework (TUF) for providing the chain of trust and implementing signatures for crates and releases.
- Implementing real-time crate security scanning, utilizing Foundation-developed tools such as Typomania, a library for detecting potential typosquatting in software registries, and Sandpit, our tool that scans available crates to try to detect malicious crates based on available heuristics.
- Further developing Painter, an open source tool for building dependency graph databases, allowing users to directly know the path of potentially vulnerable code. Those running Painter will be able to obtain `unsafe` statistics, better call graph pruning, FFI boundary mapping and support for the latest version of Rust.
- Conducting comprehensive provenance tracking for all crates published on crates.io, including commit inference for crates that do not include VCS metadata. While a significant number of mismatches have been found, analysis has indicated that no popular crates differ from their upstreams in malicious ways, and no red flags have been seen across the crate ecosystem. This work is ongoing, and continues to check newly published crates.
- Making a plethora of security-minded changes to crates.io, including crate deletion, spam detection and crate owner notification tools.
- Publishing the C++/Rust Interop problem statement and strategy. The Foundation joined INCITS to participate directly in the ISO C++ standardisation process and collaborated on a Rust proposed 2025H1 Project Goal for seamless C++/Rust interoperability.

- The Safety-Critical Rust Consortium was formed to bring those in the industry together to work to ensure Rust is not only suitable for safety critical applications, but excels at it. There are two primary subcommittees, thus far: coding guidelines and tooling.
- The Rust Infrastructure Threat Model has identified the lack of out-of-band backups for critical data assets of the Rust project as a major threat. We are working on mirroring all Rust releases and crates onto a separate infrastructure. Also, as part of its ongoing effort to catalog and patch servers for security and efficiency, the Rust Project infrastructure team updated all of the servers it manages to Ubuntu 24.

You can read more about the Security Initiative's progress in our various Foundation reports:

- Rust Foundation 2024 Annual Report
- Rust Foundation September 2024 Technology Report
- Rust Foundation February 2024 Security Initiative Report

All of us at the Rust Foundation are grateful to Alpha-Omega for their generous support of our Security Initiative over the past year.

RUBY CENTRAL

With Alpha-Omega's support, Ruby Central was able to take on two significant projects in 2024, an external security audit of RubyGems.org and adding Organization Accounts to RubyGems.org.

External Security Audit

Ruby Central partnered with Trail of Bits for a comprehensive security audit on the RubyGems.org Rails application and its underlying AWS infrastructure. The audit identified 33 issues, including seven medium-severity items and one high-severity item. Notably, most of these findings do not constitute actual security breaches. Our team started addressing each finding as they were reported and used these insights to bolster RubyGems's security posture.

One of the important findings from the security audit was the lack of multi-party approval for production deployments. The team identified this as an important initiative for 2025 based on the work of Trail of Bits. This initiative will work towards limiting manual, ad-hoc manipulations, introduce better separation for projects, add regular inventorization of credentials, and rotate old credentials.

Overall, the audit attests to the effort that the core team has put into ensuring RubyGems.org is secure and reinforces that we are working in the right direction with our efforts to implement more of our infrastructure as code and to codify and constrain our access policies.

Organization Accounts

The upcoming Organization Accounts feature is a highly anticipated release that will give companies and development teams more control over their gems through structured permissions management. With Organization Accounts, teams can assign and adjust roles, ensuring that only authorized users can manage specific gems. As team members change, permissions can be easily updated, helping to prevent disruptions and maintain continuity in gem management. This is especially valuable for large organizations like AWS, which manage hundreds of gems with many contributors.

The release will roll out in two stages: first, with dedicated admin access controls within RubyGems, allowing organizations to manage permissions and add or remove members. The second stage will enable them to officially link their gems to their organization, providing added security and transparency across the ecosystem. We shared a preview of this work at RubyConf 2024 in Chicago.

This feature marks a significant step forward in aligning RubyGems with enterprise needs. It is also a stepping stone toward future work, such as scoped gems, SAML/OpenID authentication, and other enterprise features that might offer future revenue.

JENKINS

In October 2024, the Alpha Omega Foundation provided a 3-month grant to the Jenkins project to improve the implementation of Content Security Policy (CSP) across the Jenkins ecosystem. The goal is to enhance the security of Jenkins by shielding it from injection attacks like cross-site scripting (XSS).

The project has been led by technical lead **Basil Crow**, and developers **Yaroslav Afenkin** and **Shlomo Dahan**. Over the three months of the project, the team has released versions of Jenkins core and more than 40 plugins with Content Security Policy improvements.

Key Milestones and Achievements

- Dramatically improved Acceptance Test Harness (ATH) results, reducing failures in restrictive CSP mode from numerous issues down to only 5 remaining
- Updated and released over 40 widely used Jenkins plugins with improved CSP compatibility, including critical workflow plugins, and popular visualization tools
- Focused efforts on modernizing jQuery usage across the plugin ecosystem, upgrading to jQuery 3.x, and removing inline JavaScript
- Created more than 800 JIRA issues to track and address CSP-related issues across the Jenkins project

The Jenkins community welcomes participation in this important security initiative. Users can help by testing plugins with CSP restrictive mode enabled, reporting issues, and contributing to plugin modernization efforts. For more information, visit the Jenkins CSP documentation page.

Jenkins Security Improvements in 2024

OVERVIEW

In 2024, the Jenkins Security Team continued its robust commitment to enhancing the security of the Jenkins ecosystem, focusing on proactive vulnerability management, advanced scanning techniques, and comprehensive community protection.

KEY SECURITY ACHIEVEMENTS

Vulnerability Management

- Issued 17 security advisories covering both Jenkins core and plugins
- Announced and addressed 211 vulnerabilities
- Reviewed 70 pull requests in Jenkins core for security considerations

Advanced Security Scanning

CODEQL INTEGRATION

- Implemented advanced CodeQL rules for automatic security audits
- Added support for warning suppression
- Collaborated with GitHub Security Lab, which reported 27 vulnerabilities using custom CodeQL rules

Innovative Security Approaches

- Experimented with AI-assisted ticket triage to improve vulnerability detection ●
- Introduced exclusive Continuous Deployment (CD) mode for plugins to reduce supply chain risks
- Significant progress on implementing Content Security Policy (CSP) headers

Plugin Security

- Continued to enhance plugin security through rigorous review processes ●
- Maintained an active vulnerability disclosure and resolution program ●
- Automated security checks for new plugin hosting requests

Emerging Security Initiatives

- Custom web framework security scanning (https://www.jenkins.io/redirect/jenkins-security-scan/)
- Automatic checks on plugin hosting requests
- From September to November 2024:
  - 14 merged plugin hosting requests automatically checked
  - 4 open plugin hosting requests
  - 2 requests were manually reviewed with maintainer guidance

LOOKING FORWARD

The Jenkins Security Team remains committed to:

- Continuous improvement of security scanning technologies
- Proactive vulnerability management
- Community education and engagement in security best practices

By prioritizing security at every level, Jenkins continues to provide a trusted, robust automation platform for developers worldwide.

OPENJS

Node.js

The Node.js open source project has made significant strides in enhancing its security measures over the past several months. Here's a summary of the key security updates:

Automation and Process Improvements

The Node.js team has implemented several automation enhancements to streamline security releases and improve overall processes:

- A new "git node security" command automates steps of security releases, replacing four manual tasks with a single command.
- The security release process has been automated, leading to more frequent and efficient releases. Previously, a security release required one security release steward and one releaser for each active release line. For example, 4 people were needed to perform a security release for 3 active release lines. Now, this process is much easier and is being handled by a single person funded by Alpha Omega.
- Proposal to automate releases using `git node release`, with added support for security-related proposals via the `--security` flag.

Security Releases and Vulnerability Management

- The team resolved 96 HackerOne reports during 2024.
- 3 Security Releases happen through 2024 solving 8 CVEs across different release lines. ●
- Stable Threat Model — Node.js team is receiving fewer "Non-Applicable" reports after creating its threat model document.
- Regular releases including SEMVER-MAJOR have been supported by the security expert.

Permission Model and Policy Integrity

- Support for Buffer in process permission has been introduced in the Permission Model. ● Microsoft employees are working on a policy integrity feature for Node.js, which is under discussion by the security team.
- Support to callback error when using Asynchronous APIs with permission model enabled.

- Relative paths are now supported by the permission model
- WASI has been included in the resources blocked by the permission model. A new flag --allow-wasi has been introduced.

Experimental Features and Testing

- The team is re-evaluating Node.js experimental features in collaboration with the Node.js Next 10 Working Group.
- Improvements have been made to the CITGM (Canary in the Gold Mine) testing system, including the addition of warnings for module failures across all platforms.

Is-my-node-vulnerable-

- This package was created to help Node.js users to ensure they are using a safe version of Node.js. This package helps ensure the security of your Node.js installation by checking for known vulnerabilities. It compares the version of Node.js you have installed (process.version) to the Node.js Security Database and alerts you if a vulnerability is found.
- The team received a very good response from the community
    - https://www.linkedin.com/posts/node-js_to-ensure-your-nodejs-version-is-secure-acti vity-7253071981380583424-Rzib

Resources

COMMUNITY ENGAGEMENT AND CONTRIBUTIONS

- Team members spoke at events like CityJS Medellín showcasing "The Journey of the Node.js Permission Model."
- A security expert gave a workshop at the Grace Hopper Celebration Day event on "How to Contribute to Node.js Security."
- A Slack channel (#nodejs-mentoring) has been set up for first-time contributors. Live sessions happened on Twitch.tv as a tutorial on how to contribute to a big open source project such as Nodejs

ISRG Prossimo

The goal of Internet Security Research Group (ISRG)'s Prossimo project is to bring memory safety to the most critical software infrastructure on the Internet. This shift to memory safety will eliminate vulnerabilities that lead to security breaches and data leaks that can cause personal and financial harm to Internet users, result in the mass denial of essential public services, and threaten basic human rights and safety. **Thanks to Alpha-Omega's support, ISRG improved Internet security by making progress toward three Prossimo initiatives: Rustls, Rust for Linux, and rav1d.**

Rustls: Rustls is an open source and memory safe TLS library. Our work centered on improving Rustls performance and functionality to make it competitive with the leading TLS library, OpenSSL. Open SSL is not memory safe, making it vulnerable to an entire class of prevalent bugs. Because billions of phones, computers, servers, IoT devices, and embedded systems rely on TLS to securely communicate over networks, providing a performant memory safe alternative to OpenSSL is critical to improving Internet security.

We accomplished the following toward Rustls:

- merged no-alloc API support in Rustls, which is important for high performance consumers;
- completed, merged, and shipped a benchmarking system to allow us to gauge our progress;
- made AWS Libcrypto for Rust (aws-lc-rs) cryptography library the default in Rustls, with the option to enable FIPS support; FIPS is an important requirement for many organizations and so this removed a barrier to using Rustls;
- landed support for Post-Quantum Key Exchange, protecting Rustls users from adversaries looking to intercept encryption keys used in a TLS connection;
- gained Nginx and OpenSSL Compatibility, allowing Nginx users to switch from OpenSSL to Rustls with minimal effort
- showed that Rustls outperformed both OpenSSL and BoringSSL in every handshake and throughput scenario tested, evidence that our work moved us closer to Rustls being a viable, performant, and memory safe alternative to OpenSSL.

Rust for Linux: Rust for Linux is the project adding support for the Rust language to the Linux kernel, in order to improve the security and reliability of billions of devices and computer systems that use the Linux kernel around the world. Support for the memory safe Rust programming language was

merged into the open source Linux kernel in late 2022, which was a key milestone towards that goal. In 2024, we worked with the primary maintainer of Rust for Linux, Miguel Ojeda, to continue to improve the support for Rust in the Linux kernel so that the first major Rust production users c be merged upstream.

We accomplished the following toward Rust for Linux:

- grew community: new major key users were announced (such as the Nova driver for GSP-based Nvidia GPUs), new companies got involved in Rust for Linux development (such as Red Hat), doubled the number of unique patch submitters to 110+, onboarded a new core team member, got 200 more Zulip users and 100 more mailing list subscribers;
- submitted Rust subsystem pull requests into the Linux kernel (6.8, 6.9, 6.10, 6.11, 6.12 and 6.13), containing the work from a multitude of developers and companies, to support the first Rust users that have already landed in Linux (Asix PHY driver, AMCC QT2025 PHY drive Null Block driver, DRM panic screen QR code generator);
- performed development work such as the patch series on supporting multiple Rust versions and the reconfig/rebuild support, supporting mitigations for CPU vulnerability improvements for Rust lints, among other patches;
- maintained the Rust support in the stable & LTS (6.1.y, 6.6.y, 6.12.y) kernels, including back-porting fixes as neede
- started to support building Linux with a range of Rust releases, which in turn allowed us to start supporting Linux distribution toolchains, which was a top requirement from key upstream Linux maintainers;
- started hosting the generated Rust code documentation in the Linux kernel website;
- collaborated with other projects, such as our regular meetings with upstream Rust to make progress on getting Rust for Linux into stable Rust (one of the 2024H2 Rust flagship goals with milestones such as the Linux kernel being tested in the Rust's CI to ensure it can be built with upcoming Rust compilers until the kernel no longer needs to rely on unstable features;
- co-organized Kangrejos (the Rust for Linux conference), keynoted at RustConf, organized the Rust Microconference at LPC (the main kernel conference), presented in the LPC Kernel Summit track, and spoke at the invitation-only Kernel Maintainers Summit about committing to Rust in the kernel.

Rav1d: Complex data parsing is one of the most security-critical operations in modern software. It's a particularly big issue for video decoders, and the fact that they are usually written in C doesn't help. AV1 is set to become one of the most important media formats on the Internet.

Because of this, we undertook work toward building a highly performant memory safe AV1 decoder called rav1d, which can be used for both video and images.

We accomplished the following toward rav1d:

- moved dav1d's C code to Rust for memory safety;
- created drop-in C API compatibility;
- optimized to achieve performance that is nearly on par with the C implementation;
- reused the assembly code from dav1d and made it easy to frequently synchronize it;
- added X86-64 and ARM64 support.

Free BSD

The FreeBSD Project, a cornerstone of global digital infrastructure, has bolstered its security posture with a $137,500 grant from Alpha Omega. The funds, managed by the FreeBSD Foundation, have supported critical initiatives to strengthen the operating system's resilience against evolving threats.

The grant provided by Alpha Omega has been a pivotal investment in the Project's security posture, providing both short- and long-term improvements. Two major security initiatives were launched in June 2024: a comprehensive Code Audit of critical system components and a Process Audit of development procedures.

The Code Audit, conducted by offensive security firm Synacktiv, has been completed and provided critical insights into vulnerabilities and security practices. The Process Audit is currently in progress and is due for completion by the end of 2024. It aims to identify and address potential security improvements in the FreeBSD Project's development processes.

Several vulnerabilities were identified in FreeBSD's bhyve hypervisor, which could allow attackers to execute code and potentially escalate privileges from guest VMs to the host system. The protection offered by the Capsicum sandbox, however, was found to be generally robust. One critical kernel vulnerability and several minor issues in optional services were discovered.

The FreeBSD Project's Security Team worked diligently to manage the remediation of the vulnerabilities, issuing Security Advisories according to its stated processes. The Foundation, supported by the Security Team, led the move to publicize the full code audit report.

"The FreeBSD Foundation's sponsorship of a security audit of bhyve and Capsicum is an important step for the FreeBSD Project," said Gordon Tetlow, Security Officer of The FreeBSD Project. "Through publicly disclosing its findings, we are taking proactive measures to secure FreeBSD and the broader software ecosystem. With open source software underpinning much of today's critical digital infrastructure, The FreeBSD Foundation, in collaboration with the Alpha-Omega Project, is ensuring the security of the software supply chain."

Beyond discovering and fixing the vulnerabilities themselves, this code audit has also identified patterns and general classes for the vulnerabilities discovered. This helps the project engage with the committers to reduce the future incidence of similar vulnerabilities.

The FreeBSD Foundation used the code audit as the basis for a full review of the existing security processes and authored a **report with analysis and recommendations**. It recommends using the code audit findings to deliver developer education and training that create a security-conscious development mindset and to steward an Advisory Committee for security to materially support the FreeBSD Project in resourcing security-focused work.

The Process Audit is underway at the time of writing and is expected to be completed by the end of 2024.

Open Refactory

The main goal in 2024 was to collaborate with the maintainers of three open source projects (Apache Airflow, Jenkins, Kubernetes), perform in depth analysis of all the supply chain dependencies of the latest release of these projects, provide a unique risk signal about previously undetected vulnerabilities and deliver actionable advice on how to manage the risk. We will describe the result of the Jenkins engagement in depth.

We analyzed Jenkins v2.476 released on September 12, 2024. Jenkins is widely popular and is used by over 26 million users. It has been around for about 20 years. Jenkins v2.476 had 172 dependencies, all written in Java. Our security analysis focused on the following high severity security categories: SQL Injection, Cross-Site Scripting (XSS), Command Injection, Path Manipulation, Deserialization, XML External Entity (XXE) Injection. In addition, we looked into concurrency bugs, null pointer dereference bugs, cryptography bugs and some logical bugs.

We found 88 new bugs in 61 dependencies. No bugs were identified in 107 dependencies. And. 4 dependencies were not analyzed because the source code was not available.

Of the 88 new bugs, 18 were of high severity, 54 medium severity and 16 low severity. Of the 18 new high severity bugs, 8 are Insecure Protocol related issues (using http instead of https), 7 Data Races, 1 Remote Code Execution (RCE) via Insecure Deserialization, 1 Path Traversal and 1 Weak Cryptography issue. We also ran other well-known SAST tools on these specific artifacts. CodeQL found ten of the issues, Semgrep found one of them, Snyk found the same one and SonarCloud did not find any. **OpenRefactory found 2X more previously undetected bugs than other SAST tools.**

The 18 high severity bugs were found in 12 packages. Of these, 7 packages have the bugs fixed in a newer release; updating to a newer release fixes the supply chain. 3 packages are archived. The maintainers of Jenkins may have to switch to alternative packages to fix. This leaves 2 packages exposed. These have two vulnerabilities even in the latest release. The maintainers of Jenkins will have to work with the maintainers of these packages to get the bug fixed.

We also looked into the known vulnerabilities, as reported using SCA tools, for Jenkins. There were 9 packages with 29 high severity vulnerabilities (CVSS 7 or higher). Most interesting is the fact that there was only one package that had both unknown and known vulnerabilities. When comparing the number of SCA reported vulnerabilities with those newly discovered ones, OpenRefactory has **uncovered 2X more artifacts than were signaled by the software composition analysis (SCA) tools.**

Just as importantly, we provided actionable advice on how to manage the risk for each of these dependencies. The Alpha Omega grant has helped create a playbook that we plan to repeat for other projects in 2025. This will help the OSS maintainers not only to clean the dependencies but also to have an off-ramp to get to a less risky state.

OSTIF

Over the last 10 years, OSTIF has developed an efficient and repeatable (and therefore scalable) process for directly injecting security resources and expertise into project code bases and communities, illustrated by our public track record. This progress and impact isn't possible without the funding and support of organizations like Alpha-Omega that understand our mission and have helped us realize our goals.

In 2024, our 9th year, we have:

- Completed 21 Project Security Audits and 25 AI Project Reviews.
    - 9 more security audits in progress

    - Identified and reported:
        - 23 Critical/High findings
        - 228 Medium/Low/Informational findings
- Added or Improved 156+ Fuzzers
    - Extended coverage, new harnesses, calibrated existing fuzzers.
    - Accurate and updated fuzzers help maintainers triage reported bugs quickly. ●
- Created 17 Project Threat Models
    - Developed original documentation by security experts who provide analysis of the project's function and threat actors.
- Shared Educational/Instructional Materials about Security Audits
    - Security Audit Minimum Standards
    - Security Best Practices Guide (coming to Github end of November 2024). ●
- Fostered Security Community via Meetups and Conferences.
    - Attended FOSDEM, LF Energy Summit, FrOSCon, Open Source Summit EU, UN OSPOs for Good.
        - OSTIF at FrOScon 2024 by Amir Montazery
    - Meetup 001: Leave Home Safe: The Good, The Bad, and the Ugly with Abraham Aranguren
        - Sign Up to Present at a OSTIF Meetup!

OSTIF's primary goals are to support the security of open source projects and to educate the community about the importance of proactive security. These objectives continue to be necessary a decade later for the future success of open source and the people who make our digital infrastructure possible. While it's easy to find need in open source, it's hard to fill it in an ecosystem that primarily works off decentralized volunteers and advocates. Efforts that directly affect the work maintainers do will always be essential for the success of open source. Foundations like Alpha-Omega make it possible for us at OSTIF to bring security resources and knowledge to those who need it in a meaningful way.

In a sentence: OSTIF is a nonprofit dedicated to bridging the gap between open source projects and organizing security experts to help directly address short and long term security needs.

Trail of Bits

Alpha-Omega's funding has enabled Trail of Bits to make **significant security a sustainability improvements** to both Homebrew and the Python Packaging Ecosystem. Through the last 12 months of funding, we have:

- Implemented Sigstore-based attestations and provenance generation for the Homebrew package manager, making Homebrew the **first major packagi ecosystem with pervasive (100%) attestation coverage** for all packages in the official ind
- Implemented a user-side verification flow for all Homebrew attestations, one requires **no key or identity management.**
- Implemented key **UX, API, and CLI improvements** to the Python reference implementation of Sigstore, enabling both downstream applications (PEP 740 index attestations for PyPI) and serving as reference material for the Ruby and Go Sigstore clients.

Our work comes with multiple public materials:

- OpenSSF Securing Software Repos WG: Build Provenance and Code-signing for Homebrew
- Trail of Bits Blog: Adding build provenance to Homebrew
- Trail of Bits Blog: A peek into build provenance for Homebrew
- Sigstore Blog: Homebrew's Sigstore-powered provenance is in beta •
- SOSS Community Day 2024: Build Provenance: Lessons (so Far) from Homebrew

Completed Efforts

- Completed a 6 month (Dec 2023 — May 2024) engineering effort on Homebre culminating in 100% provenance generation for homebrew-core bottles, which in turn are responsible for over 100,000,000 downloads to macOS and Linux downstreams annually.
    - All homebrew-core attestations are generated automatically, minimizing maintenance overhead. Attestations are continuously produced within CI and are made available for external reference via GitHub's attestations panel.
    - Built an external brew verify subcommand that's suitable for non-installation verification of Homebrew's attestations, e.g. in scripti contexts.
- Completed sidecar engineering efforts on sigstore-python and gh-action-sigstore-python, including:

- Support for "v3" Sigstore bundles: #901
- Support for DSSE signing and verification: #628, #804, #904

Ongoing Efforts

- Outside of our funded work, we are continuing to engage with the Homebrew maintainers and community to make attestation verification **opt-out instead opt-in**, completing the "Generally Available" plan for homebrew-core's provenance.
- We are engaged with Alpha-Omega on a separate, ongoing (October 2024 —) engineering effort on PyPI's package lifecycle management functionality, including
  - Finalizing and implementing PEP 694, which standardizes and modernizes PyPI's upload API;
  - Minimizing and/or outright removing hard deletion support from PyPI, in favor of soft deletion ("yanking");
  - Implementing index-side project status markers, allowing maintainers to mark projects as "archived," "deprecated," and so forth.

Eclipse

Our goal is to establish the Eclipse Foundation as the leading open source community exemplifying security best practices. Achieving this would be impossible without the support of the Alpha-Omega project, which has profoundly enhanced the Eclipse Foundation's capacity to deliver and sustain significant improvements in our community's security posture. I cannot overstate the positive impact it has had. It has enabled us to establish a professional security team, implement major infrastructure upgrades, and comprehensively overhaul our security processes.

The activities have provided us with the credibility needed to position ourselves as a key stakeholder for institutions and agencies from a cybersecurity perspective. This credibility enabled us to maintain a leading role in 2024, ensuring that open source communities remain visible to policymakers, particularly within the European Commission, regarding the Cyber Resilience Act. We have worked diligently to ensure that open source foundations and communities can continue to thrive, even under the growing pressures of cybersecurity regulations.

Without a doubt, Alpha-Omega is dramatically improving the security of the entire open source ecosystem. The tech industry relies on open source to innovate and operate cost effectively.

These investments by Alpha-Omega are going to pay dividends for years to come.

Here are the key achievements in advancing security and regulatory readiness across the Eclipse Foundation in 2024:

- Established a consistent and effective vulnerability management system across over 420 projects at the Eclipse Foundation. Our team presented this approach at VulnCon and continues to share insights globally on scalable vulnerability management. This initiative has ensured that all vulnerabilities are resolved before disclosure, with consistent response times for researchers reporting new issues.
- Led the creation of the Open Regulatory Compliance Working Group (ORC WG) to support open source organisations in navigating the risks and challenges posed by the Cyber Resilience Act (CRA) and other impending regulatory changes. The team has specifically focused on evaluating dependency management solutions in anticipation of new regulations and the growing need for SBOMs.
- Leveraged Otterdog to automate provisioning and policy management for over 1,900 repositories across GitHub for all Eclipse Foundation projects, achieving 81% adoption.
- Integrated Sigstore signing into the existing common code-signing infrastructure to enhance security across all projects.
- Advanced approximately 12% of projects to level one of the newly developed Eclipse Foundation security policy framework, leading to continued engagement with the SLSA working group.
- Migrated 40% of Eclipse Foundation infrastructure applications and services to the newly integrated IAM service (Keycloak).
- Completed MFA rollout across all repositories (supported by Otterdog) and initiated plans for MFA adoption across internal applications and services as part of the Keycloak migration.

Python Software Foundation

Alpha-Omega's support has enabled the Python Software Foundation to staff our Security Developer-in-Residence Seth Larson full-time in 2024. Seth was able to build the infrastructure for automating CVE record updates and tracking CPython dependencies to generate and publish authoritative accurate Software Bill-of-Materials documents. Seth also secured CPython releases by auditing the entire process, hardening the build pipeline, and fixing issues with Sigstore verification materials. Thanks to this work Python users can be confident in the security and integrity of their Python application runtime and its dependencies.

From authoring guides and blog posts to speaking at conferences and podcasts, Seth's continued public collaboration across the Python and security communities has catalyzed improvements beyond just the Python ecosystem. Using guides that Seth authored, the Linux kernel became a CVE Numbering Authority and Nuget and Crates.io created proposals to adopt Trusted Publishers.

Supply-chain integrity with Sigstore

CPython's artifacts have been **signed using Sigstore since Python 3.11**. Sigstore will play a larger role in the future of supply-chain integrity for Python. Auditing our current Sigstore verification material found a handful of discrepancies in policy and existing Sigstore materials were migrated to the latest bundle format to allow offline verification and using the latest tools.

Seth **proposed PEP 761** which would deprecate PGP signatures for CPython artifacts to improve the lives of volunteer Python Release Managers and spur adoption of Sigstore tooling within downstream distributions of CPython. This work is done anticipating the broad availability of Sigstore signatures for Python packages **thanks to Trusted Publishers and PEP 740**.

To further this pattern of Trusted Publishers bootstrapping into build provenance in other ecosystems, Seth authored the guide for **implementing Trusted Publishers as a Package Repository** in the **OpenSSF Securing Software Repositories** working group. This guide is already being used by Nuget and Crates.io for their own Trusted Publishers implementations.

Software Bill-of-Materials for CPython

Early in 2024, thanks to Seth's work, it was **announced that CPython now provided Software Bill-of-Materials documents** (SBOMs) for both source artifacts and binary Windows installers starting in Python 3.12.2. These documents are available for all users and detail both CPython and all its dependencies and can be used for accurate vulnerability scanning of compiled binaries.

These SBOM documents meet the **NTIA Minimum Elements of an SBOM** and will aid Python users in complying with new security regulations like the Cyber Resilience Act and Secure Software Development Framework. Standards are being developed today to enable accurate SBOMs for Python packages without adding additional burden to open source maintainers.

Security Response Team and CVE Numbering Authority (CNA)

The **Python Security Response Team** (PSRT) continues to triage and fix vulnerabilities in CPython and pip. The PSRT published **11 advisories** in 2024 with CVEs ranging from low to high severity.

Seth was able to automate much of the vulnerability data process, including automatically detecting backported fixes and updating OSV and CVE record fixed versions.

Thanks to Seth's full-time status, the PSRT was able to give an "all-clear" message for the entire Python ecosystem only a few hours after the xz-utils backdoor had come to light after checking CPython, PyPI, and all packages on PyPI for the backdoored binaries.

The open source Linux kernel project announced becoming their own CNA using the guide to becoming a CNA which was authored by Seth in 2023. The PSF CNA was highlighted in the CVE Program's 25th Anniversary report for the growing involvement of Open Source projects.

Building security culture in Python and beyond

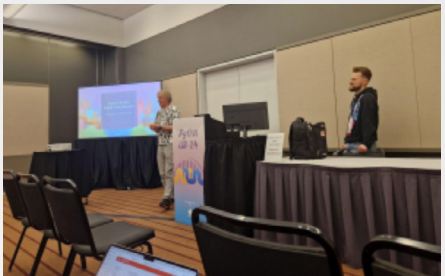
Our Security Developer-in-Residence has been actively spreading the message of "doing your part" while lowering the bar for participation in securing the Python ecosystem. Seth shares the details of his work across many blog posts on the PSF and his own personal blog.

Seth has participated and spoken at many community events, from keynoting PyCon Taiwan about security data and tools, discussing xz-utils at the summit of Python core developers, hosting an open space on vulnerability management and State of Python Supply Chain Security at PyCon US, representing open source maintainers at the inaugural OpenSSF Tabletop Session, and being a guest on popular Python and Open Source security podcasts like Talk Python, PyCast, and the OSS Security Podcast.

"Vuln Together" open space hosted by Madison Oliver and Seth Larson at PyCon US 2024

Discussing vulnerability management with open source maintainers and users at PyCon US 2024

OpenSSF Tabletop Session at Open Source Summit NA 2024



LEFT: Presenting "State of Python Supply Chain Security" at PyCon US 2024 with Michael Winser | RIGHT: Keynoting PyCon Taiwan 2024 about building a culture of security using today's tools and systems