An In-memory Embedding of CPython for Offensive Use

Ateeq Sharfuddin, Brian Chapman, Chris Balles
Overview

1. Why?
2. Our contributions
3. Results
Why?

- Assist security researchers and enterprise Red Teams
- Many security research scripts are available in Python
Our contributions

- CPython core shared library
- Frozen custom zip module loader
- Stock Python packages
- Special cases
CPython core shared library

1. Dynamic-Loading from Memory*
   ○ LoadLibraryFromMemory

2. Isolated configuration
   ○ PyConfig_InitIsolatedConfig/Py_InitializeFromConfig

3. Python packages during initialization
   ○ encodings, codecs, abc, etc.
1. A derivation from zipimport
2. Load any number of packages residing in a single zip file in memory
   ○ cba_zipimport.install_cba_metafinder(package_name, package_zip_bytes)
3. Frozen
   ○ Add reference in PyImport_FrozenModules table.
   ○ loaded by FrozenImporter
4. Call _CBAZipImport_Init in pylifecycle.c
5. Installs at offset 2 in sys.meta_path
   ○ after BuiltinImporter and FrozenImporter
6. Can load Python C Extensions
Bundling Python Packages

Stock installation of CPython contains a prepackaged collection of modules

a. Offer this same collection of modules
b. Create a ZIP archive of these .py and .pyc files as cba_python38_lib.zip
c. Use our xxd.py to generate a C array of this ZIP file (_CBA_python38_lib)
d. During _CBAZipImport_Init perform:
   i. cba_zipimport.install_cba_metafinder("#cba_python38_lib.zip", _CBA_python38_lib)
Python C Extensions

Python C Extensions that come bundled with CPython (e.g., win64)

a. Recompiled such that non-system shared libraries are statically-linked
b. Create a ZIP archive of these .pyd files as cba_python38_win64.zip
c. Use our xxd.py to generate a C array _CBA_python38_pyd_win64
d. During _CBA_ZipImport_Init perform:
   ■ cba_zipimport.install_cba_metafinder("#cba_python38_pyd.zip", _CBA_python38_pyd_win64)
e. _zip_searchorder in cba_zipimport updates process native C Extensions
f. create_dynamic_inmemory function added to builtin importer to handle loading native C Extensions from memory
Special Cases

1. ctypes package
   a. Store GetModuleHandle() for this library in sys.dllhandle for modules that call core shared library C functions via ctypes.pythonapi

2. Threading
   a. Don't forget to first call PyGILState_Ensure to acquire global interpreter lock (GIL) before running Python code, then release with PyGILState_Release

3. GetModuleHandle/GetModuleHandleEx in C Extensions will not give you what you want (use sys.dllhandle instead)

4. Expects DLL version of C Runtime to exist on device (same requirement as stock CPython)
Results

1. Demonstrations (artifacts available in Appendix)
2. Variations of this are in use in production by customer Red Teams for a year
3. Source code for Python 3.8.2 is available
4. Artifacts available (password in the paper):
   - https://github.com/farfella/woot2021
   - https://doi.org/10.5281/zenodo.4638251

Thank you! Questions?

ateeq@scythe.io / https://ateeq.dev