



DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

GuLoader deploys Remcos

Malware Lab Analysis Report

(part 1)

Summary

1. Our Malware Lab	03
2. Executive Summary	05
3. Analysis	07
3.1 NSIS Script Analysis	09
3.2 Overview of GuLoader	11
3.3 IOC	18
4. Conclusions	19

This document is protected by copyright laws and contains material proprietary to the Defence Tech Holding S.p.A Società Benefit. It or any components may not be reproduced, republished, distributed, transmitted, displayed, broadcast or otherwise exploited in any manner without the express prior written permission of Defence Tech Holding S.p.A Società Benefit. The receipt or possession of this document does not convey any rights to reproduce, disclose, or distribute its contents, or to manufacture, use, or sell anything that it may describe, in whole or in part.

1

Our Malware Lab

1. Our Malware Lab

Defence Tech Malware Lab daily performs dissection of malware with the aim of timely understanding the technological evolutions of attacks, consolidating the knowledge of necessary to make more effective and faster the process of incidents responding, contributing to spreading information about emerging threats into the expert's community and among its clients.

Malware Lab analysts are continuously engaged in searching and experimenting new analysis tools, for increasing accuracy and scope of action with regard to

the proliferation of new evasion and anti-analysis techniques adopted by malwares.

The Malware Lab is also committed to the development of proprietary tools for malware analysis and supporting the management and response of incidents.

Besides malware analysis, Malware Lab ideated and implemented an automatic process of extraction of **Indicators of Compromise (IOC)** that is daily run on dozens of new malwares, intercepted in the wide for populating our Knowledge Base.



CORRADO AARON VISAGGIO

Group Chief Scientist Officer & Malware Lab Director

a.visaggio@defencetech.it



DEFENCE TECH

2

Executive Summary

2. Executive Summary

Since the beginning of the 2023, malicious campaigns involving GuLoader (alias CloudEyE) have been reported in Italy^{1 2}.

GuLoader is a sophisticated shellcode-based downloader that typically plays a crucial role in downloading and distributing a wide variety of trojans and stealers as encrypted payloads. It acts as the first stage of a more complicated attack performed by the payload it carries. The key feature of GuLoader lies in its ability to evade the detection of traditional anti-malware solutions through advanced packing and encryption techniques. As if it's not enough, GuLoader is continuously updated with new advanced anti-debugging techniques, anti-analysis mechanisms and sandbox detection, which makes the process of reverse engineering significantly challenging for the analysts. Therefore, even commercial sandboxes sometimes are not always able to detect it³.

Earlier this year, GuLoader has been observed deploying malware such as Remcos⁴ spread with Phishing e-mails. The distribution is performed by tricking users into opening an attached PDF file that will redirect them to a cloud platform, where a ZIP file containing the malicious executable is available for download. It is worth noticing that these malware distribution campaigns follow a consistent modus-operandi, so we will not go into specific details here.

Our analysis will cover the initial stage of the sample we intercepted, consisting in an NSIS installer package and an overview of the less-known GuLoader capabilities, while the features of Remcos, that is the payload distributed by this specific GuLoader sample, will be described in the second part of the report.

¹ <https://www.securityopenlab.it/news/2721/malware-in-italia-occhio-alla-nuova-versione-di-guloader.html>

² <https://www.difesaesicurezza.com/cyber/cybercrime-massiccia-campagna-guloader-anche-in-italia/>

³ <https://app.any.run/tasks/c5192bf6-a86e-406a-a80e-734f29330019/>

⁴ <https://www.cyfirma.com/outofband/guloader-deploying-remcos-rat/>

3

Analysis

3. Analysis

The sample we analysed is an installer created using Nullsoft Scriptable Install System⁵ (NSIS). NSIS is a widely used open-source script-based tool to build Windows installers which is able to handle complex installation tasks.

Launching the executable shows a typical installer window for a few seconds which executes the “installation” process without any other confirmation from the user, closing itself once the process completes. Actually, the installer copies some decoy files and the malware onto the disk, then executes it in the background.

As the first static analysis step, we tried to extract the content from the installer using many archive manager tools. Among the various files, we observed the presence of a suspicious binary file, named “Fiskeredskabernes.Rad”. Inspecting the content of the file led us to believe that it’s likely encrypted, indicating that it may contain executable code as part of the infection chain.

To properly understand the use of this file we must disassemble the installer package to obtain the original NSIS script,

however most extraction tools we used either did not offer this feature or it wasn’t working properly.

There are various tools capable of decompiling NSIS scripts⁶, we opted for using 7-zip 15.06. Note that this is an old version of 7-zip as modern releases for some reason removed the NSIS disassembler feature. Since this version is outdated and potentially vulnerable, we took precautions by conducting the extraction process within a Virtual Machine environment.

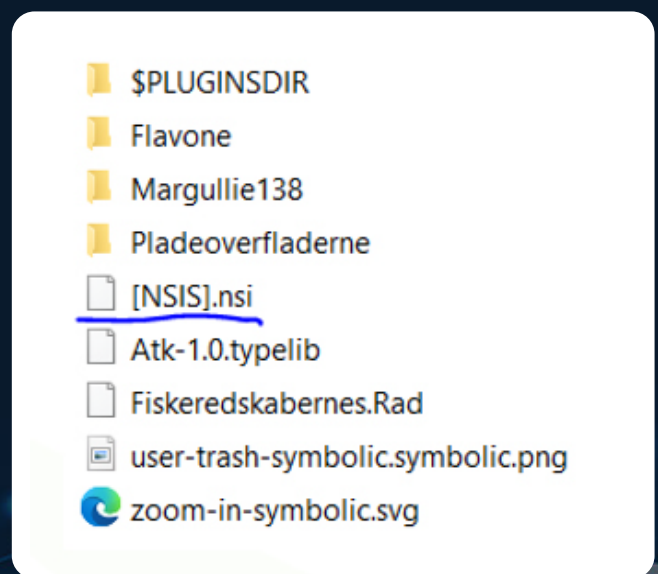


Figure 1. NSIS installer extraction

⁵ https://nsis.sourceforge.io/Main_Page

⁶ https://nsis.sourceforge.io/Can_I_decompile_an_existing_installer

We have successfully obtained the necessary NSI script which will be used during the installation process and that we are going to analyse in the next section.

3.1 NSIS Script Analysis

The installation process appears normal as all the extracted files from the installer archive are dumped onto the disk. However, the install script does indeed check for the existence of "Fiskeredskabernes.Rad", as seen in Figure 2.

```
IfFileExists $INSTDIR\Fiskeredskabernes.Rad label_241 label_221
```

Figure 2. Check existence of the file

The install script contains many obfuscated functions and a pervasive control flow obfuscation, since this is a rather uncommon language there are not many tools to help understand it. We observed that the code relies on one critical "System::Call" function which allows the NSIS script to interact with outside functions.

As seen in figure 3, the Call function takes a single string argument which the runtime will deserialize and use to execute the instruction, since the script is obfuscated it's impossible to statically recover the arguments it receives throughout the execution.

```
label_26:  
ExpandEnvStrings $ _43_ %Atomiserendes%\progesterone  
SetRegView 64  
Call func_0  
SectionSetSize 2 1841083292  
SetDetailsPrint textonly  
System::Call *$8  
; Call Initialize_____Plugins  
; SetOverwrite off  
; File $PLUGINS\DIR\System.dll  
; SetDetailsPrint lastused  
; Push *$8  
; CallInstDLL $PLUGINS\DIR\System.dll Call  
GetDlgItem $ _34_ $ _35_ 12330  
StrCpy $ _99_ $ _101_  
IfRebootFlag label_39 label_40
```

Figure 3. Invoking the function named "Call" from System.dll

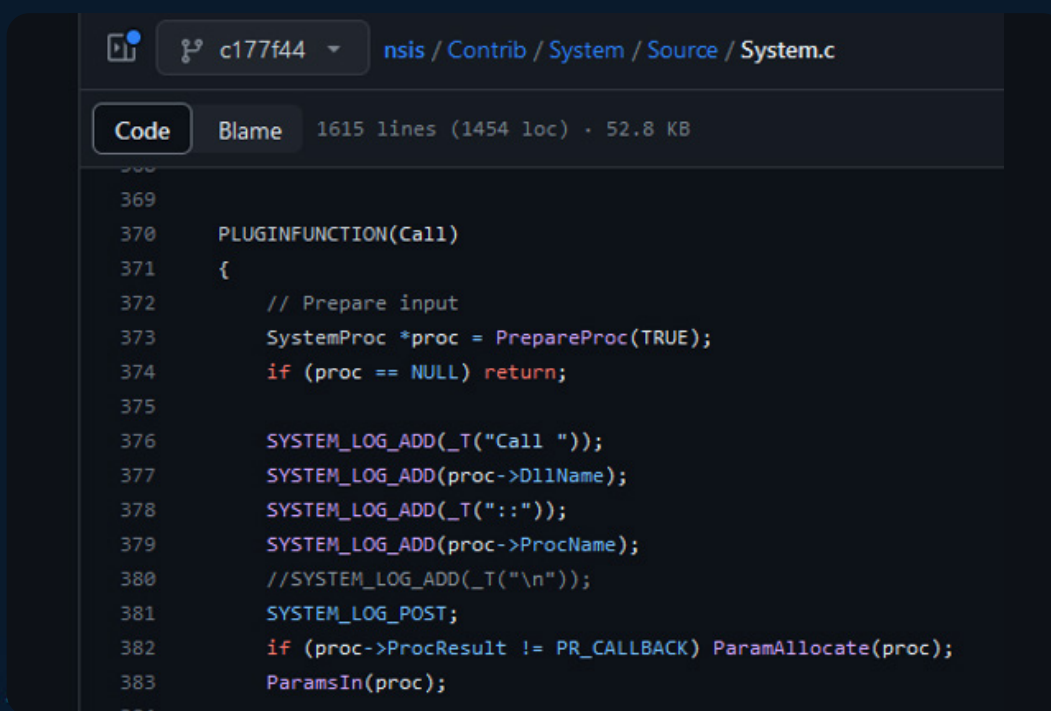
After some research, we found out that System:Call refers to the "Call" function of System.dll, a legitimate component of NSIS.

Moreover, analysing System.dll with the Pestudio⁷ tool reveals the use of LoadLibrary, which further indicates the dynamic behaviour of System.dll and likely its Call function.

technique (2)	value (215)
T1106 Execution through API	<u>LoadLibrary</u>
T1055 Process Injection	<u>VirtualFree</u>
T1055 Process Injection	<u>VirtualAlloc</u>
T1055 Process Injection	<u>VirtualProtect</u>

Figure 4. Suspicious imports of System.dll from pestudio

At this point we decided the best way forward would be dynamic analysis by hooking functions in System.dll. Our target is the "Call" function, we could easily find it in the disassembled binary by using the NSIS project GitHub mirror⁸ as a reference.



```
369
370  PLUGINFUNCTION(Call)
371  {
372      // Prepare input
373      SystemProc *proc = PrepareProc(TRUE);
374      if (proc == NULL) return;
375
376      SYSTEM_LOG_ADD(_T("Call "));
377      SYSTEM_LOG_ADD(proc->DllName);
378      SYSTEM_LOG_ADD(_T("::"));
379      SYSTEM_LOG_ADD(proc->ProcName);
380      //SYSTEM_LOG_ADD(_T("\n"));
381      SYSTEM_LOG_POST;
382      if (proc->ProcResult != PR_CALLBACK) ParamAllocate(proc);
383      ParamsIn(proc);
384
```

Figure 5. Source code of the "Call" function

⁷ <https://www.winitor.com/>

⁸ <https://github.com/kichik/nsis/blob/c177f44ce2089f5aa5d1a0f49a6c02c487e33c1f/Contrib/System/Source/System.c#L373>

From the source in figure 5 we can see that both the target DLL and function names are stored in the *SystemProc* structure, by setting a breakpoint on this function we can intercept the call names. Using this technique, we observed the install script perform the following sequence of native Win32 API calls:

1. *kernel32::CreateFileA*: This is used to open the .rad file
2. *kernel32::VirtualAlloc*: This is used to allocate a memory buffer with Read, Write and Execute permissions
3. *kernel32::ReadFile*: This is used to read the content of the .rad file to the just allocated buffer
4. *kernel32::EnumResourceTypesA*⁹: This is an enumeration function that iterates over the executable resources and calls a function via pointer for each; in practice it's used as an obfuscation trick to dynamically invoke the just extracted code without using suspicious functions or assembly routines.

3.2 Overview of GuLoader

At this point the dynamically loaded payload takes control, and the first stage of the shellcode is the unpacking process. It consists in an unpacker that will decrypt and execute the next stage.

The interesting thing about this unpacker is its control flow obfuscation: the whole code is composed by small blocks linked with unconditional jumps (see figure 6); this is a common pattern used to confuse the disassembler. To overcome this challenge, we used IDA Pro's tracing features¹⁰ which executes the program while producing a detailed trace of the execution, in a way similar to emulators.

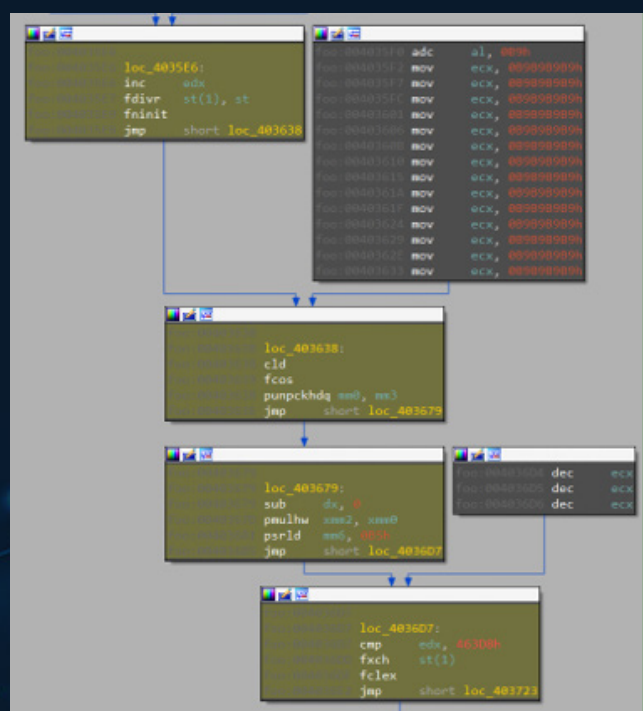


Figure 6. Blocks linked by unconditional jumps

⁹ <https://learn.microsoft.com/en-us/windows/win32/api/winbase/nf-winbase-enumresourcetypesa>

¹⁰ <https://hex-rays.com/wp-content/uploads/2019/12/tracing.pdf>

Then, we could simply identify the entry point for the next stage by searching the logs for the part where the control flow leaves the packer code region. As a result, we effectively bypassed the packer.

Once the packer is defeated, we were able to extract the resulting shellcode as a standalone EXE file in order to statically analyse it.

Then, we faced the real code of GuLoader and the first step of its execution is dynamically importing system functions

through a technique called PEB (Process Environment Block) walking. PEB Walking consists in manually parsing the Windows dynamic loader data structures which can be found from the PEB to find pointers to loaded libraries and their respective functions, as shown in figure 7. The PEB data structure is not completely documented, and programs are not supposed to access such implementation details, however this doesn't stop malware developers from using them to evade static and dynamic analysis.

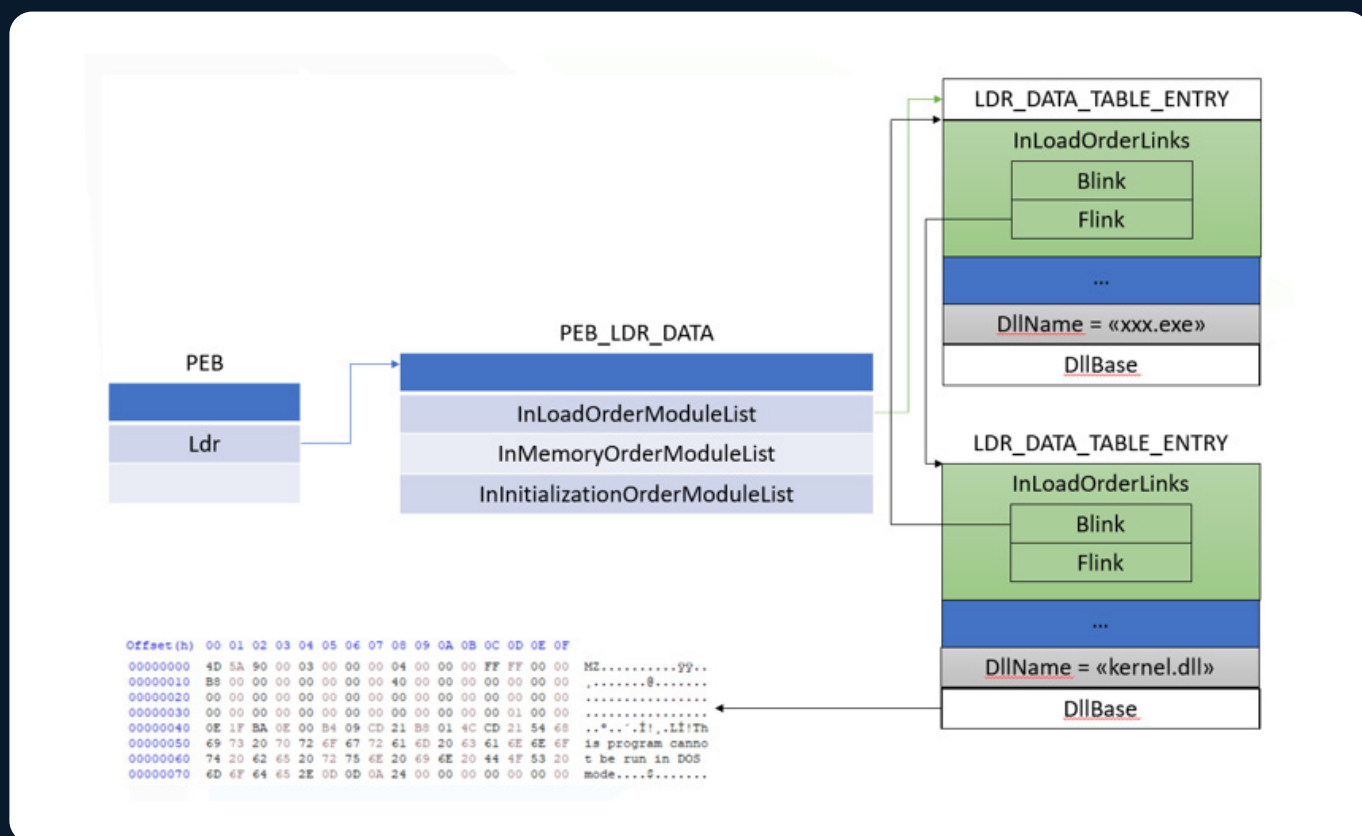


Figure 7. PEB walking

Once the malware identifies all the necessary dynamic functions, it proceeds to set up an exception handler using the "RtlAddVectoredExceptionHandler"¹¹ function from ntdll. The handler will be used as an anti-debugging technique and as a means to obfuscate the control flow. In fact, from this point on, the malware's code contains several special instruction patterns such as int3, which causes an exception and triggers the custom exception handler.

The real execution flow is restored by the exception handler, seen in figure 8, which would read a byte following the instruction where the exception occurred, then would perform a xor operation with a constant value and use the result to determine the number of bytes to skip in order to resume the program's real execution flow.

```
if ( ExceptionCode == EXCEPTION_BREAKPOINT )
{
    v6 = anti_debug_1(retaddr);
    Eip = (_BYTE *)v6->Eip;
    if ( *Eip == 0xCC )
    {
        v8 = Eip[1] ^ 0xFB;
        for ( i = (_BYTE *)(v8 + v6->Eip - 1); (_BYTE *)(v6->Eip + 2) != i; --i )
        {
            if ( *i == 0xCC )
                return EXCEPTION_CONTINUE_SEARCH;
        }
        v6->Eip += v8;
        goto LABEL_10;
    }
}
return EXCEPTION_CONTINUE_SEARCH;
```

Figure 8. Pattern which causes the exception

¹¹ https://www.x86matthew.com/view_post?id=windows_no_exec

There are several instruction sequences that can trigger this exception-based obfuscation and they're handled with some slight differences. Other than the `int3` instruction, we also observed the "pushf", "or", "popf" and "mov" sequence wherein the first three instructions enable the hardware single step flags causing an exception at the fourth instruction, jumping to the exception handler. Another recurring pattern involves a series of "mov", "xor" and "add" instructions executed on a single register, followed by a "mov" instruction which dereferences that register. This causes a segmentation fault exception, indicating that an invalid memory address was

accessed. However, when this address falls within a specific range, the exception handler recognises it as one of the previously discussed cases.

To defeat this technique, we wrote a script that matches these patterns and corrects the control flow by replacing the offending instructions with an unconditional branch to the target, this also allows IDA to properly detect function boundaries and decompile them.

One more interesting thing to note about the exception handler is its anti-debugging implementation, shown in figure 9.

```
1 _CONTEXT *__usercall exception_handler_anti_debug@<eax>(_EXCEPTION_POINTERS *a1@<eax>)
2 {
3     _CONTEXT *result; // eax
4
5     result = a1->ContextRecord;
6     if ( result->Dr0 || result->Dr1 || result->Dr2 || result->Dr3 || result->Dr6 || result->Dr7 )
7         return 0;
8     return result;
9 }
```

Figure 9. Checking Debug Registers to verify the presence of hardware breakpoints

When an exception occurs in a program, the operating system calls the exception handler passing it the CPU context.

The CPU context is a data structure which represents a snapshot of the processor's state at the time of the exception containing information such as the value of all registers and information about what caused the exception.

The exception handler is supposed to analyse the context struct and decide if the exception is recoverable, and if that's the case manually fix the context before returning to the OS, or terminate the program (in practice, before the program is terminated, the exception will travel "upwards" to pass through all the registered handlers in case this specific exception must be handled by a piece of code higher on the call stack.)

But in the case of GuLoader, the malicious exception handler abuses the information in the CPU Context to access the Debug Registers, which represent hardware breakpoints; if any of them is set, it means that there is a debugger attached. In this case GuLoader does not handle the

exception, causing the process to crash.

At this point the main GuLoader will perform several anti-analysis checks ranging from simple process detection, by searching for well-known virtual machine guest tools process names, to unhooking low level ntdll.dll functions.

Once all the anti-analysis checks are successful, GuLoader will attempt to migrate into a different process through code injection. In this case, it creates a suspended "CasPol.exe" process by passing the "CREATE_SUSPENDED" flag¹², using CreateProcessInternalW API¹³. CasPol is a tool part of the .NET Framework and typically present in the following system directories:

- %Windows Directory%\Microsoft.NET\Framework\<<version on 32-bit systems>
- %Windows Directory%\Microsoft.NET\Framework64\<<version on 64-bit systems>



¹² <https://learn.microsoft.com/en-us/windows/win32/procthread/process-creation-flags>

¹³ <https://medium.com/@Achilles8284/the-birth-of-a-process-part-2-97c6fb9c42a2>

GuLoader attempts to inject itself to this new process through process hollowing, in particular, instead of hollowing the main executable of the process it will try to replace the content of one of its libraries and if this process fails it resorts to simply allocating executable memory as a fallback.

The new process is tasked to download the payload from the C2 server; the payload is encrypted, and the encryption

scheme has already been analysed in depth by McAfee¹⁴.

In this case the C2 server is "*wearethestandard[.]com[.]au/mt/iaJrXrOivtG81.bin*" and the downloaded payload is a sample of the RemCos malware family.

Based on the observed behaviour in figure 10, the GuLoader payload drops and launches two more executable files: F777.exe and serv2.exe

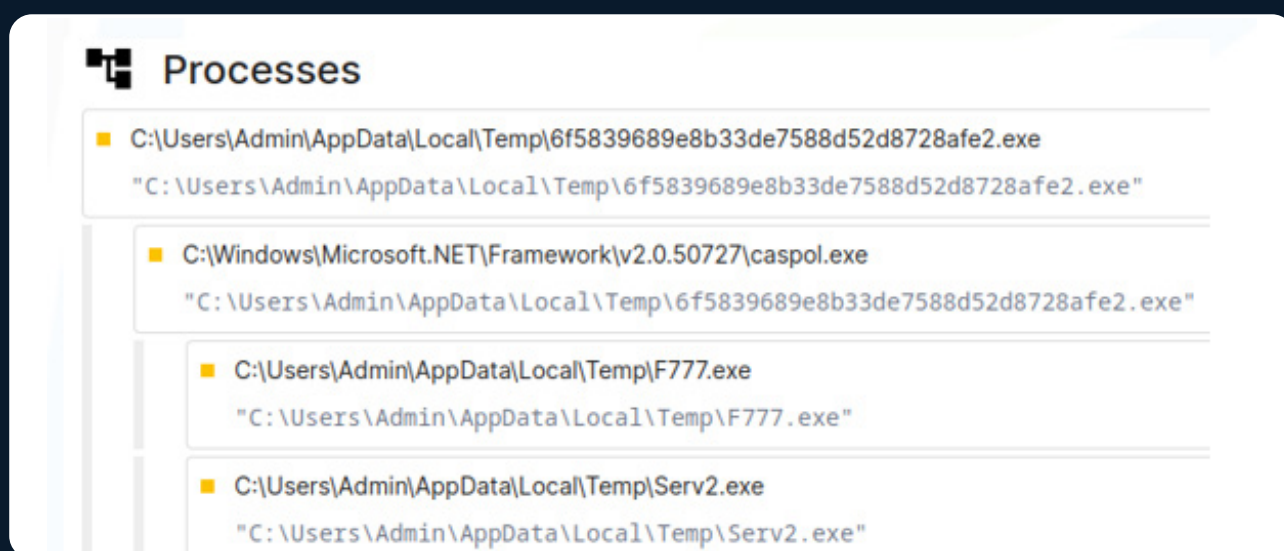


Figure 10. Process tree in Hatching Triage Sandbox

The first file, F777.exe is a .NET executable and can be quickly identified by its metadata: it's a file produced by a legitimate 4dots¹⁵ ¹⁶ tool suite to convert Microsoft Excel files to standalone executables and to create slideshow from photos as a standalone exe. This file plays no role in the infection chain but is only used to show a document to the user as a cover, so they don't realize they have been infected.

¹⁴ <https://www.mcafee.com/blogs/other-blogs/mcafee-labs/guloader-campaigns-a-deep-dive-analysis-of-a-highly-evasive-shellcode-based-loader/>

¹⁵ <https://www.4dots-software.com/convert-excel-to-exe/>

¹⁶ <https://www.4dots-software.com/exe-slideshow-maker/>

```
1 // C:\Users\Analyst\Desktop\F777.exe
2 // EXESlideshow, Version=1.0.0.0, Culture=neutral, PublicKeyToken=
3
4 // Entry point: EXESlideshow.Program.Main
5 // Timestamp: 64930C00 (6/21/2023 4:41:04 PM)
6
7 using System;
8 using System.Diagnostics;
9 using System.Reflection;
10 using System.Runtime.CompilerServices;
11 using System.Runtime.InteropServices;
12
13 [assembly: AssemblyVersion("1.0.0.0")]
14 [assembly: RuntimeCompatibility(WrapNonExceptionThrows = true)]
15 [assembly: Guid("3508e632-2d22-4675-ba79-620e5a5553a3")]
16 [assembly: CompilationRelaxations(8)]
17 [assembly: AssemblyFileVersion("1.0.0.0")]
18 [assembly: ComVisible(false)]
19 [assembly: AssemblyTrademark("")]
20 [assembly: AssemblyCopyright("Copyright © 2023 4dots Software")]
21 [assembly: AssemblyProduct("Convert Excel To EXE 4dots")]
22 [assembly: AssemblyCompany("4dots Software")]
23 [assembly: AssemblyConfiguration("")]
24 [assembly: AssemblyDescription("")]
25 [assembly: AssemblyTitle("EXE Slideshow")]
```

Figure 11. Source code of F777.exe

On the other hand, serv2.exe is the actual Remcos sample, the executable does not appear to be packed as self-identifying strings can be immediately detected as observed in figure 12. It will be the focus of the second part of the report.

```
LABEL_45:
    sub_402093("Remcos Agent initialized");
    sub_402093("i");
```

Figure 12. Code snippet of the Main function

3.3 IOC

In the next table we inserted IoC of the GuLoader sample analysed in this report.

Note: detection rates are as of time of writing, given the low rates they are likely to increase over the course of the following days as AV vendors update their products.

Type	Value	Note
SHA-256	0d771bed67134df3cfcbafe953d9378ca9a40ba93f05f726b9286638a08318e4	NSIS Installer VirusTotal - 15/71
SHA-256	b735367eeba455050c2bfb7c3100b5d1b3c742b1aa1d32d12ad9bc5a6f44f045	Fiskeredska- bernes.Rad VirusTotal - 0/59
SHA-256	1b7edf5ca77dd80a55cec04c89faa7c5fbc7a0352083547ba990514d6eef4ba1	GuLoader's payload VirusTotal - 0/59
SHA-256	2da6c07bdd6d897ac282e773149df7e0118c00257532dc598c40a28a36e49bce	F777.exe VirusTotal - 2/71
SHA-256	b917583a1bd2371f4c1916cf0a4831e4d26a1d2fc7103005900d4cb775e73359	serv2.exe (Remcos) VirusTotal - 53/70
Domain	wearethestandard[.]com[.]au	C2 VirusTotal AlienVault

Table 1. Indicators of compromise

4

Conclusions

4. Conclusions

GuLoader is a sophisticated malware loader that employs multiple evasion and obfuscation techniques to deliver its payload. GuLoader has been observed to deploy a wide variety of stealers and remote access tools, including Remcos, which will be the focus in the second part of this report.

As this report has shown, GuLoader uses advanced anti-debugging and anti-analysis methods exploiting exception handlers to obfuscate the code control flow, in order to complicate the reverse engineering process.

Furthermore, GuLoader can detect isolated environments such as Virtual Machines or sandboxes, used by security researchers and analysts for testing and analyses. This environment awareness allows the malware to remain undetected and evade detection systems. Commercial sandboxes must continuously update their detection capabilities to keep up with GuLoader's evolving tactics.

To strengthen an organization's security posture against the distribution of these

downloaders, we recommend a multi-layered approach that starts with the training of the employees on cybersecurity best practices and emerging malware trends. In addition to training, critical measures include the deployments of anti-malware solutions, robust e-mail filtering, behaviour-based detection mechanism and advanced threat detection solutions.

Moreover, to restrict lateral movements and to prevent the spread of infections, it's highly recommended to also implement network segmentation. By segmenting the network, an additional layer of defence is established, limiting the impact of potential infections, and isolating malicious activities.

Finally, it's important to continuously monitor network traffic on all the endpoints to intercept the communication with the malicious C2 servers.

By combining these layers of security, organizations can significantly reduce the malware attack surface and the risk of malware infiltrations.



DEFENCE TECH

Terra, Cielo, Mare, Spazio, Spazio cibernetico.
PROTEGGIAMOLI

DONE IT
IT SECURITY

NEXT
INGEGNERIA DEI SISTEMI

FORAMIL
RADAR TECHNOLOGIES & DEFENCE SYSTEMS

INN·DESI
electronic systems

Defence Tech Holding S.p.A Società Benefit

Via Giacomo Peroni, 452 - 00131 Roma

tel. 06.45752720 - fax 06.45752721

info@defencetech.it - www.defencetech.it