OST
Eastern Switzerland
University of Applied Sciences

Yara Rules

OST Cyber Defense

Ivan Bütler

YARA

YARA is a widely used and powerful tool for identifying and classifying files or data based on patterns, rules, or signatures. It is particularly valuable in cybersecurity and malware analysis. Here's a quick explanation:

1. **Pattern Matching**: YARA uses a combination of text and binary patterns to search for specific content within files, streams, or memory.
2. **Rules**: YARA rules are defined to specify patterns to search for, conditions to trigger, and optional metadata. These rules are written in a language specifically designed for YARA.
3. **Usage**: Security professionals and researchers use YARA to detect and classify malware, viruses, and other threats. It can also be used for general data classification and content identification tasks.
4. **Extensible**: YARA is highly customizable and extensible. Users can create their own rules or use existing ones from the YARA community.
5. **Command-Line Tool and Libraries**: YARA provides both a command-line tool for rule execution and libraries for integration into other software.

Overall, YARA is a versatile tool that aids in threat detection and content classification by allowing users to define rules to identify specific patterns or behaviors in data.

What is YARA?

- Yara is a search tool for **binary data**
- The Linux tool “grep” is very nice to search in ASCII files
- Of course, there are other binary search tools around, but none as powerful as YARA
  https://www.baeldung.com/linux/binary-files-pattern-search

Tool	Found All 6 Occurrences	Loaded Entire File to Memory	Elapsed Time
grep	No (0)	No	1m34.240s
bbe	Yes	No	0m59.665s
bgrep	Yes	No	0m58.054s
GHex	No (2)	Yes	-
Bless	Yes	No	4m0.000s

Basics

- Identifying malware: hash code
  - Problem: very restrictive
  - Changing one bit ➔ different hash
- Alternative:
  - Identify “marker” patterns
  - Standard: YARA rules
  - Documentation is here

YARA: Yet Another Recursive/Ridiculous Acronym

https://www.govcert.ch/blog/exchange-vulnerability-2021/

Swiss Government Computer Emergency Response

govcert.ch/blog/exchange-vulnerability-2021/

yara 1/2

The Federal Council › NCSC › GovCERT.ch

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Government Computer Emergency Response Team

Homepage | GovCERT.ch Blog | Whitepapers | Report an Incident | Statistics

GovCERT.Ch › Blog › Exchange Vulnerability 2021

Recent Blog Posts

- Exchange Vulnerability 2021
- Cyber Security for the Healthcare Sector During Covid19
- Security of the Swiss Domain Landscape (ccTLD ch)

Blog Archive by Year

- 2021
- 2020
- 2019
- 2018

Exchange Vulnerability 2021

Published on March 9, 2021 13:21 +0100 by GovCERT.ch (permalink)
Last updated on March 9, 2021 13:21 +0100

Introduction

In the past days, there was a lot of press coverage about several critical zero day vulnerabilities in Microsoft Exchange Server that are being tracked under the following CVEs:

- CVE-2021-26855
- CVE-2021-26857
- CVE-2021-26858
- CVE-2021-27065

Unfortunately, we recently became aware of several hundred organizations in Switzerland that got compromised by a threat actor that exploited the said vulnerability. While Microsoft attributed the initial, in-the-wild observed compromises to a Chinese state-sponsored group called HAFNIUM, several other threat actors quickly got hold of this exploit since the publication of patches by Microsoft. As a result, we have started informing possible compromised organizations based on information provided to

OST

https://www.govcert.ch/blog/exchange-vulnerability-2021/

Swiss Government Computer Emergency Response

govcert.ch/blog/exchange-vulnerability-2021/

Search for Forensic Artifacts

- Detect if webshells are present. There are a few places where no .aspx should exist or only very few provided by the Exchange Server itself:

```
\inetpub\wwwroot\aspnet_client\ (any .aspx file under this folder or sub folders)
\<exchange install path>\FrontEnd\HttpProxy\ecp\auth\ (TimeoutLogoff.aspx is legit)
\<exchange install path>\FrontEnd\HttpProxy\owa\auth\ (newer files (e.g. after 2nd of March 20
21 that do not belong to the installation))
\<exchange install path>\FrontEnd\HttpProxy\owa\auth\Current\ (should not contain .aspx files)
\<exchange install path>\FrontEnd\HttpProxy\owa\auth\<folder with version number>\ (should not
contain .aspx files)
```

Microsoft provided a list of currently known locations where webshell have been dropped.

You can also scan for the presence of webshells using YARA, e.g. by the free Loki Scanner provided by Nextron Systems or by using YARA rules provided by Volexity

https://github.com/Neo23x0/signature-base/blob/master/yara/apt_hafnium.yar

Swiss Government Computer Emergency Response

github.com/Neo23x0/signature-base/blob/master/yara/apt_hafnium.yar

Search or jump to... / Pull requests Issues Marketplace Explore

Neo23x0 / signature-base Public

Code | Issues 21 | Pull requests 3 | Actions | Projects | Wiki | Security | Insights

master signature-base / yara / apt_hafnium.yar

Neo23x0 docs: remove webshell tag

1 contributor

395 lines (365 sloc) | 16.2 KB

```
rule WEBSHELL_ASP_Embedded_Mar21_1 {
   meta:
      description = "Detects ASP webshells"
      author = "Florian Roth"
      reference = "Internal Research"
      date = "2021-03-05"
      score = 85
   strings:
      $s1 = "<script runat=\"server\">" nocase
      $s2 = "new System.IO.StreamWriter(Request.Form["
      $s3 = ".Write(Request.Form["
   condition:
      filesize < 100KB and all of them
}
```


OST

Yara Rules

Malware Analysis > Identifying Malware

By Example (I / III)

What might this do?

```
rule macrocheck : maldoc {
    meta: // this is a comment
        Author      = ”aFireeye Labs"
        Description = "Identify office documents with the MACROCHECK credential stealer in them..."
        Reference   = "https://www.fireeye.com/blog/threat-research/2014/11/fin4_stealing_insid.html"
    strings:
        $PARAMpword = "pword=" ascii wide
        $PARAMmsg = "msg=" ascii wide
        $PARAMuname = "uname=" ascii
        $userform = "UserForm" ascii wide
        $userloginform = "UserLoginForm" ascii wide
        $invalid = "Invalid username or password" ascii wide
        $up1 = "uploadPOST" ascii wide
        $up2 = "postUpload" ascii wide
    condition:
        all of ($PARAM*) or (($invalid or $userloginform or $userform) and ($up1 or $up2))
}
```

By Example (I / III)

```
rule macrocheck : maldoc {
    meta: // this is a comment
        Author      = "aFi
        Description = "Identify                    OCHECK credential stealer in them..."
        Reference   = "https://                    rch/2014/11/fin4_stealing_insid.html"
    strings:
        $PAR
        $PAR
        $PARAMuname = "uname=" ascii
        $userform = "UserForm" ascii wide
        $                    inForm" asc
                             e or password
                             i wide
        $up2 = "postUpload" ascii wide
    condition:
        all of ($PARAM*) or (($invalid or $userloginform or $userform) and ($up1 or $up2))
}
```

Tag for reporting

Rule name
- Case sensitive
- Alphanumeric & underscore (_) only

- Arbitrary key / value pairs
- Cannot be referenced in **strings** / **condition**

- Variable declaration
- Starts with a $ sign
- Alphanumeric & underscore (_) only

- nocase: match case insensitive
- wide: two bytes per character
- ascii: one byte per character (default)
- fullword: only match if delimited by non-alphanumeric character

Wildcard!

Boolean operator

Reference a string defined in **strings**

By Example (II / III)

What might this do?

```
rule foo {
    meta: …
    strings:
        $a1 = { 64 8B (05|0D|15|1D|25|2D|35|3D) 30 00 00 00 }
        $a2 = {64 A1 30 00 00 00}
        $a3 = {FF 75 ?? FF 55 ?? A?}
        $a4 =  {68 [-3] 07 00 [1-5] FF 15}
    condition:
        3 of them
        and for any i in (1 .. #a3): (uint8(@a3[i] + 2) == uint8(@a3[i] + 5))
        and !a4 > 10
}
```

By Example (II / III)

```
rule foo {
    meta: …
    strings:
        $a1 = { 64 8B (05|0D|15|1D|25|2D|35|3D) 30 00 00 00 }
        $a2 = {64 A1 3[illegible]00}
        $a3 = {FF 75 ?? FF 55 ?? A?}
        $a4 = {68 [-3] 07 00 [1-5] FF 15}
    condition:
        3 of them
        and for any i in (1 .. #a3): (uint8(@a3[i] + 2) == uint8(@a3[i] + 5))
        and !a4 > 10
}
```

Byte

Any byte

Alternative

Any nibble (half-byte)

0 to 3 bytes

1 to 5 bytes

List of all **strings**

unsigned **int**eger: **8** bits

Offset of ith occurrence of $a3

offset of 5 bytes

range

Number of times $a3 occurs

length

Default is Little Endian.
➔ e.g. uint8be for Big Endian

By Example (III / III)

What might this do?

Referenced rule **before** reference to it

```
private rule WindowsPE {
    strings:
        $mz = { 4D 5A }
    condition:
        $mz at 0 and filesize < 250KB and uint32(uint32(0x3C)) == 0x00004550
}


rule bar {
    meta: …
    strings:
        $re = /bn1=.{32}&sk1=[0-9a-zA-Z]{32}/
    condition:
        WindowsPE and $re
}
```

By Example (III / III)

```
private rule WindowsPE {
    strings:
        $mz = { 4D 5A }
    condition:
        $mz at 0 and filesize < 250KB and uint32(uint32(0x3C)) == 0x00004550
}

rule bar {
    meta: …
    strings:
        $re = /bn1=.{32}&sk1=[0-9a-zA-Z]{32}/
    condition:
        WindowsPE and $re
}
```

Not reported

KB: 2^10. MB: 2^20

Pointer to pointer

IMAGE_NT_SIGNATURE in DOS header

String at given location in file.
Alternative: `uint16(0) == 0x5A4D`

RegEx (own engine) – delimited by /

Reuse rule

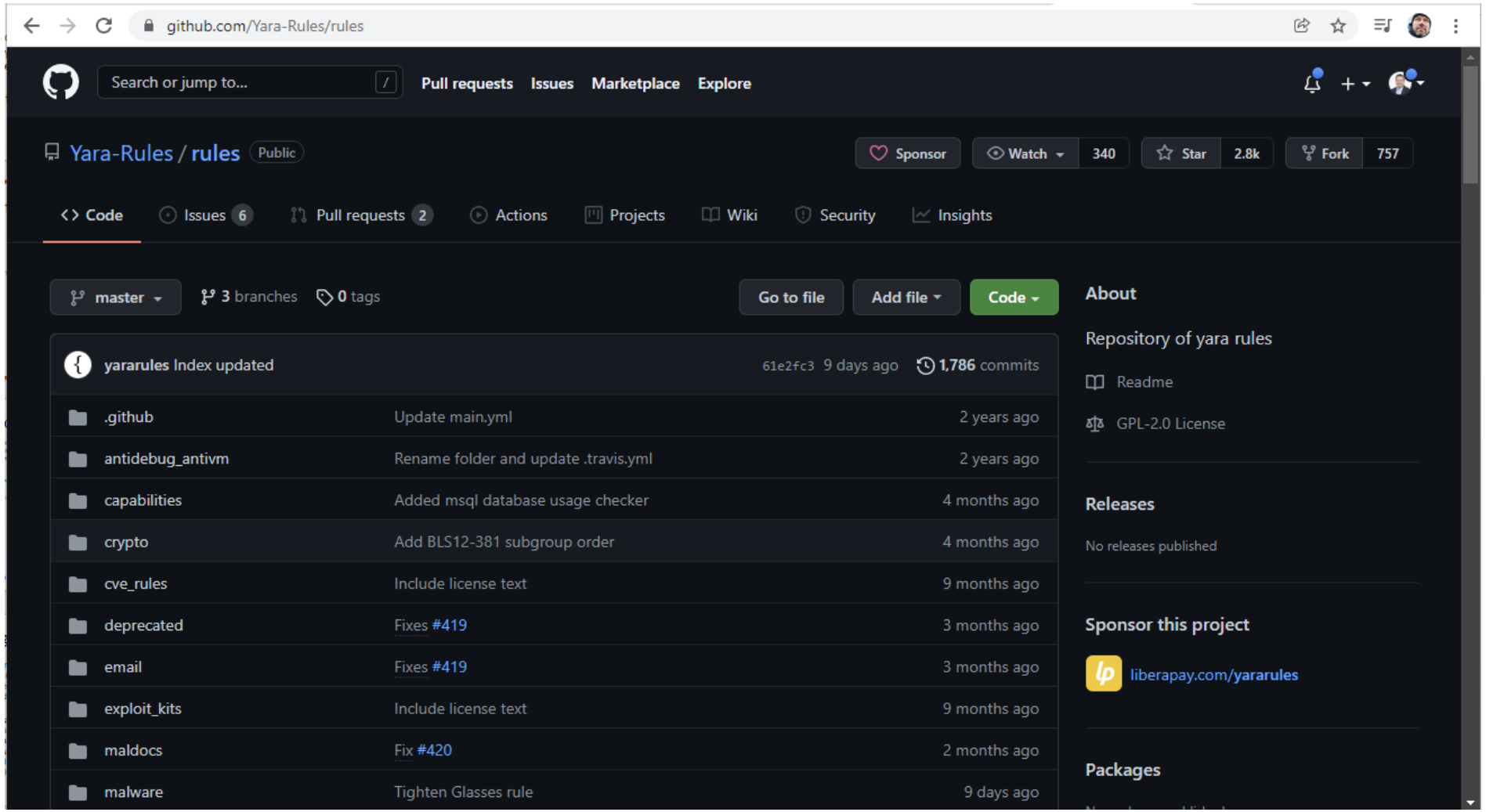
Referenced rule **before** reference to it

Also: **global rule**

- implicitly part of all other rules
- evaluated first
- **serves to extract common prerequisites**

Community Yara Rules

https://github.com/Yara-Rules/rules



https://yara.readthedocs.io/en/stable/writingrules.html#strings

Hex Strings

```
rule WildcardExample
{
    strings:
        $hex_string = { E2 34 ?? C8 A? FB }

    condition:
        $hex_string
}
```

```
rule NotExample
{
    strings:
        $hex_string = { F4 23 ~00 62 B4 }
        $hex_string2 = { F4 23 ~?0 62 B4 }
    condition:
        $hex_string and $hex_string2
}
```

```
rule JumpExample
{
    strings:
        $hex_string = { F4 23 [4-6] 62 B4 }

    condition:
        $hex_string
}
```

```
rule AlternativesExample2
{
    strings:
        $hex_string = { F4 23 ( 62 B4 | 56 | 45 ?? 67 ) 45 }

    condition:
        $hex_string
}
```

https://yara.readthedocs.io/en/stable/writingrules.html#strings

Text Strings

```
rule TextExample
{
    strings:
        $text_string = "foobar"

    condition:
        $text_string
}
```

`\"`	Double quote
`\\`	Backslash
`\r`	Carriage return
`\t`	Horizontal tab
`\n`	New line
`\xdd`	Any byte in hexadecimal notation

https://yara.readthedocs.io/en/stable/writingrules.html#strings

Case Insensitive

```
rule CaseInsensitiveTextExample
{
    strings:
        $text_string = "foobar" nocase

    condition:
        $text_string
}
```

Text strings in YARA are case-sensitive by default, however you can turn your string into case-insensitive mode by appending the modifier `nocase` at the end of the string definition, in the same line:

https://yara.readthedocs.io/en/stable/writingrules.html#strings

Wide Characters

- The wide modifier can be used to search for strings encoded with two bytes per character, something typical in many executable binaries.

```
rule WideCharTextExample1
{
    strings:
        $wide_string = "Borland" wide

    condition:
        $wide_string
}
```

- The wide modifier just interleaves the ASCII codes of the characters in the string with zeroes, it does not support truly UTF-16 strings containing non-English characters. If you want to search for strings in both ASCII and wide form, you can use the ascii modifier in conjunction with wide , no matter the order in which they appear.

```
rule WideCharTextExample2
{
    strings:
        $wide_and_ascii_string = "Borland" wide ascii

    condition:
        $wide_and_ascii_string
}
```

https://yara.readthedocs.io/en/stable/writingrules.html#strings

XOR String

- The following rule will search for every single byte XOR applied to the string "This program cannot" (including the plaintext string):

```
rule XorExample1
{
    strings:
        $xor_string = "This program cannot" xor

    condition:
        $xor_string
}
```

https://yara.readthedocs.io/en/stable/writingrules.html#strings

base64 Strings

- The following rule will search for the three base64 permutations of the string "This program cannot“

```
rule Base64Example1
{
    strings:
        $a = "This program cannot" base64

    condition:
        $a
}
```

- This will cause YARA to search for these three permutations:
  - VGhpcyBwcm9ncmFtIGNhbm5vd
  - RoaXMgcHJvZ3JhbSBjYW5ub3
  - UaGlzIHByb2dyYW0gY2Fubm90

https://yara.readthedocs.io/en/stable/writingrules.html#strings

Private Strings

- All strings in YARA can be marked as private which means they will never be included in the output of YARA. They are treated as normal strings everywhere else, so you can still use them as you wish in the condition, but they will never be shown with the -s flag or seen in the YARA callback if you're using the C API.

```
rule PrivateStringExample
{
    strings:
        $text_string = "foobar" private

    condition:
        $text_string
}
```

https://yara.readthedocs.io/en/stable/writingrules.html#strings

String Modifiers

Keyword	String Types	Summary	Restrictions
`nocase`	Text, Regex	Ignore case	Cannot use with `xor`, `base64`, or `base64wide`
`wide`	Text, Regex	Emulate UTF16 by interleaving null (0x00) characters	None
`ascii`	Text, Regex	Also match ASCII characters, only required if `wide` is used	None
`xor`	Text	XOR text string with single byte keys	Cannot use with `nocase`, `base64`, or `base64wide`
`base64`	Text	Convert to 3 base64 encoded strings	Cannot use with `nocase`, `xor`, or `fullword`
`base64wide`	Text	Convert to 3 base64 encoded strings, then interleaving null characters like `wide`	Cannot use with `nocase`, `xor`, or `fullword`
`fullword`	Text, Regex	Match is not preceded or followed by an alphanumeric character	Cannot use with `base64` or `base64wide`
`private`	Hex, Text, Regex	Match never included in output	None

https://yara.readthedocs.io/en/stable/writingrules.html#strings

Private Strings

- Regular expressions are one of the most powerful features of YARA. They are defined in the same way as text strings but enclosed in forward slashes instead of double-quotes, like in the Perl programming language.

```
rule RegExpExample1
{
    strings:
        $re1 = /md5: [0-9a-fA-F]{32}/
        $re2 = /state: (on|off)/

    condition:
        $re1 and $re2
}
```

Yara Binary and Yara Rules on HL LiveCD

- Please run `apt-get install hl-volatility-kali`
- `Online: https://github.com/VirusTotal/yara`

Jun 12

plusvic

v4.3.2

d1ff3ec

Compare

YARA v4.3.2 Latest

- BUGFIX: assertion triggered with certain hex patterns when scanning arbitrary files (bcc6312). Reported by Huawei Central Software Institute Security Team.

Assets 4

yara-4.3.2-2150-win32.zip	1.47 MB	Jun 12
yara-4.3.2-2150-win64.zip	2.12 MB	Jun 12
Source code (zip)		Jun 12
Source code (tar.gz)		Jun 12

9 1 10 people reacted

Yara Rule Generator

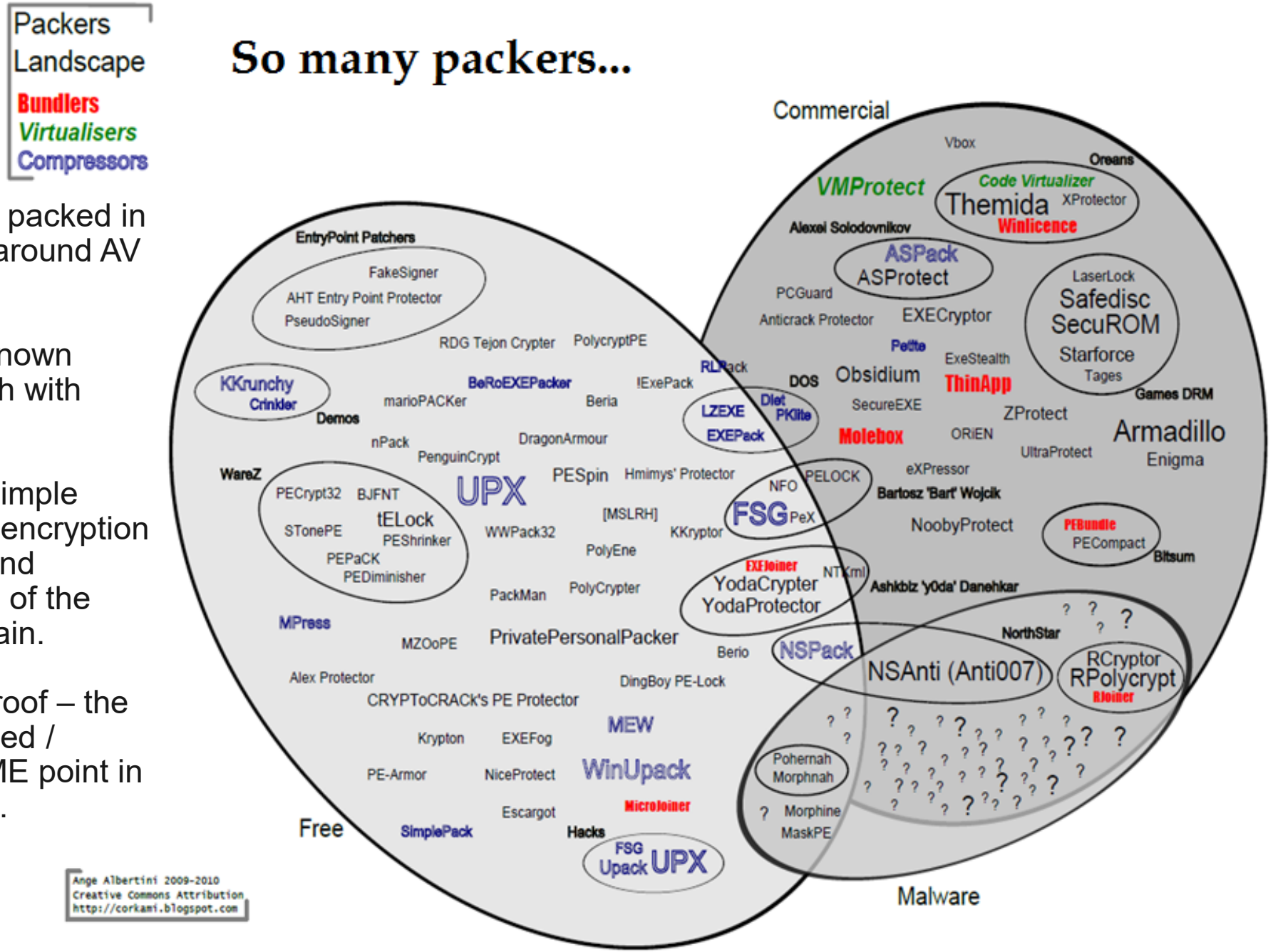
- https://github.com/InQuest/awesome-yara
- https://github.com/Neo23x0/yarGen

```
git clone https://github.com/Neo23x0/yarGen.git
pip install -r requirements.txt
./yarGen.py --update
./yarGen.py -m /tmp/malware
yara -w -r ./yargen_rules.yar /tmp/malware
```

```
/home/hacker/yarGen 〉 master !2 ?4 〉 yara -w -r ./yargen_rules.yar /usr/sbin
_tmp_malware_passwd /usr/sbin/groupmod
_tmp_malware_passwd /usr/sbin/chgpasswd
_tmp_malware_passwd /usr/sbin/usermod
_tmp_malware_passwd /usr/sbin/groupadd
_tmp_malware_passwd /usr/sbin/grpunconv
_tmp_malware_passwd /usr/sbin/pwconv
_tmp_malware_passwd /usr/sbin/vipw
_tmp_malware_passwd /usr/sbin/groupmems
_tmp_malware_passwd /usr/sbin/cppw
_tmp_malware_passwd /usr/sbin/cpgr
_tmp_malware_passwd /usr/sbin/pwunconv
_tmp_malware_passwd /usr/sbin/vigr
_tmp_malware_passwd /usr/sbin/pwck
_tmp_malware_passwd /usr/sbin/grpck
_tmp_malware_passwd /usr/sbin/grpconv
_tmp_malware_passwd /usr/sbin/groupdel
_tmp_malware_passwd /usr/sbin/useradd
_tmp_malware_passwd /usr/sbin/chpasswd
_tmp_malware_passwd /usr/sbin/userdel
_tmp_malware_passwd /usr/sbin/newusers
```

Packers

- Most malware today is packed in some way to help get around AV signature detection
- There are over 8000 known packers out there, each with their own signatures.
- They can range from simple compression to full on encryption / debugger detection and generally make the life of the Malware Reverser a pain.
- Packers are not fool proof – the exe HAS to be decrypted / decompressed at SOME point in order to run on the OS.



YARA in Velociraptor

Velociraptor Response and Moni

Not secure | https://127.0.0.1:8889/app/index.html?#/artifacts/Windows.Detection.ProcessMemory

Search clients | Show All | admin

yara

Windows.Detection.ProcessMemory

Type: client

Scanning process memory for signals is powerful technique. This artifact scans processes for a yara signature and when detected, the process memory is dumped and uploaded to the server.

Parameters

Name	Type	Default	Description
processRegex		notepad	
yaraRule		wide nocase ascii: this is a secret	

Source

- Generic.Applications.Office.Keywords
- Windows.Detection.ProcessMemory
- Windows.Detection.ProcessMemory.CobaltStrike
- Windows.Detection.RemoteYara.Process
- Windows.Detection.Yara.NTFS
- Windows.Forensics.FilenameSearch
- Windows.Forensics.SolarwindsSunburst
- Windows.Persistence.PowershellRegistry
- Windows.Search.Yara

OST

Yara Rules

How to use them

OST

File System Scan

- yara -r /opt/applic/yara-rules/index.yar /home/hacker/malware-samples

recursively

yara index

search dir

- yara -w -r -t Packer -t Compiler /opt/applic/yara-rules/index.yar /home/hacker/malware-samples

suppress warnings

search for tags (Packer)

search for tags (Compiler)

Forensic Image Scan

Volatility

+ yara

plugin

Volatility & YARA Examples

- Volatility2
  - volatility2 -f ./0zapftis.vmem yarascan -y /opt/applic/yara-rules/malware/MALW_LuaBot.yar
  - volatility2 -f ./0zapftis.vmem yarascan --yara-file /opt/applic/yara-rules/malware/MALW_LuaBot.yar
  - volatility2 -f ./0zapftis.vmem yarascan --yara-rule "http:"

- Volatility3
  - volatility3 -f ./0zapftis.vmem yarascan.YaraScan --yara-rules "http:"
  - volatility3 -f /home/hacker/Downloads/0zapftis.vmem **windows.vadyarascan.VadYaraScan** --yara-file ./packers_index.yar
  - volatility3 -f /home/hacker/Downloads/0zapftis.vmem **yarascan.YaraScan** --yara-file ./packers_index.yar

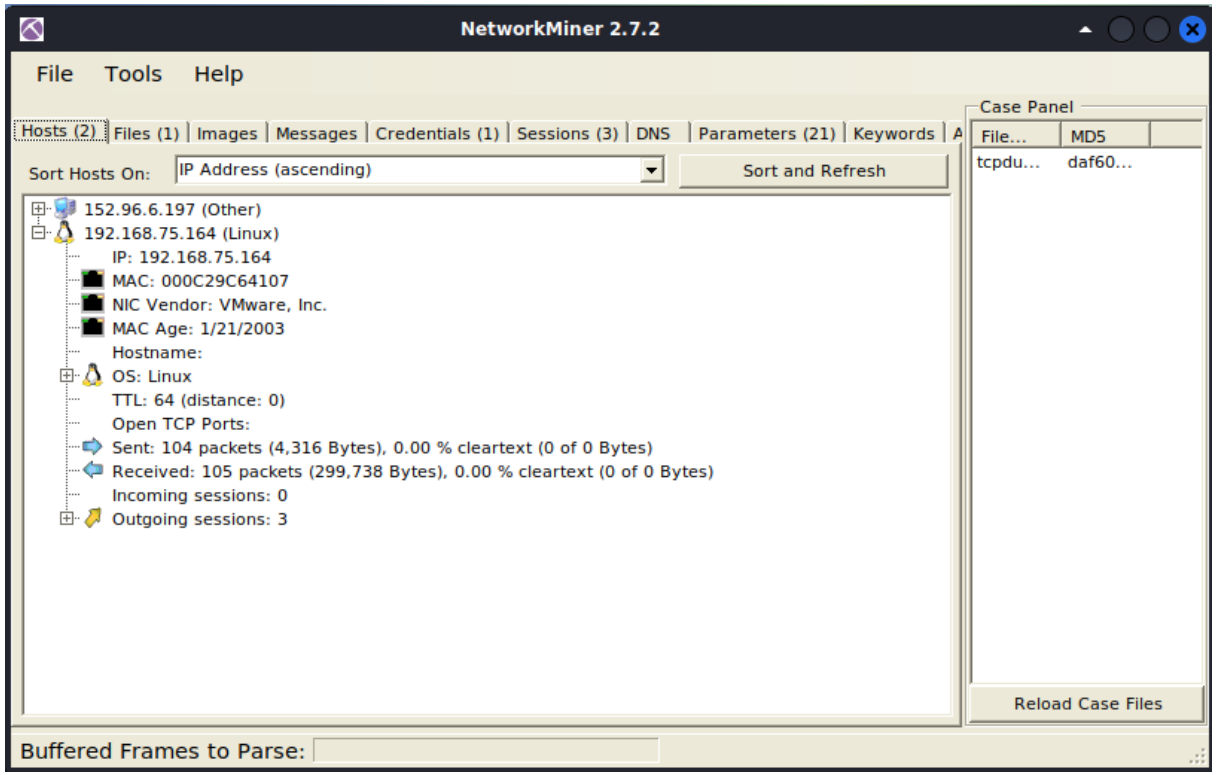
Example

- cd /opt/applic/yara-rules
- volatility3 –f <file> --yara-file <rule-file>

```
xfce4-terminal - volatility3 -f /home/hacker/Downloads/0zapftis.vmem yarascan.YaraScan
File  Edit  View  Terminal  Tabs  Help
root@hlkali:/home/hacker/Downloads        volatility3 -f /home/hacker/Downloads/0zapftis.vmem yarascan.YaraScan
/home/hacker/Downloads  cd /opt/applic/yara-rules
/opt/applic/yara-rules > master !13  volatility3 -f /home/hacker/Downloads/0zapftis.vmem yarascan.YaraScan --yara-file ./packers_index.yar
Volatility 3 Framework 2.0.0
Progress:  100.00               PDB scanning finished
Offset  Rule    Component       Value

0x7c9000c0      HasRichSignature        $a0     52 69 63 68
```

Network Dump



yara cannot be used to search in pcap files directly

File Extraction from Network Dump (PCAP)

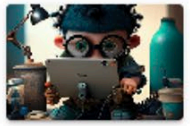
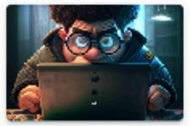
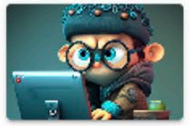
- foremost -t all -f tcpdump.pcap
- binwalk -e tcpdump.pcap
- tcpflow -r tcpdump.pcap -o .
- chaosreader tcpdump.pcap
- Wireshark – Follow TcpStream -> SaveAs …

yara cannot be used to search in pcap files directly

Hacking-Lab Exercises

2021-11-29 YARA and Volatility

Show Solved

#		Name	Categories	Level	Mode	Grading	Points
1		Kookarai: yara and yara-rule installation f381471e-4c9b-47f2-8522-6cbaf1a784bb		novice			0% 0/50
2		Kookarai: yara and yarGen 2147e060-bdf3-4482-91ec-3159b344b3ce		novice			0% 0/50
3		Kookarai: tcpdump and yara effc6dd4-64c0-4c34-a58f-8c4fe3492879		novice			0% 0/50
4		Kookarai: yara and CyberChef 4b9a41f4-99bf-479a-8108-e46728c84c08		novice			0% 0/50