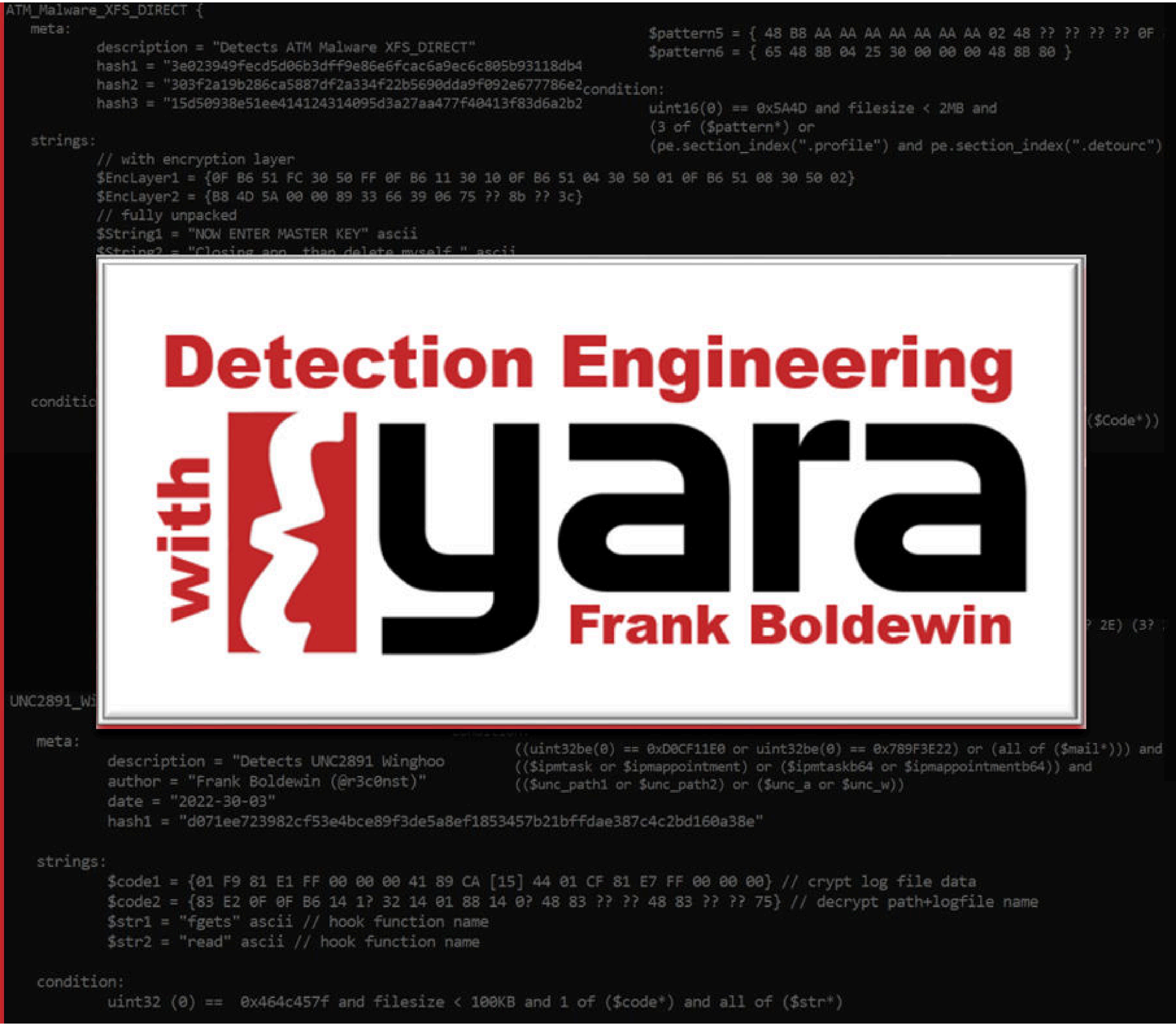
Detection Engineering with yara

Frank Boldewin



```
ATM_Malware_XFS_DIRECT {
   meta:
         description = "Detects ATM Malware XFS_DIRECT"
         hash1 = "3e023949fecd5d06b3dff9e86e6fcac6a9ec6c805b93118db4
         hash2 = "303f2a19b286ca5887df2a334f22b5690dda9f092e677786e2
         hash3 = "15d50938e51ee414124314095d3a27aa477f40413f83d6a2b2

   strings:
         // with encryption layer
         $EncLayer1 = {0F B6 51 FC 30 50 FF 0F B6 11 30 10 0F B6 51 04 30 50 01 0F B6 51 08 30 50 02}
         $EncLayer2 = {B8 4D 5A 00 00 89 33 66 39 06 75 ?? 8b ?? 3c}
         // fully unpacked
         $String1 = "NOW ENTER MASTER KEY" ascii
         $String2 = "Closing app, than delete myself." ascii

   conditic
```

```
         $pattern5 = { 48 B8 AA AA AA AA AA AA AA 02 48 ?? ?? ?? ?? 0F
         $pattern6 = { 65 48 8B 04 25 30 00 00 00 48 8B 80 }

condition:
         uint16(0) == 0x5A4D and filesize < 2MB and
         (3 of ($pattern*) or
         (pe.section_index(".profile") and pe.section_index(".detourc")
```

```
($Code*))
```

```
? 2E) (3?
```

```
UNC2891_Wi

   meta:
         description = "Detects UNC2891 Winghoo
         author = "Frank Boldewin (@r3c0nst)"
         date = "2022-30-03"
         hash1 = "d071ee723982cf53e4bce89f3de5a8ef1853457b21bffdae387c4c2bd160a38e"

   strings:
         $code1 = {01 F9 81 E1 FF 00 00 00 41 89 CA [15] 44 01 CF 81 E7 FF 00 00 00} // crypt log file data
         $code2 = {83 E2 0F 0F B6 14 1? 32 14 01 88 14 0? 48 83 ?? ?? 48 83 ?? ?? 75} // decrypt path+logfile name
         $str1 = "fgets" ascii // hook function name
         $str2 = "read" ascii // hook function name

   condition:
         uint32 (0) ==  0x464c457f and filesize < 100KB and 1 of ($code*) and all of ($str*)
```

```
((uint32be(0) == 0xD0CF11E0 or uint32be(0) == 0x789F3E22) or (all of ($mail*))) and
((($ipmtask or $ipmappointment) or ($ipmtaskb64 or $ipmappointmentb64)) and
(($unc_path1 or $unc_path2) or ($unc_a or $unc_w))
```

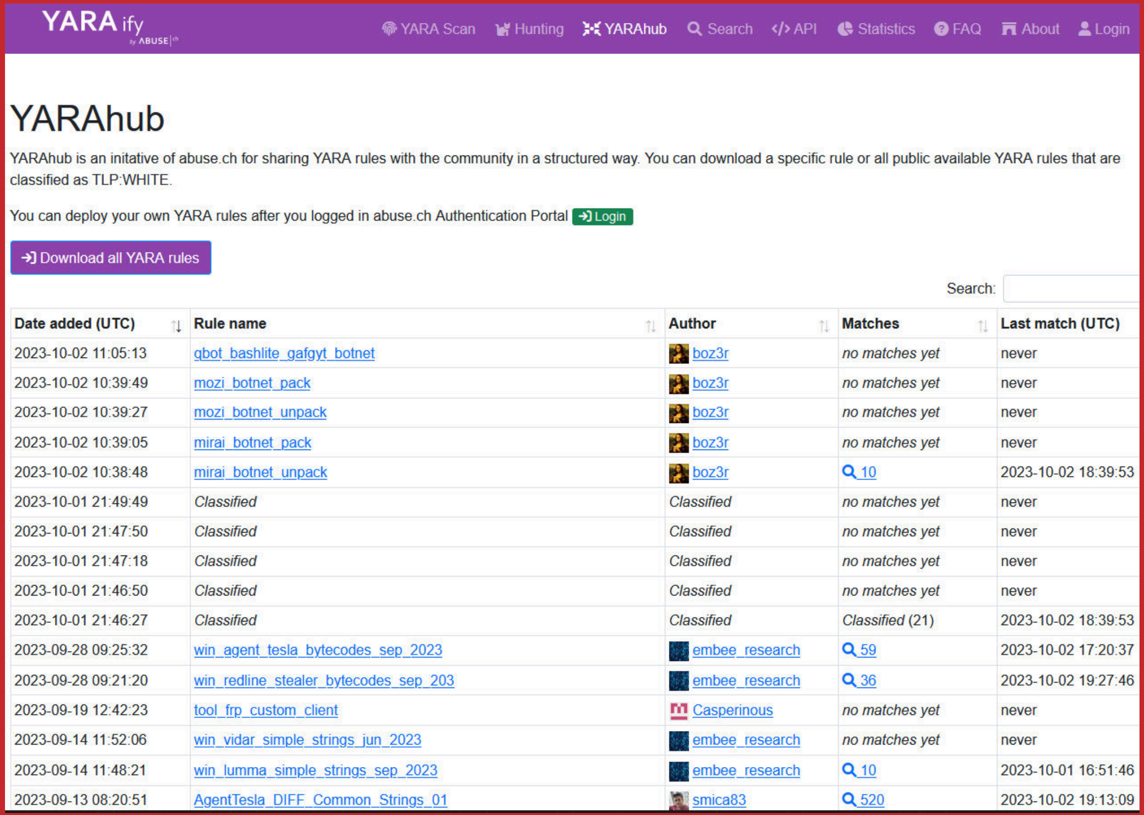
What is YARA?

- **YARA** is **aimed** to **identify** and **classify malware**.
- Rules **consist of strings or values and** a **boolean expression** which **determine** their **logic**.
- They are used for **threat hunting**, as well as for **incident response** and **compromise assessment** investigations. This involves **searching** for corresponding **artifacts** (IoCs) in **memory** and on **hard disks** using a special scanner which utilizes the YARA API.
- In addition, YARA **is being used in a range of cybersecurity products**, **tools** and **services**.
- YARA is **supported on multiple platforms** and is **open source**.

- **References:**
  - https://github.com/VirusTotal/yara/releases
  - https://buildmedia.readthedocs.org/media/pdf/yara/latest/yara.pdf

Popular YARA rules repositories

https://yaraify.abuse.ch/yarahub/



YARAhub

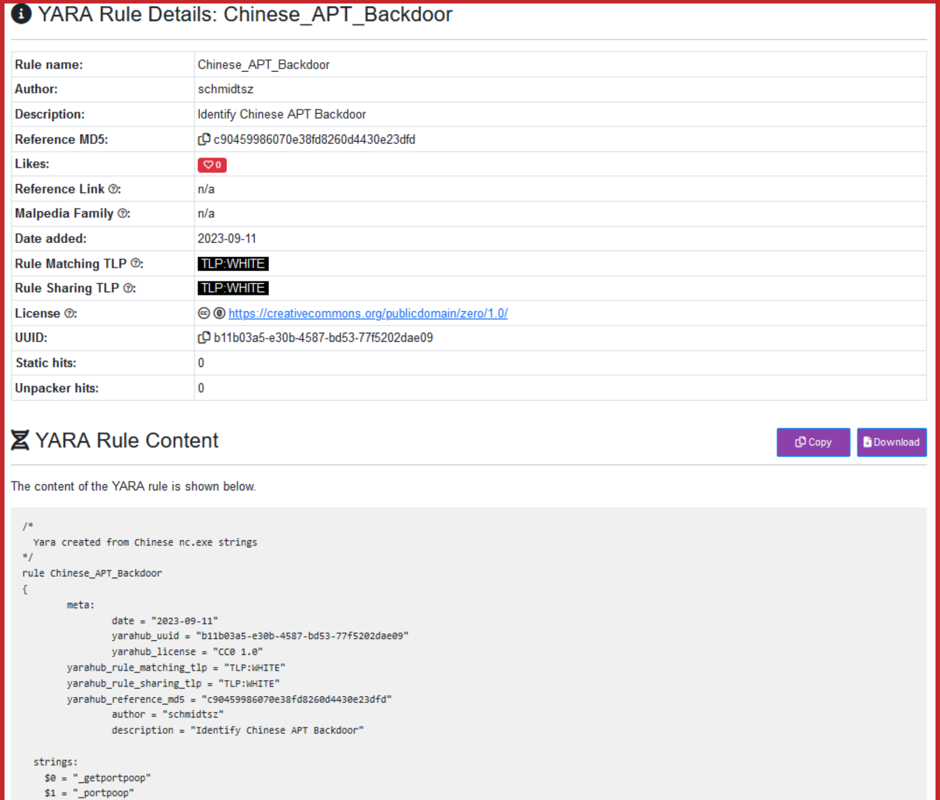
YARAhub is an initative of abuse.ch for sharing YARA rules with the community in a structured way. You can download a specific rule or all public available YARA rules that are classified as TLP:WHITE.

You can deploy your own YARA rules after you logged in abuse.ch Authentication Portal Login

Download all YARA rules

Search:

Date added (UTC)	Rule name	Author	Matches	Last match (UTC)
2023-10-02 11:05:13	qbot_bashlite_gafgyt_botnet	boz3r	*no matches yet*	never
2023-10-02 10:39:49	mozi_botnet_pack	boz3r	*no matches yet*	never
2023-10-02 10:39:27	mozi_botnet_unpack	boz3r	*no matches yet*	never
2023-10-02 10:39:05	mirai_botnet_pack	boz3r	*no matches yet*	never
2023-10-02 10:38:48	mirai_botnet_unpack	boz3r	10	2023-10-02 18:39:53
2023-10-01 21:49:49	*Classified*	*Classified*	*no matches yet*	never
2023-10-01 21:47:50	*Classified*	*Classified*	*no matches yet*	never
2023-10-01 21:47:18	*Classified*	*Classified*	*no matches yet*	never
2023-10-01 21:46:50	*Classified*	*Classified*	*no matches yet*	never
2023-10-01 21:46:27	*Classified*	*Classified*	*Classified* (21)	2023-10-02 18:39:53
2023-09-28 09:25:32	win_agent_tesla_bytecodes_sep_2023	embee_research	59	2023-10-02 17:20:37
2023-09-28 09:21:20	win_redline_stealer_bytecodes_sep_203	embee_research	36	2023-10-02 19:27:46
2023-09-19 12:42:23	tool_frp_custom_client	Casperinous	*no matches yet*	never
2023-09-14 11:52:06	win_vidar_simple_strings_jun_2023	embee_research	*no matches yet*	never
2023-09-14 11:48:21	win_lumma_simple_strings_sep_2023	embee_research	10	2023-10-01 16:51:46
2023-09-13 08:20:51	AgentTesla_DIFF_Common_Strings_01	smica83	520	2023-10-02 19:13:09



YARA Rule Details: Chinese_APT_Backdoor

Rule name:	Chinese_APT_Backdoor
Author:	schmidtsz
Description:	Identify Chinese APT Backdoor
Reference MD5:	c90459986070e38fd8260d4430e23dfd
Likes:	♡0
Reference Link ⓘ:	n/a
Malpedia Family ⓘ:	n/a
Date added:	2023-09-11
Rule Matching TLP ⓘ:	TLP:WHITE
Rule Sharing TLP ⓘ:	TLP:WHITE
License ⓘ:	https://creativecommons.org/publicdomain/zero/1.0/
UUID:	b11b03a5-e30b-4587-bd53-77f5202dae09
Static hits:	0
Unpacker hits:	0

YARA Rule Content

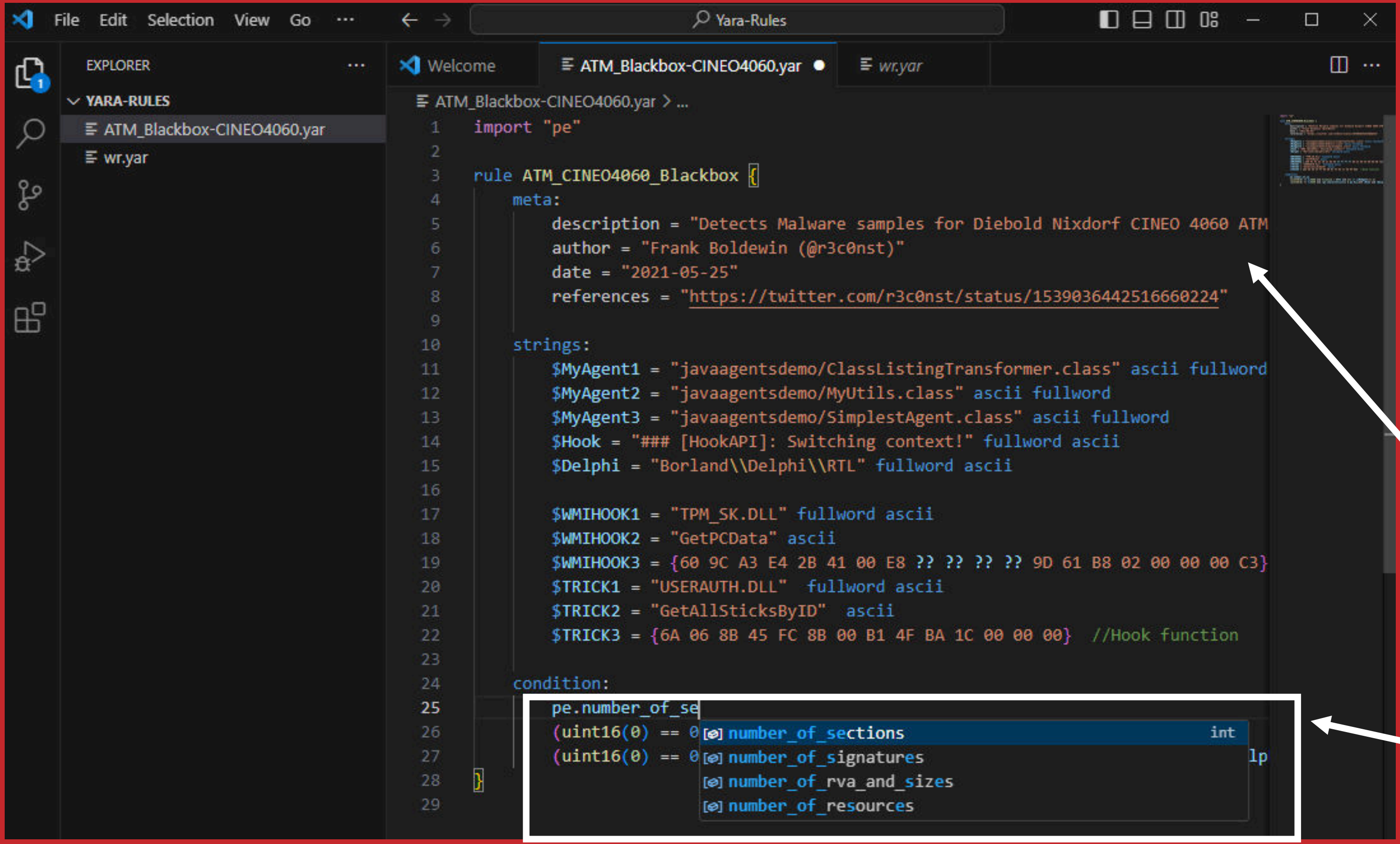
The content of the YARA rule is shown below.

```
/*
  Yara created from Chinese nc.exe strings
*/
rule Chinese_APT_Backdoor
{
        meta:
                date = "2023-09-11"
                yarahub_uuid = "b11b03a5-e30b-4587-bd53-77f5202dae09"
                yarahub_license = "CC0 1.0"
        yarahub_rule_matching_tlp = "TLP:WHITE"
        yarahub_rule_sharing_tlp = "TLP:WHITE"
        yarahub_reference_md5 = "c90459986070e38fd8260d4430e23dfd"
                author = "schmidtsz"
                description = "Identify Chinese APT Backdoor"

  strings:
    $0 = "_getportpoop"
    $1 = "_portpoop"
```

- **Other YARA rules repositories:**
  - https://github.com/Neo23x0/signature-base/tree/master/yara https://github.com/reversinglabs/reversinglabs-yara-rules/tree/develop/yara
  - https://github.com/100DaysofYARA/2023/

YARA rule editor – YARA Language Server (YLS)

https://engineering.avast.io/yls-first-step-towards-yara-development-environment/



```
import "pe"

rule ATM_CINEO4060_Blackbox {
    meta:
        description = "Detects Malware samples for Diebold Nixdorf CINEO 4060 ATM
        author = "Frank Boldewin (@r3c0nst)"
        date = "2021-05-25"
        references = "https://twitter.com/r3c0nst/status/1539036442516660224"

    strings:
        $MyAgent1 = "javaagentsdemo/ClassListingTransformer.class" ascii fullword
        $MyAgent2 = "javaagentsdemo/MyUtils.class" ascii fullword
        $MyAgent3 = "javaagentsdemo/SimplestAgent.class" ascii fullword
        $Hook = "### [HookAPI]: Switching context!" fullword ascii
        $Delphi = "Borland\\Delphi\\RTL" fullword ascii

        $WMIHOOK1 = "TPM_SK.DLL" fullword ascii
        $WMIHOOK2 = "GetPCData" ascii
        $WMIHOOK3 = {60 9C A3 E4 2B 41 00 E8 ?? ?? ?? ?? 9D 61 B8 02 00 00 00 C3}
        $TRICK1 = "USERAUTH.DLL"  fullword ascii
        $TRICK2 = "GetAllSticksByID"  ascii
        $TRICK3 = {6A 06 8B 45 FC 8B 00 B1 4F BA 1C 00 00 00}  //Hook function

    condition:
        pe.number_of_se
        (uint16(0) == 0
        (uint16(0) == 0                                          lp
}
```

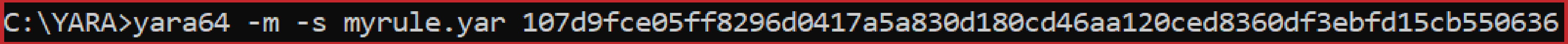
Edit your rules inside Microsoft VS-Code.

YARA language support Including autocompletion.

INSTALLATION INSTRUCTIONS

1. pip install yls-yara
2. Download + Install VS-Code (https://code.visualstudio.com/Download)
3. Install YLS Extension inside VS-Code
4. Pip install yari
   Add debugging support with YARI (https://engineering.avast.io/yari-a-new-era-of-yara-debugging/)

YARA cli scanner basics 1/3



```
C:\YARA>yara64 -m -s myrule.yar 107d9fce05ff8296d0417a5a830d180cd46aa120ced8360df3ebfd15cb550636
```

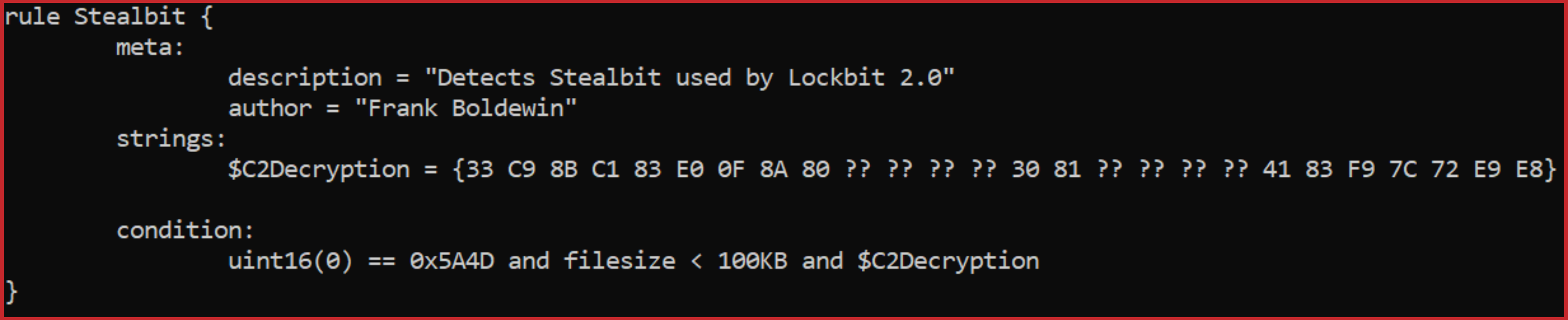
Yara cli scanner

Print metadata of rule

Print matching strings

Yara rule

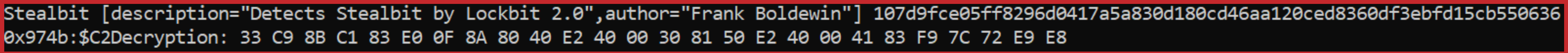
File to scan



```
rule Stealbit {
        meta:
                description = "Detects Stealbit used by Lockbit 2.0"
                author = "Frank Boldewin"
        strings:
                $C2Decryption = {33 C9 8B C1 83 E0 0F 8A 80 ?? ?? ?? ?? 30 81 ?? ?? ?? ?? 41 83 F9 7C 72 E9 E8}

        condition:
                uint16(0) == 0x5A4D and filesize < 100KB and $C2Decryption
}
```

Matching meta data information

Scan results

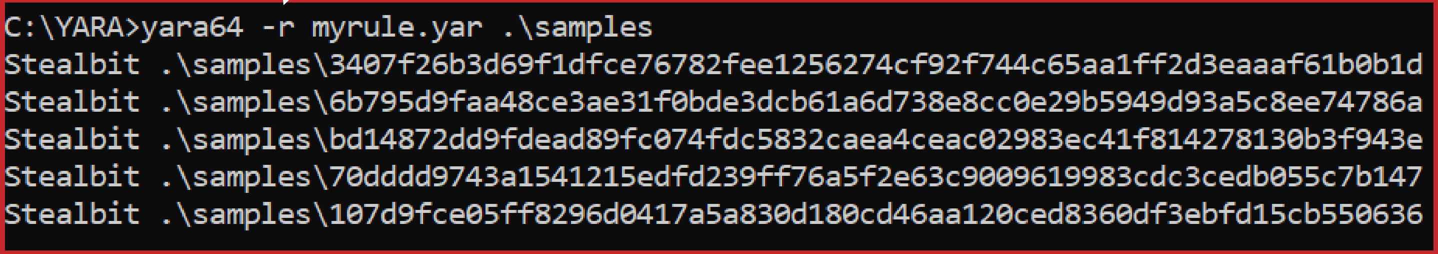


```
Stealbit [description="Detects Stealbit by Lockbit 2.0",author="Frank Boldewin"] 107d9fce05ff8296d0417a5a830d180cd46aa120ced8360df3ebfd15cb550636
0x974b:$C2Decryption: 33 C9 8B C1 83 E0 0F 8A 80 40 E2 40 00 30 81 50 E2 40 00 41 83 F9 7C 72 E9 E8
```

Matching strings information (hex values are treated like text)

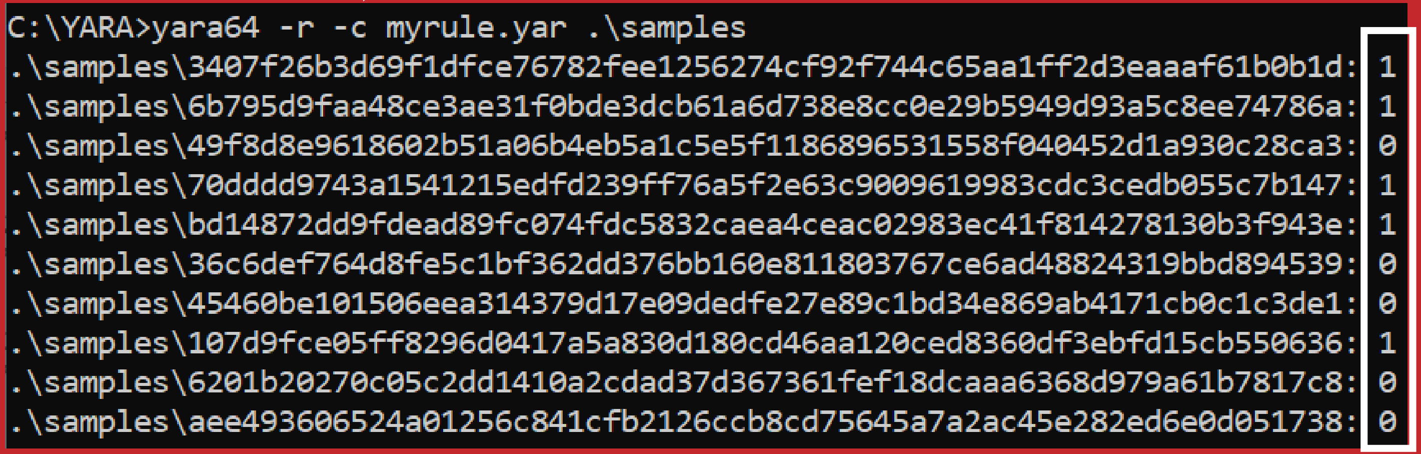
YARA cli scanner basics 2/3

Scan recursively through directories with parameter -r



```
C:\YARA>yara64 -r myrule.yar .\samples
Stealbit .\samples\3407f26b3d69f1dfce76782fee1256274cf92f744c65aa1ff2d3eaaaf61b0b1d
Stealbit .\samples\6b795d9faa48ce3ae31f0bde3dcb61a6d738e8cc0e29b5949d93a5c8ee74786a
Stealbit .\samples\bd14872dd9fdead89fc074fdc5832caea4ceac02983ec41f814278130b3f943e
Stealbit .\samples\70dddd9743a1541215edfd239ff76a5f2e63c9009619983cdc3cedb055c7b147
Stealbit .\samples\107d9fce05ff8296d0417a5a830d180cd46aa120ced8360df3ebfd15cb550636
```

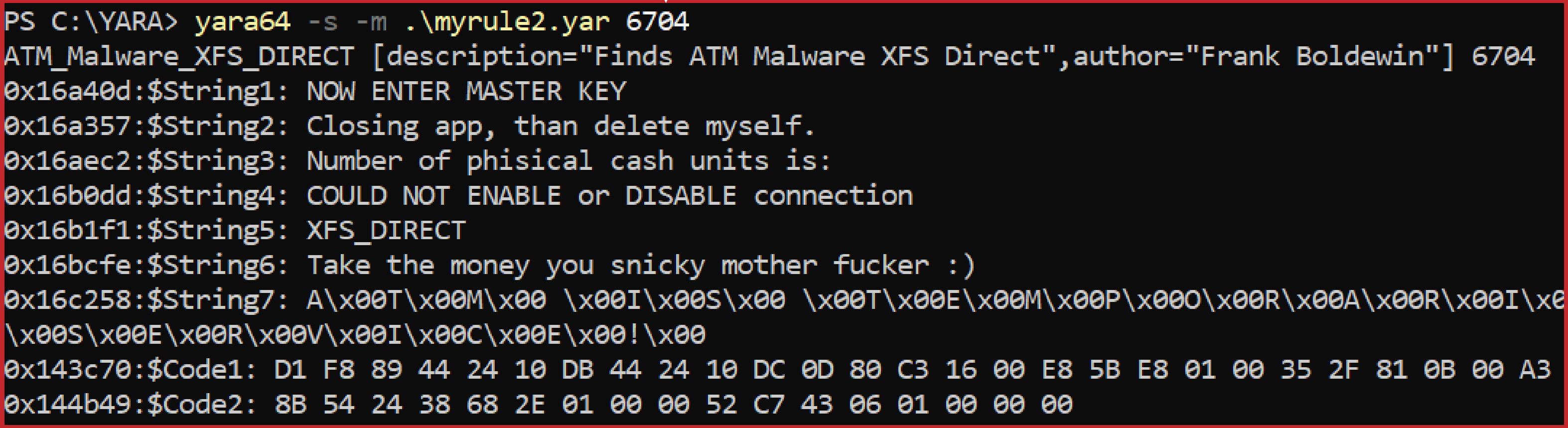
Just print rule matches (0 - no match / 1 - match)



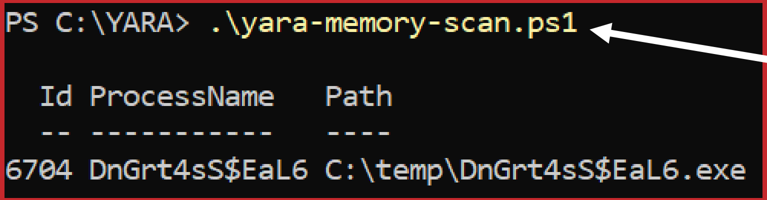
```
C:\YARA>yara64 -r -c myrule.yar .\samples
.\samples\3407f26b3d69f1dfce76782fee1256274cf92f744c65aa1ff2d3eaaaf61b0b1d: 1
.\samples\6b795d9faa48ce3ae31f0bde3dcb61a6d738e8cc0e29b5949d93a5c8ee74786a: 1
.\samples\49f8d8e9618602b51a06b4eb5a1c5e5f1186896531558f040452d1a930c28ca3: 0
.\samples\70dddd9743a1541215edfd239ff76a5f2e63c9009619983cdc3cedb055c7b147: 1
.\samples\bd14872dd9fdead89fc074fdc5832caea4ceac02983ec41f814278130b3f943e: 1
.\samples\36c6def764d8fe5c1bf362dd376bb160e811803767ce6ad48824319bbd894539: 0
.\samples\45460be101506eea314379d17e09dedfe27e89c1bd34e869ab4171cb0c1c3de1: 0
.\samples\107d9fce05ff8296d0417a5a830d180cd46aa120ced8360df3ebfd15cb550636: 1
.\samples\6201b20270c05c2dd1410a2cdad37d367361fef18dcaaa6368d979a61b7817c8: 0
.\samples\aee493606524a01256c841cfb2126ccb8cd75645a7a2ac45e282ed6e0d051738: 0
```

YARA cli scanner basics 3/3

Scan process memory of specific PID

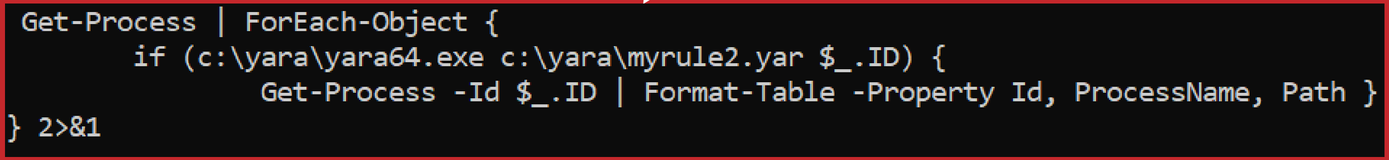


```
PS C:\YARA> yara64 -s -m .\myrule2.yar 6704
ATM_Malware_XFS_DIRECT [description="Finds ATM Malware XFS Direct",author="Frank Boldewin"] 6704
0x16a40d:$String1: NOW ENTER MASTER KEY
0x16a357:$String2: Closing app, than delete myself.
0x16aec2:$String3: Number of phisical cash units is:
0x16b0dd:$String4: COULD NOT ENABLE or DISABLE connection
0x16b1f1:$String5: XFS_DIRECT
0x16bcfe:$String6: Take the money you snicky mother fucker :)
0x16c258:$String7: A\x00T\x00M\x00 \x00I\x00S\x00 \x00T\x00E\x00M\x00P\x00O\x00R\x00A\x00R\x00I\x0
\x00S\x00E\x00R\x00V\x00I\x00C\x00E\x00!\x00
0x143c70:$Code1: D1 F8 89 44 24 10 DB 44 24 10 DC 0D 80 C3 16 00 E8 5B E8 01 00 35 2F 81 0B 00 A3
0x144b49:$Code2: 8B 54 24 38 68 2E 01 00 00 52 C7 43 06 01 00 00 00
```



```
PS C:\YARA> .\yara-memory-scan.ps1

  Id ProcessName   Path
  -- -----------   ----
6704 DnGrt4sS$EaL6 C:\temp\DnGrt4sS$EaL6.exe
```

Powershell script to scan process memory of all PIDs accessible (depending on user access rights)

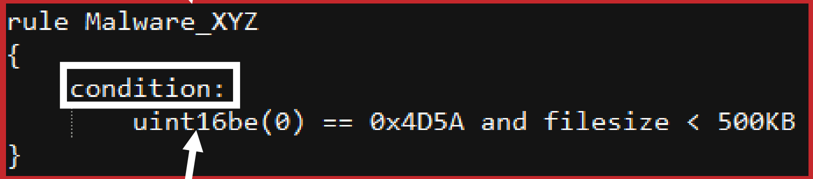


```
 Get-Process | ForEach-Object {
        if (c:\yara\yara64.exe c:\yara\myrule2.yar $_.ID) {
                Get-Process -Id $_.ID | Format-Table -Property Id, ProcessName, Path }
} 2>&1
```

YARA rules 101 - Condition and strings section

Yara consists **essentially** of **two sections: strings** and **conditions**. However, the **latter** is the **only necessary** one to generate a **valid rule**.

Name of the rule

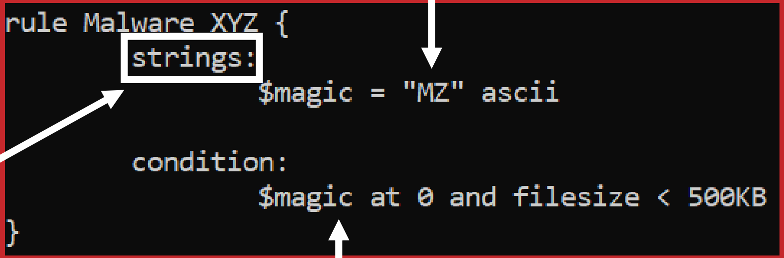


```
rule Malware_XYZ
{
    condition:
        uint16be(0) == 0x4D5A and filesize < 500KB
}
```

Condition → Find matches for PE file header magic 0x4D5A (MZ) at offset 0 and file must be smaller than 500KB

uint16be → unsigned int value big endian

MZ must be ascii and is case sensitive



```
rule Malware_XYZ {
        strings:
                $magic = "MZ" ascii

        condition:
                $magic at 0 and filesize < 500KB
}
```

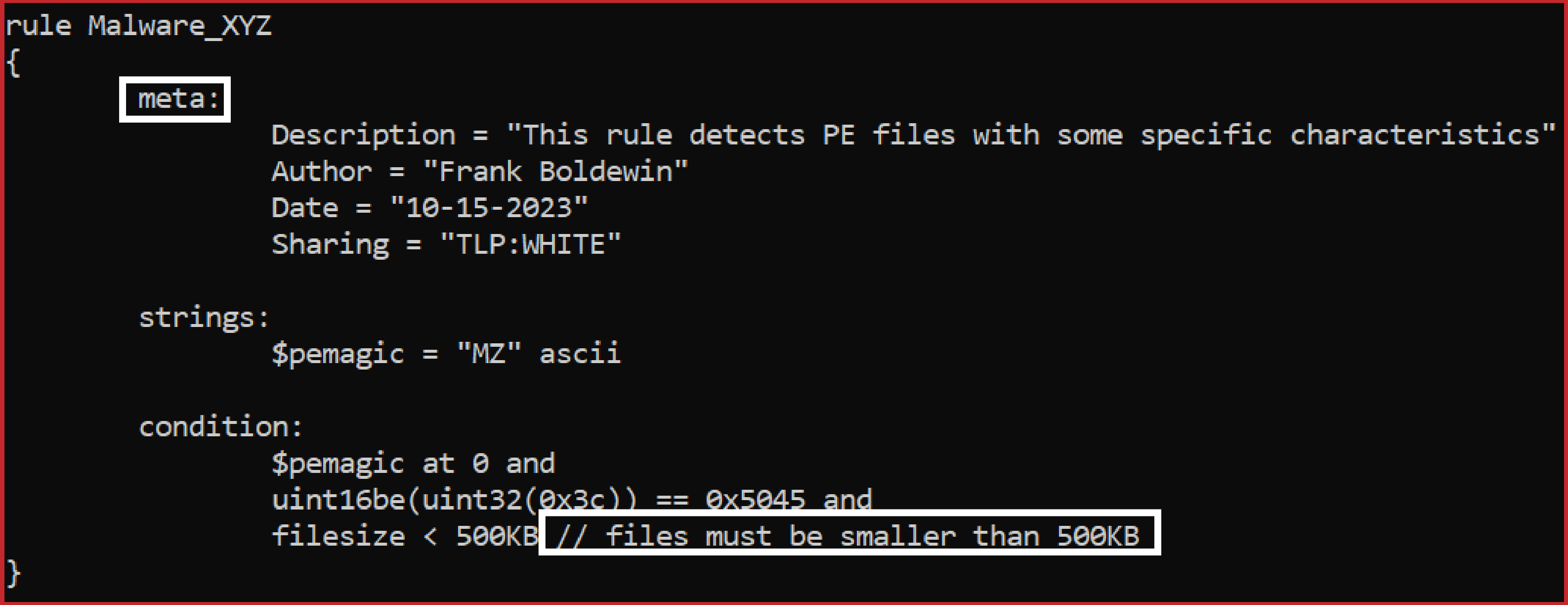
Same condition but defining the header magic in the strings section

„at 0“ is the same as „(0)“ shown on the left side

Check links below for more on keywords and conditions:
https://yara.readthedocs.io/en/stable/writingrules.html#table-1
https://yara.readthedocs.io/en/stable/writingrules.html#conditions

YARA rules 101 - Meta section and comments

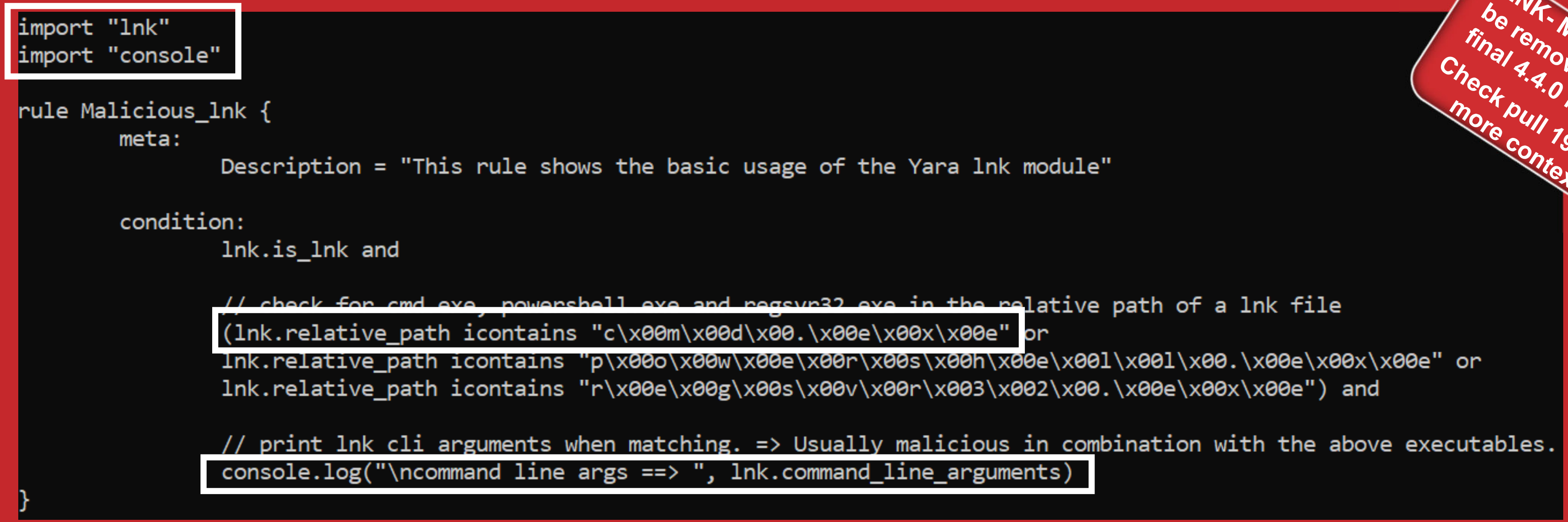
- YARA **rules** are often **shared within** a **team or** a broader **community**. Therefore, it is **good practice** to **leave** relevant **information about** the developed **rules for third parties**.
- In YARA, **such information** is **stored in** the "**meta**" **section**. A look at various public YARA repos shows that this **information may vary**, as it is **not standardized** and the **YARA engine doesn't check** for **certain keywords**. However, **some keywords** such as Description, Author, Date, Reference, Hash, Sharing resp. Classification are well **established within** the **community**.
- These and other **keywords** within the meta section are also **often checked as part of automated workflows**, such as scanning engines to process only certain rules, sort them by date, or classify the rules.
- **To describe specific parts in** a YARA **rule comments can be used** → // This is a comment



```
rule Malware_XYZ
{
        meta:
                Description = "This rule detects PE files with some specific characteristics"
                Author = "Frank Boldewin"
                Date = "10-15-2023"
                Sharing = "TLP:WHITE"

        strings:
                $pemagic = "MZ" ascii

        condition:
                $pemagic at 0 and
                uint16be(uint32(0x3c)) == 0x5045 and
                filesize < 500KB // files must be smaller than 500KB
}
```

YARA rules 101 - Introduction to modules 1/2

- In addition to its standard functions, **YARA provides feature extensions**, **so called modules**.
- By using these extensions, **more complex rules can be developed**.
- **Use** the **-M parameter** to **display all** available **modules**.
- A detailed **description how to develop** your **own modules** can be found in the **official YARA documentation**. https://yara.readthedocs.io/en/stable/writingmodules.html#writing-modules



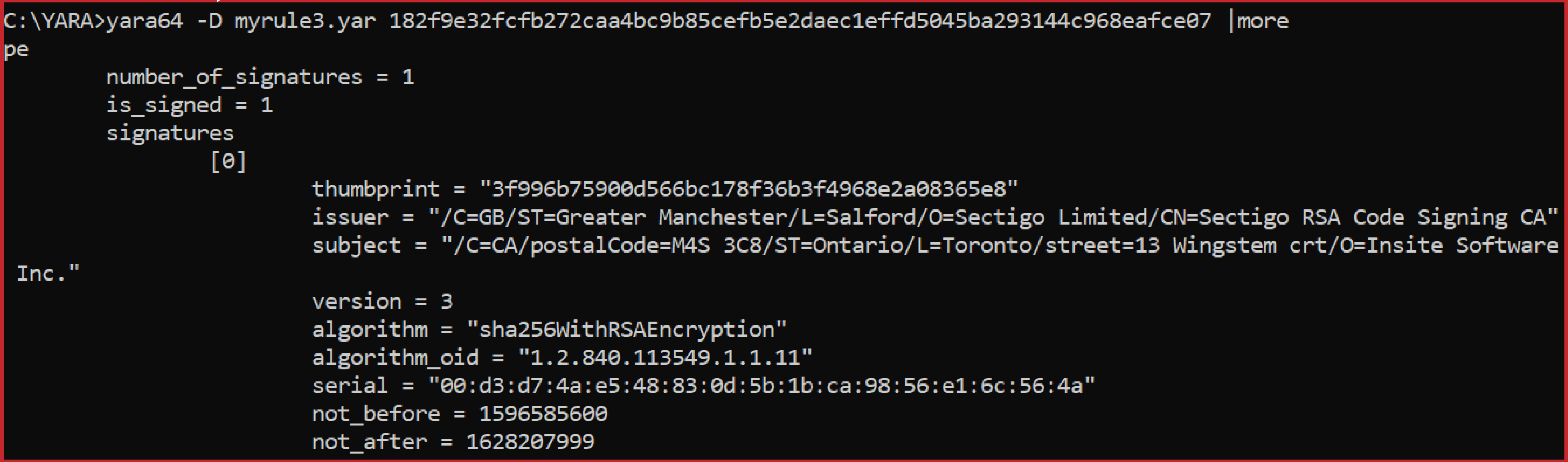
```
import "lnk"
import "console"

rule Malicious_lnk {
        meta:
                Description = "This rule shows the basic usage of the Yara lnk module"

        condition:
                lnk.is_lnk and

                // check for cmd.exe, powershell.exe and regsvr32.exe in the relative path of a lnk file
                (lnk.relative_path icontains "c\x00m\x00d\x00.\x00e\x00x\x00e" or
                lnk.relative_path icontains "p\x00o\x00w\x00e\x00r\x00s\x00h\x00e\x00l\x00l\x00.\x00e\x00x\x00e" or
                lnk.relative_path icontains "r\x00e\x00g\x00s\x00v\x00r\x003\x002\x00.\x00e\x00x\x00e") and

                // print lnk cli arguments when matching. => Usually malicious in combination with the above executables.
                console.log("\ncommand line args ==> ", lnk.command_line_arguments)
}
```

LNK- Module will be removed in the final 4.4.0 release. Check pull 1957 for more context.

- This **rule parses Microsoft Shell links** by importing the “**lnk**” **module** and **checks** for **specific fields** to **find different unicode strings** in the relative_path, by ignoring case sensitivity (**icontains**). When found it uses the “console” module to print the results.
- The following **command formats** the **results** for **better reading →** *yara64 Malicious_lnk.yar .\samples* **|** *sed "s/\\x00//g“*

YARA rules 101 - Introduction to modules 2/2

When working with modules like pe, elf, dotnet etc. assure to make use of the parameter -D. It dumps all module data of a scanned object. It‘s a useful helper while developing rules!

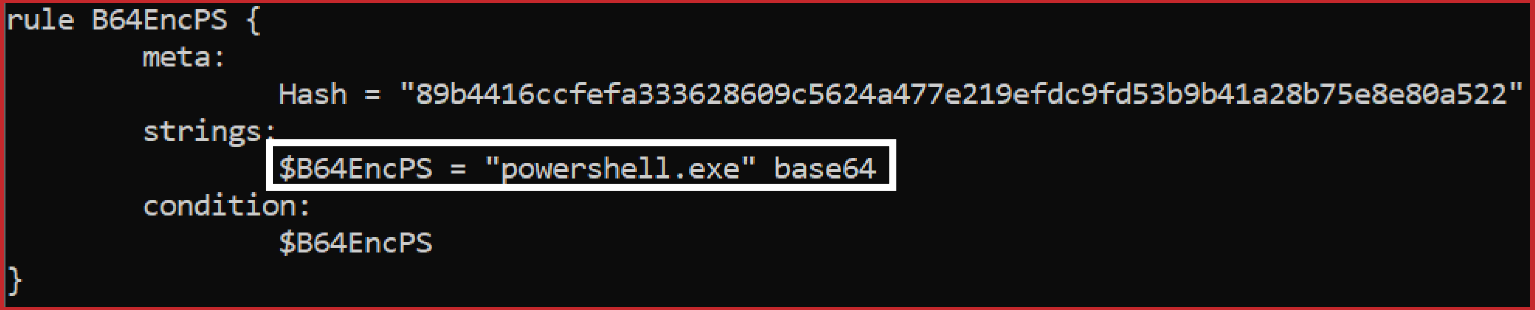


```
C:\YARA>yara64 -D myrule3.yar 182f9e32fcfb272caa4bc9b85cefb5e2daec1effd5045ba293144c968eafce07 |more
pe
        number_of_signatures = 1
        is_signed = 1
        signatures
                [0]
                        thumbprint = "3f996b75900d566bc178f36b3f4968e2a08365e8"
                        issuer = "/C=GB/ST=Greater Manchester/L=Salford/O=Sectigo Limited/CN=Sectigo RSA Code Signing CA"
                        subject = "/C=CA/postalCode=M4S 3C8/ST=Ontario/L=Toronto/street=13 Wingstem crt/O=Insite Software
 Inc."
                        version = 3
                        algorithm = "sha256WithRSAEncryption"
                        algorithm_oid = "1.2.840.113549.1.1.11"
                        serial = "00:d3:d7:4a:e5:48:83:0d:5b:1b:ca:98:56:e1:6c:56:4a"
                        not_before = 1596585600
                        not_after = 1628207999
```

For more yara cli scanner parameters check with --help

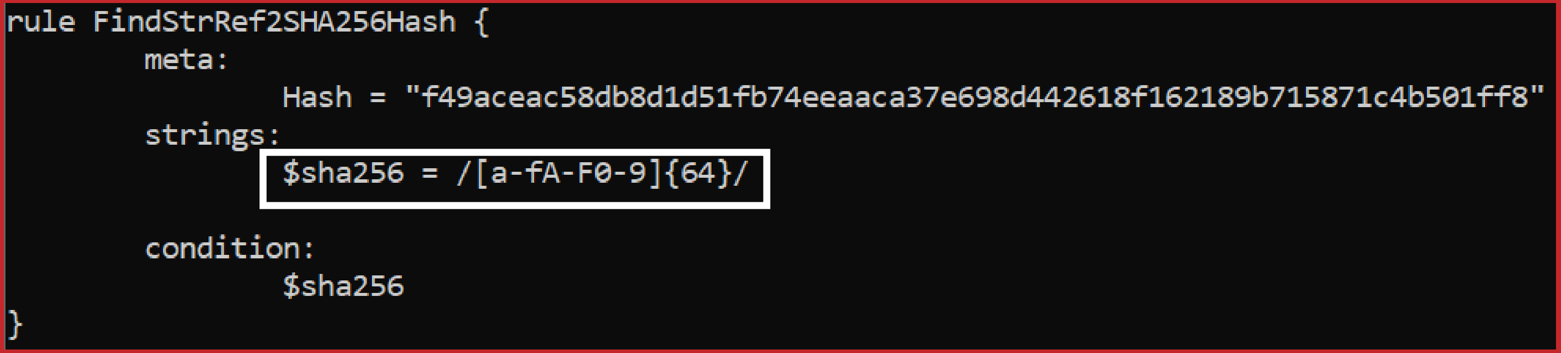
YARA rules 101 - Examples

Find base64 encoded string „powershell.exe“



```
rule B64EncPS {
        meta:
                Hash = "89b4416ccfefa333628609c5624a477e219efdc9fd53b9b41a28b75e8e80a522"
        strings:
                $B64EncPS = "powershell.exe" base64
        condition:
                $B64EncPS
}
```

Find strings matching a SHA256 hash.



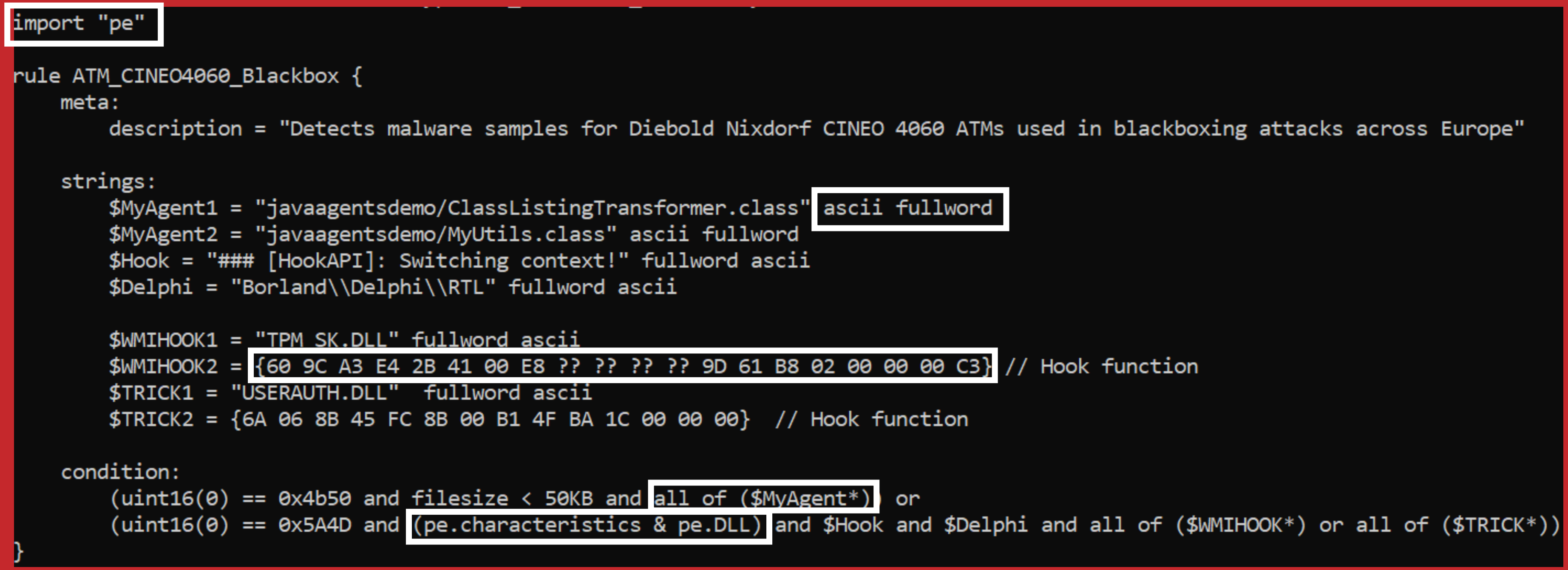
```
rule FindStrRef2SHA256Hash {
        meta:
                Hash = "f49aceac58db8d1d51fb74eeaaca37e698d442618f162189b715871c4b501ff8"
        strings:
                $sha256 = /[a-fA-F0-9]{64}/

        condition:
                $sha256
}
```

ATTENTION: Yara will show warnings when running the rule above!

Be careful when using regular expressions in YARA rules as they might have a serious impact on the performance!

The topic performance issues is discussed further on slide 26.

YARA rules 101 - Examples



```
import "pe"

rule ATM_CINEO4060_Blackbox {
    meta:
        description = "Detects malware samples for Diebold Nixdorf CINEO 4060 ATMs used in blackboxing attacks across Europe"

    strings:
        $MyAgent1 = "javaagentsdemo/ClassListingTransformer.class" ascii fullword
        $MyAgent2 = "javaagentsdemo/MyUtils.class" ascii fullword
        $Hook = "### [HookAPI]: Switching context!" fullword ascii
        $Delphi = "Borland\\Delphi\\RTL" fullword ascii

        $WMIHOOK1 = "TPM_SK.DLL" fullword ascii
        $WMIHOOK2 = {60 9C A3 E4 2B 41 00 E8 ?? ?? ?? ?? 9D 61 B8 02 00 00 00 C3} // Hook function
        $TRICK1 = "USERAUTH.DLL"  fullword ascii
        $TRICK2 = {6A 06 8B 45 FC 8B 00 B1 4F BA 1C 00 00 00}  // Hook function

    condition:
        (uint16(0) == 0x4b50 and filesize < 50KB and all of ($MyAgent*)) or
        (uint16(0) == 0x5A4D and (pe.characteristics & pe.DLL) and $Hook and $Delphi and all of ($WMIHOOK*) or all of ($TRICK*))
}
```

ascii → string must consist of ascii characters (use “wide” for wide chars, “nocase” to ignore case sensitivity)

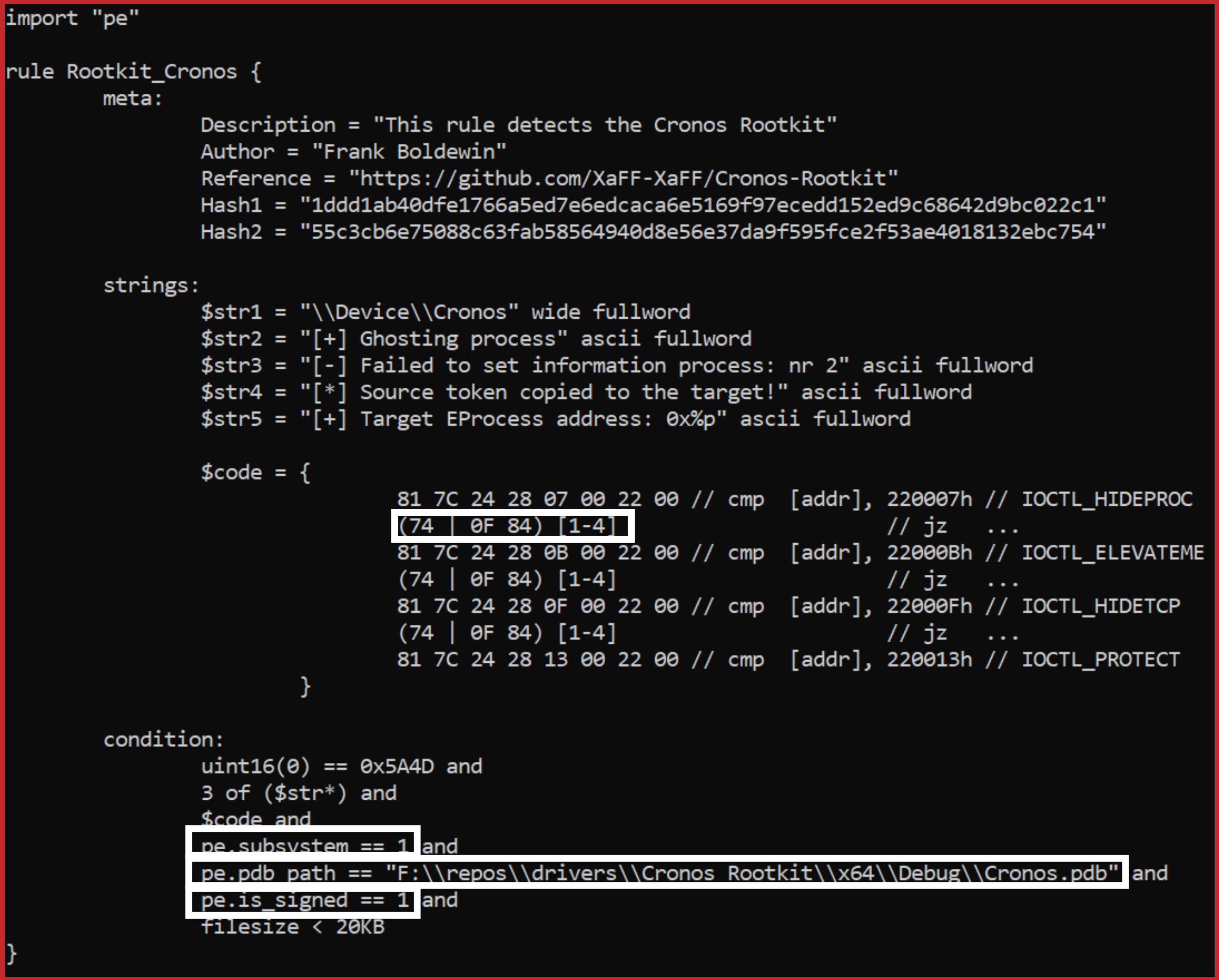
fullword → assure the string only matches when delimited by non-alphanumeric characters

{60 9c A3 2B 41 00 ... } → check for a sequence of hex bytes (wildcards are nibble-wise e.g. ?? or 8? or ?7)

all of ($MyAgent*) → all $MyAgent strings have to match, (1 of ($MyAgent*) means only one of both $MyAgent strings has to match

pe.characteristics & pe.DLL → file must be a dynamic link library (pe.<…> conditions require importing the “pe” module)

YARA rules 101 - Examples



```
import "pe"

rule Rootkit_Cronos {
        meta:
                Description = "This rule detects the Cronos Rootkit"
                Author = "Frank Boldewin"
                Reference = "https://github.com/XaFF-XaFF/Cronos-Rootkit"
                Hash1 = "1ddd1ab40dfe1766a5ed7e6edcaca6e5169f97ecedd152ed9c68642d9bc022c1"
                Hash2 = "55c3cb6e75088c63fab58564940d8e56e37da9f595fce2f53ae4018132ebc754"

        strings:
                $str1 = "\\Device\\Cronos" wide fullword
                $str2 = "[+] Ghosting process" ascii fullword
                $str3 = "[-] Failed to set information process: nr 2" ascii fullword
                $str4 = "[*] Source token copied to the target!" ascii fullword
                $str5 = "[+] Target EProcess address: 0x%p" ascii fullword

                $code = {
                                81 7C 24 28 07 00 22 00 // cmp  [addr], 220007h // IOCTL_HIDEPROC
                                (74 | 0F 84) [1-4]                             // jz  ...
                                81 7C 24 28 0B 00 22 00 // cmp  [addr], 22000Bh // IOCTL_ELEVATEME
                                (74 | 0F 84) [1-4]                             // jz  ...
                                81 7C 24 28 0F 00 22 00 // cmp  [addr], 22000Fh // IOCTL_HIDETCP
                                (74 | 0F 84) [1-4]                             // jz  ...
                                81 7C 24 28 13 00 22 00 // cmp  [addr], 220013h // IOCTL_PROTECT
                        }

        condition:
                uint16(0) == 0x5A4D and
                3 of ($str*) and
                $code and
                pe.subsystem == 1 and
                pe.pdb_path == "F:\\repos\\drivers\\Cronos_Rootkit\\x64\\Debug\\Cronos.pdb" and
                pe.is_signed == 1 and
                filesize < 20KB
}
```

(74 | 0F 84) [1-4] → jz instruction opcode can be short '74' or near '0F 84' in samples. Depending on the instruction the operands length can be between [1-4]

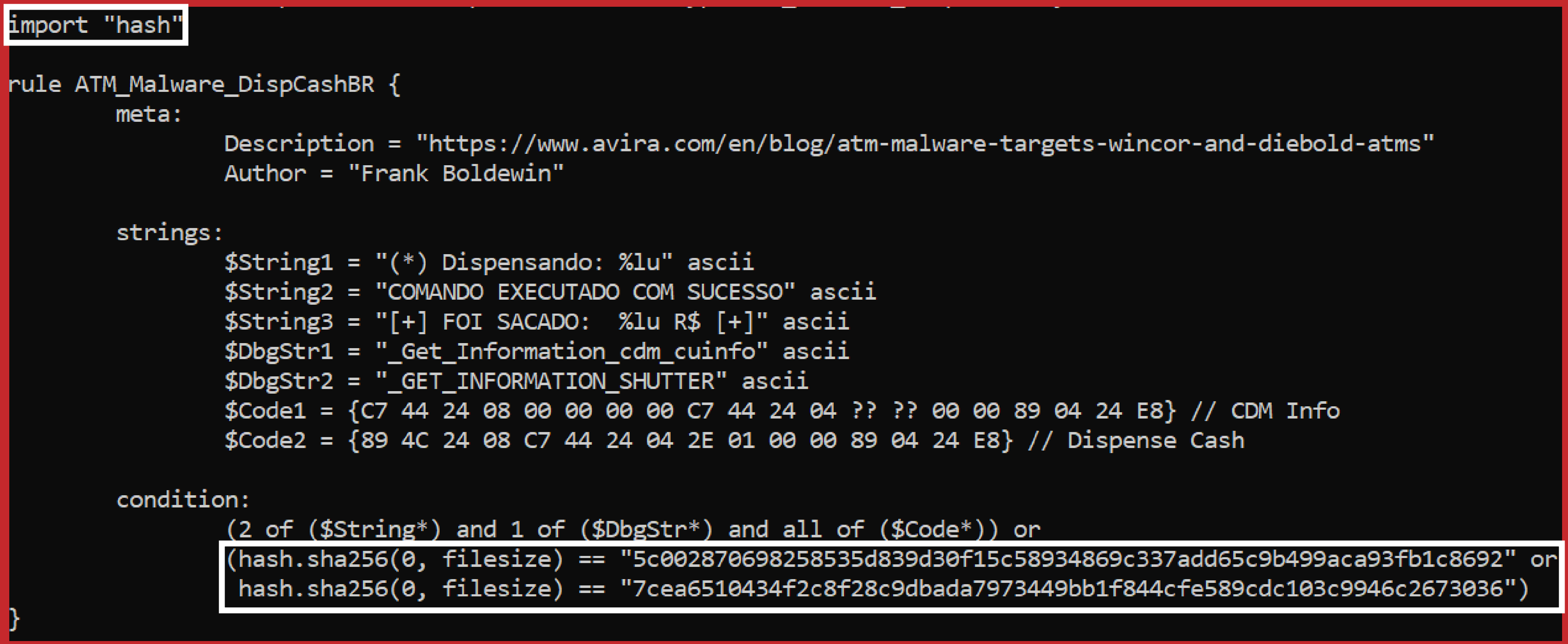
pe.subsystem == 1 → file must be of type driver, thus the subsystem is 1 (e.g. 2 is a Windows GUI app, 3 is Windows console app)

pe.pdb_path == "F:\\repos…" → checks for a specific pdb debug path artifact

pe.is_signed == 1 → checks if any of the included authenticode signatures are formally correct

YARA rules 101 - Examples

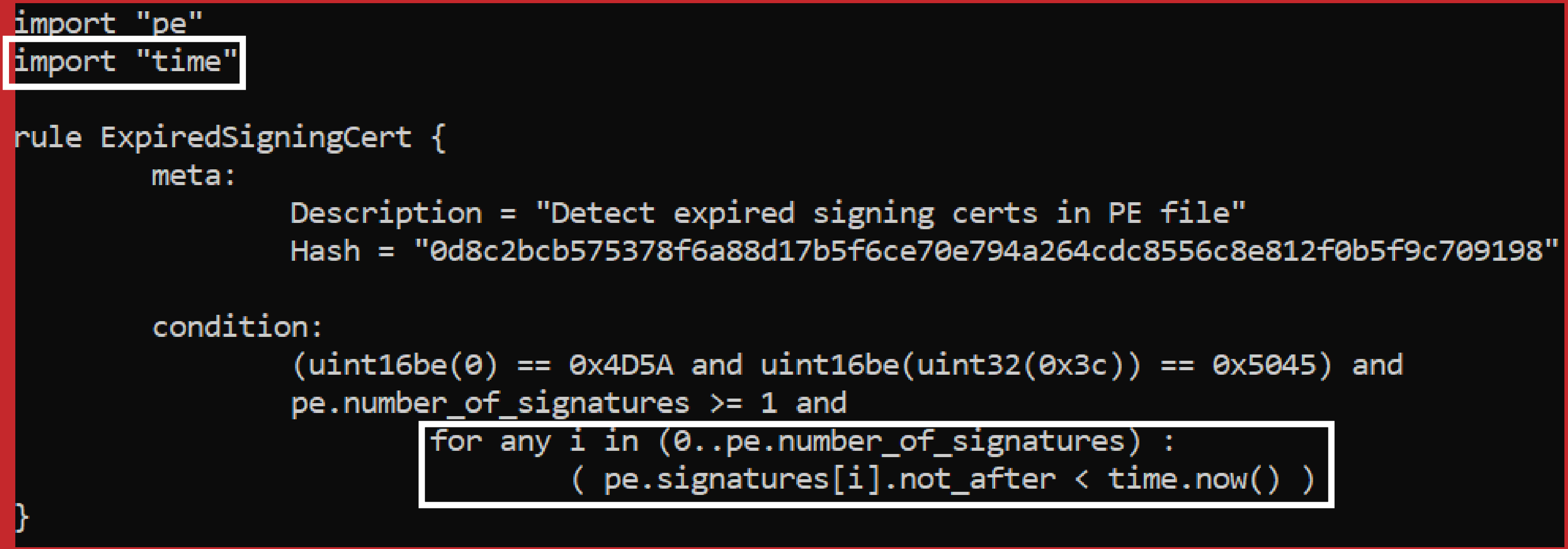
- The “**hash” module supports** the calculation of **MD5**, **SHA1**, **SHA256**, **CHECKSUM32** and **CRC32 values**, in the combination **offset + size or** by specifying a **string**.
  Examples:
  hash.sha1(0, filesize) == "36ea4b0be901ea9deddb8e862c29ef7f50b6ceae“
  hash.sha1(pe.rich_signature.clear_data) == "2f1a7d8b4351b2773c1b7befaa5e64443e6045fe"



```
import "hash"

rule ATM_Malware_DispCashBR {
        meta:
                Description = "https://www.avira.com/en/blog/atm-malware-targets-wincor-and-diebold-atms"
                Author = "Frank Boldewin"

        strings:
                $String1 = "(*) Dispensando: %lu" ascii
                $String2 = "COMANDO EXECUTADO COM SUCESSO" ascii
                $String3 = "[+] FOI SACADO:  %lu R$ [+]" ascii
                $DbgStr1 = "_Get_Information_cdm_cuinfo" ascii
                $DbgStr2 = "_GET_INFORMATION_SHUTTER" ascii
                $Code1 = {C7 44 24 08 00 00 00 00 C7 44 24 04 ?? ?? 00 00 89 04 24 E8} // CDM Info
                $Code2 = {89 4C 24 08 C7 44 24 04 2E 01 00 00 89 04 24 E8} // Dispense Cash

        condition:
                (2 of ($String*) and 1 of ($DbgStr*) and all of ($Code*)) or
                (hash.sha256(0, filesize) == "5c002870698258535d839d30f15c58934869c337add65c9b499aca93fb1c8692" or
                 hash.sha256(0, filesize) == "7cea6510434f2c8f28c9dbada7973449bb1f844cfe589cdc103c9946c2673036")
}
```

YARA rules 101 - Examples

- The „**time**“ **module** comes in handy when **measuring temporal conditions**. The **returned** int **value** is the **number of seconds since 1st January 1970**.

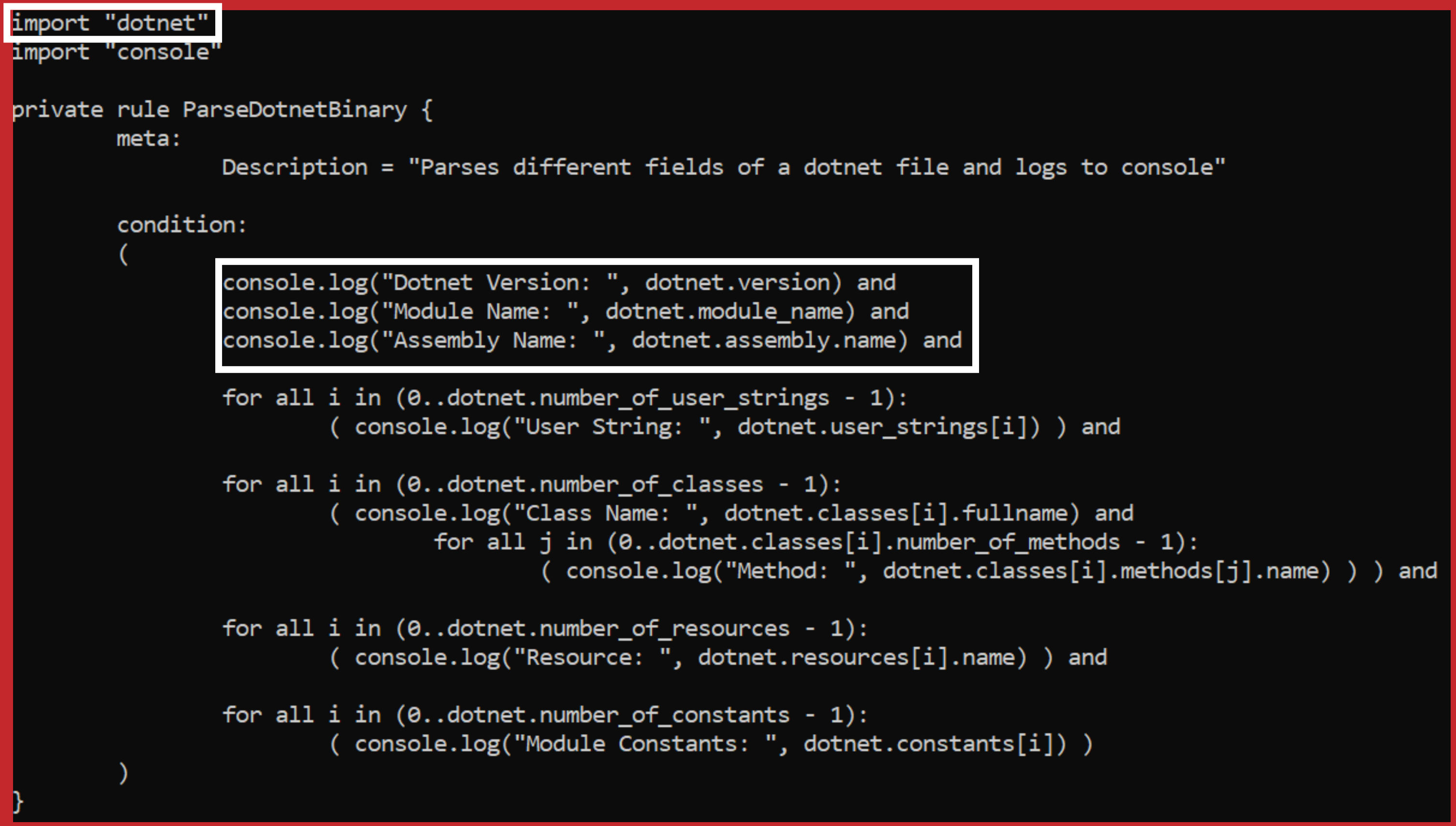


```
import "pe"
import "time"

rule ExpiredSigningCert {
        meta:
                Description = "Detect expired signing certs in PE file"
                Hash = "0d8c2bcb575378f6a88d17b5f6ce70e794a264cdc8556c8e812f0b5f9c709198"

        condition:
                (uint16be(0) == 0x4D5A and uint16be(uint32(0x3c)) == 0x5045) and
                pe.number_of_signatures >= 1 and
                        for any i in (0..pe.number_of_signatures) :
                                ( pe.signatures[i].not_after < time.now() )
}
```

- **Iterate through** all authenticode **signatures** and **check** if they are **expired** by **comparing** the **not_after** field with the **current time**.

YARA rules 101 - Examples

- **Import** the **“dotnet” module**, **parse** different **fields** and **print** its **results** to **console**.
- The following **command formats** the **results** for **better reading**:
  yara64 ParseDotnetBinary.yar 9b7182ceada457d4d204a277e86e2b6a7b5aa6c892033fe7c1b750a17d023955 | sed "s/\\x00//g" |more

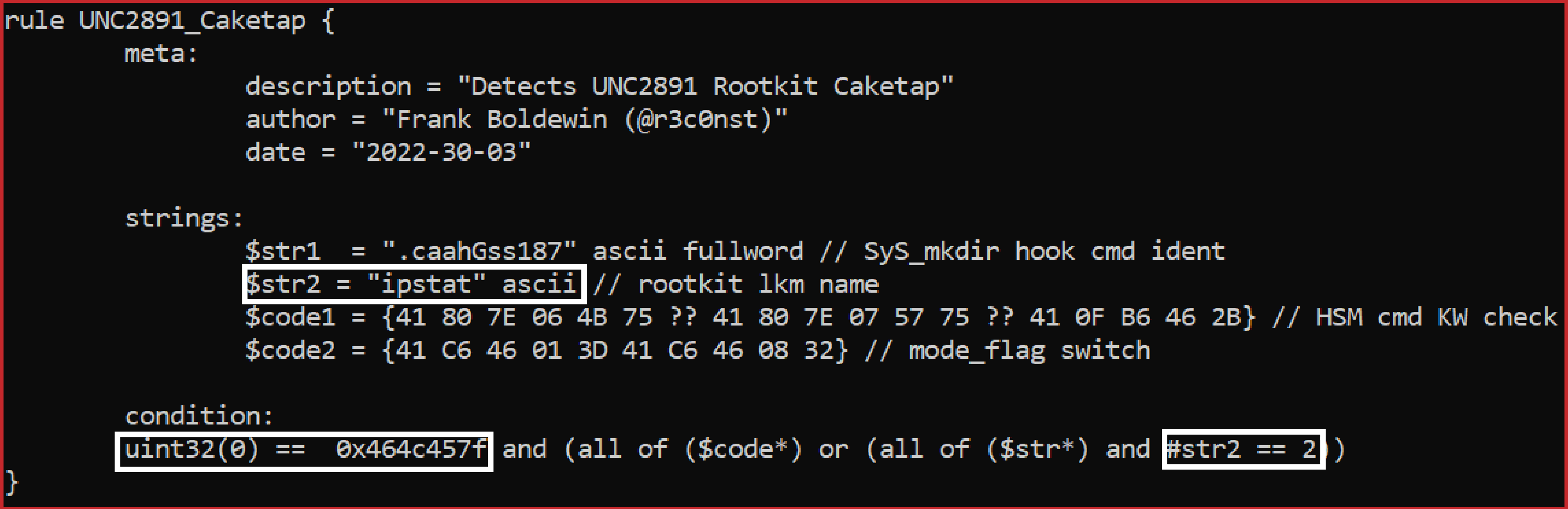


```
import "dotnet"
import "console"

private rule ParseDotnetBinary {
        meta:
                Description = "Parses different fields of a dotnet file and logs to console"

        condition:
        (
                console.log("Dotnet Version: ", dotnet.version) and
                console.log("Module Name: ", dotnet.module_name) and
                console.log("Assembly Name: ", dotnet.assembly.name) and

                for all i in (0..dotnet.number_of_user_strings - 1):
                        ( console.log("User String: ", dotnet.user_strings[i]) ) and

                for all i in (0..dotnet.number_of_classes - 1):
                        ( console.log("Class Name: ", dotnet.classes[i].fullname) and
                                for all j in (0..dotnet.classes[i].number_of_methods - 1):
                                        ( console.log("Method: ", dotnet.classes[i].methods[j].name) ) ) and

                for all i in (0..dotnet.number_of_resources - 1):
                        ( console.log("Resource: ", dotnet.resources[i].name) ) and

                for all i in (0..dotnet.number_of_constants - 1):
                        ( console.log("Module Constants: ", dotnet.constants[i]) )
        )
}
```

YARA rules 101 - Examples

#str2 == 2 → ascii string „ipstat“ must occur 2 times

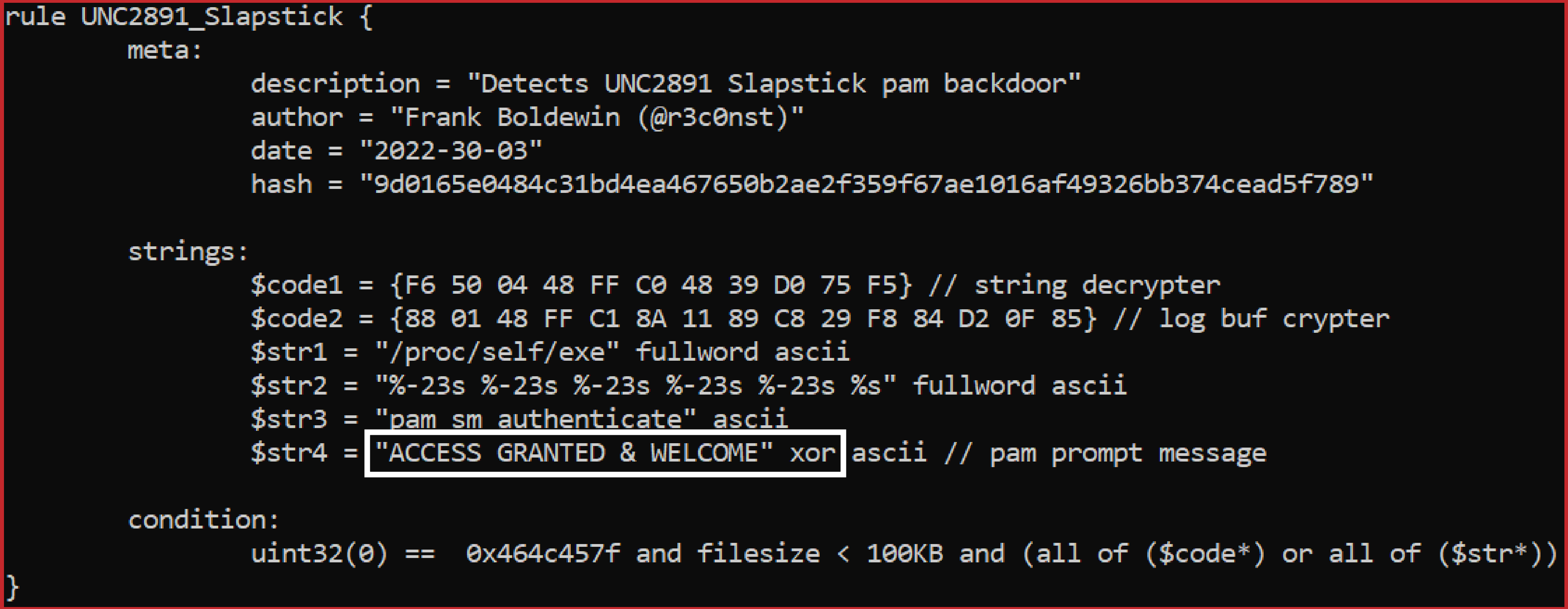
uint32 (0) == 0x464c457f → check for ELF header in little endian format at offset 0



```
rule UNC2891_Caketap {
        meta:
                description = "Detects UNC2891 Rootkit Caketap"
                author = "Frank Boldewin (@r3c0nst)"
                date = "2022-30-03"

        strings:
                $str1  = ".caahGss187" ascii fullword // SyS_mkdir hook cmd ident
                $str2 = "ipstat" ascii // rootkit lkm name
                $code1 = {41 80 7E 06 4B 75 ?? 41 80 7E 07 57 75 ?? 41 0F B6 46 2B} // HSM cmd KW check
                $code2 = {41 C6 46 01 3D 41 C6 46 08 32} // mode_flag switch

        condition:
        uint32(0) ==  0x464c457f and (all of ($code*) or (all of ($str*) and #str2 == 2))
}
```

YARA rules 101 - Examples

$str4 = „ACCESS GRANTED & WELCOME“ xor → find encoded ascii string by using a 1-byte XOR

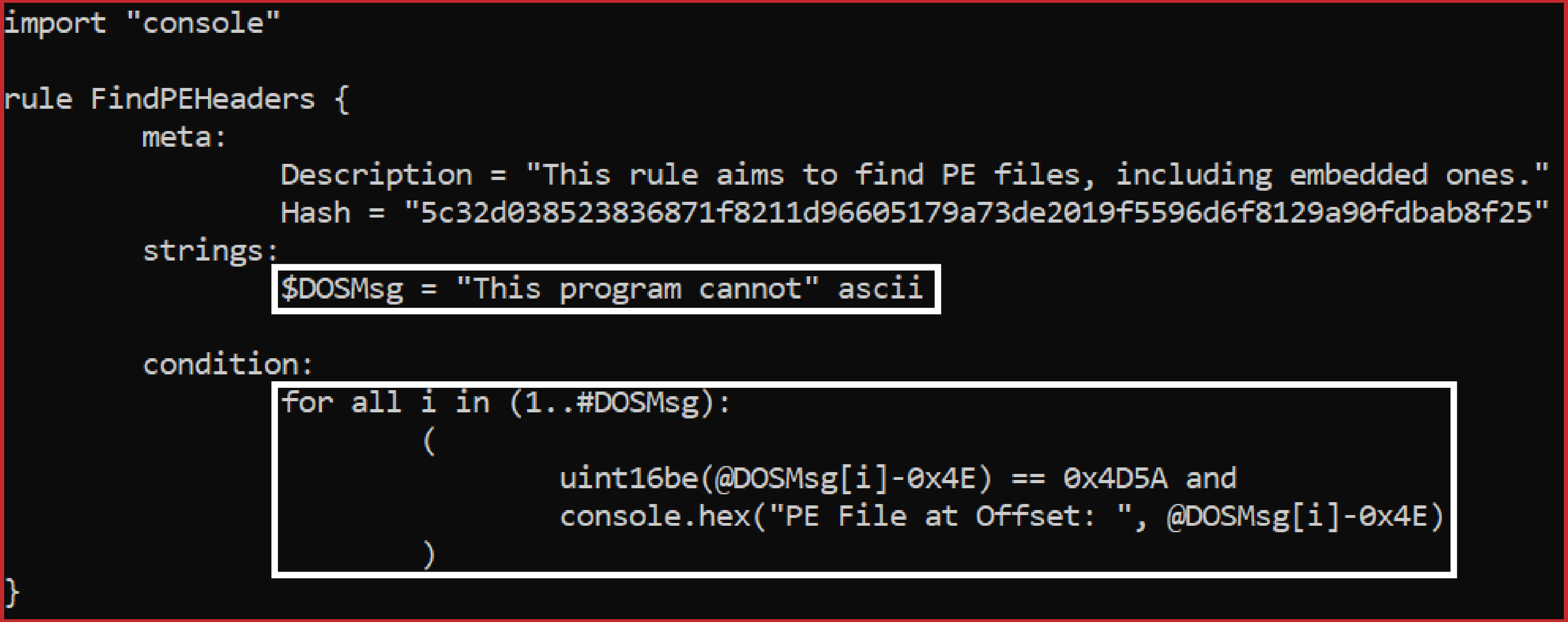


```
rule UNC2891_Slapstick {
        meta:
                description = "Detects UNC2891 Slapstick pam backdoor"
                author = "Frank Boldewin (@r3c0nst)"
                date = "2022-30-03"
                hash = "9d0165e0484c31bd4ea467650b2ae2f359f67ae1016af49326bb374cead5f789"

        strings:
                $code1 = {F6 50 04 48 FF C0 48 39 D0 75 F5} // string decrypter
                $code2 = {88 01 48 FF C1 8A 11 89 C8 29 F8 84 D2 0F 85} // log buf crypter
                $str1 = "/proc/self/exe" fullword ascii
                $str2 = "%-23s %-23s %-23s %-23s %-23s %s" fullword ascii
                $str3 = "pam_sm_authenticate" ascii
                $str4 = "ACCESS GRANTED & WELCOME" xor ascii // pam prompt message

        condition:
                uint32(0) ==  0x464c457f and filesize < 100KB and (all of ($code*) or all of ($str*))
}
```

YARA rules 101 - Examples

for all i in (1..#DOSMsg) → Iterate trough all matches of $DOSMsg (These indexes are 1-based, meaning the first match is @DOSMsg[1], the next @DOSMsg[2] etc.

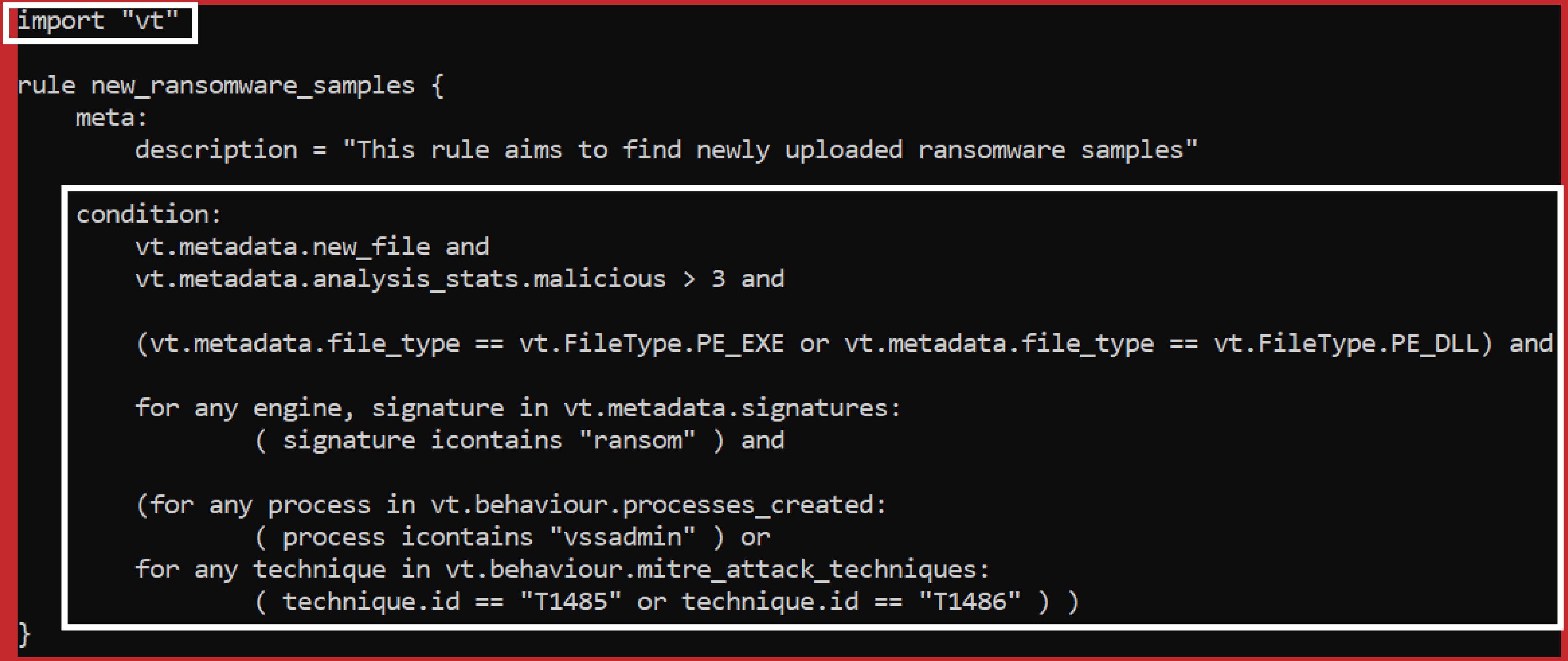
uint16be(@DOSMsg[i]-0x4E) == 0x4D5A → @DOSMsg[i] is the offset to the current match. If the unsigned 16-bit big endian value -0x4E is equal 0x4D5A (MZ-Header) it's assumed, we found a PE-File.



```
import "console"

rule FindPEHeaders {
        meta:
                Description = "This rule aims to find PE files, including embedded ones."
                Hash = "5c32d038523836871f8211d96605179a73de2019f5596d6f8129a90fdbab8f25"
        strings:
                $DOSMsg = "This program cannot" ascii

        condition:
                for all i in (1..#DOSMsg):
                        (
                                uint16be(@DOSMsg[i]-0x4E) == 0x4D5A and
                                console.hex("PE File at Offset: ", @DOSMsg[i]-0x4E)
                        )
}
```

Virustotal Hunting with the YARA “vt” module 1/2

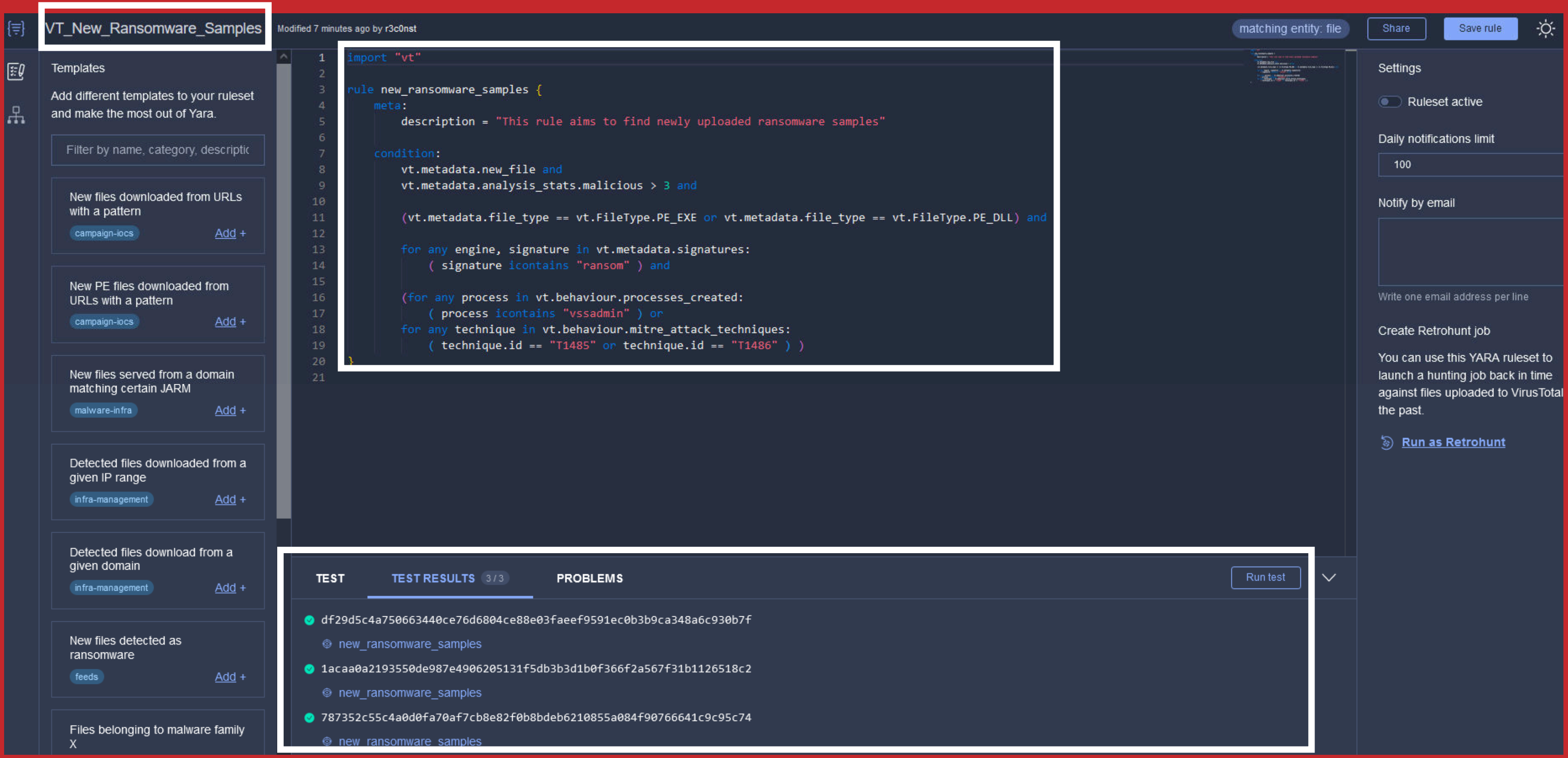
- For those with Virustotal Enterprise access, a **special YARA module** is available via Retro+Livehunt to **search** the **VT database** for file metadata and behavior, as well as URLs, domains and IP addresses.
- The example **rule** below **matches** for **newly uploaded EXE + DLL files**, when **at least 3 AV engines classified** them as **malicious**, **contain** the **engine signature** “**ransom**” and a **process** named “**vssadmin**” has been **created at runtime or MITRE ATT&CK techniques T1485** (Data Destruction) **or T1486** (Data Encrypted for Impact) has **occured**.



```
import "vt"

rule new_ransomware_samples {
    meta:
        description = "This rule aims to find newly uploaded ransomware samples"

    condition:
        vt.metadata.new_file and
        vt.metadata.analysis_stats.malicious > 3 and

        (vt.metadata.file_type == vt.FileType.PE_EXE or vt.metadata.file_type == vt.FileType.PE_DLL) and

        for any engine, signature in vt.metadata.signatures:
                ( signature icontains "ransom" ) and

        (for any process in vt.behaviour.processes_created:
                ( process icontains "vssadmin" ) or
        for any technique in vt.behaviour.mitre_attack_techniques:
                ( technique.id == "T1485" or technique.id == "T1486" ) )
}
```

Virustotal Hunting with the YARA “vt” module 2/2

VT YARA ruleset editor for writing, testing and hunting these kind of rules



VT_New_Ransomware_Samples

Modified 7 minutes ago by r3c0nst

Templates

Add different templates to your ruleset and make the most out of Yara.

New files downloaded from URLs with a pattern

New PE files downloaded from URLs with a pattern

New files served from a domain matching certain JARM

Detected files downloaded from a given IP range

Detected files download from a given domain

New files detected as ransomware

Files belonging to malware family X

```
import "vt"

rule new_ransomware_samples {
    meta:
        description = "This rule aims to find newly uploaded ransomware samples"

    condition:
        vt.metadata.new_file and
        vt.metadata.analysis_stats.malicious > 3 and

        (vt.metadata.file_type == vt.FileType.PE_EXE or vt.metadata.file_type == vt.FileType.PE_DLL) and

        for any engine, signature in vt.metadata.signatures:
            ( signature icontains "ransom" ) and

        (for any process in vt.behaviour.processes_created:
            ( process icontains "vssadmin" ) or
        for any technique in vt.behaviour.mitre_attack_techniques:
            ( technique.id == "T1485" or technique.id == "T1486" ) )
}
```

```
● df29d5c4a750663440ce76d6804ce88e03faeef9591ec0b3b9ca348a6c930b7f
    new_ransomware_samples
● 1acaa0a2193550de987e4906205131f5db3b3d1b0f366f2a567f31b1126518c2
    new_ransomware_samples
● 787352c55c4a0d0fa70af7cb8e82f0b8bdeb6210855a084f90766641c9c95c74
    new_ransomware_samples
```

Settings

Ruleset active

Daily notifications limit: 100

Notify by email

Write one email address per line

Create Retrohunt job

You can use this YARA ruleset to launch a hunting job back in time against files uploaded to VirusTotal the past.

https://support.virustotal.com/hc/en-us/articles/360007088057-Writing-YARA-rules-for-Livehunt
https://blog.virustotal.com/2023/07/actionable-threat-intel-iii-introducing.html
https://developers.virustotal.com/docs/nethunt

YARA - Exercise

→ Now **practice** the **materials taught**. The presented **rules and** corresponding **samples** can be **found** in the **folder C:\YARA-Workshop\Examples**

→ **Play with** the different Yara **rules** from the examples 1-10 and **gain experience** by choosing different parameters of the Yara cli scanner and check the results. Also customize the rules to your needs if you like.

→ **30 minutes for** this **exercise**

YARA rules 101 – Things to look for when developing rules 1/2

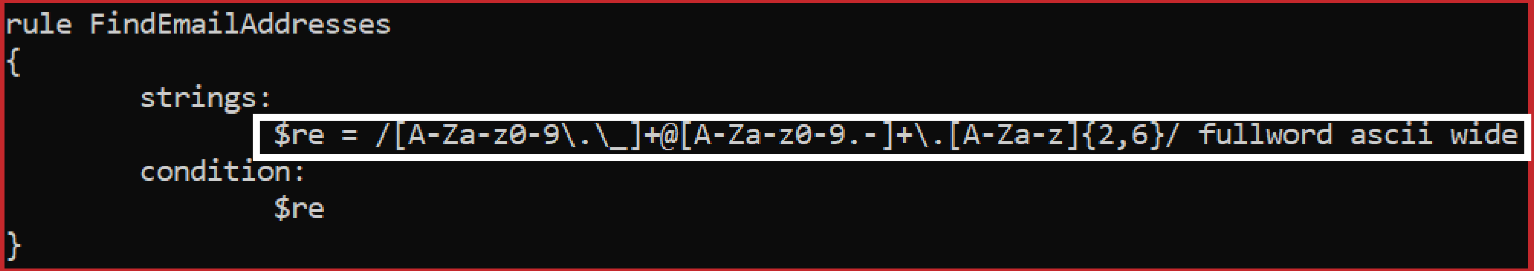
- The **simplest type** of detection engineering (**next to detection** by **hashes**) is the **identification** of **unique strings** within a **specific malware**.
- The **more samples** you have of a **particular variant resp. family**, the **more precisely** such **unique strings** can be **determined**.
- The **combination** in which the **strings occur with other characteristics** in samples can **also** be **helpful when developing** detection **rules**.
- You should **pay attention** if they are available as **Unicode or Ascii** strings, if they are **misspelled** or if **other** special **characteristics** can be identified.
- When analyzing **document format samples** such as PDF, DOC, XLS, ONE, PPT etc., it should be **determined** whether **patterns** of an **exploit** or **shellcode** are hidden in the file, as well as other **embedded objects** which are intended for **malicious purposes**.
- In addition, **document meta data** (title, author, creation and modification date etc.) **or signing certificates** in a sample **can be** good detection **indicators**.

YARA rules 101 – Things to look for when developing rules 2/2

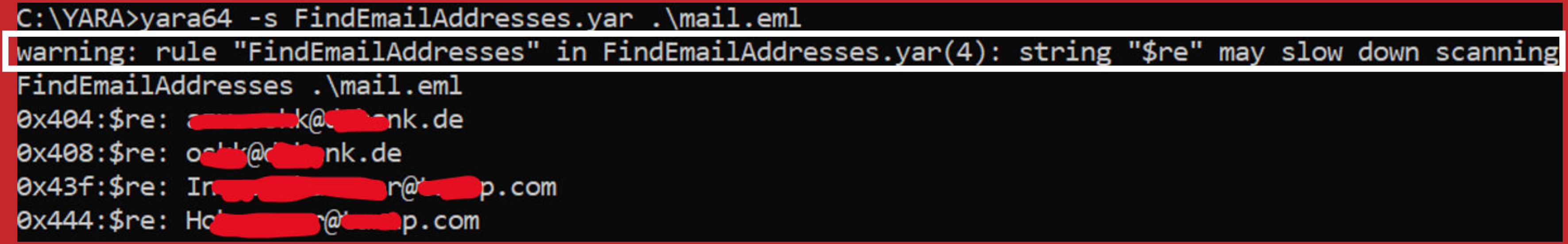
- **Executable file formats** such as PE, ELF, MACH-O or XCOFF almost always have a **number** of **unique indicators** (**especially in combination with others**), including **import hashes**, **resources**, **header properties**, **section names**, **compiler artifacts**, **high file entropies** in specific sections, **overlays** and so forth.
- One **key factor** when **developing rules** is a **decent understanding** of the **structure** of **file formats**, whether executable files or document formats.
- **Tools** such **as** the **010 editor**, **Cerbero Suite** and other file format editors (**PE-Bear**, **XELFViewer** etc.) are extremely **helpful** utilities **when developing rules**.
- When searching for suitable **code patterns as indicators**, **routines** which are usually **not constantly changed** by the malware authors are the ones to look for, e.g. routines for **string deobfuscation**, **API hashing**, **proprietary crypto** for C2 communication or config en-/decryption, **anti-VM**, **anti-emulation** or **anti-debugging** tricks.
- The same applies to **polymorphic layers**. **Looking** kinda **random** on a first view, they are also **based on certain patterns** and can therefore be **identified for detection**.

Some YARA rules performance issues + optimization tips

There are **several recommendations** regarding **scanning performance**. **Some** of them **have** rather **minor effects**, **such as** the **use** of **xor** or the **calculation** of **hashes**. However, the **messages from** the **YARA compiler** are the **most relevant**. **If** there are **warnings** stating, for **example**, that a **regular expression** is **slowing down** the **scanning performance**, **fixed it!** Such a regular expression **may not have** a **major impact at first glance**, **but** the **larger** the **files** are, **the slower** the **scans become**.



```
rule FindEmailAddresses
{
        strings:
                $re = /[A-Za-z0-9\.\_]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,6}/ fullword ascii wide
        condition:
                $re
}
```



```
C:\YARA>yara64 -s FindEmailAddresses.yar .\mail.eml
warning: rule "FindEmailAddresses" in FindEmailAddresses.yar(4): string "$re" may slow down scanning
FindEmailAddresses .\mail.eml
0x404:$re: [redacted]k@[redacted]nk.de
0x408:$re: o[redacted]@[redacted]nk.de
0x43f:$re: I[redacted]r@[redacted]p.com
0x444:$re: H[redacted]@[redacted]p.com
```

Next to this, **beware** of **unbounded loops**, which **can become** a **problem** if, for example, **#a** has **thousands of instances** in case of: for any i in (0..#a).
In such cases, the **math module should be used** to **set** an **upper bound value**. Example: for any i in (0..math.min(200..#a)…

Further recommendations on **performance optimization** can be found in these two blog posts:
https://engineering.avast.io/know-your-yara-rules-series-1-know-your-yara/
https://engineering.avast.io/know-your-yara-rules-series-2-rewrite-your-rules/

YARA - Python API (basic usage)

Import the YARA Python module

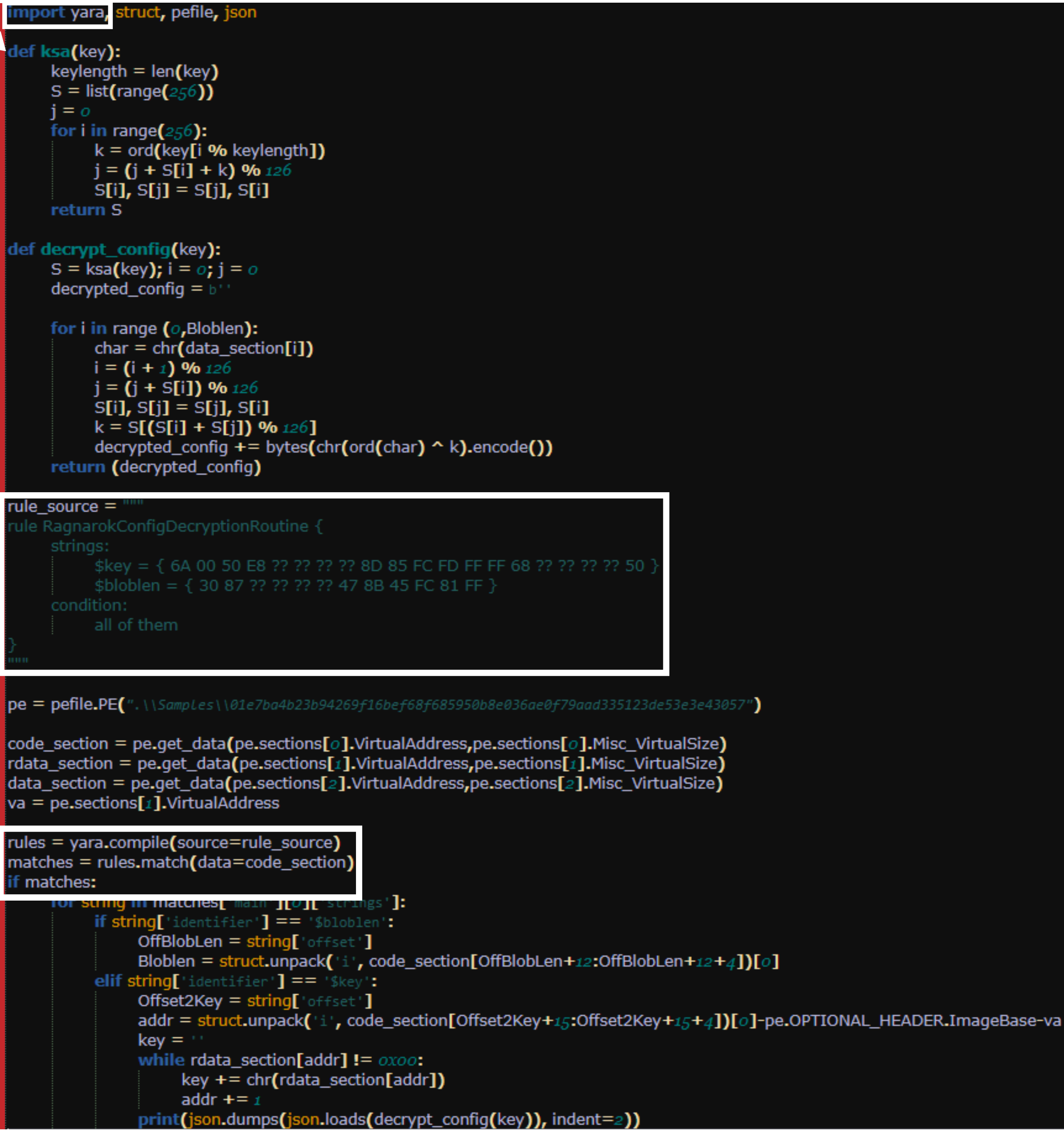
Get it here → pip install yara-python

One way to work with YARA rules is to compile them directly from the Python code

Other methods are described here: https://yara.readthedocs.io/en/stable/yarapython.html

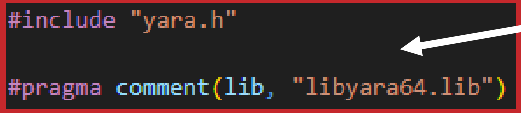
Once compiled matching results are accessible in a dictionary

Also check out Florian Roth‘s Yara Python scanner Loki to get more practical insights how to use of Yara Python https://github.com/Neo23x0/Loki



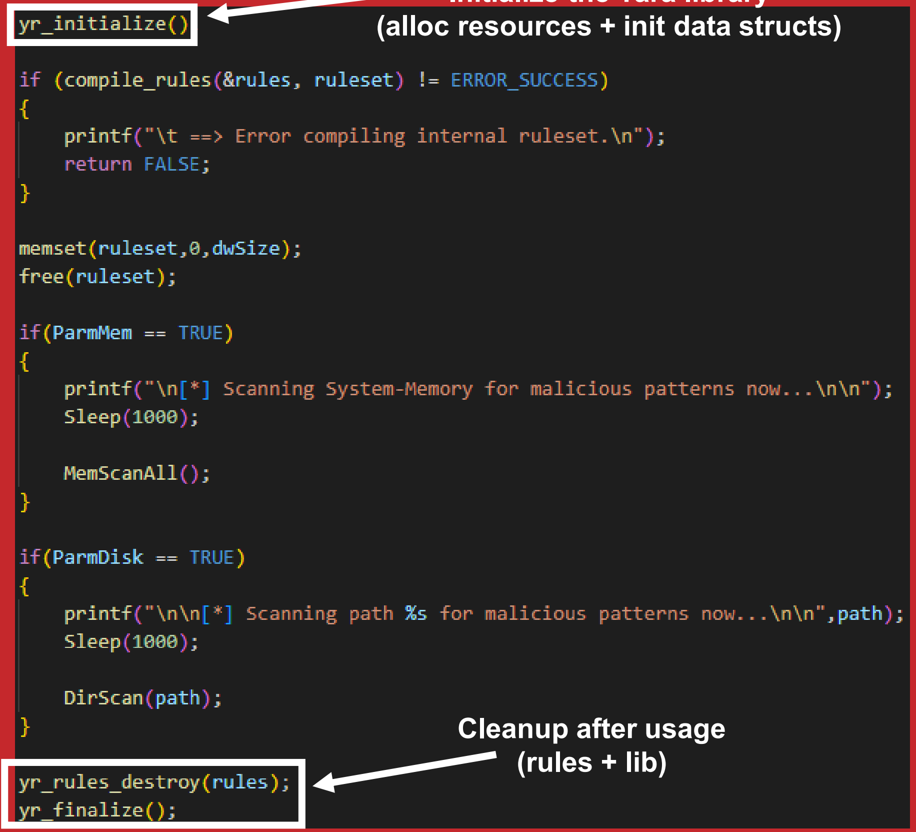
```
import yara, struct, pefile, json

def ksa(key):
	keylength = len(key)
	S = list(range(256))
	j = 0
	for i in range(256):
		k = ord(key[i % keylength])
		j = (j + S[i] + k) % 126
		S[i], S[j] = S[j], S[i]
	return S

def decrypt_config(key):
	S = ksa(key); i = 0; j = 0
	decrypted_config = b''

	for i in range (0,Bloblen):
		char = chr(data_section[i])
		i = (i + 1) % 126
		j = (j + S[i]) % 126
		S[i], S[j] = S[j], S[i]
		k = S[(S[i] + S[j]) % 126]
		decrypted_config += bytes(chr(ord(char) ^ k).encode())
	return (decrypted_config)

rule_source = """
rule RagnarokConfigDecryptionRoutine {
	strings:
		$key = { 6A 00 50 E8 ?? ?? ?? ?? 8D 85 FC FD FF FF 68 ?? ?? ?? ?? 50 }
		$bloblen = { 30 87 ?? ?? ?? ?? 47 8B 45 FC 81 FF }
	condition:
		all of them
}
"""

pe = pefile.PE(".\\Samples\\01e7ba4b23b94269f16bef68f685950b8e036ae0f79aad335123de53e3e43057")

code_section = pe.get_data(pe.sections[0].VirtualAddress,pe.sections[0].Misc_VirtualSize)
rdata_section = pe.get_data(pe.sections[1].VirtualAddress,pe.sections[1].Misc_VirtualSize)
data_section = pe.get_data(pe.sections[2].VirtualAddress,pe.sections[2].Misc_VirtualSize)
va = pe.sections[1].VirtualAddress

rules = yara.compile(source=rule_source)
matches = rules.match(data=code_section)
if matches:
	for string in matches['main'][0]['strings']:
		if string['identifier'] == '$bloblen':
			OffBlobLen = string['offset']
			Bloblen = struct.unpack('i', code_section[OffBlobLen+12:OffBlobLen+12+4])[0]
		elif string['identifier'] == '$key':
			Offset2Key = string['offset']
			addr = struct.unpack('i', code_section[Offset2Key+15:Offset2Key+15+4])[0]-pe.OPTIONAL_HEADER.ImageBase-va
			key = ''
			while rdata_section[addr] != 0x00:
				key += chr(rdata_section[addr])
				addr += 1
			print(json.dumps(json.loads(decrypt_config(key)), indent=2))
```

YARA - C API usage



```
#include "yara.h"

#pragma comment(lib, "libyara64.lib")
```

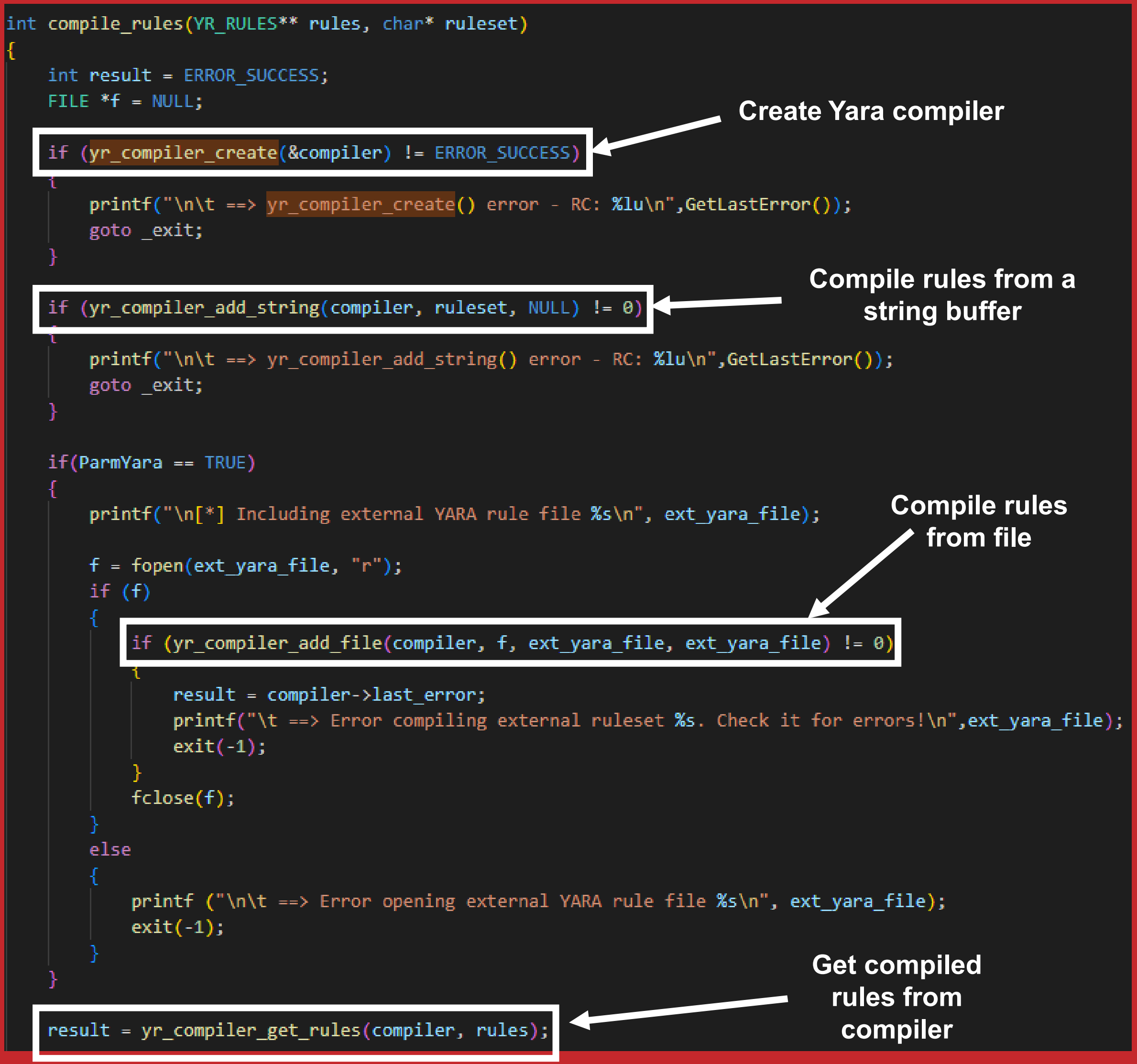
Required header/library



```
yr_initialize()

if (compile_rules(&rules, ruleset) != ERROR_SUCCESS)
{
    printf("\t ==> Error compiling internal ruleset.\n");
    return FALSE;
}

memset(ruleset,0,dwSize);
free(ruleset);

if(ParmMem == TRUE)
{
    printf("\n[*] Scanning System-Memory for malicious patterns now...\n\n");
    Sleep(1000);

    MemScanAll();
}

if(ParmDisk == TRUE)
{
    printf("\n\n[*] Scanning path %s for malicious patterns now...\n\n",path);
    Sleep(1000);

    DirScan(path);
}

yr_rules_destroy(rules);
yr_finalize();
```

Initialize the Yara library (alloc resources + init data structs)

Cleanup after usage (rules + lib)



```
int compile_rules(YR_RULES** rules, char* ruleset)
{
    int result = ERROR_SUCCESS;
    FILE *f = NULL;

    if (yr_compiler_create(&compiler) != ERROR_SUCCESS)
    {
        printf("\n\t ==> yr_compiler_create() error - RC: %lu\n",GetLastError());
        goto _exit;
    }

    if (yr_compiler_add_string(compiler, ruleset, NULL) != 0)
    {
        printf("\n\t ==> yr_compiler_add_string() error - RC: %lu\n",GetLastError());
        goto _exit;
    }

    if(ParmYara == TRUE)
    {
        printf("\n[*] Including external YARA rule file %s\n", ext_yara_file);

        f = fopen(ext_yara_file, "r");
        if (f)
        {
            if (yr_compiler_add_file(compiler, f, ext_yara_file, ext_yara_file) != 0)
            {
                result = compiler->last_error;
                printf("\t ==> Error compiling external ruleset %s. Check it for errors!\n",ext_yara_file);
                exit(-1);
            }
            fclose(f);
        }
        else
        {
            printf ("\n\t ==> Error opening external YARA rule file %s\n", ext_yara_file);
            exit(-1);
        }
    }

    result = yr_compiler_get_rules(compiler, rules);
```

Create Yara compiler

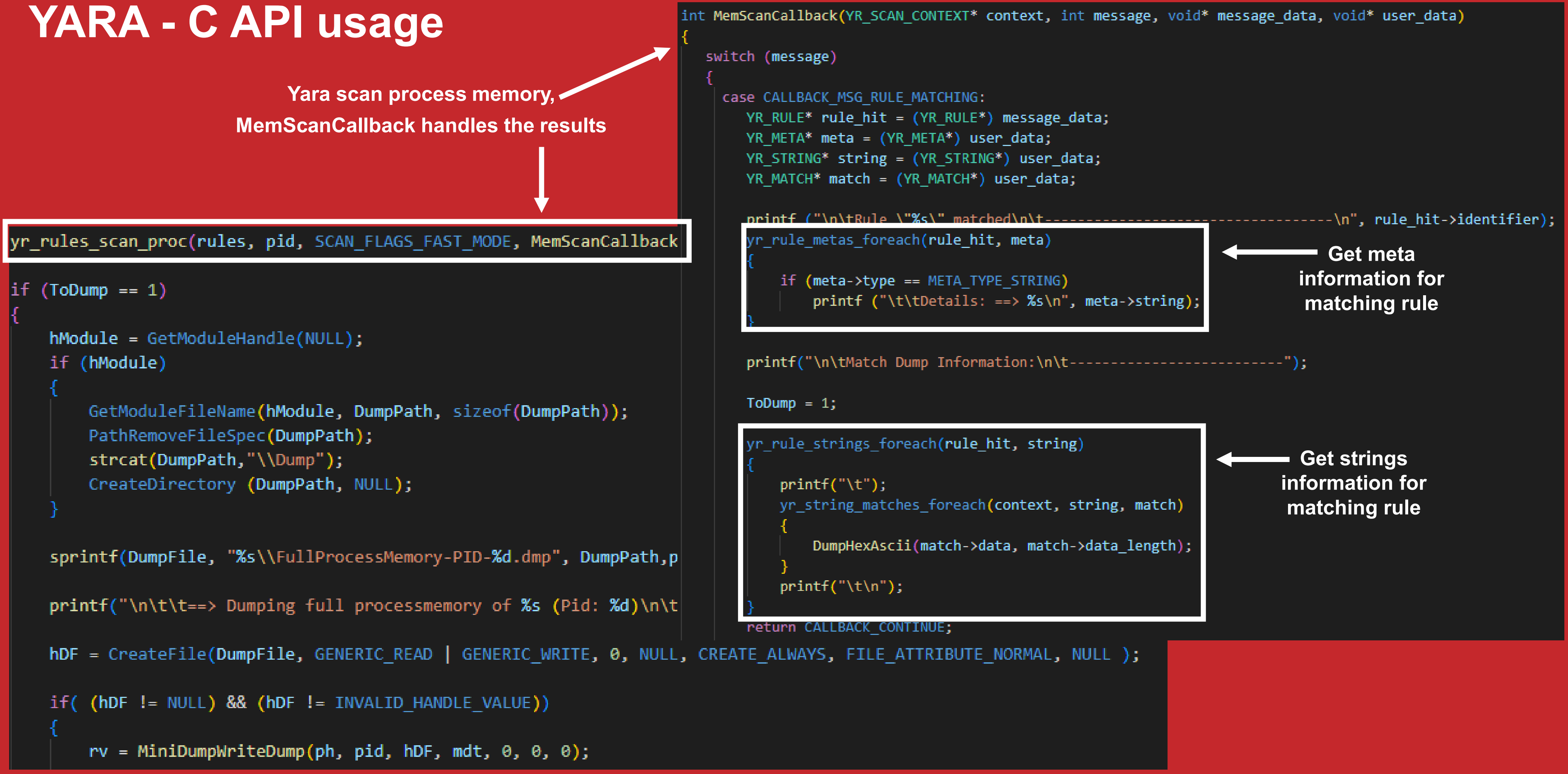
Compile rules from a string buffer

Compile rules from file

Get compiled rules from compiler

YARA - C API usage

Yara scan process memory, MemScanCallback handles the results



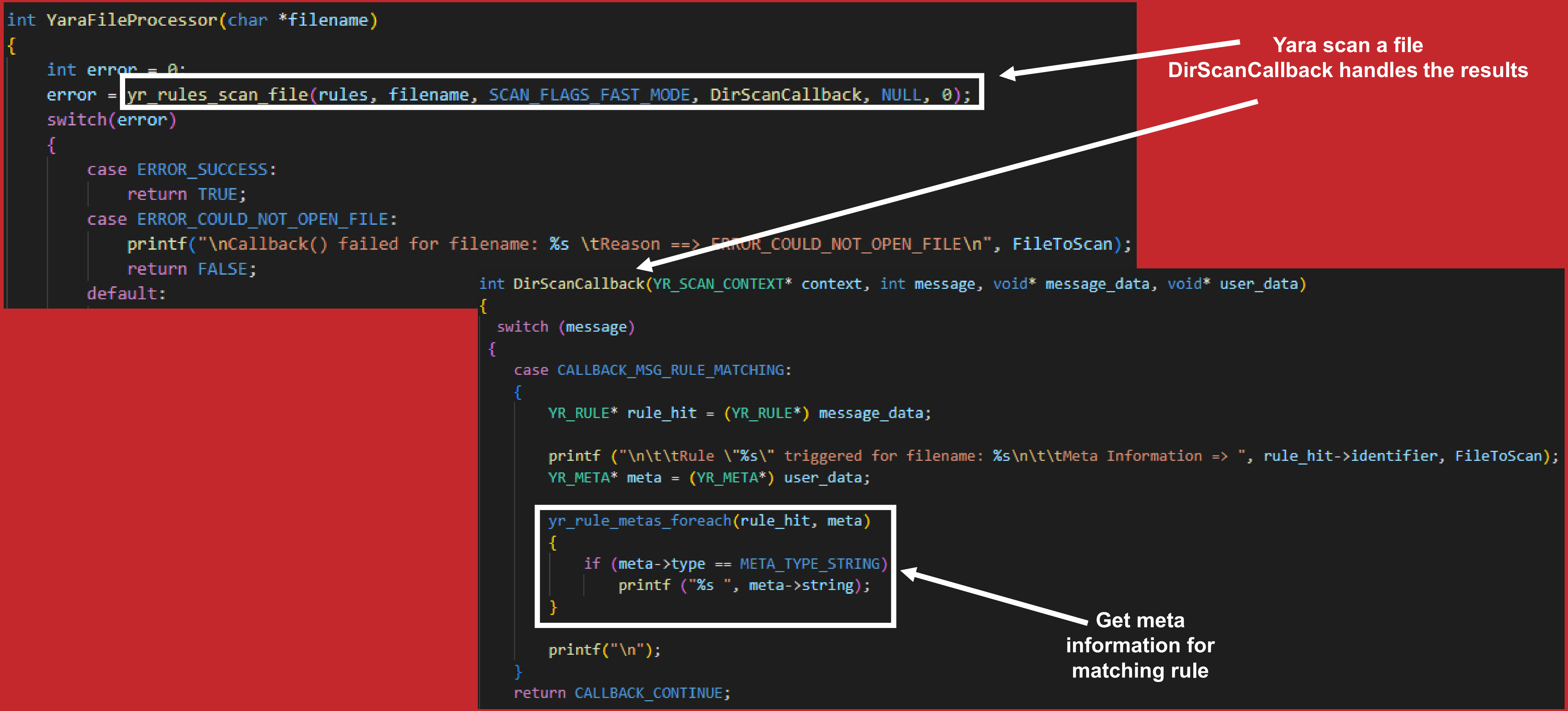
```
int MemScanCallback(YR_SCAN_CONTEXT* context, int message, void* message_data, void* user_data)
{
   switch (message)
   {
     case CALLBACK_MSG_RULE_MATCHING:
        YR_RULE* rule_hit = (YR_RULE*) message_data;
        YR_META* meta = (YR_META*) user_data;
        YR_STRING* string = (YR_STRING*) user_data;
        YR_MATCH* match = (YR_MATCH*) user_data;

        printf ("\n\tRule \"%s\" matched\n\t---------------------------------\n", rule_hit->identifier);
        yr_rule_metas_foreach(rule_hit, meta)
        {
            if (meta->type == META_TYPE_STRING)
                printf ("\t\tDetails: ==> %s\n", meta->string);
        }

        printf("\n\tMatch Dump Information:\n\t-------------------------");

        ToDump = 1;

        yr_rule_strings_foreach(rule_hit, string)
        {
            printf("\t");
            yr_string_matches_foreach(context, string, match)
            {
                DumpHexAscii(match->data, match->data_length);
            }
            printf("\t\n");
        }
        return CALLBACK_CONTINUE;
```

```
yr_rules_scan_proc(rules, pid, SCAN_FLAGS_FAST_MODE, MemScanCallback

if (ToDump == 1)
{
    hModule = GetModuleHandle(NULL);
    if (hModule)
    {
        GetModuleFileName(hModule, DumpPath, sizeof(DumpPath));
        PathRemoveFileSpec(DumpPath);
        strcat(DumpPath,"\\Dump");
        CreateDirectory (DumpPath, NULL);
    }

    sprintf(DumpFile, "%s\\FullProcessMemory-PID-%d.dmp", DumpPath,p

    printf("\n\t\t==> Dumping full processmemory of %s (Pid: %d)\n\t

    hDF = CreateFile(DumpFile, GENERIC_READ | GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL );

    if( (hDF != NULL) && (hDF != INVALID_HANDLE_VALUE))
    {
        rv = MiniDumpWriteDump(ph, pid, hDF, mdt, 0, 0, 0);
```

Get meta information for matching rule

Get strings information for matching rule

YARA - C API usage



```
int YaraFileProcessor(char *filename)
{
    int error = 0;
    error = yr_rules_scan_file(rules, filename, SCAN_FLAGS_FAST_MODE, DirScanCallback, NULL, 0);
    switch(error)
    {
        case ERROR_SUCCESS:
            return TRUE;
        case ERROR_COULD_NOT_OPEN_FILE:
            printf("\nCallback() failed for filename: %s \tReason ==> ERROR_COULD_NOT_OPEN_FILE\n", FileToScan);
            return FALSE;
        default:
```

Yara scan a file
DirScanCallback handles the results

```
int DirScanCallback(YR_SCAN_CONTEXT* context, int message, void* message_data, void* user_data)
{
  switch (message)
 {
    case CALLBACK_MSG_RULE_MATCHING:
    {
        YR_RULE* rule_hit = (YR_RULE*) message_data;

        printf ("\n\t\tRule \"%s\" triggered for filename: %s\n\t\tMeta Information => ", rule_hit->identifier, FileToScan);
        YR_META* meta = (YR_META*) user_data;

        yr_rule_metas_foreach(rule_hit, meta)
        {
            if (meta->type == META_TYPE_STRING)
                printf ("%s ", meta->string);
        }

        printf("\n");
    }
    return CALLBACK_CONTINUE;
```

Get meta information for matching rule

YARA - Challenge 1 (Cobaltstrike Shellcode)

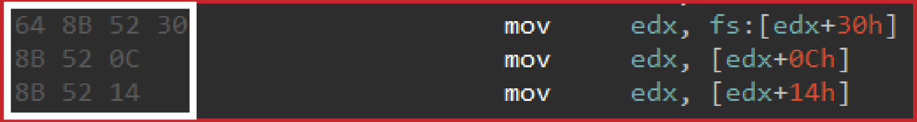
→ For this challenge **analyze** the different **Cobalt Strike shellcode samples** with IDA Pro or another disassembler of your choice, **identify 5 detection patterns matching** for **all samples**, where **1 of them** is a **regular expression. Write** a YARA **rule** and **test** it **against** the **given samples**.

→ **Samples** can be **found** in the **folder C:\YARA-Workshop\Challenges\01-CS_SMB_Stager_X86_Shellcode**

→ **30 minutes for** this **exercise**

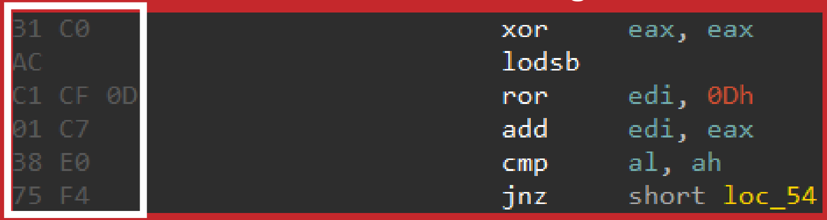
YARA - Challenge 1 (Cobaltstrike Shellcode) - Suitable patterns for rule

Find Kernel32Base
(for more flexibility in the pattern 0x52 values should be replaced by ?? to make a catch even when other registers as edx are used)

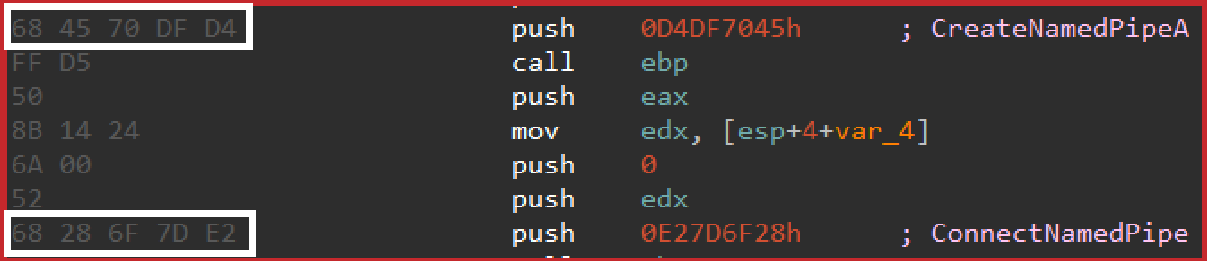


```
64 8B 52 30          mov     edx, fs:[edx+30h]
8B 52 0C             mov     edx, [edx+0Ch]
8B 52 14             mov     edx, [edx+14h]
```

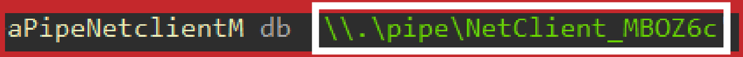
Function Hashing



```
31 C0                xor     eax, eax
AC                   lodsb
C1 CF 0D             ror     edi, 0Dh
01 C7                add     edi, eax
38 E0                cmp     al, ah
75 F4                jnz     short loc_54
```



```
68 45 70 DF D4       push    0D4DF7045h      ; CreateNamedPipeA
FF D5                call    ebp
50                   push    eax
8B 14 24             mov     edx, [esp+4+var_4]
6A 00                push    0
52                   push    edx
68 28 6F 7D E2       push    0E27D6F28h      ; ConnectNamedPipe
```



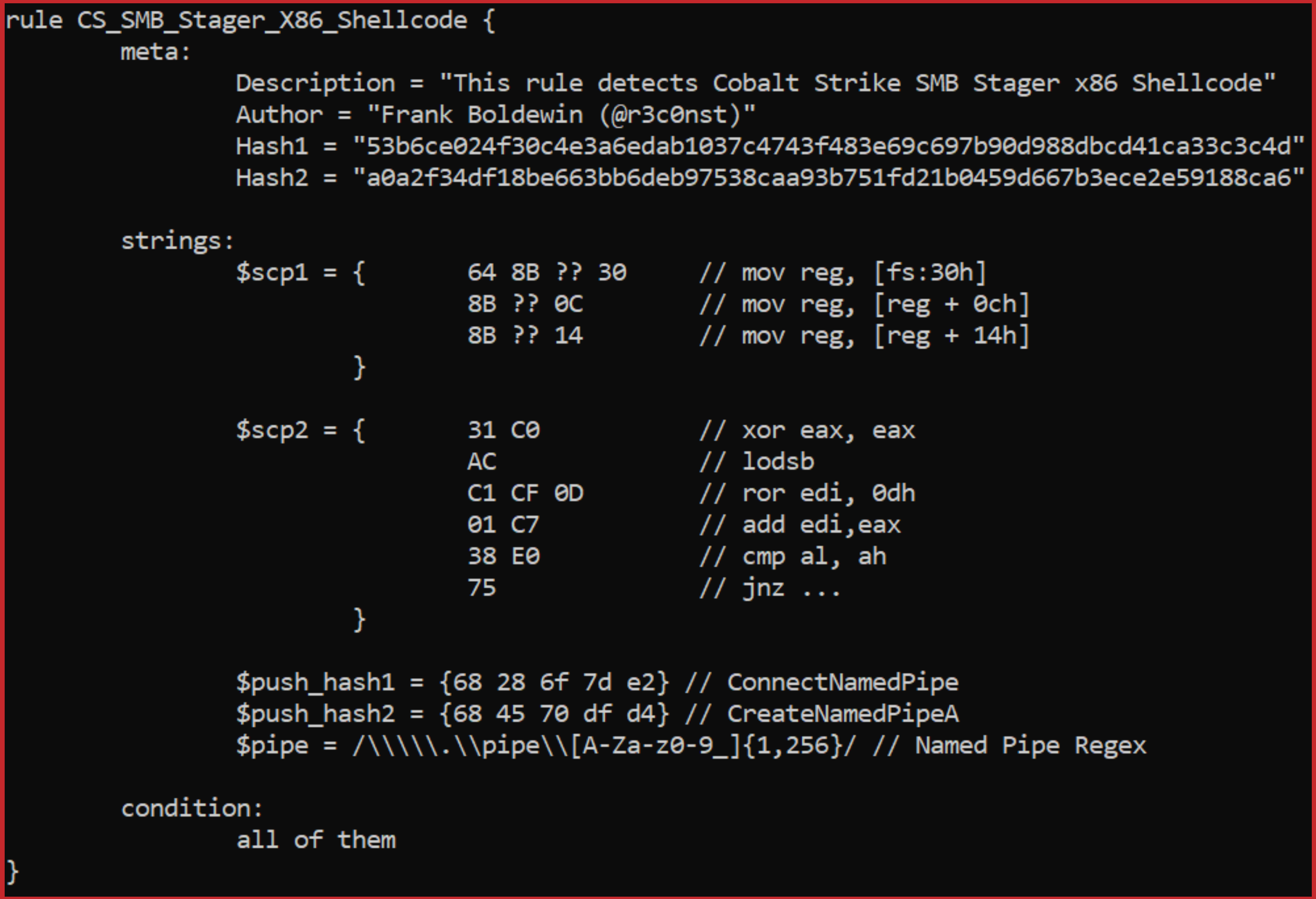
```
aPipeNetclientM db  \\.\pipe\NetClient_MBOZ6c
```

https://learn.microsoft.com/en-us/windows/win32/ipc/pipe-names

"The entire pipe name string can be up to 256 characters long."

Derived from many CS named pipes itw the following regex should be good to go → **/\\\\\.\\pipe\\[A-Za-z0-9_]{1,256}/**

YARA - Challenge 1 (Cobaltstrike Shellcode) - Solution



```
rule CS_SMB_Stager_X86_Shellcode {
        meta:
                Description = "This rule detects Cobalt Strike SMB Stager x86 Shellcode"
                Author = "Frank Boldewin (@r3c0nst)"
                Hash1 = "53b6ce024f30c4e3a6edab1037c4743f483e69c697b90d988dbcd41ca33c3c4d"
                Hash2 = "a0a2f34df18be663bb6deb97538caa93b751fd21b0459d667b3ece2e59188ca6"

        strings:
                $scp1 = {       64 8B ?? 30     // mov reg, [fs:30h]
                                8B ?? 0C        // mov reg, [reg + 0ch]
                                8B ?? 14        // mov reg, [reg + 14h]
                        }

                $scp2 = {       31 C0           // xor eax, eax
                                AC              // lodsb
                                C1 CF 0D        // ror edi, 0dh
                                01 C7           // add edi,eax
                                38 E0           // cmp al, ah
                                75              // jnz ...
                        }

                $push_hash1 = {68 28 6f 7d e2} // ConnectNamedPipe
                $push_hash2 = {68 45 70 df d4} // CreateNamedPipeA
                $pipe = /\\\\\.\\pipe\\[A-Za-z0-9_]{1,256}/ // Named Pipe Regex

        condition:
                all of them
}
```

YARA - Challenge 2 (Redline Stealer)

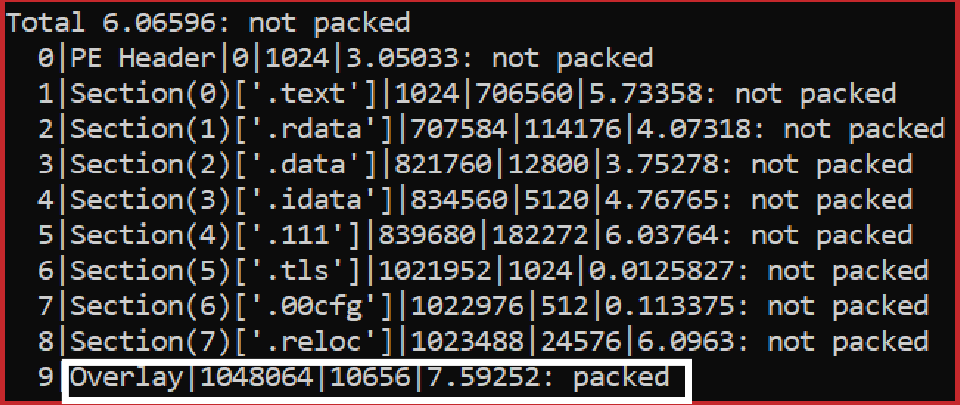
→ For this challenge **analyze** the different **Redline Stealer samples by its file format characteristics** with a PE Editor and other useful tools and **identify at least 6 commonalities**. Make **use** of the **modules pe**, **math** and **hash** in your YARA rule and **test** it **against** the **given samples**.

→ **Samples** can be **found** in the **folder C:\YARA-Workshop\Challenges\02-Redline_Stealer**

→ **30 minutes for** this **exercise**

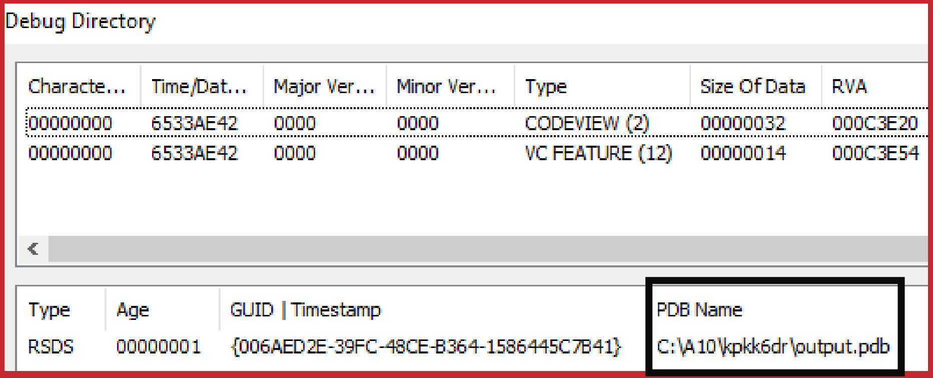
YARA - Challenge 2 (Redline Stealer) - Suitable patterns for rule 1/2

PE has overlay and is packed (Shannon entropy > 7)
Check with Detect It Easy v3.04, e.g. → diec.exe -e 0f99677095ca8199a4988003afe3eb753de31f6137225ff3c90bbe6ecc9c2dbc



```
Total 6.06596: not packed
  0|PE Header|0|1024|3.05033: not packed
  1|Section(0)['.text']|1024|706560|5.73358: not packed
  2|Section(1)['.rdata']|707584|114176|4.07318: not packed
  3|Section(2)['.data']|821760|12800|3.75278: not packed
  4|Section(3)['.idata']|834560|5120|4.76765: not packed
  5|Section(4)['.111']|839680|182272|6.03764: not packed
  6|Section(5)['.tls']|1021952|1024|0.0125827: not packed
  7|Section(6)['.00cfg']|1022976|512|0.113375: not packed
  8|Section(7)['.reloc']|1023488|24576|6.0963: not packed
  9|Overlay|1048064|10656|7.59252: packed
```

Debug information artifact → PDB Path
Check with PETools or adequate editors



Debug Directory

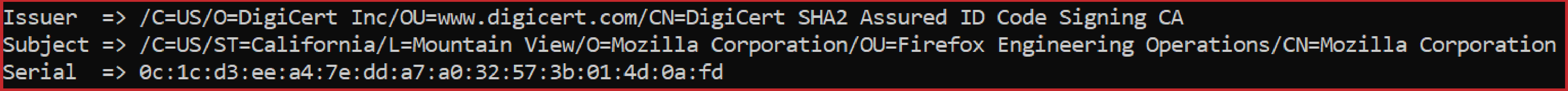
Characte...	Time/Dat...	Major Ver...	Minor Ver...	Type	Size Of Data	RVA
00000000	6533AE42	0000	0000	CODEVIEW (2)	00000032	000C3E20
00000000	6533AE42	0000	0000	VC FEATURE (12)	00000014	000C3E54

Type	Age	GUID | Timestamp	PDB Name
RSDS	00000001	{006AED2E-39FC-48CE-B364-1586445C7B41}	C:\A10\kpkk6dr\output.pdb

YARA - Challenge 2 (Redline Stealer) - Suitable patterns for rule 2/2

Authenticode certificate data

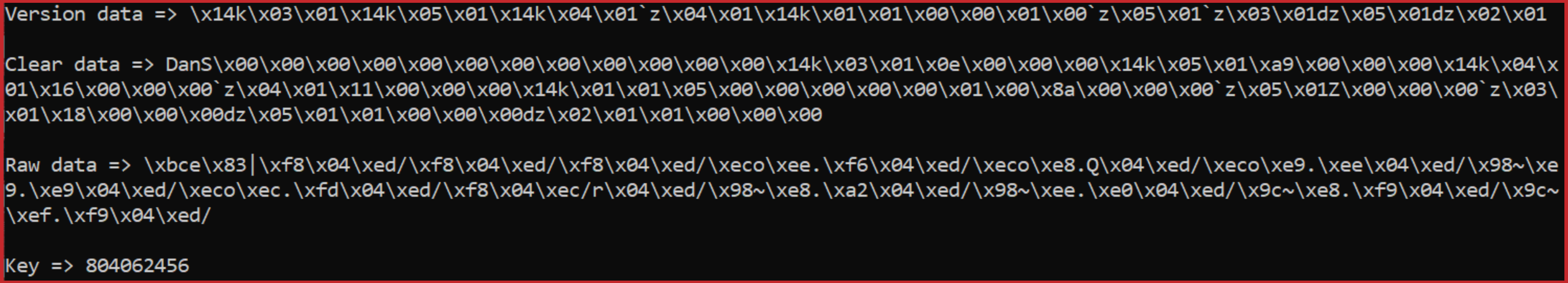
yara64 C:\YARA-Workshop\Helpers\BasicAuthenticodeInfos.yar 0f99677095ca8199a4988003afe3eb753de31f6137225ff3c90bbe6ecc9c2dbc



```
Issuer  => /C=US/O=DigiCert Inc/OU=www.digicert.com/CN=DigiCert SHA2 Assured ID Code Signing CA
Subject => /C=US/ST=California/L=Mountain View/O=Mozilla Corporation/OU=Firefox Engineering Operations/CN=Mozilla Corporation
Serial  => 0c:1c:d3:ee:a4:7e:dd:a7:a0:32:57:3b:01:4d:0a:fd
```

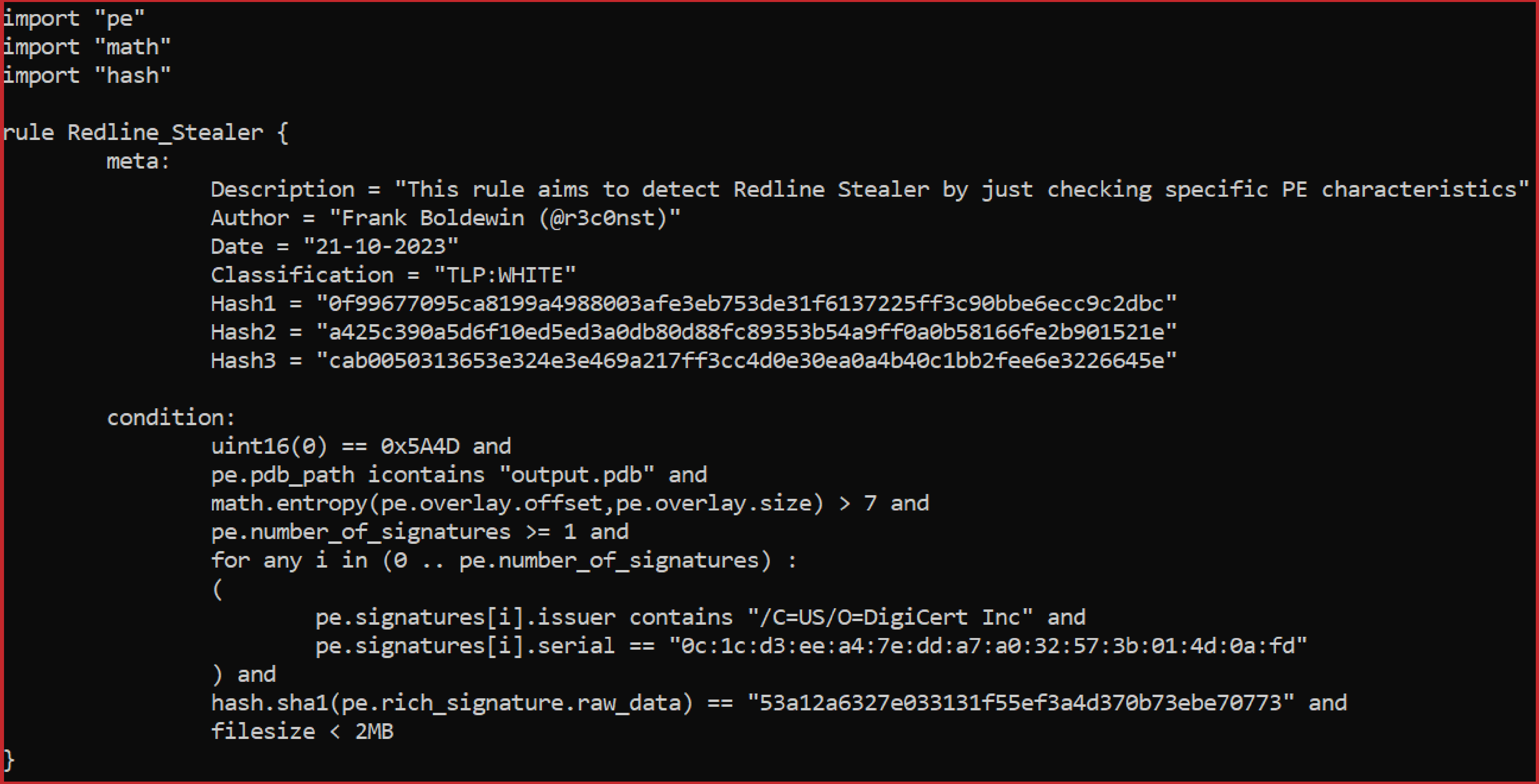
Rich Header data

yara64 C:\YARA-Workshop\Helpers\RichHeaderInfos.yar 0f99677095ca8199a4988003afe3eb753de31f6137225ff3c90bbe6ecc9c2dbc



```
Version data => \x14k\x03\x01\x14k\x05\x01\x14k\x04\x01`z\x04\x01\x14k\x01\x01\x00\x00\x01\x00`z\x05\x01`z\x03\x01dz\x05\x01dz\x02\x01

Clear data => DanS\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x14k\x03\x01\x0e\x00\x00\x00\x14k\x05\x01\xa9\x00\x00\x00\x14k\x04\x
01\x16\x00\x00\x00`z\x04\x01\x11\x00\x00\x00\x14k\x01\x01\x05\x00\x00\x00\x00\x00\x01\x00\x8a\x00\x00\x00`z\x05\x01Z\x00\x00\x00`z\x03\
x01\x18\x00\x00\x00dz\x05\x01\x01\x00\x00\x00dz\x02\x01\x01\x00\x00\x00

Raw data => \xbce\x83|\xf8\x04\xed/\xf8\x04\xed/\xf8\x04\xed/\xeco\xee.\xf6\x04\xed/\xeco\xe8.Q\x04\xed/\xeco\xe9.\xee\x04\xed/\x98~\xe
9.\xe9\x04\xed/\xeco\xec.\xfd\x04\xed/\xf8\x04\xec/r\x04\xed/\x98~\xe8.\xa2\x04\xed/\x98~\xee.\xe0\x04\xed/\x9c~\xe8.\xf9\x04\xed/\x9c~
\xef.\xf9\x04\xed/

Key => 804062456
```

YARA - Challenge 2 (Redline Stealer) - Solution



```
import "pe"
import "math"
import "hash"

rule Redline_Stealer {
        meta:
                Description = "This rule aims to detect Redline Stealer by just checking specific PE characteristics"
                Author = "Frank Boldewin (@r3c0nst)"
                Date = "21-10-2023"
                Classification = "TLP:WHITE"
                Hash1 = "0f99677095ca8199a4988003afe3eb753de31f6137225ff3c90bbe6ecc9c2dbc"
                Hash2 = "a425c390a5d6f10ed5ed3a0db80d88fc89353b54a9ff0a0b58166fe2b901521e"
                Hash3 = "cab0050313653e324e3e469a217ff3cc4d0e30ea0a4b40c1bb2fee6e3226645e"

        condition:
                uint16(0) == 0x5A4D and
                pe.pdb_path icontains "output.pdb" and
                math.entropy(pe.overlay.offset,pe.overlay.size) > 7 and
                pe.number_of_signatures >= 1 and
                for any i in (0 .. pe.number_of_signatures) :
                (
                        pe.signatures[i].issuer contains "/C=US/O=DigiCert Inc" and
                        pe.signatures[i].serial == "0c:1c:d3:ee:a4:7e:dd:a7:a0:32:57:3b:01:4d:0a:fd"
                ) and
                hash.sha1(pe.rich_signature.raw_data) == "53a12a6327e033131f55ef3a4d370b73ebe70773" and
                filesize < 2MB
}
```

YARA - Challenge 3 (Kiteshield ELF Protector)

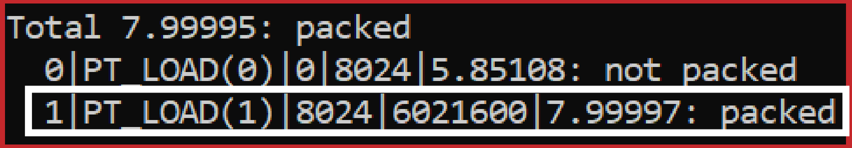
→ For this challenge **analyze** the different **Kiteshield** protected **samples** with **IDA Pro** (or adequate tool) and **XELFViewer** to identify **at least 1 unique code pattern** and **7 ELF** file **characteristics**. Make **use** of the YARA **modules elf and math**. Then **test** the **rule against** the given **samples** and **assure** it **matches all**.

→ **Samples** can be **found** in the **folder C:\YARA-Workshop\Challenges\03-Kiteshield_ELF_Protector**

→ **30 minutes for** this **exercise**

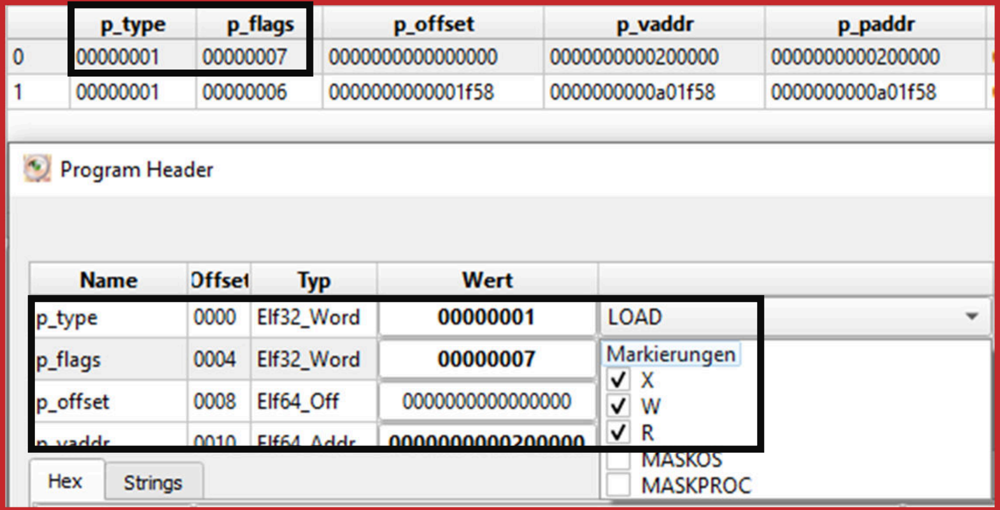
YARA - Challenge 3 (Kiteshield ELF Protector) - Suitable patterns for rule

Segment 1 is packed (Shannon entropy > 7) across all given samples
Check with Detect It Easy v3.04, e.g. → diec.exe -e 283129b67793bcb6ab9371eb8fab52ef88aac400d282d02f1bfebb98e21f01d1



```
Total 7.99995: packed
  0|PT_LOAD(0)|0|8024|5.85108: not packed
  1|PT_LOAD(1)|8024|6021600|7.99997: packed
```

Segment 0 is type LOAD and has set flags RWX
Check with XELFVIEWER

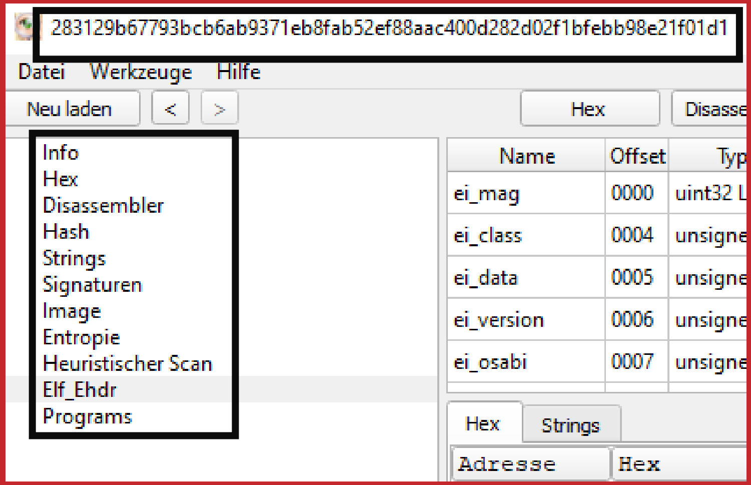


	p_type	p_flags	p_offset	p_vaddr	p_paddr
0	00000001	00000007	0000000000000000	0000000000200000	0000000000200000
1	00000001	00000006	0000000000001f58	0000000000a01f58	0000000000a01f58

Program Header

Name	Offset	Typ	Wert	
p_type	0000	Elf32_Word	00000001	LOAD
p_flags	0004	Elf32_Word	00000007	Markierungen ☑ X ☑ W ☑ R ☐ MASKOS ☐ MASKPROC
p_offset	0008	Elf64_Off	0000000000000000	
p_vaddr	0010	Elf64_Addr	0000000000200000	

Samples have no sections
Check with XELFVIEWER

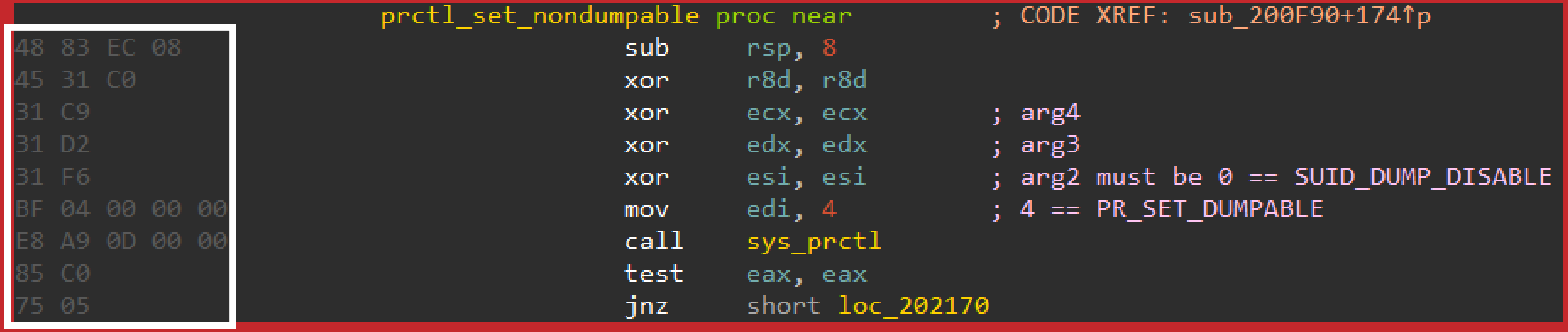


283129b67793bcb6ab9371eb8fab52ef88aac400d282d02f1bfebb98e21f01d1

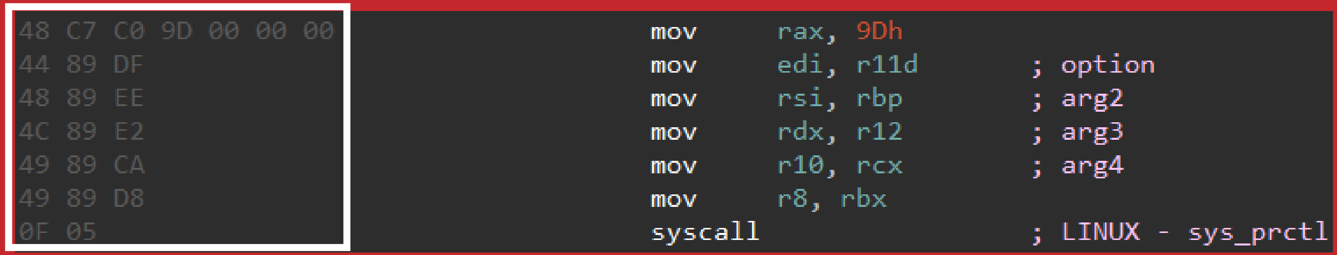
Name	Offset	Typ
ei_mag	0000	uint32 L
ei_class	0004	unsigne
ei_data	0005	unsigne
ei_version	0006	unsigne
ei_osabi	0007	unsigne

YARA - Challenge 3 (Kiteshield ELF Protector) - Suitable patterns for rule

One of Kiteshield's Antidebug features is to set the process dumpable flag to 0, to make ptrace attaching impossible and to disable coredumping.

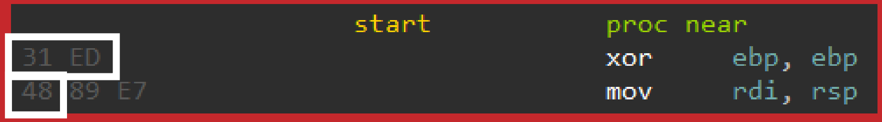


```
                           prctl_set_nondumpable proc near          ; CODE XREF: sub_200F90+174↑p
48 83 EC 08                                sub     rsp, 8
45 31 C0                                   xor     r8d, r8d
31 C9                                      xor     ecx, ecx         ; arg4
31 D2                                      xor     edx, edx         ; arg3
31 F6                                      xor     esi, esi         ; arg2 must be 0 == SUID_DUMP_DISABLE
BF 04 00 00 00                             mov     edi, 4           ; 4 == PR_SET_DUMPABLE
E8 A9 0D 00 00                             call    sys_prctl
85 C0                                      test    eax, eax
75 05                                      jnz     short loc_202170
```

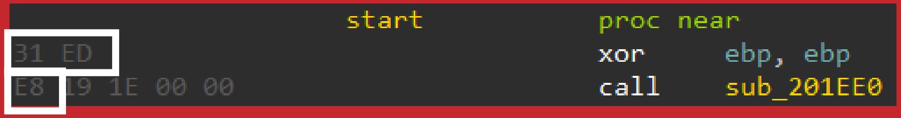


```
48 C7 C0 9D 00 00 00                       mov     rax, 9Dh
44 89 DF                                   mov     edi, r11d        ; option
48 89 EE                                   mov     rsi, rbp         ; arg2
4C 89 E2                                   mov     rdx, r12         ; arg3
49 89 CA                                   mov     r10, rcx         ; arg4
49 89 D8                                   mov     r8, rbx
0F 05                                      syscall                  ; LINUX - sys_prctl
```

First 3 bytes are 31 ED [48 | E8] across given samples at elf.entrypoint

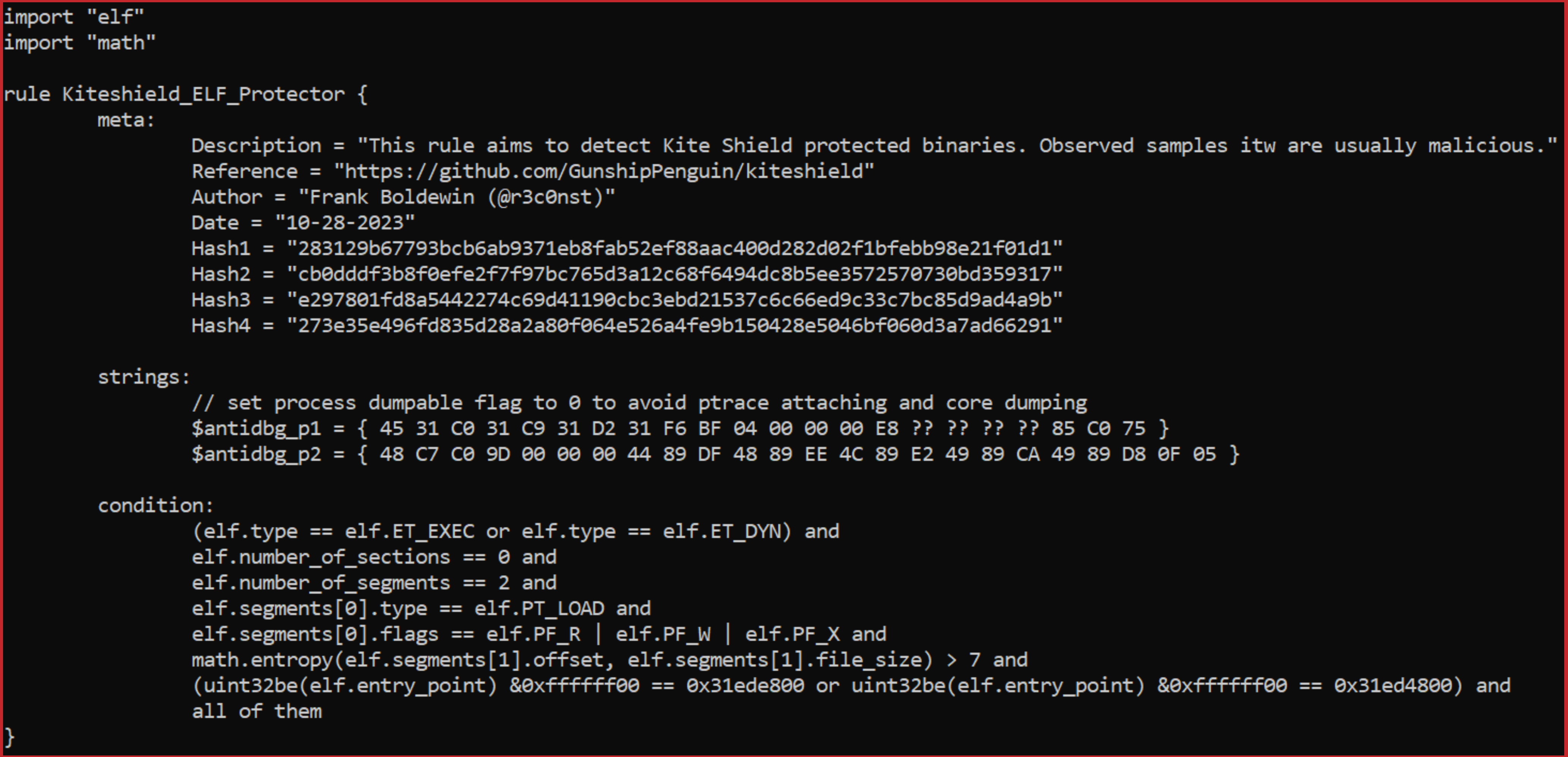


```
                      start           proc near
31 ED                                 xor     ebp, ebp
48 89 E7                              mov     rdi, rsp
```



```
                      start           proc near
31 ED                                 xor     ebp, ebp
E8 19 1E 00 00                        call    sub_201EE0
```

YARA - Challenge 3 (Kiteshield ELF Protector) - Solution



```
import "elf"
import "math"

rule Kiteshield_ELF_Protector {
        meta:
                Description = "This rule aims to detect Kite Shield protected binaries. Observed samples itw are usually malicious."
                Reference = "https://github.com/GunshipPenguin/kiteshield"
                Author = "Frank Boldewin (@r3c0nst)"
                Date = "10-28-2023"
                Hash1 = "283129b67793bcb6ab9371eb8fab52ef88aac400d282d02f1bfebb98e21f01d1"
                Hash2 = "cb0dddf3b8f0efe2f7f97bc765d3a12c68f6494dc8b5ee3572570730bd359317"
                Hash3 = "e297801fd8a5442274c69d41190cbc3ebd21537c6c66ed9c33c7bc85d9ad4a9b"
                Hash4 = "273e35e496fd835d28a2a80f064e526a4fe9b150428e5046bf060d3a7ad66291"

        strings:
                // set process dumpable flag to 0 to avoid ptrace attaching and core dumping
                $antidbg_p1 = { 45 31 C0 31 C9 31 D2 31 F6 BF 04 00 00 00 E8 ?? ?? ?? ?? 85 C0 75 }
                $antidbg_p2 = { 48 C7 C0 9D 00 00 00 44 89 DF 48 89 EE 4C 89 E2 49 89 CA 49 89 D8 0F 05 }

        condition:
                (elf.type == elf.ET_EXEC or elf.type == elf.ET_DYN) and
                elf.number_of_sections == 0 and
                elf.number_of_segments == 2 and
                elf.segments[0].type == elf.PT_LOAD and
                elf.segments[0].flags == elf.PF_R | elf.PF_W | elf.PF_X and
                math.entropy(elf.segments[1].offset, elf.segments[1].file_size) > 7 and
                (uint32be(elf.entry_point) &0xffffff00 == 0x31ede800 or uint32be(elf.entry_point) &0xffffff00 == 0x31ed4800) and
                all of them
}
```

YARA - Challenge 4 (Malicious OneNote)

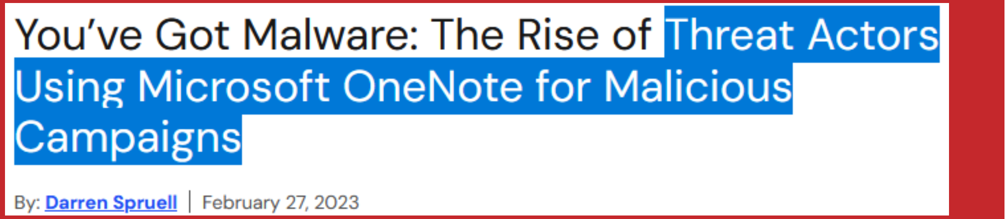
→ For this challenge first **find blog posts about malicious OneNote usage** in phishing campaigns, next to this **study** the **file format specs** to **understand** its **structures** and then **analyze** the **different OneNote samples with** a **hex editor** of your choice to **identify** the different **malicious** embedded **objects**. Afterwards **write** a YARA **rule** to **identify all of them** and **test** it **against** the **given samples** and **assure** it **matches all**.
Hint → Magic values and parsing offsets will help writing a decent rule.

→ **Samples** can be **found** in the **folder C:\YARA-Workshop\Challenges\04-Malicious_OneNote**

→ **45 minutes for** this **exercise**

YARA - Challenge 4 (Malicious OneNote) - Understanding the threat

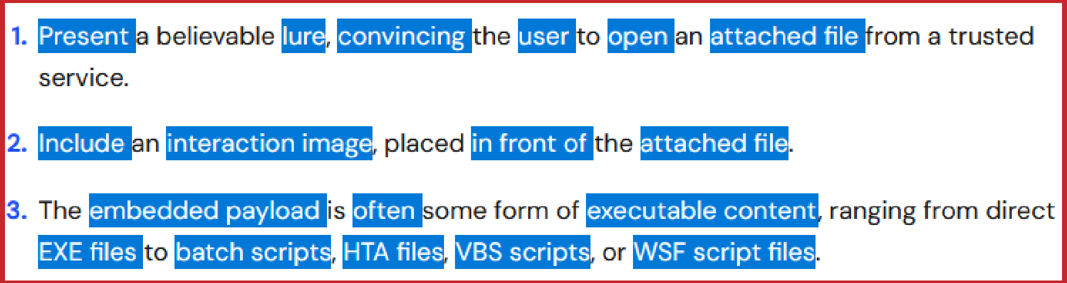
The threat



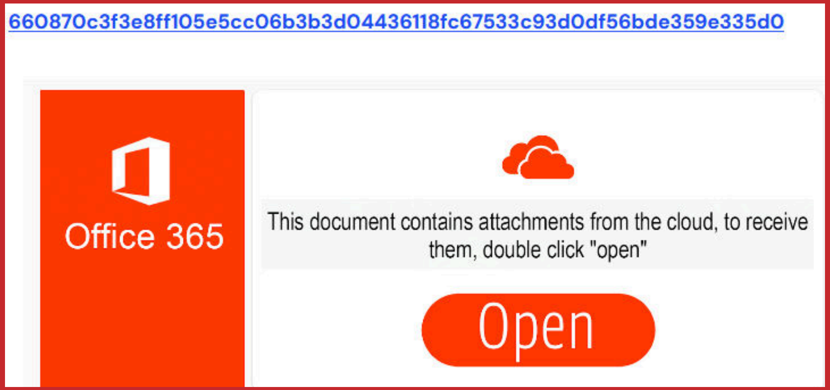
You've Got Malware: The Rise of Threat Actors Using Microsoft OneNote for Malicious Campaigns

By: Darren Spruell | February 27, 2023

Modus Operandi



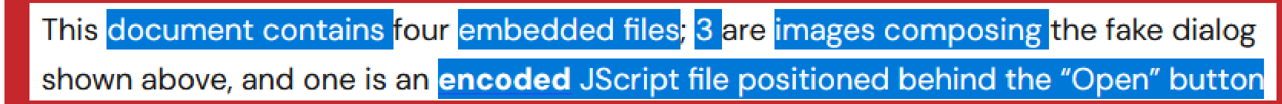
1. Present a believable lure, convincing the user to open an attached file from a trusted service.
2. Include an interaction image, placed in front of the attached file.
3. The embedded payload is often some form of executable content, ranging from direct EXE files to batch scripts, HTA files, VBS scripts, or WSF script files.



660870c3f3e8ff105e5cc06b3b3d04436118fc67533c93d0df56bde359e335d0

This document contains attachments from the cloud, to receive them, double click "open"

Example content of a malicious OneNote file

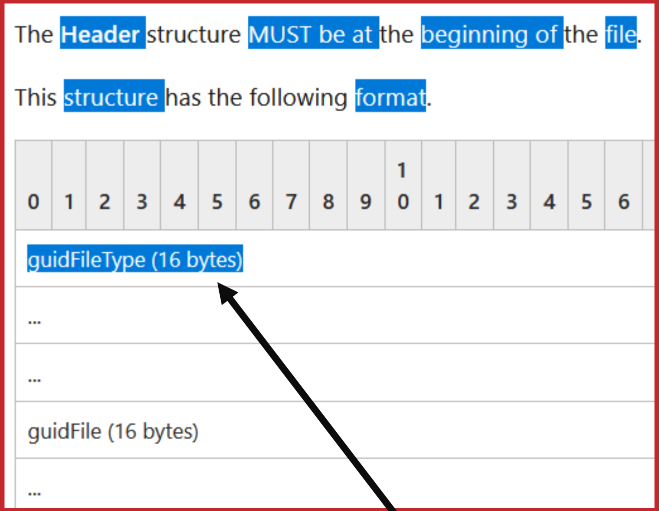


This document contains four embedded files; 3 are images composing the fake dialog shown above, and one is an **encoded** JScript file positioned behind the "Open" button

https://inquest.net/blog/youve-got-malware-rise-threat-actors-using-microsoft-onenote-malicious-campaigns/

YARA - Challenge 4 (Malicious OneNote) – Understanding the file format

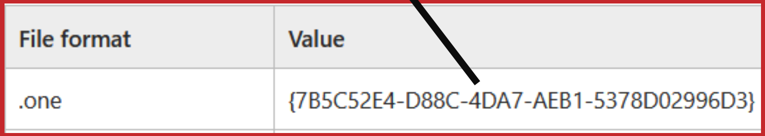
2.3.1 Header



The Header structure MUST be at the beginning of the file.

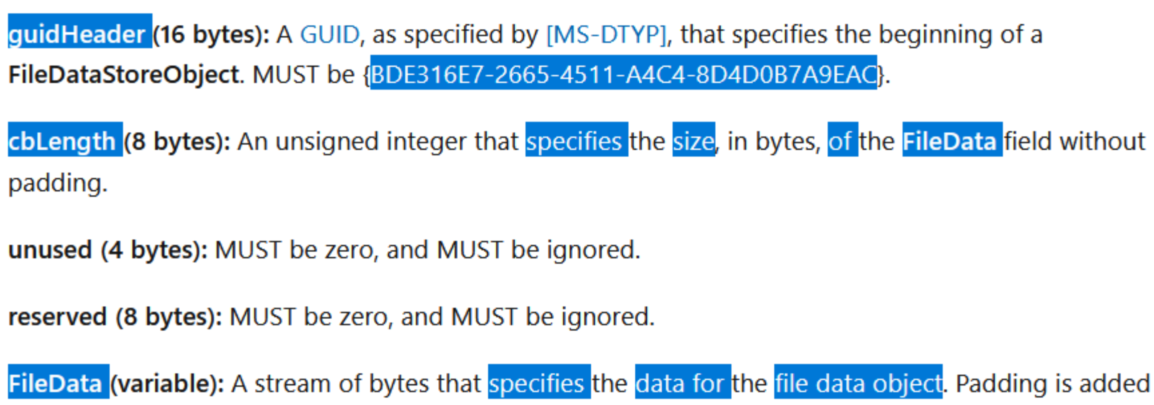
This structure has the following format.

0	1	2	3	4	5	6	7	8	9	10	1	2	3	4	5	6
guidFileType (16 bytes)																
...																
...																
guidFile (16 bytes)																
...																



File format	Value
.one	{7B5C52E4-D88C-4DA7-AEB1-5378D02996D3}

2.6.13 FileDataStoreObject



guidHeader (16 bytes): A GUID, as specified by [MS-DTYP], that specifies the beginning of a **FileDataStoreObject**. MUST be {BDE316E7-2665-4511-A4C4-8D4D0B7A9EAC}.

cbLength (8 bytes): An unsigned integer that specifies the size, in bytes, of the **FileData** field without padding.

unused (4 bytes): MUST be zero, and MUST be ignored.

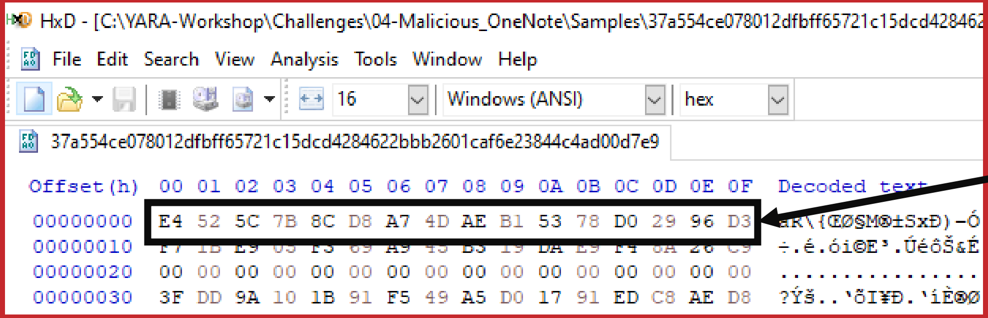
reserved (8 bytes): MUST be zero, and MUST be ignored.

FileData (variable): A stream of bytes that specifies the data for the file data object. Padding is added

https://learn.microsoft.com/en-us/openspecs/office_file_formats/ms-onestore/2b394c6b-8788-441f-b631-da1583d772fd
https://learn.microsoft.com/en-us/openspecs/office_file_formats/ms-onestore/8806fd18-6735-4874-b111-227b83eaac26

YARA - Challenge 4 (Malicious OneNote) - Parsing for malicious entries

Malicious OneNote file → Header starts with guidFileType

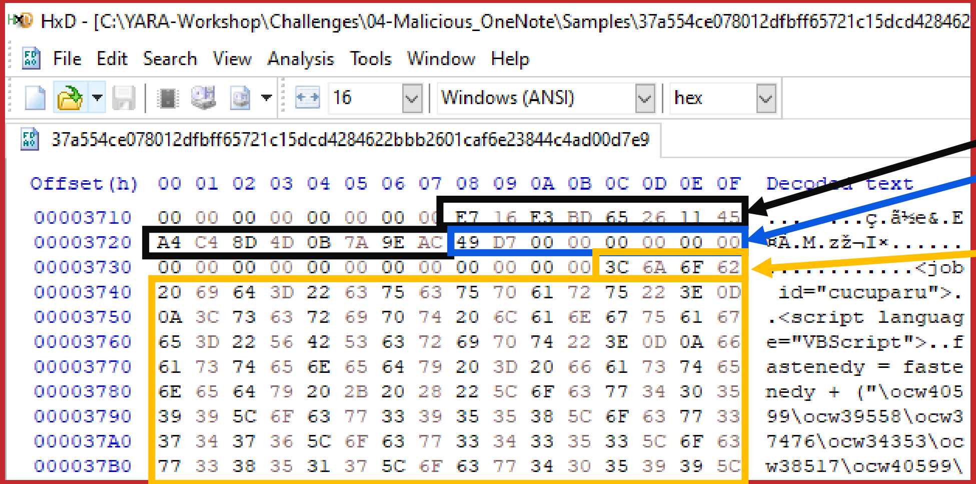

HxD - [C:\YARA-Workshop\Challenges\04-Malicious_OneNote\Samples\37a554ce078012dfbff65721c15dcd428462
File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
37a554ce078012dfbff65721c15dcd4284622bbb2601caf6e23844c4ad00d7e9

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00000000   E4 52 5C 7B 8C D8 A7 4D AE B1 53 78 D0 29 96 D3  äR\{ŒØ§M®±SxÐ)–Ó
00000010   F7 1B E9 05 F3 69 A9 45 B3 19 DA E9 F4 8A 26 C9  ÷.é.óiÉ³.ÚéôŠ&É
00000020   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
00000030   3F DD 9A 10 1B 91 F5 49 A5 D0 17 91 ED C8 AE D8  ?Ýš..'õI¥Ð.'íÈ®Ø
```

Note endianness!

File format	Value
.one	{7B5C52E4-D88C-4DA7-AEB1-5378D02996D3}

FileDataStoreObject data guid header


HxD - [C:\YARA-Workshop\Challenges\04-Malicious_OneNote\Samples\37a554ce078012dfbff65721c15dcd428462
File Edit Search View Analysis Tools Window Help
16
Windows (ANSI)
hex
37a554ce078012dfbff65721c15dcd4284622bbb2601caf6e23844c4ad00d7e9

```
Offset(h)  00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F  Decoded text
00003710   00 00 00 00 00 00 00 00 E7 16 E3 BD 65 26 11 45  ........ç.ã½e&.E
00003720   A4 C4 8D 4D 0B 7A 9E AC 49 D7 00 00 00 00 00 00  ¤Ä.M.zž¬I×......
00003730   00 00 00 00 00 00 00 00 00 00 00 00 3C 6A 6F 62  ............<job
00003740   20 69 64 3D 22 63 75 63 75 70 61 72 75 22 3E 0D   id="cucuparu">.
00003750   0A 3C 73 63 72 69 70 74 20 6C 61 6E 67 75 61 67  .<script languag
00003760   65 3D 22 56 42 53 63 72 69 70 74 22 3E 0D 0A 66  e="VBScript">..f
00003770   61 73 74 65 6E 65 64 79 20 3D 20 66 61 73 74 65  astenedy = faste
00003780   6E 65 64 79 20 2B 20 28 22 5C 6F 63 77 34 30 35  nedy + ("\ocw405
00003790   39 39 5C 6F 63 77 33 39 35 35 38 5C 6F 63 77 33  99\ocw39558\ocw3
000037A0   37 34 37 36 5C 6F 63 77 33 34 33 35 33 5C 6F 63  7476\ocw34353\oc
000037B0   77 33 38 35 31 37 5C 6F 63 77 34 30 35 39 39 5C  w38517\ocw40599\
```

Note endianness!

FileDataStoreObject. MUST be {BDE316E7-2665-4511-A4C4-8D4D0B7A9EAC}.

Note endianness!

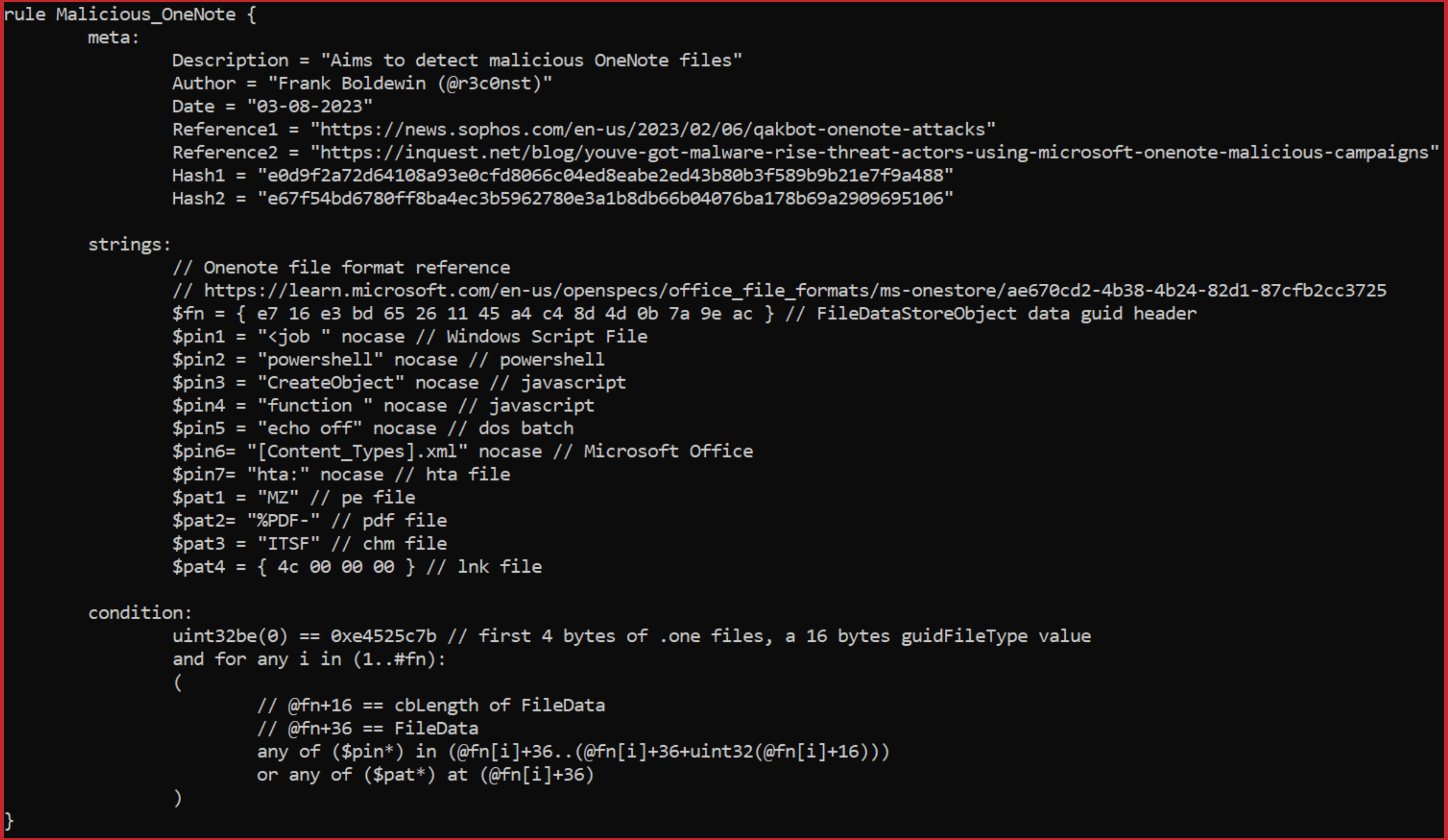
cbLength (8 bytes): An unsigned integer that specifies the size, in bytes, of the **FileData** field

FileData (variable): A stream of bytes that specifies the data for the file data object.

Calculation for parsing
FileDataStoreObject + 16 == cbLength
FileDataStoreObject + 36 == FileData

Remember → OneNote files can have multiple FileDataStoreObjects!

YARA – Challenge 4 (Malicious OneNote) - Solution



```
rule Malicious_OneNote {
        meta:
                Description = "Aims to detect malicious OneNote files"
                Author = "Frank Boldewin (@r3c0nst)"
                Date = "03-08-2023"
                Reference1 = "https://news.sophos.com/en-us/2023/02/06/qakbot-onenote-attacks"
                Reference2 = "https://inquest.net/blog/youve-got-malware-rise-threat-actors-using-microsoft-onenote-malicious-campaigns"
                Hash1 = "e0d9f2a72d64108a93e0cfd8066c04ed8eabe2ed43b80b3f589b9b21e7f9a488"
                Hash2 = "e67f54bd6780ff8ba4ec3b5962780e3a1b8db66b04076ba178b69a2909695106"

        strings:
                // Onenote file format reference
                // https://learn.microsoft.com/en-us/openspecs/office_file_formats/ms-onestore/ae670cd2-4b38-4b24-82d1-87cfb2cc3725
                $fn = { e7 16 e3 bd 65 26 11 45 a4 c4 8d 4d 0b 7a 9e ac } // FileDataStoreObject data guid header
                $pin1 = "<job " nocase // Windows Script File
                $pin2 = "powershell" nocase // powershell
                $pin3 = "CreateObject" nocase // javascript
                $pin4 = "function " nocase // javascript
                $pin5 = "echo off" nocase // dos batch
                $pin6= "[Content_Types].xml" nocase // Microsoft Office
                $pin7= "hta:" nocase // hta file
                $pat1 = "MZ" // pe file
                $pat2= "%PDF-" // pdf file
                $pat3 = "ITSF" // chm file
                $pat4 = { 4c 00 00 00 } // lnk file

        condition:
                uint32be(0) == 0xe4525c7b // first 4 bytes of .one files, a 16 bytes guidFileType value
                and for any i in (1..#fn):
                (
                        // @fn+16 == cbLength of FileData
                        // @fn+36 == FileData
                        any of ($pin*) in (@fn[i]+36..(@fn[i]+36+uint32(@fn[i]+16)))
                        or any of ($pat*) at (@fn[i]+36)
                )
}
```

YARA - Challenge 5 (Play ransomware)

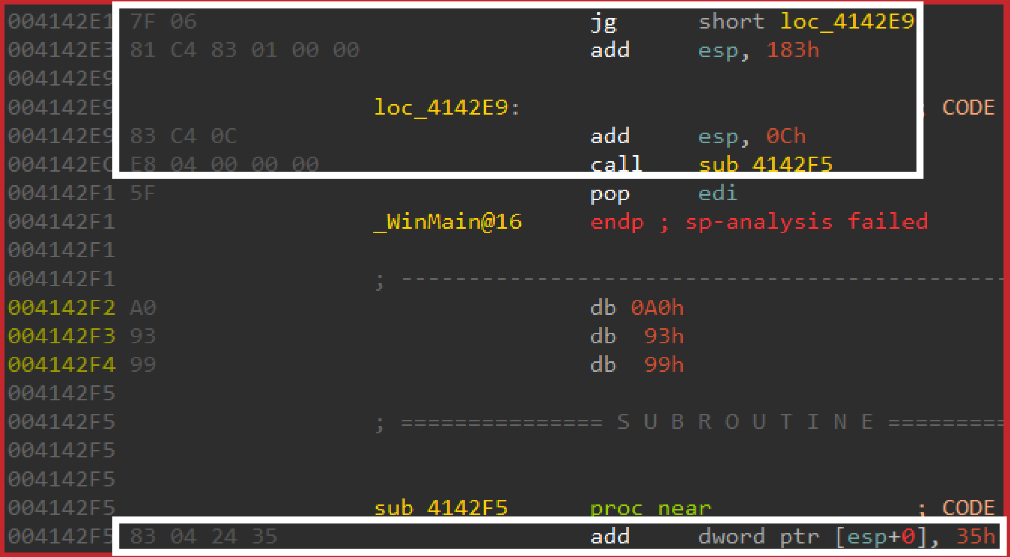
- → For this challenge **analyze** the different **Play ransomware samples** with IDA Pro (or adequate tool) and **identify unique code patterns** to match all given samples (this might be a little tricky due to the polymorphic layer, but there are commonalities which can be used for detection). Next to this **add** at least **2 PE characteristics** to your YARA rule. Then **test** it **against** the **given samples** and **assure** it **matches all**.

- → **Samples** can be **found** in the **folder C:\YARA-Workshop\Challenges\05-PlayRansomwarePolymorphicLayer**

- → **45 minutes for** this **exercise**

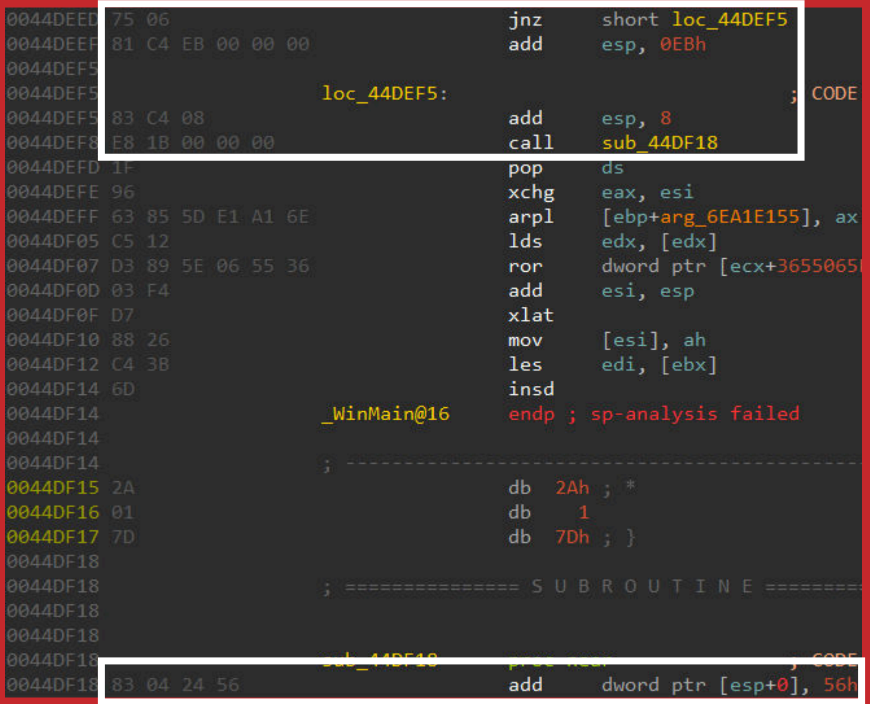
YARA - Challenge 5 (Play ransomware) - Suitable patterns for rule

006ae41910887f0811a3ba2868ef9576bbd265216554850112319af878f06e55



```
004142E1 7F 06                               jg      short loc_4142E9
004142E3 81 C4 83 01 00 00                   add     esp, 183h
004142E9
004142E9                 loc_4142E9:                                    CODE
004142E9 83 C4 0C                            add     esp, 0Ch
004142EC E8 04 00 00 00                      call    sub_4142F5
004142F1 5F                                  pop     edi
004142F1                 _WinMain@16         endp ; sp-analysis failed
004142F1
004142F1                 ; -----------------------------------------------
004142F2 A0                                  db 0A0h
004142F3 93                                  db  93h
004142F4 99                                  db  99h
004142F5
004142F5                 ; =============== S U B R O U T I N E =========
004142F5
004142F5
004142F5                 sub_4142F5          proc near                  ; CODE
004142F5 83 04 24 35                         add     dword ptr [esp+0], 35h
```

28e2f00a7c4b86fe98a184437cc1ea219b7853e4773b592ab83828c1cac77876

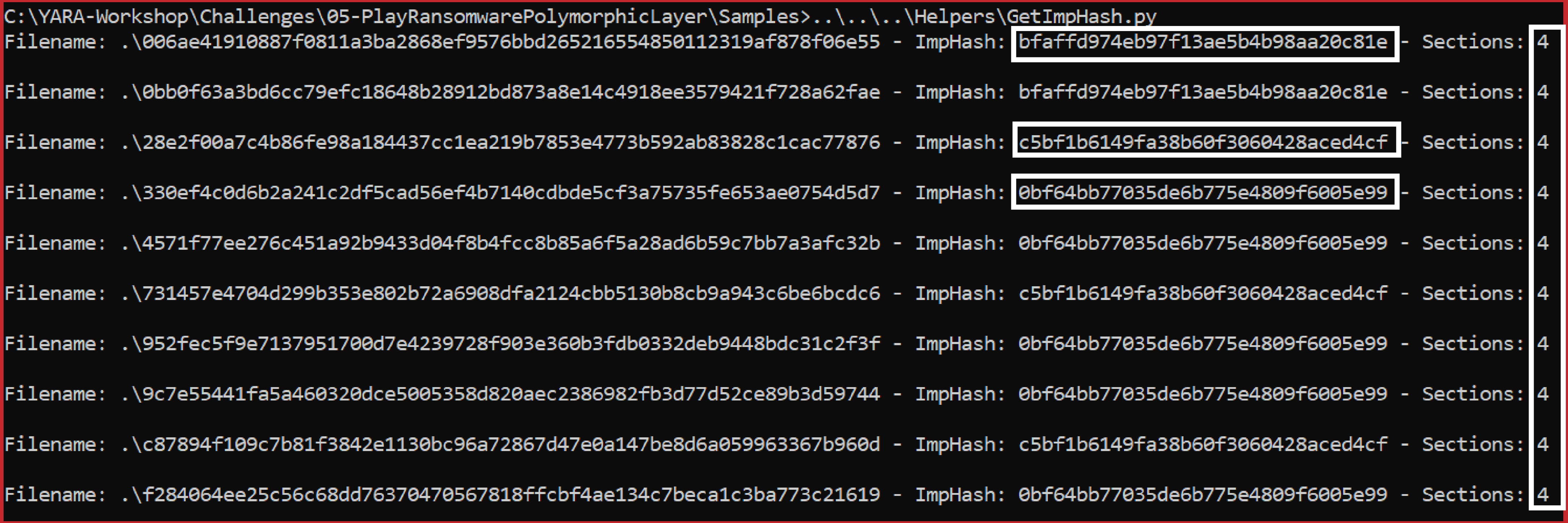


```
0044DEED 75 06                               jnz     short loc_44DEF5
0044DEEF 81 C4 EB 00 00 00                   add     esp, 0EBh
0044DEF5
0044DEF5                 loc_44DEF5:                                    ; CODE
0044DEF5 83 C4 08                            add     esp, 8
0044DEF8 E8 1B 00 00 00                      call    sub_44DF18
0044DEFD 1F                                  pop     ds
0044DEFE 96                                  xchg    eax, esi
0044DEFF 63 85 5D E1 A1 6E                   arpl    [ebp+arg_6EA1E155], ax
0044DF05 C5 12                               lds     edx, [edx]
0044DF07 D3 89 5E 06 55 36                   ror     dword ptr [ecx+3655065
0044DF0D 03 F4                               add     esi, esp
0044DF0F D7                                  xlat
0044DF10 88 26                               mov     [esi], ah
0044DF12 C4 3B                               les     edi, [ebx]
0044DF14 6D                                  insd
0044DF14                 _WinMain@16         endp ; sp-analysis failed
0044DF14
0044DF14                 ; -----------------------------------------------
0044DF15 2A                                  db  2Ah ; *
0044DF16 01                                  db    1
0044DF17 7D                                  db  7Dh ; }
0044DF18
0044DF18                 ; =============== S U B R O U T I N E =========
0044DF18
0044DF18
0044DF18                 sub_44DF18          proc near                  ; CODE
0044DF18 83 04 24 56                         add     dword ptr [esp+0], 56h
```

The polymorphic layer across the given samples follows a specific pattern which can used as a first anchor for further parsing.
The following code sequence is found across all given samples at the WinMain routine → 7? ?? 81 C4 ?? ?? ?? ?? 83 C4 ?? E8
To further reduce false positive hits, let‘s add another check to make it more reliable, by doing a calculation:
value in ((matching offset + 12) + value in (matching offset+12)+4)&ffffff00 == 0x83042400

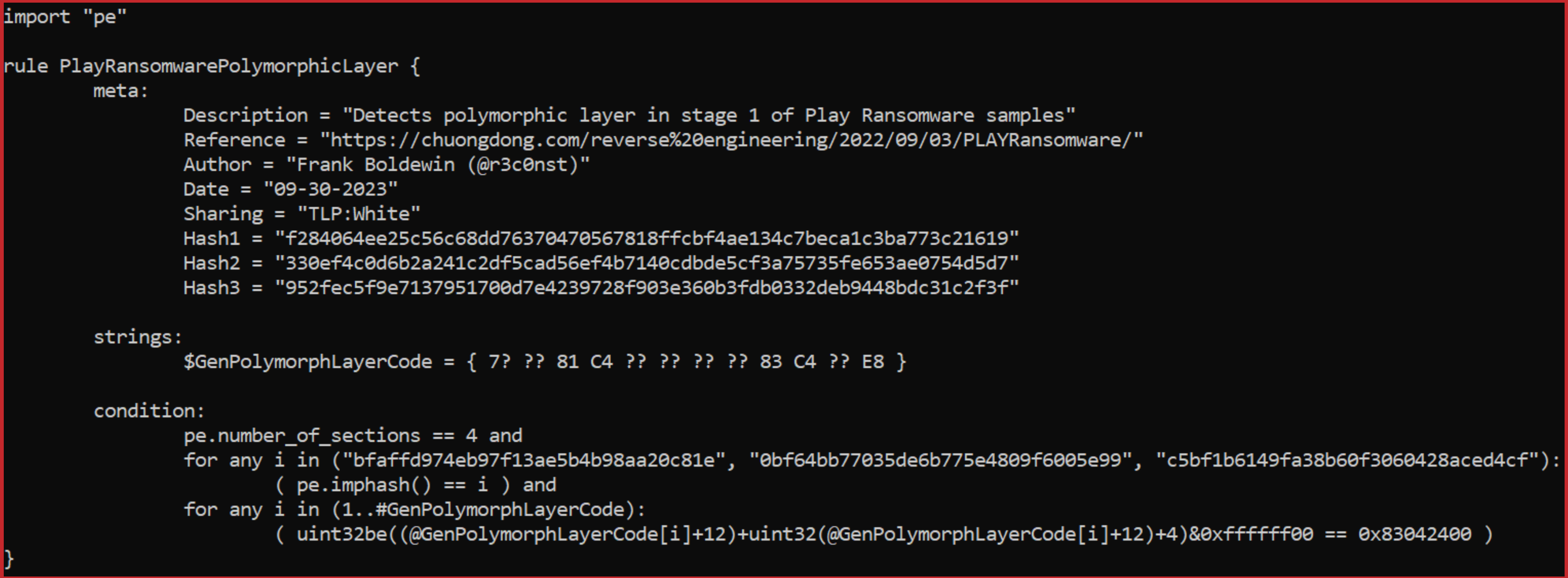
YARA - Challenge 5 (Play ransomware) - One more pattern for the rule

All given samples have 4 PE sections and 3 different import hashes



```
C:\YARA-Workshop\Challenges\05-PlayRansomwarePolymorphicLayer\Samples>..\..\..\Helpers\GetImpHash.py
Filename: .\006ae41910887f0811a3ba2868ef9576bbd265216554850112319af878f06e55 - ImpHash: bfaffd974eb97f13ae5b4b98aa20c81e - Sections: 4

Filename: .\0bb0f63a3bd6cc79efc18648b28912bd873a8e14c4918ee3579421f728a62fae - ImpHash: bfaffd974eb97f13ae5b4b98aa20c81e - Sections: 4

Filename: .\28e2f00a7c4b86fe98a184437cc1ea219b7853e4773b592ab83828c1cac77876 - ImpHash: c5bf1b6149fa38b60f3060428aced4cf - Sections: 4

Filename: .\330ef4c0d6b2a241c2df5cad56ef4b7140cdbde5cf3a75735fe653ae0754d5d7 - ImpHash: 0bf64bb77035de6b775e4809f6005e99 - Sections: 4

Filename: .\4571f77ee276c451a92b9433d04f8b4fcc8b85a6f5a28ad6b59c7bb7a3afc32b - ImpHash: 0bf64bb77035de6b775e4809f6005e99 - Sections: 4

Filename: .\731457e4704d299b353e802b72a6908dfa2124cbb5130b8cb9a943c6be6bcdc6 - ImpHash: c5bf1b6149fa38b60f3060428aced4cf - Sections: 4

Filename: .\952fec5f9e7137951700d7e4239728f903e360b3fdb0332deb9448bdc31c2f3f - ImpHash: 0bf64bb77035de6b775e4809f6005e99 - Sections: 4

Filename: .\9c7e55441fa5a460320dce5005358d820aec2386982fb3d77d52ce89b3d59744 - ImpHash: 0bf64bb77035de6b775e4809f6005e99 - Sections: 4

Filename: .\c87894f109c7b81f3842e1130bc96a72867d47e0a147be8d6a059963367b960d - ImpHash: c5bf1b6149fa38b60f3060428aced4cf - Sections: 4

Filename: .\f284064ee25c56c68dd76370470567818ffcbf4ae134c7beca1c3ba773c21619 - ImpHash: 0bf64bb77035de6b775e4809f6005e99 - Sections: 4
```

YARA - Challenge 5 (Play ransomware) - Solution



```
import "pe"

rule PlayRansomwarePolymorphicLayer {
        meta:
                Description = "Detects polymorphic layer in stage 1 of Play Ransomware samples"
                Reference = "https://chuongdong.com/reverse%20engineering/2022/09/03/PLAYRansomware/"
                Author = "Frank Boldewin (@r3c0nst)"
                Date = "09-30-2023"
                Sharing = "TLP:White"
                Hash1 = "f284064ee25c56c68dd76370470567818ffcbf4ae134c7beca1c3ba773c21619"
                Hash2 = "330ef4c0d6b2a241c2df5cad56ef4b7140cdbde5cf3a75735fe653ae0754d5d7"
                Hash3 = "952fec5f9e7137951700d7e4239728f903e360b3fdb0332deb9448bdc31c2f3f"

        strings:
                $GenPolymorphLayerCode = { 7? ?? 81 C4 ?? ?? ?? ?? 83 C4 ?? E8 }

        condition:
                pe.number_of_sections == 4 and
                for any i in ("bfaffd974eb97f13ae5b4b98aa20c81e", "0bf64bb77035de6b775e4809f6005e99", "c5bf1b6149fa38b60f3060428aced4cf"):
                        ( pe.imphash() == i ) and
                for any i in (1..#GenPolymorphLayerCode):
                        ( uint32be((@GenPolymorphLayerCode[i]+12)+uint32(@GenPolymorphLayerCode[i]+12)+4)&0xffffff00 == 0x83042400 )
}
```

Hunt, detect, disrupt!

May the defenders force be with you!