



DEFENCE TECH

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

Hijackloader distributes Vidar

Malware Lab Analysis Report

Summary

1. Our Malware Lab	03
2. Executive Summary	05
3. Analysis	08
3.1 Online distribution	09
3.2 DLL Side-loading	11
3.3 Technical analysis and behaviour	13
3.4 IOC	16
4. Conclusions	17

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

During OSINT activity an interesting online ad caught our attention. It was a "Download Now" button which linked to a website with an odd domain name.

The website, shown in figure 1, distributes a ZIP file via a MediaFire link and a

password to be able to open it. The archive name contains multiple special characters, inside of it there is a similarly named ZIP file which is password-protected. Inside this second ZIP file we find multiple files and one single executable called "setup.exe".

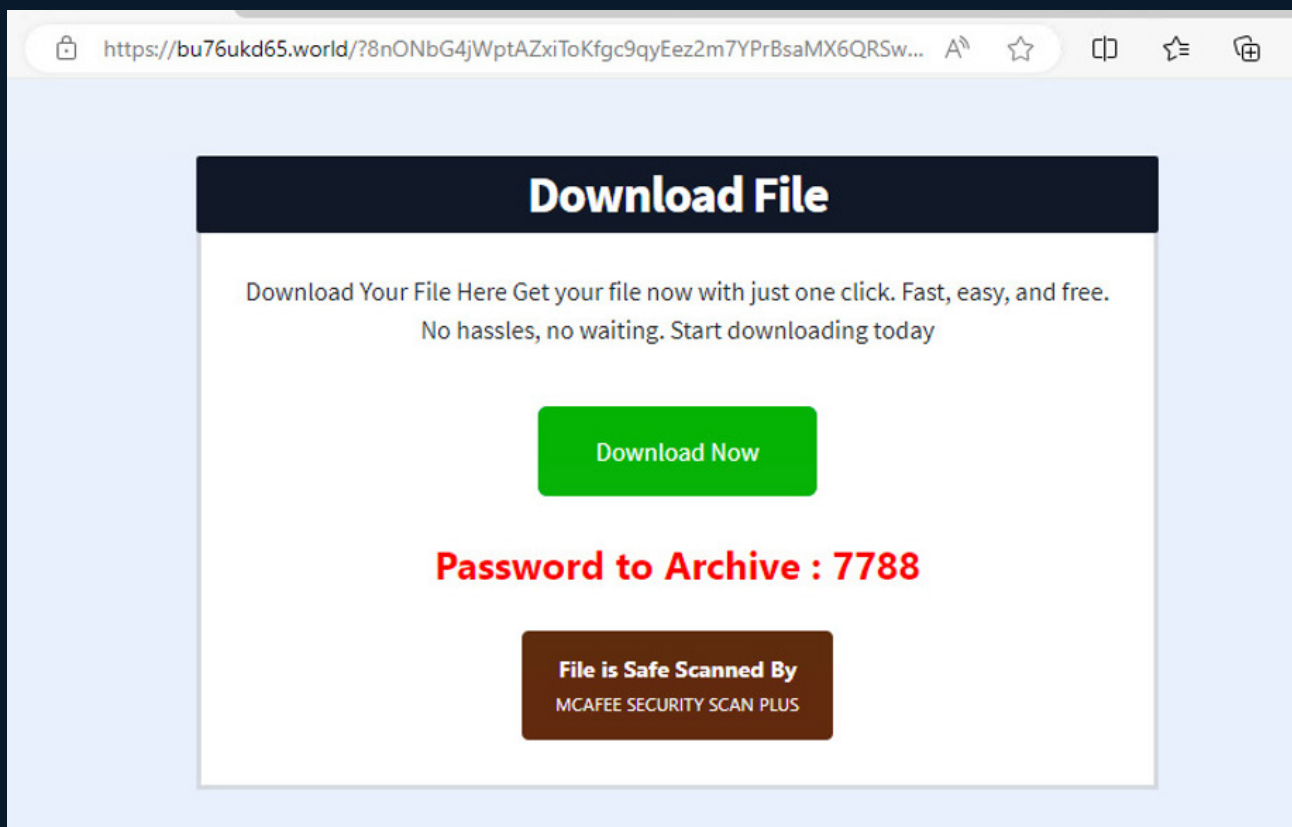


Figure 1. Initial malicious download page

The main executable file itself is legitimate, as it is signed by CISCO, however executing it on AnyRun to observe its behaviour¹ clearly shows traces of malicious activity. This is because during execution, it gets infected by one of the DLLs included in the ZIP. This technique is called DLL Side-loading².

The infection chain involves HijackLoader³ which distributes Vidar⁴, that collects credentials from several application pro-

files and bank details and is one of the first malware that grabs information on 2FA Software and Tor Browser. HijackLoader is a malware loader employed by attackers to deliver payloads through evasive techniques.

In this report we look at how DLL Side-loading works and how it is exploited by this sample to gain unauthorised code execution from a legitimate installation program.



¹ <https://app.any.run/tasks/fcfbb5a2-524b-45f6-89f2-64c1d2a51293/>

² <https://attack.mitre.org/techniques/T1574/002/>

³ <https://malpedia.caad.fkie.fraunhofer.de/details/win.hijackloader>

⁴ <https://malpedia.caad.fkie.fraunhofer.de/details/win.vidar>

3

Analysis

3. Analysis

3.1 Online distribution

The malicious page shown in Figure 1 is hosted on the bu76ukd65[.]world domain. The use of an uncommon gTLD and the random-looking nature of it indicates that it is likely part of a group of domains automatically generated and purchased all with the same intent of distributing malware.

The download button on the initial page links to a regular MediaFire download page for a ZIP file. The file name contains multiple special characters as well as the ZIP password, this is shown in Figure 2.

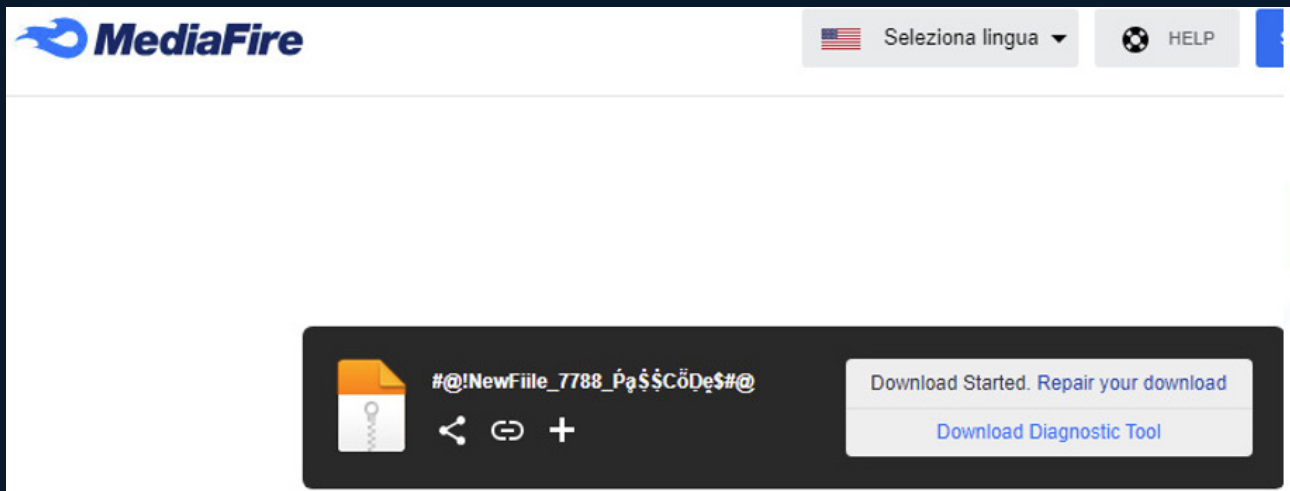
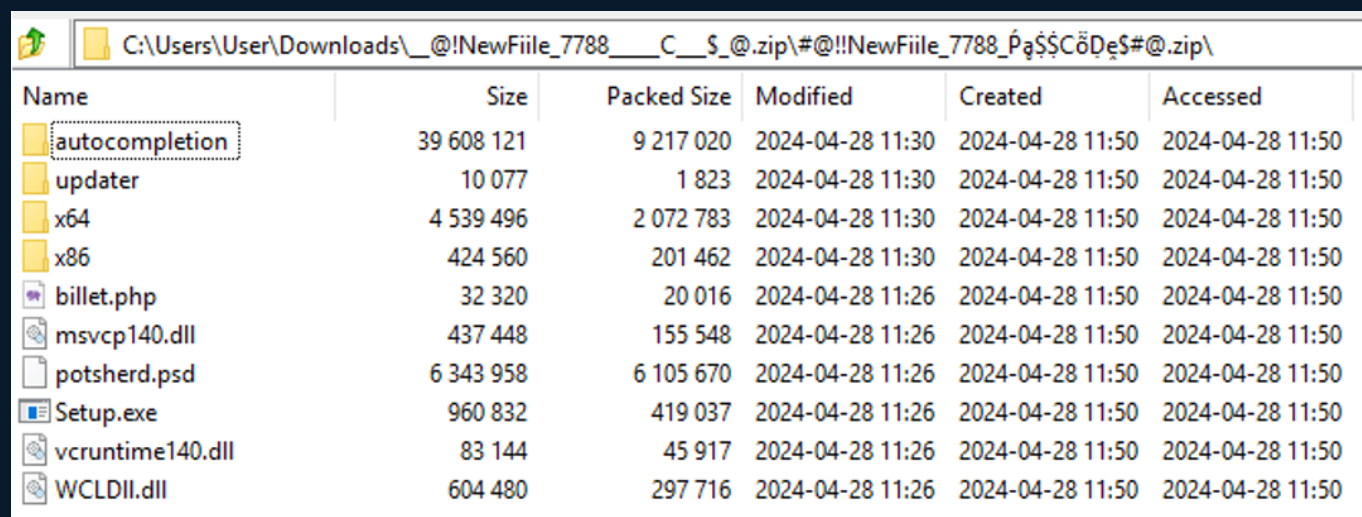


Figure 2. The MediaFire download page

We can only speculate that the special characters in the name help to bypass certain security solutions or cause name confusion due to name normalization applied by different software. While the original name " #@!NewFiile_7788_Þà\$ŠCöDè\$#@.zip " is a valid Windows file name, during our analysis the file got renamed to ' __@!NewFiile_7788_____C___\$_@.zip' by the various tools replacing special characters with the ' _ ' character.

The ZIP file contains a second ZIP file with the same name as the first, however it is password protected. Password protected archives are a common technique used to prevent anti-malware software from scanning the content of the inner files.

From now on we will only refer to the inner ZIP file as it contains all the files needed for the infection, the content is shown in figure 3.



Name	Size	Packed Size	Modified	Created	Accessed
autocompletion	39 608 121	9 217 020	2024-04-28 11:30	2024-04-28 11:50	2024-04-28 11:50
updater	10 077	1 823	2024-04-28 11:30	2024-04-28 11:50	2024-04-28 11:50
x64	4 539 496	2 072 783	2024-04-28 11:30	2024-04-28 11:50	2024-04-28 11:50
x86	424 560	201 462	2024-04-28 11:30	2024-04-28 11:50	2024-04-28 11:50
billet.php	32 320	20 016	2024-04-28 11:26	2024-04-28 11:50	2024-04-28 11:50
msvcp140.dll	437 448	155 548	2024-04-28 11:26	2024-04-28 11:50	2024-04-28 11:50
potsherd.psd	6 343 958	6 105 670	2024-04-28 11:26	2024-04-28 11:50	2024-04-28 11:50
Setup.exe	960 832	419 037	2024-04-28 11:26	2024-04-28 11:50	2024-04-28 11:50
vcruntime140.dll	83 144	45 917	2024-04-28 11:26	2024-04-28 11:50	2024-04-28 11:50
WCLDII.dll	604 480	297 716	2024-04-28 11:26	2024-04-28 11:50	2024-04-28 11:50

Figure 3. Content of the ZIP file

The archive contains multiple folders, one executable, several DLL files and two odd files.

All the folders contain seemingly random files varying from JSON or Javascript files to legitimate DLL files, they don't seem to be used in the infection chain.

Two files catch our eye, "billet.php" and "potsherd.psd". According to the file extensions they should be a PHP source

file and an Adobe Photoshop image document. However, none of the two is a valid file, with the PHP one being especially suspicious since PHP is supposed to be a source code format, but the file contains only binary data.

Finally, we look at the executable files, as previously stated, the Setup.exe file is a legitimate and signed installer for a Cisco product related to Webex, the file signature can be seen in Figure 4.

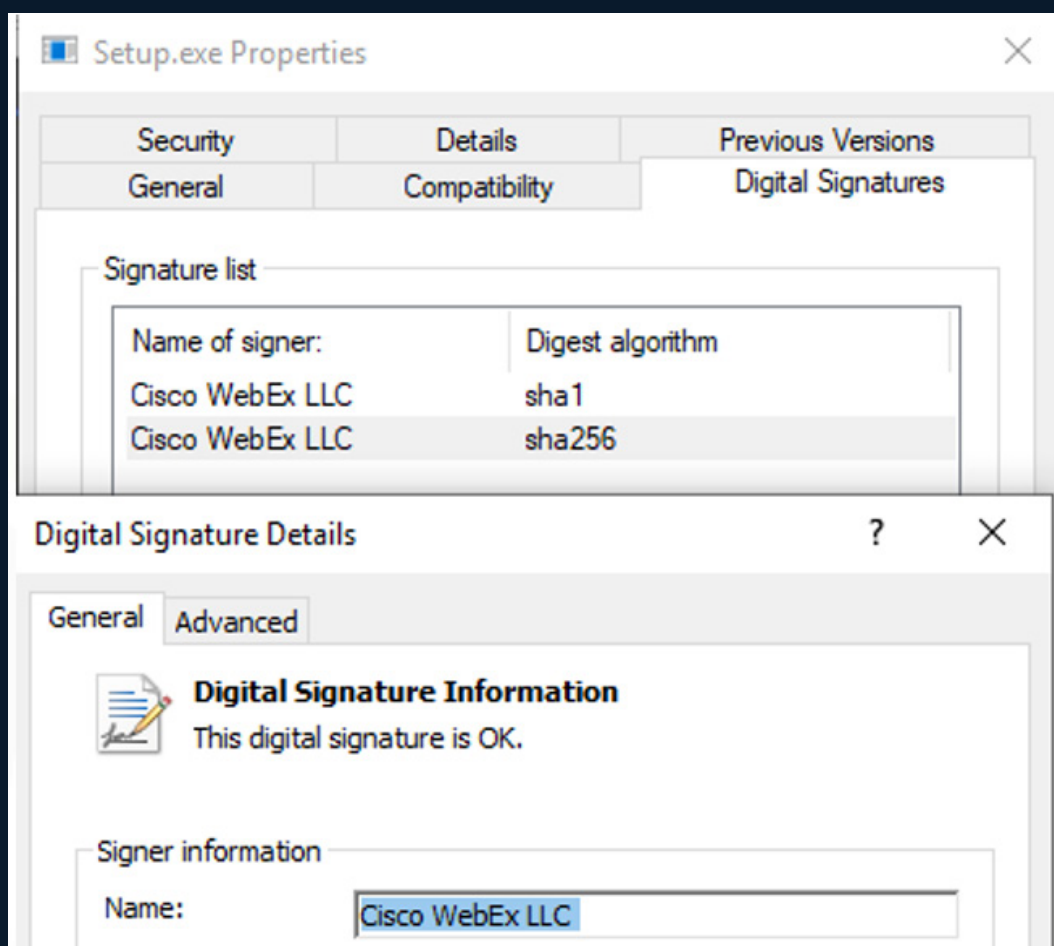


Figure 4. Setup.exe file signature

While all the other DLL files have a signature as well, the one in "WCLDII.dll" is not valid. This indicates that this file has been manipulated and it is the most likely candidate for the file that actually holds the malicious payload.

3.2 DLL Side-loading

DLL Side-loading is a cyber attack method which exploits the way Microsoft Windows handles loading Dynamic Link Library (DLL) files. As it is already well-known, Windows applications may load

additional code from DLL files at runtime, this can be done by manually calling the LoadLibrary function or automatically through the imports directory of the PE file.

When an application loads a DLL without specifying the exact location, Windows attempts to locate the DLL by searching a predefined set of directories in a specific order⁵.

As soon as a DLL file is loaded it can execute code as part of the DllMain entrypoint, allowing attackers to execute code even if the DLL does not match what the target program is expecting.

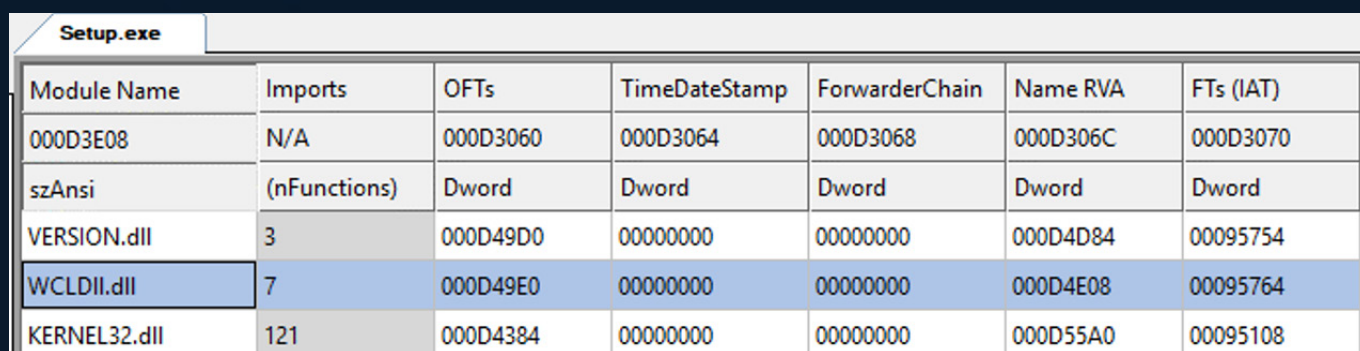
The default configuration for most applications is to prioritise DLL files present in the current working directory or the same directory as the EXE file itself.

An attacker with knowledge of the application's behaviour can manipulate this process by placing malicious DLL files in the directories from where they will be loaded (or replacing existing ones). This results in the payload's actions being hidden under the execution of a legitimate process, making the malicious activity harder to detect.

This technique poses significant challenges for both users and Endpoint Detection and Response (EDR) solutions due to its evasive nature. Traditional EDR systems may not flag this activity during execution because it appears to be a legitimate procedure. Similarly, for a user auditing the process, the main EXE file would look correctly signed in tools like the Windows task manager or process explorer.

Moreover, the obfuscation employed on the DLL further complicates detection efforts.

DLL Side-loading usually is used to take over a system DLL by loading it from the application directory, in this case WCLDll.dll is a component of the setup and a direct dependency in the import table of the EXE, as shown in Figure 5, meaning that even in the legitimate case it will be loaded automatically from the application folder by Windows during the creation of the process.



Module Name	Imports	OFTs	TimeDateStamp	ForwarderChain	Name RVA	FTs (IAT)
000D3E08	N/A	000D3060	000D3064	000D3068	000D306C	000D3070
szAnsi	(nFunctions)	Dword	Dword	Dword	Dword	Dword
VERSION.dll	3	000D49D0	00000000	00000000	000D4D84	00095754
WCLDll.dll	7	000D49E0	00000000	00000000	000D4E08	00095764
KERNEL32.dll	121	000D4384	00000000	00000000	000D55A0	00095108

Figure 5. The import directory of Setup.exe

⁵ <https://learn.microsoft.com/en-us/windows/win32/dlls/dynamic-link-library-search-order>

3.3 Technical analysis and behaviour

After a quick analysis we confirmed that the malicious code has been injected in the C runtime code of WCLDII, specifically in the “__scrt_release_startup_lock” function. This function is reached from the DllMain function called by the Windows loader as part of the normal DLL file lifecycle.

The payload starts by loading billet.php using normal File I/O APIs. The file path is obtained by calling GetModuleFileNameW⁶ of the current process and then replacing everything after the last ‘\’ with the “billet.php” string.

Interestingly, the strings used by the loader are located in an odd area of the binary: in the middle of the RTTI tables. RTTI (RunTime Type Information) is metadata added by the compiler about object-oriented data structures of the code. This is used among other things for debugging exceptions. The presence of strings unrelated to RTTI here indicates manipulation of the binary, which is shown in figure 6.

```
.data:74ED7834 ; public struct IUnknown /* mdisp:0 */
.data:74ED7834 ; struct AT::GXText `RTTI Type Descriptor'
.data:74ED7834 ??_R0?AUGXText_@AT@@@8 dd offset ??_7type_info@66B@
.data:74ED7834 ; DATA XREF: .rdata:74EBF770fo
.data:74ED7834 ; .pdata:74E7834:GXText_? RTTI Base Class Descriptor at (0,-1,0,64)'fo
.data:74ED7834 ; reference to RTTI's vftable
.data:74ED7838 dd 0 ; internal runtime reference
.data:74ED783C aAugxtextAt db '.?AUGXText_@AT@@',0 ; type descriptor name
.data:74ED784E align 10h
.data:74ED7850 unk_74ED7850 db 90h ; DATA XREF: .rdata:74EC75FCfo
.data:74ED7850 ; .pdata:off_74EC78BCfo
.data:74ED7851 db 69h, 6, 8 dup(0), 0Ch, 4 dup(0), 20h, 4Bh, 0, 8, 66h ; DATA XREF: DynamicImports9fo
.data:74ED7851 ; DATA XREF: DynamicImports9fo
.data:74ED7865 db 0Ah dup(0)
.data:74ED786F dd offset aKernel32Dll ; "KERNEL32.dll"
.data:74ED7873 dd offset aPotsherdPsd ; "potsherd.psd"
.data:74ED7877 db 89h, 09h dup(0)
.data:74ED7887 dd offset unk_74E50000
.data:74ED7888 db 5 dup(0), 000h, 7, 5 dup(0), 8, 3 dup(0)
.data:74ED7888 ; DATA XREF: .rdata:off_74EC2D08fo
.data:74ED7890 dd offset aBilletPhp ; "billet.php"
.data:74ED789F aObjAt db 'Obj@AT@@',0
.data:74ED78A8 ; public class ATL::ComEnumStruct IEnumFORMATETC, &struct _GUID const IID_IEnumFORMATETC, struct tagFORMATETC, class ATL::Copy<struct tagFORMATETC>, class ATL::ComMultiThreadModel> /* mdisp:0 */ /:
.data:74ED78A8 ; public class ATL::ComEnumImpl<struct IEnumFORMATETC, &struct _GUID const IID_IEnumFORMATETC, struct tagFORMATETC, class ATL::Copy<struct tagFORMATETC>> /* mdisp:0 */ /:
.data:74ED78A8 ; public struct IEnumFORMATETC /* mdisp:0 */ /:
.data:74ED78A8 ; public struct IUnknown /* mdisp:0 */ /;
```

Figure 6. Payload strings between RTTI data structures

⁶ <https://learn.microsoft.com/it-it/windows/win32/api/libloaderapi/nf-libloaderapi-getmodulefilenamew>

Once `billet.php` is loaded, it is decrypted with a simple addition-cypher. The payload contains malicious shellcode and an initial configuration. We will refer to the shellcode from `bilet.php` as stage 2.

The shellcode is executed by loading a legitimate DLL file from the configuration with `LoadLibrary` and replacing parts of it using `VirtualProtect`. In this sample the target library is `dbghelp.dll`.

This technique allows spoofing stack frames. When executing arbitrary shellcode allocated via `VirtualAlloc`, the call stack will contain return addresses that don't belong to any module, however by first loading a DLL file and then replacing its content one can hide traces left in the stack trace.

When analysing shellcode, the preferred way is to extract it into a standalone binary so it can be emulated or debugged in isolation from the application it is injected into. In this case the shellcode requires a series of arguments from the caller which make applying this setup more challenging. Stage 2 will then load another Stage 3 payload in a similar way.

Stage 2 is a simple self-contained loader which resolves a series of Windows APIs by walking the loader data structures from the PEB. The libraries and functions

it needs are identified by the hash of the name; the hashing algorithm is very simple:

- For DLL names it is the sum of all the characters of the name converted to lowercase;
- For function names it also includes a constant from the configuration in the calculation, this prevents fingerprinting and automated static analysis looking for known constants.

Once all the needed functions have been resolved, the loader will read the content of the `potshred.psd` file, decrypt it with a XOR-based algorithm and call the system `RtlDecompressBuffer` function to decompress it. The compression format is `COMPRESSION_FORMAT_LZNT1`.

The stage 3 file name comes from the arguments passed by the initial loader.

Once stage 3 has been loaded and decrypted, it is executed in the same way as stage 2: by loading an arbitrary library from its configuration and replacing parts of it using `VirtualProtect` and then `memcpy`.

Stage 3 is a much more complex piece of code, using static signatures we could identify it as `HijackLoader`.

This time dynamic functions are imported using CRC32 hashes which makes identifying the malware family with signatures much easier, furthermore we could use the IDA HashDB⁷ extension to quickly triage the sample by labelling all the function it imports, some of which can be seen in figure 7.

```
v150 = GetComputerNameW_0;  
v205 = ZwGetContextThread_0;  
v239 = ZwProtectVirtualMemory_0;  
v238 = ZwWriteVirtualMemory_0;  
v237 = ZwResumeThread_0;  
v236 = NtSuspendThread_0;  
v235 = NtSetContextThread_0;  
v93 = RtlWow64GetThreadContext;  
v92 = RtlWow64SetThreadContext;  
v142 = ZwCreateSection_0;  
v141 = ZwMapViewOfSection_0;  
v140 = ZwOpenProcess_0;  
v139 = WinHttpOpen_0;
```

Figure 7. Some of the name hashes that were identified by HashDB

Once we ascertained the nature of the loader, we used dynamic analysis to explode it and capture the delivered payload using memory dumps.

Over multiple runs from different IP addresses, we observed that it consistently drops Vidar Stealer, however in one specific instance it also deployed the xmrig cryptominer through the Amadey botnet.

⁷ <https://github.com/OALabs/hashdb>

3.4 IOC

In the next table we inserted IoC of the 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	52b44eaf40d4c53717d22875834b57c94d4c7625c69028 bbbdd77b0ea9ae6ab9	WCLDII.dll VirusTotal – 10/71
SHA-256	c837fc63f95adeaae0ea2231376f525fd10f927f2c28 cc1cd7feda1edba7d82b	billet.php VirusTotal – 0/62
SHA-256	bdf0dfdf3690733c1af3d84f70120c5f12a4 b89fa92b7151ed3845e7f969822	potsherd.psd VirusTotal – 0/62
Domain	u87yuo9ojh[.]world	VirusTotal – 0/92 AlienVault
Domain	n76yuhui86[.]world	VirusTotal – 0/92 AlienVault
Domain	vgy6yhji9[.]world	VirusTotal – 4/92 AlienVault
Domain	graims[.]xyz	VirusTotal – 14/92 AlienVault
Domain	rtattack[.]ralyjya9[.]online	VirusTotal – 16/92 AlienVault
Domain	bestfitnessgymintheworld[.]com	VirusTotal – 6/92 AlienVault

Table 1. Indicators of compromise

4

Conclusions

4. Conclusions

This report highlights the complexity of the HijackLoader's DLL side-loading technique for stealthy payload delivery. Mitigating these types of attacks requires a versatile approach such as user training and awareness which are crucial in educating them about the risks of downloading and executing files from untrusted sources.

However, the most important factor is to implement strong application control, which can prevent unauthorised softwa-

re, including malicious DLLs, from executing. Moreover, enhancing Endpoint Detection and Response (EDR) capabilities to include advanced behavioural analysis can help detect anomalies associated with DLL side-loading and process hollowing.

Finally, regular software updates and patching reduce known vulnerabilities that attackers might exploit to infect the system.



DEFENCE TECH

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



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