Python Obfuscation and Evasion Techniques

Nick Beede

Approved for Public Release; Distribution Unlimited. Case Number 18-2156
©2018 The MITRE Corporation. ALL RIGHTS RESERVED

MITRE

© 2018 The MITRE Corporation. All rights reserved.

Whoami

- Cyber Security Engineer at MITRE
- Malware/Forensic analyst
- Security researcher
- Python is life
- Also mountains

Origins

- University of Nebraska at Omaha
- CTFs
- NULLify co-founder
- SFS Scholarship

Objective

- **Cover the current state of Python malware observed in the wild and provide an overview of available obfuscation tools and techniques**

Outline

- Background
- Python malware in the news
- Visual obfuscation
- Python internals
- Packaging tools
- Advanced techniques

Background Research

- **Objectives**
  - Survey publicly available Python hardening techniques
  - Categorized techniques into families and their effectiveness
  - Techniques researched were limited to FOSS
- **Summary of findings from prior work**
  - RE complexity dramatically increases as the number of unique families of techniques in use increases
  - Identifying which techniques have been applied is more difficult after several iterations of obfuscation
  - Compiler modifications are incredibly effective

Python Malware in the Wild

Security for... Products Services Resources Partners Company

Blog Home > Malware > Unit 42 Technical Analysis: Seaduke

Unit 42 Technical Analysis: Seaduke

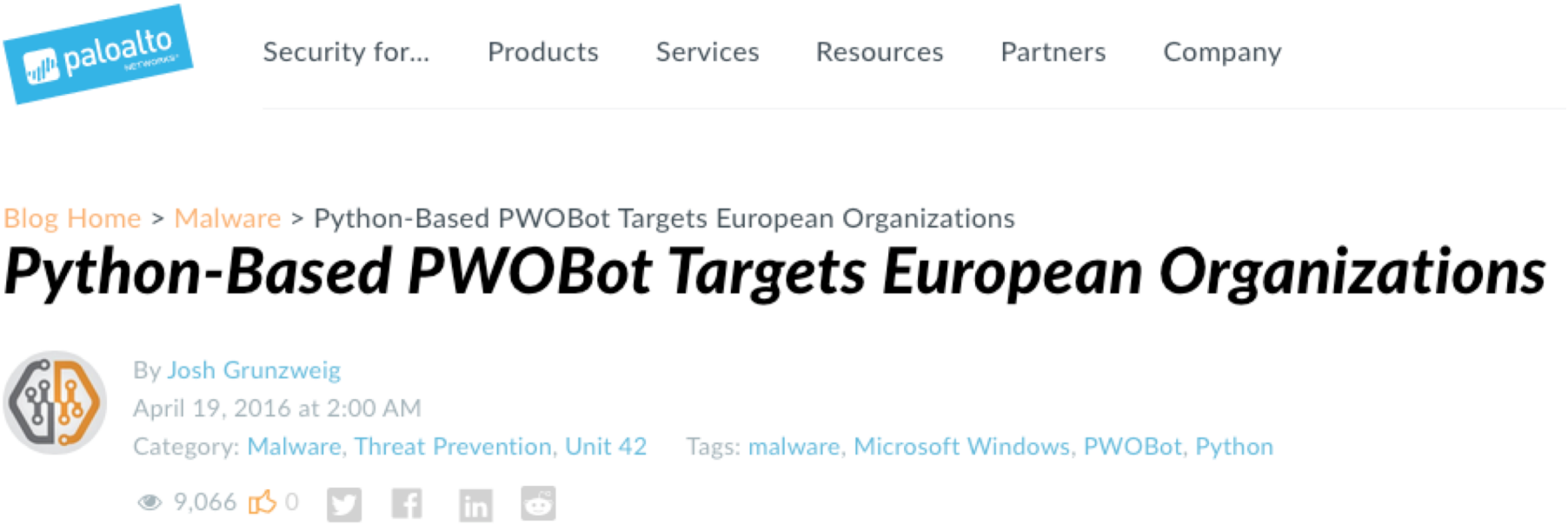
By Josh Grunzweig
July 14, 2015 at 9:47 AM
Category: Malware, Threat Prevention, Unit 42 Tags: Duke malware, forkmeiamfamous, malware, Seaduke, Symantec, Trojan

6,856 1

https://researchcenter.paloaltonetworks.com/2015/07/unit-42-technical-analysis-seaduke/

MITRE

Python Malware in the Wild



https://researchcenter.paloaltonetworks.com/2016/04/unit42-python-based-pwobot-targets-european-organizations/

MITRE

Python Malware in the Wild

https://threatvector.cylance.com/en_us/home/el-machete-malware-attacks-cut-through-latam.html

Python Malware in the Wild

https://blog.talosintelligence.com/2018/02/cannibalrat-targets-brazil.html

MITRE

Python Malware in the Wild

MALWARE | THREAT ANALYSIS

PBot: a Python-based adware

Posted: April 18, 2018 by hasherezade
Last updated: April 19, 2018

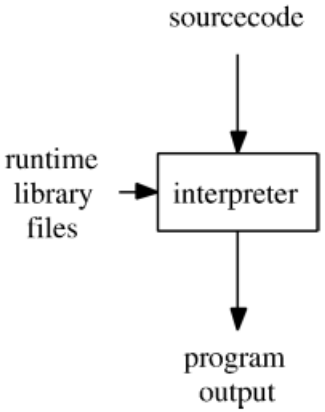
Recently, we came across a Python-based sample dropped by an exploit kit. Although it arrives under the disguise of a MinerBlocker, it has nothing in common with miners. In fact, it seems to be PBot/PythonBot: a Python-based adware.

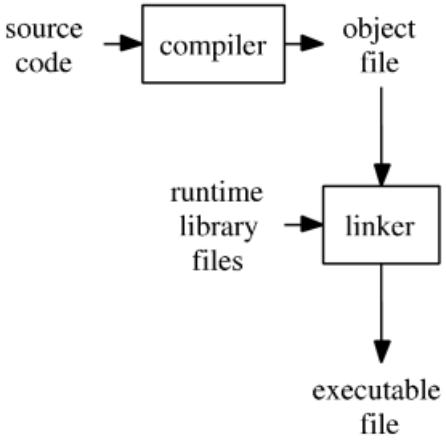
https://blog.malwarebytes.com/threat-analysis/2018/04/pbot-python-based-adware/

MITRE

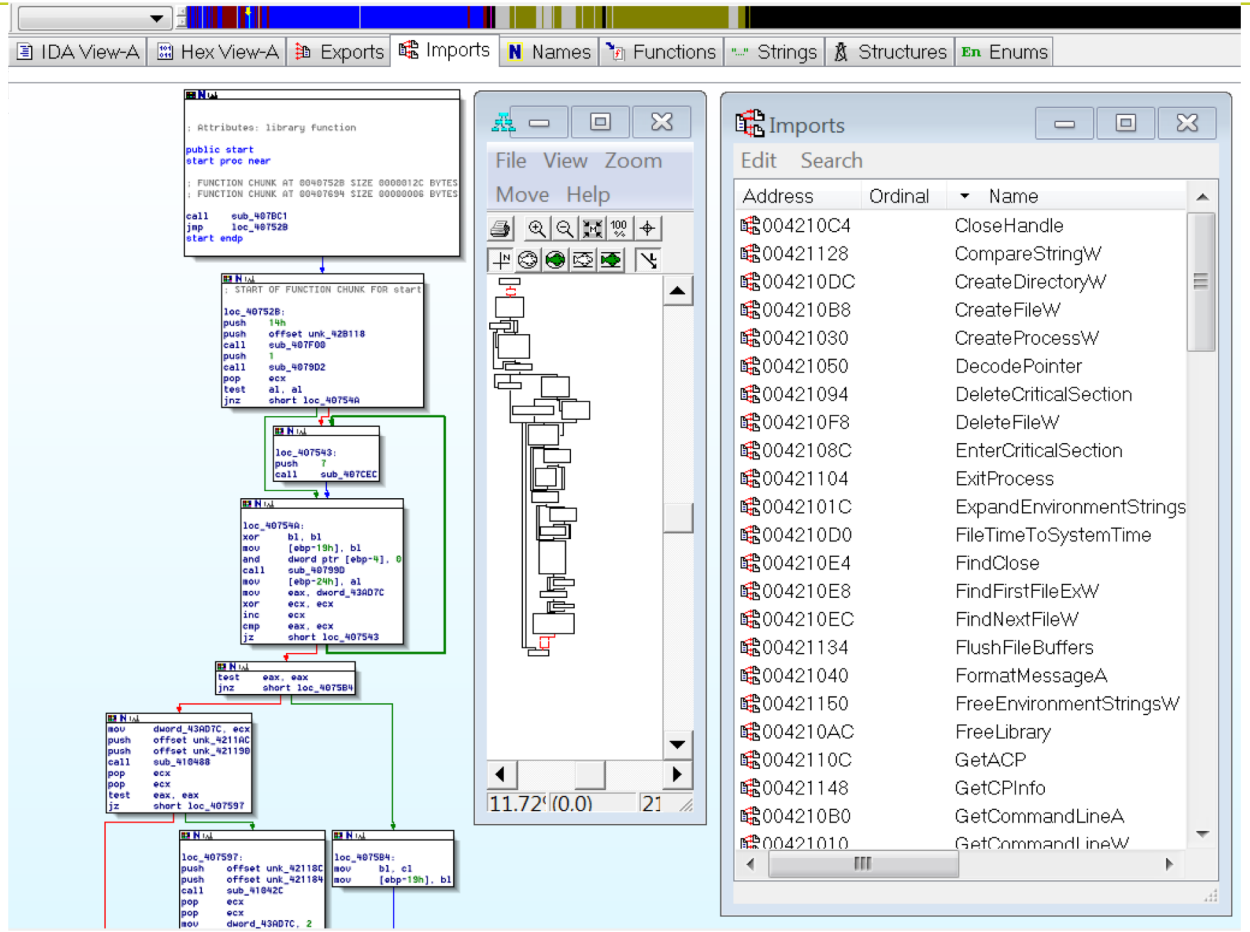
Why Python?

- **Developers trending to high level languages**
  - Python/Ruby/etc.
  - Python is easier to write than C
- **Introduces new reverse engineering challenges**
  - Interpreted bytecode VS compiled executable





Reversing Python like it's C



MITRE

Visual Obfuscation

- **Low difficulty when applied with no additional obfuscation techniques**

- **Opy –** https://github.com/QQuick/Opy
- **Oneliner-izer –** https://github.com/csvoss/onelinerizer
- **pyminifier –** https://github.com/liftoff/pyminifier
- **pyobfuscate -** https://github.com/astrand/pyobfuscate

Opy

```
# coding: UTF-8
import sys
l111_opy_ = sys.version_info [0] == 2
ll_opy_ = 2048
lll_opy_ = 7
def l111_opy_ (l1111_opy_):
    global l111_opy_
    l11111_opy_ = ord (l1111_opy_ [-1])
    ll_opy_ = l1111_opy_ [:-1]
    l11111_opy_ = l11111_opy_ % len (ll_opy_)
    l11111_opy_ = ll_opy_ [:l11111_opy_] + ll_opy_ [l11111_opy_:]
    if l111_opy_:
        lll_opy_ = l111_opy_ () .join ([l1111_opy_ (ord (char) - ll_opy_ - (l11111_opy_ + l11111_opy_) % lll_opy_) for l11111_opy_, char in enumerate
(l1111_opy_)])
    else:
        lll_opy_ = str () .join ([chr (ord (char) - ll_opy_ - (l11111_opy_ + l11111_opy_) % lll_opy_) for l11111_opy_, char in enumerate (l1111_opy_)])
    return eval (lll_opy_)
def main():
  print(l111_opy_ (u"□□□□□□□□□□□□□□□□□□"))
if __name__ == l111_opy_ (u"□□□□□□□□□□□□□□"):
  main()
```

MITRE

Oneliner-izer

- "*No newlines allowed. No semicolons, either. No silly file I/O tricks. No silly eval or exec tricks. Just good, old-fashioned λ.*"

- **No additional runtime overhead…**

```
(lambda __g, __print: [(lambda __after: (main(), __after())[1] if (__name__ ==
'__main__') else __after())(lambda: None) for __g['main'], main.__name__ in
[(lambda : (__print('Hello World'), None)[1], 'main')]][0])(globals(),
__import__('__builtin__').__dict__['print'])
```

Python Filetypes

- **.py**
  - Human readable source code
- **.pyc**
  - Bytecode representation
  - Created on import or compile()
- **.pyo**
  - Similar to .pyc but removes assert statements and docstrings
  - Compact file size
- **.pyd**
  - Essentially a Windows DLL

.pyc

- 4 byte magic number
- 4 byte modify timestamp
- Marshalled code object

```
  0 03F30D0A D1FE225B 63000000 00000000  Û  –¸"[c
 16 00010000 00400000 00730900 00006400      @    s     d
 32 00474864 01005328 02000000 730C0000  GHd   S(     s
 48 0068656C 6C6F2077 6F726C64 214E2800  hello world!N(
 64 00000028 00000000 28000000 00280000     (     (     (
 80 00007309 00000073 696D706C 652E7079   s     simple.py
 96 74080000 003C6D6F 64756C65 3E010000 t     <module>
112 00730000 0000                        s
```

```
magic 03f30d0a
moddate d1fe225b (Thu Jun 14 19:48:33 2018)
code
   argcount 0
   nlocals 0
   stacksize 1
   flags 0040
   code 640000474864010053
  1           0 LOAD_CONST               0 ('hello world!')
              3 PRINT_ITEM
              4 PRINT_NEWLINE
              5 LOAD_CONST               1 (None)
              8 RETURN_VALUE
   consts
      'hello world!'
      None
   names ()
   varnames ()
   freevars ()
   cellvars ()
   filename 'simple.py'
   name '<module>'
   firstlineno 1
   lnotab
```

MITRE

Python Bytecode

```
magic 03f30d0a
moddate d1fe225b (Thu Jun 14 19:48:33 2018)
code
   argcount 0
   nlocals 0
   stacksize 1
   flags 0040
   code 640000474864010053
  1           0 LOAD_CONST               0 ('hello world!')
              3 PRINT_ITEM
              4 PRINT_NEWLINE
              5 LOAD_CONST               1 (None)
              8 RETURN_VALUE
   consts
      'hello world!'
      None
   names ()
   varnames ()
   freevars ()
   cellvars ()
   filename 'simple.py'
   name '<module>'
   firstlineno 1
   lnotab
```

```
# Instruction opcodes for compiled coo
# Blank lines correspond to available

def_op('POP_TOP', 1)
def_op('ROT_TWO', 2)
def_op('ROT_THREE', 3)
def_op('DUP_TOP', 4)
def_op('DUP_TOP_TWO', 5)

def_op('NOP', 9)
def_op('UNARY_POSITIVE', 10)
def_op('UNARY_NEGATIVE', 11)
def_op('UNARY_NOT', 12)

def_op('UNARY_INVERT', 15)

def_op('BINARY_MATRIX_MULTIPLY', 16)
def_op('INPLACE_MATRIX_MULTIPLY', 17)

def_op('BINARY_POWER', 19)
def_op('BINARY_MULTIPLY', 20)
```

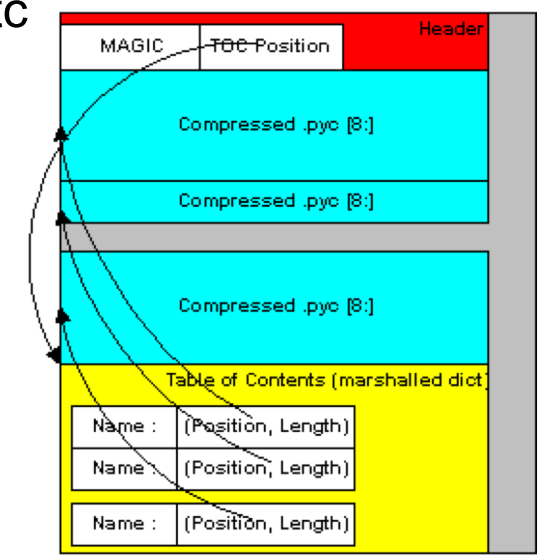
Python Packaging

- Package Python into a native binary format (PE/ELF/Mach-O)
- Distribute and run applications without needing Python installed
- Bundles Python runtime and dependencies with application bytecode

Solution	Windows	Linux	OS X	Python 3	License	One-file mode	Zipfile import	Eggs	pkg_resources support
bbFreeze	yes	yes	yes	no	MIT	no	yes	yes	yes
py2exe	yes	no	no	yes	MIT	yes	yes	no	no
pyInstaller	yes	yes	yes	yes	GPL	yes	no	yes	no
cx_Freeze	yes	yes	yes	yes	PSF	no	yes	yes	no
py2app	no	no	yes	yes	MIT	no	yes	yes	yes

PyInstaller

- **Single folder/file bundling**
- **Binary bootloader**
  - Extracts bundled files to temppath/_MEIxxxxx
  - Load the Python dynamic lib embedded in the executable
  - Initialize interpreter – set sys.path, sys.prefix, etc
  - Run python code
- **Archives**
  - Zlib
    - Used for Python modules
  - CArchive
    - Contains any other files



MITRE

PyInstaller Protections

- **Encrypting Python Bytecode**
  - Runtime arg --key AES encrypts all embedded files within ZLibArchive files
- **UPX**
  - Bootloader is executed after program decompression

Basic PyInstaller RE

- **Pyinstxtractor.py - extract contents of PyInstaller generated Windows executables**
  - .pyc source files now available minus the header information
  - Prepend 03F30D0A00000000 and rename to <source_file>.pyc
- **Uncompyle6 - cross-version python bytecode decompiler**
  - Run against <source_file>.pyc to get original .py source code

Encrypted PyInstaller RE

- **Pyinstxtractor will still extract content without decrypting original files**
  - Pyimod00_crypto_key is a .pyc file that contains the encryption key
  - Prepend magic+timestamp and uncompyle6 the .pyc

```
$ uncompyle6 pyimod00_crypto_key.pyc
# uncompyle6 version 3.2.0
# Python bytecode 3.6 (3379)
# Decompiled from: Python 2.7.12 (default, Jun 29 2016, 14:05:02)
# [GCC 4.2.1 Compatible Apple LLVM 7.3.0 (clang-703.0.31)]
# Embedded file name: C:\Users\jimmy\Desktop\python_test_bins\build\hello_world\pyimod00_crypto_key.py
# Compiled at: 2018-06-17 14:31:40
# Size of source mod 2**32: 26 bytes
key = '000000s3cr3t_K3y'
# okay decompiling pyimod00_crypto_key.pyc
```

```python
from Crypto.Cipher import AES
import zlib

CRYPT_BLOCK_SIZE = 16

# key obtained from pyimod00_crypto_key
key = 's3cr3t_K3y'

inf = open('hello_world', 'rb') # encrypted file input
outf = open('hello_world.decrypted', 'wb') # output file

# Initialization vector
iv = inf.read(CRYPT_BLOCK_SIZE)

cipher = AES.new(key, AES.MODE_CFB, iv)

# Decrypt and decompress
plaintext = zlib.decompress(cipher.decrypt(inf.read()))

# Write pyc header
outf.write('\x03\xf3\x0d\x0a\0\0\0\0')

# Write decrypted data
outf.write(plaintext)

inf.close()
outf.close()
```

Py2exe

- **Python bytecode stored in PYTHONSCRIPT resource**
- **Unpy2exe**
  - Extract PYTHONSCRIPT .rsrc
  - Extract and dump each code_object as a .pyc file

Advanced Obfuscation Techniques

- **Compiler modification**
  - Custom opcodes
  - Code object modification
  - Standard package removal
  - Instruction hooks

Custom Opcodes

- Python bytecode is an integer representation of an operation
- Opcodes defined in CPython opcode.h header file
- Reserved space for 255 opcodes but only about 130 are defined

```
/* Instruction opcodes for compiled code */
#define POP_TOP                   1
#define ROT_TWO                   2
#define ROT_THREE                 3
#define DUP_TOP                   4
#define DUP_TOP_TWO               5
#define ROT_FOUR                  6
#define NOP                       9
#define UNARY_POSITIVE           10
#define UNARY_NEGATIVE           11
#define UNARY_NOT                12
#define UNARY_INVERT             15
#define BINARY_MATRIX_MULTIPLY   16
#define INPLACE_MATRIX_MULTIPLY  17
#define BINARY_POWER             19
#define BINARY_MULTIPLY          20
#define BINARY_MODULO            22
#define BINARY_ADD               23
#define BINARY_SUBTRACT          24
#define BINARY_SUBSCR            25
```

Custom Opcodes

```
    /* Instruction opcodes for compiled code */
#define POP_TOP                   1
#define ROT_TWO                   2
#define ROT_THREE                 3
#define DUP_TOP                   4
#define DUP_TOP_TWO               5
#define ROT_FOUR                  6
#define NOP                       9
#define UNARY_POSITIVE           10
#define UNARY_NEGATIVE           11
#define UNARY_NOT                12
#define UNARY_INVERT             15
#define BINARY_MATRIX_MULTIPLY   16
#define INPLACE_MATRIX_MULTIPLY  17
#define BINARY_POWER             19
#define BINARY_MULTIPLY          20
#define BINARY_MODULO            22
#define BINARY_ADD               23
#define BINARY_SUBTRACT          24
#define BINARY_SUBSCR            25
```

```
    /* Instruction opcodes for compiled code */
#define POP_TOP                   5
#define ROT_TWO                   4
#define ROT_THREE                 3
#define DUP_TOP                   2
#define DUP_TOP_TWO               1
#define NOP                      10
#define UNARY_POSITIVE           9
#define UNARY_NEGATIVE           11
#define UNARY_NOT                12
#define UNARY_INVERT             15
#define BINARY_MATRIX_MULTIPLY   16
#define INPLACE_MATRIX_MULTIPLY  17
#define BINARY_POWER             19
#define BINARY_MULTIPLY          20
#define BINARY_MODULO            24
#define BINARY_ADD               23
#define BINARY_SUBTRACT          22
#define BINARY_SUBSCR            25
```

Custom Opcodes

- **Defeats modern Python decompilers**

```
$ uncompyle6 hello_world.pyc
Traceback (most recent call last):
  File "build/bdist.linux-x86_64/egg/xdis/load.py", line 185, in load_module_from_file_object
    co = marshal.loads(bytecode)
ValueError: bad marshal data (unknown type code)
Traceback (most recent call last):
  File "/usr/local/bin/uncompyle6", line 9, in <module>
    load_entry_point('uncompyle6==3.2.0', 'console_scripts', 'uncompyle6')()
  File "build/bdist.macosx-10.11-x86_64/egg/uncompyle6/bin/uncompile.py", line 181, in main_bin
  File "build/bdist.macosx-10.11-x86_64/egg/uncompyle6/main.py", line 223, in main
  File "build/bdist.macosx-10.11-x86_64/egg/uncompyle6/main.py", line 126, in decompile_file
  File "build/bdist.linux-x86_64/egg/xdis/load.py", line 108, in load_module
  File "build/bdist.linux-x86_64/egg/xdis/load.py", line 198, in load_module_from_file_object
ImportError: Ill-formed bytecode file hello_world.pyc
<type 'exceptions.ValueError'>; bad marshal data (unknown type code)
```

PyREtic

- **Extensible framework for in-memory Python bytecode reverse engineering**
- **Compare bytecode from obfuscated .pyc files against unaltered stdlib .pyc files**
  - Produce a new opcode.py to use for decompiling
- **Relies on Unpyc so no Python3 support**
- **Assumes one-to-one opcode mapping**
- **Needs updates to their decompiler**
  - Xdis to the rescue

Instruction Hooks

- **CPython defines how serialized code objects are generated and complexity can be added here**
  - AES encryption
  - Custom encodings

```
#define LOAD_FAST_ZERO_LOAD_CONST         148
```

```
case LOAD_FAST_ZERO_LOAD_CONST:
  x = GETLOCAL(0);
  if (x != NULL) {
    Py_INCREF(x);
    PUSH(x);
    x = GETITEM(consts, oparg);
    Py_INCREF(x);
    PUSH(x);
    goto fast_next_opcode;
}
format_exc_check_arg(PyExc_UnboundLocalError,
                     UNBOUNDLOCAL_ERROR_MSG,
                     PyTuple_GetItem(co->co_varnames, oparg));
break;
```

Standard Package Removal

- **Removing modules bundled with the Python standard library can prevent RE using built-in tools**

- **Dis**
  - Supports analysis of CPython bytecode through disassembly
  - Relies on opcode.py for instruction mappings
- **Pdb**
  - Interactive debugger used for monitoring a running Python program
  - Relies on .py source to be present

Advanced Obfuscation in the Wild

MITRE

Applying Obfuscation

- **Visual obfuscation is easy**
  - More difficult with module imports
- **Compiler modifications are hard**
  - Setting up a custom Windows Python build environment takes time
  - Compiling CPython after making changes is a crapshoot
  - Opcode definitions change between Python minor versions
  - Xdis module would help track changes

PEP 551

- **Proposed two new APIs that will let security tools detect when Python is executing potentially dangerous operations**
  - Audit Hook
  - Verified Open Hook

- **Ship your own runtime in an executable ¯_(ツ)_/¯**

Conclusion

- Most Python malware today use very little to no obfuscation
- Compiler obfuscation is simple and effective given lack of RE tools
- Interpreted language malware is growing in popularity
- This is only going to get worse

Questions?

nbeede@mitre.org

MITRE

MITRE

MITRE is a not-for-profit organization whose sole focus is to operate federally funded research and development centers, or FFRDCs. Independent and objective, we take on some of our nation's—and the world's—most critical challenges and provide innovative, practical solutions.

Learn and share more about MITRE, FFRDCs, and our unique value at www.mitre.org