Zloader | ThreatLabz

zscaler.com/blogs/security-research/zloader-learns-old-tricks

Santiago Vicente

Introduction

Zloader (a.k.a. Terdot, DELoader, or Silent Night) is a modular trojan based on leaked ZeuS source code. As detailed in our previous blog, Zloader reemerged following an almost two-year hiatus with a new iteration that included modifications to its obfuscation techniques, domain generation algorithm (DGA), and network communication.

Most recently, Zloader has reintroduced an anti-analysis feature similar to one that was present in the original ZeuS 2.x code. The feature restricts Zloader's binary execution to the infected machine. This characteristic of ZeuS was abandoned by many malware variants derived from the leaked source code including Zloader, until now. In this blog post, we explain how this anti-analysis feature works and how it differs from the original ZeuS implementation.

Key Takeaways

- Zloader (a.k.a. Terdot, DELoader, or Silent Night) is a modular trojan based on the leaked ZeuS source code dating back to 2015.
- Zloader has continued to evolve since its resurrection around September 2023 after an almost two-year hiatus.
- The latest version, 2.4.1.0, introduces a feature to prevent execution on machines that differ from the original infection. A similar anti-analysis feature was present in the leaked ZeuS 2.X source code, but implemented differently.

Technical Analysis

In the upcoming sections, we explore the technical intricacies of Zloader's latest anti-analysis feature introduced in versions 2.4.1.0 and 2.5.1.0. We also draw comparisons to ZeuS to provide a comprehensive understanding of their respective approaches.

Registry check

Zloader samples with versions greater than 2.4.1.0 will abruptly terminate if they are copied and executed on another system after the initial infection. This is due to a Windows registry check for the presence of a specific key and value.

The screenshot below shows the Windows Registry check failing in a malware sandbox.

Key Opened

Key Path	Completion	Count	Source Address	Symbol
HKEY_CURRENT_USER	success or wait	1	7FF6A9F7480E	RegOpenKeyExW
HKEY_CURRENT_USER\Software\Microsoft\Lfbq	object name not found	1	7FF6A9F7480E	RegOpenKeyExW

Figure 1: Registry key check performed in a sandbox.

The registry key and value are generated based on a hardcoded seed that is different for each sample.

The Python code below replicates the algorithm to generate the registry key.

```
#!/usr/bin/env python3
SEED = 0x1C5EE76F0FE82329
def calculate_registry_key(seed):
   key = ""
   key_length = 1 + seed % 8

   if key_length < 4:
       key_length = 4

   for i in range(key_length):
       key += chr(seed % 0x1A + 0x61)
       seed = (((seed << 8) | (seed >> (64 - 8))) & 0xffffffffffffffff) + 1

   key = key.capitalize()
   return key
print(calculate_registry_key(SEED))
```

If the registry key/value pair is manually created (or this check is patched), Zloader will successfully inject itself into a new process. However, it will terminate again after executing only a few instructions. This is due to a secondary check in Zloader’s MZ header.

MZ header check

A bit further in the code, there is an additional check that involves a `DWORD` present in the MZ header at the offset 0x30, which is only executed after being injected into a new process. The `DWORD` used in the check of the analyzed sample can be seen in the image below.

```
Offset(h) 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F

00000000  4D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00  MZ..........ÿÿ..
00000010  B8 00 00 00 00 00 00 00 40 00 00 00 00 00 00 00  ¸.......@.......
00000020  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030  44 12 D0 AA 00 00 00 00 00 00 00 00 E8 00 00 00  D.Ðª........è...
00000040  0E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68  ..º..´.Í!¸.LÍ!Th
00000050  69 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00  mode....$.......
00000080  2D 23 CB 40 69 42 A5 13 69 42 A5 13 69 42 A5 13  -#Ë@iB¥.iB¥.iB¥.
00000090  7D 29 A3 12 68 42 A5 13 7D 29 A4 12 6E 42 A5 13  })£.hB¥.})¤.nB¥.
```

Figure 2: MZ header with random `DWORD` at 0x30 offset.

The `DWORD` at the 0x30 offset is part of the ten reserved `WORDs` that go from offset 0x28 to offset 0x3C of the MZ header. These bytes are usually `null`. However, in the example above, the malware contained an integer value (0xAAD01244), which is compared with the file size (0x29A00). Since this integer is a very large number, the check fails. The decompiled code of the file size check is shown in the figure below.

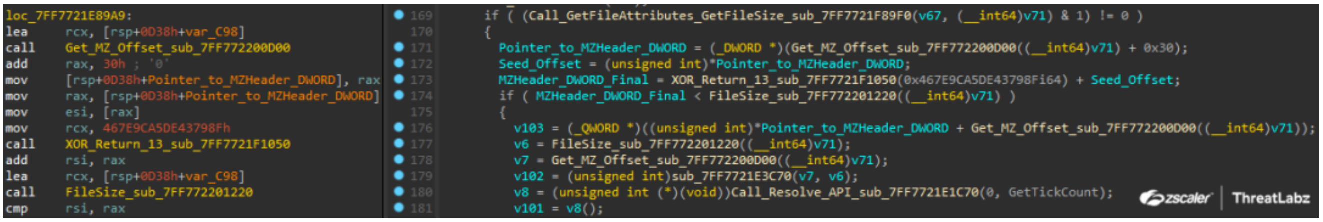

Figure 3: Decompiled code of the file size check against the MZ `DWORD`.

What the malware developers are doing here is utilizing the additional MZ header `DWORD` as a pointer to the seed's offset, which explains the purpose of the check. This is due to the `DWORD` being overwritten after the initial execution. If the pointer points beyond the binary, it indicates that the seed has already been written, eliminating the need for reinitialization.

This suggests that the initial binary for system infection must include a `null` seed, with the MZ `DWORD` at 0x30 holding the seed's offset. Subsequently, this offset is initialized with a pseudo-random `QWORD` generated via the Mersenne Twister algorithm, leaving a hardcoded seed that differs per infected sample.

The figure below shows the decompiled code where the seed is being generated and written.

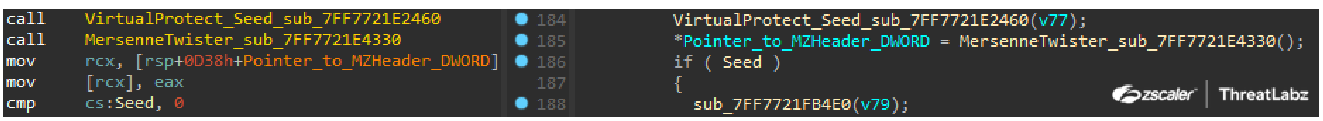

Figure 4: Decompiled code where the seed is first created.

Without the seed and MZ header values set correctly, the Zloader sample won't run or install on a different machine, unless it is patched or if the environment is replicated with all the registry and disk paths/names, alongside all the original artifacts from the original victim's machine.

Registry value content

In previous versions of Zloader, there was a single registry key and value containing some machine information (install path, computer/bot ID, victim-specific RC4 key, etc.), similar to the ZeuS `PeSettings` we will examine in the next section. The key/value pair was encrypted with the ZeuS `VisualEncrypt` algorithm and RC4, using the RSA key present in the static configuration as the key, but it wasn't used to avoid infecting a new machine, as it was created again when executed in a different environment.

Now, there is an additional value created using the seed previously mentioned.

The figure below shows the registry keys and values added to the victim's system during the infection process.

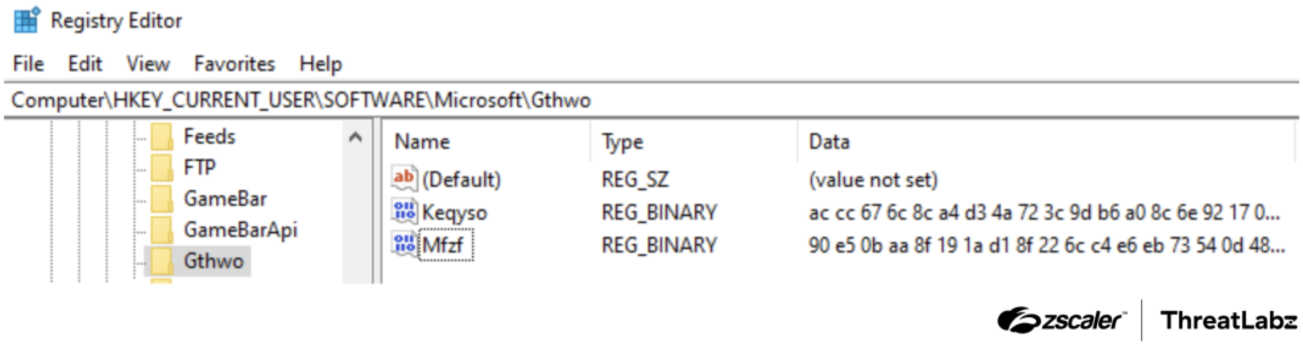

Figure 5: Registry keys and values added when infecting the machine.

The content has a fixed length of 1,418 bytes and is encrypted with RC4, but without the additional `VisualEncrypt` layer. The RC4 key is also based on the seed generated while performing the infection, which is then used to create the names of the registry key and value.

The decrypted format and content are as follows:

```
00000000  41 00 64 00 6f 00 62 00  65 00 5c 00 49 00 6e 00  |A.d.o.b.e.\.I.n.|
00000010  66 00 72 00 61 00 42 00  61 00 73 00 65 00 2e 00  |f.r.a.B.a.s.e...|
00000020  65 00 78 00 65 00 00 00  00 00 00 00 00 00 00 00  |e.x.e...........|
00000030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000040  57 00 61 00 62 00 75 00  75 00 5c 00 45 00 66 00  |W.a.b.u.u.\.E.f.|
00000050  79 00 63 00 79 00 64 00  6d 00 61 00 00 00 00 00  |y.c.y.d.m.a.....|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000080  57 00 61 00 62 00 75 00  75 00 5c 00 47 00 65 00  |W.a.b.u.u.\.G.e.|
00000090  78 00 61 00 6e 00 69 00  00 00 00 00 00 00 00 00  |x.a.n.i.........|
000000a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000c0  57 00 61 00 62 00 75 00  75 00 5c 00 4c 00 6f 00  |W.a.b.u.u.\.L.o.|
000000d0  6b 00 61 00 79 00 6c 00  62 00 6f 00 00 00 00 00  |k.a.y.l.b.o.....|
000000e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000000f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000100  57 00 61 00 62 00 75 00  75 00 5c 00 47 00 79 00  |W.a.b.u.u.\.G.y.|
00000110  79 00 70 00 6b 00 00 00  00 00 00 00 00 00 00 00  |y.p.k...........|
00000120  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000130  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000140  57 00 61 00 62 00 75 00  75 00 5c 00 45 00 71 00  |W.a.b.u.u.\.E.q.|
00000150  71 00 61 00 00 00 00 00  00 00 00 00 00 00 00 00  |q.a.............|
00000160  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000170  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000180  57 00 61 00 62 00 75 00  75 00 5c 00 59 00 77 00  |W.a.b.u.u.\.Y.w.|
00000190  77 00 75 00 00 00 00 00  00 00 00 00 00 00 00 00  |w.u.............|
000001a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001c0  57 00 61 00 62 00 75 00  75 00 5c 00 49 00 76 00  |W.a.b.u.u.\.I.v.|
000001d0  76 00 65 00 64 00 00 00  00 00 00 00 00 00 00 00  |v.e.d...........|
000001e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000001f0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000200  57 00 61 00 62 00 75 00  75 00 5c 00 48 00 61 00  |W.a.b.u.u.\.H.a.|
00000210  6b 00 6f 00 67 00 69 00  00 00 00 00 00 00 00 00  |k.o.g.i.........|
00000220  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000230  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000240  59 00 66 00 6f 00 77 00  76 00 6f 00 5c 00 46 00  |Y.f.o.w.v.o.\.F.|
00000250  75 00 76 00 61 00 61 00  71 00 00 00 00 00 00 00  |u.v.a.a.q.......|
00000260  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000270  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000280  59 00 66 00 6f 00 77 00  76 00 6f 00 5c 00 4d 00  |Y.f.o.w.v.o.\.M.|
00000290  79 00 6c 00 75 00 6b 00  00 00 00 00 00 00 00 00  |y.l.u.k.........|
000002a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
000002c0  59 00 66 00 6f 00 77 00  76 00 6f 00 5c 00 45 00  |Y.f.o.w.v.o.\.E.|
000002d0  73 00 6e 00 6f 00 00 00  00 00 00 00 00 00 00 00  |s.n.o...........|
000002e0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
0000058A
```

The structure is divided into 64 bytes for each entry. The first structure is the binary path inside `%APPDATA%`, and the following are the Zloader modules.

ZeuS implementation

It’s been thirteen years since the ZeuS 2.0.8 source code was leaked, but it is still widely leveraged by threat actors and some of its concepts are still relevant. The technique described in the section above, and used by Zloader to store the installation information and avoid being run on a different system, was also performed by ZeuS v2, but implemented in a different way.

In ZeuS, the binary had an overlay section called `PeSettings`, where the installation information was stored instead of in the registry. The encrypted ZeuS overlay section is shown in the figure below.

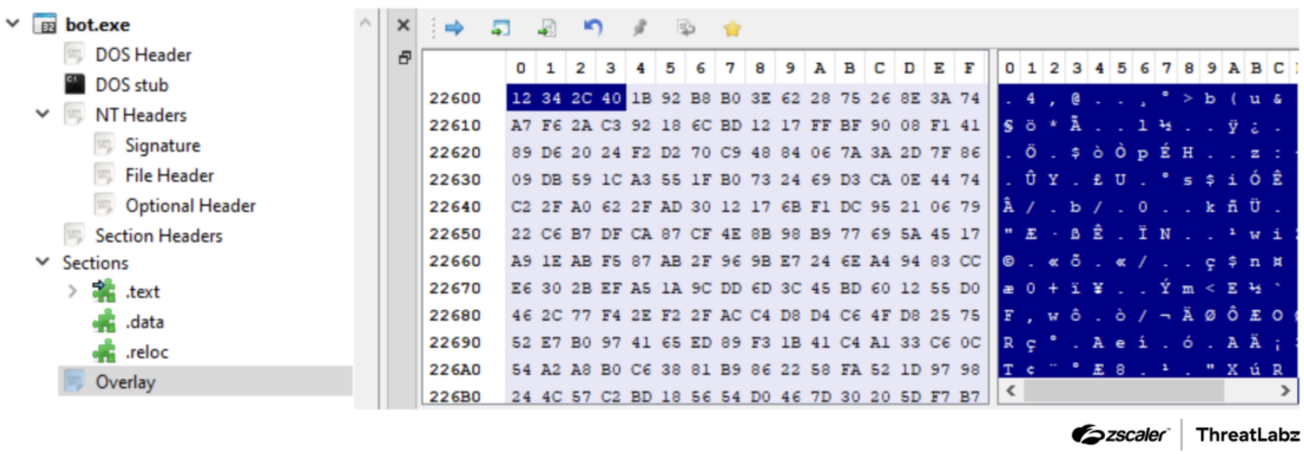

Figure 6: The encrypted ZeuS overlay section.

The header of this section is decrypted with the RC4 key present in the static config. The figure below shows the ZeuS section header.

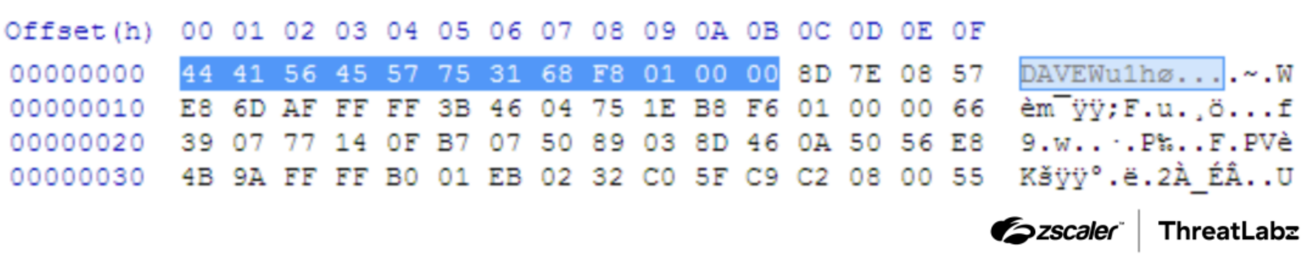

Figure 7: ZeuS overlay section header.

The decrypted header is composed of three `DWORDs`:

- Magic word (`DAVE`)
- CRC32 of the data
- Size of the data

If the size of the data is equal to 0xC, it means the trojan is not installed and will proceed with the infection to generate all the required information, such as the computer/bot ID, install paths, and machine-specific RC4 key, which is generated per install and stored as an initialized RC4 S-box.

Then, ZeuS will encrypt the `PeSettings` again and replace the overlay data with it, while changing the header CRC and data size `DWORDs`.

Below you can see the `PeSettings` structure in its decrypted form:

```
00000000  e6 01 00 00 41 00 44 00  4d 00 49 00 4e 00 2d 00  |....A.D.M.I.N.-.|
00000010  50 00 43 00 5f 00 45 00  35 00 33 00 32 00 36 00  |P.C._.E.5.3.2.6.|
00000020  34 00 38 00 41 00 34 00  34 00 43 00 43 00 37 00  |4.8.A.4.4.C.C.7.|
00000030  46 00 31 00 43 00 00 00  00 00 00 00 00 00 00 00  |F.1.C...........|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
*
00000070  00 00 00 00 00 00 00 00  00 00 00 00 e4 50 d2 69  |.............P.i|
00000080  18 6c e3 11 b3 bc 80 6e  6f 6e 69 63 01 89 b5 78  |.l.....nonic...x|
00000090  79 63 ae 4b f3 14 94 9a  ab db c2 be 09 32 df 16  |yc.K.........2..|
000000a0  bc a3 0a 33 57 6f 49 e5  21 62 c6 5f 12 e2 97 25  |...3WoI.!b._...%|
000000b0  87 55 b7 a0 da a8 67 36  29 dc 08 f1 8a 6d c9 e8  |.U....g6)....m..|
000000c0  91 13 90 54 6b 8f 2b 5e  68 46 9b 9e 69 80 e4 76  |...Tk.+^hF..i..v|
000000d0  88 85 cc bd bb 40 ce 10  6a 71 75 5d 93 dd 4d 07  |[email protected]]..M.|
000000e0  92 7e ba 61 ad 1d 34 f6  ac 98 a5 af 59 86 3d 27  |.~.a..4.....Y.='|
000000f0  5c 38 b6 c7 aa c0 9c 52  d0 64 77 5a 3e 8e fe 0d  |\8.....R.dwZ>...|
00000100  7f bf 1b 20 f8 00 a4 6c  45 3b 41 8d 81 05 e6 d4  |... ...lE;A.....|
00000110  f9 e3 9f 02 37 b1 d9 60  ef 83 1f e9 cd a2 17 8c  |....7..`........|
00000120  2c c4 c1 15 65 4c d5 8b  ca 3c 26 1e ec 6e 30 d8  |,...eL...<&..n0.|
00000130  a9 4a 2f 7d 18 a7 7b 56  0f f7 ea 39 1a 96 c8 4e  |.J/}..{V...9...N|
00000140  73 b3 d2 f5 cb d3 74 e0  5b 51 50 eb 84 0c b4 b2  |s.....t.[QP.....|
00000150  3a ee 4f fb 58 1c 28 70  a6 43 82 66 7c 04 22 0b  |:.O.X.(p.C.f|.".|
00000160  cf 3f f4 42 44 c5 23 47  53 19 0e 35 11 7a 95 48  |.?.BD.#GS..5.z.H|
00000170  ed 2a f2 c3 99 b8 2e 06  24 ff e7 fc 9d fd d7 b0  |.*......$.......|
00000180  b9 d6 31 e1 d1 fa f0 de  a1 2d 72 03 00 00 55 76  |..1......-r...Uv|
00000190  71 69 63 75 5c 79 70 77  75 66 2e 65 78 65 00 00  |qicu\ypwuf.exe..|
000001a0  00 00 47 61 75 6c 5c 75  6d 70 75 68 2e 62 79 67  |..Gaul\umpuh.byg|
000001b0  00 00 00 00 00 00 4f 74  68 65 79 6e 00 00 00 00  |......Otheyn....|
000001c0  55 71 63 75 73 00 00 00  00 00 50 69 67 6f 63 6f  |Uqcus.....Pigoco|
000001d0  00 00 00 00 43 61 73 75  73 61 00 00 00 00 8a 2d  |....Casusa.....-|
000001e0  48 10 30 a0 77 68 15 00  00 83                    |H.0.wh....|
```

When trying to run a sample that's already installed, it will generate the computer/bot ID, and if it doesn't match with the one stored in the `PeSettings`, ZeuS will exit. The same thing occurs if the install paths don't match.

Conclusion

In recent versions, Zloader has adopted a stealthy approach to system infections. This new anti-analysis technique makes Zloader even more challenging to detect and analyze. The samples analyzed by ThreatLabz have all been pre-initialized, suggesting a more targeted distribution strategy.

Zscaler ThreatLabz continues to track this threat and add detections to protect our customers.

Zscaler Coverage

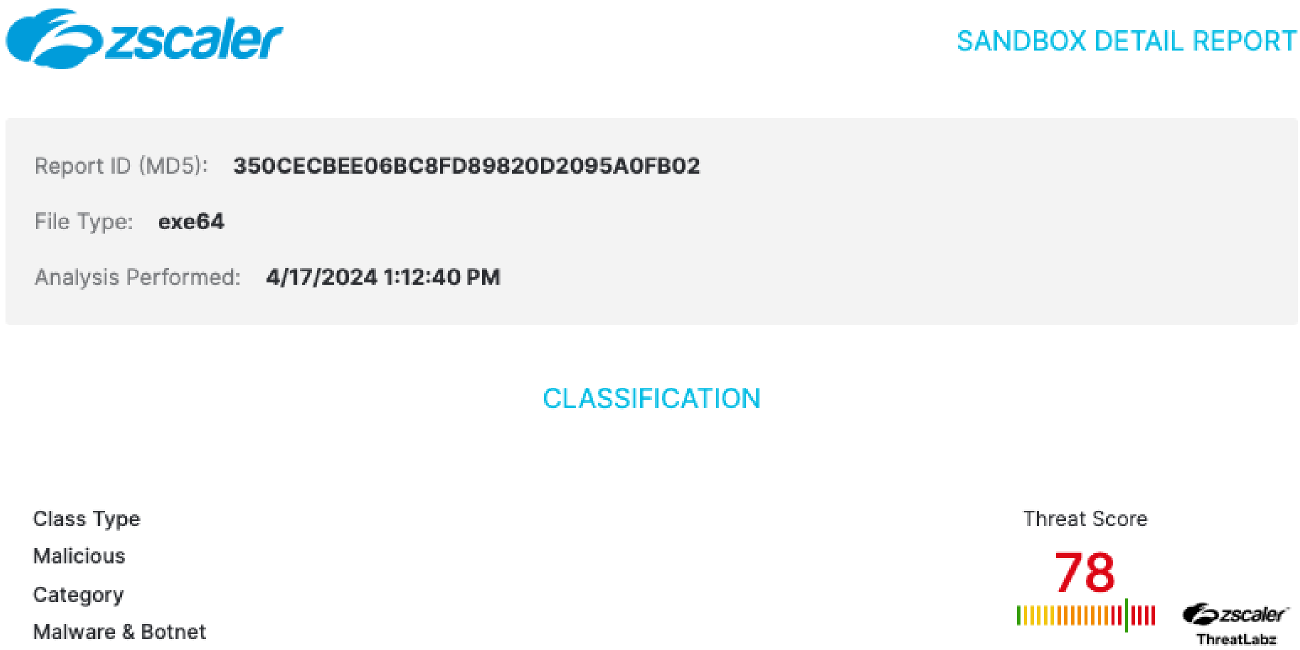

Figure 8: Zscaler Cloud Sandbox report

In addition to sandbox detections, Zscaler's multilayered cloud security platform detects indicators related to Zloader at various levels with the following threat names:

- Win64.Downloader.Zloader

Indicators Of Compromise (IOCs)

SHA256	Description
cba9578875a3e222d502bb6a85898939bb9e8e247d30fcc0d44d83a64919f448	Zloader sample
85962530c71cd31c102853d64a8829f93b63bd1406bdec537b9d8c200f8f0bcc	Zloader sample
b1a6bf93d4ee659db03e51a3765d4d3c2ee3f1b56bd9b701ab5939d63f57d9ee	Zloader sample
85b1a980eb8ced59f87cb5dd7702e15d6ca38441c4848698d140ffd37d2b55e6	Zloader sample

URLs

URL	Description
https://eingangfurkunden[.]digital/	Zloader C2
https://citscale[.]com/api.php	Zloader C2
https://adslsdfdsfmo[.]world/	Zloader C2
https://gycltda[.]cl/home/wp-api.php	Zloader C2

Get the latest Zscaler blog updates in your inbox