**Yara基础入门**

**参考：**

https://zhuanlan.zhihu.com/p/361359544?utm\_source=qq&utm\_medium=social&utm\_oi=831418544871854080

https://bbs.pediy.com/thread-226011.htm

**一、yara规则字段解释:**

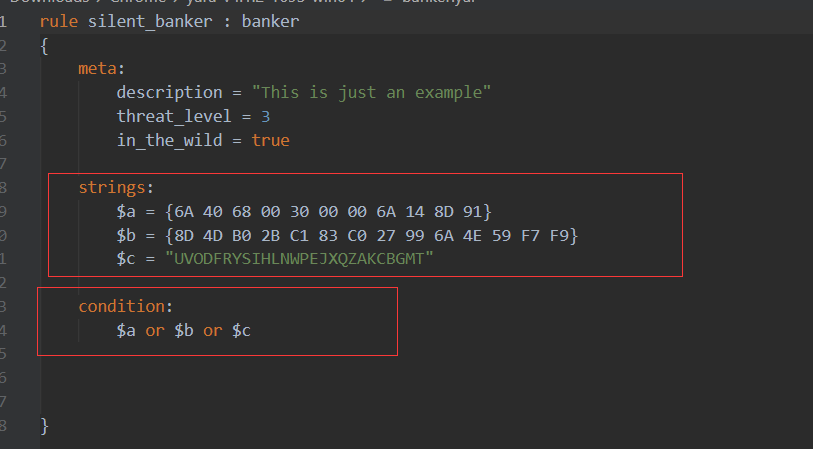
meta：后面的是一些描述信息，比如可以有规则说明、作者信息、威胁等级、在野情况、文件MD5、来源等内容；

strings：后面是检测规则，有字符串、字节序列等内容；

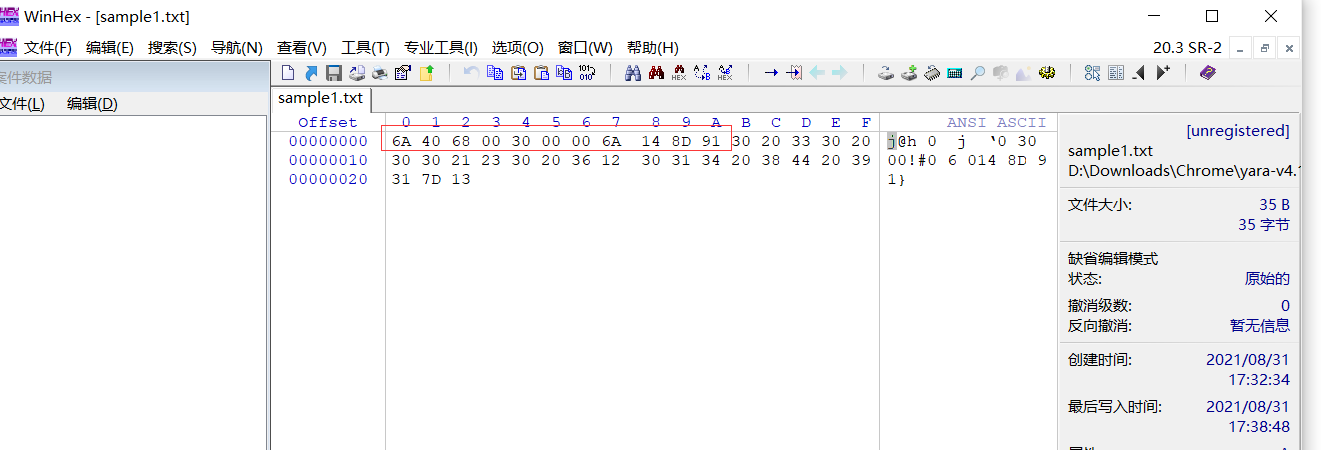
condition：为判断条件

**二、例子1：**

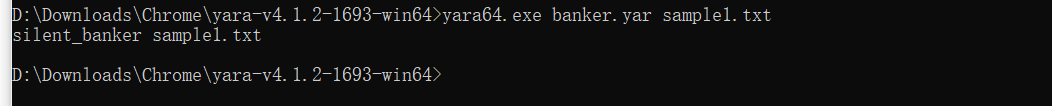
**yara规则：检测任意字符串或字节码**



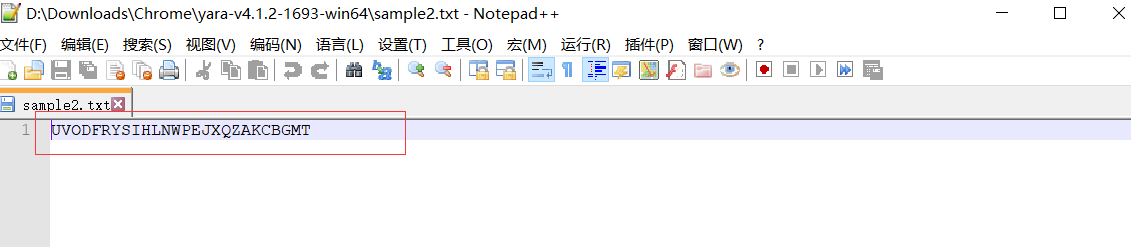
**样本1：**



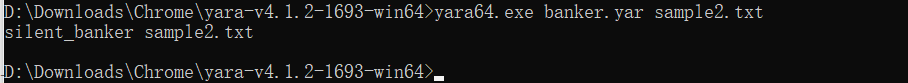
**告警：**



**样本2：**



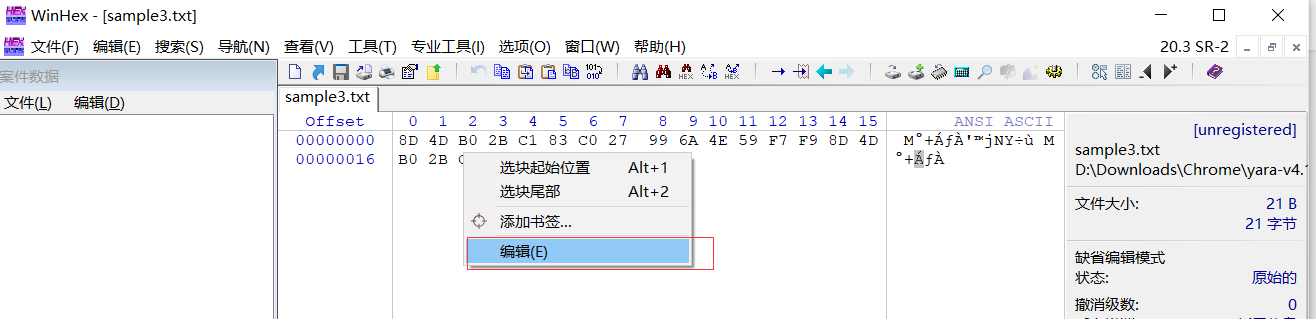
**告警：**



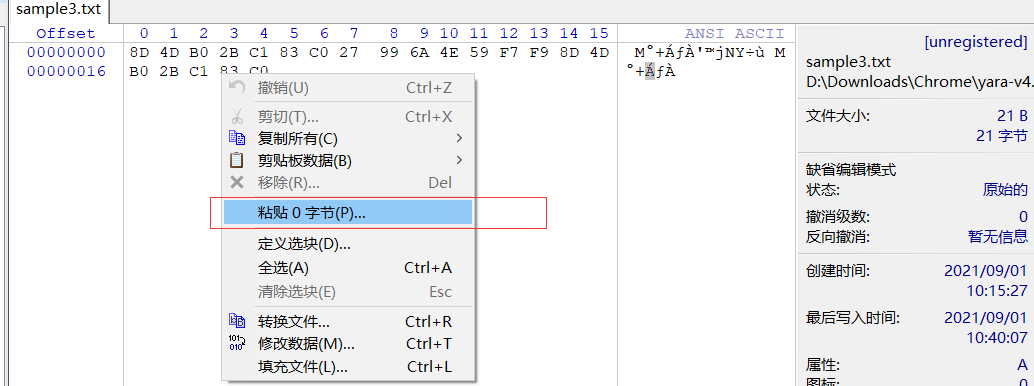
**样本3：**

**0x00批量添加16进制**

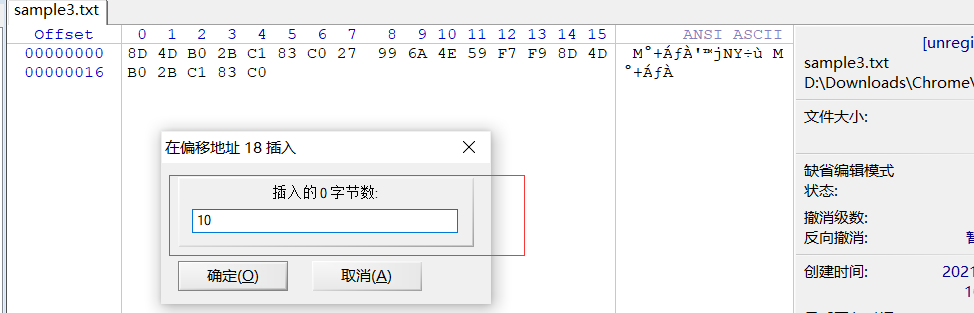
**0x01右键——编辑**



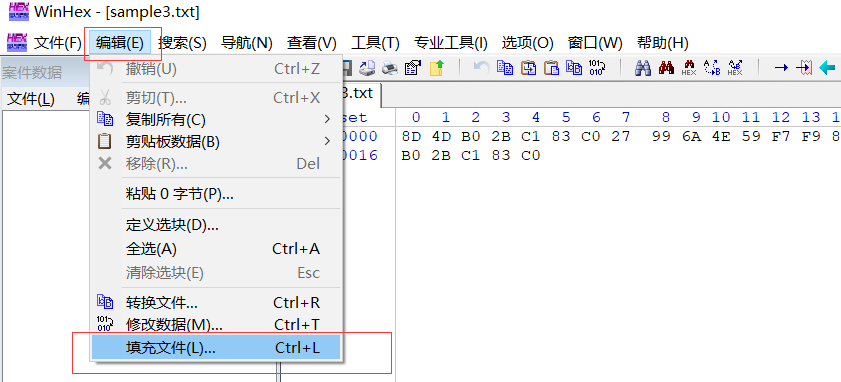
**0x02粘贴0字节**



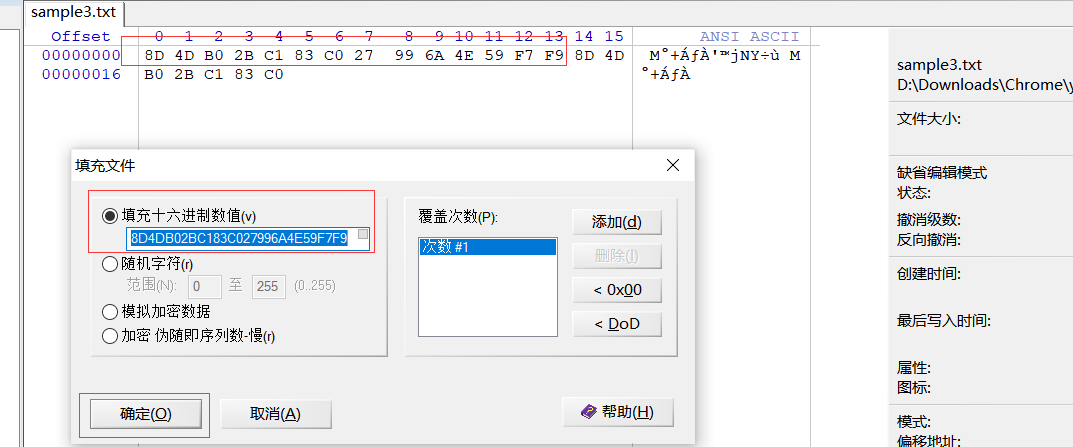
**0x03填入00的字节数**



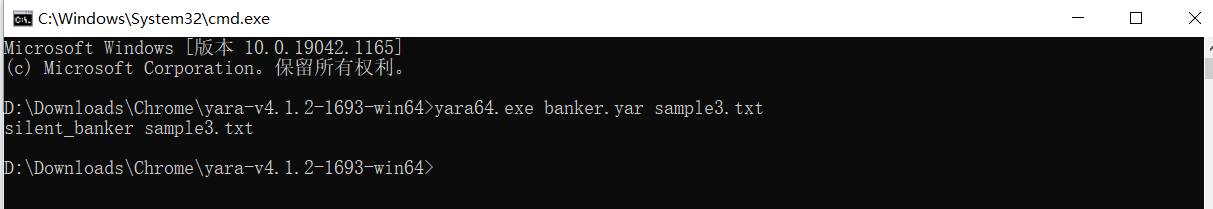
**0x04编辑——填充文件**



**0x05输入十六进制数，点击确定，成功添加：**



**告警：**



**规则解释：**

在本例检测规则处定义的字节序列和字符串，判断条件是or ，所以只要匹配到

$a $b $c三个字符串或字节序列中的任意一个，那么样本就会被识别成silent\_banker

**三、例子2：**

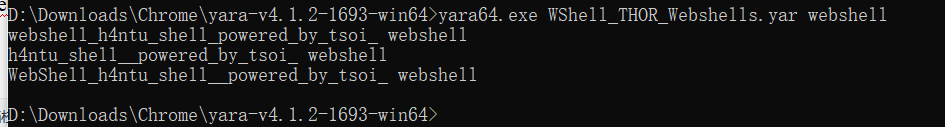
**yara规则：检测webshell**



**样本hash:**

06ed0b2398f8096f1bebf092d0526137

**告警：**



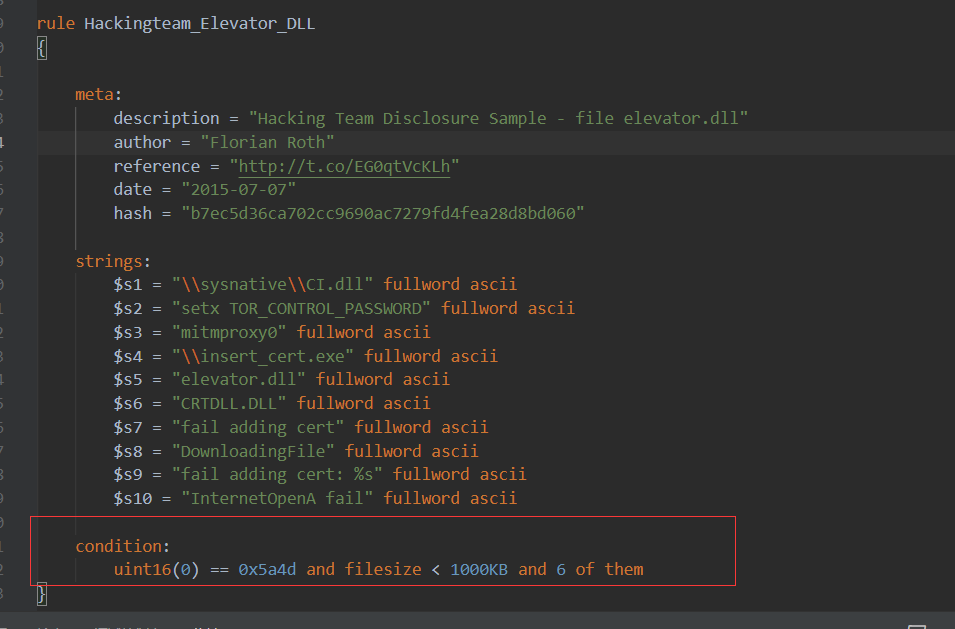
**规则解释：**

当样本都存在以上四个字符串，则表示是

webshell\_h4ntu\_shell\_powered\_by\_tsoi\_ : webshell样本

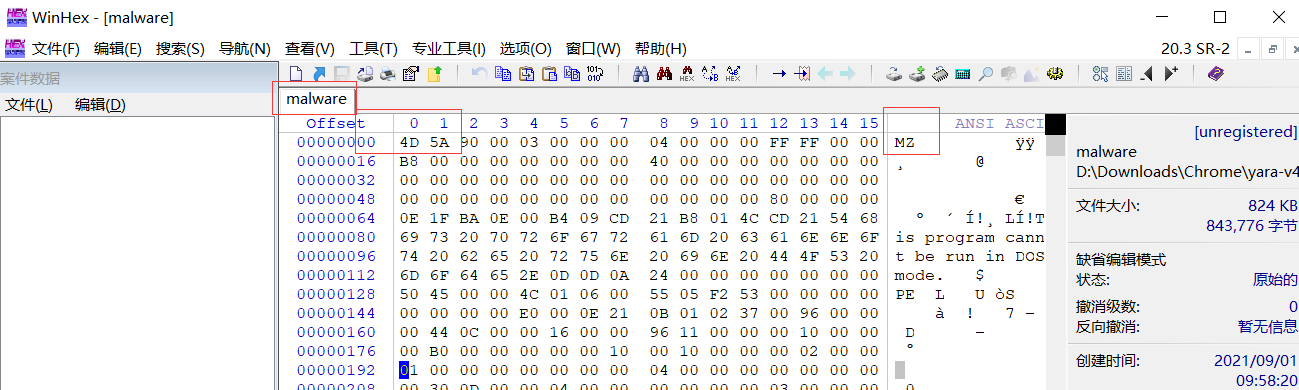
**四、例子3：**

**yara规则：检测malware**

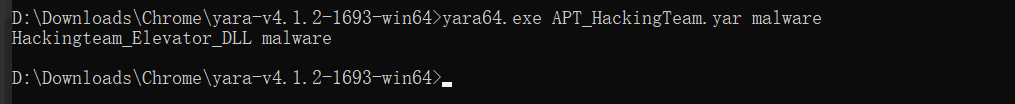


**样本hash：**

b7ec5d36ca702cc9690ac7279fd4fea28d8bd060



**告警：**



**规则解释：**

1、当样本文件大小小于1000KB

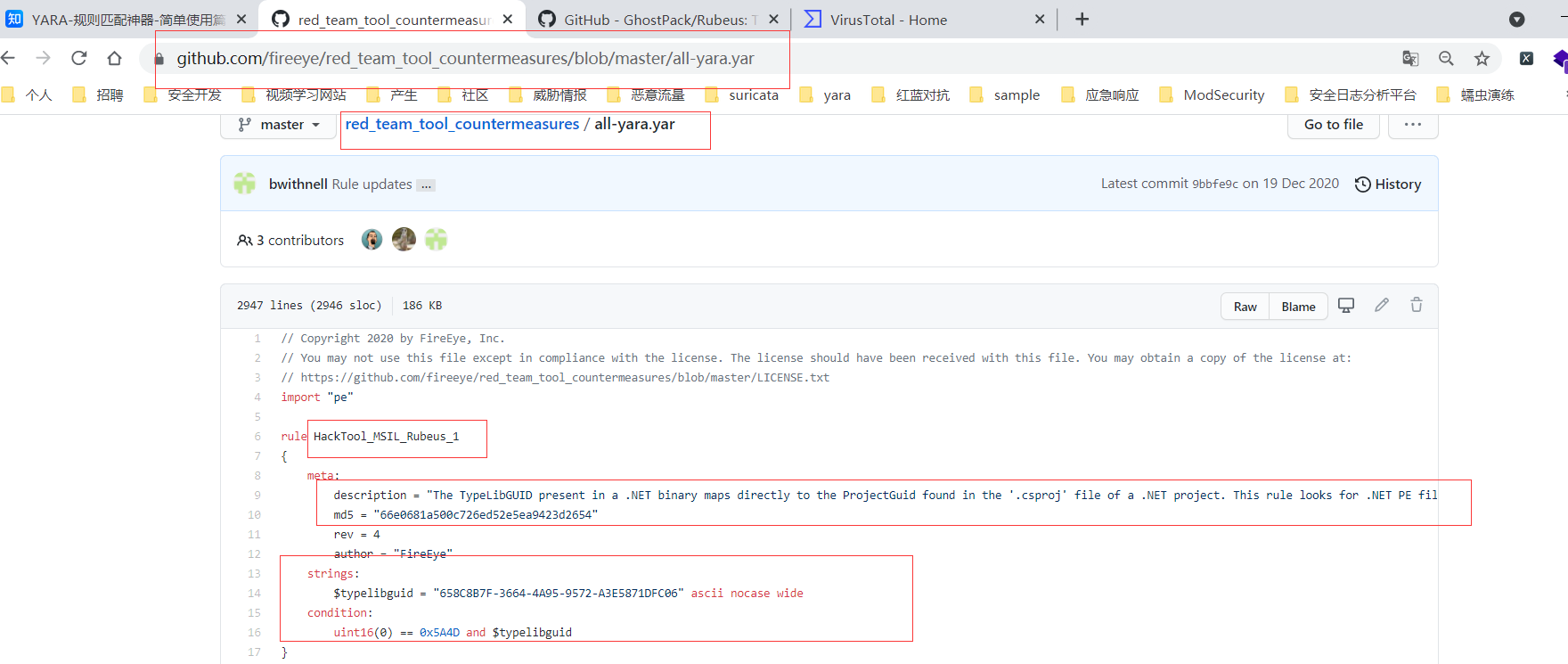
2、当样本文件文件头为0x5a4d，即mz 头（window的PE文件标志）

3、当样本都涵盖了以上至少6个字符串

则表示是Hackingteam\_Elevator\_DLL 样本

**五、实战分析：**

**yara规则：检测黑客工具**



**样本hash:**

66e0681a500c726ed52e5ea9423d2654

**规则解释：**

Nocase：忽略大小写

Ascii：ascii匹配中了都可以

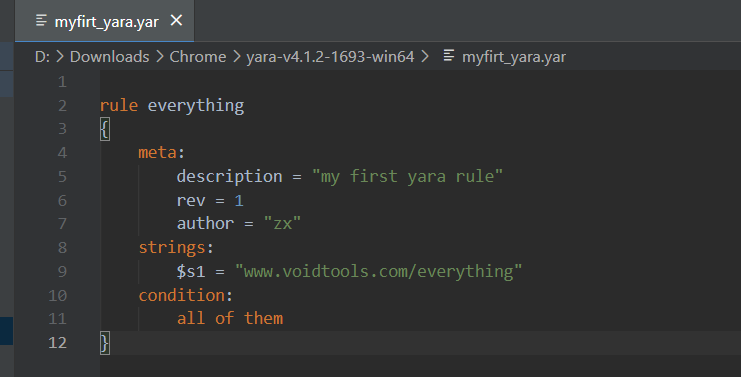
Wide：宽字节字符串匹配中了都可以

同样是检测是pe文件，并且当658C8B7F-3664-4A95-9572-A3E5871DFC06 字符串存在样本里

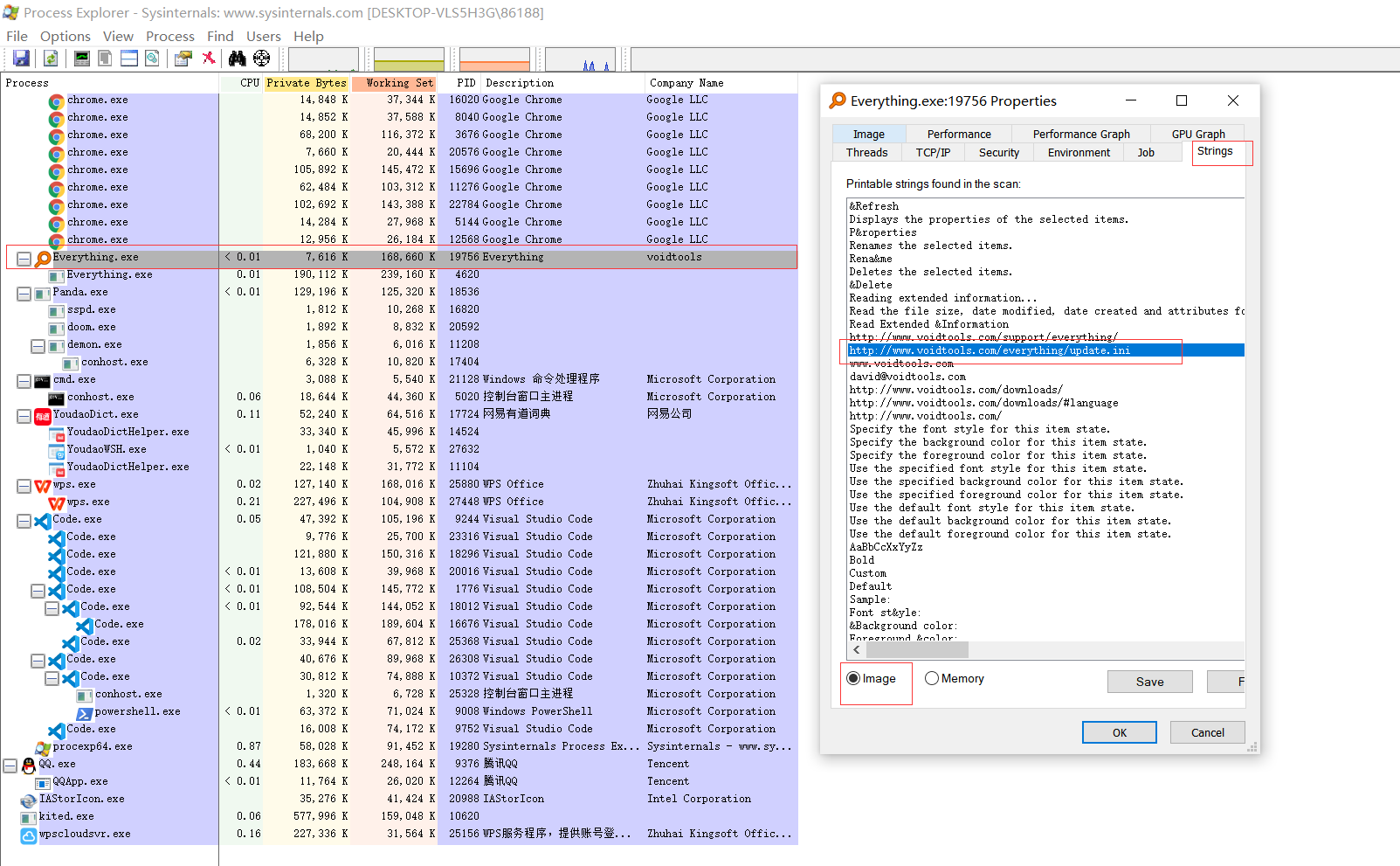
则说明是HackTool\_MSIL\_Rubeus\_1 样本

**六、扫描进程：yara rules\rules.yar pid**

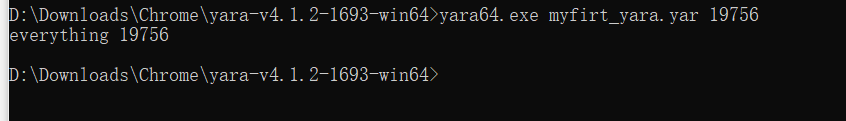
**规则：检测进程**



**进程PID:19756**

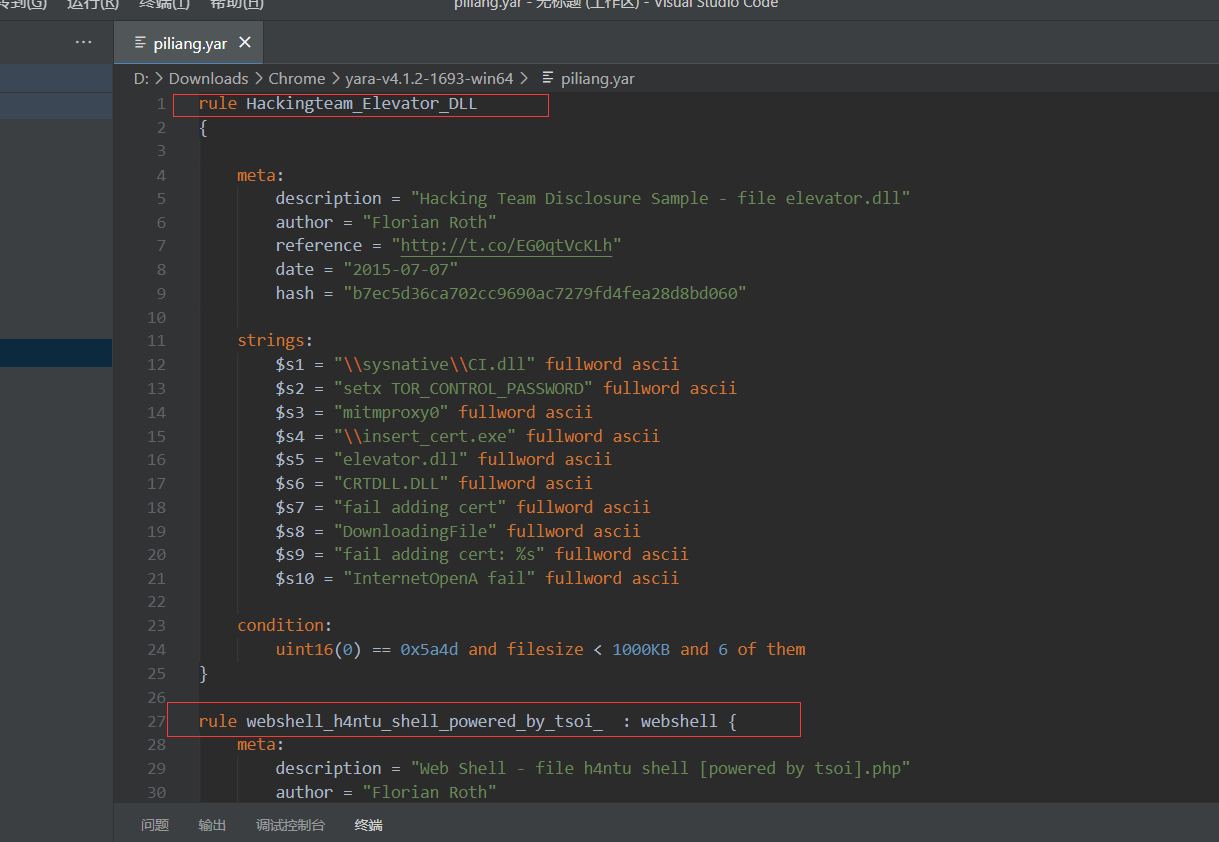


**告警：进程告警**



**七、扫描文件夹：yara rules\rules.yar folder**

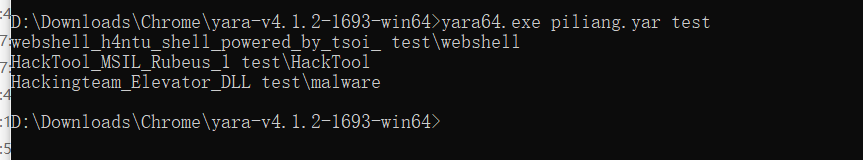
**规则：批量规则**



**文件夹test：批量样本**



**告警：批量告警**



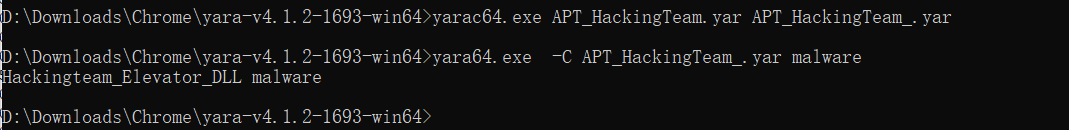
**八、编译yara规则**

**编译：**

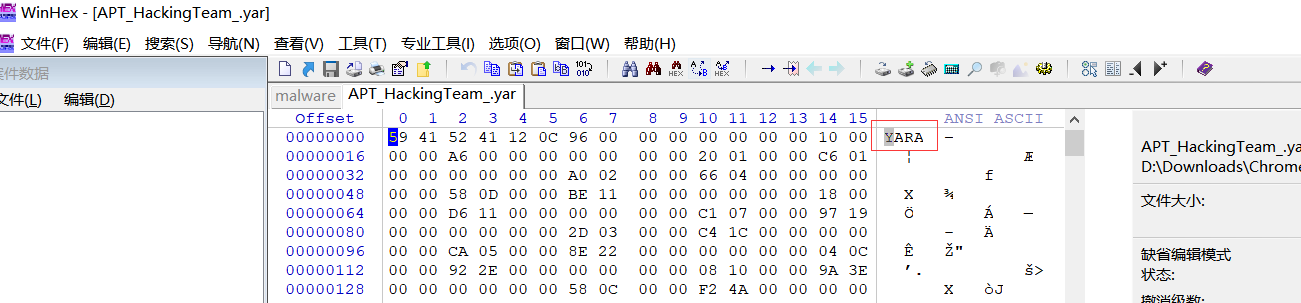
yarac64.exe APT\_HackingTeam.yar APT\_HackingTeam\_.yar

**规则匹配告警：**

yara64.exe -C APT\_HackingTeam\_.yar malware



**可见编译后的字节码规则检测出了样本，字节码文件的文件头有YARA 标识**



**九、总结：**

如果在文件或进程内存中找到字符串，则该变量评估为真，否则为假。不能匹配白文件或其他类型木马，只能匹配同一类木马家族，规则要通用性，不能只单独匹配到一个文件，正则最好不能出现"扫描速度慢"，写的时候可以写多些字符串（除非敢打包票一定对，可以写少点）或者字符串跟16进制结合，也可以思考攻击队使用的一些手法来拓展检测思路

**结束！**

# 基础语法知识

# 一、？通配符：

代表任意半字节

### 规则：

rule WildcardExample

{

    strings:

        $hex\_string = { E2 34 ?? C8 A? FB }

    condition:

        $hex\_string

}

### 匹配：？代表任意

E2 34 ?? C8 A? FB

# 二、[4-6]:跳转：不定长通配符

任何 4 到 6 个字节的任意序列都可以占据跳转的位置，任何跳转 [XY] 都必须满足条件 0 <= X <= Y

### 规则：

rule JumpExample

{

    strings:

        $hex\_string = { F4 23 [4-6] 62 B4 }

    condition:

        $hex\_string

}

### 匹配：

F4 23 01 02 03 04 62 B4

F4 23 00 00 00 00 00 62 B4

F4 23 15 82 A3 04 45 22 62 B4

### 更多用法例子：

FE 39 45 [0-8] 89 00

FE 39 45 [23-45] 89 00

FE 39 45 [1000-2000] 89 00

### 如果下限和上限相等，您可以写一个括在括号中的数字，如下所示：

FE 39 45 [6] 89 00

### 上面的字符串等效于以下两个：

FE 39 45 [6-6] 89 00

FE 39 45 ?? ?? ?? ?? ?? ?? 89 00

### 从 YARA 2.0 开始，您还可以使用无界跳转：

FE 39 45 [10-] 89 00

FE 39 45 [-] 89 00

# 三、| ：为十六进制字符串的给定片段提供不同的替代方案

### 规则：

rule AlternativesExample2

{

    strings:

        $hex\_string = { F4 23 ( 62 B4 | 56 | 45 ?? 67 ) 45 }

    condition:

        $hex\_string

}

### 匹配：

F4 23 62 B4 45

F4 23 56 45

F4 23 45 ?? 67 45

# 四、nocase：不区分大小写

### 规则：

rule CaseInsensitiveTextExample

{

    strings:

        $text\_string = "foobar" nocase

    condition:

        $text\_string

}

# 五、wide:宽字符

wide修饰符可以用来搜寻以2字节表示1字符这种方式编码的字符串, 这种宽字符串在许多二进制文件中都有出现. 如果字符串"Borland"以2字节表示1字符的方式编码并在二进制文件中出现, 我们就可以使用wide修饰符将其捕获. 因为"Borland"也可能是"borland", 我们也可以添加nocase修饰符以免错过.

[!]重要提示: 请记住, 该修饰符只是将字符串中字符的ASCII码和\x00交错起来组成宽字符, 它并不支持包含非英文字符的UTF-16字符串. 要想对既有ASCII字符和宽字符的字符串进行搜索, 请使用如下命令:

wide ascii nocase

### 规则：

rule WideCharTextExample1

{

    strings:

        $wide\_string = "Borland" wide

    condition:

        $wide\_string

}

### 样本：

### 匹配：

42 00 6F 00 72 00 6C 00 61 00 6E 00 64 00

# 六、Fullwords修饰符:

该修饰符可用于匹配那些前后没有附加其他字符的单词(全词匹配).

### 规则:

rule ShadyDomain

{

    strings:

        $shady\_domain = "faceebook" fullword

    condition:

        $shady\_domain

}

### 匹配：

www.faceebook.com

www.myportal.faceebook.com

https://secure.faceebook.com

### 但这个规则不能匹配以下的字符串:

www.myfaceebook.com

thefaceebook.com

两者区别在于匹配的全词前后可以附加特殊字符, 不能是普通字符.

# 七、xor修饰符：

每个单字节默认分别和0000 0001或0000 0010 进行异或运算（应用于加密运算，再进行异或就是解密），想更好地控制与xor修饰符一起使用的字节范围，请使用：xor(0x01-0xff)

### **位运算：**

同或：同真，不同假  
异或：同假，不同真

### 规则:

rule XorExample1

{

    strings:

        $xor\_string = "This program cannot" xor

    condition:

        $xor\_string

}

### 上述规则在逻辑上等价于：

rule XorExample2

{

    strings:

        $xor\_string\_00 = "This program cannot"

        $xor\_string\_01 = "Uihr!qsnfs`l!b`oonu"

        $xor\_string\_02 = "Vjkq\"rpmepco\"acllmv"

        // Repeat for every single byte XOR

    condition:

        any of them

}

### 解释：

字符串：This

十六进制：54 68 69 73

二进制：0101 0100 0110 1000 0110 1001 0111 0011

与0000 0001进行异或运算：

二进制：0101 0101 0110 1001 0110 1000 0111 0010

十六进制：55 69 68 72

字符串：Uihr

与0000 0010进行异或运算：

二进制：0101 0110 0110 1010 0110 1011 0111 0001

十六进制：56 6A 6B 71

字符串：Vjkq

### 更多用法：

以下两条规则在逻辑上是等价的：

rule XorExample3

{

    strings:

        $xor\_string = "This program cannot" xor wide

    condition:

        $xor\_string

}

rule XorExample4

{

    strings:

        $xor\_string\_00 = "T\x00h\x00i\x00s\x00 \x00p\x00r\x00o\x00g\x00r\x00a\x00m\x00 \x00c\x00a\x00n\x00n\x00o\x00t\x00"

        $xor\_string\_01 = "U\x01i\x01h\x01r\x01!\x01q\x01s\x01n\x01f\x01s\x01`\x01l\x01!\x01b\x01`\x01o\x01o\x01n\x01u\x01"

        $xor\_string\_02 = "V\x02j\x02k\x02q\x02\"\x02r\x02p\x02m\x02e\x02p\x02c\x02o\x02\"\x02a\x02c\x02l\x02l\x02m\x02v\x02"

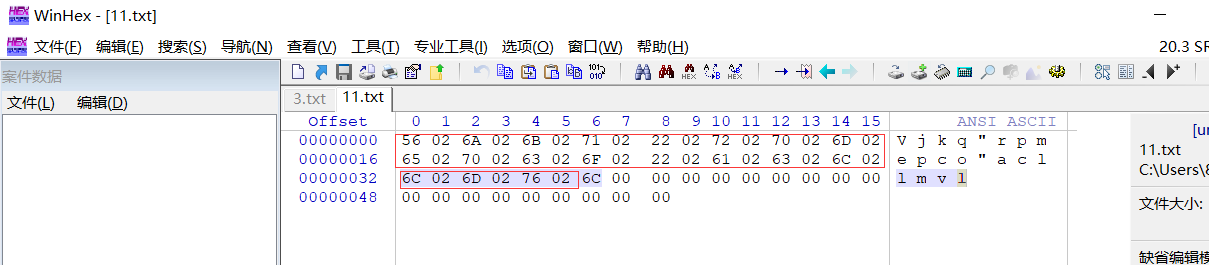
        // Repeat for every single byte XOR operation.

    condition:

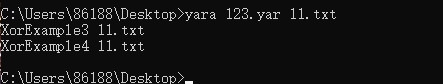
        any of them

}

### 样本：



### 告警:



### 匹配：

56 02 6A 02 6B 02 71 02 22 02 72 02 70 02 6D 02 65 02 70 02 63 02 6F 02 22 02 61 02 63 02 6C 02 6C 02 6D 02 76 02

### **xor(0x01-0xff):**

从 YARA 3.11 开始，如果您想更好地控制与xor修饰符一起使用的字节范围，请使用：

rule XorExample5

{

    strings:

        $xor\_string = "This program cannot" xor(0x01-0xff)

    condition:

        $xor\_string

}

# 八、base64 字符串:

base64修饰符可以用于搜索已base64编码字符串

### 规则:

rule Base64Example1

{

    strings:

        $a = "This program cannot" base64

    condition:

        $a

}

### 匹配字符串“This program cannot”的三个 base64 排列：

VGhpcyBwcm9ncmFtIGNhbm5vd

RoaXMgcHJvZ3JhbSBjYW5ub3

UaGlzIHByb2dyYW0gY2Fubm90

# 九、正则表达式

yara规则允许使用正则表达式, 不过要用正斜杠而非双引号括起来使用(像Perl编程那样)

### 规则:

rule RegularShow

{

    strings:

        $re1 = /md5: [0-9a-fA-F]{32}/

        $re2 = /state: (on|off)/

    condition:

        $re1 and $re2

}

### 匹配:

md5: 123131313123123131313123110000090

state: on

# 十、字符串集

如果你想要中列表中选择一定数量的字符串, 你可以执行以下操作:

### 规则:

rule MigosPresent

{

    strings:

        $m1 = "Quavo"

        $m2 = "Offset"

        $m3 = "Takeoff"

    condition:

        2 of ($m1,$m2,$m3)

}

### 匹配：

如果$m1, $m2和$m3任意存在两个, 那么就满足上述规则中的条件.

Quavo

Offset

或

Quavo

Takeoff

或

Offset

Takeoff

### 通配符\*：与上一条规则等价，通配符来表示一个字符集，如下：

rule MigosPresent

{

    strings:

        $m1 **=** "Quavo"

        $m2 **=** "Offset"

        $m3 **=** "Takeoff"

    condition:

        2 of ($m**\***)

}

### 关键字them：要表示strings中的所有变量, 你可以使用

rule ThreeRappersPresent

{

    strings:

        $m1 = "Quavo"

        $m2 = "Offset"

        $m3 = "Takeoff"

        $q1 = "Cardi B"

    condition:

        3 of them // equivalent to 3 of ($\*)

}

### 匹配：

Quavo

Offset

Cardi B

# 十一、使用关键字any和all

### any:匹配of后面的任意一个

rule Squad

{

    strings:

        $m1 = "Quavo"

        $m2 = "Offset"

        $m3 = "Takeoff"

        $q1 = "Cardi B"

    condition:

        any of ($\*)

}

### all:匹配of后面的所有

rule Squad

{

    strings:

        $m1 = "Quavo"

        $m2 = "Offset"

        $m3 = "Takeoff"

        $q1 = "Cardi B"

    condition:

        all of ($\*)

}

# 十二、匿名字符串

### 规则：

rule AnonymousStrings

{

    strings:

        $ = "dummy1"

        $ = "dummy2"

    condition:

        1 of them

}

### 匹配：

dummy1或dummy2

# 十三、布尔运算：or 和and :

字符串标识符也可在条件中充当布尔变量

### 规则:

rule Example

{

    strings:

        $hero1a = "Batman"

        $hero1b = "Robin"

        $hero2a = "Edward"

        $hero2b = "Alphonse"

    condition:

        ($hero1a or $hero1b) and ($hero2a or $hero2b)

}

### 匹配：

Robin

Edward

# 十四、计数字符串实例：#符引用，字符串出现的次数，

字符串在文件或进程内存中出现的次数，变量名是用#代替$的字符串标识符. 例如:

### 规则:

rule Ransomware

{

    strings:

        $a = "encrypted"

        $b = "btc"

    condition:

        #a == 2 and #b > 2

}

### 匹配：

匹配任何包含两个字符串$a以及出现至少两次字符串$b的文件或进程.

encrypted

encrypted

btc

btc

btc

# 十五、at:字符串偏移(虚拟地址)

需要知道该字符串是否在文件的某个特定偏移处，或是在进程地址空间的某个虚拟地址处

### 规则；

rule Offset

{

    strings:

        $a = "encrypted"

        $b = "btc"

    condition:

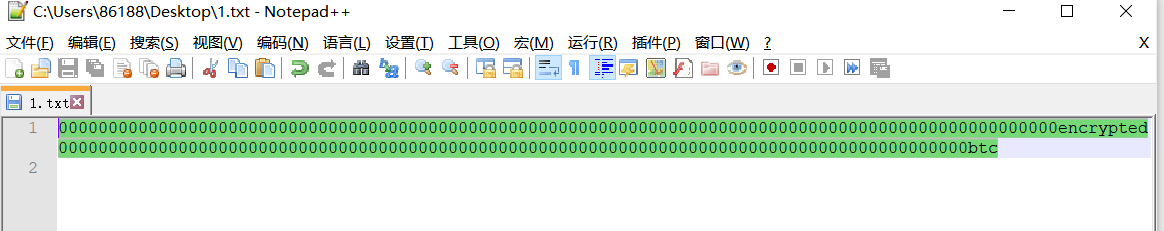
        $a at 100 and $b at 200

}

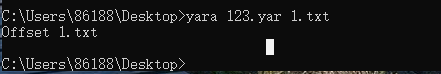
### 匹配：从开头偏移100后，第101处开始匹配，开头偏移200后，第201处开始匹配

0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000encrypted0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000btc

### 文件：



### 告警：



### 可以使用十六进制表示而不一定要十进制.

备注：Python：十进制转十六进制：hex(100),十六进制转十进制：int(‘0x64’,16)

rule Offset

{

    strings:

        $a = "encrypted"

        $b = "btc"

    condition:

        $a at 0x64 and $b at 0xC8

}

### 操作符in：at操作符指定到一个具体的偏移量, 使用操作符in来指定字符串的位置范围.

rule InExample

{

    strings:

        $a = "encrypted"

        $b = "btc"

    condition:

        $a in (0..100) and $b in (100..filesize)

}

### 解释：

字符串$a必须在偏移0-100之间才能找到, 而$b则必须是在偏移100到文件末尾位置(才能找到).

### 匹配：

0000000000000000000000000000000000000encrypted0000000000000000000000000000000000000000000000000000000000000000000000000000000btc000000000000000000000000000000000000000000000000000000000000000000000000000

### @符：使用@a[i]来取得字符串$a第i个字符串的偏移量或虚拟地址，字符串索引以1开头

rule InExample

{

    strings:

        $a="hello"

        $b="world"

    condition:

        $b in (@a[2]..filesize)

}

匹配文件：

0000hello0000000000000000hello0000000world000

解释：

@a[1]==4：匹配第一个hello字符串获取h的偏移量

@a[2]==25：匹配第二个hello字符串获取h的偏移量

故

$b in (@a[2]..filesize)

等价于：

$b in (25..filesize)

# 十六、匹配长度：!符引用：如:!rel[1]，字符串的长度，先匹配字符串，在获取长度值,

如果你有一个正则表达式/fo\*/, 可以匹配字符串fo, foo和fooo, 那么各个的匹配长度都是不同的.字符串标识符前加一个!得到匹配长度，!a[1]是第一个匹配到的字符串$a的长度，而!a[2]就是第二个匹配到的字符串的长度, 依此类推. !a是!a[1]的缩写.

### 规则:

rule Hak5

{

    strings:

        $re1 = /hack[\w]{0,10}/    // Will catch on hacker, hacked, hack, hack\*

    condition:

        !re1[1] == 4 and !re1[2] > 6

}

### 匹配：

We hack things. We are hackers.

# 十七、filesize:文件大小

字符串标识符并不是唯一可以在条件中出现的变量(实际上, 可以不定义任何字符串来编写一个规则), 还可以使用其他变量. filesize就保存着正在扫描的文件的大小. 大小以字节为单位.

### 规则:

rule FileSizeExample

{

    condition:

       filesize < 1KB

}

[!]重要提示: filesize仅在规则应用于文件的时候生效. 如果应用于正在运行的进程, 那么它会永远都匹配不了.

# 十八、for…of: 对许多字符串应用同一个条件

要用for循环来检查一组字符串是否满足特定条件, 如

### 例子1：

rule ForExample

{

    strings:

        $s1="123"

        $s2="abc"

        $s3="qwe"

    condition:

        for all of them : (# > 1)

}

### 解释：所有字符串出现的次数都大于1

### 匹配：

123

abc

qwe

123

abc

qwe

### 例子2：

rule ForExample2

{

    strings:

        $a="hello"

        $b="world"

    condition:

        for all of ($a\*) : (@>@b)

}

### 匹配：

0000000000000world0000000000000hello000000000000

### 解释：

所有a开头的字符串的第一个出现的首字符偏移量偏移量都要大于字符串b首字符偏移量

本例子为：

31>13

# 十九、引用其他规则：如:$a and Rule1，直接使用规则名即可

### 规则:

rule Rule1

{

    strings:

        $a = "dummy1"

    condition:

        $a

}

rule Rule2

{

    strings:

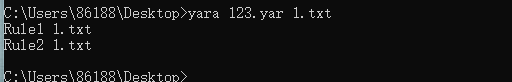
        $a = "dummy2"

    condition:

        $a and Rule1

}

### 告警:



# 二十、规则标签

使用 YARA 时，您只能输出那些用您提供的一个或多个标签标记的规则。

### 规则:

rule TagsExample1 : Foo Bar Baz

{

        condition:

        filesize < 2MB

}

rule TagsExample2 : Bar

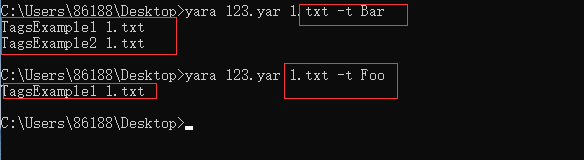
{

        condition:

        filesize < 2MB

}

### 告警：



# 二十一、使用模块:import “pe”

比如PE和Cukoo模块

### 规则:

import "pe"

rule CobaltStrike4\_1:beacon

{

    meta:

        description = "This rule is used to detect CobaltStrike4.1 Trojan"

        author = "lengyi"

        md5 = "3f0e09ee93d6fa767e7d8c2c891f96e6"

        refer = "http://cn-sec.com/archives/140061.html"

    strings:

        $name = "%c%c%c%c%c%c%c%c%cMSSE-%d-server"

    condition:

        uint16(0) == 0x5A4D and pe.entry\_point ==0x8b0 and filesize > 10KB and filesize < 24KB and $name

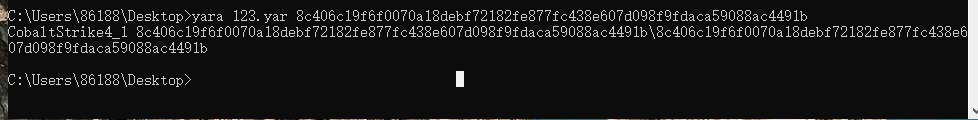
}

### 解释：

1. 文件头为MZ
2. 入口点代码为0x8b0
3. 文件大小大于10kb，小于24kb

4、包含$name字符串

### 告警：



# 二十二、外部变量

### 规则：

rule ExternalVariable1

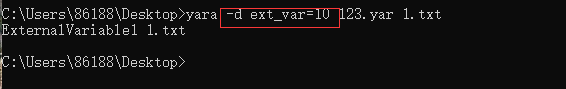
{

    condition:

       ext\_var == 10

}

### 告警：



### 解释：-d：自定义外部变量ext\_var=10



### contains和matches操作符：外部变量可以和操作符contains和matches一起使用

### 规则：

rule ExternalVariable2

{

    condition:

        string\_ext\_var contains "text"

}

rule ExternalVariable3

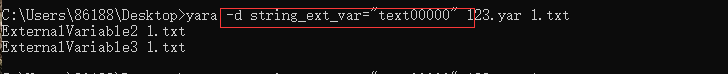
{

    condition:

        string\_ext\_var matches /[a-z]+/

}

### 告警：



### 解释：

contains在字符串包含特定子串的情况下返回true

matches在字符串匹配给定的正则表达式时返回true.

### /[a-z]+/is中的i表示匹配时不区分大小写. s表示是在单行(single line)模式

### 规则:

rule ExternalVariableExample5

{

    condition:

        /\* case insensitive single-line mode \*/

        string\_ext\_var matches /[a-z]+/is

}

# 二十三、文件包含:include “1.yar”

你可以在包含时使用相对路径或绝对路径. 如果是windows系统, 甚至还可以是驱动设备的路径.

### 规则：

include "1.yar"

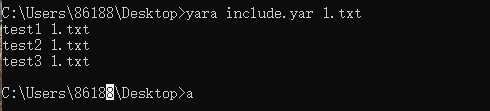
include "../2.yar"

include "C:\\Users\\86188\\3.yar"

//include "/home/user/yara/IsRapper.yar"

//include "c://yara/includes/oldRappers.yar"

### 告警：



# 结束!