GitHub - naksyn/Pyramid: a tool to help operate in EDRs' blind spots

By naksyn

Archived: 2026-04-06 00:46:34 UTC

python 3.7+ Naksyn

What is it

Pyramid is composed of:

1. a Python HTTP/S server that can deliver encrypted files (chacha, xor)
2. Python modules that can load in-memory dependencies of offensive tooling such as Bloodhound-py, secretsdump, LaZagne, Pythonnet, DonPAPI, pythonmemorymodule, paramiko, pproxy.
3. fixed Python dependencies (zip files) that can be imported in memory
4. Python cradle that can download, decrypt and execute in memory Pyramid modules

Why is it useful

Pyramid is useful to perform post-exploitation task in an evasive manner, executing offensive tooling from a signed binary (e.g. python.exe) by importing their dependencies in memory. This can be achieved because:

1. the Python Embeddable package provides a signed Python interpreter with good reputation;
2. Python has many legit applications, so there is a lot of different telemetry coming from the python.exe binary since the interpreter natively runs the APIs. This can be abused by operating within the Python.exe process and trying to blend in the huge "telemetry fingerprint" of python.exe binary.
3. There is a lack of auditing for Python code execution - PEP-578 tried to solve that but the stock python.exe binary does not have auditing capabilities enabled by default.

4. Operations can be done natively from within python.exe natively using Python language to perform post exploitation tasks such as dynamically importing Python modules to run offensive tools and executing Beacon Object Files (after some BOF modifications) directly within python.exe.

For more information please check the **DEFCON30 - Adversary village talk "Python vs Modern Defenses" slide deck** and this **post on my blog**.

Disclaimer

This tool was created to demostrate a bypass strategy against EDRs based on some blind-spots assumptions. It is a combination of already existing techniques and tools in a (to the best of my knowledge) novel way that can help evade defenses. The sole intent of the tool is to help the community increasing awareness around this kind of usage and accelerate a resolution. It's not a 0day, it's not a full fledged shiny C2, Pyramid exploits what might be EDRs blind spots and the tool has been made public to shed some light on them. A defense paragraph has been included, hoping that experienced blue-teamers can help contribute and provide better possible resolution on the issue Pyramid aims to highlight. All information is provided for educational purposes only. Follow instructions at your own risk. Neither the author nor his employer are responsible for any direct or consequential damage or loss arising from any person or organization.

Credits

Pyramid's in-memory loading was initially inspired and expanded upon xorrior 's Empyre - Finder Class

Contributors

snovvcrash built the modules mod-DonPAPI.py - mod-LaZagne.py - mod-clr.py

Current features

Pyramid modules capabilities can be executed directly from a Python interpreter and are currently:

1. Downloading, decryption and in-memory loading of Python dependencies.
2. Dynamic loading and execution of BloodHound Python, impacket secretsdump, DonPAPI, LaZagne.
3. In-memory loading of a remotely fetched dll or exe via PythonMemoryModule
4. SOCKS5 proxying through SSH reverse port forward tunnel.
5. In-memory .NET assembly loading via Pythonnet

Pyramid HTTP server main features:

1. on-the-fly encryption (chacha,xor) of files to be delivered
2. auto-generation of Server configs based on pyramid command line
3. decoding and decryption of HTTP parameters (URL)
4. Basic HTTP Authentication

Cradle main features:

1. Downloading, decryption and in-memory execution of Pyramid modules.
2. Python-standard-libraries-only dependancy

Description

Pyramid can be used with a Python Interpreter already existing on a target machine, or unpacking an official embeddable Python package and then running python.exe to execute a Python download cradle. This is a simple way to avoid creating uncommon Process tree pattern and looking like a normal Python application usage.

In Pyramid the download cradle is used to reach a Pyramid Server via HTTP/S to fetch modules and dependencies.

Modules are specific for the feature you want to use and contain:

1. Custom Finder class to in-memory import required dependencies (zip files).
2. Code to download the required dependencies.
3. Main logic for the program you want to execute (bloodhound, secretsdump, paramiko etc.).

The Python dependencies have been already fixed and modified to be imported in memory without conflicting.

There are currently 8 Pyramid modules available:

1. **bh.py** will in-memory import and execute python-BloodHound.
2. **secretsdump.py** will in-memory import and execute Impacket secretsdump.
3. **shellcode.py** is a simple in-memory shellcode injector.
4. **DonPAPI.py** script will in-memory import and execute DonPAPI. Results and credentials extracted are saved on disk in the Python Embeddable Package Directory.
5. **LaZagne.py** script will in-memory import and execute LaZagne
6. **tunnel-socks5** script import and executes paramiko on a new Thread to create an SSH remote port forward to an SSH server, then a socks5 proxy server is executed locally on target and made accessible remotely through the SSH tunnel.
7. **clr** script imports Pythonnet to load and execute a .NET assembly in-memory.
8. **pythonmemorymodule** script import PythonMemoryModule to load a dll from memory.

Usage

Starting the server

```
git clone https://github.com/naksyn/Pyramid
```

Generate SSL certificates for HTTP Server:

```
openssl req -x509 -newkey rsa:2048 -keyout key.pem -out cert.pem -days 365
```

If you want to use your own signed SSL certificate be sure to:

1. use pem format

2. rename the files with key.pem and cert.pem
3. place both files into the Server folder.

Configuring the module

As an example, if you want to use pythonmemorymodule with Pyramid, put your payload in the **Delivery_files** folder, then open pythonmemorymodule.py and configure the needed parameters in the top of the script, such as the name of the payload file and the procedure you want to call after the PE has been loaded.

Unzip embeddable package and execute the download cradle on target

Once the Pyramid server is running and the Base script is ready you can set the variable `pyramid_module` in **Agent/cradle.py** file and execute it on the target. The cradle is built to be run with python standard libraries.

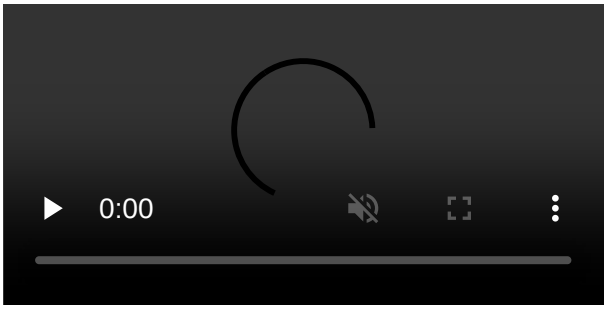
Example

Example of running Pyramid HTTP Server using SSL certificate providing Basic Authentication, encrypting delivery files using ChaCha and auto-generating server configuration in modules and printing a pastable cradle for pythonmemorymodule:

```
python3 pyramid.py -p 443 -ssl -u testuser -pass Sup3rP4ss! -enc "chacha20" -passenc "TestPass1" -se
```

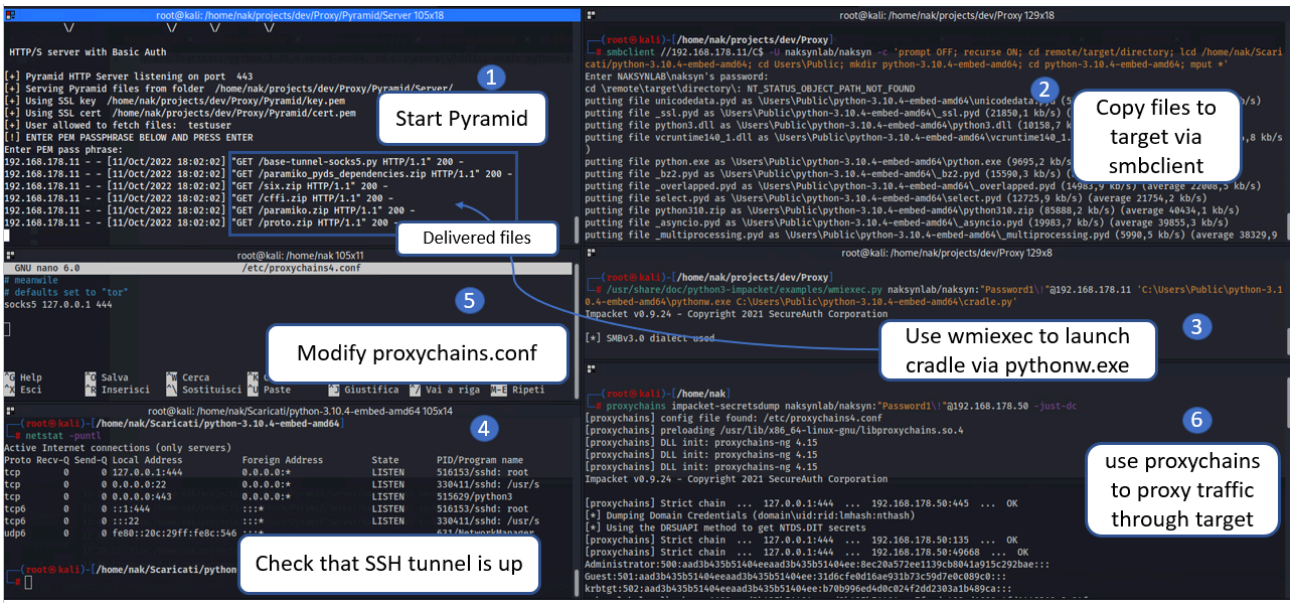
Upon startup pyramid.py will parse its own folder structure to look for key.pem, cert.pem and will deliver files from Server folder.

▼ PyMM.mp4



Tip for executing Pyramid without visible prompt

To execute Pyramid without bringing up a visible python.exe prompt you can leverage pythonw.exe that won't open a console window upon execution and is contained in the very same Windows Embeddable Package. The following picture illustrate an example usage of pythonw.exe to execute base-tunnel-socks5.py on a remote machine without opening a python.exe console window.



Limitations

Dynamically loading Python modules does not natively support importing *.pyd files that are essentially dlls. The only public solution to my knowledge that solves this problem is provided by Scythe *(in-memory-execution) by re-engineering the CPython interpreter. In ordrer not to lose the digital signature, one solution that would allow using the native Python embeddable package involves dropping on disk the required pyd files or wheels. This should not have significant OPSEC implications in most cases, however bear in mind that the following wheels containing pyd files are dropped on disk to allow Dinamic loading to complete: *. Cryptodome - needed by Bloodhound-Python, Impacket, DonPAPI and LaZagne *. bcrypt, cryptography, nacl, cffi - needed by paramiko

How to defend from this technique

Python.exe is a signed binary with good reputation and does not provide visibility on Python dynamic code. Pyramid exploits these evasion properties carrying out offensive tasks from within the same python.exe process.

For this reason, one of the most efficient solution would be to block by default binaries and dlls signed by Python Foundation, creating exceptions only for users that actually need to use python binaries.

Alerts on downloads of embeddable packages can also be raised.

Deploying PEP-578 is also feasible although complex, [this is a sample implementation](). However, deploying PEP-578 without blocking the usage of stock python binaries could make this countermeasure useless.
