Malware report
CCN-CERT ID-09/20

Guloader

April 2020
LIMITACIÓN DE RESPONSABILIDAD
El presente documento se proporciona de acuerdo con los términos en él recogidos, rechazando expresamente cualquier tipo de garantía implícita que se pueda encontrar relacionada. En ningún caso, el Centro Criptológico Nacional puede ser considerado responsable del daño directo, indirecto, fortuito o extraordinario derivado de la utilización de la información y software que se indican incluso cuando se advierta de tal posibilidad.

AVISO LEGAL
Quedan rigurosamente prohibidas, sin la autorización escrita del Centro Criptológico Nacional, bajo las sanciones establecidas en las leyes, la reproducción parcial o total de este documento por cualquier medio o procedimiento, comprendidos la reprografía y el tratamiento informático, y la distribución de ejemplares del mismo mediante alquiler o préstamo públicos.
INDEX

1. ABOUT CCN-CERT, NATIONAL GOVERNMENTAL CERT ........................................... 4
2. EXECUTIVE SUMMARY .......................................................................................... 4
3. GENERAL DETAILS ................................................................................................. 5
4. INFECTION PROCESS ............................................................................................... 6
  4.1 VISUAL BASIC LOADER ...................................................................................... 6
  4.2 SHELLCODE ........................................................................................................ 7
    4.2.1 ANTI-ANALYSIS ............................................................................................ 8
    4.2.2 IMPORTED FUNCTIONS .............................................................................. 10
    4.2.3 INJECTION .................................................................................................... 10
  4.3 INJECTED SHELLCODE ....................................................................................... 12
    4.3.1 PERSISTENCE ............................................................................................... 12
    4.3.2 COMMUNICATIONS ..................................................................................... 12
    4.3.3 ADDITIONAL BINARY EXECUTION .............................................................. 14
5. DISINFECTION .......................................................................................................... 15
6. DETECTION RULES ................................................................................................. 16
  6.1 YARA RULE ......................................................................................................... 16
7. INDICATORS OF COMPROMISE ............................................................................... 16
1. ABOUT CCN-CERT, NATIONAL GOVERNMENTAL CERT

The CCN-CERT is the Computer Emergency Response Team of the National Cryptologic Centre, CCN, within the National Intelligence Centre, CNI. This service was created in 2006 as a Spanish National Governmental CERT and its functions are included in Law 11/2002 regulating the CNI, RD 421/2004 regulating the CCN and RD 3/2010, dated 8th January, regulating the National Security Scheme (ENS), modified by RD 951/2015 of 23rd October.

Its mission therefore is to contribute to the improvement of Spanish cybersecurity, being the national alert and response centre that cooperates and helps to respond quickly and efficiently to cyberattacks and to actively confront cyber threats, including the coordination at the national public level of the different Incident Response Teams or existing Security Operations Centres.

Its ultimate aim is to make cyberspace more secure and reliable, preserving classified information (as stated in Article 4.F of Law 11/2002) and sensitive information, defending Spanish Technological Heritage, training expert personnel, applying security policies and procedures and using and developing the most appropriate technologies for this purpose.

In accordance with these regulations and Law 40/2015 on the Legal Regulation for the Public Sector, the CCN-CERT is responsible for the management of cyber incidents affecting any public body or company. In the case of critical operators in the public sector, the management of cyber incidents will be carried out by the CCN-CERT in coordination with the CNPIC.

2. EXECUTIVE SUMMARY

There has been evidence of a malware campaign in Spain and Portugal, in which using a false COVID-19 vaccine as a hook, the recipient is urged to open an attached file, called 'COVID-19.exe' inside 'COVID-19.tar', which contains a compressed version of Guloader.

This document presents the analysis performed over the downloader variant known as Guloader, identified by the SHA256 signature 5d91ff8d079c5d890da78adb8871e146749872911efe2ebf22cfd02c698ed33d.

The main goal of the binary is to load in memory a shellcode aimed with downloading an additional payload from a remote server for its execution. Guloader appeared in the malware scene in December 2019, but it has been the rise it has shown in April 2020 the fact that has motivated this research. The Guloader name given to this family comes from the words “Google loader”, as in early stages of its development the actors used Google Drive locations to store additional payloads. The
project is being actively updated and the obfuscation scheme it uses is playing a key role to maintain the samples undetected.

Nowadays, downloaders provide an essential service regarding malware distribution. Actors willing to increment the number of bots from their botnets definitely appreciate an effective tool able to provide installs at the same time it remains undetected. If it is true the lifespan of downloader families does not last long, families like Smokeloader or Emotet prove otherwise.

In the next sections of the document, technical details are provided about the behavior of the initial binary, as well as of the shellcode it loads in memory. A YARA rule and indicators of compromise to detect the analyzed sample are provided too.

3. GENERAL DETAILS

The SHA256 signature below identifies the loader component triggering the infection process.

<table>
<thead>
<tr>
<th>File</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>guloader.exe</td>
<td>5d91ff8d079c5d890da78adb8871e146749872911efe2ebf22cfd02c698ed33d</td>
</tr>
</tbody>
</table>

The sample, a Portable Executable for 32-bit Windows systems developed in Visual Basic, fakes its compilation timestamp going back to 2013.

Figure 1. Properties of the initial binary

It was first seen on April 16th 2020.

While a packer does not protect the executable, the obfuscation layer it shows, which changes across samples, seems to be working against anti-virus detections.
4. INFECTION PROCESS

The infection process is divided in three stages, defining this way the structure of the current section:

- **Loader**, the code aimed with loading, decoding and executing the shellcode.
- **Shellcode**, the aimed is to detect an analysis environment, import libraries and finally inject itself into a legitimate binary of the system.
- Download an additional payload and finally achieving persistence in the system.

4.1 VISUAL BASIC LOADER

The initial binary serves as a loader for the piece of shellcode responsible for subsequent stages of the infection process. Despite the binary has been developed in Visual Basic, the code aimed with loading, decoding and executing the shellcode can be analyzed with any disassembler.

The piece of shellcode is stored in the `.text` section of the initial binary. After copying it to a new allocated buffer and decoding it in its new location with a 4-byte **XOR** routine, execution continues from the entry point of the shellcode, still within the process space of the initial binary.
The code tasked with the execution of the shellcode shows an obfuscation layer consisting of the insertion of junk instructions, aimed to increase the code complexity while the semantics of the code remain the same. Figures 2 and 3 highlight the instructions of interest for the analyst while the rest correspond with the aforementioned junk code. This obfuscation technique is used in both the initial binary and in the shellcode to be explained in the following subsections.

4.2 SHELLCODE

The second stage of the infection process begins with the first execution of the shellcode.

Aside from the obfuscation scheme described above, the shellcode implements a variety of techniques to complicate dynamic analysis, where the majority of them are aimed to break the debugger.
4.2.1 ANTI-ANALYSIS

The first of the techniques to prevent the malicious code to run, in what the developers would consider an analysis environment, is based in enumerating the windows of the system by calling the function `EnumWindows`. If the returning value is lower than the expected one, execution finalizes in this stage.

![Figure 5. Process is terminated if EAX value is lower than 12](image)

To prevent debuggers to attach to running samples, Guloader implements hooks for functions `DbgBreakPoint` and `DbgUiRemoteBreakin`, as they are responsible for taking control over the process whose execution is to be interrupted.

The 0xCC opcode from the function body from `DbgBreakPoint` is substituted by a NOP.

![Figure 6. Hooked DbgBreakPoint function](image)

On the other hand, the first instructions from `DbgUiRemoteBreakin` are substituted by a small piece of code, aimed with generating an exception if this function is called.

![Figure 7. Hooked DbgUiRemoteBreakin function](image)

Continuing with debugger exceptions, the shellcode hides its running thread from debuggers by calling the function `NtSetInformationThread` pushing the value `ThreadHideFromDebugger` as a parameter.
If after performing such call the execution flow reaches a breakpoint, the process would be terminated due to a crash. In addition to the described methods to crash the debugged process, certain functions are not called directly from the malicious code but are inspected in further detail to check for the presence of breakpoints set to give control to the analyst.

![Figure 8. Thread hide to debuggers](image)

The function checks for both hardware breakpoints, by checking DR0-DR7 registers, and software breakpoints by looking for the opcodes \texttt{0xCC}, \texttt{0x3CD} and \texttt{0xB0F} whenever a function is to be checked. If breakpoints are found, the program modifies its flow resulting in a crash finalizing the execution of the shellcode.

![Figure 9. Software and hardware breakpoint checks](image)
4.2.2 IMPORTED FUNCTIONS

As observed in figure 8, before calling the function \texttt{NtSetInformationThread} its address is retrieved dynamically. This technique of resolving functions addresses at run time provides an additional layer of protection against static analysis, so at first sight there is no way to tell which functions from Windows libraries are imported and where are they called.

Functions to be imported are identified by the \texttt{djb2} value of their name. To import a function, its hashed name is placed into EDX register and the exports from the respective library are iterated and \texttt{djb2} hashed until finding a match.

```c
void __stdcall djb2_hash( WORD *string )
{
    int hash; // edx
    hash = 5381;
    do
    {
        hash = *(unsigned __int32 *)string + 33 * hash;
        ++string;
    } while ( *string );
}
```

Figure 10. djb2 hash implementation

4.2.3 INJECTION

If execution has not been interrupted until this point, the last step of the shellcode when running from the process space of the initial binary consists of injecting itself into a legitimate binary of the system. The analyzed sample chooses the Internet Explorer Add-on Installer to perform the injection.

C:\Program Files(x86)\Internet Explorer\ieinstal.exe

To migrate to the selected process, it is first created in a suspended state.

<table>
<thead>
<tr>
<th>Name</th>
<th>PID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>guloader.exe</td>
<td>1758</td>
<td>Syndedraget</td>
</tr>
<tr>
<td>ieinstal.exe</td>
<td>2180</td>
<td>Instalador de complementos de Internet Explorer</td>
</tr>
</tbody>
</table>

Figure 11. ieinstal.exe suspended process

Next, the Windows \texttt{msvbm60.dll} library is mapped into the address 0x00400000 of the new process.
Finally, by calling the function `NtWriteVirtualMemory`, the shellcode being executed from within the process space of the initial binary is copied into the new spawned process.

It is possible to observe the opcodes from the injected shellcode into the `ieinstall.exe` process being the same as the opcodes shown in the entry point of the shellcode from figure 4.
The first execution of the shellcode finalizes after the injection, continuing the infection process from the new created process, from where the shellcode resumes execution choosing an alternative flow.

4.3 INJECTED SHELLCODE

In this final stage of the execution process, the main goal of the shellcode is finally shown, to download an additional payload from a remote server to execute it, achieving persistence in the infected machine to grant that behavior after each reboot.

4.3.1 PERSISTENCE

Before writing an entry in the registry to achieve the persistence, the installation directory is created after a folder and an executable name hardcoded in the shellcode.

```
C:\Users\[User]\Historiels\Diadelphan.exe
```

The registry key value IMPATIENT, also hardcoded, is created in the registry key `Software\Microsoft\Windows\CurrentVersion\Run` pointing to the binary in its installation directory, achieving by these means the persistence in the system.

![Figure 14. Persistence of Guloader](image)

4.3.2 COMMUNICATIONS

Before executing the final payload, first the shellcode needs to download it. The analyzed sample of Guloader shows an URL location from where to retrieve the final payload and a second URL, which shows what seems to be a default placeholder from the builder.

<table>
<thead>
<tr>
<th>Additional downloads</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="https://www.seashotbin.%5Dcom/Lord/Glx_encrypted_3277CA0.bin">https://www.seashotbin.]com/Lord/Glx_encrypted_3277CA0.bin</a></td>
</tr>
<tr>
<td><a href="http://myurl/myfile.bin">http://myurl/myfile.bin</a></td>
</tr>
</tbody>
</table>
The binary is retrieved performing a GET request with the hardcoded user-agent defined in the shellcode, which is characteristic of Internet Explorer 11.

Mozilla/5.0 (Windows NT 6.1; WOW64; Trident/7.0; rv:11.0) like Gecko

Remote servers hosting additional downloads from Guloader usually allow to list their content and the name of the binaries usually follows the pattern below.

[random]_encrypted_[A-F0-9]{7}.bin

![Remote server open directory](image)

**Figure 15. Remote server open directory**

Something worthwhile to highlight is the fact that Guloader checks the size of the downloaded payloads to check for a specific length before executing them.

```
seg000:005BAF38 fetch_and_execute_payload: ; ...
    cmp   edx, edx
    ...
```

**Figure 16. Downloaded file size check**

If the calculated size does not match with the expected one, which is hardcoded in the shellcode, the payload is ignored. If the size matches, the payload is decoded following a XOR routine prior to its execution. Taking a look at the encoded binary, there is a header formed of bytes following the distribution `[a-f0-9]{64}`, which is directly discarded before decoding the rest of the file to retrieve the final executable. The instruction at address `0x005B0AF6` shows how these 64 (0x40) bytes are discarded before checking the size. The bytes from the discarded header seem to have changed starting from April 20th 2020, so instead of the hexadecimal-like characters they now can take any value from `[0x00-0xFF]{64}`. This change does not affect the malware behavior in any way as the 64 bytes header is still discarded.
On April 20th, 2020, the actors appear to have decided to leave the aforementioned header format behind and instead of \([a-f0-9] \{64\}\), the range is extended to any value resulting in \([0x00-0xFF] \{64\}\). Despite the change in format, the header is still discarded.

For the decoding, the shellcode stores the **XOR** key, 575 bytes in length for this sample. The sample downloaded for execution by the analyzed Guloader turned out to be a remote access tool, known as **Netwire**.

<table>
<thead>
<tr>
<th>File</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Netwire</td>
<td>af0b56fffc1e8df83dc104e0afe91f8921ecfc66fb5599214189f9c90ec1a4d</td>
</tr>
</tbody>
</table>

### 4.3.3 ADDITIONAL BINARY EXECUTION

Instead of executing the additional payload in a new process, Guloader maps the new executable in the memory address 0x00400000 (where it previously mapped the **msvbfm60.dll** library) to run the sample in a new thread, also hidden from debuggers.
Figure 18. Fetched payload is run in an additional thread

After loading the additional payload, the tasks of Guloader are completed and its execution thread is terminated.

5. DISINFECTION

For disinfecting a system where the analyzed sample has been installed, the steps listed below are proposed.

1. Delete the installation directory from the persistence section and its content.
2. Delete the registry key from the persistence section pointing to the installation directory.
3. Reboot the system.

As Guloader is able to download and execute additional samples, the complete disinfection of the machine cannot be granted after following the described steps.
6. DETECTION RULES

6.1 YARA RULE

```plaintext
rule guloader
{
    meta:
        date = "2020-04-16"
        author = "CCN-CERT"
        sha256 = "5d91ff8d079c5d890da78adb8871e146749872911efe2ebf22cfd02c698ed33d"
    strings:
        $encoded_shellcode = {82 0A C4 0D 7B 8B 18 B0 2F 02 FD 19 7A 8B 18
            F1 22 08 F0 E1 F3 CE 5C 19 AF 25 18 F1 93 CA}
        $msvbvm60 = "MSVBVM60.DLL" ascii
    condition:
        uint16(0) == 0x5A4D and all of them
}
```

7. INDICATORS OF COMPROMISE

<table>
<thead>
<tr>
<th>File</th>
<th>SHA256</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loader Visual Basic</td>
<td>5d91ff8d079c5d890da78adb8871e146749872911efe2ebf22cfd02c698ed33d</td>
</tr>
<tr>
<td>Netwire</td>
<td>af0b56fffcc1e8df83dc104e0afe91f8921ecfc66fb5599214189fdc90ec1a4d</td>
</tr>
</tbody>
</table>

**Installation directory**

C:\Users\[User]\Historiels\Diadelphian.exe

**Final payload location**

https://www.seashotbin\.com/Lord/Glx_encrypted_3277CA0.bin

<table>
<thead>
<tr>
<th>Registry key</th>
<th>Key value</th>
</tr>
</thead>
<tbody>
<tr>
<td>HKCU\Software\Microsoft\Windows\CurrentVersion\Run\IMPATIENT</td>
<td>C:\Users[User]\Historiels\Diadelphian.exe</td>
</tr>
</tbody>
</table>