Mystic Stealer | Zscaler

By Brett Stone-Gross

Published: 2023-06-15 · Archived: 2026-04-06 00:20:34 UTC

Technical Analysis

Looking at the existing releases, it seems clear that the developer of Mystic Stealer is looking to produce a stealer on par with the current trends of the malware space while attempting to focus on anti-analysis and defense evasion.

In terms of capabilities, it's a fairly standard set of functionality as seen with many stealers today. The malware collects system information which is packaged together for a check-in to the C2 server:

- Keyboard layout
- Locale
- CPU information
- Number of CPU processors
- Screen dimensions
- Computer name
- Username
- Running processes
- System architecture
- Operating system version

Key data theft functionality includes the ability to capture history and auto-fill data, bookmarks, cookies, and stored credentials from nearly 40 different web browsers. In addition, it collects Steam and Telegram credentials as well as data related to installed cryptocurrency wallets. The malware targets more than 70 web browser extensions for cryptocurrency theft and uses the same functionality to target two-factor authentication (2FA) applications. The approach used by Mystic Stealer is similar to what was reported for Arkei Stealer. Further details on targeted browsers, cryptocurrency plugins, and 2FA apps are available in the appendix.

Depending on a configuration provided by the C2 server, the malware will capture a screenshot of the desktop, which is exfiltrated to the C2 server.

On May 20, the Mystic Stealer seller posted updates that include loader functionality and a persistence capability to forums as shown in Figure 1. *Loader* refers to the ability to download and execute additional malware payloads. This is reflective of a continuing trend where loaders allow one threat actor to support the distribution of affiliate malware being loaded on compromised devices. This is already a notable risk for many organizations due to the use of malware distribution networks and initial access brokers for the distribution of high-severity payloads like ransomware. It underscores the need to take preventative steps to ensure a security posture that reduces the risk of malware delivery and footholds early on in attack campaigns.

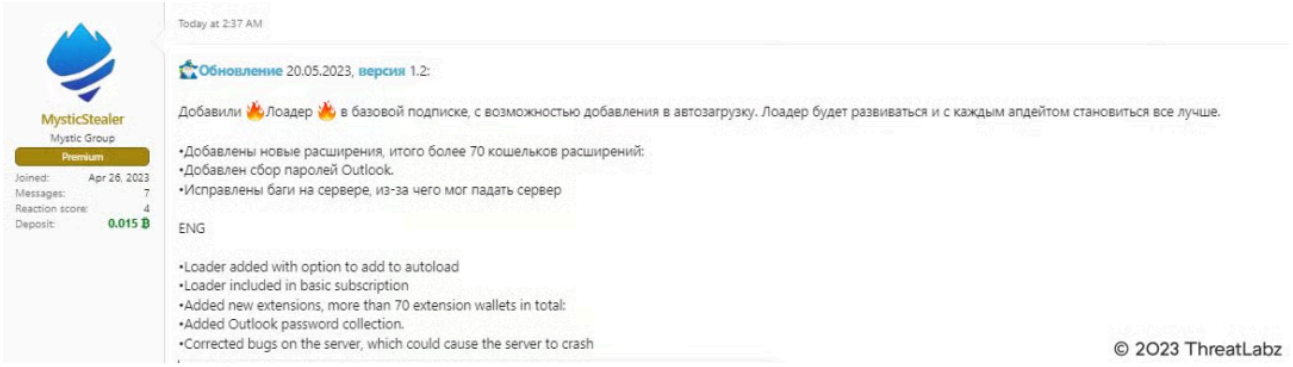

Figure 1. MysticStealer forum post advertising v1.2 update with loader support

As previously noted, there are several anti-analysis and evasion features additionally present in Mystic Stealer:

Binary expiration. The trojan will terminate execution if the running build is older than a specified date. This is likely an execution guardrail that attempts to prevent anti-malware researchers and sandboxes that analyze the sample much later than when it was intended to be distributed or executed on victim machines. Figure 2 shows a Mystic Stealer sample that retrieves the current system time and compares the value to 1685318914 (0x6473ED02), which when converted from an epoch to a timestamp translates to Sun May 28 17:08:34 2023.

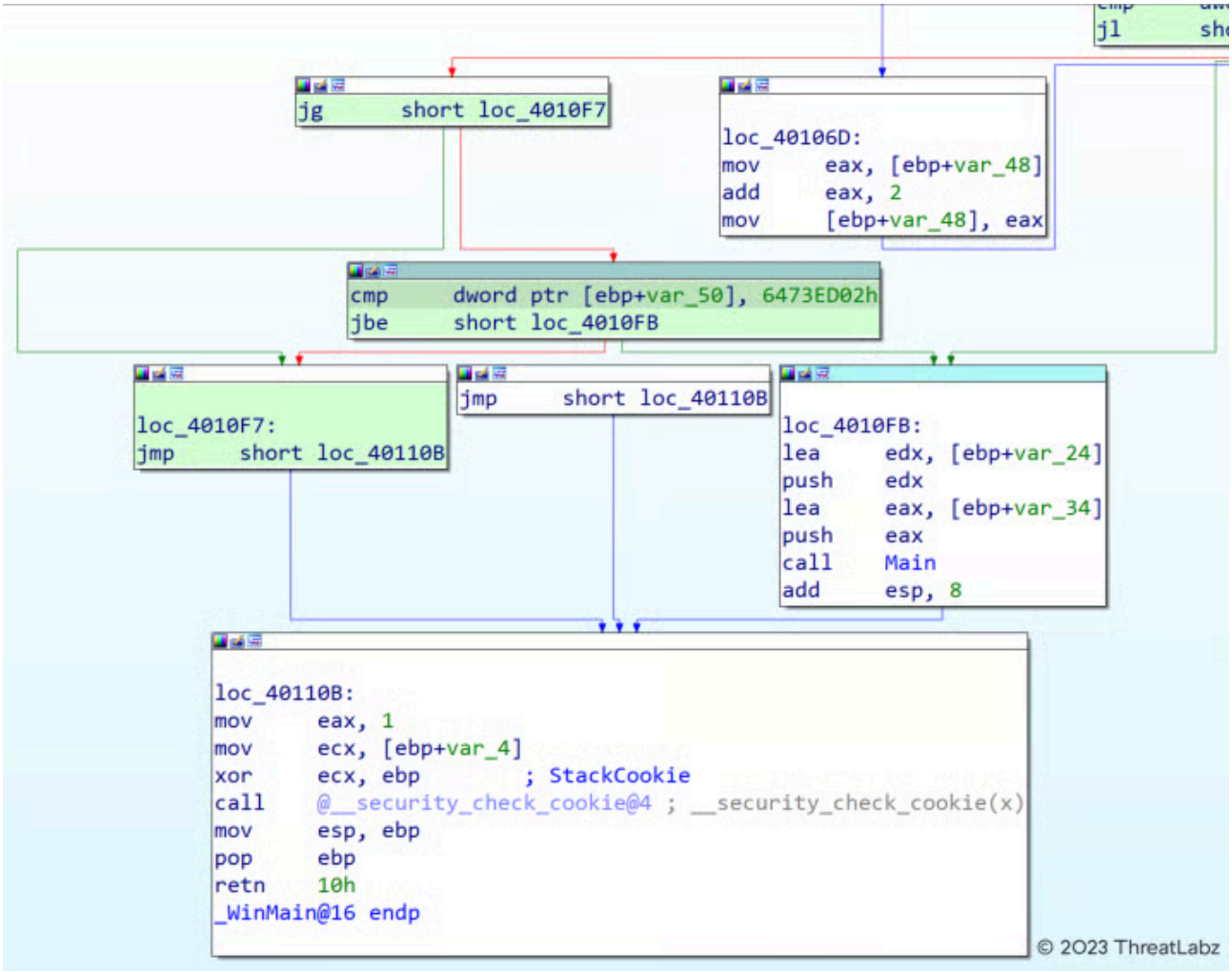

Figure 2. Example Mystic Stealer date expiration feature

Anti-virtualization. Mystic Stealer is configurable and some samples contain anti-VM features, detecting hypervisor runtime environments, and avoiding execution. This is helpful for avoiding execution in sandbox environments but it isn't always effective.

Mystic uses the *CPUID* assembly instruction to detect virtual environments by inspecting the result for specific values that are indicative of virtualization software. In particular, the code checks for the manufacturer ID string (with a length of 12 bytes) for the following values:

- “XenVMMXenVMM” (Xen HVM)
- “VMwareVMware” (VMware)
- “Microsoft Hv” (Microsoft Hyper-V)
- “ KVMKVMKVM “ (KVM)
- “prl hyperv “ (Parallels)
- “VBoxVBoxVBox” (VirtualBox)

This detection code is likely derived from Pafish.

Windows APIs imported by hash. The stealer resolves and dynamically loads Windows APIs using a custom XOR based hashing algorithm represented in the Python snippet shown below:

```
i = 0
for val in function_name:
    i = ord(val) ^ ((0x240CE91 * i) & 0xffffffff)
print(hex(i))
```


Note that the constant value (e.g., 0x240CE91) changes between Mystic samples. The malware walks the export tables for the following Windows DLLs and hashes each export name until a match is found:

- Kernel32.dll
- Advapi32.dll
- Kernel32.dll
- Gdiplus.dll
- Crypt32.dll
- User32.dll
- Ws2_32.dll
- Ole32.dll
- Gdi32.dll
- Ntdll.dll

Dynamic constant calculation. Constant values in the code are obfuscated and dynamically calculated at runtime. For example, the API hashing algorithm shown above uses the constant 0x240CE91. However, this constant does not directly exist in the code. Instead, the value 0x240CEA6 is present and the code performs an XOR operation with the value 0x37 to produce the actual constant 0x240CE91 as shown in Figure 3.

```
.text:0041ACF7                          ; ---------------------------------------------------------------------------
.text:0041ACF7
.text:0041ACF7                          loc_41ACF7:                             ; CODE XREF: HashImportFunctionName+C2↓j
.text:0041ACF7                                                                  ; HashImportFunctionName+D3↓j
.text:0041ACF7 5F                                       pop     edi
.text:0041ACF8
.text:0041ACF8                          loc_41ACF8:                             ; CODE XREF: HashImportFunctionName+71↑j
.text:0041ACF8 0F BE 06                                 movsx   eax, byte ptr [esi]
.text:0041ACFB 33 D2                                    xor     edx, edx
.text:0041ACFD 3B 45 FC                                 cmp     eax, [ebp+var_4]
.text:0041AD00 74 2B                                    jz      short loc_41AD2D
.text:0041AD02
.text:0041AD02                          loc_41AD02:                             ; CODE XREF: HashImportFunctionName+A7↓j
.text:0041AD02 C7 45 DC 7E F7 00 00                     mov     [ebp+var_24], 0F77Eh
.text:0041AD09 81 75 DC B9 00 00 00                     xor     [ebp+var_24], 0B9h
.text:0041AD10 C7 45 F4 A6 CE 40 02                     mov     [ebp+var_C], 240CEA6h
.text:0041AD17 83 75 F4 37                              xor     [ebp+var_C], 37h
.text:0041AD1B 0F AF 55 F4                              imul    edx, [ebp+var_C]
.text:0041AD1F 0F BE 06                                 movsx   eax, byte ptr [esi]
.text:0041AD22 33 D0                                    xor     edx, eax
.text:0041AD24 46                                       inc     esi
.text:0041AD25 0F BE 0E                                 movsx   ecx, byte ptr [esi]
.text:0041AD28 3B 4D FC                                 cmp     ecx, [ebp+var_4]
.text:0041AD2B 75 D5                                    jnz     short loc_41AD02
.text:0041AD2D
.text:0041AD2D                          loc_41AD2D:                             ; CODE XREF: HashImportFunctionName+7C↑j
.text:0041AD2D 8B C2                                    mov     eax, edx
.text:0041AD2F 5E                                       pop     esi
.text:0041AD30 C9                                       leave
.text:0041AD31 C3                                       retn
.text:0041AD32                          ; ---------------------------------------------------------------------------
                                                                                © 2023 ThreatLabz
```

Figure 3. Example Mystic Stealer constant obfuscation technique

Encrypted binary custom protocol. The client communicates with the C2 server using a custom protocol over TCP, which we discuss in more depth later.

Polymorphic string obfuscation. We identified that the malware obfuscates strings using a library that is very similar to ADVobfuscator. The obfuscator generates code at compile time that builds strings on the stack, which are then decrypted at runtime. The obfuscation is polymorphic, and therefore, every sample will contain strings that are uniquely encrypted with simple mathematical operations such as addition, subtraction, and XOR. As a result, this technique may bypass static antivirus signatures and complicate malware reverse engineering.

The Mystic Stealer seller refers to this obfuscation as a *morpher* that obfuscates builds with full undetectability (FUD) in sales threads. In one forum, the seller advertised that the project's morpher enabled the bypass of SmartScreen, which members identified as a dubious claim based on the operation of obfuscators and SmartScreen. Some forum users suspected the use of an open-source obfuscator. This ended up as a point of contention in the forum, lowering the perception and trust of the project with some users.
