TIDELIFT CASE STUDY

urllib3:

SECURE DEVELOPMENT PRACTICES AND PYTHON SUPPLY CHAIN IMPACT

TIDELIFT

INTRODUCTION

Making the invisible visible

Have you ever wondered what the open source maintainers that your business relies on do to keep our software running? Here's one story about urllib3, a Python package that is downloaded billions of times a year, and what it takes to keep it well maintained and up to date.

What is urllib3?

urllib3 is a HTTP client for Python. It sits at the underpinning of many components of the Python ecosystem:

- **requests**, a high-level HTTP library, is built on top of urllib3
- **boto**, the Amazon AWS SDK library, uses urllib3 to communicate with AWS
- **pip**, the Python package manager, uses urllib3

And that's just a fraction of its uses—urllib3 is used, directly or indirectly, by nearly 1 million direct dependent repositories[1] (as of September 2022). It's regularly among the top 3 downloaded Python projects[2]. **If you're building with Python, you're using urllib3.**

Why is the security of urllib3 important?

First, there's the functionality of urllib3 itself—it handles web requests, TLS/SSL, certificate validation, and more. If any of that functionality has issues, it can leave users open to compromise from hostile websites, man-in-the-middle attacks, and more.

Second, there's the sheer scope of urllib3's usage. It's downloaded over 250 million times per month, as part of usage, packaging, installation, and continuous integration workflows. Code that is installed and used that often needs to be maintained and trusted[3].

The cost of repository hijacking, trojaned code, and other systemic compromises could be staggering.

[1] https://github.com/urllib3/urllib3/network/dependents?package_id=UGFja2FnZS01MjY4MDQ4Mw%3D%3D

[2] https://pypistats.org/top

[3] https://twitter.com/ChristianHeimes/status/1251609864197025800

Who maintains urllib3?

Seth Michael Larson

Seth is the Security Developer-in-Residence at the Python Software Foundation and a Python Software Foundation fellow. Seth has been working on and maintaining urllib3 since 2016, and is currently the lead maintainer. He lives and works in Minneapolis.

Andrey Petrov

Andrey is an open source coder and former Google engineer. Andrey is the original author of urllib3, and has been maintaining it in whole or in part since 2008. Andrey currently acts as the meta-maintainer who ensures there is always a second person that can approve and release items. Andrey lives and works in Toronto.

Quentin Pradet

Quentin is a senior performance engineer at Elastic. Quentin has been a maintainer of urllib3 since 2019, and does much of the day to day development. Quentin lives and works on Réunion Island.

The urllib3 team maintains project documentation, a testing framework, and secure release practices in order to maintain their community of contributors that work on bugs, features, and other fixes[4]. They are leveraging the reliable income from Tidelift and other sources to pay contributors for some of this work. Knowing that almost 60% of maintainers have quit, or considered quitting[5], Tidelift's strategic investment in urllib3 is critical to managing all of the workload that goes into keeping urllib3 secure and stable.

[4] https://urllib3.readthedocs.io/en/stable/contributing.html

[5] https://tidelift.com/subscription/the-tidelift-maintainer-survey

WHY TIDELIFT'S PARTNERSHIP WITH MAINTAINERS IS CRITICAL TO REDUCING FIRE DRILLS

Handling of CVEs via a coordinated disclosure process

When a package is as widely used as urllib3 is, handling of CVEs is paramount. When vulnerabilities are discovered, it's important that maintainers be properly informed. This information starts a process where fixes must be developed quickly for supported branches, and the appropriate industry sources such as NIST and MITRE have accurate information. Finally, patched releases make their way to developers and users. If these processes aren't in place, it becomes a firedrill for users as they have to react to random pings from their SCA tools about vulnerabilities, with limited information about how to fix them.

Take Log4Shell as an example. In late 2021, a critical vulnerability was discovered in the Java Log4j logging framework that required immediate remediation, and when the zero-day vulnerability was announced, only a release candidate fix was available. The fallout was immense, according to an **(ISC)[6] survey:**

"Due to the ubiquitous nature of the vulnerability, 52% of respondents said their team collectively spent weeks or more than a month remediating Log4j and nearly half (48%) of cybersecurity teams gave up holiday time and weekends to assist with remediation."

urllib3 is a critical part of the Python ecosystem, in the same way that Log4j is for Java. To help in situations like this, urllib3 partners with Tidelift to handle their coordinated security disclosure process. You can read more about how in 2019 urllib3's partnership with Tidelift helped them respond to a security report from a Python maintainer, inform MITRE, and provide fixed releases all in one day on our blog[6]. Users of urllib3 had a fix for the issue on the day the vulnerability was made public, ensuring they weren't blindsided with a vulnerability they had no remediation for.

"Tidelift has made offering a comprehensive vulnerability disclosure process simple for the urllib3 team," said Seth at the time.

"This makes delivering secure code and responding quickly to vulnerabilities easy even for a small team."

[6] https://blog.tidelift.com/why-coordinated-security-vulnerability-disclosure-policies-are-important

Diagnosing the impact of a supply chain breach

In 2021, attackers were able to infiltrate Codecov, a service used for testing code coverage by over 20,000 projects and enterprises. By discovering a flaw in how Codecov built their images, they were able to modify a Codecov artifact that allowed the attackers to gain access to all environment variables that Codecov users used when accessing the service for a period of two months. The leaked environment variables could include passwords, tokens, and other secrets. Multiple organizations had their private source code and service access tokens leaked to the attackers.

For projects, it is critical that maintainers have the time to track, investigate, and respond to these issues. When learning of this attack, the urllib3 team was able to investigate what variables were leaked, audit to confirm there wasn't any unauthorized use of the leaked API token, and then change the token going forwards. If this wasn't caught quickly , it's possible that API token could have been used to compromise accounts or release trojaned software. Many users of urllib3 may not even have been aware this breach happened, but the quick work of the maintainers ensured their supply chain was not compromised.

Projects maintained by volunteer developers doing part time work are less likely to be able to detect these sorts of incidents, and may not be able to prepare a timely response. **Thanks to having the financial incentives in place that made it possible for maintainers to allocate time, complete the necessary tasks to resolve the issues, and minimize the downstream impact on the same day the breach was announced.**

Securing their maintainers' access

Seth recently shared on his blog[7]:

Some time ago I was chatting with a friend about OSS supply chain security. During the conversation I mentioned that ***I'd prefer having my bank account compromised compared to my GitHub or PyPI accounts.***

This isn't hyperbole.

In late 2021, a Javascript maintainer had their NPM account compromised. An attacker was able to release versions of the ua-parser-js package that installed cryptocurrency miners on users' machines. ua-parser-js is depended on by nearly 4000 other JavaScript packages, and users of any of those packages were potentially left vulnerable to this attack until the malicious packages were discovered and pulled.

The urllib3 maintainers know how critical their software is to the Python ecosystem and the internet at large, so to prevent a similar situation, **they have taken steps to ensure their accounts are properly secure and protected.**

They performed a number of steps to harden their accounts.

- **Email:** Using a sufficiently secure email provider; email accounts are where account resets happen, and are a key target for hijackers
- **Passwords:** Using a password manager; relying on strong auto-generated passwords stored securely rather than remembered in an ad-hoc fashion.
- **Using two-factor authentication:** for all services used email, source control, and package manager

Steps like these help prevent real attacks.To prove how important account security is, in 2022, a self-professed security researcher was able to hijack the CTX package by registering an expired domain used for the email on record for the Python maintainer. The attacker then used that domain to reset their package manager password and take over the package. They injected it with code that caused user's environment variables, potentially containing passwords and tokens, to be sent to their servers.

[7] https://sethmlarson.dev/blog/

Beyond these account security steps, the urllib3 maintainers have taken steps to secure even how their own maintainers access the repository and build software. Among the changes they have added:

- Separating permissions for their contributors between reviewers (can review changes), release managers (can perform releases), and owners (can set permissions and roles)
- Using GitHub's CODEOWNERS feature to ensure that any changes that affect the build and release pipeline requires specific approval by the core maintainers
- Using API tokens with limited scope for all build, test, and release processes, instead of using their elevated personal credentials. This use of least-privilege architecture limits the risk of any single credential compromise.

All these steps work to ensure that their accounts and the software itself is less vulnerable to hijacking and compromise, and every user benefits from this increased security.

Backwards compatibility

"When you're at the bottom of the world, any change is gonna ruin someone's workflow," Seth said. "Our team is super cautious. It's something we pride ourselves on."

You may ask, "what does backwards compatibility have to do with security?" If a piece of software takes extreme care to maintain backwards compatibility, then that means that upgrades become painless. Users can upgrade with confidence, and when you can upgrade with confidence, you can more easily pull in fixes and security remediations.

Let's go back to the Log4Shell vulnerability. The vulnerability affected the 1.x series. While initial fixes were added for that series, going forwards, only Log4j 2 is maintained, which has a different API for developers to use. To stay current with Log4j, developers and users either need to migrate their code to a new API, or install an additional API bridge package. This hinders their ability to stay current and apply future security updates.

The urllib3 team works to maintain backwards compatibility — each change passes a rigorous test suite to ensure there are no unintended breaks to functionality. They go above and beyond to ensure that users will not have their workflow interrupted — they still support in their 1.x branch end-of-life Python releases such as 2.7 or 3.5. **This ensures that any time they do have to release an important change that users will be able to quickly pick up a release.**

They are continuing this work as they prepare a new major release, 2.0, where they are retaining 99% functional API compatibility[8] in the new version, with a goal of making it "the simplest major version upgrade you've ever completed"

Their policy ensures that all users have a smooth upgrade path to new releases, picking up security fixes when and where necessary.

Automating and streamlining release processes

Manual processes lead to mistakes. The more complicated the process, the higher the chance of mistakes. Releasing software is one of those complicated processes — from tagging the release in source control, to creating the release artifacts, to pushing to the package manager, there are many steps that are all often run with elevated privileges. **And if any of them are done improperly, the released software may have the wrong code, credentials could be compromised, or more.**

Seth was having "the worst possible day" one day, where everything was going wrong, when the time came to perform a urllib3 release. He realized that to make this better, the process had to become easier and more automated. Working with Quentin, they did the work to build a checklist-based automated release process, with built-in ability to get approvals from maintainers. **By automating the release process, the maintainers can be sure that the same process is run every time, and the software is built the same way every time. They can embed any credentials that are needed, and scope them to only the permissions that are required. They can hook in their continuous integration processes, to ensure that any software is fully tested before it goes out.**

Now when releases are needed, any maintainer can contribute and suggest a release candidate. That candidate is then sent to CI and tested, and if it's approved, the tag, build, and release process is all run automatically via GitHub actions. This ensures that the release process is safe, repeatable, and reliable, and that their users are protected from potential compromise and accidental oopses.

[8] https://urllib3.readthedocs.io/en/stable/v2-roadmap.html

Reproducible and verifiable builds

The urllib3 team's automation of their build process makes sure they know their builds are being done the same way every time. But to fully trust code, it's good to have more assurances than that. You want to know that the artifacts are built from the code that they're supposed to be built from. You want to know that if you build the same code, you'll get the same artifact. If you can prove both of those items, you can have confidence to see the provenance of your code—from upstream source control, to the package manager, to your downstream environment.

In 2015 malware was discovered that attacked the Apple Xcode development platform. Infected versions of Xcode would inject additional malware silently into applications built by that version of Xcode. That malware ended up being released as part of multiple iOS applications.

By working to create reproducible builds with a chain of trust from source through the build system to the final artifact, the urllib3 team can bypass this class of attacks, and ensure that their build processes are not silently compromised.

The urllib3 team has done a large amount of work to ensure that this is possible for their code. In 2022 they moved to the Flit build system and adjusted their build process to ensure that every time a specific set of source code is built, it produces the same, byte-for-byte, artifact[9] — **all their builds are fully reproducible.**

After ensuring their builds are reproducible, the maintainers moved to ensuring their builds are verifiable — that they can certify that each build came from the proper release branch and tag, and that nothing has been tampered with along the way. First they integrated support for Sigstore to generate a chain of trusted signatures of their built artifacts, and then worked beyond that to perform provenance generation and verification via Supply Chain Levels for Software Artifacts (SLSA). urllib3 now supports being verified as compliant with SLSA Level 3, which states that the **"source and build platforms meet specific standards to guarantee the auditability of the source and the integrity of the provenance respectively."** Downstream users can now be assured that the software they get is the software that they intended to get, and can begin to build their own internal guarantees based on this provenance that they get from upstream.

[9] https://github.com/urllib3/urllib3/pull/2549

Deprecation of `[secure]` extension and making secure-by-default

In 2015, urllib3 introduced the **`[secure]`** extension. This extension would install the **`pyOpenSSL`** package and use it for connection security, rather than Python's built-in 'ssl' module, due to additional features like SNI that pyOpenSSL supported at the time.

Downstream users such as 'requests' and other projects would enable **`urllib3[secure]`** to get this extension. Some other packages and users would follow suit — after all, it says "secure" right there in the name.

Fast-forward to 2022, and Python's built-in **`ssl`** module contains all the needed features of **`pyOpenSSL`**. While the extension was once used to ensure security, it's now no more secure than the default, and presents an extra code path and set of dependencies that users will get–that can actually open them up to more vulnerabilities.

The urllib3 maintainers want to remove this extra extension, but responsibly so that their downstream users don't break. They know the stress that inflicted change can be for other open source maintainers. In Seth's own words:

> "Most projects would not have done that but I wanted to avoid causing pain to downstream projects, and avoid barrages of questions from their users asking each project to remove the extra to stop deprecation warnings. Many firestorms happen because of deprecations — projects can get blindsided sometimes. I didn't want to be the cause of that for anyone."

This involves:

- Creating a new package that 'urllib3[secure]' depends on, that emits a deprecation warning to inform users to move away from it
- Have that deprecation warning point to a page on what's going on, and how to fix it
- Track installation of this new package, to see how often people are using urllib3[secure] (and whether it goes down over time)
- Creating a new dataset to find out which packages directly depend on `urllib3[secure]`
- Creating an issue for each package, and eventually pull requests to fix them

All of that is a lot of extra work for maintainers to take on, and that work takes time — which the urllib3 maintainers only have so much of.

Yet they take this time to make sure that their downstream users don't have to deal with pain and can continue to use and upgrade in confidence.

All this work to improve urllib3 is great, as it helps secure and maintain a foundational piece of the Python ecosystem. Seth and the urllib3 team, however, thinks bigger — how can they take the lessons they've learned and spread them across the ecosystem and make software better as a whole?

Documenting processes for package maintainers

The urllib3 team went through many steps, as noted above, to secure their accounts and how they used them to develop software. This then became a blog post on how developers should practice secure behaviors[10].

Improving scores against industry standards

If you can't measure it, you can't improve it. — Peter Drucker

The urllib3 team is constantly looking for ways to improve the security posture of their code. Recently, the Open Software Security Foundation introduced their Scorecard project[11], which scans a project repository and scores it against a number of best practice criteria for secure development.

The urllib3 team investigated their project's scorecard[12], and immediately saw room for improvement. Within just a couple of weeks, their scores had improved and **they became the first Python project to score at least a 9.0 out of 10 (as of September 2022)**.

This measurable improvement in their security posture shows their commitment to best practices to protect their users.

After working on their own improvements of their OpenSSF scorecard score, Seth was curious how the rest of the Python ecosystem stacks up. This led to the analysis of scorecards across the entirety of PyPI[13] as a public dataset that can be used for analysis to drive improvements.

[10] https://sethmlarson.dev/blog/security-for-package-maintainers
[11] https://github.com/ossf/scorecard
[12] https://deps.dev/pypi/urllib3
[13] https://twitter.com/sethmlarson/status/1557350262779641859

Improving the standards themselves

As part of improving their own project's security, Seth and the urllib3 team are improving the standards they are asserting to. This includes discovering multiple bugs in provenance generation across tools, and improving OpenSSF scorecard checks to include SLSA provenance [14].

Building a standard secure framework for developers

Having one package do the work to use secure, verifiable, development and release practices is great. **But what if they all could?**

Seth is taking his lessons learned in building a secure framework for the development of urllib3 and creating a secure project template for Python developers[15].

By cloning this template, a project can immediately start taking advantage of Seth's research and create a project that can build using and attest to secure practices.

All of these initiatives benefit the ecosystem as a whole—not just Python, but across open source software as a whole as standards are improved.

Every user of software that adopts these practices and standards will benefit from the work that the urllib3 team has done.

[14] https://github.com/ossf/scorecard/issues/1776

[15] https://github.com/sethmlarson/secure-python-package-template

THE HOW

How urllib3 maintainers can do this work

It's not a spoiler—all of these things take time.

Handling CVEs? Takes time to verify reports, produce fixes, and perform releases.

Diagnosing breaches? Takes time to investigate what was breached, and what the consequences might be.

Building automated release processes? Time to trial and error different setups, test release processes, and hook platforms together.

Improving packages against industry standards? Time to investigate each new standard, and perform necessary remediation (or fix the standard[16]).

Making reproducible, verifiable builds? Time to research new tools, implement them, and migrate to new build processes.

Deprecating extensions and making it secure by default? Time to analyze the ecosystem, find users, and get all of them to update.

Quentin Pradet, one of the urllib3 maintainers, said "the most important thing to maintainers is time. Time to do the development work, but also time to do all the other things—research new software standards, perform non-development maintenance tasks, secure infrastructure, and even recruit new potential maintainers."

How do you give maintainers time? **You don't do it by giving them a list of tasks from your software composition analysis (SCA) tool.** You don't do it by going to their issue tracker to ask them about a two-year old CVE they already have marked as not exploitable. You don't do it by giving them new unfunded mandates[17] on how to secure their software.

The easiest way for an organization to ensure maintainers have time... is to pay them.

"People are willing to absorb way more pain [if they are paid]," Seth said. "Paying people to be around a project just makes it more secure intrinsically. There's always someone around to deal with the burstiness aspect of open source issues."

[16] https://github.com/ossf/scorecard/issues/2196

[17] https://blog.tidelift.com/pay-to-play-dont-expect-maintainers-to-solve-your-supply-chain-issues-for-free

Not every maintainer has time, all the time... but if you're paying the maintainers, someone will always be there no matter what.

Paying the maintainers **can also scale time**. urllib3 is taking the money they make via Tidelift, GitHub Sponsors, and other funding sources and using it to set up bounty programs to incentivize more time to fix issues. Their deprecation of the '[secure]' extra utilizes bounties for providing fixes for other projects that still depend on it. They set bounties for additional maintenance work, such as ensuring their software works with upcoming Python cryptography changes ahead of time[18]. They also use these bounties as recruiting tools for new maintainers–contributors who tackle multiple bounties can be invited to become co-maintainers.

Paying the maintainers is the way

In short, the urllib3 maintainers are able to do all these things a professional software organization might do, because they are paid to do professional software development. That's no accident — they could not do all these things if they were doing it in their spare time while they did other work to keep their lights on.

The urllib3 team is a group of open source professionals, doing professional open source work, all aligned on keeping the code secure, backwards compatible, and easy for the entire world to continue depending on. They then take this work, and make it available for other projects to build on, securing the ecosystem as a whole. That sort of work doesn't just happen — it requires time, effort, and the money that supports that time and effort. Is your organization paying the maintainers like Seth to provide these professional services? If not, you should be.

[18] https://github.com/urllib3/urllib3/issues/2686