Zloader Analysis | ThreatLabz

zscaler.com/blogs/security-research/zloader-no-longer-silent-night

Santiago Vicente, Ismael Garcia Perez

Amplifying the voices of real-world digital and zero trust pioneers

Visit now

Resource Center

Stay up to date on best practices

Events & Trainings

Find programs, certifications, and events

Security Research & Services

Get research and insights at your fingertips

Tools

Tools designed for you

Community & Support

Connect and find support

Industry & Market Solutions

See solutions for your industry and country

Resource Center

Stay up to date on best practices

Events & Trainings

Find programs, certifications, and events

Security Research & Services

Get research and insights at your fingertips

Tools

Tools designed for you

Community & Support

Industry & Market Solutions

About Zscaler
Discover how it began and where it’s going

Partners
Meet our partners and explore system integrators and technology alliances

News & Announcements

Introduction

Zloader (aka Terdot, DELoader, or Silent Night), is a modular trojan born from the leaked Zeus source code. It surfaced publicly in 2016 during a targeted campaign against German banks[1], but its malicious activity traces back to at least August 2015. Zloader's first run persisted until the beginning of 2018 when its activities abruptly ceased. Its resurgence at the end of 2019, marketed in underground forums as "Silent Night", came with substantial alterations. The evolution of Zloader progressed steadily, leading to the development of version 2.0.0.0 around September 2021. Similar to Qakbot, the threat actors using Zloader also pivoted from conducting banking fraud to ransomware. In April 2022, security researchers executed a takedown operation[2] to dismantle the botnet leading to an extended period of inactivity.

After an almost two-year hiatus, Zloader reemerged with a new iteration that appears to have started development in September 2023. These new changes include new obfuscation techniques, an updated domain generation algorithm (DGA), RSA encryption for network communications, and the loader now has native support for 64-bit versions of Windows. Initially, this new version was labeled with the old version number 2.0.0.0. However, over the past several months, they released version 2.1.6.0 and 2.1.7.0. In this blog, we will explore these new updates to Zloader.

Key Takeaways

- Zloader dates back to 2015 and has been advertised in underground cybercriminal forums under the name "Silent Night" since the end of 2019.
- Zloader has returned after an almost two-year hiatus after being taken down in April 2022 by security researchers.
- The new version of Zloader made significant changes to the loader module, which added RSA encryption, updated the domain generation algorithm, and is now compiled for 64-bit Windows operating systems for the first time.
- Zloader continues to use junk code for obfuscation, as well as API import hashing and string encryption in an attempt to hinder malware analysis.

Technical Analysis

In the following sections, we dive into the technical details surrounding Zloader's new updates to their anti-analysis techniques, embedded configuration, DGA, and network encryption.

Anti-analysis techniques

Zloader uses a combination of API import hashing, junk code, a filename check, and string obfuscation. The following sections analyze each technique.

Imports and API resolution

The newest Zloader samples only import a few functions from the kernel32 library. The remaining imports are resolved at runtime using checksums to obfuscate the functions that are used. This technique, already present in older versions, changes its implementation, adding an XOR constant which changes between samples. Python code that replicates the API hashing algorithm is shown below.

```python
def calculate_checksum(func_name, xor_constant):
    checksum = 0
    for element in func_name.upper():
        checksum = 16*checksum - (0 - (ord(element)+1))
        if checksum & 0xf0000000 != 0:
            checksum = ((((checksum & 0xf0000000) >> 24) ^ checksum) & 0xfffffff)
    return checksum ^ xor_constant
```


Code sample available on GitHub.

Junk code

Similar to previous versions, Zloader uses custom obfuscation. The new version of Zloader adds junk code that consists of various arithmetic operations, as shown in Figure 1 below.

```
__int64 __fastcall malware_resolve_int(int a1)
{
  int v3; // [rsp+24h] [rbp-Ch]
  int v4; // [rsp+28h] [rbp-8h]
  int v5; // [rsp+28h] [rbp-8h]
  int v6; // [rsp+2Ch] [rbp-4h]
  int v7; // [rsp+2Ch] [rbp-4h]

  v3 = a1 ^ 0x34420CDD;
  v4 = a1 ^ 0x172;
  v6 = sub_140016430(a1 ^ 0x172u, a1);
  if ( a1 == v3 || a1 != v4 )
  {
    v4 = v6 + a1 + a1 - v3;
    v6 = v4 + 849;
  }
  v5 = v4 & v6;
  v7 = v5 & 0x200;
  if ( a1 == v3 || a1 <= v3 && a1 != v5 )
  {
    v5 = 914 * v7;
    v7 = v3 | (914 * v7);
  }
  if ( a1 == v5 && ((sub_1400180D0(a1, v3) & 1) != 0 || a1 == v5) )
    v5 = v3 ^ (v7 * a1 - 637);
  dword_14002B268 = v5;
  return (unsigned int)v3;
}
```


Figure 1. Example Zloader 2.1 junk code

In Figure 1, the instructions inside the red box are the junk code.

Anti-sandbox

Each Zloader sample expects to be executed with a specific filename. If the filename does not match what the sample expects, it will not execute further. This could evade malware sandboxes that rename sample files. Figure 2 shows an example of a Zloader sample that expects its filename to be **CodeForge.exe**.

```
proc_addr_ptr = malware_getProcAddr((__int64)"CodeForge.exe");
if ( proc_addr_ptr )
{
  if ( (malware_Init_alt_api_tables() & 1) != 0 )
  {
```


Figure 2. Example of Zloader's anti-analysis filename check

ThreatLabz has observed Zloader use the following filenames:

- CodeForge.exe
- CyberMesh.exe
- EpsilonApp.exe
- FusionBeacon.exe
- FusionEcho.exe
- IonBeacon.dll
- IonPulse.exe

- KineticaSurge.dll
- QuantumDraw.exe
- SpectraKinetic.exe
- UltraApp.exe

String obfuscation

Similar to prior versions, Zloader implements a string obfuscation algorithm for some of the malware's important strings such as registry paths, DLL names, and the DGA's top-level domain (TLD) using XOR with a hardcoded key. Python code that replicates the string obfuscation algorithm is shown below:

```
def str_deobfuscate(enc_bin, enc_key):
    res = ''
    for i, element in enumerate(enc_bin):
      res += chr( ((element ^ 0xff) & (enc_key[i % len(enc_key)])) | (~(enc_key[i % len(enc_key)]) & element))
    return res
```


Code sample available on GitHub.

The encryption key differs between samples and is also hardcoded in the *.rdata* section as shown in Figure 3 below.

```
; _BYTE STRING_OBFUSCATION_KEY[32]
STRING_OBFUSCATION_KEY db 8Ah, 0BDh, 0F7h, 6Bh, 9, 0C7h, 0EDh, 40h, 59h, 3Fh
                                        ; DATA XREF: Decrypt_String_WIDE_sub_
                                        ; sub_140011550+62↑o ...
                db 59h, 2Ah, 0FDh, 14h, 0ACh, 13h, 39h, 0Fh dup(0)
str_NtAllocateVirtualMemory db 0C4h ; Д ; DATA XREF: sub_1400032F0+1F↑o
                                        ; sub_1400218F0+CA↑o
                                        ; NtAllocateVirtualMemory
                db 0C9h ; Й
                db 0B6h ; ¶
                db    7
                db  65h ; e
```


Figure 3. Example string obfuscation key used by Zloader

A list of Zloader's obfuscated strings is shown in the Appendix.

Static configuration encryption and structure

The Zloader static configuration is still encrypted using RC4 with a hardcoded alphanumeric key, but the structure is slightly different. The botnet ID, campaign name, and command-and-control servers (C2s) are set at fixed offsets, in addition to an RSA public key that replaces the old RC4 key that was used for network encryption. ThreatLabz has observed 15 unique new Zloader samples and all of them have the same RSA public key, likely indicating there is currently only a single threat actor using the malware.

An example Zloader static configuration is shown below.

```
00000000  00 00 00 00 42 69 6e 67  5f 4d 6f 64 35 00 00 00  |....Bing_Mod5...|
00000010  00 00 00 00 00 00 00 00  00 4d 31 00 00 00 00 00  |.........M1.....|
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 68 74  |..............ht|
00000030  74 70 73 3a 2f 2f 61 64  73 6c 73 74 69 63 6b 65  |tps://adslsticke|
00000040  72 68 69 2e 77 6f 72 6c  64 00 00 00 00 00 00 00  |rhi.world.......|
00000050  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  02 00 00 00 01 00 00 00  |................|
000002c0  2d 2d 2d 2d 2d 42 45 47  49 4e 20 50 55 42 4c 49  |-----BEGIN PUBLI|
000002d0  43 20 4b 45 59 2d 2d 2d  2d 2d 0a 4d 49 47 66 4d  |C KEY-----.MIGfM|
000002e0  41 30 47 43 53 71 47 53  49 62 33 44 51 45 42 41  |A0GCSqGSIb3DQEBA|
000002f0  51 55 41 41 34 47 4e 41  44 43 42 69 51 4b 42 67  |QUAA4GNADCBiQKBg|
00000300  51 44 4b 47 41 4f 57 56  6b 69 6b 71 45 37 54 79  |QDKGAOWVkikqE7Ty|
00000310  4b 49 4d 74 57 49 38 64  46 73 61 0a 6c 65 54 61  |KIMtWI8dFsa.leTa|
00000320  4a 4e 58 4d 4a 4e 49 50  6e 52 45 2f 66 47 43 7a  |JNXMJNIPnRE/fGCz|
00000330  71 72 56 2b 72 74 59 33  2b 65 78 34 4d 43 48 45  |qrV+rtY3+ex4MCHE|
00000340  74 71 32 56 77 70 70 74  68 66 30 52 67 6c 76 38  |tq2Vwppthf0Rglv8|
00000350  4f 69 57 67 4b 6c 65 72  49 4e 35 50 0a 36 4e 45  |OiWgKlerIN5P.6NE|
00000360  79 43 66 49 73 46 59 55  4d 44 66 6c 64 51 54 46  |yCfIsFYUMDfldQTF|
00000370  30 33 56 45 53 38 47 42  49 76 48 71 35 53 6a 6c  |03VES8GBIvHq5Sjl|
00000380  49 7a 37 6c 61 77 75 77  66 64 6a 64 45 6b 61 48  |Iz7lawuwfdjdEkaH|
00000390  66 4f 6d 6d 75 39 73 72  72 61 66 74 6b 0a 49 39  |fOmmu9srraftk.I9|
000003a0  67 5a 4f 38 57 52 51 67  59 31 75 4e 64 73 58 77  |gZO8WRQgY1uNdsXw|
000003b0  49 44 41 51 41 42 0a 2d  2d 2d 2d 2d 45 4e 44 20  |IDAQAB.-----END |
000003c0  50 55 42 4c 49 43 20 4b  45 59 2d 2d 2d 2d 2d 0a  |PUBLIC KEY-----.|
000003d0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```


Domain generation algorithm

When the primary C2 server is not available, Zloader reverts to a DGA. The DGA algorithm has changed in the latest version and no longer contains a different seed per botnet. Python code that replicates Zloader's new DGA algorithm is shown below.

```python
import time
from datetime import datetime, timedelta

def uint32(val):
    return val & 0xffffffff

def get_dga_time():
    now = datetime.now()
    ts = time.time()
    utc_offset = (datetime.fromtimestamp(ts) - datetime.utcfromtimestamp(ts)).total_seconds() / 3600
    midnight = now.replace(hour=0, minute=0, second=0, microsecond=0)
    midnight = midnight + timedelta(hours=utc_offset)
    return int(midnight.timestamp())

def generate_zloader_dga_domains():
    domains = []
    t = get_dga_time()
    for i in range(32): # number of domains to generate
        domain = ""
        for j in range(20): # domain name length
            v = uint32(ord('a') + (t % 25 ))
            t = uint32(t + v)
            t = (t >> 24) & ((t >> 24) ^ 0xFFFFFF00) | uint32(t << 8)
            domain += chr(v)
        domains.append(domain+".com")
    return domains
```


Code sample available on GitHub.

The code generates 32 domains per day by using the local system time at midnight (converted to UTC) as a seed. Each of the DGA domains have a length of 20 characters followed by the ".com" TLD.

Network communications

Zloader continues to use HTTP POST requests to communicate with its C2 server. However, the network encryption is now using 1,024-bit RSA with RC4 and the Zeus "visual encryption" algorithms. Zloader uses the custom Zeus BinStorage format where the first 128 bytes are the RSA encrypted RC4 key (32 random bytes) and, the remaining bytes are encrypted with the RC4 key and visual encryption as shown in Figure 4:

Figure 4. Zloader BinStorage object for a *hello* message (prior to encryption)

The Zeus BinStorage structure uses an ID integer value to represent the information stored, followed by the length and data. The BinStorage ID values in this example are shown in Table 1.

Value (Decimal)	Value (Hexadecimal)	Description

Value (Decimal)	Value (Hexadecimal)	Description
10002	0x2712	Botnet ID
10025	0x2729	Campaign ID
10001	0x2711	Bot ID
10003	0x2713	Malware version
10006	0x2716	Unknown flag (set to 0x1)

Table 1. Zloader BinStorage *hello* message fields

ThreatLabz has observed samples containing the following botnet IDs:

- Bing_Mod2
- Bing_Mod3
- Bing_Mod4
- Bing_Mod5

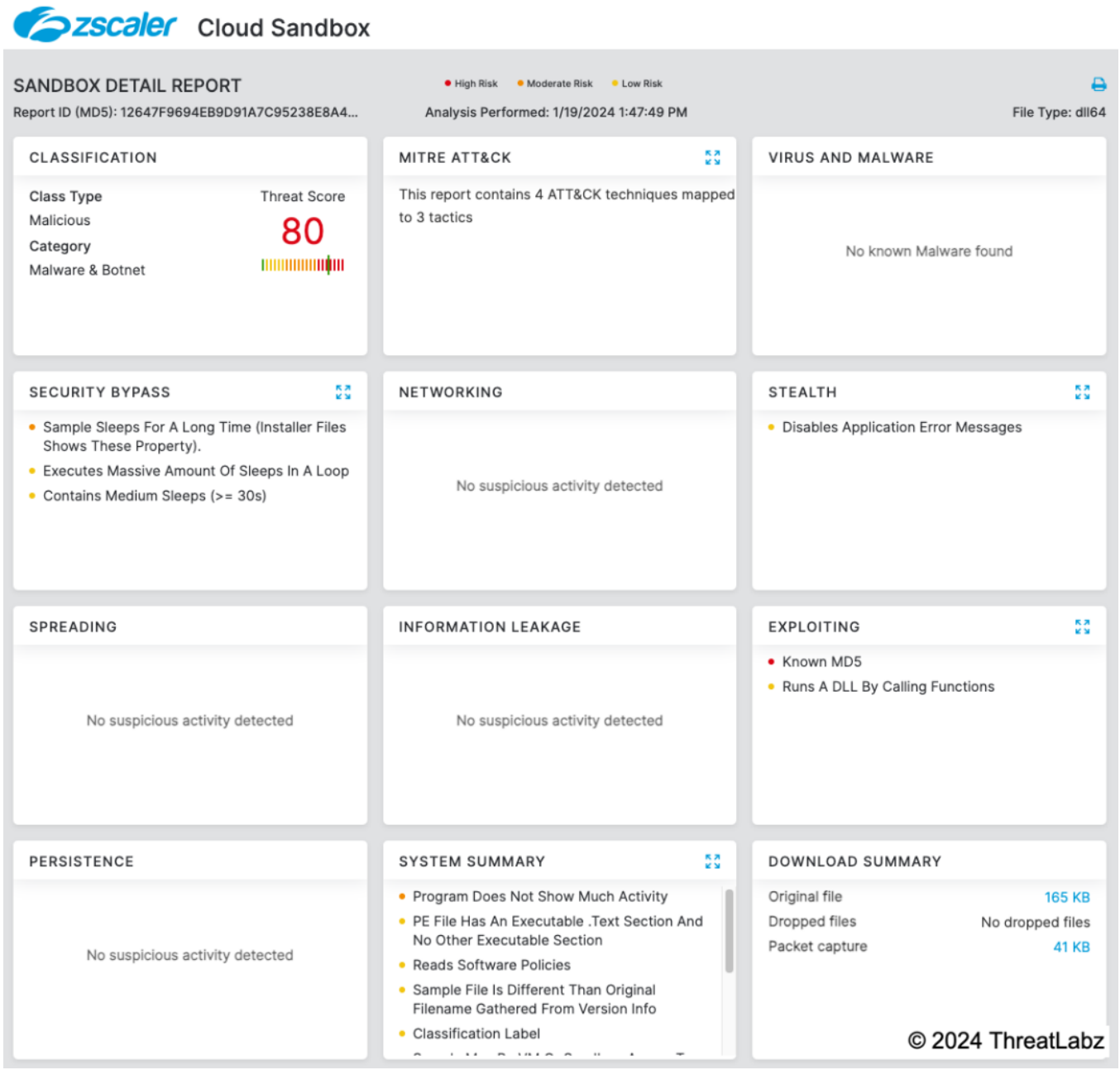
All of the campaign IDs have been set to the value **M1**.

Conclusion

Zloader was a significant threat for many years and its comeback will likely result in new ransomware attacks. The operational takedown temporarily stopped the activity, but not the threat group behind it. Returning after almost two years, Zloader has brought notable improvements to the loader module such as RSA encryption, an updated DGA, and enhanced obfuscation techniques, with more junk code, API import hashing, and string encryption to thwart malware analysis.

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

Zscaler Coverage



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
038487af6226adef21a29f3d31baf3c809140fcb408191da8bc457b6721e3a55	Zloader sample
16af920dd49010cf297b03a732749bb99cc34996f090cb1e4f16285f5b69ee7d	Zloader sample
25c8f98b79cf0bfc00221a33d714fac51490d840d13ab9ba4f6751a58d55c78d	Zloader sample
2cdb78330f90b9fb20b8fb1ef9179e2d9edfbbd144d522f541083b08f84cc456	Zloader sample
83deff18d50843ee70ca9bfa8d473521fd6af885a6c925b56f63391aad3ee0f3	Zloader sample
98dccaaa3d1efd240d201446373c6de09c06781c5c71d0f01f86b7192ec42eb2	Zloader sample
adbd0c7096a7373be82dd03df1aae61cb39e0a155c00bbb9c67abc01d48718aa	Zloader sample

SHA256	Description
b206695fb128857012fe280555a32bd389502a1b47c8974f4b405ab19921ac93	Zloader sample
b47e4b62b956730815518c691fcd16c48d352fca14c711a8403308de9b7c1378	Zloader sample
d92286543a9e04b70525b72885e2983381c6f3c68c5fc64ec1e9695567fb090d	Zloader sample
eb4b412b4fc58ce2f134cac7ec30bd5694a3093939d129935fe5c65f27ce9499	Zloader sample
f03b9dce7b701d874ba95293c9274782fceb85d55b276fd28a67b9e419114fdb	Zloader sample
f6d8306522f26544cd8f73c649e03cce0268466be27fe6cc45c67cc1a4bdc1b8	Zloader sample
fa4b2019d7bf5560b88ae9ab3b3deb96162037c2ed8b9e17ea008b0c97611616	Zloader sample
fbd60fffb5d161e051daa3e7d65c0ad5f589687e92e43329c5c4c950f58fbb75	Zloader sample

URL	Description
https://adslstickerhi[.]world	Zloader C2
https://adslstickerni[.]world	Zloader C2
https://dem.businessdeep[.]com	Zloader C2

Appendix

Tools

The code snippets in this blog have also been uploaded to our GitHub tools repository here.

Decoded strings

user32.dll	nbsp;
reg add HKCU\SOFTWARE\Microsoft\Windows\CurrentVersion\Run /f /t REG_SZ /v %s /d "%s"	wininet.dll
tr	br
h3	Local\
POST	gdiplus.dll
https://	*
ntdll.dll	ws2_32.dll
NtProtectVirtualMemory	NtGetContextThread
%s %s	psapi.dll
S-1-15	ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/
\"%s\"	samlib.dll
NtCreateThreadEx	regsvr32.exe /s \"%s\"
bcrypt.dll	netapi32.dll

strtoul	winsta.dll
NtReadVirtualMemory	Basic
version.dll	h2
h5	NtAllocateVirtualMemory
cabinet.dll	S:(ML;;NRNWNX;;;LW)
kernel32.dll	%s\tmp_%08x
aeiouy	div
{%08X-%04X-%04X-%08X%08X}	iphlpapi.dll
C:\Windows\System32\ntdll.dll	Connection: close
C:\Windows\System32\msiexec.exe	
wtsapi32.dll	NtCreateUserProcess
RtlUserThreadStart	%s
HTTP/1.1	ncrypt.dll
_aullrem	Software\Microsoft\Windows\CurrentVersion\Run
ole32.dll	.dll
bcdfghklmnpqrstvwxz	ftllib.dll
ThreadStart	MSIMG32.dll
JKLMNOPQRSTUVW$$$$$$XYZ[\]^_`abcdefghijklmnopq	h1
/	GET
urlmon.dll	Software\Microsoft\Windows NT\CurrentVersion
dxgi.dll	NtOpenSection
/post.php	advapi32.dll
secur32.dll	imagehlp.dll
%s_%s_%X	winscard.dll

References

[1] The Curious Case of an Unknown Trojan Targeting German-Speaking Users

[2] Dismantling ZLoader: How malicious ads led to disabled security tools and ransomware | Microsoft Security Blog

Thank you for reading

Was this post useful?

Get the latest Zscaler blog updates in your inbox

By submitting the form, you are agreeing to our privacy policy.