

Bootstrap script exposes PyPI to domain takeover attacks

 reversinglabs.com/blog/bootstrap-script-exposes-pypi-to-domain-takeover-attack

November 26, 2025

Proving the road to takeover is paved with setuptools alternatives, the script for a popular Python package for building and installing PyPI

packages leaves them vulnerable.

Python bootstrap script exposes PyPI to domain takeover attacks

ReversingLabs researchers have discovered vulnerable code in legacy Python packages that could make possible an attack on the Python Package Index (PyPI) via a domain compromise. Although the vulnerable code is mostly unused in modern development environments, it may still be used in legacy production.

RL Spectra Assure Community's machine learning model, which detects packages with behaviors similar to known malware, found the vulnerability in bootstrap files for a build tool that installs the Python package *distribute* and performs other tasks in the bootstrapping process. The scripts automate the process of downloading, building, and installing the required libraries and tools. Specifically, when the bootstrap script is executed, it fetches and executes an installation script for the package *distribute* from *python-distribute[.]org* — a legacy domain that is now available for sale in the premium price range while being managed to drive ad revenue.

Python packages that include a bootstrap script that access the released domain are affected, including *tornado* (an asynchronous networking library), *pypiserver* (used for setting up private PyPI-like servers), *slapos.core* (overlay operating system for distributed POSIX infrastructures), *roman* (for converting numerals), *xlutils* (editing excel files), *testfixtures* (unit tests and docs) and at least a dozen other popular packages on PyPI.

Discussion

When it comes to utilizing the power of open-source code, it's important to have an agreed-upon way of packaging the artifacts and resolving dependencies during installation (among many other things). It may be taken for granted today that almost all PyPI packages use *setuptools*, a library that simplifies the installation processes. However, this hasn't always been the case.

Packaging matters

[In the dawn of Python packaging](#), the *distutils* package was intended to be used for that purpose and was included in the standard library up to Python 3.12 when it was removed, having been deprecated in Python 3.10. The *setuptools* library was developed to fill in the missing features, like automated download and installation of package dependencies. The package provided a command-line utility called *easy_install*, which did exactly that. At some point, *distribute* forked from *setuptools* due to perceived slow development of *setuptools* by the community, but was merged back a couple of years later in 2013. This relationship can be seen in Figure 1.

There was also an effort to implement the missing features in what was envisioned to be a new add-in to the Python standard library in the form of *distutils2* package, which would be renamed to *packaging* for later Python versions. That effort was unfortunately abandoned, and what was left was a mess of packaging utilities mostly compatible with each other, but in fact, not in situations where it mattered.

 Packaging utilities used by the PyPI community in early 2010s



Figure 1: Packaging utilities used by the PyPI community in early 2010s

This situation was later cleared up by the [Python Packaging Authority](#) (PyPA) with [packaging](#) becoming a key project and *setuptools* becoming a *de facto* industry standard. The later PEP 517 ([Python Enhancement Proposal](#)) liberalized the choice of build backend in a controlled manner by providing the developers with a standard build backend interface for interacting with package source trees and source distributions. However, before that was done, developers and providers of application build tools had to work with — and support — each approach that gained sufficient traction in the open source community. In doing so, they laid the groundwork for potential supply chain compromises via a domain takeover attack.

The bootstrap.py nostalgia

In the midst of all the available options for packaging, a different tool emerged: *zc.buildout*, which was an automation tool used in the 2010s to build complex applications with high granularity while ensuring reproducibility, e.g. using different versions of the same package within a single project.

The *zc.buildout* tool provided a *bootstrap.py* file to its users to simplify the installation of the buildout package — enabling the users to pin buildout versions in *.cfg* files. With the intention to support both *setuptools* and *distribute* users, it left the door open to a domain takeover attack — an opportunity available to attackers since 2014 that appears to not have been seized to this day.

The bootstrap scripts differ from each other in whether they use *distribute* by default, or only if the appropriate flag was provided. In any case, the user would execute the bootstrap script. Or, alternatively, the script might be invoked by a Makefile, a standard text file used by the *make* utility that automates software builds. If the script was instructed to use *distribute*, it would fetch and execute its installation script. As it can be seen in Figure 2, the *distribute* installation script is fetched from *python-distribute[.]org*, a domain which has been up for sale since 2014.

 Code fetching and executing the distribute setup in the bootstrap.py file



Figure 2: Code fetching and executing the distribute setup in the bootstrap.py file

While PyPI's resolution of the issue would push the developers to abandon the *distribute* package (mostly what has happened), but the migration was voluntary, not mandatory, and therefore was not done thoroughly. Many packages continued shipping the bootstrap script, which installed *distribute* either by default or when the users provided the appropriate command-line flag.

Today, *distribute* is figuratively dead — merged back into *setuptools* with a published compatibility layer that simply installs the version of *setuptools* published after the *distribute* merger took place. But the bootstrap scripts still try to install *distribute*, opening the doors to the execution of arbitrary code hosted on the abandoned domain – thus putting the developers who run a bootstrap script at risk of fetching and executing malicious code. Here is a list of some interesting examples:

- *slapos.core* at the time of writing still includes a bootstrap script
- *tornado* includes a [bootstrap script for the maintainers](#), but not as part of the published package
- *pypiserver* [removed the bootstrap script](#) only this year
- *imio.pm.locales* v4.2.19 was released in 2024 and contained the bootstrap script and a *Makefile* that executes it. Bootstrap script was [removed later](#), but the *Makefile* is still invoking the script (at the time of writing).
- *pyquery* contained the script up to 2021 when it was [removed](#).

A flag to trigger the bad old days

It is important to stress that the bootstrap files aren't executed automatically during the package installation. Thus, the potentially dangerous behavior of the bootstrap script could manifest itself only with manual execution of the script by an unsuspecting developer or via the *Makefile*. Nonetheless, it leaves an unnecessary attack surface in modern packages that the attackers could exploit if they were to provide the developers with code that causes the execution of the bootstrap script.

 Proof-of-concept script that exploits the vulnerability in slapos.core

Figure 3: *Proof-of-concept script that exploits the vulnerability in slapos.core*

To demonstrate how this could happen, RL researchers created a simple proof-of-concept script that exploits the vulnerability. It should be noted here that the team chose to use Python 2 in producing this example, because the bootstrap script itself is written in Python 2 and can't be

executed with Python 3 without modifications. The code uses the bootstrap script shipped with the *slapos.core* package, as shown in Figure 3. It locates the bootstrap script (lines 7 to 10) and stores the path into the list of command-line arguments for later access (line 13). This is done because the bootstrap script re-executes this script with an *-S* flag in order to not load the *site-packages* into the path, as shown in Figure 4. Python module *imp* is used to load the source as a module, with the flag *-d* set before calling it to make the bootstrap script use *distribute*.

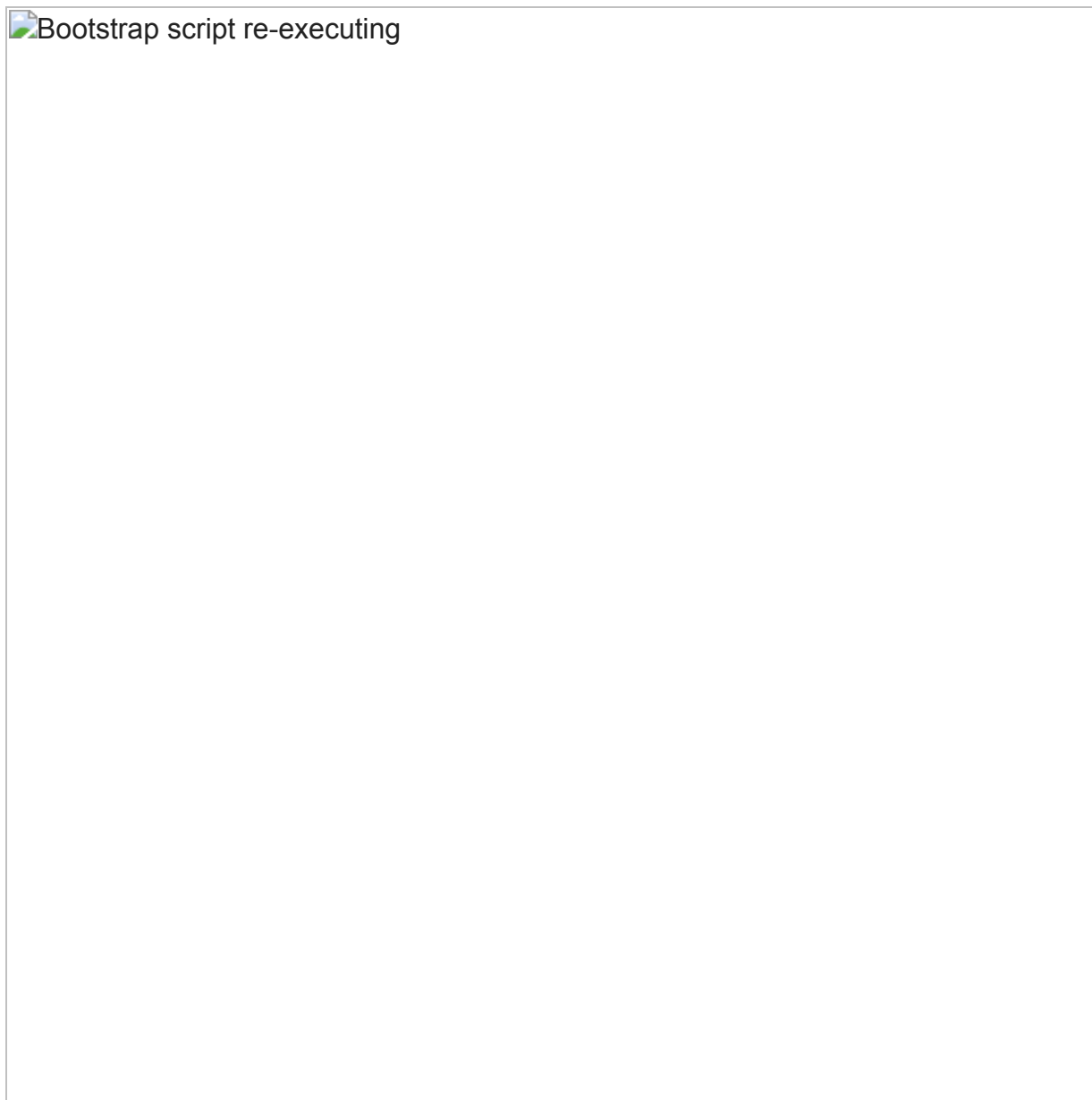


Figure 4: Bootstrap script re-executing

The only changes RL made in the bootstrap script were to comment out the fetching and execution from the domain in question and instead print a message, the URL it was about to access and exit the programme. The output can be seen in Figure 5.

 Terminal output after running the PoC script



Figure 5: Terminal output after running the PoC script

C2 real estate: A domain for sale

A more detailed look at how the *python-distribute[.]org* domain has been managed since PyPA took on the task of bringing order into the Python packaging practices found that the *distribute* fork of *setuptools* was [merged back into setuptools in June of 2013](#). [DNS history records](#) shed some light on how that affected the management of the domain. The records show that the domain was dropped in October of 2013, which was [noticed by the users of zc-buildout](#), only to be created again two days later because it was [dropped by mistake](#). It was then held for another year, dropped again in October of 2014 and parked to drive ad revenue since December of 2014, shown in the screenshot from DNS history in Figure 6. This could easily be exploited by an attacker, because, since it was dropped for good, anybody could have bought the domain and served whatever malicious code they wanted as their bootstrap script.

 DNS history showing that the domain was parked in 2014



Figure 6: DNS history showing that the domain was parked in 2014

Was this a unique event?

Such an attack would not be new. In 2023, the popular npm package [fsevents](#) was compromised by an attacker who had taken control of an unclaimed cloud resource. This domain takeover issue was noticed in 2023, and affected package versions from 1.0.0 up to 1.2.11, which is reflected in [CVE-2023-45311](#). The versions in the specified range fetch an executable from a hardcoded URL pointing to a cloud storage resource at `https://fsevents-binaries.s3-us-west-2.amazonaws.com`. The resource was claimed by the threat adversary, delivering malicious executables to the users installing the package.

The *fsevents* example shows how domain takeovers have been exploited in the wild already and how important it is to be proactive in catching such risks. RL's [Spectra Assure Community](#), helps detect the vulnerability in both *fsevents* and packages affected by the inclusion of a bootstrap script fetching and executing content from the *python-distribute.org* domain (Figure 7).

 RL's community page shows DomainTakeover in pypiserver, version 2.3.2



Figure 7: RL's community page shows *DomainTakeover* in *pypiserver*, version 2.3.2

The programming pattern is the problem

With PyPI being an open-source community with diverse approaches to solving the problem of packaging, it was no surprise that without any oversight or guidelines, the solutions studied here ran into compatibility issues. Ensuring compatibility ad-hoc meant employing a hardcoded Internet domain. However, that exposed build automation tools like *zc.buildout* to receiving malicious scripts from that domain during the bootstrapping process — should the domain fall under the control of malicious actors. The issue lies in the programming pattern that includes fetching and executing a payload from a hardcoded domain, which is a pattern commonly observed in malware exhibiting downloader behavior.

The Python community has matured and overcome many difficulties related to packaging, including moving away from *distribute* to more reliable and carefully managed alternatives such as *setuptools*. However, the failure to formally decommission the *distribute* module allowed vulnerable bootstrap scripts to linger and left unknown numbers of projects exposed to a potential attack.

The lesson for the open-source community is clear: a reliance on hard-coded domains comes with increased risks of supply chain compromises. As a result, such domains shouldn't be released from the control of trusted owners without first ensuring that users have migrated away from accessing them. This also serves as a reminder about the larger problem of code rot that plagues the open-source community and how important it is to ensure that your products don't rely on outdated and insecure coding practices. In the case of *python-distribute[.]org* domain, there is no documented abuse of this vulnerability, but the same can't be said for the *fsevents* package on npm.

Indicators of Compromise (IoCs)

Indicators of Compromise (IoCs) refer to forensic artifacts or evidence related to a security breach or unauthorized activity on a computer network or system. IOCs play a crucial role in cybersecurity investigations and cyber incident response efforts, helping analysts and cybersecurity professionals identify and detect potential security incidents.

The following IOCs were collected as part of ReversingLabs investigation of this software supply chain vulnerability.

PyPI packages:

package name version range

[pypiserver](#) >=1.1.1, < 2.4.0

[slapos.core](#) >=0, <=1.19.0 (newest version at the time of writing)

[roman](#) >=2.0.0, <3.2

[xlutils](#) >=1.6.0, <2.0.0

[testfixtures](#) >=2.3.4, <3.0.2

[imio-pm-locales](#) >=4.1.18.1, <4.2.20

[pyquery](#) >= 1.2.10, <2.0.0

Bootstrap scripts:

sha1

357f2fe2684c54339fb78ff447d8cbc127071633

76eccfddbec55f435145e2620826ba30e22e5653

8285c1c8188f198e9440c97e8aed933704b32d82

df52f054feb6f2e0df2db1a22c5781d3c56e8ffa

e68c89e553a2e70380492bdb8cfb74c224456766

3d4970cbd4540e4bf1dc67ca554228b6369629d8

9f02932adcba7f4c681df66b597df10f34b134

0cbb7df358b8772f4d5f2d346fe87bf6f5b911c1

b6664ae860ca4793e368abb0c569ddefcbc7ac96

3cdd1cfcc254a382338588e406493924d4aadb4f

cc298ff510dee97bf5f8abac68bafc22bcfadc11

Tags: [Threat Research](#)

Topics

SUBSCRIBE

Get the best of RL Blog delivered to your in-box weekly. Stay up to date on key trends, analysis and best practices across threat intelligence and software supply chain security.