Snake Keylogger's Many Skins: Analysing Code Reuse Among Infostealers

By Patrick Schläpfer

Published: 2021-06-28 · Archived: 2026-04-05 21:02:59 UTC

Snake is a modular .NET keylogger and credential stealer first spotted in late November 2020. Since then, we've seen campaigns spreading this malware almost daily. Snake's name was derived from strings found in its log files and string obfuscation code. Using the malware's builder, a threat actor can select and configure desired features then generate new payloads. For this reason, the capabilities of samples found in the wild can vary. This article describes Snake's capabilities, its infection chain and code similarities with four other commodity keyloggers.

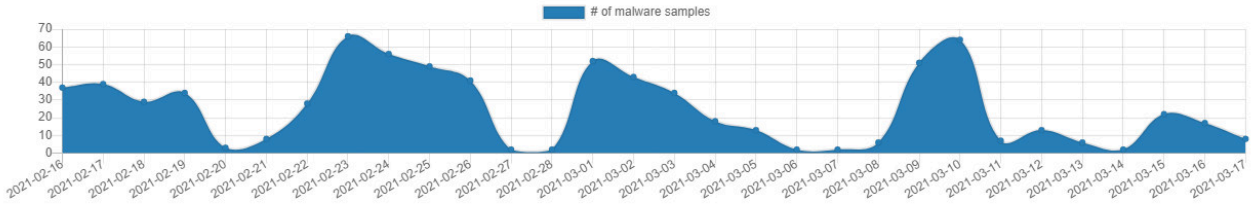

Figure 1 – Publicly reported Snake keylogger detections over time.

Infection Chain

Campaigns delivering Snake in 2021 used malicious spam to distribute the malware, either in RTF or archive attachments.

Delivery

The first type of downloader we've seen used to deploy Snake are RTF documents containing the well-known Microsoft Office Equation Editor exploit (CVE-2017-11882). The documents were renamed with .DOC file extensions and attached to emails themed as legitimate business communications. If the recipient runs a vulnerable version of Microsoft Office, the exploit downloads an executable from a remote server and executes it. This file is a packed version of Snake keylogger.

Figure 2 – A malicious RTF email attachment exploiting CVE-2017-11882 isolated by HP Wolf Security.

With the second method, attackers send spam with archive files attachments containing packed Snake executables. When the recipient opens the archive file, it contains a packed copy of Snake, requiring the user to double click the executable to run it. We found Snake being distributed in IMG, ZIP, TAR, Z, GZ, ISO, CAB, 7z and RAR attachments.

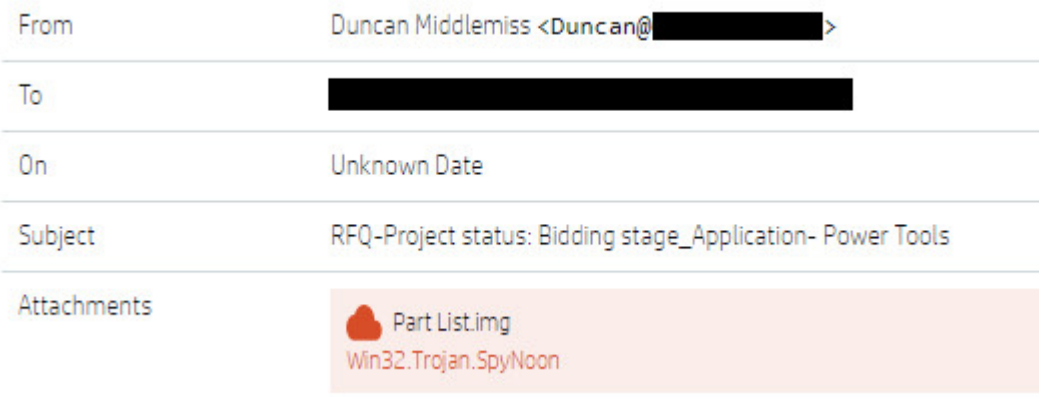

Figure 3 – A malicious IMG email attachment delivering Snake Keylogger isolated by HP Wolf Security.

Unpacking and execution

To reduce chances of detection by endpoint security tools, the Snake samples we analysed were packed. Shortly after execution, the malware unpacks itself by following the process in Figure 4.

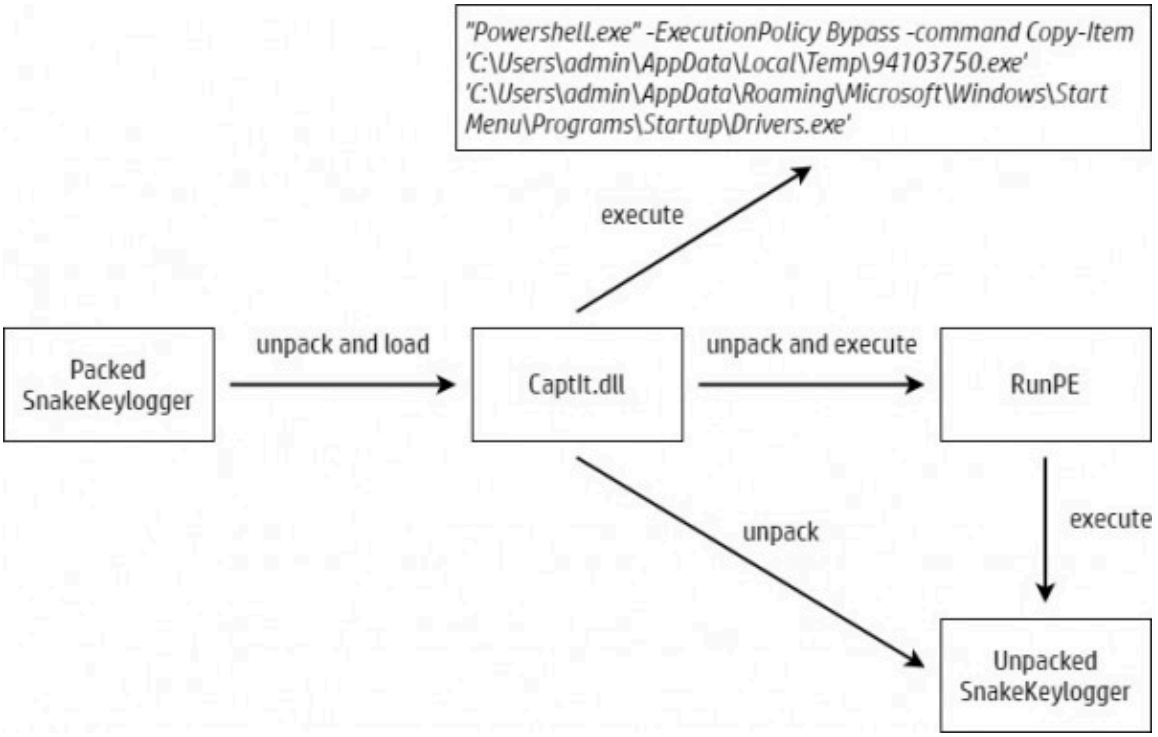

Figure 4 – Snake's unpacking and execution process.

First, the malware decrypts and loads an encrypted file in the resources section of the .NET file using DES, a common block cipher.

```
78
79          // Token: 0x04000018 RID: 24
80          public static byte[] Primer = CaptureConfig.PerSec(Resources.CaptIt, "西فنى弗AnSci");
81
82          // Token: 0x04000019 RID: 25
83          public static Type Adope = CaptureConfig.ValueExport.Load(ScreenCapture.GDI32.Primer).GetExportedTypes()[0];
84
```

Figure 5 – Call to decryption method with .NET resource and key as argument.

The key used to decrypt the file are the first 8 bytes of the SHA256 hash of a string in the source code. Since the UTF-16 encoded so you need to choose the correct encoding when calculating the hash to decrypt the file. Figure 5 shows the key (a string of Unicode characters) being passed as the second argument to the *PerSec* function. Since DES is a block cipher, a mode of operation needs to be specified so that the decryption method knows how to process multiple blocks of ciphertext. Snake's packer uses electronic codebook (ECB) as its mode, meaning no initialisation vector is required and therefore only the ciphertext and the key are needed.

```
158
159         // Token: 0x06000006 RID: 6 RVA: 0x000023C0 File Offset: 0x000005C0
160         public static byte[] PerSec(byte[] B, string ikey)
161         {
162             DESCryptoServiceProvider descryptoServiceProvider = new DESCryptoServiceProvider();
163             SHA256CryptoServiceProvider sha256CryptoServiceProvider = new SHA256CryptoServiceProvider();
164             byte[] array = new byte[8];
165             byte[] sourceArray = sha256CryptoServiceProvider.ComputeHash(Encoding.BigEndianUnicode.GetBytes(ikey));
166             Array.Copy(sourceArray, 0, array, 0, 8);
167             descryptoServiceProvider.Key = array;
168             descryptoServiceProvider.Mode = CipherMode.ECB;
169             return descryptoServiceProvider.CreateDecryptor().TransformFinalBlock(B, 0, B.Length);
170         }
```

Figure 6 – DES decryption method.

The decrypted file is a dynamic link library (DLL) called CaptIt.dll, which is loaded. This file contains two encrypted files as resources, which are unpacked and executed at runtime. The first is the Snake keylogger payload, and the other is an executable file that is used to run and hide the malware's presence. Before decrypting these files, CaptIt.dll copies the packed executable to the user's start-up folder (*C:\Users\<Username>\AppData\Roaming\Microsoft\Windows\Start Menu\Programs\Startup*) so that it runs whenever the device is restarted.

```
44    public static byte[] D3LLCODE = UpdateForm.AESDecrypt(Resources.OnBBQxxhGLnvZEpikDixiTCvwEZrkkQvfTpGLnhBZfOpCL, "艾勒<bpS私0V3J<+s");
45
46    // Token: 0x04000002 RID: 2
47    public static string UpdateVersionsc = Conversions.ToString
        ("AZJCJfpyUsnAfJiyTLOifhLwQLhZwGQnrnOfJOnTpCiTDfBnyfynAAJxwhxnBJTLsQ.vrQBCQEUkZhMBwkZOZJQvLwhJxnADLpQChAphAJZfsMfxEiQLivpkxrTwsUwEkZMph");
48
49    // Token: 0x04000003 RID: 3
50    public static string DataAccepter = SET_AutoSave.RijndaelDecrypt("kWJGgxWm8LEUQ58EH1EBeHypuSVB1qqmDtHQmBn2JutUkLUlsN2F8tueZogKhskmn2Nv/w2NAOl/
        AfnYGRedAM+G8pKtdZ7ykRVJS69KmuU=", "W艾艾ㄣXO马ぎLMㄇQQ");
```

Figure 7 – DLL decryption and string decryption calls.

This time, AES in ECB mode is used to decrypt another DLL file in CapIt.dll. Again, a SHA256 hash of a string is used as the key.

```
45        public static byte[] AESDecrypt(byte[] byt, string ikey)
46        {
47            AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider();
48            SHA256CryptoServiceProvider sha256CryptoServiceProvider = new SHA256CryptoServiceProvider();
49            byte[] key = sha256CryptoServiceProvider.ComputeHash(Encoding.BigEndianUnicode.GetBytes(ikey));
50            aesCryptoServiceProvider.Key = key;
51            aesCryptoServiceProvider.Mode = CipherMode.ECB;
52            return aesCryptoServiceProvider.CreateDecryptor().TransformFinalBlock(byt, 0, byt.Length);
53        }
```

Figure 8 – AES decryption method.

In addition to decrypting the DLL, the function decrypts a string that corresponds to the method name of the DLL that has just been unpacked. This method, which is started in a new thread, expects two arguments: a file path and payload name (Figure 9).

```
18    // Token: 0x06000004 RID: 4 RVA: 0x00002F6C File Offset: 0x0000116C
19    public static object Datagetteraccept()                    Content of DLL
20    {
21        return Thread.GetDomain().Load(QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.D3LLCODE).GetType
            (QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.UpdateVersionsc).GetMethod(QsJAksvOJQZGMrkQGUrJCZfDxJspOiApOTEDEDQQQBBEDh.DataAccepter).Invoke
            (null, new object[]
22        {                                  Absolute class
23            SShotCaptured.locatedplace(),  name inside DLL
24            UpdateForm.jjjjjjddd           Arguments passed to
25        });                                the method
```

Figure 9 – DLL load and execution.

The second encrypted resource in CapIt.dll is decrypted in the same way as CaptIt.dll, using DES in ECB mode with a key passed to the method as an argument.

```
142       // Token: 0x04000093 RID: 147
143       public static byte[] jjjjjjddd = SET_ImageViewer.DESDecrypt(Resources.ZUZUBase, "bJyEGVlHQz");
144
```

Figure 10 – Call to DES decryption method with resource and key as argument.

The Snake payload is hidden inside a legitimate process using RunPE, a well-known process hollowing technique. First, the file path passed as the first argument is used to start a new process in suspended mode. This is followed by calls to Windows APIs GetThreadContext, ZwUnmapViewOfSection, VirtualAllocEx, WriteProcessMemory,

SetThreadContext and ResumeThread. These replace the context of the newly-created process with the malware and then launches it. We found a C# implementation of this function on GitHub. At this point, the Snake payload is unpacked and starts running the modules configured by the attacker.

Code Obfuscation

Snake is written in .NET so instead of being compiled into native code, it is compiled into Microsoft Intermediate Language with meta data about the project. It's possible to recover a decompiled version of the source code that has almost the same structure as the original code using tools such as dnSpy, which significantly simplifying analysis. However, many threat actors who use .NET malware try to prevent decompilation by obfuscating their code. In March 2021, we found Snake samples reported to MalwareBazaar were obfuscated using five tools:

- ConfuserEx / Beds Protector
- DeepSea 4.1
- Babel .NET
- NET
- Obfuscar

These obfuscators make it difficult to analyse the source code by renaming symbols, complicating the code's control flow and encrypting strings and resources. Fortunately, there are often counter-techniques to reverse these obfuscations, such as those in the de4dot and DeObfuscar-Static projects.

Snake's Features

Because Snake is modular, the behaviour of samples in the wild varies depending on what features are enabled when the malware is built. It is highly likely that functionality is configured and created using a builder tool. Instead of adding only the selected functionalities to the payload, the builder appears to integrate all features and simply activates the ones selected by the operator. Figure 11 shows Snake's features mapped to MITRE ATT&CK techniques.

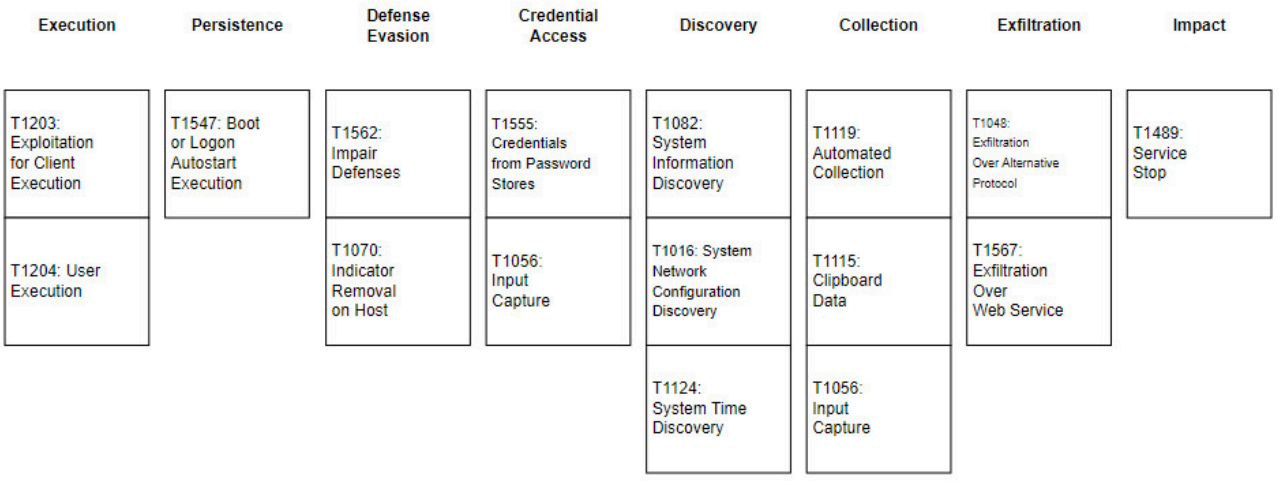

Figure 11 – MITRE ATT&CK techniques that Snake can use.

Unsurprisingly, most of Snake keylogger's features fall into the tactic categories of Credential Access, Collection and Exfiltration. Here are some descriptions of Snake's features:

Persistence

As described in the unpacking and execution section, the malware copies itself into the start-up folder to be activated again after a reboot of the device. As a further persistence functionality, Snake keylogger has the option of creating a run registry key, which has the same effect and launches the malware when the computer is started.

Defense Evasion

Snake uses two defense evasion techniques. The first is a function that terminates a list of security tool processes. These include anti-virus processes and tools that are commonly used to analyse malware. In the case of OllyDbg, the process name was misspelled meaning Snake would would fail to kill the process. Moreover, modern anti-virus programs will not be terminated by this technique because they will just restart themselves, or use a kernel driver to prevent the malware from opening a handle to their process without the right permissions. The second technique is a self-termination function where Snake removes itself from the infected system and terminates its own process.

Credential Access

Most of Snake's functionality relates to collecting sensitive information from infected computers, such as credentials and configurations. Snake parses login credentials from web browser databases, email clients, Discord and Pidgin chat clients and stored passwords for WiFi networks. Snake can extract credentials from Google Chrome, Mozilla Firefox and Microsoft Edge as well as several other less well-known web browsers. It can also steal credentials from a variety of email clients, including Microsoft Outlook, Mozilla Thunderbird, Foxmail and Postbox. A full list of all credential stealing modules can be found in the appendix.

Discovery

Before harvesting credentials, Snake runs a series of host and network checks to determine if it is running in a sandbox and to orientate the attacker within the victim's environment. The host discovery function checks the operating system version, the computer name, hard disk size, time and date configuration as well as installed RAM. One sample we analysed needed the host to have more than 3.75 GB of RAM and its hostname not match a list of blocklisted names to pass the checks. Further checks are executed by the network discovery module which first resolves the client's public IP address using the online service, *checkip.dyndns[.]org*. If the public IP matches a blocklist of addresses used by malware analysis services, Snake does not run any credential stealing. Using the public IP address, Snake resolves an approximate geolocation using the *freegeoip[.]app* service. Those domains with Snake's User-Agent (see appendix) can be used to detect its presence.

Collection

Snake also collects configuration files that can contain sensitive information. One program targeted by Snake is FileZilla. FileZilla can be used as FTP client and therefore saves configurations of previously accessed servers. Snake also looks for the infected system's Windows product key. Another feature is Snake's clipboard stealer,

which differs from its other stealing functionality because it runs periodically rather than once at first execution. This module saves the contents of the clipboard, which may contain sensitive information such as passwords that are temporarily stored in the clipboard when copied from a password manager. Snake also periodically takes screenshots of the infected system. The images are stored in a folder called “Snake keylogger” in the user’s documents folder until they are sent to the attacker. Afterwards, the images are deleted. Snake’s keylogger function runs permanently in the background by calling the SetWindowsHookExA API, thereby adding itself to the keyboard hook chain. Each time a key is pressed, a callback function is called, which saves the input and passes the call to the next link in the hook chain. Collected input is periodically sent back to the attacker using one of multiple available exfiltration functionalities. Interestingly, the keylogging module was not activated in the Snake keylogger samples found since the end of February 2020. Whether this was a mistake on the part of the attacker or a deliberate choice is not clear.

Exfiltration

Snake has several configurable data exfiltration channels. These include the option of uploading the data to a server using the FTP protocol, sending the data as an attachment via email, or publishing it in a Telegram channel via a POST request. The attacker decides which of these exfiltration techniques to choose when configuring the malware using the builder. Therefore, the necessary credentials, configurations and tokens, are stored directly in the malware. In the samples we observed, attackers preferred to exfiltrate data via Telegram or email.

Impact

In addition to the function that terminates security programs to bypass endpoint defenses, the malware also has a function that terminates the Chrome web browser. Since terminating a web browser is not done for prevention or detection reasons, there must be another reason. One possibility is that Snake can only access credentials stored in Chrome if it isn’t running. Another keylogger called Matiex also had this flaw, raising the suspicion that the two families share code.

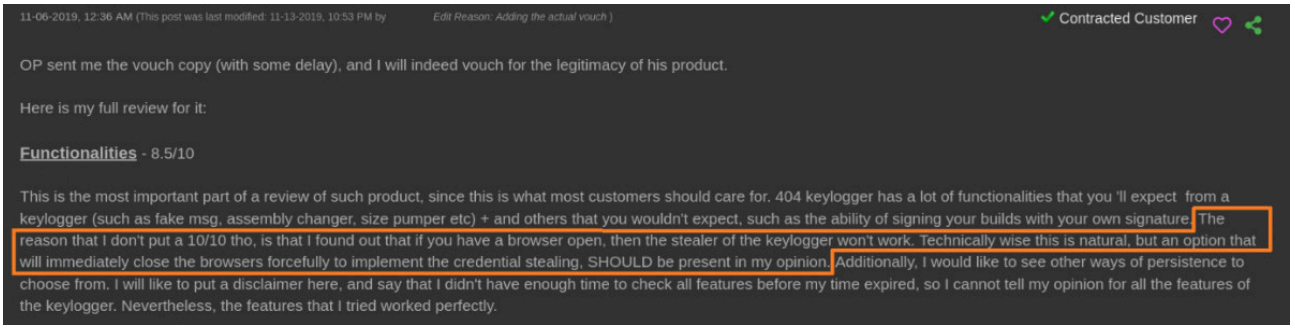

Figure 12 – Excerpt from a review of Matiex from a hacking forum.

Snake’s Link to Matiex Keylogger

In November 2020, a malware researcher commented about possible code links between Snake and Matiex. According to adverts on hacking forums, Matiex’s developer sold a keylogger builder used to configure and generate malware clients using a subscription model that is common for malware-as-a-service operations.

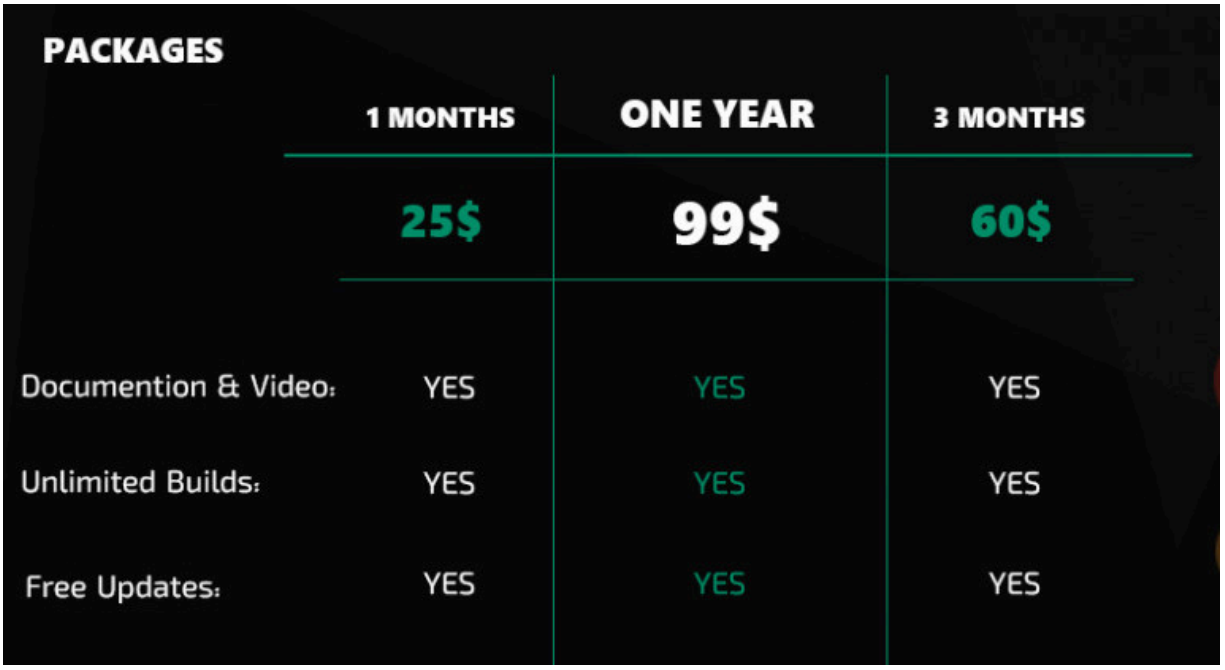

Figure 13 – Matiex's subscription model advertised on a hacking forum.

Comparing the source code of the two malware families uncovered many similarities. Both families have functions to terminate a list of the same anti-virus and security tool processes.

Figure 14 – Comparison of process termination functions.

Furthermore, both have procedures to uninstall themselves, using the same command.

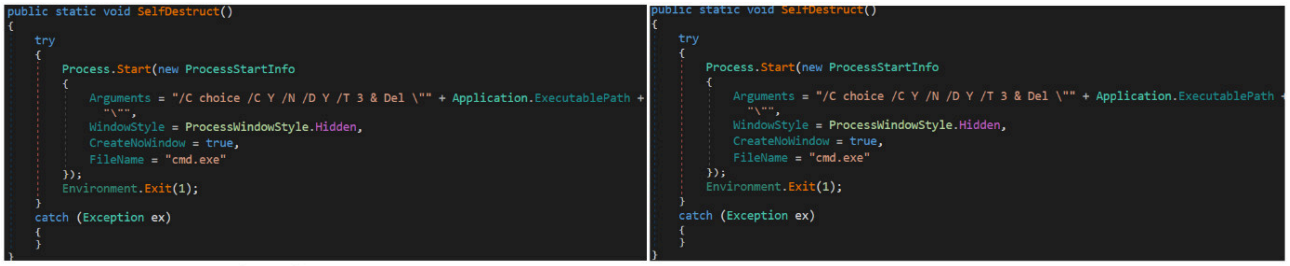

Figure 15 – Comparison of self-destruction functions.

Their functions to upload recorded keystrokes via POST requests are also identical.

Figure 16 – Comparison of keystroke exfiltration functions.

Both families have the option to exfiltrate data to a Telegram channel. The functions are identical, but for their names.

Figure 17 – Comparison of Telegram exfiltration functions.

The only difference between the families' GeoIP functions are the names.

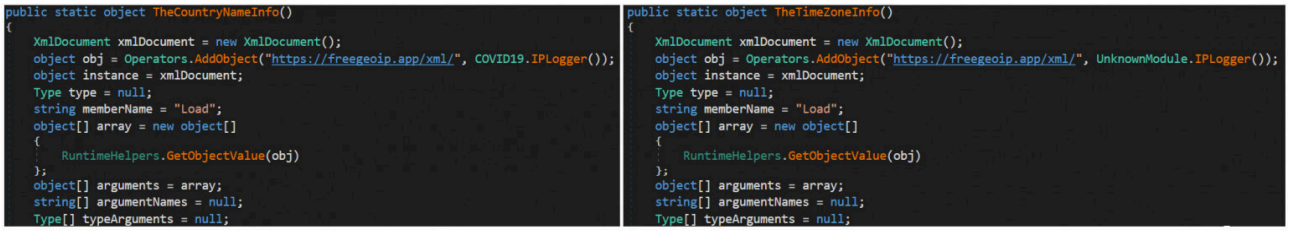

Figure 18 – Comparison of GeoIP orientation functions.

The logging functions reveals the names of the keyloggers. But the code is nearly identical, including variable names.

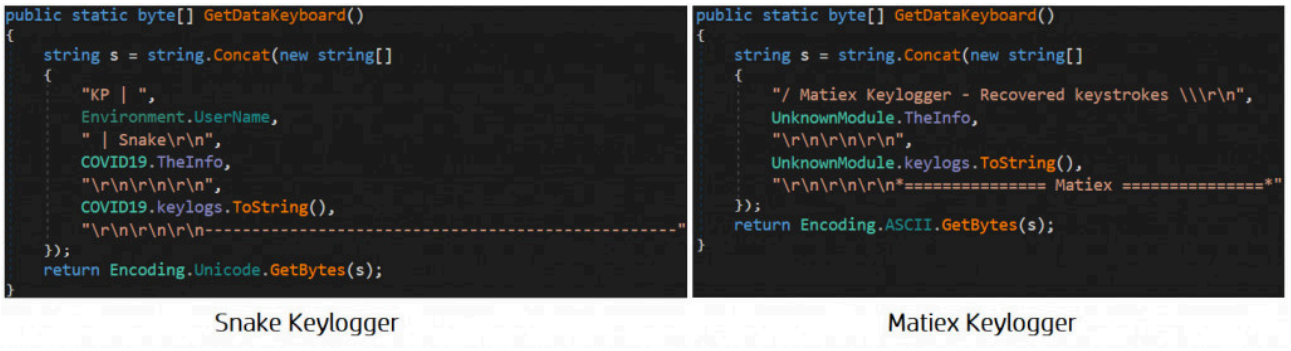

Figure 19 – Comparison of logging functions.

TheInfo is a variable that contains information about the infected system. Matiex collects far more information than Snake, which is then transmitted to the attacker. Snake, however, has an *AntiBot* function that terminates the malware if the infected system has a blocklisted IP address or hostname.

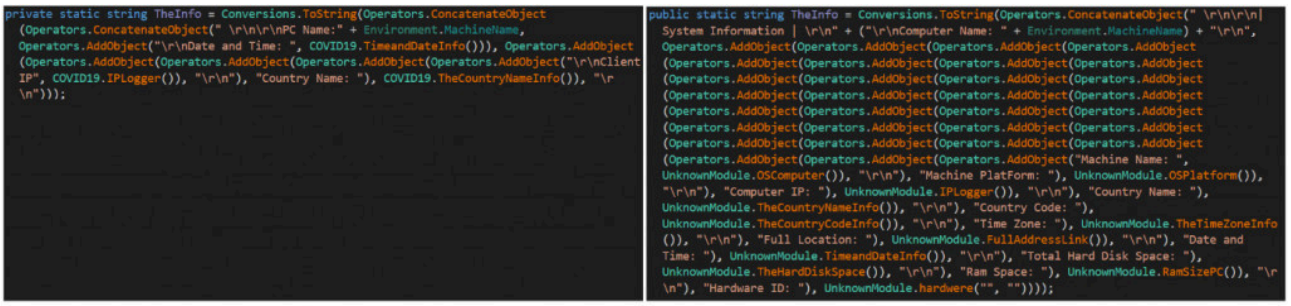

Figure 20 – Comparison of variables showing collected information.

Many of the variable names used between the families are similar. For example, the exfiltration channel is defined analogously via a string variable in both families. Furthermore, the keylogger class is named identically and both have a *HSHChecker* variable with the same initialisation value.

Figure 21 – Similar initialised variables between Snake and Matiex.

Based on the code similarities shown and the corresponding malware analysis, we think it is likely that Matiex and Snake share the same code base and that Snake is derived from the former. The families do have a few minor differences between them:

- Matiex does not check for a list of excluded IP addresses, hostnames or the amount of RAM installed.
- Matiex can make audio recordings.
- Matiex can exfiltrate data over Discord, an instant messaging service.

The question is when and how was Matiex's source code first reused in Snake? In February 2021, Matiex's source code was offered for sale in an underground forum. One possibility is that someone bought the source code of Matiex and rebranded it into Snake. However, our first sighting of Snake was months before the source code was put up for sale, so Snake's developers must have obtained Matiex's source code by another means.

Matiex's Link to Other Keyloggers

During this research, we found other .NET keyloggers advertised as a service, similar to Matiex. Analysing the code of the 404, Cheetah and Phoenix keyloggers revealed they were also very similar to Matiex. Figure 22 shows when each family was offered for sale as a service and when its source code was offered for sale.

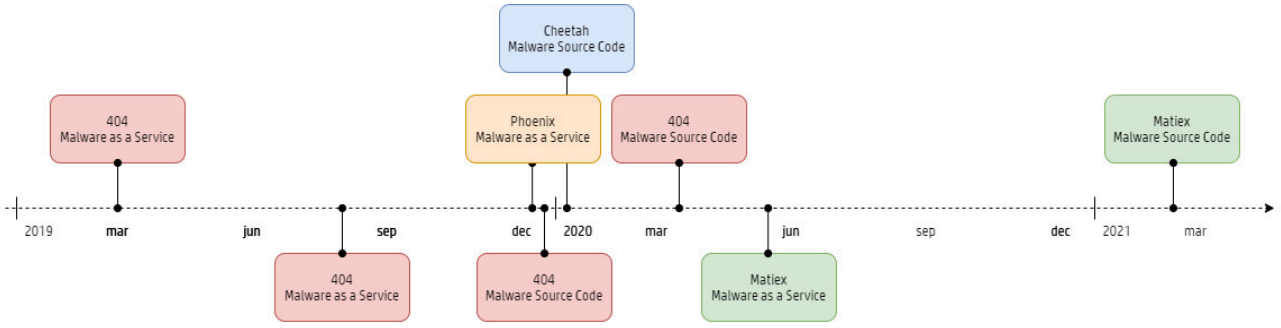

Figure 22 – Timeline of sales activity across hacking forums.

Looking at the code similarity of the different keyloggers, it struck us how similar their code is. 404 and Phoenix keylogger use a slightly different function to record keystrokes and can exfiltrate data using the Pastebin API, but are otherwise extremely similar to the other families. Cheetah uses the same keylogging functions as Snake and

Matiex, but has fewer modules for extracting passwords. The data is exfiltrated in the same way as Snake via Telegram, FTP or SMTP.

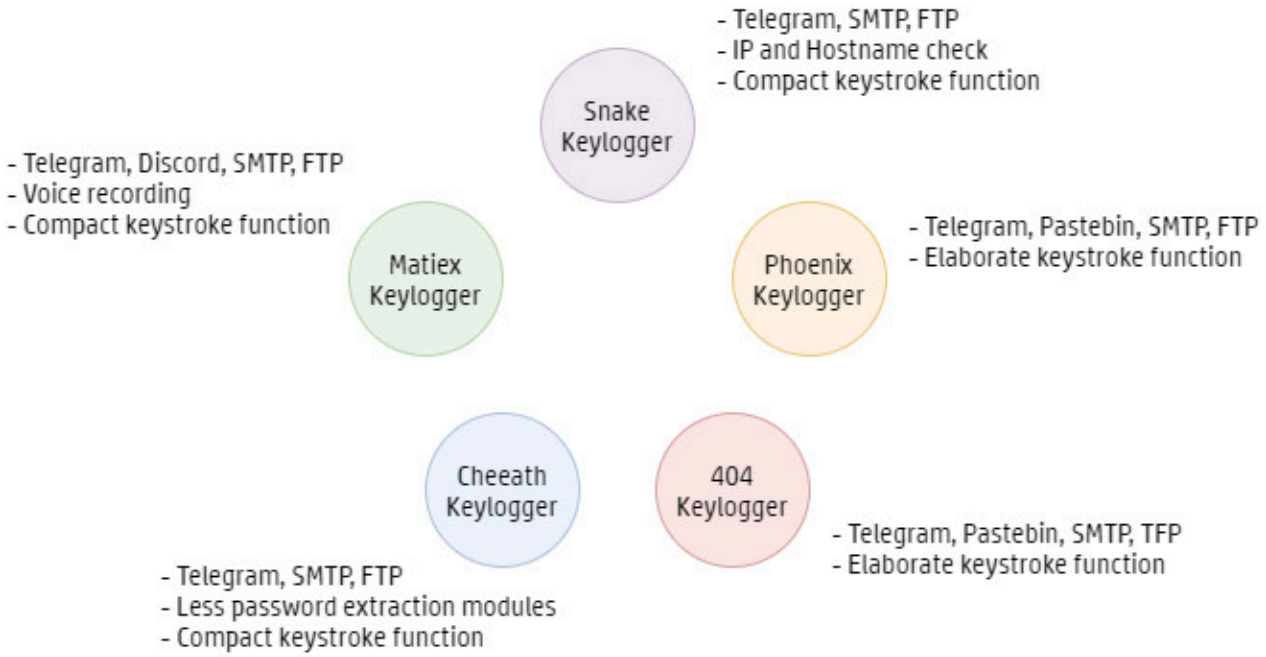

Figure 23 – An overview of the capabilities of keyloggers linked to Snake.

Conclusion

Analysing Snake reveals that it is indeed a comprehensive keylogger and data stealer. Based on our analysis, we think five keylogger families active in the last two years are likely derived from the same code base. However, this is not an exhaustive list so it is possible that other keyloggers with similar code are also in circulation. This "remix" behaviour of opportunistically copying source code from established malware families demonstrates how easy it is for cybercriminals to create their own malware-as-service businesses – and the importance for enterprise defenses to stay ahead of malware developers.

Indicators of Compromise

Snake's Credential Stealing Modules:

Outlook	Chrome	Cent	Yandex	FireFox
Foxmail	Coowon	xVast	Nichrome	WaterFox
Kinzaa	CocCoc	Chedot	Amigo	Thunderbird
Sputnik	Uran	Superbird	Kometa	SeaMonkey
Falkon	QQ	360_English	Xpom	IceDragon
SalamWeb	orbitum	360_China	Elements	CyberFox

CoolNovo	Slimjet	Comodo	Microsoft	Slim
QIPSurf	Iridium	Brave	Opera	IceCat
BlackHawk	Vivaldi	Torch	FileZilla	PostBox
7Star	Iron	UC	Pidgin	PaleMoon
Sleipnir	Chromium	Blisk	Liebao	TheWiFi_Orginal
Citrio	Ghost	Epic	avast	WindowsProductKey_Orginal
Chrome_Canary	Discord			

Excluded IP Addresses:

- 1.254.1[.]255
- 34.122.197[.]93
- 89.187.165[.]47
- 89.208.29[.]133
- 92.118.13[.]18
- 91.132.136[.]174
- 95.26.248[.]29
- 95.26.253[.]176
- 170.55.59[.]2
- 185.220.101[.]5
- 192.64.6[.]217
- 195.74.76[.]237
- 195.239.51[.]117

Excluded Hostnames:

- John
- admin
- Admin
- ADMIN
- USER
- User
- user
- JOHN
- JOHN-PC
- WALKER-PC
- John-PC
- WALKER

User Agents

Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR1.0.3705;)

Contacted Domains

- dyndns[.]org
- freegeoip[.]app

Source: https://threatresearch.ext.hp.com/the-many-skins-of-snake-keylogger/