Python Security Documentation

Release 0.0

Victor Stinner

Nov 10, 2017

Contents

This page is an attempt to document security vulnerabilities in Python and the versions including the fix.

CHAPTER 1

Pages

1.1 Packages and PyPI

1.1.1 Check for known vulnerabilities

- https://github.com/pyupio/safety-db and https://pyup.io/
- safety package: Safety checks your installed dependencies for known security vulnerabilities.

1.1.2 GPG

- Verifying PyPI and Conda Packages by Stuart Mumford (2016-06-21)
- Sign a package using GPG and Twine

1.1.3 pip security

- pip: Implement "hook" support for package signature verification

1.1.4 PyPI

- PEP 458 – Surviving a Compromise of PyPI (27-Sep-2013)
- PEP 480 – Surviving a Compromise of PyPI: The Maximum Security Model (8-Oct-2014)
- Making PyPI security independent of SSL/TLS by Nick Coghlan

1.1.5 Vulnerabilites in the Package Index

Index Vulnerability: Unchecked File Deletion

Improper checking of ACLs would have allowed any authenticated user to delete any release file hosted on the Package Index by supplying its md5 to the `:files` action in the pypi-legacy code base.

- Disclosure date: **2017-10-12** (Reported via security policy on pypi.org)
- Disclosed by: Max Justicz

Fixed In

- PyPI "Legacy Codebase" (2017-10-12) fixed by commit 18200fa (2017-10-12)

Audit

After mitigating the attack vector and deploying it, the responding Package Index maintainer worked to verify that no release files had been improperly removed using this exploit.

The Package Index maintains an audit log in the form of a "Journal" for all actions initiated. It was determined that exploitation of this attack vector would still remove files via the existing interface an audit log would still be written.

Using this information, we were able to reconstruct the users with access to legitimately remove release files at point in time of each file removal using the audit log.

The output of this script were used to determine that no malicious actors exploited this vulnerability. All flagged journal entries were related to one of the following scenarios:

- Username updates that were not properly updated in the Journal
- Administrator intervention to remove packages

Timeline

Timeline using the disclosure date **2017-10-12** as reference:

- 2017-10-12: Issue reported by Max Justicz following guidelines in security policy on pypi.org
- 2017-10-12 (**+0days**): Report investigated by Ernest W. Durbin III and determined to be exploitable
- 2017-10-12 (**+0days**): Fix implemented and deployed in commit 18200fa
- 2017-10-12 (**+0days**): The audit journals maintained by PyPI were used to reconstruct the full history of file removals to determine that no malicious deletions were performed.

PyPI credential exposure on GitHub

Introduction

A common mistake made by users is committing and publishing "dotfiles" containing private material such as passwords, API keys, or cryptographic keys to public repositories on services such as GitHub.

Compounding this issue, the Python packaging ecosystem historically and currently encourages—albeit with some level of caution—the use of a `.pypirc` file for storage of passwords consumption by packaging tools. For a summary of the dangers of this methodology, see this article on securing PyPI credentials.

With ever strengthening search tools on GitHub attackers are able to formulate queries which quickly identify and obtain credentials from such hosting sites.

- Disclosure date: **2017-11-05** (Reported via security policy on pypi.org)
- Disclosed by: Joachim Jablon

Report

The PyPI security team was notified by Joachim Jablon that `.pypirc` files containing valid PyPI credentials were obtainable with a straightforward search and scrape of GitHub.

Using tools developed by the reporter the PyPI security team was able to identify 77 valid PyPI logins in 85 public files published to GitHub. These 77 logins had maintainer or administrator access to 146 unique projects on PyPI.

Audit

Action Taken by PyPI team

The PyPI security team followed up by auditing and extending the Proof of Concept tools supplied by the reporter to verify the report.

After running the tooling against the full result set of the GitHub code search the PyPI administrators unset the passphrases for all valid logins found and issued an administrative password reset for exposed users.

Additionally an audit of PyPI's journals showed no signs of malicious access for the exposed accounts.

The email sent to affected users took the form

Recommendations

All users of PyPI should ensure that their PyPI login credentials are safe and have not been inadvertently exposed in a public repository of dotfiles, in the root of a project directory, or in some other public or shared medium.

The PyPI team does not have the resources to search or scrape all such services and may not have identified all forms of this exposure.

Additionally, reviewing the Audit Journal for your projects on pypi.python.org for suspicous activity is a good idea. If you identify any such activity, please report it per our published security policy.

Timeline

Timeline using the disclosure date **2017-11-05** as reference:

- 2017-11-05 Issue reported by Joachim Jablon to a single member of the security team listed in our security policy on pypi.org
- 2017-11-08 (**+3days**):Issue reported by Joachim Jablon to an additional member of the security team listed in our security policy on pypi.org
- 2017-11-08 (**+3days**):Issue reported by Joachim Jablon to all members of the security team listed in our security policy on pypi.org
- 2017-10-08 (**+3days**): Report investigated by Ernest W. Durbin III and determined to be valid.
- 2017-10-09 (**+4days**): Administrative password resets issued.

1.1.6 PyPI typo squatting

- Typosquatting programming language package managers by Nikolai Tschacher (8 June, 2016)
- LWN: Typosquatting in package repositories (July 20, 2016)
- Building a botnet on PyPi by Steve Stagg (May 19, 2017)
- warehouse bug (pypi.org): Block package names that conflict with core libraries (reported at June 28, 2017)
- 2017-09-09: skcsirt-sa-20170909-pypi-malicious-code advisory

fate0:

- 2017-05-27 04:38 - 2017-05-31 12:24 (5 days): 10,685 downloads
- May-June, 2017
- https://mail.python.org/pipermail/distutils-sig/2017-June/030592.html
- http://blog.fatezero.org/2017/06/01/package-fishing/
- https://github.com/pypa/pypi-legacy/issues/644
- http://evilpackage.fatezero.org/
- https://github.com/fate0/cookiecutter-evilpy-package
- Packages (this list needs to be validated):
  - caffe
  - ffmpeg
  - ftp
  - git
  - hbase
  - memcached
  - mkl
  - mongodb
  - opencv
  - openssl
  - phantomjs
  - proxy
  - pygpu
  - python-dev
  - rabbitmq
  - requirement.txt
  - requirements.txt
  - rrequirements.txt
  - samba
  - shadowsock
  - smb

  - tkinter
  - vtk
  - youtube-dl
  - zookeeper
  - ztz
  - ...

See also:

- pytosquatting.org project

Example of typos:

- `urllib`, `urllib2`: part of the standard library
- `urlib3` instead of `urllib3`

1.1.7 Links

- The Update Framework (TUF): Like the S in HTTPS, a plug-and-play library for securing a software updater.

1.2 Python SSL and TLS security

Evolutions of the `ssl` module.

1.2.1 Cipher suite

Python 2.7 and 3.5-3.7:

```
_DEFAULT_CIPHERS = (
    'ECDH+AESGCM:ECDH+CHACHA20:DH+AESGCM:DH+CHACHA20:ECDH+AES256:DH+AES256:'
    'ECDH+AES128:DH+AES:ECDH+HIGH:DH+HIGH:RSA+AESGCM:RSA+AES:RSA+HIGH:'
    '!aNULL:!eNULL:!MD5:!3DES'
    )
```

Pytohn 3.4:

```
_DEFAULT_CIPHERS = (
    'ECDH+AESGCM:DH+AESGCM:ECDH+AES256:DH+AES256:ECDH+AES128:DH+AES:ECDH+HIGH:'
    'DH+HIGH:ECDH+3DES:DH+3DES:RSA+AESGCM:RSA+AES:RSA+HIGH:RSA+3DES:!aNULL:'
    '!eNULL:!MD5'
)
```

Python 3.3:

```
_DEFAULT_CIPHERS = 'DEFAULT:!aNULL:!eNULL:!LOW:!EXPORT:!SSLv2'
```

1.2.2 Options

- `SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS`: CBC IV attack countermeasure (CVE-2011-3389)
- `SSL_OP_NO_SSLv2`: SSLv2 is unsafe
- `SSL_OP_NO_SSLv3`: SSLv3 is unsafe
- `SSL_OP_NO_COMPRESSION`: CRIME countermeasure
- `SSL_OP_CIPHER_SERVER_PREFERENCE`
- `SSL_OP_SINGLE_DH_USE`
- `SSL_OP_SINGLE_ECDH_USE`

Python 3.7:

```
/* Defaults */
    options = SSL_OP_ALL & ~SSL_OP_DONT_INSERT_EMPTY_FRAGMENTS;
    if (proto_version != PY_SSL_VERSION_SSL2)
        options |= SSL_OP_NO_SSLv2;
    if (proto_version != PY_SSL_VERSION_SSL3)
        options |= SSL_OP_NO_SSLv3;
    /* Minimal security flags for server and client side context.
     * Client sockets ignore server-side parameters. */
#ifdef SSL_OP_NO_COMPRESSION
    options |= SSL_OP_NO_COMPRESSION;
#endif
#ifdef SSL_OP_CIPHER_SERVER_PREFERENCE
    options |= SSL_OP_CIPHER_SERVER_PREFERENCE;
#endif
#ifdef SSL_OP_SINGLE_DH_USE
    options |= SSL_OP_SINGLE_DH_USE;
#endif
#ifdef SSL_OP_SINGLE_ECDH_USE
    options |= SSL_OP_SINGLE_ECDH_USE;
#endif
    SSL_CTX_set_options(self->ctx, options);
```

1.2.3 CA store

`SSLContext.load_default_certs()` new in Python 3.4.

- Windows: `ssl.enum_certificates(store_name)`, new in Python 3.4. Use CertOpenStore() and `CertEnumCertificatesInStore()` functions.
- Linux: xxx
- macOS: xxx

See also

- certifi: "Python package for providing Mozilla's CA Bundle".
- [Python-Dev] SSL certificates recommendations for downstream python packagers

1.2.4 SSLContext

New in Python 3.2.

1.2.5 CRLs

- `SSLContext.verify_flags`: New in Python 3.4
- `SSLContext.load_verify_locations()`: This method can also load certification revocation lists (CRLs) in PEM or DER format. New in Python 3.5.
- `ssl.enum_crls(store_name)`: new in Python 3.4, specific to Windows

1.2.6 Validate TLS certificates

- Python decides for certificate validation (September, 2014)
- CVE-2014-9365
- Python 2.7.9 (2014-12-10)
- Python 3.4.3 (2015-02-23)
- PEP 476: Enabling certificate verification by default for stdlib http clients: Python 3.4.3, 3.5
- PEP 466: Python 2.7.9
- Version matrix?
  - HTTP
  - SMTP
  - FTP
  - IMAP
  - POP3
  - XML-RPC
  - NNTP

1.2.7 TLS versions

- SSLv2 now black listed
- SSLv3 now black listed

1.2.8 OpenSSL versions

Python bundled OpenSSL in Windows and macOS installers.

OpenSSL versions (read from the Windows installer):

- Python 3.6.1: OpenSSL 1.0.2k
- Python 2.7.13, 3.5.3 and 3.6.0: OpenSSL 1.0.2j
- Python 2.7.12, 3.5.2: OpenSSL 1.0.2h
- Python 2.7.11, 3.4.4, 3.5.0, 3.5.1: OpenSSL 1.0.2d
- Python 2.7.10: OpenSSL 1.0.2a
- Python 2.7.9: OpenSSL 1.0.1j

- Python 3.3.5: OpenSSL 1.0.1e

Windows: see PCbuild/get_externals.bat (or PCbuild/readme.txt in older versions).

macOS: see Mac/BuildScript/build-installer.py.

macOS:

```
# Since Apple removed the header files for the deprecated system
# OpenSSL as of the Xcode 7 release (for OS X 10.10+), we do not
# have much choice but to build our own copy here, too.
```

Example of OpenSSL update: Upgrade installers to OpenSSL 1.0.2k (March 2017).

1.2.9 Links

- The future of the Python ssl module (June, 2016)
- cryptography (cryptography.io): Python library which exposes cryptographic recipes and primitives
- pyOpenSSL
- M2Crypto
- *urllib3 <https://urllib3.readthedocs.io/>_*
- LibreSSL
- borringssl
- multissl (by Christian Heimes): Run Python tests against multiple installations of OpenSSL and LibreSSL

1.3 Python Security

1.3.1 Python branches

- (Latest update: 2017-03-28) Python 2.6, 3.0, 3.1, 3.2 don't get security fixes anymore and so should be considered as vulnerable
- Branches getting security fixes: 2.7, 3.3, 3.4 and 3.5
- See Status of Python branches

1.3.2 Dangerous functions and modules

- Python 2 input()
- Python 2 execfile()
- eval()
- subprocess.Popen(shell=True)
- str.format(), Python 3 str.format_map, and Python 2 unicode.format() all allow arbitrary attribute access on formatted values, and hence access to Python's introspection features: Be Careful with Python's New-Style String Format (Armin Ronacher, December 2016)
- The `pickle` module executes arbitrary Python code: never use it with untrusted data.
- archives:

  - tarfile: Never extract archives from untrusted sources without prior inspection. It is possible that files are created outside of path, e.g. members that have absolute filenames starting with "/" or filenames with two dots "..".
  - zipfile: Never extract archives from untrusted sources without prior inspection. It is possible that files are created outside of path, e.g. members that have absolute filenames starting with "/" or filenames with two dots "..". zipfile attempts to prevent that.

1.3.3 Security model

Bytecode

CPython doesn't verify that bytecode is safe. If an attacker is able to execute arbitrary bytecode, we consider that the security of the bytecode is the least important issue: using bytecode, sensitive code can be imported and executed.

For example, the `marshal` doesn't validate inputs.

Sandbox

Don't try to build a sandbox inside CPython. The attack surface is too large. Python has many introspection features, see for example the `inspect` module. Python also many convenient features which executes code on demand. Examples:

- the literal string `'\N{Snowman}'` imports the `unicodedata` module
- the code to log a warning might be abused to execute code

The good design is to put CPython into a sandbox, not the opposite.

Ok, understood, but I want a sandbox in Python. Well...

- Eval really is dangerous (Ned Batchelder, June 2012)
- PyPy sandboxing
- For Linux, search for SECCOMP

1.3.4 RNG

- CSPRNG:
  - `os.urandom()`
  - `random.SystemRandom`
  - secrets module (Python 3.6)
- `os.urandom()` uses:
  - Python 3.6: `CryptGenRandom()`, `getentropy()`, `getrandom(0)` (blocking) or `/dev/urandom`
  - Python 3.5: `CryptGenRandom()`, `getentropy()`, `getrandom(GRND_NONBLOCK)` (non-blocking) or `/dev/urandom`
  - Python 2.7: `CryptGenRandom()`, `getentropy()` or `/dev/urandom`
  - PEP 524: Make os.urandom() blocking on Linux: Python 3.6
- `ssl.RAND_bytes()` fork issue:

  - Python issue: Re-seed OpenSSL's PRNG after fork
  - OpenSSL Random fork-safety

The `random` module must not be used in security sensitive code, except of the `random.SystemRandom` class.

1.3.5 CPython Security Experts

- Alex Gaynor
- Antoine Pitrou
- Christian Heimes
- Donald Stufft

1.3.6 Windows

ASLR and DEP

ASLR and DEP protections enabled since Python 3.4 (and Python 2.7.11 if built using `PCbuild/` directory).

Unsafe Python 2.7 default installation directory

Python 2.7 installer uses C:Python27directory by default. The created directory has the "Modify" access rights given to the "Authenticated Users" group. An attacker can modify the standard library or even modify python.exe. Python 3 installer now installs Python in "C:Program Files" by default to fix this issue. Override the default installation directory, or fix the directory permissions.

DLL injection

On Windows 8.1 and older, the installer is vulnerable to DLL injection: evil DLL written in the same download directory that the downloaded Python installer. See DLL Hijacking Just Won't Die.

DLL injection using PATH

Inject a malicious DLL in a writable directory included in PATH. The "pip" step of the Python installer will run this DLL.

We consider that it is not an issue of Python (Python installer) itself.

Once you have write access to a directory on the system PATH (not the current user PATH) and the ability to write binaries that are not validated by the operating system before loading, there are many more interesting things you can do rather than wait for the Python installer to be run.

1.3.7 Misc

- python3 -E: ignore `PYTHON*` environment variables like `PYTHONPATH`
- python3 -I: isolated mode, also implies `-E` and `-s`
- Python 3.7 adds a `is_safe` attribute to uuid.UUID objects: http://bugs.python.org/issue22807
- XML: defusedxml, XML bomb protection for Python stdlib modules

- Coverity:
  - Coverity Scan: Python
  - devguide info about Coverity
  - analysis of 2012 by Coverity Software resulted in CPython receiving their highest quality rating.
- sys.path:
  - CVE-2008-5983: http://bugs.python.org/issue5753 added `PySys_SetArgvEx()`
  - CVE-2015-5652: Untrusted search path vulnerability in python.exe in Python through 3.5.0 on Windows allows local users to gain privileges via a Trojan horse readline.pyd file in the current working directory. NOTE: the vendor says "It was determined that this is a longtime behavior of Python that cannot really be altered at this point."
  - `python -E`, `python -I`
- Python at HackerOne
- humans.txt of python.org with the list of "people who found security bugs in the website". For the rationale, see humanstxt.org.

1.3.8 Python Security Response Team (PSRT)

- Handle security@python.org incoming emails
- PSRT issues (private)
- LWN: The Python security response team (June, 2016)

1.3.9 Links

- Reporting security issues in Python
- OWASP Python Security Project (pythonsecurity.org)
- bandit: Python AST-based static analyzer from OpenStack Security Group
- Python CVEs (cvedetails.com)
- https://gemnasium.com/
- owasp-pysec: OWASP Python Security Project
- LWN: Python ssl module update by Christian Heimes at the Python Language Summit 2017 (during Pycon US, Portland, OR)

1.4 TODO list

TODO list for this python-security documentation.

- Get Red Hat impact from a Red Hat URL?

1.4.1 cookielib

Add https://hackerone.com/reports/26647 vulnerability.

https://bugs.python.org/issue16611 #16611: BaseCookie now parses 'secure' and 'httponly' flags.

https://bugs.python.org/issue22796 Regression in Python 3.2 cookie parsing

https://bugs.python.org/issue25228 Support for httponly/secure cookies reintroduced lax parsing behavior

https://code.djangoproject.com/ticket/26158 cookie parsing fails with python 3.x if request contains unnamed cookie

YAML template:

```
- name: "Issue #22796"
  summary: >
    hardened HTTP cookie parsing
  links:
    - http://bugs.python.org/issue22796
  disclosure: "2014-11-04 (issue #22796 created)"
  fixed-in:
   - b1e36073cdde71468efa27e88016aa6dd46f3ec7 # 3.x
  description: >
    HTTP cookie parsing is now stricter, in order to protect against potential
    injection attacks.

    Reported by Tim Graham.
```