Zloader 2: The Silent Night

By Threat Research TeamThreat Research Team

Archived: 2026-04-05 15:36:22 UTC

In this study we are considering one of *Zeus* successors – *Zloader 2*. We'll show how it works and its code peculiarities. We'll present the result of our deep dive into the botnets and campaigns and show some interesting connections between Zloader and other malware families.

Introduction

Zloader 2 (also known as *Silent Night*) is a multifunctional modular banking malware, aimed at providing unauthorized access to online banking systems, payment systems and other financial-related services. In addition to these functions it's able to download and execute arbitrary files, steal files, inject arbitrary code to visited HTML pages and so on.

History

According to ZeusMuseum, first versions of *Zeus* were observed in 2006-2008. Later, in 2011 its source code leaked. As a result, new versions and variants appeared. One of the *Zeus* successors named *Zloader* appeared at the turn of 2016 and 2017. Finally, another successor named *Silent Night* appeared in 2019. It was for sale on the underground market.

The earliest version of this variant we found has a SHA256:
`384f3719ba4fbcf355cc206e27f3bfca94e7bf14dd928de62ab5f74de90df34a`
Timestamp 4 December 2019 and version number 1.0.2.0. In the middle of July 2021 the version 2.0.0.0 was spotted.

Microsoft recently announced a joint investigation of multiple security companies and information sharing and analysis centers (ISACs) with the aim to take down the *Zloader* botnet and took the whole case to court.

Although the original name of the malware likely was *Silent Night* and the ZeusMuseum calls it *Zloader 2* we are simply going to use the name *Zloader*.

Technical analysis

Modules and components

Zloader consists of different modules and components:

- **Downloader** – initial infector
- **Backdoor** – main module, exists in x86 and x64 versions
- **VNC module** (x86 and x64)

- **Web Injects** – received from C&C
- Additional libraries (**openssl**, **sqlite**, **zlib**, Mozilla libraries)

Backdoors, VNC modules and additional libraries have assigned module IDs that are used by other components to refer to them.

Distribution

Zloader was distributed using classic email spam. In 2021 the attackers abused *Google AdWords* to advertise sites with fake *Zoom* communication tool which actually installed *Zloader*. Another campaign in 2021 used fake pornsites, where users needed to download additional software to watch video. Downloaders are distributed in a packed form sometimes signed with a valid digital signature.

Map showing the distribution of infected systems:

Code peculiarities

Zloader code is very recognizable. First of all, it is diluted with functions which will never be called. Downloader module may contain functions from the Backdoor module and vice versa. In total, about a half of the code will never be called.

Second, simple x86 instructions like CMP, ADD and XOR are replaced with special functions. These functions contain a lot of useless code to complicate the analysis and they can call other “replacement” functions. To add more insult to the injury multiple “replacement” functions exist for a particular instruction. Also some constants are calculated in runtime using aforementioned “replacement” functions.

Strings are encrypted with a simple XOR algorithm.

Samples have very little imported functions. APIs are resolved in runtime by the hashes of their names.

As a result, more than a half of the file size is useless and serves as an obfuscation of simple operations.

Configuration

Both **Downloader** and **Backdoor** modules have built in configuration encrypted with RC4. The decryption key is stored in a plaintext and looks like `vcvslrpvwwfanquofupxt`. The structure of earlier versions (1.0.x, for example) differs from later versions (1.6.x and 1.8.x). Modern versions store the following information in config:

- Botnet name (`divader` on the picture below)
- Campaign name (`xls_s_2010`)
- List of hardcoded C&Cs
- RC4 key (`03d5ae30a0bd934a23b6a7f0756aa504`)

BinStorage

We have to briefly cover the *BinStorage* – the data format used by *Zloader* to communicate with C&Cs and to store various data: web injects, system information, stolen data and logs. *BinStorages* consists of the header and

records (also called fields). Main header stores information about the number of records, data size (in bytes) and their MD5. Records have their own small headers, containing `FieldID - DWORD` describing the meaning of the data.

Some FieldIDs are hardcoded. For example, in `FieldID=0x4E20` the last working C&C is stored. Other FieldIDs are derived from file paths (used to store stolen files).

Registry usage

Zloader modules (at least Downloaders and Backdoors) use a registry to store various data necessary for their work. The ROOT_KEY for this data is `HKEY_CURRENT_USER\Software\Microsoft\`

The most important and interesting data structure, stored by the Zloader in the registry is called `MAIN_STRUCT`. It's subkey in the `ROOT_KEY` and the value name is derived from the RC4 key found in the configuration. We suppose that bots from one actor use the same RC4 key, so they can easily find and read the `MAIN_STRUCT`.

`MAIN_STRUCT` is encrypted using RC4 with the key from the configuration. It stores:

- Registry paths to other storages, used by Zloader
- Files and directories path, used by Zloader
- Encryption key(s) to decrypt those storages

Files usage

Root path is `%APPDATA%`. Zloader creates directories with random names inside it to store modules, stolen data and logs. These paths are stored into the `MAIN_STRUCT`.

Networking

As was mentioned before, communication between the bot and C&C is done using BinStorages. Depending on the actual type of the message, field list may be changed, but there are 5 constant fields sent to C&C:

- Some DWORD from the Configuration
- Botnet name from the Configuration
- BotID, derived from the system information
- Debug flag from the Configuration
- 16 random bytes

Requests are encrypted using the RC4 key from the Configuration. C&C responses are signed with RSA.

PING request

This request is used to check if C&C is alive. Response contains only random bytes sent by a bot.

DOWNLOAD MODULE request

This request is used to download modules by their ID from the C&C. The response is not in a BinStorage form!

GET CONFIG request

Used to receive configuration updates: new C&Cs, WebInjects, tasks for downloading etc.

C&Cs and DGA

As was shown before, built in configuration has a list of hardcoded C&Cs. Actually, these lists have not changed for years. To bypass blocking of these hardcoded C&Cs, Zloader uses DGA – Domain Generation Algorithm. In the Zloader, DGA produces 32 domains, based on the current date and RC4 key from the configuration.

There is a 3rd type of C&Cs – received in the response from the server. They're stored into the Registry.

Downloader module

Analysis based on version 1.6.28.0, `44ede6e1b9be1c013f13d82645f7a9cff7d92b267778f19b46aa5c1f7fa3c10b`

Function of Downloader is to download, install and run the next module – the Backdoor.

Main function

Just after the start of the Downloader module, junk code is started. It consists of many junk functions, which forms a kind of a "network". In the image below there is a call graph from just a single junk function. These functions also trying to read, write and delete some `*.txt` files `%TEMP%`. The purpose of this is to delay the execution of the payload and, We suppose, to complicate the emulation, debugging and analysis.

The second and the last task of the Main function is to start `msiexec.exe` and perform the PE injection of the code into it. Injected data consists of two buffers: the big one, where the Downloader is stored in the encrypted form and the small one (0x42 bytes) with decryption code. Just after the injection Downloader terminates himself.

Injected code

Control flow passed to the small buffer, which decrypts the Downloader in the address space of msiexec.exe After the decryption, Downloader begins to execute its main task.

First of all, the injected code tries to read `MAIN_STRUCT` from the registry. If this fails, it thinks it was not installed on this system and the installation process begins: `MAIN_STRUCT` is created, Downloader module is copied into `%APPDATA%` and added to the autorun key `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` with random value name.

In any case, the Backdoor module is requested from the disk or from the network and executed.

Backdoor module

Analysis based on version 1.6.28.0, `c7441a27727069ce11f8d54676f8397e85301b4d65d4d722c6b239a495fd0282`

There are actually two Backdoor modules: for 32-bit systems (moduleID `0x3EE`) and for 64-bit systems (moduleID `0x3E9`). Downloader always requests a 32-bit Backdoor.

Backdoors are much more complicated than Downloaders. If we compare the size of our samples (after unpacking), Backdoor will be twice bigger.

Key Backdoor abilities:

- Starting VNC module
- Injecting WebInjects into the pages visited using browsers
- Downloading and execute arbitrary file
- Keylogging
- Making screenshots
- Stealing files and sending to C&C

Stealing files

The largest group of software from which Zloader steal files is crypto wallets:

- Electrum
- Ethereum
- Exodus cryptowallet
- Zcash
- Bitcoin-Qt
- Etc.

It also steals data from browsers: cookies from Chrome, Firefox and IE; saved logins from Chrome. And, finally, it is able to steal accounts information from Microsoft Outlook.

Hooking

To achieve his goals, Zloader performs WinAPI hooking. In order to perform it, Backdoor module enumerates processes and injects itself into the following ones:

- explorer.exe
- msiexec.exe
- iexplore.exe
- firefox.exe
- chrome.exe
- msedge.exe

64-bit version of Backdoor is injected into 64-bit processes, 32-bit version – into 32-bit processes.

Injected code hooks the following WinAPI functions:

- NtCreateUserProcess
- NtCreateThread
- ZwDeviceIoControlFile
- TranslateMessage

- CertGetCertificateChain
- CertVerifyCertificateChainPolicy

Hooks might be divided in 3 groups, depending on the purpose:

1. `NtCreateUserProcess` and `NtCreateThread` are hooked to inject a Backdoor module to newly created threads and processes.
2. `ZwDeviceIoControlFile`, `CertGetCertificateChain` and `CertVerifyCertificateChainPolicy` are hooked to support WebInjection mechanism
3. `TranslateMessage` is hooked to log the keys pressed and to create screenshots

Web Injecting

First of all, browsers must have a Backdoor module injected. At this moment, there are multiple instances of Backdoor Modules running in the system: one, started by Downloader which is “Main Instance” and others, running in browsers. Main Instance starts Man-in-the-browser proxy, other modules hooks `ZwDeviceIoControlFile` and cert-related WinAPIs (see above). Proxy port number is stored in the BinStorage structure into the Registry, so it is synchronized between Backdoor instances.

Hooked ZwDeviceIoControlFile function is waiting for `IOCTL_AFD_CONNECT` or `IOCTL_AFD_SUPER_CONNECT` and routing connections to the proxy. Hooked cert-related functions inform browsers what everything is good with certificates.

Botnets, Campaigns and their activity

Most active botnets and campaigns use RC4 key `03d5ae30a0bd934a23b6a7f0756aa504` and we’ll focus on them in our analysis. Samples with the aforementioned key have versions 1.x, usually 1.6.28, but some have even 1.0.x.

Botnet and Campaign names

Among botnet names it is worth mentioning the following groups:

1. `DLLobnova`, `AktualizacjaDLL`, `googleaktualizacija`, `googleaktualizacija1`, `obnovlenie19`, `vasja`, `ivan`
2. `9092zi`, `9092ti`, `9092ca`, `9092us`, `909222`, `9092ge`

The first one contains transliterated Slavic words and names (`vasja`, `ivan`), maybe with errors. It sheds light on the origins of bad guys – they are definitely Slavs.

Samples with botnet names from the second group were first observed in November 2021 and we found 6 botnet names from this group in the next two months. Letters after numbers, like `ca` and `us` might be country codes.

We see the same picture with campaign names: quite a big amount of Slavic words and the same `9092*` group.

WebInjects

We analyzed webinjects and can confirm that they are targeting financial companies: banks, brokerage firms, insurance companies, payment services, cryptocurrency-related services etc.

Injected code is usually small: from dozens of bytes up to 20 kb. To perform its tasks, it loads JavaScript code from external domains, controlled by bad guys. Analysis of these domains allowed us to find connections between Zloader operators and other cybercrime gangs.

Download tasks

Zloader is able to download and execute arbitrary files by the commands from his C&Cs, but for a long time we haven't seen these commands at all. Things changed on 24 November 2021, when botnet `9092ca` received a command to download and execute the file from `teamworks455[.]com`. This domain was mentioned in [6].

Another two download tasks contained `braves[.]fun` and `endoftheendi[.]com`

Connections

During our tracking we have noticed links to other malware families we originally thought were unrelated.

Raccoon Stealer

Two out of three download tasks contained links to Raccoon Stealer. Downloaded samples have the following sha256 hashes:

- `5da3db74eee74412c1290393a0a0487c63b2c022e57aebcd632f0c3caf23d8bc`
- `5b731854c58c2c1316633e570c9ec82474347e64b07ace48017d0be2b6331eed`

Both of them have the same Raccoon configuration with Telegram channel `kumchakl1`.

Moreover, Raccoon was mentioned in [6] before we received commands from C&Cs with links to Raccoon. We are lost in conjecture why Zloader operators used Raccoon Stealer? You can read our dive into Racoon stealer here.

Ursnif

Ursnif, also known as Gozi and ISFB is another banking malware family with similar functions.

Digital Signatures

It was quite a big surprise when we found Zloader samples and Ursnif samples signed with the same digital signature!

As an example, consider a signature:
Issuer `BABJNCXZHQCJUVWAJJ`
Thumbprint `46C79BD6482E287647B1D6700176A5F6F5AC6D57`.

Zloader sample signed with it has a SHA256 hash:
`2a9ff0a0e962d28c488af9767770d48b9128b19ee43e8df392efb2f1c5a696f`.

Signed Ursnif sample has a SHA256 hash:
`54e6e6b23dec0432da2b36713a206169468f4f9d7691ccf449d7d946617eca45`

It is not the only digital signature, shared among Ursnif and Zloader samples.

Infrastructure

As we mentioned before, the first observed download command contained a link to `teamworks455[.]com`. We checked the TLS certificate for this site and realized that it was for another site – `dotxvcnjlvdajkwerwoh[.]com`. We saw this hostname on 11 November 2021 in Ursnif webinjects, it was used to receive stolen data.

Another example – `aerulonoured[.]su` – host used by Zloader to receive stolen data at least from August 2021. It also appeared in Ursnif webinjects in November 2021.

Third example – `qyfurihpsbhbuvitilgw[.]com` which was found in Zeus configuration update, received from C&C on 20 October 2021. It must be added to a C&C list and then used by Zloader bots. The same domain was found in Ursnif webinjects on 1 November 2021

And, finally, 4th example – `etjmejjcxjtwweitluuw[.]com` This domain was generated using DGA from key `03d5ae30a0bd934a23b6a7f0756aa504` and date – 22 September 2021. We have very strong evidence that it was active on that date as a Zloader C&C. The same host was found in Ursnif WebInjects on 1 November 2021

Conclusion

We are proud we could be part of the investigation as we continue our mission to make the world a safer place for everybody. We hope for a successful takedown of the *Zloader* botnet and prosecution of people who created and operated it.

Source: https://decoded.avast.io/vladimirmartyanov/zloader-the-silent-night/