

欢迎来到:

一场意料之外的旅途，深入

Microsoft Defender的签名世界

https://shorturl.at/bPCzW

https

:

//retooling.io/blog

你需要:

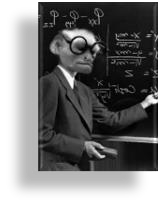
1. ResourceHacker (~3MB)
2. 基于msys64构建的环境 (~350MB)
3. 测试环境

https://shorturl.at/bPCzW

我们是谁…. "一群意大利博士" 以及 …

# Silvio Antonio Davide

|  |
| --- |
| @t0nvi |

* Co-founder of Retooling LLC ❑ Co-founder of Retooling LLC ❑ Master’s degree student in Cybersecurity

@ University of Sapienza

* Former Senior Cyber Security Architect @ ❑ Former Senior Cyber Security

LEONARDO Spa - Cyber Security Architect @ LEONARDO Spa - Cyber ❑ Bachelor’s degee in Information Division Security Division Technology @ University of L’Aquila

@DrCh40s ❑ Senior Security Researcher @ EMC/RSA -> ❑ Cyber Threat Analyst / Reverse ❑ Passionate about malware analysis and

 DELL – Center of Excellence reverse engineering

Engineer

* Malware reverse engineer @ Symantec -
* PhD System Security @ University of

Security Response

Roma Tre

* PhD Network Security @ University of

Pisa ❑ M.Sc. in Computer Science

* M.Sc. in Computer Engineering

@davidefont96

|  |
| --- |
| antonio@retooling.io |

silvio@retooling.iodavidefontana96.df@gmail.com

一场意外的旅途：



反转

恶意软件



模拟执行



特征检测



检测逃避



感觉易如反掌？不，一点也不

|  |
| --- |
| *PingPull基于Visual C++ 构建，为攻击者提供了在被入侵主机上执行命令并执行反向Shell连接的能力。*  *PingPull 有三种变种，它们功能相同，但使用不同协议与其 C2 通信：ICMP、HTTP(S) 和原始 TCP。*  *——Palo Alto, Unit42* |

|  |
| --- |
| 将新威胁整合入Retooling Revo |

* 起点：PingPull.exe
* 初始目标：PingPull.sln
* 终点： Defender.IDB

Microsoft Defender Antivirus 的框架

MpSvc.dll

MpEngine.dll

MsMpEng.exe

WdFilter.sys

WdBoot.sys

MpClient.dll

MpCmdRun.exe

MpClient.dll

MpRtp.dll

远程过程调用(RPC)

用户态

内核态

VDM

注册表

读取

1

2

1.

ELAM 驱动

2.

Minifilter

驱动

3.

PPL

Ioctl用户/内核通信

3

# Microsoft Defender的签名文件

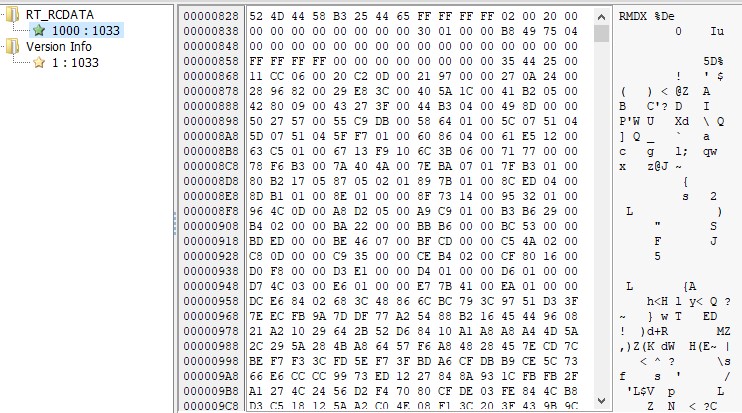
* 位于: C:\ProgramData\Microsoft\Windows

Defender\Definition Updates\<RandomGUID>\ ❑ 可移植应用程序:

* mpa{s,v}base.vdm : 每月更新, 包含反恶意软件/反间谍软件签名
* mpa{s,v}dlta.vdm: 不定期频繁更新, 包含对上述基础库的反恶意软件/反间谍软件更新
* 关注mpavbase.vdm和mpasbase.vdm

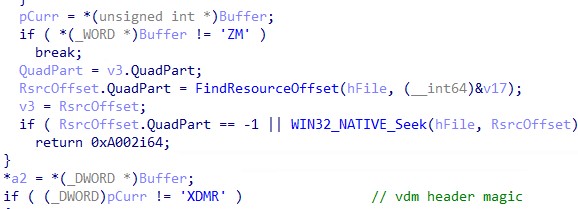
# mpavbase.vdm 和 mpasbase.vdm

* 均在其资源字节段（.rsrc）包含压缩的数据（签名）
* 引导时, mpengine合并 \*base.vdm和\*delta.vdm文件



魔法字符串

LoadMoudleHeader



压缩数据



偏移量



校验值？

# 多种类型的签名

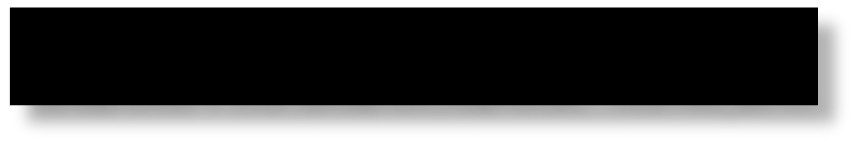
|  |
| --- |
| switch (a1) { ... case 0x79u:  return "SIGNATURE\_TYPE\_VDLL\_X86"; case 0x6Bu:  return "SIGNATURE\_TYPE\_WVT\_EXCEPTION"; case 0x6Cu:  return "SIGNATURE\_TYPE\_REVOKED\_CERTIFICATE"; case 0x70u:  return "SIGNATURE\_TYPE\_TRUSTED\_PUBLISHER"; case 0x71u:  return "SIGNATURE\_TYPE\_ASEP\_FILEPATH"; case 0x73u:  return "SIGNATURE\_TYPE\_DELTA\_BLOB"; case 0x74u:  return "SIGNATURE\_TYPE\_DELTA\_BLOB\_RECINFO"; case 0x75u:  return "SIGNATURE\_TYPE\_ASEP\_FOLDERNAME"; case 0x77u:  return "SIGNATURE\_TYPE\_PATTMATCH\_V2"; case 0x78u: return "SIGNATURE\_TYPE\_PEHSTR\_EXT"; ...  } |

# 实验0: 解压 Windows Defender的签名文件

1. 打开文件夹 C:\ProgramData\Microsoft\Windows Defender\Definition Updates\<Your\_GUID\_Here>\
2. 复制 mpavbase.vdm 到你的工作目录

|  |
| --- |
| 无gz文件头 |

1. 剪切出上述的压缩数据，并保存为x.gz



import

zlib

compressed

=

open

(

'x.gz'

,

'

rb

'

).

read

()

decompressed

=

zlib

.

decompress

(

compressed

,

-

zlib

.

MAX\_WBITS

)

4. 在同目录下运行此 python3脚本

Labab

# 预计从解压的 vdm 文件中输出

|  |  |  |
| --- | --- | --- |
| |  | | --- | | * 包含涉及到威胁名称的ASCII 字符串 ： * !Hupigon * !Plugx.C * … * 各威胁名称间距不一 | |  |
|  |

Microsoft Defender Antivirus 结构（重放）

MpSvc.dll

MpEngine.dll

MsMpEng.exe

WdFilter.sys

WdBoot.sys

MpClient.dll

MpCmdRun.exe

MpClient.dll

MpRtp.dll

RPC

usermode

kernelmode

VDM

Registry

Read

1

2

1.

ELAM Driver

2.

Minifilter

Driver

3.

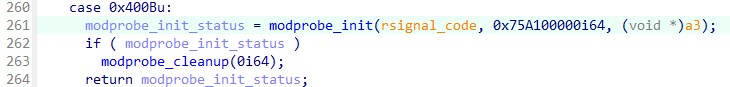
PPL

ioctl

3

# 阶段1: 签名数据库预加载

一旦对签名文件的预处理完成，Defender模块初始化启动……

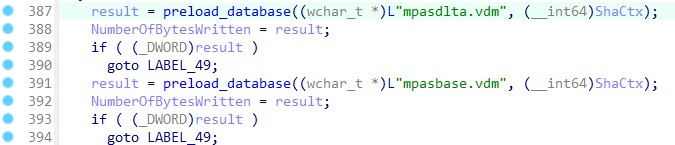


读取文件头并检索一般信息，如签名版本和数字

LoadModuleHeader

:

加载数据库头前16字节



modprobe\_init\_worker

ksignal

**签名版本 1.401.1166.0**

阶段2 ：初始化Defender各模块



AutoInitModules

Initialize

::

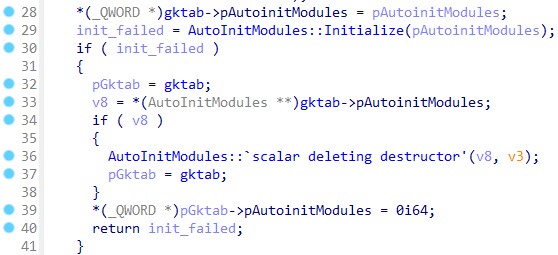
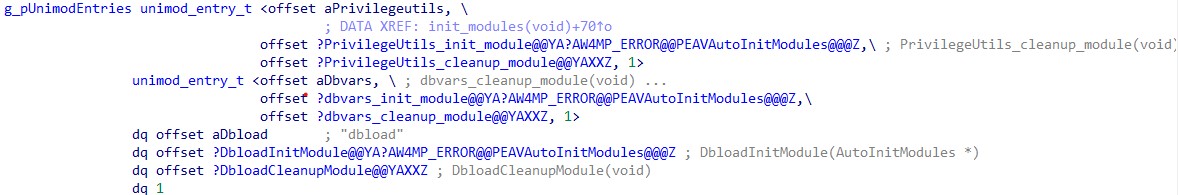
循环所有在

g\_pUnimodEntries中的模块

并调用模块专用的初始化函数

pfnInit

()



init\_modules

1

2

3

# cksig\_init\_module pattsearch\_init

包括 和

❑

调用

pattsearch\_init

初始化包含签名的数据结构

g\_HstrSigs

g\_DynamicHstrSigs

❑

这些符号是指向一个哈希表的指针，包含所有文件类型（原文HSTR）的指针（例如ELF PE MACOS等）

❑

load\_database/load\_database\_cache

会

DispatchRecords（分配上述记录）到正确的存储栈



x?

g\_HstrSigs

0

x

61

x78

Pehstr

records

Pehstrext

records

Pehstrext2 records



0x61

0x78

PEHSTR records

PEHSTREXT records

PEHSTREXT2 records

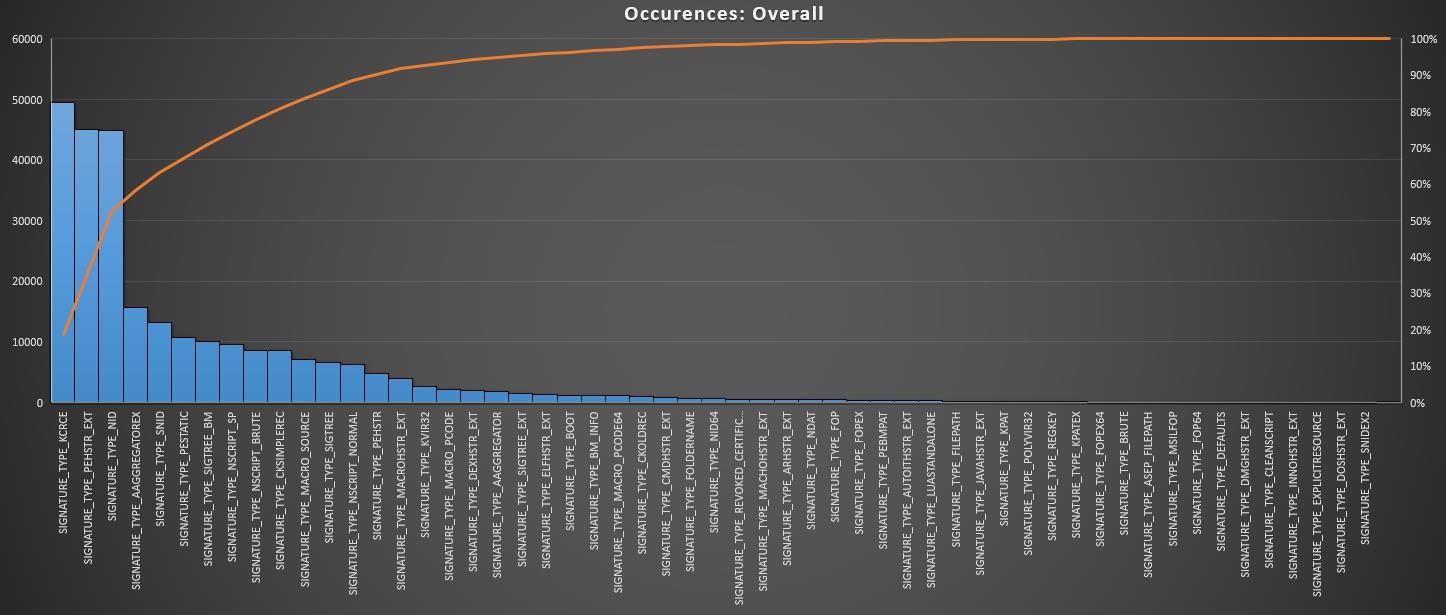
hstr\_search 是用于执行扫描的函数之一

其使用所有类别的hstr签名

0x85

0x?

# 数字和统计：出现次数

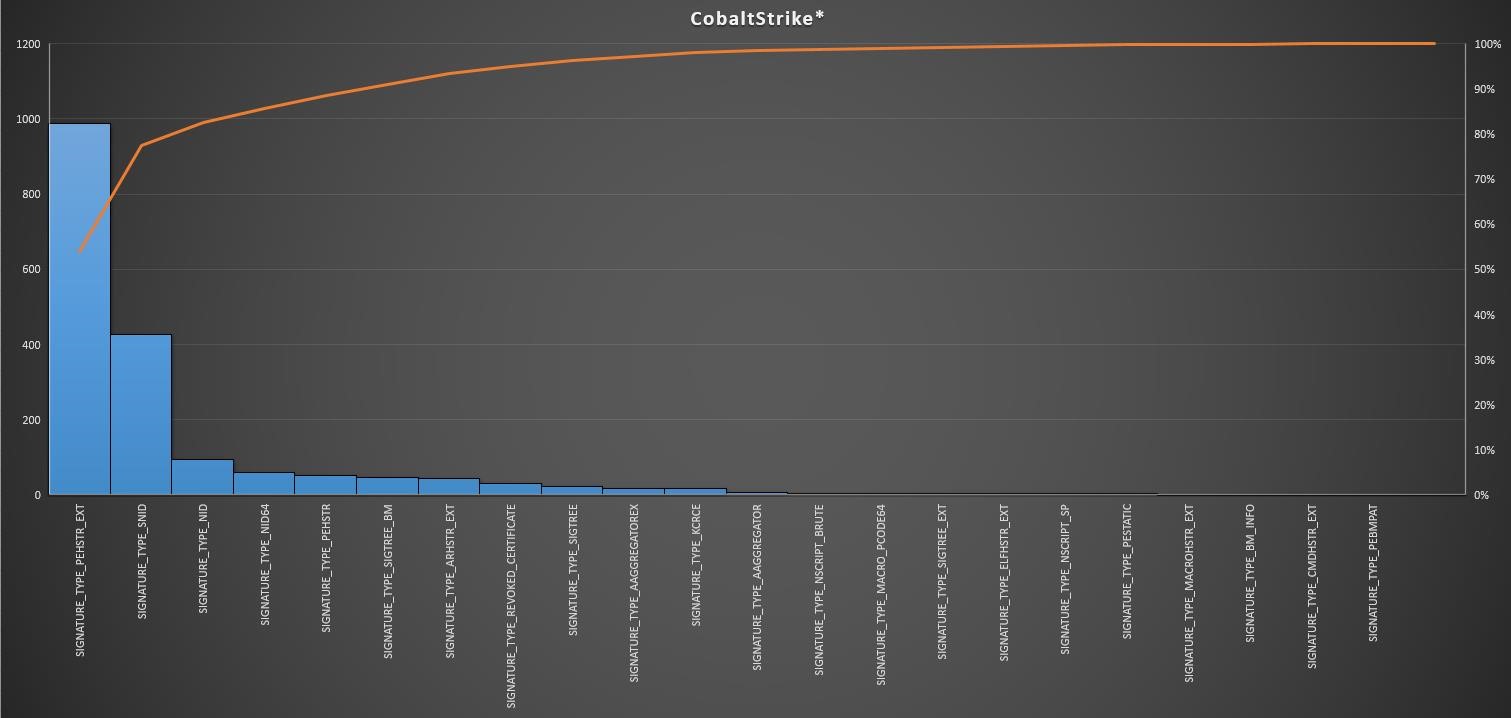
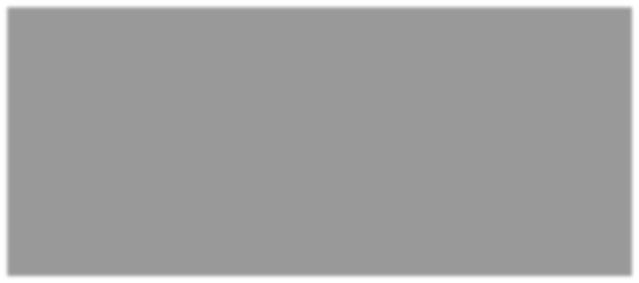
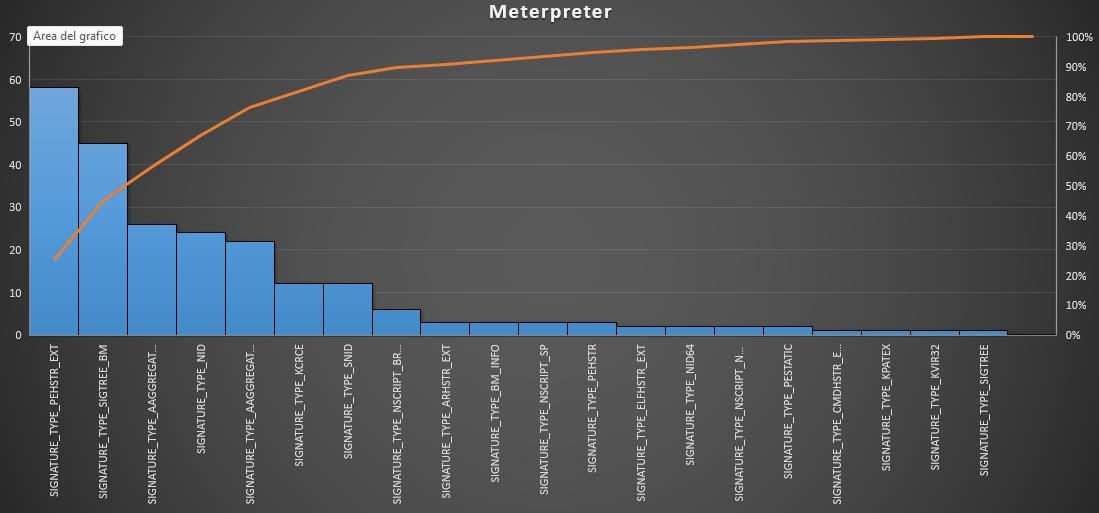
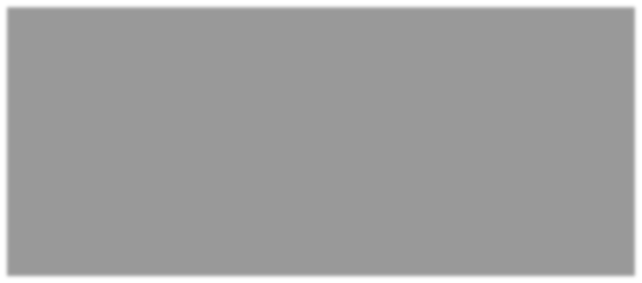
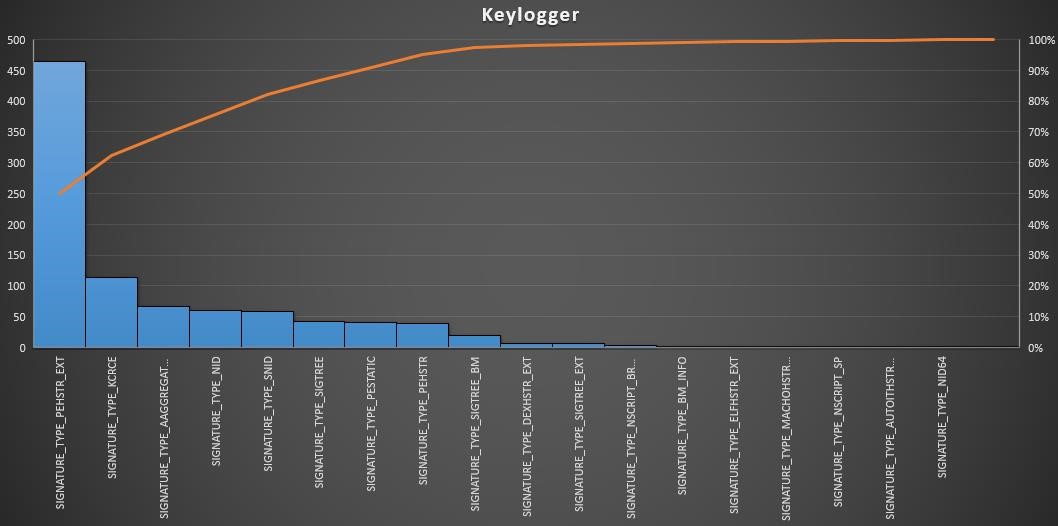
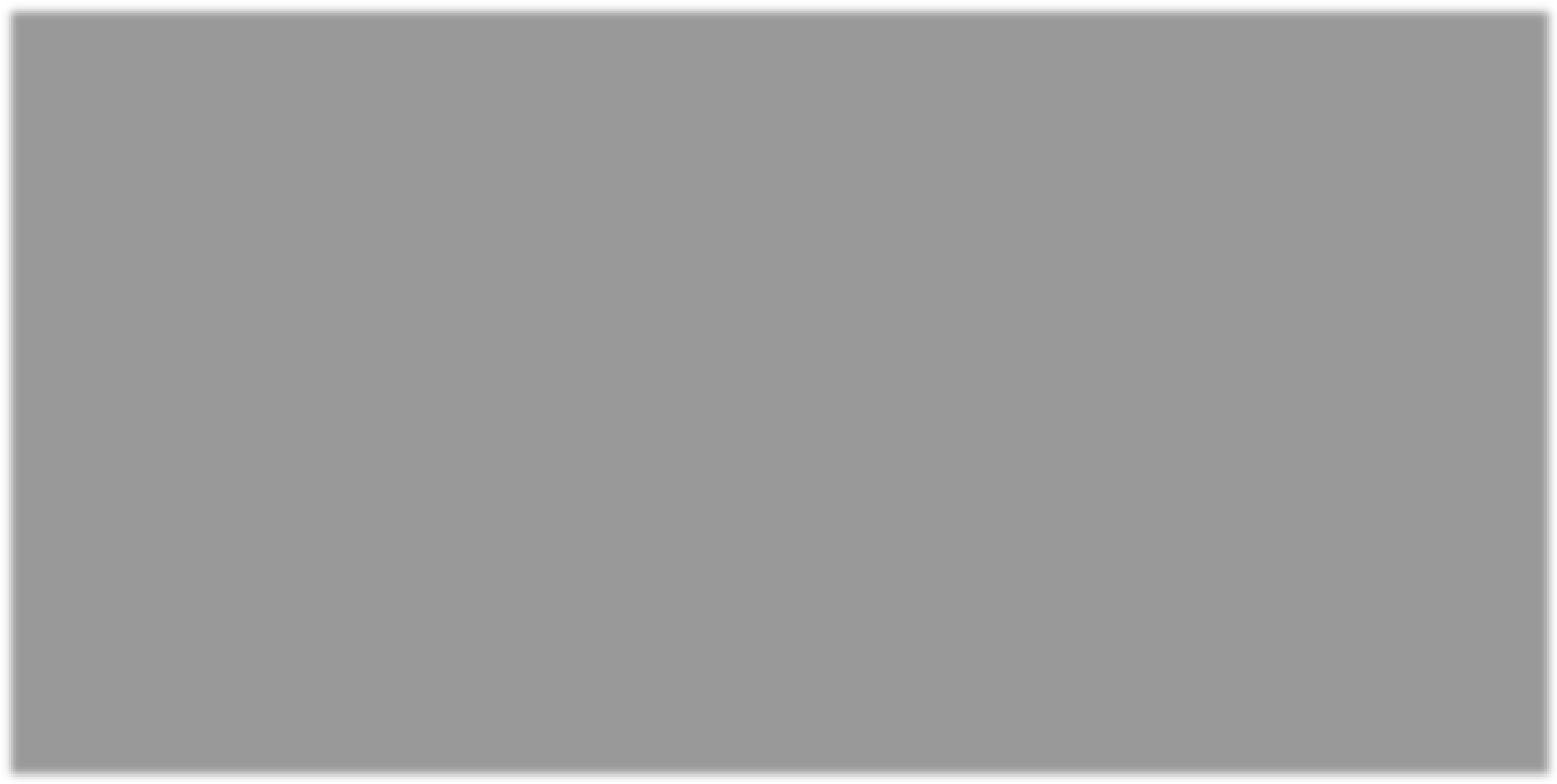
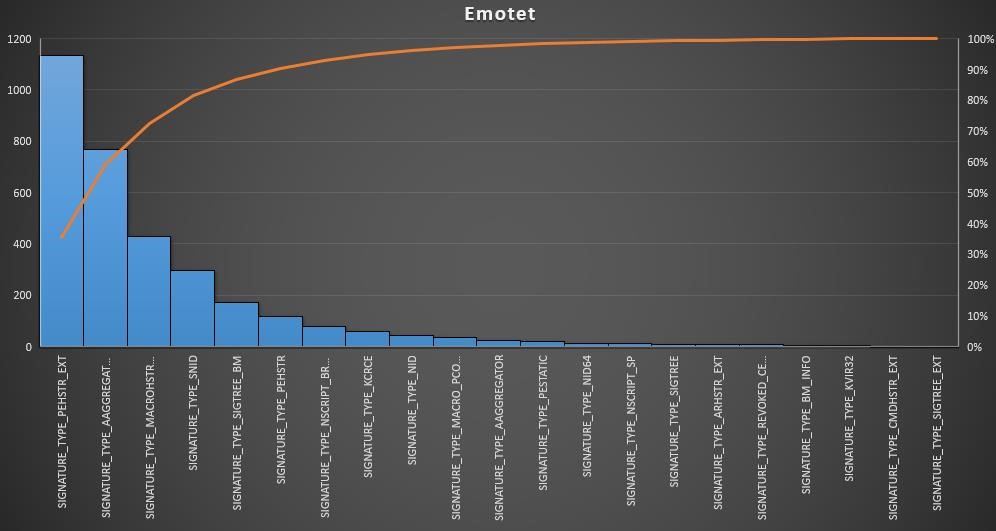
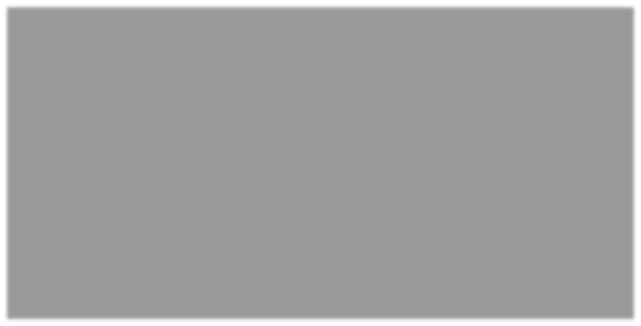


~49K

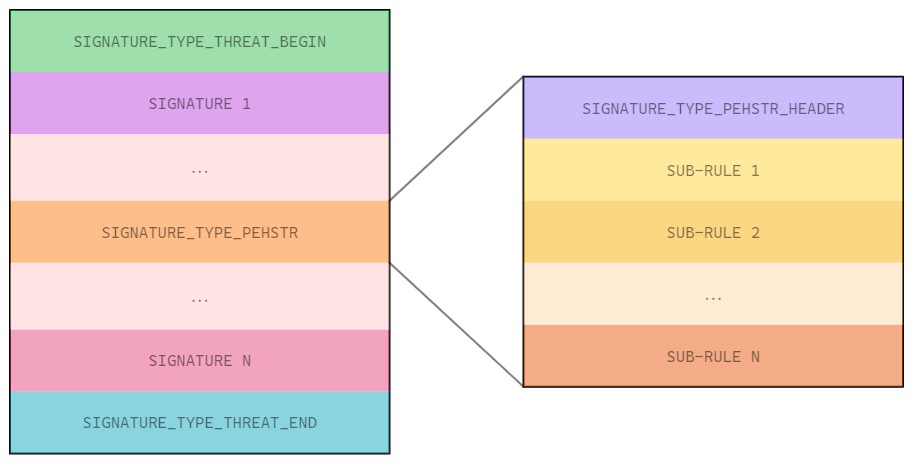
~46K

~5K

对特定威胁进行切分



# 签名通用结构



# 开始到结束

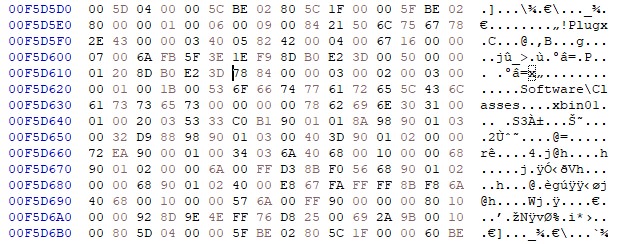
SIGNATURE\_TYPE\_KCRCE

SIGNATURE\_TYPE\_THREAT\_END

SIGNATURE\_TYPE\_STATIC

SIGNATURE\_TYPE\_THREAT\_BEGIN

SIGNATURE\_TYPE\_PEHSTR\_EXT



SIGNATURE\_TYPE\_THREAT\_BEGIN

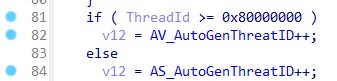
和

SIGNATURE\_TYPE\_THREAT\_END

包含自定义数据

❑

其中之一是四字节的规则ID （示例：0x8002be5f）



createrecid

typedef

struct

\_STRUCT\_

COMMON

\_SIGNATURE\_TYPE

{

UINT8

ui8SignatureType;

//

定义签名类型

UINT8

ui8SizeLow;

// 签名低位字节大小

UINT16

ui16SizeHigh;

// 签名高位字节大小

BYTE

pbRuleContent

[]

;

//

规则内容

}

;

# SIGNATURE\_TYPE\_THREAT\_BEGIN

❑

用相对检测名称定义威胁的起点，示例：

!Plugx.C

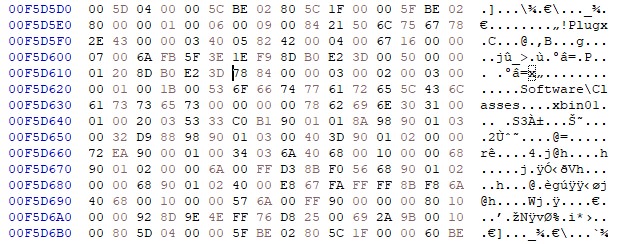
❑

代码标识符：

0x5C

❑

内部包含不同的签名，用于标记威胁



typedef

struct

\_STRUCT\_SIG\_TYPE\_THREAT\_BEGIN

{

UINT8

ui8SignatureType;

UINT8

ui8SizeLow;

UINT16

ui16SizeHigh;

UINT32

ui32SignatureId;

BYTE

unknownBytes1[6];

UINT8

ui8SizeThreatName;

BYTE

unknownBytes2[2];

CHAR

l

pszThreatName

ui8SizeThreatName];

[

BYTE

unknownBytes3[9];

}

STRUCT\_SIG\_TYPE\_THREAT\_BEGIN

,

\*

PSTRUCT\_SIG\_TYPE\_THREAT\_BEGIN

;

SIGNATURE\_TYPE\_THREAT\_END

❑

❑

定义威胁的终点

代码标识符：

0x5D

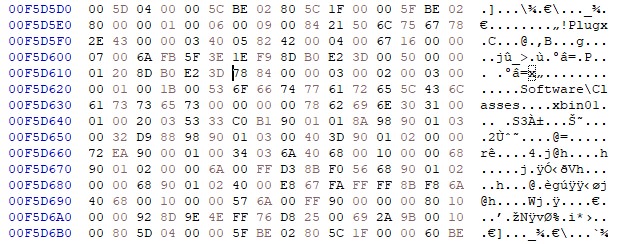
❑

pbRuleContent

值和

相应ui32SignatureId值相同

SIGNATURE\_TYPE\_THREAT\_BEGIN中使用的



typedef

struct

\_STRUCT\_SIG\_TYPE\_THREAT\_END

{

UINT8

ui8SignatureType;

UINT8

ui8SizeLow;

UINT16

ui16SizeHigh;

BYTE

pbRuleContent

;

[]

}

STRUCT\_SIG\_TYPE\_THREAT\_END,

\* PSTRUCT\_SIG\_TYPE\_THREAT\_END

;

SIGNATURE\_TYPE\_PEHSTR vs SIGNATURE\_TYPE\_PEHSTR\_EXT

❑

SIGNATURE\_TYPE\_PEHSTR\_EXT

用于匹配PE文件的字节

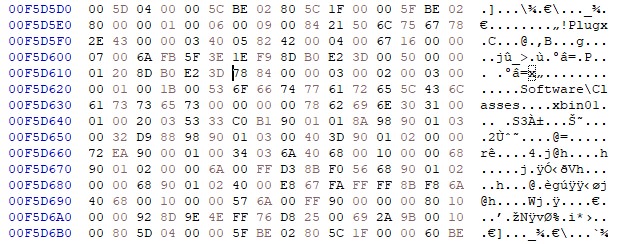
❑

代码标识符：

78

x

0



❑

SIGNATURE\_TYPE\_PEHSTR

用于实行对PE文件的字符串匹配

❑

代码标识符：

61

x

0

typedef

struct

\_STRUCT\_COMMON\_SIGNATURE\_TYPE

{

UINT8

ui8SignatureType;

UINT8

ui8SizeLow;

UINT16

ui16SizeHigh;

BYTE

pbRuleContent

;

[]

}

STRUCT\_COMMON\_SIGNATURE\_TYPE

,

\*

PSTRUCT\_COMMON\_SIGNATURE\_TYPE

;

# PEHSTR 和 PEHSTR\_EXT 公共标头

❑

ui8TresholdRequiredLow:

此阈值需要源自Windows Defender（低位）的检测

❑

ui8TresholdRequiredHigh:

此阈值需要源自Windows Defender（高位）的检测

❑

ui8SubRulesNumberLow:

在此特定签名中找到的子规则数字，用于识别威胁（低位）

❑

ui8SubRulesNumberHigh:

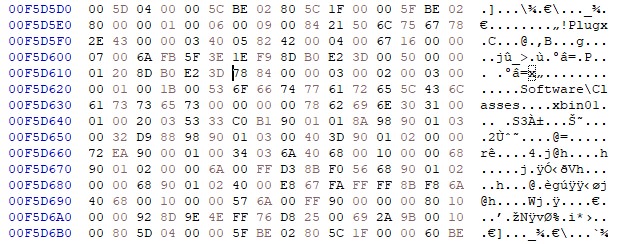
在此特定签名中找到的子规则数字，用于识别威胁（高位）

❑

pbRuleData

[]:

包含所有子规则，用于字节匹配检测



❑

所有类型的签名共享同种结构

❑

主要区别在于子规则格式略有不同

❑

SIGNATURE\_TYPE\_PEHSTR

用于检测“可读字符串”

❑

SIGNATURE\_TYPE\_PEHSTR\_EXT

可用于检测操作码并拥有额外特殊功能

typedef

struct

\_STRUCT\_PEHSTR\_HEADER

{

UINT16

ui16Unknown;

UINT8

ui8TresholdRequiredLow;

UINT8

ui8TresholdRequiredHigh;

UINT8

ui8SubRulesNumberLow;

UINT8

ui8SubRulesNumberHigh;

BYTE

bEmpty

;

BYTE

pbRuleData

;

[]

}

STRUCT\_PEHSTR\_HEADER

, \*

PSTRUCT\_PEHSTR\_HEADER

;

SIGNATURE\_TYPE\_PEHSTR\_EXT

# PEHSTR 和 PEHSTR\_EXT 子规则结构

❑

ui8SubRuleWeightLow

表示子规则在检测过程中的权重（低位）

❑

ui8SubRuleWeightHigh

表示子规则在检测过程中的权重（高位）

❑

ui8SubRuleSize

指定用于匹配给定PE的字节字符串的大小

❑

ui8CodeUnknown

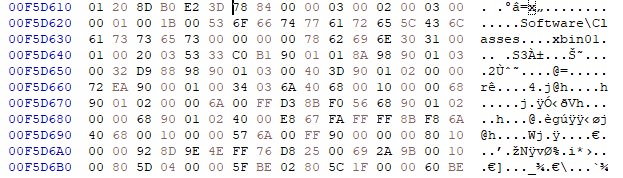
未知区域

❑

pbSubRuleBytesToMatch

[]:

为取得检测而必须找到的字节



typedef

struct

\_STRUCT\_RULE\_PEHSTR\_EXT

{

UINT8

ui8SubRuleWeightLow;

UINT8

ui8SubRuleWeightHigh;

UINT8

ui8SubRuleSize;

UINT8

ui8CodeUnknown;

//\_EXT

only

BYTE

pbSubRuleBytesToMatch

;

[]

}

STRUCT\_RULE\_PEHSTR\_EXT

,

\*

PSTRUCT\_RULE\_PEHSTR\_EXT

;

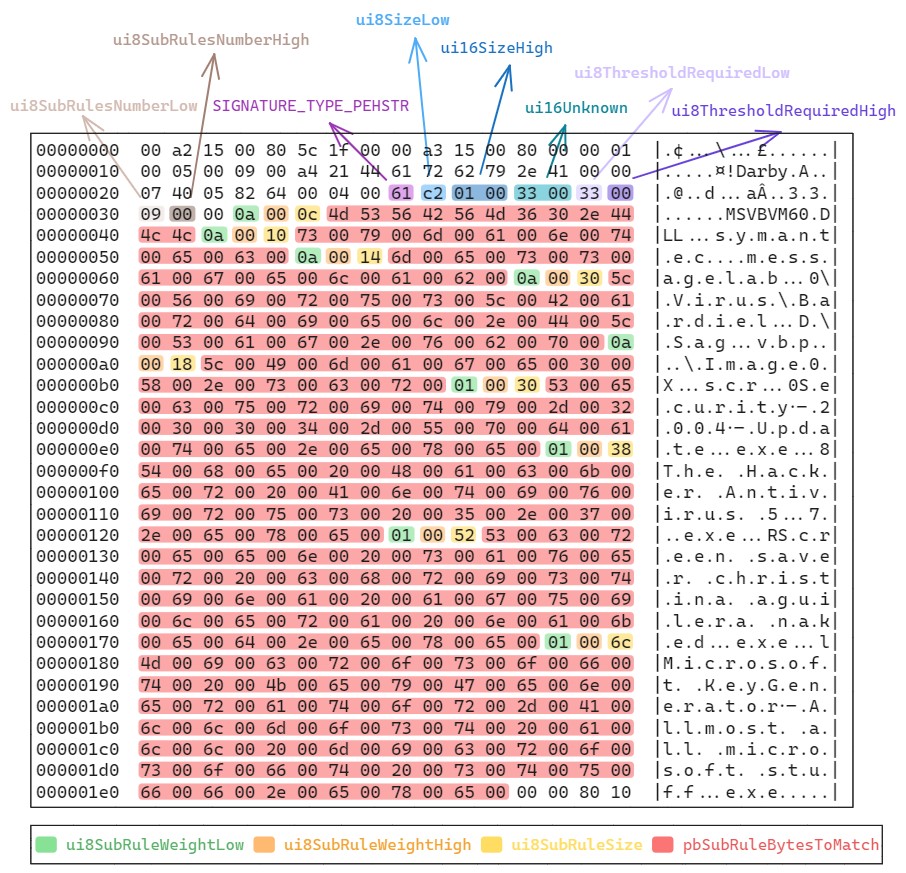
三个子规则示例

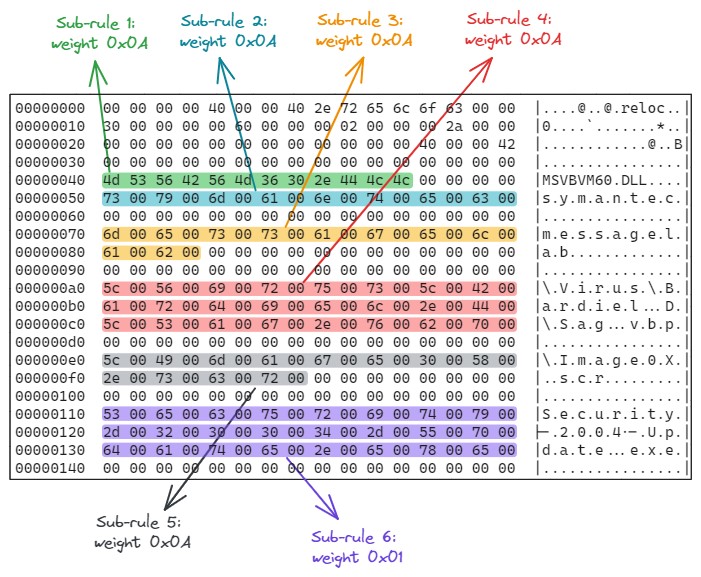
实验1: SIGNATURE\_TYPE\_PEHSTR

* 用十六进制编辑器打开解压后的 mpavbase.vdm 并找到所有属于威胁 !Darby.A的SIGNATURE\_TYPE\_PEHSTR (0x61)
* 标记每个签名的所有区域 （提示：将十六进制转存中的相关字节截图，用Windows画图截取相关字节并高亮显示）
* 识别子规则
* 识别阈值
* 识别每个子规则的权重

Labab

# 解决SIGNATURE\_TYPE\_PEHSTR: 真实样本

* 图中示例显示了一个源自威胁 !Darby.A 的SIGNATURE\_TYPE\_PEHSTR
* \_STRUCT\_PEHSTR\_HEADER:
* ui16Counter1: 用 青色 标记
* ui16TresholdRequired: 用 紫色 标记
* ui16SubRulesNumber: 用 棕色 标记
* \_STRUCT\_RULE\_PEHSTR:
* ui16SubRuleWeight: 用 绿色 标记
* ui8UnknownCode: 用 橙色 标记
* ui8SubRuleSize: 用 黄色 标记
* pbSubRuleBytesToMatch[]: 用 红色 标记

 SIGNATURE\_TYPE\_PEHSTR: 匹配 !Darby.A 签名

* 此签名包含等于0x33的ui16TresholdRequired

必须达到阈值才能取得检测

* 本样本匹配下列子规则

Sub-rule 1: 权重 0x0A.

Sub-rule 2: 权重 0x0A.

Sub-rule 3: 权重 0x0A.

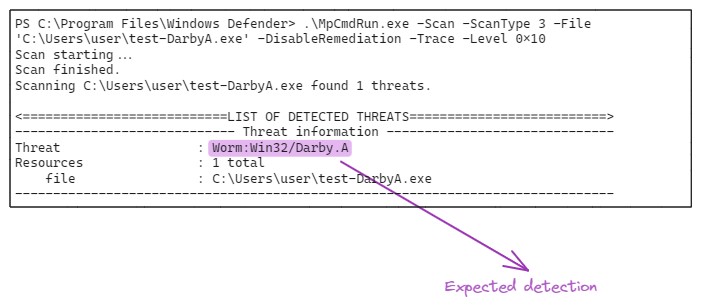
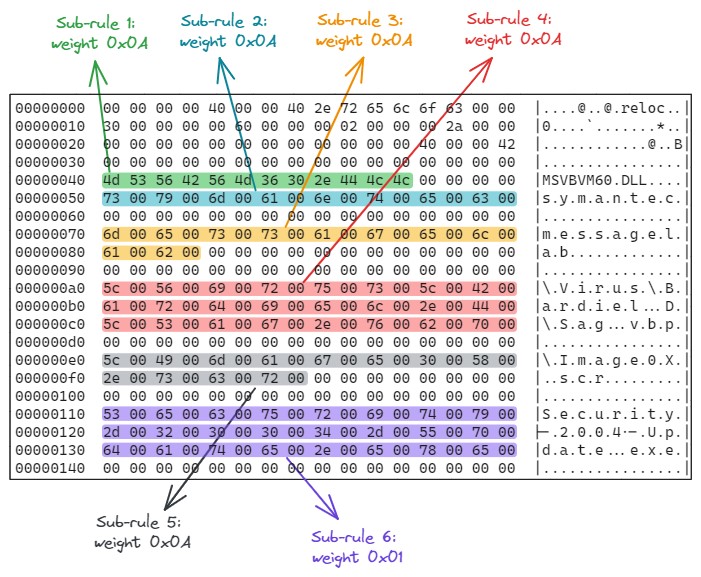
总计：0x33

Sub-rule 4: 权重 0x0A.

Sub-rule 5: 权重 0x0A.

Sub-rule 6: 权重 0x01.

使用Windows Defender提供的MpCmdRun.exe扫描你的文件



PS C:

\

Program Files

\

Windows Defender> .

\

MpCmdRun.exe

–

Scan

–

ScanType

3

–

File <

*filepath*

>

-

DisableRemediation

–

Trace

–

Level 0x10

## 实验2: 移除 Darby 的签名检测

* 添加一个文件夹到 Defender的排除项

PS> Add-MpPreference -ExclusionPath 'C:\YOUR\_PATH\_HERE'

* 将包含 Darby 的压缩文件复制到排除的文件夹并解压（密码：infected）
* 用十六进制编辑器打开二进制文件
* 找到哪些字节命中了签名并修改，用以规避检测

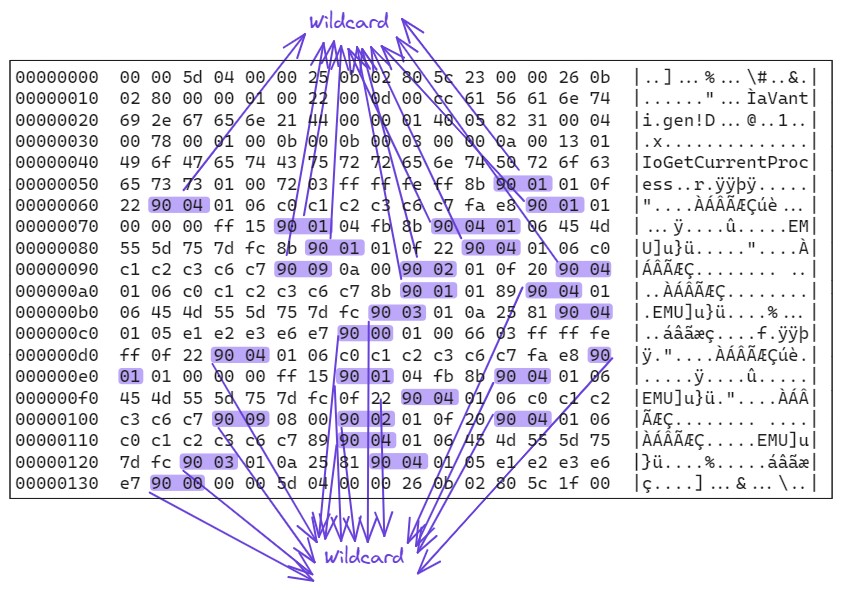
至少需要修改多少字节才能绕过检测？

* 当同一子规则出现多次，对总权重有何影响？

假设字符串 S\_1 （拥有权重 W\_1） 在二进制中出现了2次，这个二进制是否会获得2\*W\_1的权重?

Labab

为 EXT加权

* SIGNATURE\_TYPE\_PEHSTR\_EXT中的子规则里存在多种形式

可被用于检测操作码和其他类型

❑ 用于匹配特定的字节序列

* 已确定的通配符：

90 01 XX

90 02 XX

90 03 XX YY

90 04 XX YY

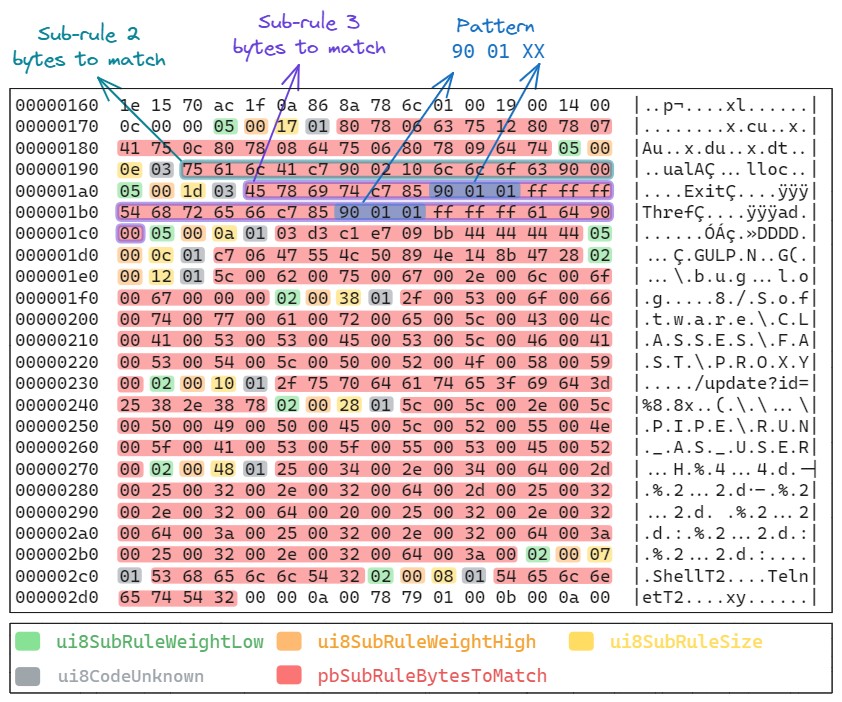
90 05 XX YY

* 未确认的通配符：

90 06 -> 90 20

# 模块90 01 XX

模式 90 01 XX:

* 用于SIGNATURE\_TYPE\_PEHSTR\_EXT的子规则
* 匹配一个顺序特定，长度由XX定义的字节段，

顺序必定紧接 XX 后出现

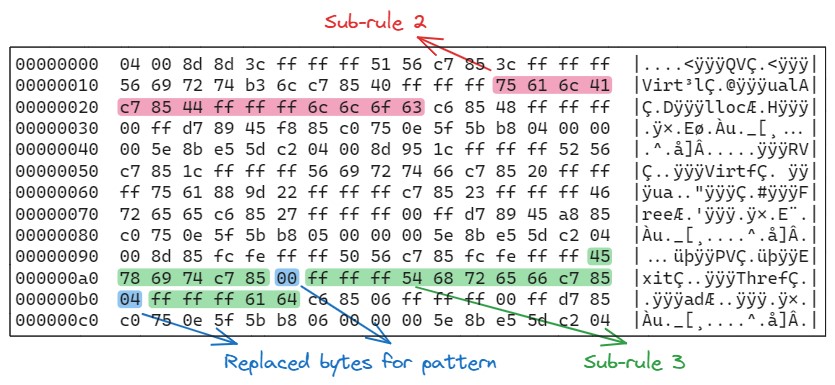
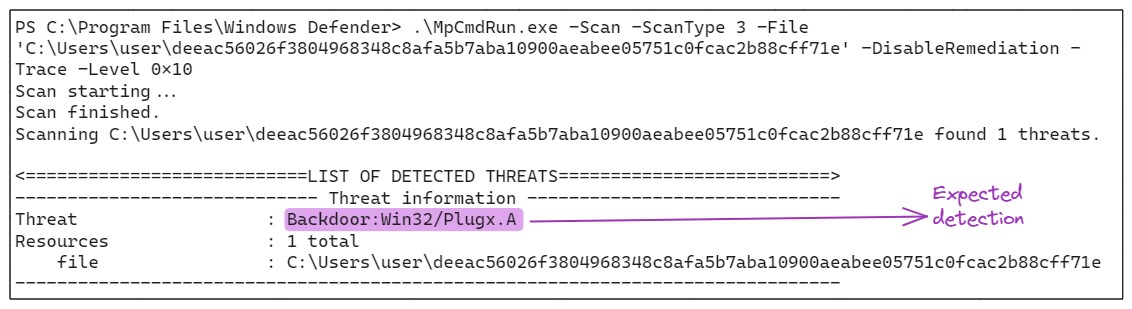
* 示例用蓝色标记

|  |
| --- |
| PlugxA-Sub-Rule3-Example{ strings:  $sub\_rule\_3\_hex = {  45 78 69 74 C7 85 ?? FF FF FF 54 68  72 65 66 C7 85 ?? 04 FF FF FF 61 64  }  condition:  $sub\_rule\_3\_hex } |

# 90 01 XX

模块 90 01 XX 检测:

* 使用MpCmdRun.exe
* 取代 90 01 01 模块的是 (用蓝色标记):



0x00

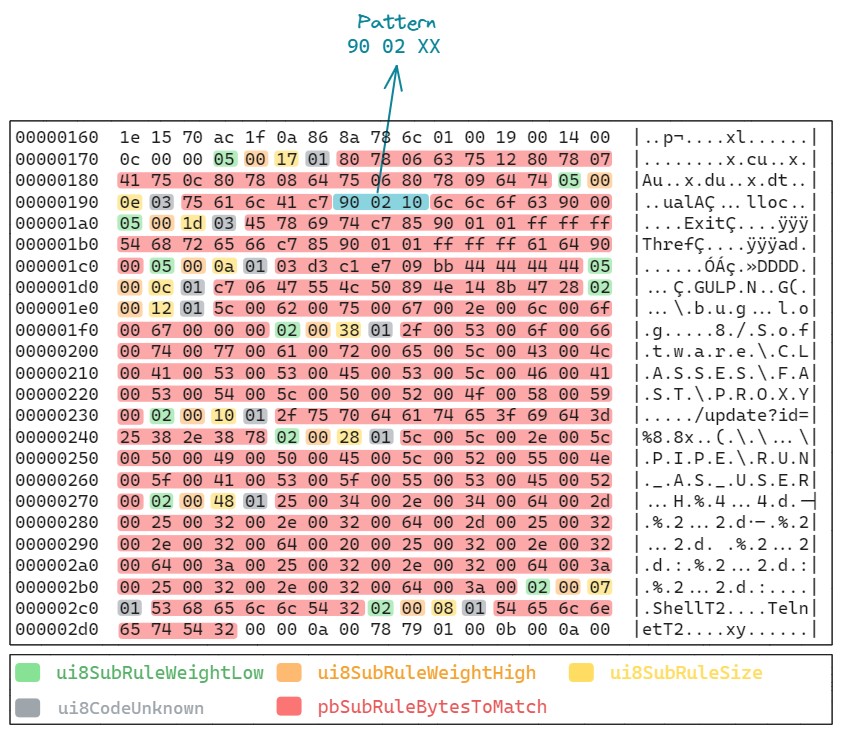
0x04

在红色子规则2中

在绿色子规则3中

* 预计检测为: Plugx.A

# 90 02 XX

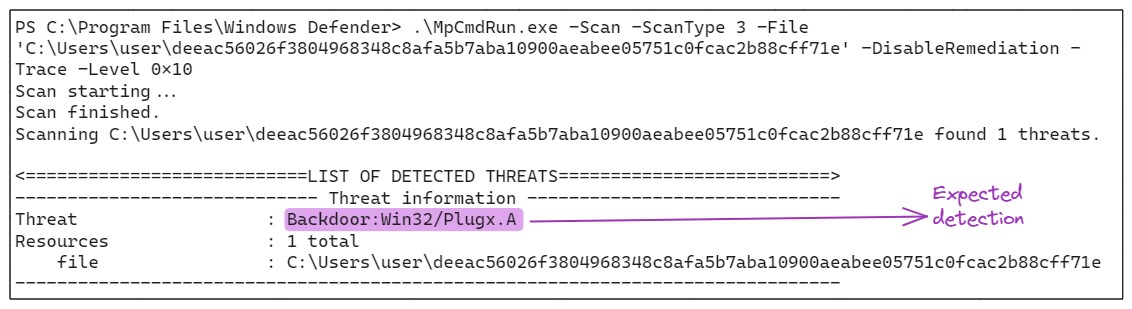
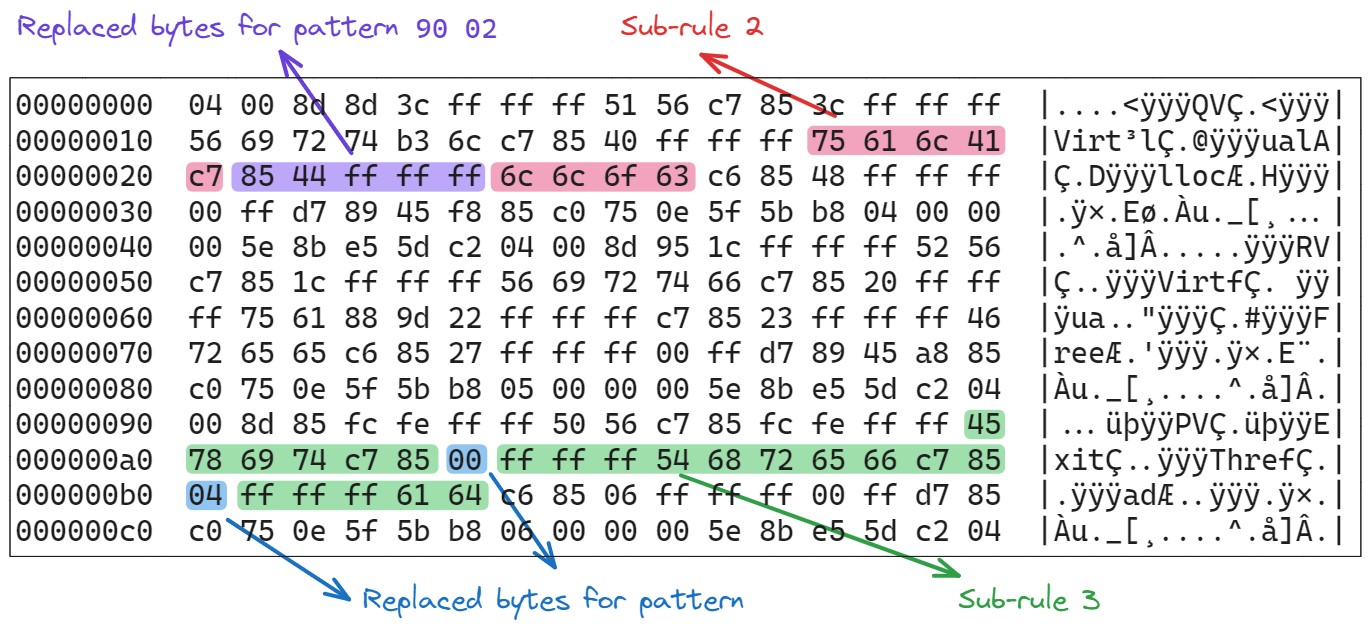
模块 90 02 XX:

用作占位符，最多可匹配 XX个特定字节

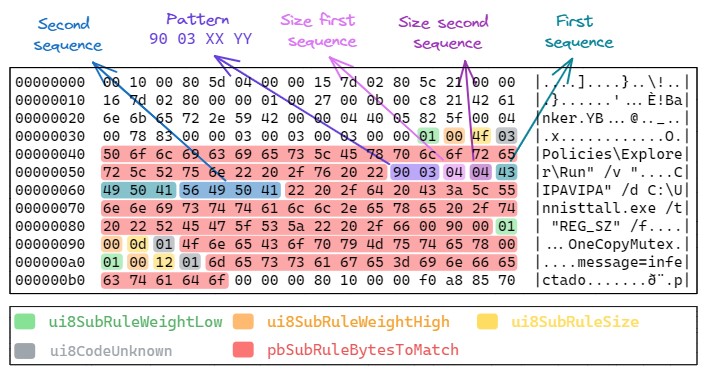
|  |
| --- |
| rule PlugxA-Sub-Rule2-Example { strings:  $sub\_rule\_2\_hex = {75 61 6C 41 C7 [0-16] 6C 6C 6F 63 }  condition:  $sub\_rule\_2\_hex } |

示例模块标记为 青色

# 90 02 XX

* 字节位于模块 90 02 10 标记为 紫色
* 整个子规则 2 标记为 红色

# 90 03 XX YY

模块 90 03 XX YY:

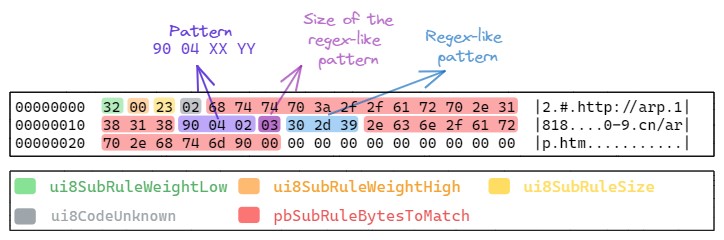
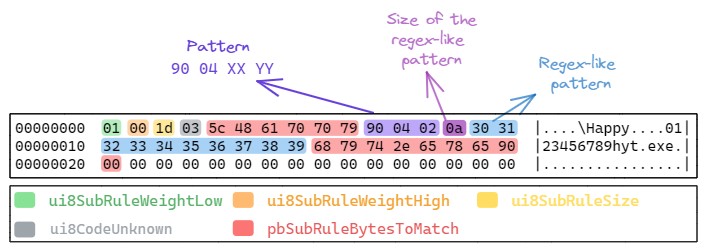
* XX : 首个序列（序列A）长度 ，标记为粉色
* YY : 次个序列 (序列\_B) ，标记为 葡萄紫
* 匹配的样本中序列A或B或许会出现

|  |
| --- |
| rule BankerYB\_Sub\_Rule1\_Example{ strings:  $sub\_rule\_1\_hex = { 50 6f 6c 69 63 69 65 73 5c 45 78 70 6c 6f 72 65 72 5c 52 75 6e 22 20 2f 76 20 22  **(43 49 50 41|56 49 50 41)** 22 20 2f 64 20 43 3a 5c 55 6e 6e 69 73 74 74 61 6c 6c 2e  65 78 65 20 2f 74 20 22 52 45 47 5f 53 5a 22 20 2f 66 00 90 00 } condition: $sub\_rule\_1\_hex  } |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| rule Pattern-90-04-example { strings:  $example1\_90\_04 =   |  |  |  | | --- | --- | --- | | **[30** | **-** | **39] [30** |   { 68 74 74 70 3a 2f 2f 61 72 70 2e 31 38 31 38 **-**   |  | | --- | | **39]** |   2e 63 6e 2f 61 72 70 2e 68 74 6d 90 00 }  $example2\_90\_04 =   |  |  |  |  |  | | --- | --- | --- | --- | --- | | **[30** | **-** | **39] [30** | **-** | **39]** |   { 5c 48 61 70 70 79 68 79 74 2e 65 78 65 90  00 }  condition:  $ example1\_90\_04 and $ example2\_90\_04 } |

# 90 04 XX YY

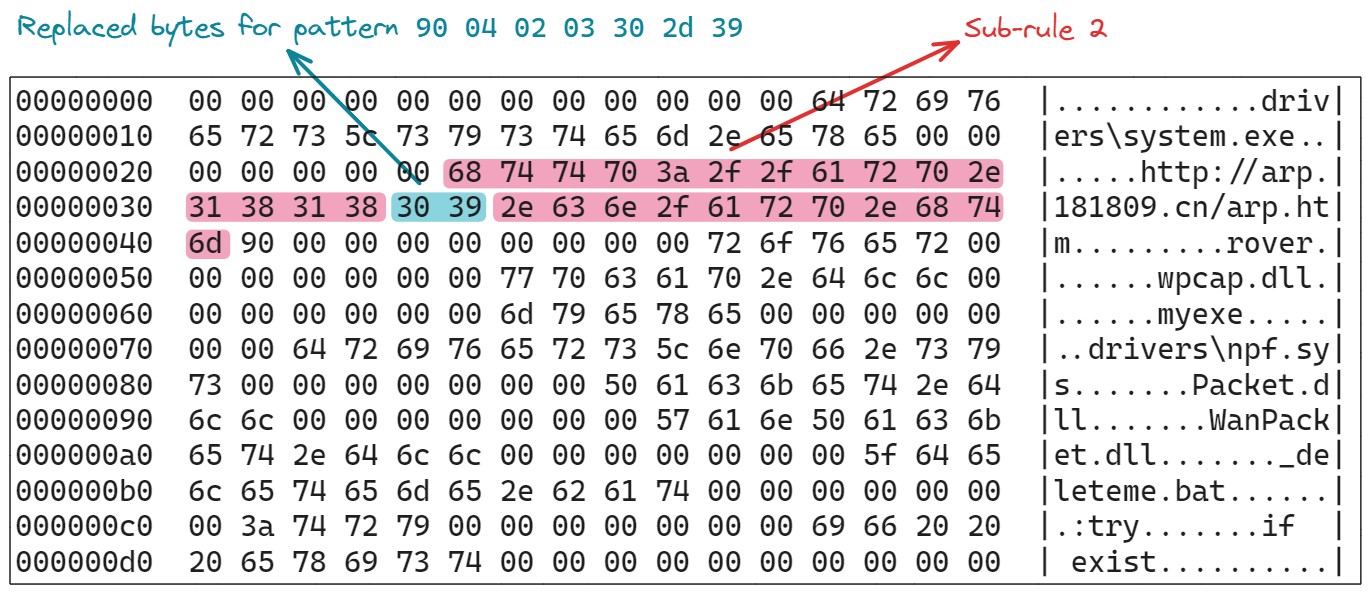
* XX : 预计字节长度
* YY : 类正则表达式模块在图中标记为紫色
* 90 04 XX YY 后字节以正则表达式方式描述模块本身
* 此示例为 30 2d 39, 用蓝色标记是0-9



example1

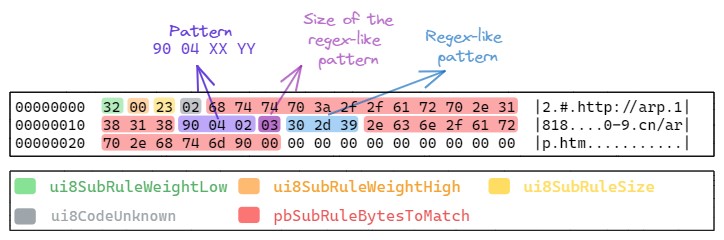
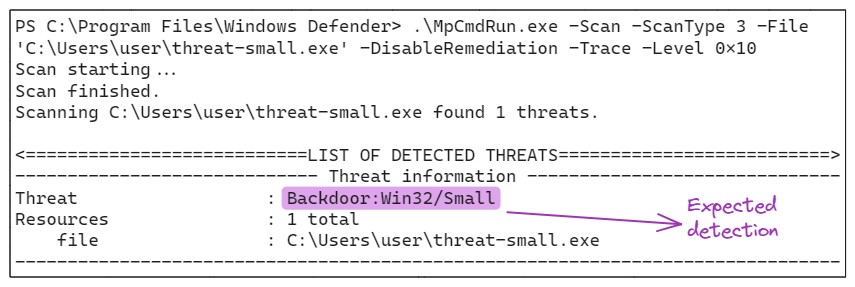
example2

# 90 04 XX YY

* 取代 90

04 02 03 30 2D 39 (示例1) 的字段为:

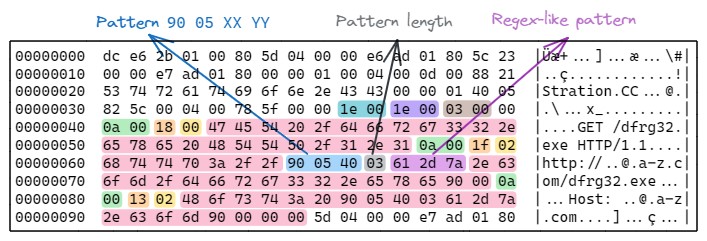
* 0x30
* 0x39
* 标记为青色



example1

* 红色字段命中子规则

# 90 05 XX YY

* XX : 预计字段最大长度
* YY : 图中正则匹配式长度标记为葡萄紫
* 和 04模块不同，此参数不区分大小写

|  |  |
| --- | --- |
| rule Pattern-90-05-example{ strings:  $example\_90\_05 =   |  | | --- | | [a-zA-Z]{0,64} |   "http:// \\.com/dfrg32\\.exe“  condition:  $example\_90\_05  } |

* 90 05 XX YY后的字段用正则匹配式描述模块本身

## 实验3: 匹配检测

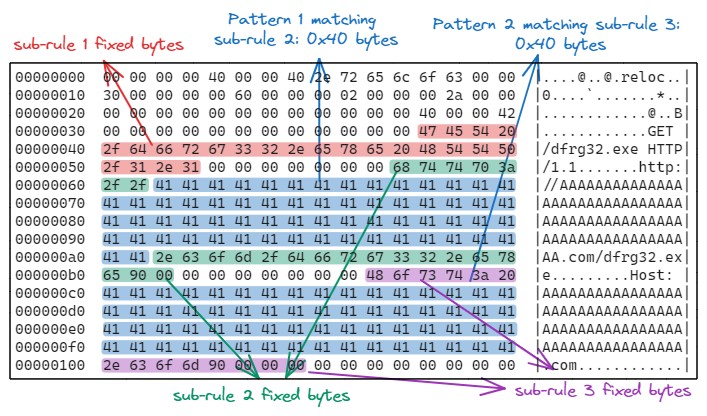
* 打开 msys64 文件夹并运行 msys64.exe
* 切换至实验根目录

cd /c/<your\_path>/lab3\_stration/Exercise ❑ 分析 Stration.CC PEHSTR 签名

了解权重和通配符

* 修改提供的 StrationCC.c 文件使其在编译后同Stration.CC检测结果匹配
* 用Build.sh脚本进行编译

解决方案: 模块: 90 05 XX YY



* 子规则2中取代了90 05 40 03 61 2D 7A模块的字段标记为蓝色
* 子规则2于绿色 中修复的字段
* 预计检测:

Stration.CC

## 最终实验

❑ 目标： 实现一个可运作的命中了Defender Backdoor:Win64/Havoc.A!MTB签名检测的样本

打开解压的签名数据库并找到签名

* + 明确签名类型
  + 明确签名字节的意义

1. 反编译在 lab4\_havoc\Exercise\sample.zip 提供的样本（这是个真样本，小心，密码：infected）
   * 找到并分析命中检测的功能
2. 修改 lab4\_havoc\Exercise\havoc\_emu\_asm.S 以实现与提供的XorAlgorithm样本相同的功能
3. 使用 build.sh 脚本构建



*That’s All Folks*

https

:

//retooling.io/blog

silvio@retooling.io

antonio@retooling.io

## 参考

* https://www.safebreach.com/blog/defender-pretender-when-windowsdefender-updates-become-a-security-risk/
* https://gist.githubusercontent.com/mattifestation/3af5a472e11b7e135273e71cb5f ed866/raw/15be4f2ae75b2d62465cf9faef72a2f61147a393/ExpandDefenderSig.p s1
* https://learn.microsoft.com/en-us/defender-endpoint/command-linearguments-microsoft-defender-antivirus