

Title	Automatic Execution of Code Upon Package Download on Python Package Manager
Subtitle/description (short - 1 line)	Automatic code execution is triggered upon downloading close to a third of the packages on PyPi.
Author	<i>Yehuda Gelb</i>
Co-author(s)	Aviad Gershon

*Automatic code execution is triggered upon downloading approximately one third of the packages on PyPi.*

A worrying feature in pip/PyPi allows code to automatically run when developers are merely downloading a package.

This feature is also alarming due to the fact that a large number of the malicious packages we are finding in the wild use this feature of code execution upon installation to achieve higher infection rates.

It is important that Python developers understand that package downloading can expose them to increased risk of a supply chain attack.

## Intro

When executing the well-known “pip install <package\_name>” command, users may expect code to be run on their machine as part of the installation process. One source of such code usually resides in the setup.py file of Python packages.

When a Python package is installed, pip, Python’s package manager, tries to collect and process the metadata of this package, such as its version and the dependencies it needs in order to work properly. This process occurs automatically in the background by pip running the main **setup.py** script that comes as part of the package structure.

```

from setuptools import setup, find_packages
from os import path
from codecs import open

with open(path.join(SCRIPIT_DIR, 'README.md'), encoding='utf-8') as f:
    long_description = f.read()

setup(
    name='prp1',
    python_requires='>=2',
    version='1.0.5',
    description='This package is a part of Checkmarx SCS research process regarding issue
#1884',
    long_description_content_type='text/markdown',
    long_description=long_description,
    url='https://github.com/checkmarx',
    author='Checkmarx SCS Research',
    author_email='supplychainsecurity@checkmarx.com',
    license='Apache v2',
    classifiers=[
        'Development Status :: 5 - Production/Stable',
        'Intended Audience :: Developers',
        'Topic :: Software Development :: Build Tools',
        'License :: OSI Approved :: Apache Software License',
        'Programming Language :: Python :: 2',
        'Programming Language :: Python :: 2.5',
        'Programming Language :: Python :: 2.6',
        'Programming Language :: Python :: 2.7',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.3',
        'Programming Language :: Python :: 3.4',
        'Programming Language :: Python :: 3.5',
        'Programming Language :: Python :: 3.6',
        'Programming Language :: Python :: 3.7',
        'Programming Language :: Python :: 3.8',
        'Programming Language :: Python :: 3.9',
    ],
    keywords='prp1 Checkmarx SCS research',
    packages=['prp1'],
    setup_requires=['requests'],
    install_requires=['requests']
)

```

*setup.py example*

The purpose of **setup.py** is to provide a data structure for the package manager to understand how to handle the package.

However, the **setup.py** file is still a regular Python script that can contain any code the developer of the package would like. An attacker who understands this process can plant malicious code in the **setup.py** file, which would then execute automatically during the package's installation. In fact, much of the malicious packages we are detecting contain malicious code in the **setup.py** file.

## What if we just download the package rather than installing it?

In addition to the “install” command, pip provides several more options, among them is the “download” command. This command is intended to allow users to download packages’ files without the need to install them.

There could be various reasons someone would need this. A developer may want to look into the package’s code before using it. A user may want or need to perform a security check, or perhaps even observe the `setup.py` file for any anomalies.

As it turns out, executing the command “*Pip download <package\_name>*” will run the **setup.py** file, as well as any potentially malicious code contained within it. It may surprise you, but this behavior is not a bug but rather a feature in the pip design. Users who intentionally only download a package do not expect code to run on their system automatically.

As a matter of fact, this concern was expressed in an issue from 2014 on the pypa project <https://github.com/pypa/pip/issues/1884>, yet it was not addressed, and the issue continues to exist to this day.

## The .whl file type

Python wheels are essentially **.whl** files that are part of the Python ecosystem and bring various performance benefits to the package installation process. But that is not the only thing that wheels bring to the table. In the past, when Python code was built into a package, the result would be a **tar.gz** file that would then be published to the PyPi platform.

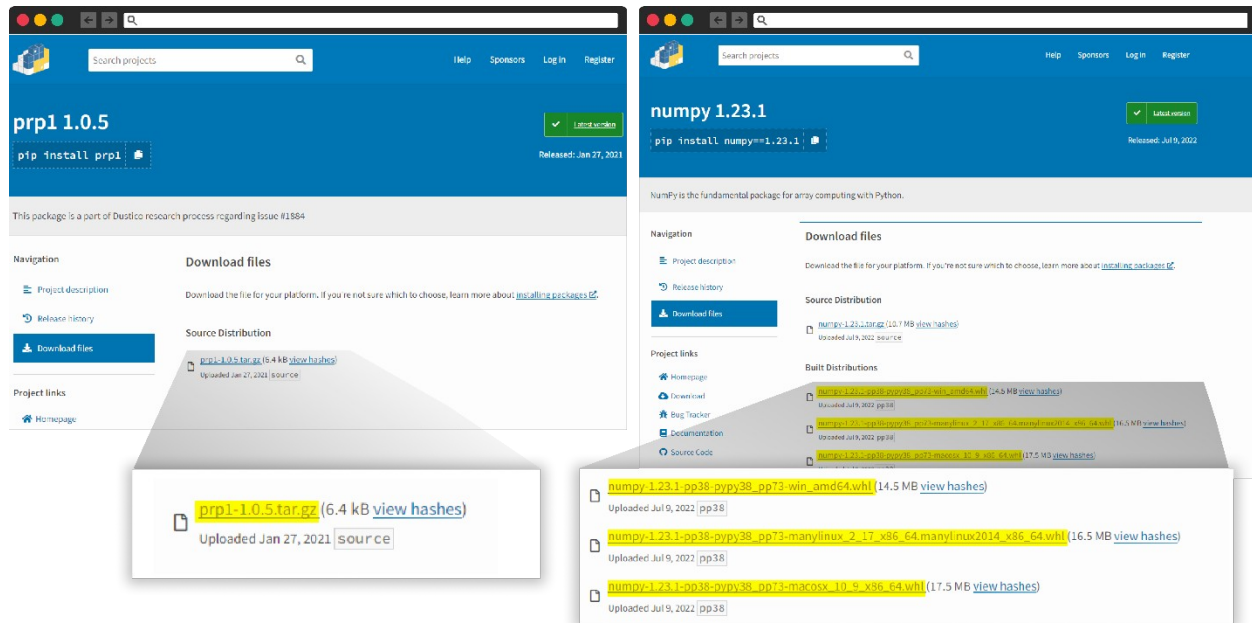
But suppose you've recently tried downloading or installing a Python package using [pip](#). In that case, you may have noticed Python supplying you with a **.whl** file. The reason for this is when developers build a Python package using, for example, the “*pip -m build*” command, in newer pip versions, pip automatically tries to create a secondary **.whl** file in addition to the **tar.gz** file, which is then published together to the Python Package manager platform. When a user downloads or installs this package, PIP will by default deliver the **.whl** file to the user's machine. The way wheels work cuts the **setup.py** execution out of the equation.

## Why is the setup.py still relevant?

Even though pip defaults to using wheels instead of **tar.gz** files, malicious actors can still intentionally publish Python packages without a **.whl** file. When a user downloads a Python package from PyPi, pip will preferentially use the **.whl** file, but will fall back to the **tar.gz** file if the **.whl** file is lacking.

## Is there anything you can do about this?

Currently, there are actions users can take to prevent automatic execution upon package download. One action is checking the package file contents at <https://pypi.org/project/<package>/#files> and observing if a `.whl` file is present. If there is a `.whl` file, the user can feel confident they will receive the `.whl` file, and no code will be executed on their machine.



The image displays two screenshots of the PyPI website. The left screenshot shows the page for package `prp1 1.0.5`, released on Jan 27, 2021. It features a search bar, navigation links, and a 'Download files' section. A callout box highlights the `prp1-1.0.5.tar.gz` file (6.4 kB) with a 'view hashes' link and a 'source' link. The right screenshot shows the page for package `numpy 1.23.1`, released on Jul 9, 2022. It features a search bar, navigation links, and a 'Download files' section. A callout box highlights several wheel files, including `numpy-1.23.1-pp38-pypy38_pp73-win_amd64.whl` (14.5 MB), `numpy-1.23.1-pp38-pypy38_pp73-manylinux_2_17_x86_64.manylinux2014_x86_64.whl` (16.5 MB), and `numpy-1.23.1-pp38-pypy38_pp73-macosx_10_9_x86_64.whl` (17.5 MB), each with a 'view hashes' link.

If there is **only** a `tar.gz` present, a user can use a safe method of download such as working directly with PyPi's "simple" API: <https://pypi.org/simple/<package-name>/>. For example, when using the package listed above, `prp1`, a user can download it from the following link <https://pypi.org/simple/prp1/>.

## Links for prp1

[prp1-1.0.0-py2.py3-none-any.whl](#)

[prp1-1.0.1-py2.py3-none-any.whl](#)

[prp1-1.0.2-py2.py3-none-any.whl](#)

[prp1-1.0.3-py2.py3-none-any.whl](#)

[prp1-1.0.4-py2.py3-none-any.whl](#)

[prp1-1.0.5.tar.gz](#)

### Conclusion

Code execution upon installation is one of the features attackers use the most in open-source attacks. Developers opting to download, instead of installing, packages, are reasonably expecting that no code will run on the machine upon downloading the files. However, PyPi includes a feature allowing just that—execution of code on the user’s machine when all that was requested was a file download.

It is possible to protect yourselves from suspicious package by following the steps detailed above.

As always, we are releasing similar blogs to help keep the open source ecosystem safe and raise the awareness of Python developers to this issue so they can avoid unwanted consequences.