When Threat Actors Fly Under the Radar: Vatet, PyXie and Defray777

By Ryan Tracey, Drew Schmitt

Published: 2020-11-07 · Archived: 2026-04-05 15:17:22 UTC

Next Up: “PyXie Lite”

An earlier version of PyXie was previously covered in-depth by BlackBerry Cylance in December 2019. We will be primarily covering an updated variant and some notable changes we have observed.

Some of these changes include:

- Hardened interpreter.
- New remapped opcode table.
- Repurposed as a data theft and reconnaissance tool.
- Exfiltration through internal servers.

We call this variant PyXie Lite because of the significantly smaller code base, but don’t let the name fool you. It still packs a punch.

Loader

The recent variant we analyzed was loaded by Vatet rather than the Goopdate.dll and LMIGuardianDll.dll side loaders seen with earlier versions of PyXie.

The decrypted Vatet payload contains the first stage of PyXie prepended by a shellcode loader responsible for mapping the first stage into memory and executing it.

The shellcode loader utilizes MurmurHash3 hashing to locate APIs needed during this process at runtime.

DLL	Function	API Hash
Kernel32.dll	GetProcAddress	0x261C88ED
Kernel32.dll	VirtualAlloc	0xC17E7EB2
Kernel32.dll	LoadLibraryExA	0x4B9B30B9

Table 2. MurmurHash3 API hashes.

Stage 1

The purpose of the first stage is to decrypt the second stage payload and execute it in memory.

Mutex

A mutex is created to prevent multiple instances from running at the same time. The following logic is used:

- Retrieve computer name with a call to GetComputerNameA. If that fails, fall back to DEFAULTCOMPNAME.
- Compute MD5 Hash of the computer name.
- XOR computed hash with 0x2.
- Convert the result to a string with StringFromGUID2.
- Create mutex using the string with a call to CreateMutexW.

String encryption

Significant strings are encrypted by a routine that increments each byte of the ciphertext by its index, masks the result with 0x7F (highest value in the ASCII character set) and XORs it against a key of equal length.

Cipher	Key	Operation	Plaintext
0x3a	0x51	(0x3a + 0) ^ 0x51	k
0x08	0x6C	(0x08 + 1) ^ 0x6C	e
0x66	0x1A	(0x66 + 2) ^ 0x1A	r
0x2D	0x5E	(0x2D + 3) ^ 0x5E	n
0x80	0xE1	(0x80 + 4) ^ 0xE1	e
0x10	0x79	(0x10 + 5) ^ 0x79	l
0x33	0x0A	(0x33 + 6) ^ 0x0A	3
0x69	0x42	(0x69 + 7) ^ 0x42	2
0x11	0x37	(0x11 + 8) ^ 0x37	.
0x38	0x25	(0x38 + 9) ^ 0x25	d
0x2d	0x5B	(0x2d + 10) ^ 0x5B	l
0x5a	0x09	(0x5a + 11) ^ 0x09	l

Table 3. String decryption example.

Decrypted Strings

uiAccess=true"

-q -s {%S} -p %u

werfault.exe

vsjitdebugger.exe

dvdplay.exe

onedrivesetup.exe

openwith.exe

%windir%\syswow64\

%windir%\system32\

kernel32.dll

KiUserExceptionDispatcher

RtlCreateUser

IsWow64Process

\StringFileInfo\%04x%04x\ProductName

Table 4. Decrypted first stage strings.

Payload Decryption

The next stage payload is stored in an encrypted 7z archive located in the .gfids section of the binary. It is decrypted with the modified RC4 algorithm previously discussed in the BlackBerry Cylance write-up using the hard-coded key: 2C01443389BDFC7330A3386981C43E154AE8B60EC6646D916F93D18137A53544

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
2020-10-07 09:55:07 ....A     10217425      5704704  release.exe
------------------- ----- ------------ ------------  ------------------------
2020-10-07 09:55:07           10217425      5704704  1 files
```

Figure 12. Decrypted 7z archive.

Payload Execution

OpenProcessToken and GetTokenInformation are called to determine if the process is running under the LocalSystem account. This is used to determine how the next stage payload is executed.

If it is determined to be running as LocalSystem, the payload is injected into a newly spawned process chosen from the Windows directory. The command line for this process follows this format and can be used as an indicator:
-q -s {{**GUID**}} -p **NUMBER**

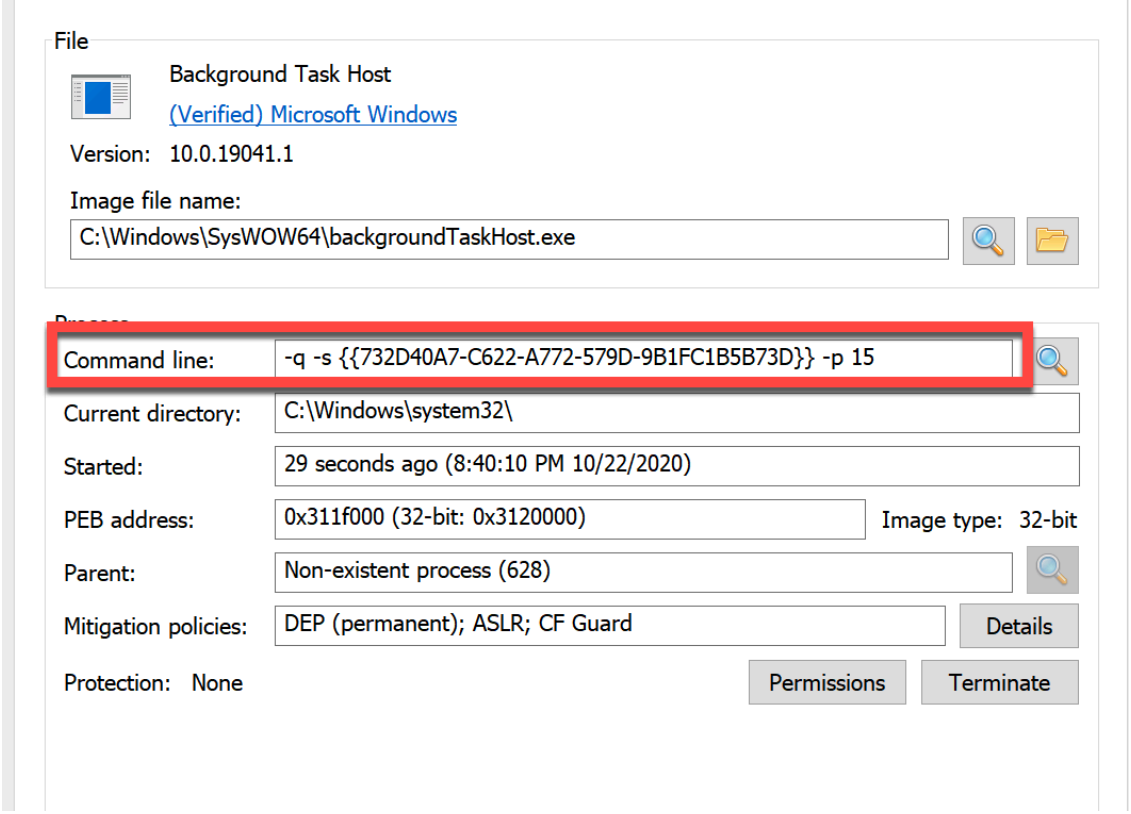

Figure 13. Command line argument.

If not found to be running as LocalSystem, the payload will execute in the memory space of the current process.

Stage 2

The second stage payload is a custom-compiled Python interpreter very similar to ones seen used with earlier variants of PyXie.

Configuration

The configuration is stored in a zlib compressed json blob and is located in the .gfids section of the interpreter. Unlike previous versions of PyXie, it is not encrypted this time around. The variable sys.builtin_json_cfg is created with a call to PySys_SetObject and the compressed configuration blob is stored in it for later use by the final stage Python component.

```
gfids = (void *)get_gfids();
if ( gfids )
{
  gfids_size = get_gfids_size();
  if ( gfids_size )
  {
    if ( PyByteArray_FromStringAndSize(gfids, gfids_size) && PySys_SetObject("builtin_json_cfg") != -1 )
    {
      ++dword_A1A2FC;
      if ( PySys_SetObject("production_flag") != -1 )
        sub_401830();
    }
  }
}
```

Figure 14. sys.builtin_json_cfg variable is created.

Decrypted Strings

The second stage uses the same string encryption that was noted in the previous stage.

kernel32.dll

openwith.exe

onedrivesetup.exe

dvdplay.exe

vsjitdebugger.exe

werfault.exe

-q -s {%S} -p %u

oleout32.dll

VariantClear

Mozilla\Firefox

Mozilla\Firefox\profiles.ini

SOFTWARE\Clients\StartMenuInternet\firefox.exe\shell\open\command

I_CryptUIProtect

cryptui.dll

RtlCreateUserThread

import core.modules.winapi_stubs as winapi_stubs

import core.zip_logs as zip_logs

import os

zip_logs.send_zip_log(winapi_stubs.get_self_executable_path(), os.getpid(), 'CERTS', r'%s')

KiUserExceptionDispatcher

uiAccess="true"

\StringFileInfo\%04x%04x\ProductName

\VarFileInfo\Translation

\\?\globalroot\systemroot\system32\drivers\null.sys

SystemDrive

IsWow64Process

core.entry_point

zipimporter

memzipimport

libs_zip_ctx

start_bind_port

Table 5. Decrypted second stage strings.

Final Stage: Libs.zip

The final stage of PyXie bytecode is contained in an encrypted ZIP file embedded within the interpreter binary. As with the earlier version of PyXie, the memzipimport library is used to import the bytecode from memory.

PyXie Lite

The “core” modules in this variant consist of 41 files versus the 79 seen in the previous version of PyXie analyzed by BlackBerry Cylance. The discrepancy is due to a shift in functionality that we will cover in the next section.

```
   Date      Time    Attr         Size   Compressed  Name
------------------- ----- ------------ ------------  ------------------------
< TRUNCATED FOR BREVITY >
2020-10-07 08:17:01 D....            0            0  core
2020-10-07 08:17:01 D....            0            0  core/conf
2020-10-07 08:16:57 ....A          633          633  core/conf/config.pyx
2020-10-07 08:16:57 ....A          104          104  core/conf/__init__.pyx
2020-10-07 08:16:57 ....A         1161         1161  core/entry_point.pyx
2020-10-07 08:17:01 D....            0            0  core/ipc
2020-10-07 08:16:57 ....A         2071         2071  core/ipc/ipc.pyx
2020-10-07 08:16:57 ....A          103          103  core/ipc/__init__.pyx
2020-10-07 08:16:57 ....A         2079         2079  core/logs.pyx
2020-10-07 08:17:01 D....            0            0  core/mimi
2020-10-07 08:16:58 ....A         1896         1896  core/mimi/mimikatz.pyx
2020-10-07 08:16:58 ....A      1500557      1500557  core/mimi/mimi_x64.pyx
2020-10-07 08:16:58 ....A      1284566      1284566  core/mimi/mimi_x86.pyx
2020-10-07 08:16:57 ....A          104          104  core/mimi/__init__.pyx
2020-10-07 08:17:02 D....            0            0  core/modules
2020-10-07 08:16:58 ....A         1685         1685  core/modules/aes_file.pyx
2020-10-07 08:16:58 ....A         1002         1002  core/modules/bot_lib.pyx
2020-10-07 08:16:58 ....A         1720         1720  core/modules/citrix.pyx
2020-10-07 08:16:58 ....A         9256         9256  core/modules/cookies.pyx
2020-10-07 08:16:58 ....A         1147         1147  core/modules/crc64.pyx
2020-10-07 08:16:58 ....A          965          965  core/modules/decorators.pyx
2020-10-07 08:16:58 ....A         5366         5366  core/modules/find_files.pyx
2020-10-07 08:16:58 ....A         1713         1713  core/modules/keepass.pyx
2020-10-07 08:16:58 ....A        13922        13922  core/modules/lnk_file.pyx
2020-10-07 08:16:58 ....A        11138        11138  core/modules/logmein.pyx
2020-10-07 08:16:58 ....A         2829         2829  core/modules/multipart.pyx
2020-10-07 08:16:58 ....A         1481         1481  core/modules/os_ver.pyx
2020-10-07 08:16:58 ....A         1205         1205  core/modules/rdp.pyx
2020-10-07 08:16:58 ....A         2799         2799  core/modules/rdp_creds.pyx
2020-10-07 08:16:58 ....A         1383         1383  core/modules/recent_files.pyx
2020-10-07 08:16:58 ....A         4234         4234  core/modules/research_domain.pyx
2020-10-07 08:16:58 ....A         1691         1691  core/modules/sharphound.pyx
2020-10-07 08:16:58 ....A        14309        14309  core/modules/smb_scan.pyx
2020-10-07 08:16:58 ....A         4646         4646  core/modules/socks5.pyx
2020-10-07 08:16:58 ....A         1290         1290  core/modules/softening.pyx
2020-10-07 08:16:58 ....A         8265         8265  core/modules/sysinfo.pyx
2020-10-07 08:16:58 ....A         8921         8921  core/modules/tools.pyx
2020-10-07 08:16:58 ....A         4769         4769  core/modules/webdav.pyx
2020-10-07 08:16:58 ....A         9940         9940  core/modules/winapi_stubs.pyx
2020-10-07 08:16:58 ....A        18510        18510  core/modules/windnsquery.pyx
2020-10-07 08:16:58 ....A         7632         7632  core/modules/winfiletime.pyx
2020-10-07 08:16:58 ....A          107          107  core/modules/__init__.pyx
2020-10-07 08:16:58 ....A         1172         1172  core/passwords.pyx
2020-10-07 08:16:58 ....A         9721         9721  core/routines.pyx
2020-10-07 08:16:58 ....A         3468         3468  core/software.pyx
2020-10-07 08:16:57 ....A           99           99  core/__init__.pyx
< TRUNCATED FOR BREVITY >
------------------- ----- ------------ ------------  ------------------------
2020-10-07 08:17:02          14868195     14868195  1312 files, 128 folders
```

Figure 15. Bytecode listing.

Interpreter hardening

Cursory analysis of the bytecode revealed that the headers had been stripped as with previous earlier versions of PyXie. Additionally, we found that the opcode table had once again been modified and the opcodes recovered from previous versions of PyXie could no longer be used to decompile this bytecode.

```
def initialize--- This code section failed: ---

0       LOAD_GLOBAL       'threading'
3       LOAD_ATTR         'Thread'
6       LOAD_CONST        'target'
9       LOAD_GLOBAL       'ipc'
12      LOAD_ATTR         'ipc_server_proc'
15      CALL_FUNCTION_256 None
18      LOAD_ATTR         'start'
21      CALL_FUNCTION_0   None
24      END_FINALLY       None

25      LOAD_GLOBAL       'time'
28      LOAD_ATTR         'sleep'
31      LOAD_CONST        5
34      CALL_FUNCTION_1   None
37      END_FINALLY       None

38      LOAD_GLOBAL       'config'
41      LOAD_ATTR         'init_config'
44      CALL_FUNCTION_0   None
47      END_FINALLY       None
48      LOAD_CONST        None
51      PRINT_ITEM        None

Syntax error at or near `END_FINALLY' token at offset 24
```

Figure 16. Attempts to decompile bytecode with previously recovered PyXie opcodes resulted in an error.

Knowing this, we attempted to force the interpreter to import DeDrop's all.py with hopes of generating bytecode that could be used for opcode recovery. To our dismay, we found that simply importing a script would no longer cause the interpreter to output bytecode.

A closer look at the interpreter found that the sys.dont_write_bytecode variable had been set to true. This has the effect of preventing bytecode from being written to disk when modules are imported. Likely, this is something the developers intentionally enabled to hinder analysis efforts.

```
00534C60    53          push ebx                    Py_DontWriteBytecodeFlag = True
00534C61    B8 A8CA9400 mov eax,release.94CAA8      eax:"dont_write_bytecode", 94CAA8:"dont_write_bytecode"
00534C66    E8 A5370000 call release.538410         SET_SYS_FROM_STRING
```

Figure 17. sys.dont_write_bytecode is set to True.

Hijacking the interpreter with a search order vulnerability

During our analysis of the interpreter, we found that it attempted to load a number of modules from the current working directory and was vulnerable to search order hijacking.

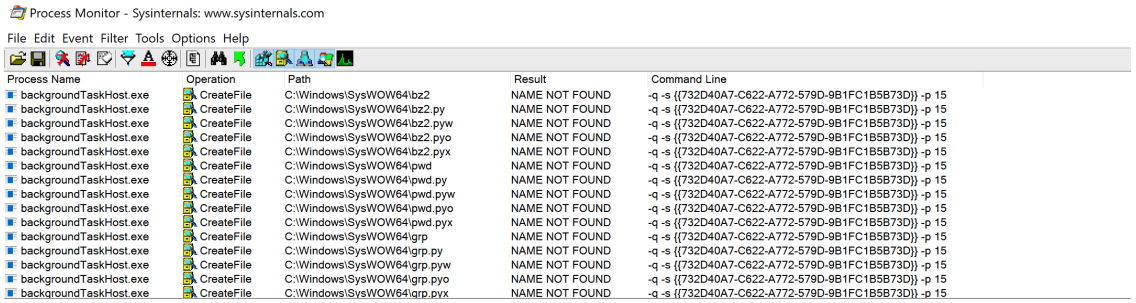

Figure 18. PyXie attempting to load libraries from the current working directory.

We were able use this to our advantage by dropping a simple Python shell into one of the modules it attempted to import. This gave us unfettered access to the interpreter, which enabled us to overwrite the sys.dont_write_bytecode variable, generate opcodes for modules on demand and even dump PyXie's configuration.

Figure 19. Search order vulnerability being used to gain control of the interpreter.

Once we were able to output bytecode for modules of our choosing, it was trivial to recover the remapped opcodes and decompile PyXie. A copy of the opcodes for this variant can be found in the appendix.

Functionality

As we previously mentioned, PyXie Lite has been repurposed to focus on the automated collection and exfiltration of data.

Upon execution, it creates a staging directory whose name is based on the output of the tempfile.NamedTemporaryFile() command.

Table 6. Example staging directory name.

Next, it collects data from the system by running a combination of routines that are dictated by the user account PyXie is found to be running under. Table 7 breaks down each of these routines and the types of account they will run under.

Routine	User type	Description
_mimi_redirector	All	Runs Mimikatz in memory. Injects into a newly spawned process from this list: write.exe, notepad.exe, explorer.exe
_main_routine	All	• Collects basic details about the system • Inventories software • Collects cookies • Collects LogMeIn data • Collects Citrix data • Collects KeePass safes
_save_sysinfo	All	Collects uninstall list from registry
_get_passwords	All	Collects passwords with Lazagne
_find_files	System	Searches for and collects files and directories based on keywords, directories and extensions specified in the configuration
_scan_network	System	Runs network scans
_run_shell_cmds	System	Runs a series of commands to gather details about the system
_get_desktop_files	User	Similar to find_files but only searches the current user's Desktop
_take_screenshot	User	Takes a screenshot
_get_ps_history	User	Collects Powershell history

Table 7. PyXie routines.

The list of keywords and directories from configuration provides us some insight into the type of data the attackers are interested in:

passw	logins	wallet	private
confidential	username	wire	access
treason	vault	operation	bribery
contraband	censored	instruction	credent
cardholder	secret	explosive	suspect
personal	cyber	restricted	balance
passport	victim	submarine	checking

saving	routing	esxi	vsphere
spy	admin	newswire	bitcoin
ethereum	n-csr	10-sb	10-q
convict	tactical	engeneering	military
disclosure	attack	infrastruct	marketwired
agreement	illegal	nda	hidden
privacy	fraud	statement	finance
marketwired	clandestine	compromate	concealed
investigation	security		

Table 8. Keywords from PyXie Lite configuration (including misspellings).

As part of the data gathering routines, a number of commands are executed to collect details about the system.

netstat -an

net user

net use

net view /all

net view /all /domain

net share

net config workstation

net group "Domain Admins"

net group "Enterprise Admins"

route print

net localgroup

ipconfig /all

tasklist /V

wmic process

arp -a

gpresult /z

cmdkey /list

net config workstation

nslookup -type=any %userdnsdomain%

vssadmin List Shadows

wmic qfe list

klist

manage-bde -status

nltest /domain_trusts

nltest /domain_trusts /all_trusts

qwinsta

ipconfig /displaydns

systeminfo

dclist

net group "domain admins" /domain

net localgroup "administrators"

wmic path win32_VideoController get name

wmic cpu get name

reg.exe save hklm\security %LOCALAPPDATA%\temp\[RANDOM]

reg.exe save hklm\system %LOCALAPPDATA%\temp\[RANDOM]

reg.exe save hklm\sam %LOCALAPPDATA%\temp\[RANDOM]

Table 9. Executed commands.

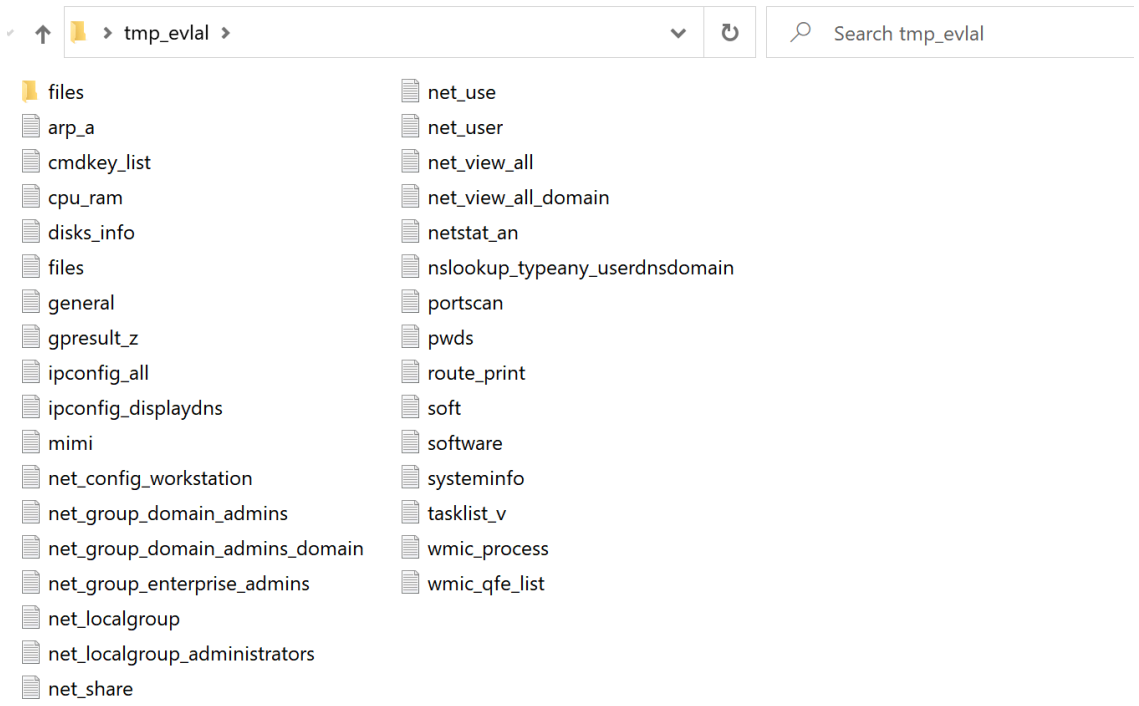

Figure 20. Data collected from the routines in a staging directory prior to exfil.

Exfiltration

The staging directory containing the collected data is added to a compressed ZIP archive and encrypted before being sent to the server specified in the gates section of the config. The archive is encrypted with AES in CBC mode and THIS_KEY_IS_FOR_INTERNAL_USE_ONLY is used as the key. A random 16-byte initialization vector (IV) is used and is prepended to the encrypted archive. The exfil servers we have seen in samples to date have typically been compromised internal servers on the victim's networks listening on ports 31337, 900 and 8443. Although we did not have visibility into how the attackers moved data off the victim network, in at least one incident the exfil server was running Cobalt Strike.

Continue reading: Last, but Not Least: Defray777
