
https://shorturl.at/bPCzW
https://retooling.io/blog
欢迎来到：
一场意料之外的旅途，深入
Microsoft Defender 的签名世界

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

✈ Co-founder of Retooling LLC

✈ Former Senior Cyber Security Architect @ LEONARDO Spa - Cyber Security Division

✈ Senior Security Researcher @ EMC/RSA -> DELL – Center of Excellence

✈ Malware reverse engineer @ Symantec - Security Response

✈ PhD System Security @ University of Pisa

✈ PhD Network Security @ University of Roma Tre

✈ M.Sc. in Computer Engineering

silvio@retooling.io

@DrCh40s

Antonio

✈ Co-founder of Retooling LLC

✈ Former Senior Cyber Security Architect @ LEONARDO Spa - Cyber Security Division

✈ Cyber Threat Analyst / Reverse Engineer

✈ M.Sc. in Computer Science

antonio@retooling.io

Davide

✈ Master's degree student in Cybersecurity @ University of Sapienza

✈ Bachelor's degee in Information Technology @ University of L'Aquila

✈ Passionate about malware analysis and reverse engineering

@t0nvi

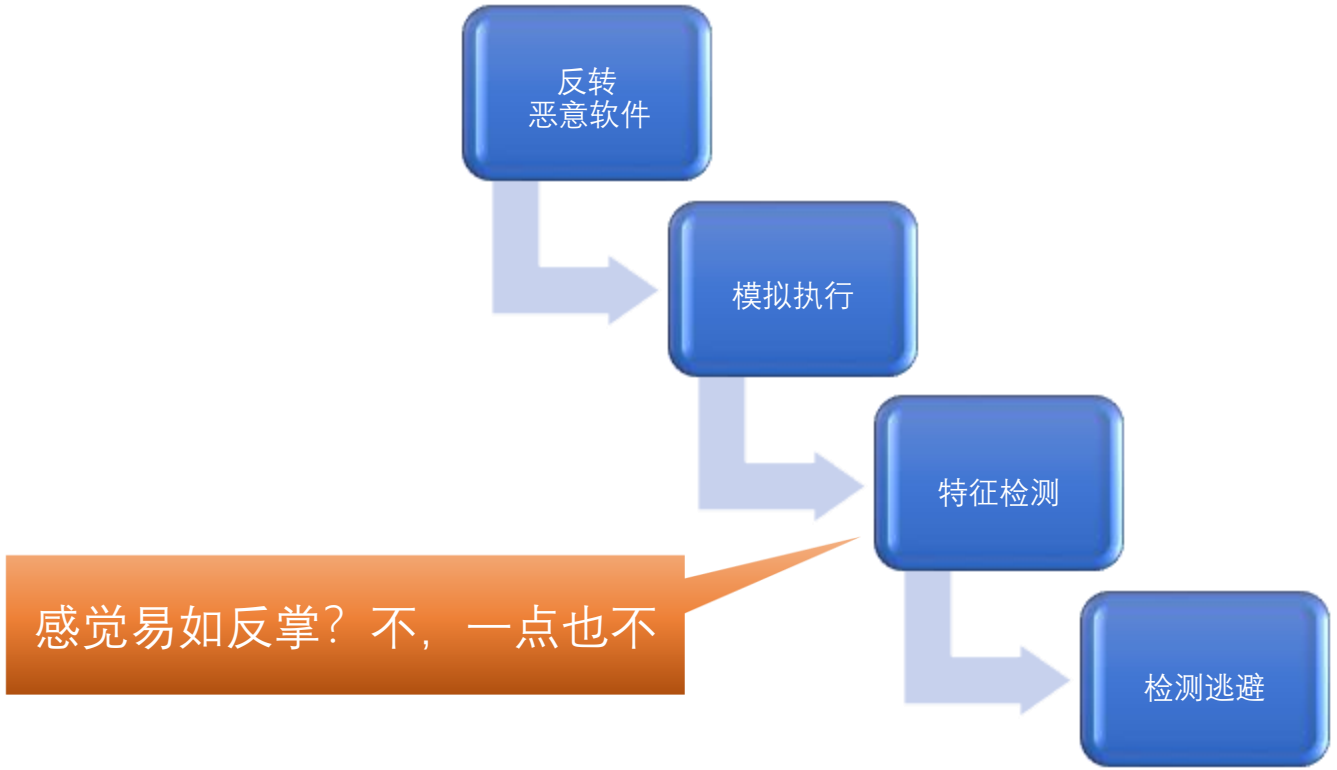
@davidefont96

davidefontana96.df@gmail.com

一场意外的旅途：

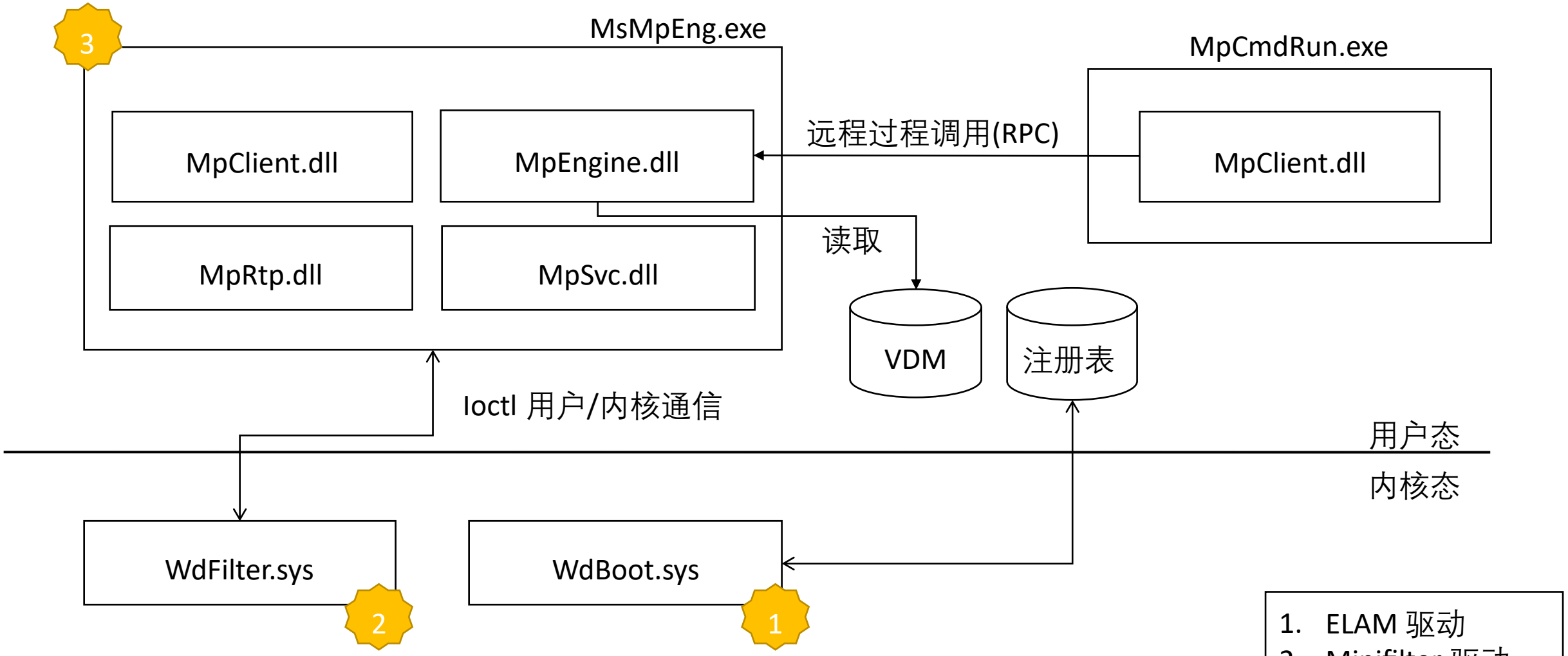
- 起点：PingPull.exe
- 初始目标：
  PingPull.sln
- 终点： Defender.IDB

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

Microsoft Defender Antivirus 的框架



1. ELAM 驱动
2. Minifilter 驱动
3. PPL

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

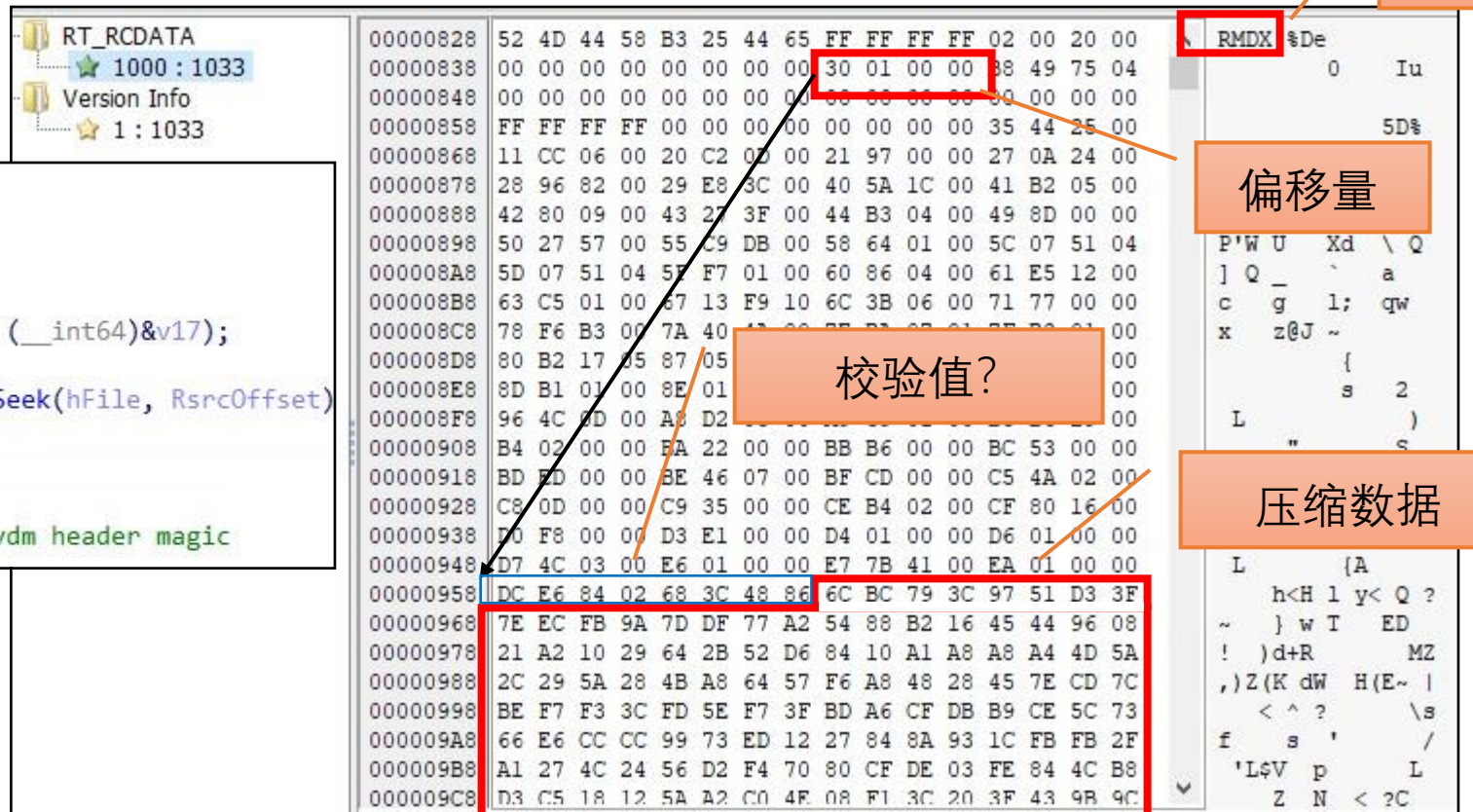
mpavbase.vdm 和 mpasbase.vdm

- 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

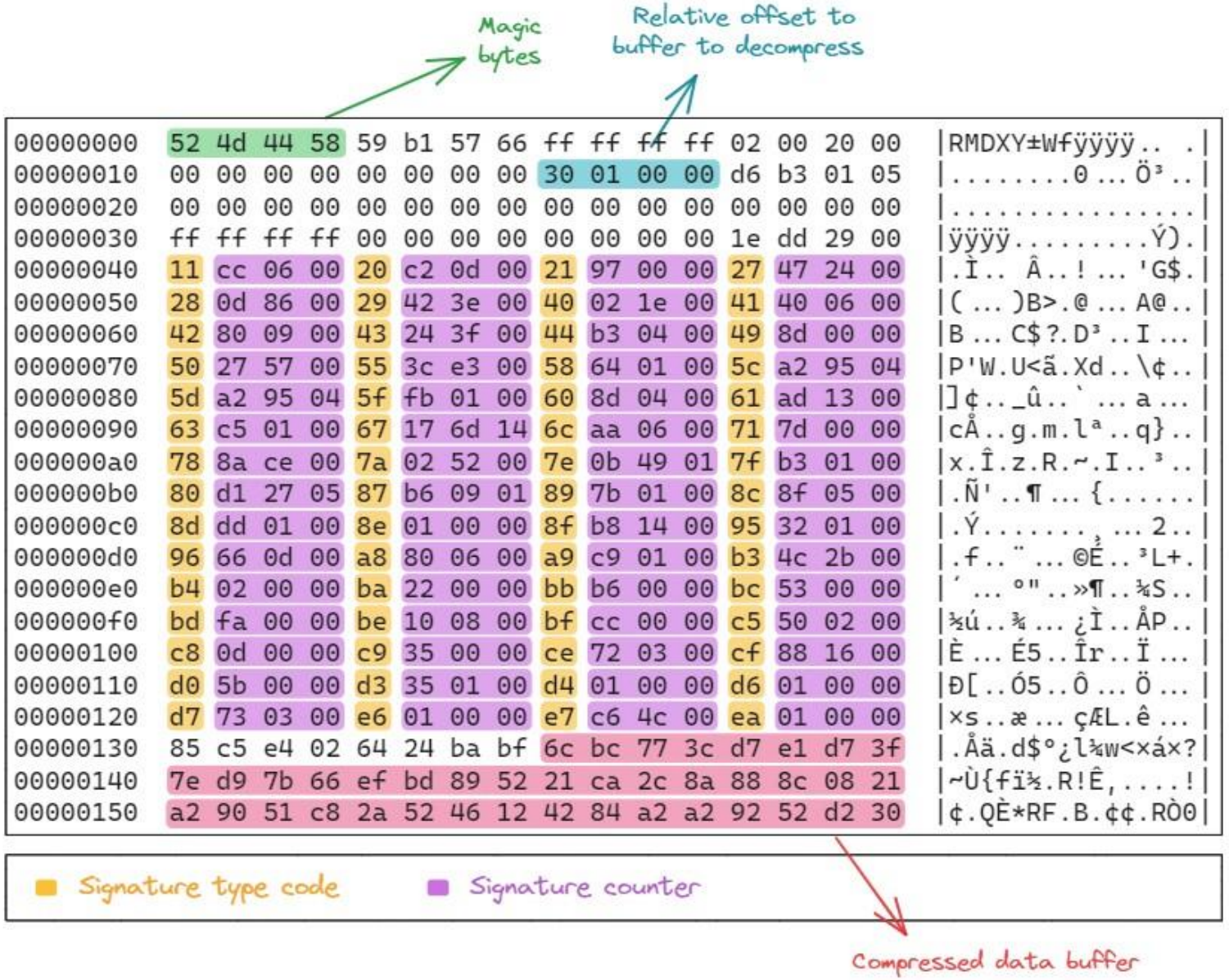
LoadMoudleHeader

```
  pCurr = *(unsigned int *)Buffer;
  if ( *(_WORD *)Buffer != 'ZM' )
    break;
  QuadPart = v3.QuadPart;
  RsrcOffset.QuadPart = FindResourceOffset(hFile, (__int64)&v17);
  v3 = RsrcOffset;
  if ( RsrcOffset.QuadPart == -1 || WIN32_NATIVE_Seek(hFile, RsrcOffset)
    return 0xA002i64;
}
*a2 = *(_DWORD *)Buffer;
if ( (_DWORD)pCurr != 'XDMR' )                  // vdm header magic
```



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
3. 剪切出上述的压缩数据，并保存为 `x.gz`

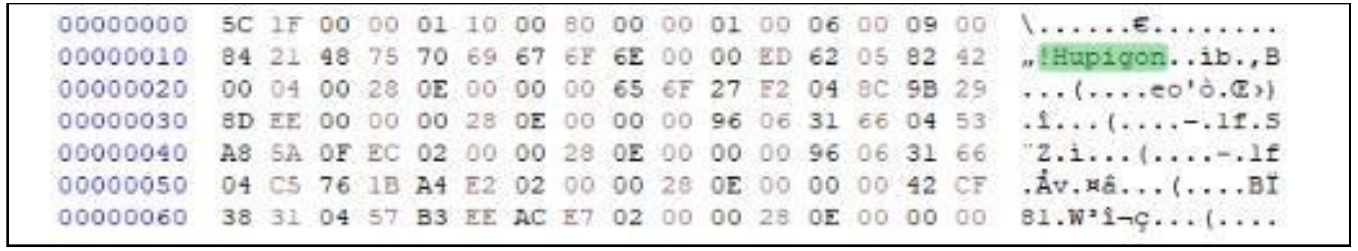
```
import zlib
compressed = open('x.gz', 'rb').read()
decompressed = zlib.decompress(compressed, -zlib.MAX_WBITS)
```

无 gz 文件头

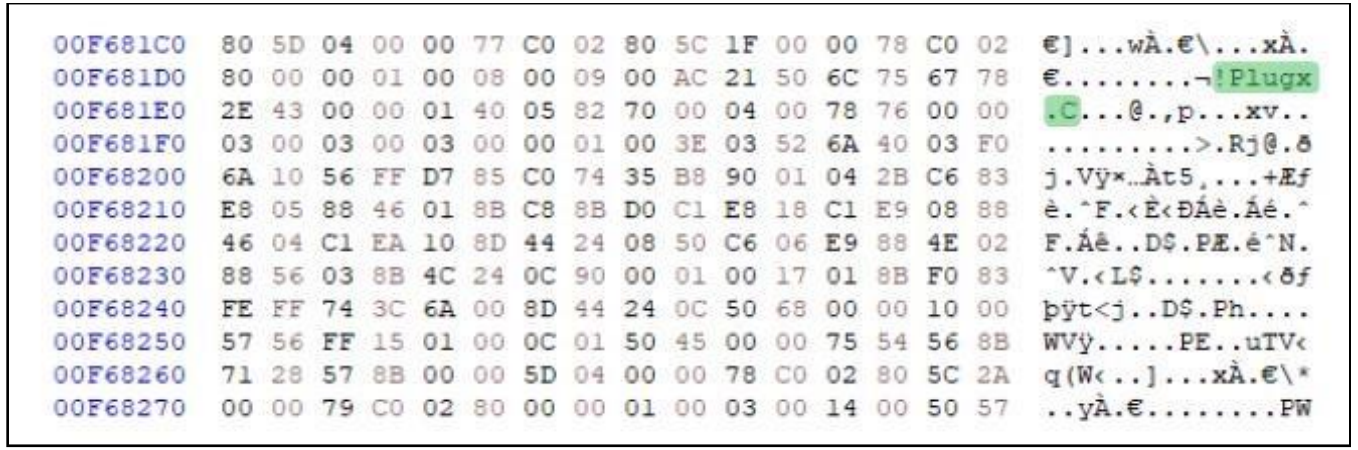
4. 在同目录下运行此 python3 脚本

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

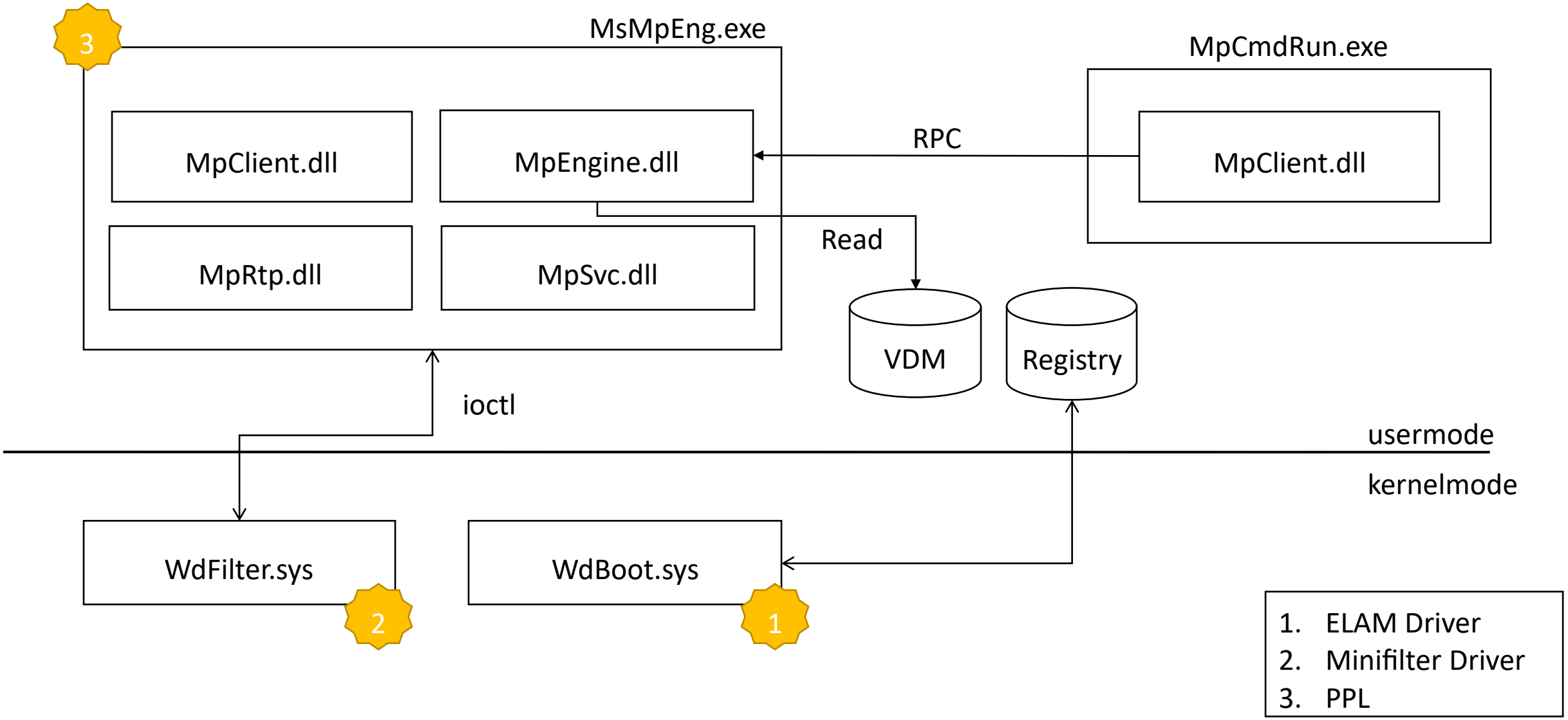


```
00000000  5C 1F 00 00 01 10 00 80 00 00 01 00 06 00 09 00  \......€........
00000010  84 21 48 75 70 69 67 6F 6E 00 00 ED 62 05 82 42  „!Hupigon..íb.,B
00000020  00 04 00 28 0E 00 00 00 65 6F 27 F2 04 8C 9B 29  ...(....eo'ò.Œ›)
00000030  8D EE 00 00 00 28 0E 00 00 00 96 06 31 66 04 53  .î...(....-.1f.S
00000040  A8 5A 0F EC 02 00 00 28 0E 00 00 00 96 06 31 66  ¨Z.ì...(....-.1f
00000050  04 C5 76 1B A4 E2 02 00 00 28 0E 00 00 00 42 CF  .Åv.¤â...(....BÏ
00000060  38 31 04 57 B3 EE AC E7 02 00 00 28 0E 00 00 00  81.W³î¬ç...(....
```



```
00F681C0  80 5D 04 00 00 77 C0 02 80 5C 1F 00 00 78 C0 02  €]...wÀ.€\...xÀ.
00F681D0  80 00 00 01 00 08 00 09 00 AC 21 50 6C 75 67 78  €.........¬!Plugx
00F681E0  2E 43 00 00 01 40 05 82 70 00 04 00 78 76 00 00  .C...@.‚p...xv..
00F681F0  03 00 03 00 03 00 00 01 00 3E 03 52 6A 40 03 F0  .........>.Rj@.ð
00F68200  6A 10 56 FF D7 85 C0 74 35 B8 90 01 04 2B C6 83  j.Vÿ×…Àt5¸...+Æƒ
00F68210  E8 05 88 46 01 8B C8 8B D0 C1 E8 18 C1 E9 08 88  è.ˆF.‹È‹ÐÁè.Áé.ˆ
00F68220  46 04 C1 EA 10 8D 44 24 08 50 C6 06 E9 88 4E 02  F.Áê..D$.PÆ.é^N.
00F68230  88 56 03 8B 4C 24 0C 90 00 01 00 17 01 8B F0 83  ^V.‹L$.......‹ðƒ
00F68240  FE FF 74 3C 6A 00 8D 44 24 0C 50 68 00 00 10 00  þÿt<j..D$.Ph....
00F68250  57 56 FF 15 01 00 0C 01 50 45 00 00 75 54 56 8B  WVÿ.....PE..uTV‹
00F68260  71 28 57 8B 00 00 5D 04 00 00 78 C0 02 80 5C 2A  q(W‹..]...xÀ.€\*
00F68270  00 00 79 C0 02 80 00 00 01 00 03 00 14 00 50 57  ..yÀ.€........PW
```

Microsoft Defender Antivirus 结构（重放）



阶段 1: 签名数据库预加载

ksignal

```
260     case 0x400Bu:
261       modprobe_init_status = modprobe_init(rsignal_code, 0x75A100000i64, (void *)a3);
262       if ( modprobe_init_status )
263         modprobe_cleanup(0i64);
264       return modprobe_init_status;
```

modprobe_init_worker

```
195   StringCchPrintfW(&pGkTab->engine_version, 0x40ui64, L"%hs", "1.1.23100.2009");
351   bDbLoaded = preload_database((wchar_t *)L"mpavdlta.vdm", (__int64)ShaCtx);
352   if ( bDbLoaded || (bDbLoaded = preload_database((wchar_t *)L"mpavbase.vdm", (__int64)ShaCtx)) != 0 )
387     result = preload_database((wchar_t *)L"mpasdlta.vdm", (__int64)ShaCtx);
388     NumberOfBytesWritten = result;
389     if ( (_DWORD)result )
390       goto LABEL_49;
391     result = preload_database((wchar_t *)L"mpasbase.vdm", (__int64)ShaCtx);
392     NumberOfBytesWritten = result;
393     if ( (_DWORD)result )
394       goto LABEL_49;
```

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

签名版本 1.401.1166.0

一旦对签名文件的预处理完成，Defender 模块初始化启动……

阶段 2： 初始化 Defender 各模块

1

init_modules

```
28    *(_QWORD *)gktab->pAutoinitModules = pAutoinitModules;
29    init_failed = AutoInitModules::Initialize(pAutoinitModules);
30    if ( init_failed )
31    {
32      pGktab = gktab;
33      v8 = *(AutoInitModules **)gktab->pAutoinitModules;
34      if ( v8 )
35      {
36        AutoInitModules::`scalar deleting destructor'(v8, v3);
37        pGktab = gktab;
38      }
39      *(_QWORD *)pGktab->pAutoinitModules = 0i64;
40      return init_failed;
41    }
```

3

```
g_pUnimodEntries unimod_entry_t <offset aPrivilegeutils, \
                                 ; DATA XREF: init_modules(void)+70↑o
                        offset ?PrivilegeUtils_init_module@@YA?AW4MP_ERROR@@PEAVAutoInitModules@@@Z,\ ; Privi
                        offset ?PrivilegeUtils_cleanup_module@@YAXXZ, 1>
                unimod_entry_t <offset aDbvars, \ ; dbvars_cleanup_module(void) ...
                        offset ?dbvars_init_module@@YA?AW4MP_ERROR@@PEAVAutoInitModules@@@Z,\
                        offset ?dbvars_cleanup_module@@YAXXZ, 1>
                dq offset aDbload       ; "dbload"
                dq offset ?DbloadInitModule@@YA?AW4MP_ERROR@@PEAVAutoInitModules@@@Z ; DbloadInitModule(AutoInitModules *)
                dq offset ?DbloadCleanupModule@@YAXXZ ; DbloadCleanupModule(void)
                dq 1
```

2

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()

```
68    while ( 1 )
69    {
70      v8 = *((_QWORD *)pCurrAutoinitModule + 5);
71      if ( v8 >= v7 )
72        break;
73      pUnimodEntry = (punimod_entry_t)(*(_QWORD *)pCurrAutoinitModule + 32 * v8);
74      v22 = (__int64 *)pUnimodEntry;
75      v10 = (HANDLE *)WPP_GLOBAL_Control;
76      if ( WPP_GLOBAL_Control != &WPP_GLOBAL_Control && (*((_BYTE *)WPP_GLOBAL_Control + 28) & 8) != 0 )
77      {
78        WPP_SF_Ps(*((_QWORD *)WPP_GLOBAL_Control + 2), 17, v8, v8, (__int64)pUnimodEntry->pModuleName);
79        v10 = (HANDLE *)WPP_GLOBAL_Control;
80      }
81      if ( !g_InsideSandbox || LOBYTE(pUnimodEntry->Unk) )
82      {
83        v11 = ((__int64 (__fastcall *)(AutoInitModules *))pUnimodEntry->pfnInit)(pCurrAutoinitModule);
84        v14 = v11;
```

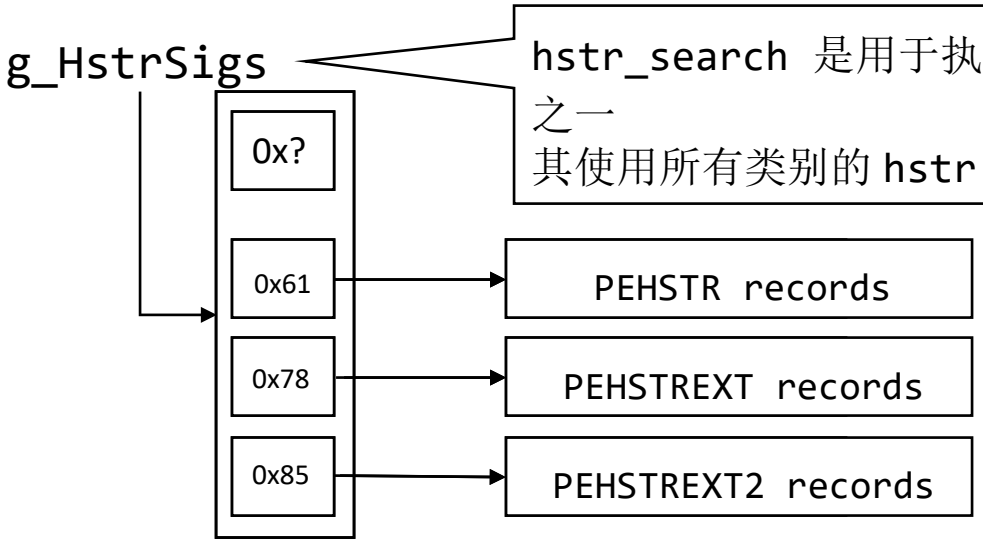
cksig_init_module

pattsearch_init

- 调用pattsearch_init 初始化包含签名的数据结构
  包括g_HstrSigs和 g_DynamicHstrSigs

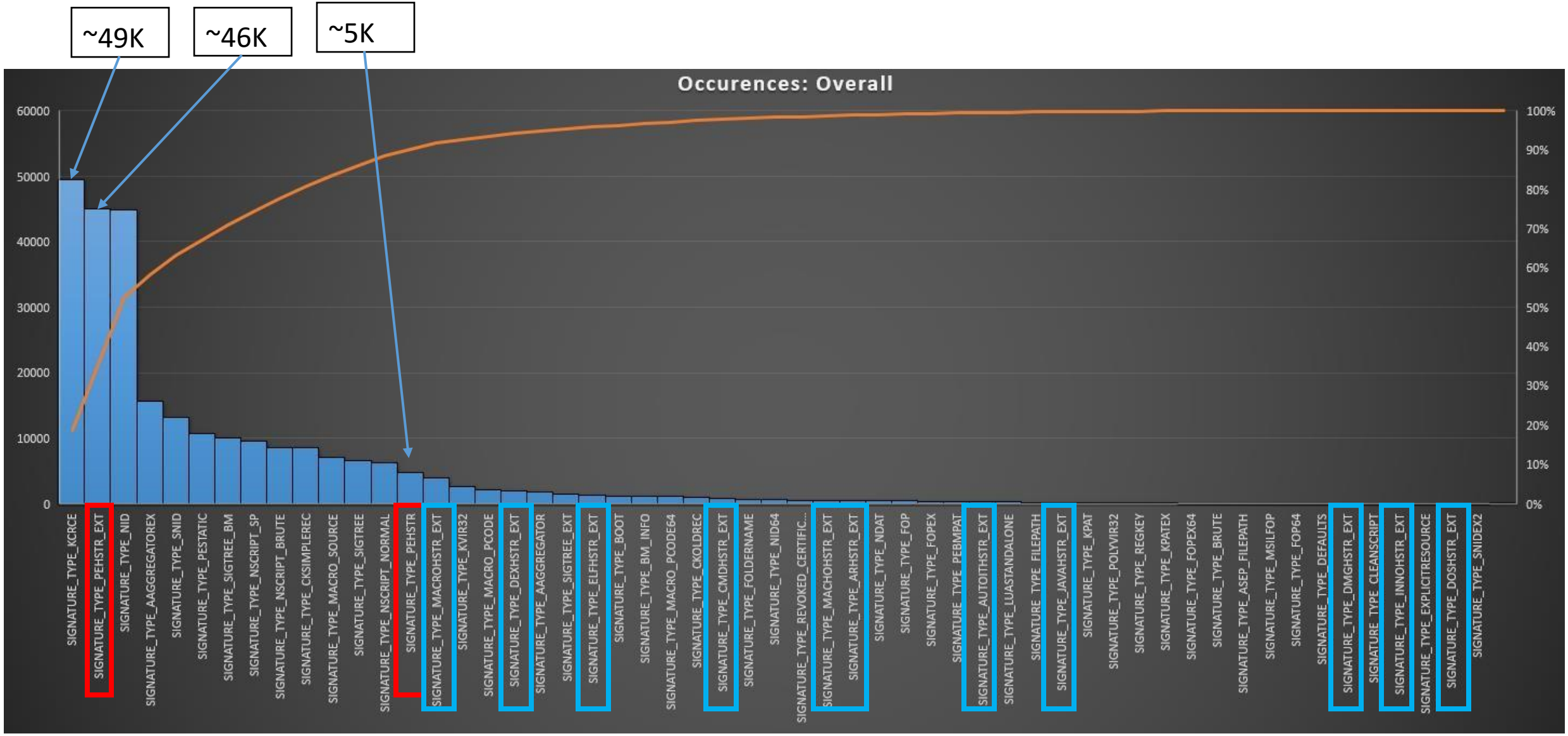
- 这些符号是指向一个哈希表的指针，包含所有文件类型（原文 HSTR）的指针（例如 ELF PE MACOS 等）

- load_database/load_database_cache会
  DispatchRecords（分配上述记录）到正确的存储栈

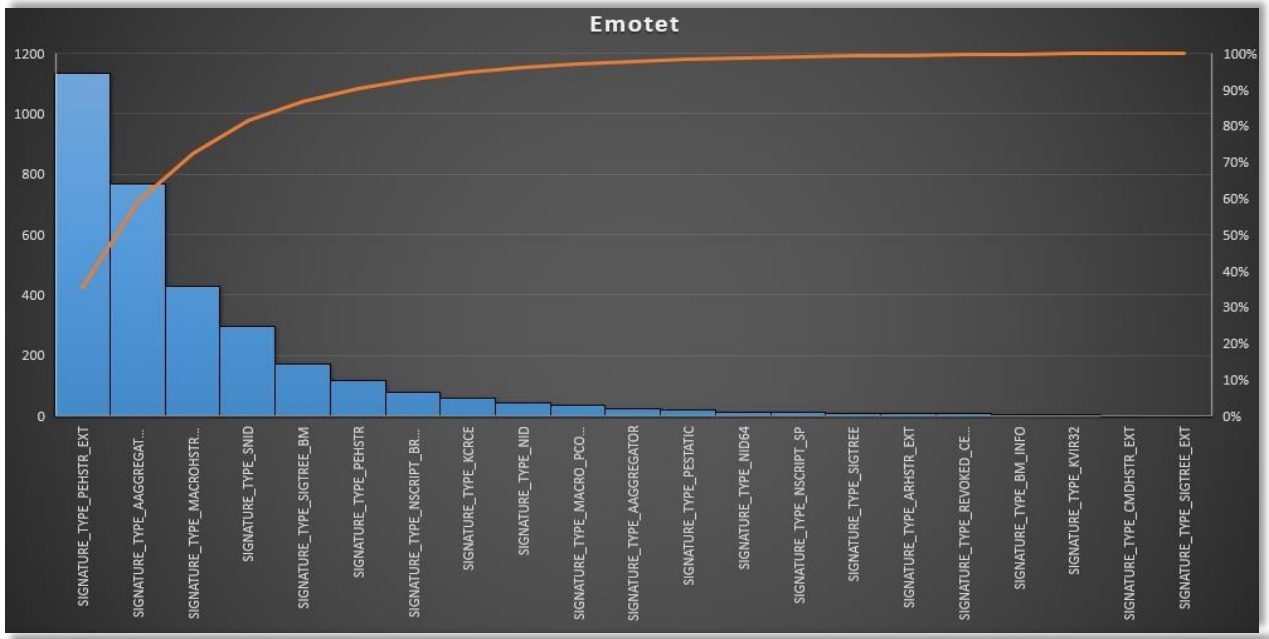


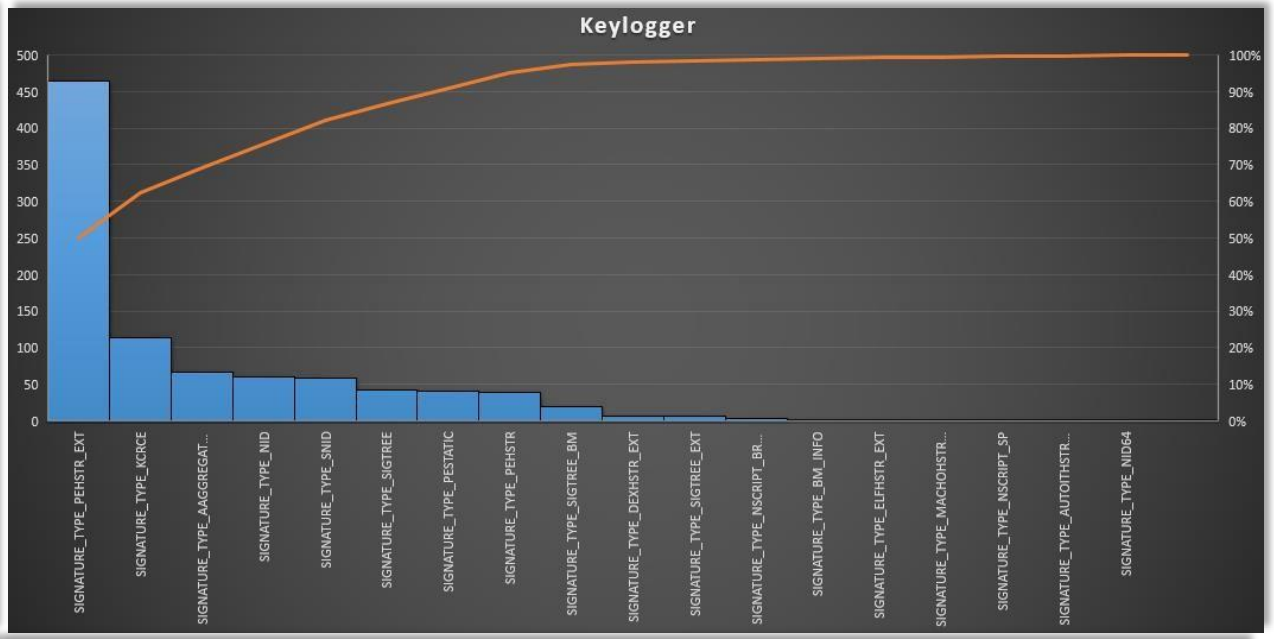

```
137     pehstr_record_cnt = ESTIMATED_RECORDS(0x61);
138     pehstr_ext_record_cnt = ESTIMATED_RECORDS(0x78);
139     pehstr_ext2_record_cnt = ESTIMATED_RECORDS(0x85);
140     if ( pehstr_ext_record_cnt + pehstr_record_cnt < pehstr_record_cnt
141       || (pe_hstr_total_cnt = pehstr_record_cnt + pehstr_ext_record_cnt + pehstr_ext2_record_cnt,
142           (unsigned int)pe_hstr_total_cnt < pehstr_ext_record_cnt + pehstr_record_cnt) )
143     {
144       v0 = 32780;
145       goto out_1;
146     }
147     g_pe_hstr_total_cnt = pehstr_record_cnt + pehstr_ext_record_cnt + pehstr_ext2_record_cnt;
148     g_p_pehstr_total = (__int64)calloc(pe_hstr_total_cnt, 0x14ui64);
149     if ( !g_p_pehstr_total )
150       goto out;
151     byte_75B1A7F70 = 0;
152     curr_handler.pfn_push = (UINT64)hstr_push;
153     curr_handler.hstr_type = 0x61;
154     curr_handler.pfn_pushend = (__int64 (__fastcall *)())hstr_pushend_common;
155     p_gHstrSigs = (char *)&g_HstrSigs;
    v0 = regcntl(&curr_handler, 0x30ui64, 0xC);
    if ( v0 )
      goto out_1;
    curr_handler.pfn_push = (UINT64)hstr_push_ext;
    curr_handler.hstr_type = 0x78;
    curr_handler.pfn_pushend = (__int64 (__fastcall *)())hstr_pushend_common;
    p_gHstrSigs = (char *)&g_HstrSigs;
    v0 = regcntl(&curr_handler, 0x30ui64, 0xC);
    if ( v0 )
      goto out_1;
    curr_handler.pfn_push = (UINT64)hstr_push_ext2;
    curr_handler.hstr_type = 0x85;
    curr_handler.pfn_pushend = (__int64 (__fastcall *)())hstr_pushend_common;
    p_gHstrSigs = (char *)&g_HstrSigs;
    v0 = regcntl(&curr_handler, 0x30ui64, 0xC);
    if ( v0 )
      goto out_1;
```

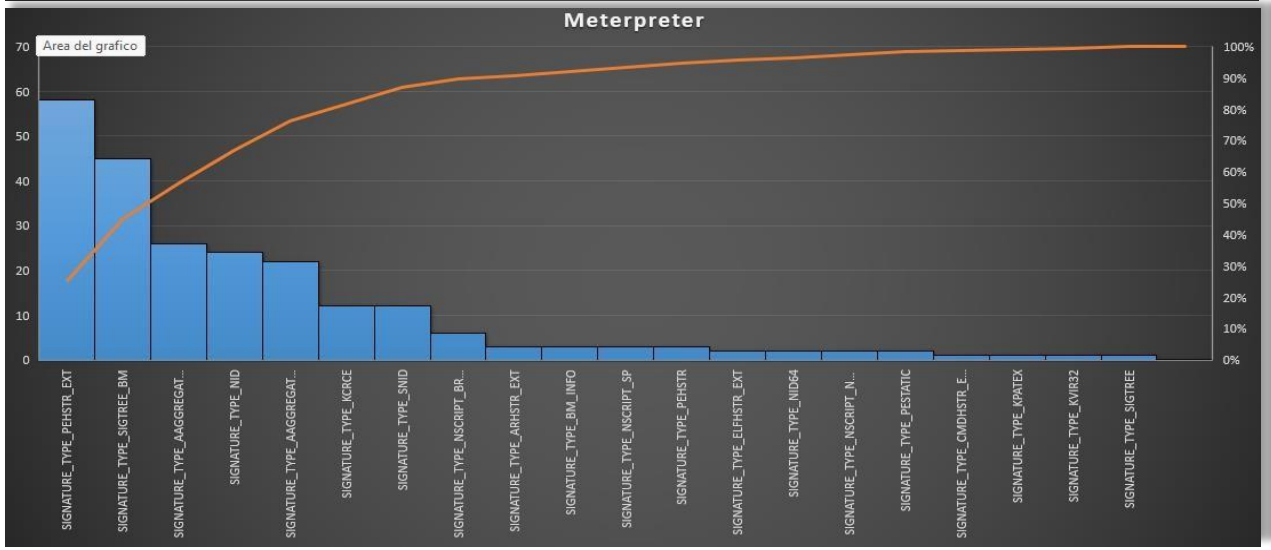
数字和统计：出现次数

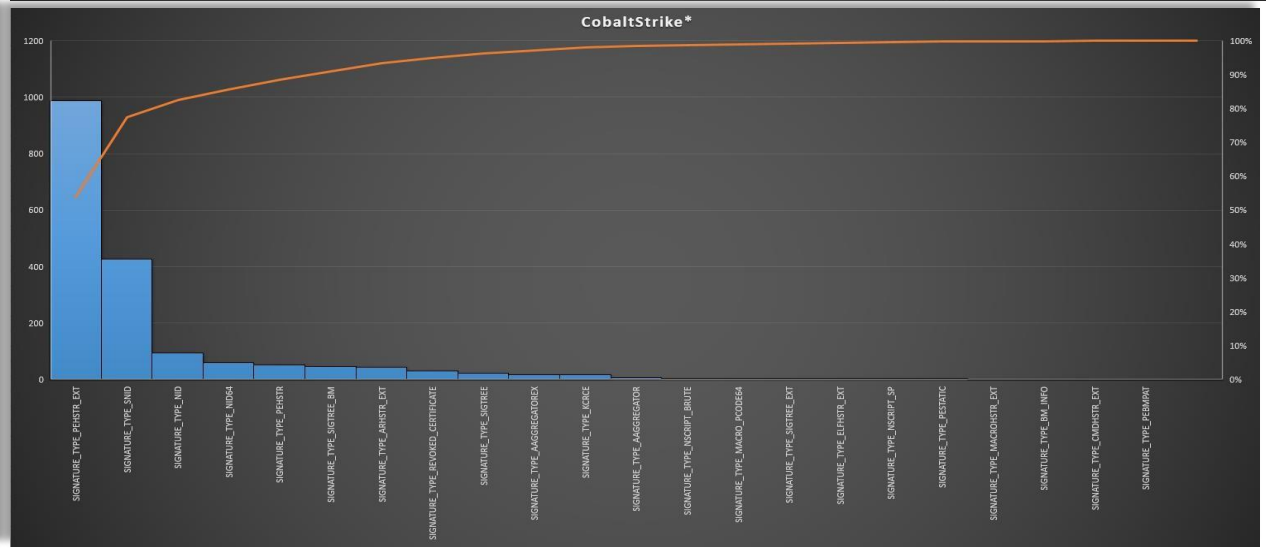

~49K
~46K
~5K
Occurences: Overall
60000
50000
40000
30000
20000
10000
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
SIGNATURE_TYPE_KCRCE
SIGNATURE_TYPE_PEHSTR_EXT
SIGNATURE_TYPE_NID
SIGNATURE_TYPE_AAGGREGATOREX
SIGNATURE_TYPE_SNID
SIGNATURE_TYPE_PESTATIC
SIGNATURE_TYPE_SIGTREE_BM
SIGNATURE_TYPE_NSCRIPT_SP
SIGNATURE_TYPE_NSCRIPT_BRUTE
SIGNATURE_TYPE_CKSIMPLEREC
SIGNATURE_TYPE_MACRO_SOURCE
SIGNATURE_TYPE_SIGTREE
SIGNATURE_TYPE_NSCRIPT_NORMAL
SIGNATURE_TYPE_PEHSTR
SIGNATURE_TYPE_MACROHSTR_EXT
SIGNATURE_TYPE_KVIR32
SIGNATURE_TYPE_MACRO_PCODE
SIGNATURE_TYPE_DEXHSTR_EXT
SIGNATURE_TYPE_AAGGREGATOR
SIGNATURE_TYPE_SIGTREE_EXT
SIGNATURE_TYPE_ELFHSTR_EXT
SIGNATURE_TYPE_BOOT
SIGNATURE_TYPE_BM_INFO
SIGNATURE_TYPE_MACRO_PCODE64
SIGNATURE_TYPE_CKOLDREC
SIGNATURE_TYPE_CMDHSTR_EXT
SIGNATURE_TYPE_FOLDERNAME
SIGNATURE_TYPE_NID64
SIGNATURE_TYPE_REVOKED_CERTIFIC...
SIGNATURE_TYPE_MACHOHSTR_EXT
SIGNATURE_TYPE_ARHSTR_EXT
SIGNATURE_TYPE_NDAT
SIGNATURE_TYPE_FOP
SIGNATURE_TYPE_FOPEX
SIGNATURE_TYPE_PEBMPAT
SIGNATURE_TYPE_AUTOITHSTR_EXT
SIGNATURE_TYPE_LUASTANDALONE
SIGNATURE_TYPE_FILEPATH
SIGNATURE_TYPE_JAVAHSTR_EXT
SIGNATURE_TYPE_KPAT
SIGNATURE_TYPE_POLYVIR32
SIGNATURE_TYPE_REGKEY
SIGNATURE_TYPE_KPATEX
SIGNATURE_TYPE_FOPEX64
SIGNATURE_TYPE_BRUTE
SIGNATURE_TYPE_ASEP_FILEPATH
SIGNATURE_TYPE_MSILFOP
SIGNATURE_TYPE_FOP64
SIGNATURE_TYPE_DEFAULTS
SIGNATURE_TYPE_DMGHSTR_EXT
SIGNATURE_TYPE_CLEANSCRIPT
SIGNATURE_TYPE_INNOHSTR_EXT
SIGNATURE_TYPE_EXPLICITRESOURCE
SIGNATURE_TYPE_DOSHSTR_EXT
SIGNATURE_TYPE_SNIDEX2

对特定威胁进行切分

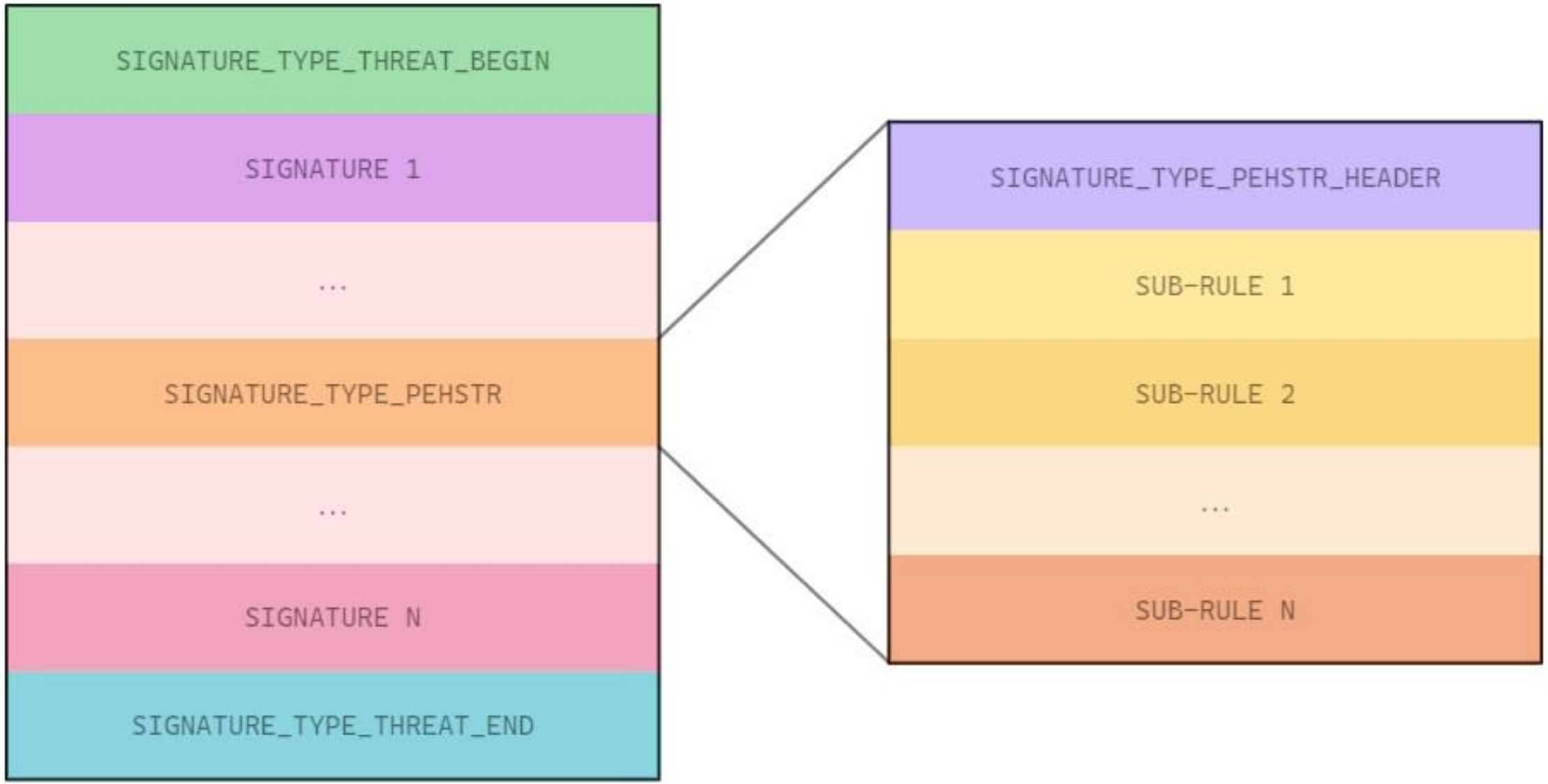

Emotet
1200
1000
800
600
400
200
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
SIGNATURE_TYPE_PEHSTR_EXT
SIGNATURE_TYPE_AAGGREGAT...
SIGNATURE_TYPE_MACROHSTR...
SIGNATURE_TYPE_SNID
SIGNATURE_TYPE_SIGTREE_BM
SIGNATURE_TYPE_PEHSTR
SIGNATURE_TYPE_NSCRIPT_BR...
SIGNATURE_TYPE_KCRCE
SIGNATURE_TYPE_NID
SIGNATURE_TYPE_MACRO_PCO...
SIGNATURE_TYPE_AAGGREGATOR
SIGNATURE_TYPE_PESTATIC
SIGNATURE_TYPE_NID64
SIGNATURE_TYPE_NSCRIPT_SP
SIGNATURE_TYPE_SIGTREE
SIGNATURE_TYPE_ARHSTR_EXT
SIGNATURE_TYPE_REVOKED_CE...
SIGNATURE_TYPE_BM_INFO
SIGNATURE_TYPE_KVIR32
SIGNATURE_TYPE_CMDHSTR_EXT
SIGNATURE_TYPE_SIGTREE_EXT


Keylogger
500
450
400
350
300
250
200
150
100
50
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
SIGNATURE_TYPE_PEHSTR_EXT
SIGNATURE_TYPE_KCRCE
SIGNATURE_TYPE_AAGGREGAT...
SIGNATURE_TYPE_NID
SIGNATURE_TYPE_SNID
SIGNATURE_TYPE_SIGTREE
SIGNATURE_TYPE_PESTATIC
SIGNATURE_TYPE_PEHSTR
SIGNATURE_TYPE_SIGTREE_BM
SIGNATURE_TYPE_DEXHSTR_EXT
SIGNATURE_TYPE_SIGTREE_EXT
SIGNATURE_TYPE_NSCRIPT_BR...
SIGNATURE_TYPE_BM_INFO
SIGNATURE_TYPE_ELFHSTR_EXT
SIGNATURE_TYPE_MACHOHSTR...
SIGNATURE_TYPE_NSCRIPT_SP
SIGNATURE_TYPE_AUTOITHSTR...
SIGNATURE_TYPE_NID64


Meterpreter
Area del grafico
70
60
50
40
30
20
10
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
SIGNATURE_TYPE_PEHSTR_EXT
SIGNATURE_TYPE_SIGTREE_BM
SIGNATURE_TYPE_AAGGREGAT...
SIGNATURE_TYPE_NID
SIGNATURE_TYPE_AAGGREGAT...
SIGNATURE_TYPE_KCRCE
SIGNATURE_TYPE_SNID
SIGNATURE_TYPE_NSCRIPT_BR...
SIGNATURE_TYPE_ARHSTR_EXT
SIGNATURE_TYPE_BM_INFO
SIGNATURE_TYPE_NSCRIPT_SP
SIGNATURE_TYPE_PEHSTR
SIGNATURE_TYPE_ELFHSTR_EXT
SIGNATURE_TYPE_NID64
SIGNATURE_TYPE_NSCRIPT_N...
SIGNATURE_TYPE_PESTATIC
SIGNATURE_TYPE_CMDHSTR_E...
SIGNATURE_TYPE_KPATEK
SIGNATURE_TYPE_KVIR32
SIGNATURE_TYPE_SIGTREE


CobaltStrike*
1200
1000
800
600
400
200
0
100%
90%
80%
70%
60%
50%
40%
30%
20%
10%
0%
SIGNATURE_TYPE_PEHSTR_EXT
SIGNATURE_TYPE_SNID
SIGNATURE_TYPE_NID
SIGNATURE_TYPE_NID64
SIGNATURE_TYPE_PEHSTR
SIGNATURE_TYPE_SIGTREE_BM
SIGNATURE_TYPE_ARHSTR_EXT
SIGNATURE_TYPE_REVOKED_CERTIFICATE
SIGNATURE_TYPE_SIGTREE
SIGNATURE_TYPE_AAGGREGATORDEX
SIGNATURE_TYPE_KCRCE
SIGNATURE_TYPE_AAGGREGATOR
SIGNATURE_TYPE_NSCRIPT_BRUTE
SIGNATURE_TYPE_MACRO_PCODE64
SIGNATURE_TYPE_SIGTREE_EXT
SIGNATURE_TYPE_ELFHSTR_EXT
SIGNATURE_TYPE_NSCRIPT_SP
SIGNATURE_TYPE_PESTATIC
SIGNATURE_TYPE_MACROHSTR_EXT
SIGNATURE_TYPE_BM_INFO
SIGNATURE_TYPE_CMDHSTR_EXT
SIGNATURE_TYPE_PEMPAT

签名通用结构


SIGNATURE_TYPE_THREAT_BEGIN
SIGNATURE 1
...
SIGNATURE_TYPE_PEHSTR
...
SIGNATURE N
SIGNATURE_TYPE_THREAT_END
SIGNATURE_TYPE_PEHSTR_HEADER
SUB-RULE 1
SUB-RULE 2
...
SUB-RULE N

开始到结束

SIGNATURE_TYPE_THREAT_BEGIN 和

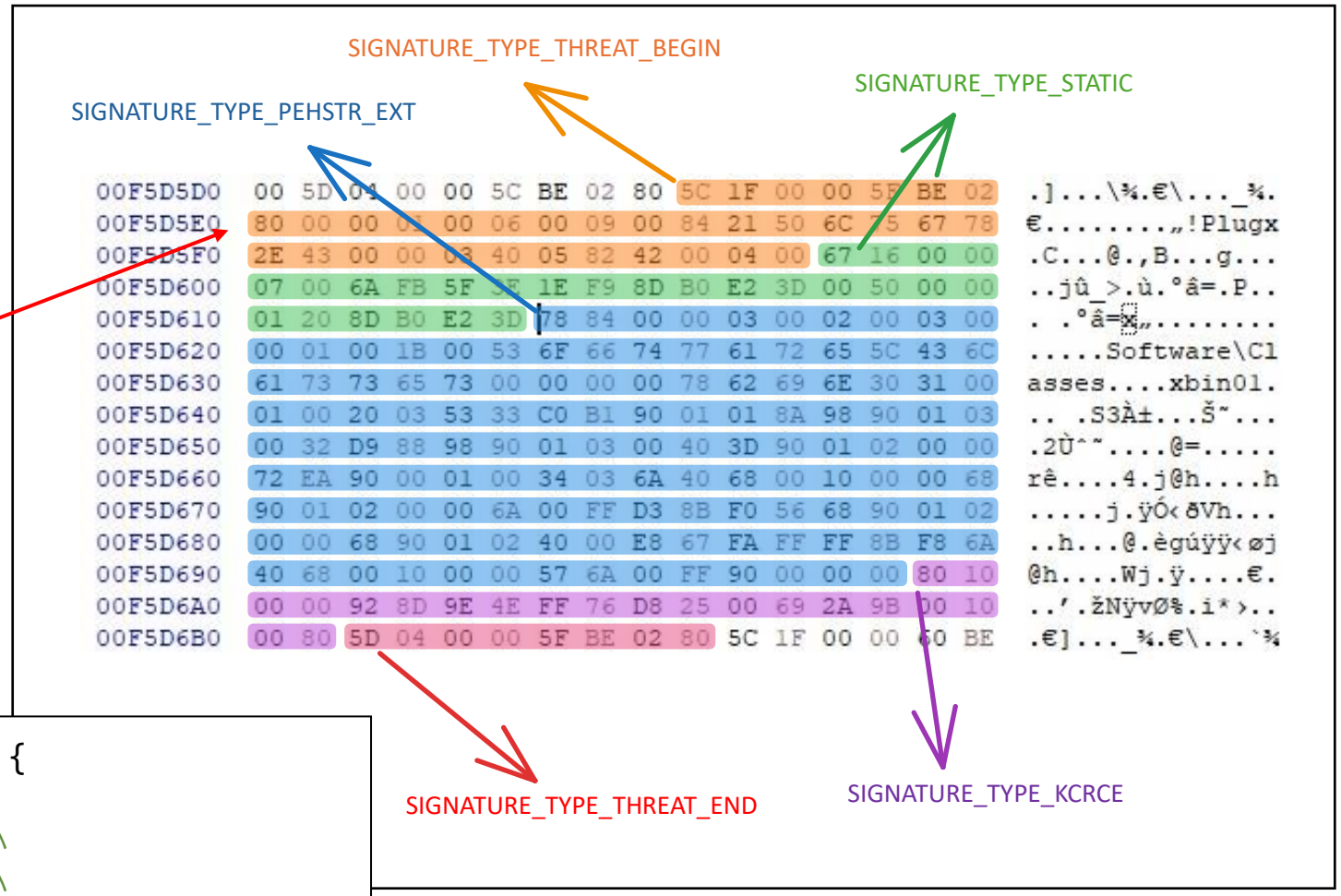
SIGNATURE_TYPE_THREAT_END

包含自定义数据

- 其中之一是四字节的规则 ID （示例：0x8002be5f）

createrecid

```
81    if ( ThreadId >= 0x80000000 )
82      v12 = AV_AutoGenThreatID++;
83    else
84      v12 = AS_AutoGenThreatID++;
```




```
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       lpszThreatName[ui8SizeThreatName];
    BYTE       unknownBytes3[9];
} STRUCT_SIG_TYPE_THREAT_BEGIN,
* PSTRUCT_SIG_TYPE_THREAT_BEGIN;
```

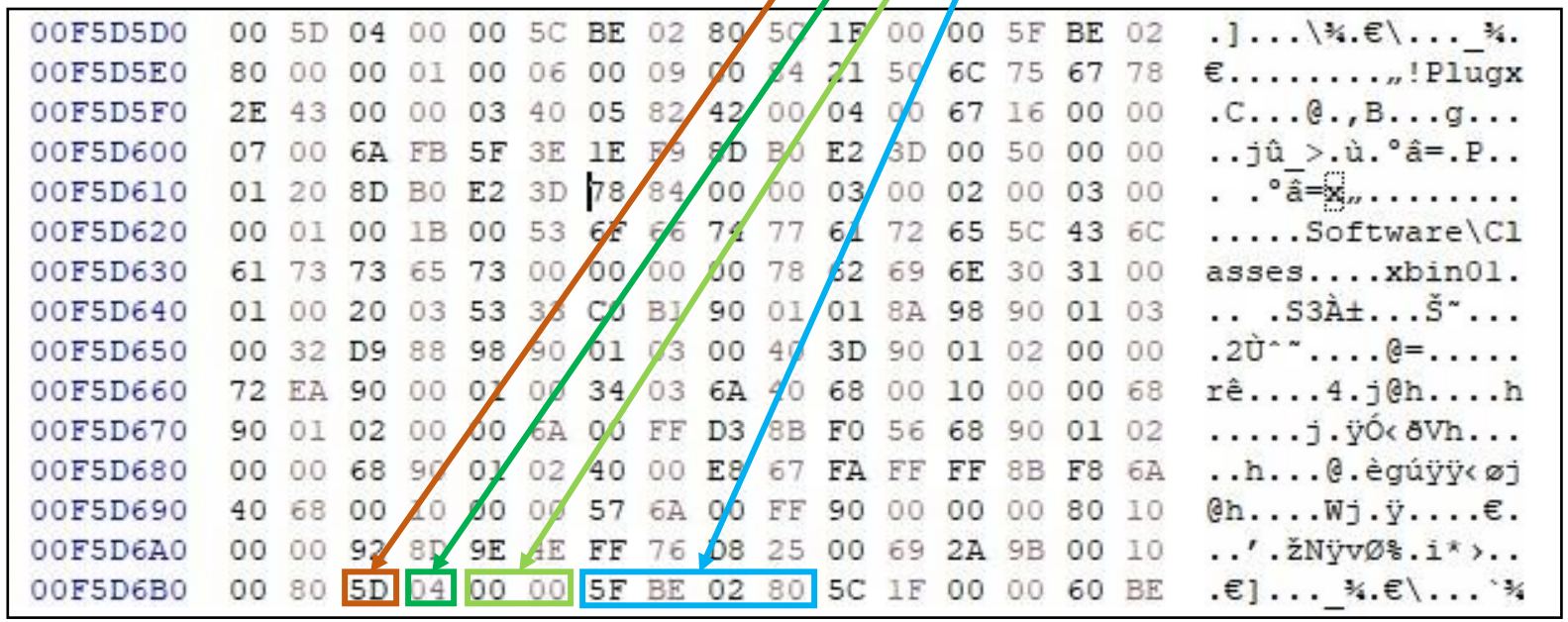
```
00F5D5D0  00 5D 04 00 00 5C BE 02 80 5C 1F 00 00 5F BE 02  .]...\¾.€\..._¾.
00F5D5E0  80 00 00 01 00 06 00 09 00 84 21 50 6C 75 67 78  €.........„!Plugx
00F5D5F0  2E 43 00 00 03 40 05 82 42 00 04 00 67 16 00 00  .C...@.‚B...g...
00F5D600  07 00 6A FB 5F 3E 1E F9 8D B0 E2 3D 00 50 00 00  ..jû_>.ù.°â=.P..
00F5D610  01 20 8D B0 E2 3D 78 84 00 00 03 00 02 00 03 00  . .°â=x„........
00F5D620  00 01 00 1B 00 53 6F 66 74 77 61 72 65 5C 43 6C  .....Software\Cl
00F5D630  61 73 73 65 73 00 00 00 00 78 62 69 6E 30 31 00  asses....xbin01.
00F5D640  01 00 20 03 53 33 C0 B1 90 01 01 8A 98 90 01 03  .. .S3À±...Š˜...
00F5D650  00 32 D9 88 98 90 01 03 00 40 3D 90 01 02 00 00  .2Ù^˜....@=.....
00F5D660  72 EA 90 00 01 00 34 03 6A 40 68 00 10 00 00 68  rê....4.j@h....h
00F5D670  90 01 02 00 00 6A 00 FF D3 8B F0 56 68 90 01 02  .....j.ÿÓ‹ðVh...
00F5D680  00 00 68 90 01 02 40 00 E8 67 FA FF FF 8B F8 6A  ..h...@.ègúÿÿ‹øj
00F5D690  40 68 00 10 00 00 57 6A 00 FF 90 00 00 00 80 10  @h....Wj.ÿ....€.
00F5D6A0  00 00 92 8D 9E 4E FF 76 D8 25 00 69 2A 9B 00 10  ..'.žNÿvØ%.i*›..
00F5D6B0  00 80 5D 04 00 00 5F BE 02 80 5C 1F 00 00 60 BE  .€]..._¾.€\...`¾
```

SIGNATURE_TYPE_THREAT_END

-
- 定义威胁的终点

  代码标识符：0x5D
- pbRuleContent 值和 SIGNATURE_TYPE_THREAT_BEGIN 中使用的相应 ui32SignatureId 值相同

```
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;
```

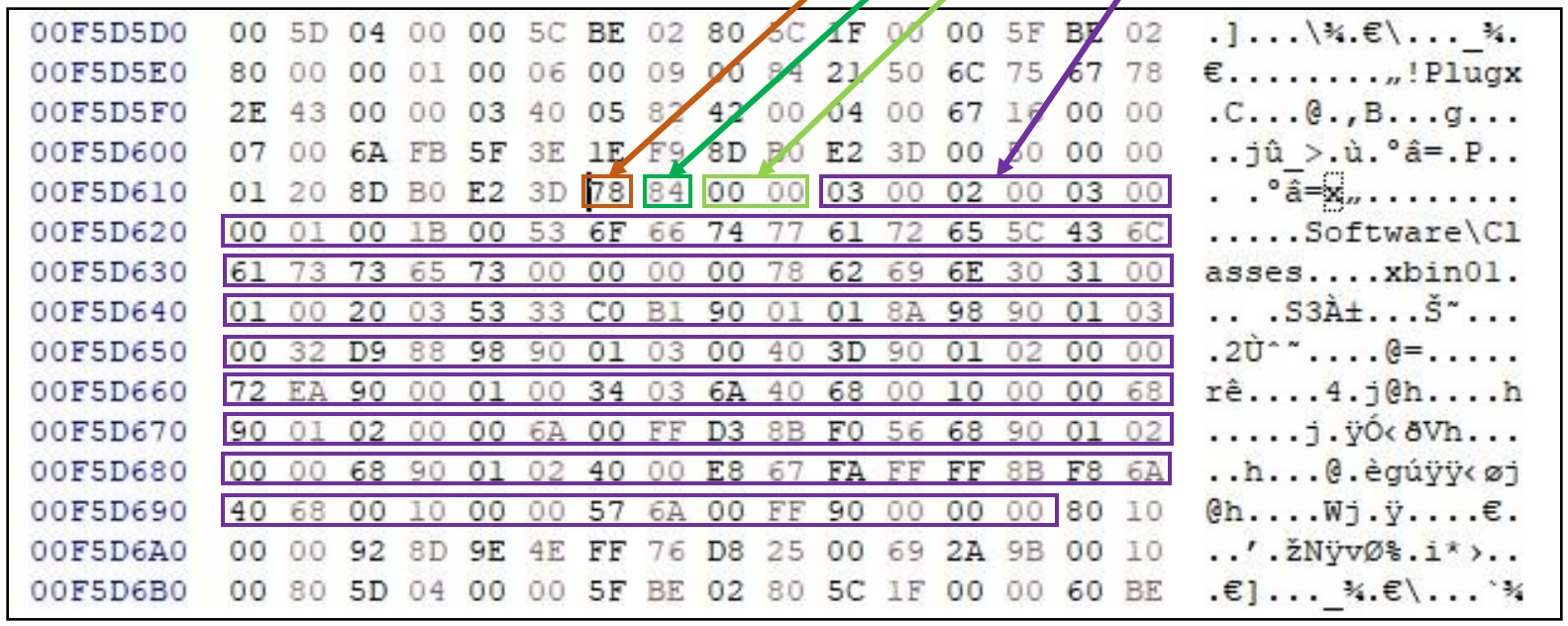


SIGNATURE_TYPE_PEHSTR vs SIGNATURE_TYPE_PEHSTR_EXT

- SIGNATURE_TYPE_PEHSTR
  用于实行对 PE 文件的字符串匹配

```
typedef struct _STRUCT_COMMON_SIGNATURE_TYPE {
        UINT8  ui8SignatureType;
        UINT8  ui8SizeLow;
        UINT16 ui16SizeHigh;
        BYTE   pbRuleContent[];
} STRUCT_COMMON_SIGNATURE_TYPE,
*PSTRUCT_COMMON_SIGNATURE_TYPE;
```

- 代码标识符： 0x61



- SIGNATURE_TYPE_PEHSTR_EXT
  用于匹配 PE 文件的字节

- 代码标识符： 0x78

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

```
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;
```

```
00F5D5D0  00 5D 04 00 00 5C BE 02 80 5C 1F 00 00 5F BE 02   .]...\¾.€\..._¾.
00F5D5E0  80 00 00 01 00 06 00 09 00 84 21 50 6C 75 67 78   €.........„!Plugx
00F5D5F0  2E 43 00 00 03 40 05 82 42 00 04 00 67 16 00 00   .C...@..B...g...
00F5D600  07 00 6A FB 5F 3E 1E F9 8D B0 E2 3D 00 50 00 00   ..jû_>.ù.°â=.P..
00F5D610  01 20 8D B0 E2 3D 78 84 00 00 03 00 02 00 03 00   . .°â=x„........
00F5D620  00 01 00 1B 00 53 6F 66 74 77 61 72 65 5C 43 6C   .....Software\Cl
00F5D630  61 73 73 65 73 00 00 00 00 78 62 69 6E 30 31 00   asses....xbin01.
00F5D640  01 00 20 03 53 33 C0 B1 90 01 01 8A 98 90 01 03   .. .S3À±...Š˜...
00F5D650  00 32 D9 88 98 90 01 03 00 40 3D 90 01 02 00 00   .2Ù^˜....@=.....
00F5D660  72 EA 90 00 01 00 34 03 6A 40 68 00 10 00 00 68   rê....4.j@h....h
00F5D670  90 01 02 00 00 6A 00 FF D3 8B F0 56 68 90 01 02   .....j.ÿÓ‹ðVh...
00F5D680  00 00 68 90 01 02 40 00 E8 67 FA FF FF 8B F8 6A   ..h...@.ègúÿÿ‹øj
00F5D690  40 68 00 10 00 00 57 6A 00 FF 90 00 00 00 80 10   @h....Wj.ÿ....€.
00F5D6A0  00 00 92 8D 9E 4E FF 76 D8 25 00 69 2A 9B 00 10   ..'.žNÿvØ%.i*›..
00F5D6B0  00 80 5D 04 00 00 5F BE 02 80 5C 1F 00 00 5F BE   .€]..._¾.€\..._¾
```

SIGNATURE_TYPE_PEHSTR_EXT

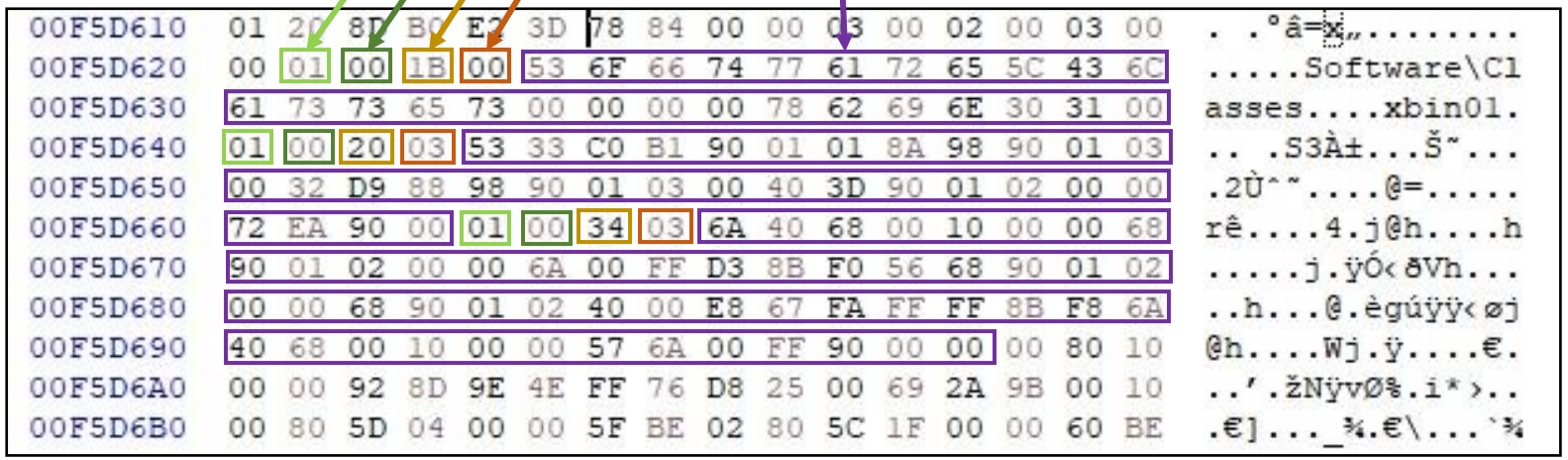
- 所有类型的签名共享同种结构
- 主要区别在于子规则格式略有不同
- SIGNATURE_TYPE_PEHSTR
  用于检测“可读字符串”
- SIGNATURE_TYPE_PEHSTR_EXT
  可用于检测操作码并拥有额外特殊功能

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



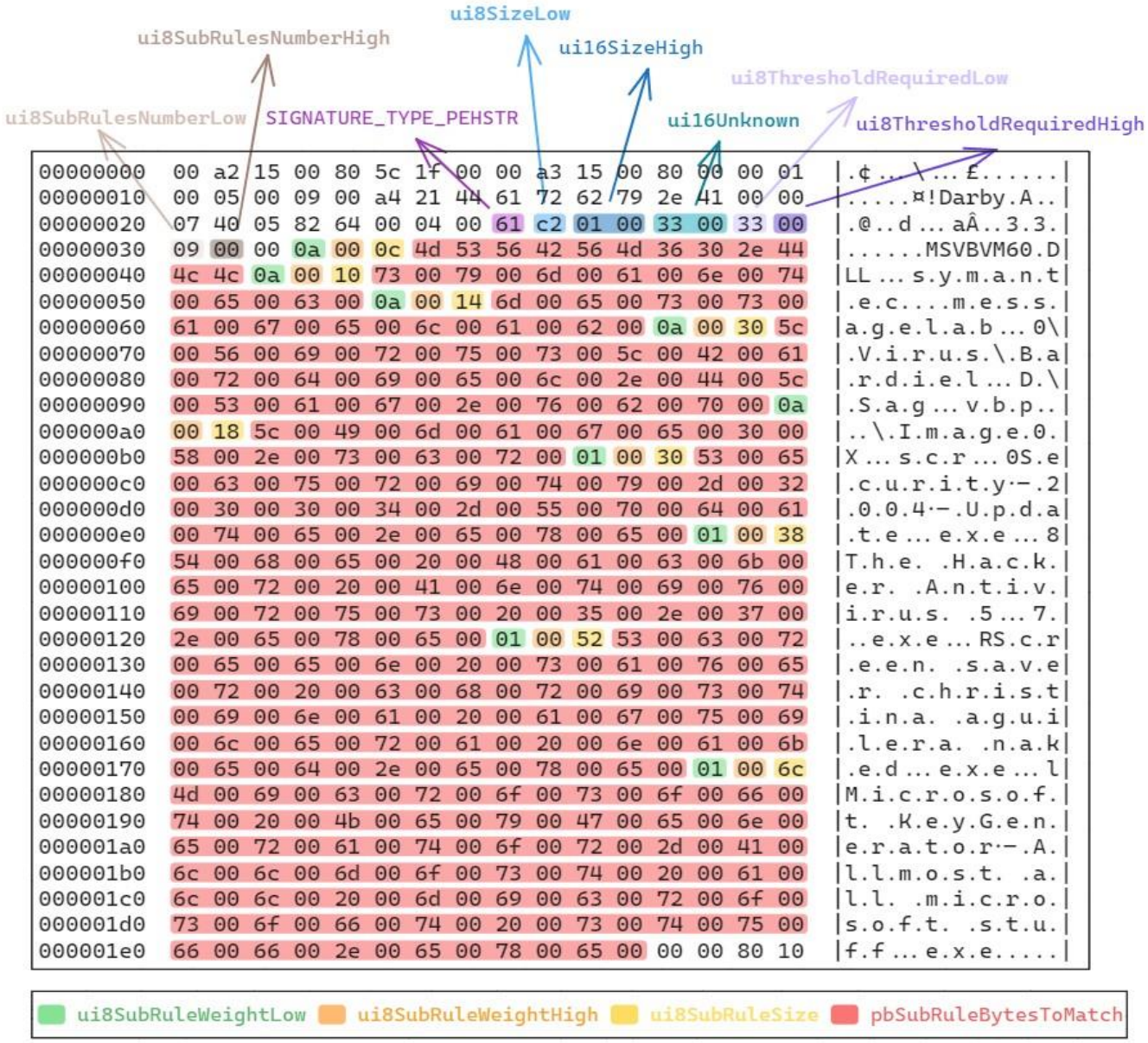
实验 1: SIGNATURE_TYPE_PEHSTR

- 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

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.

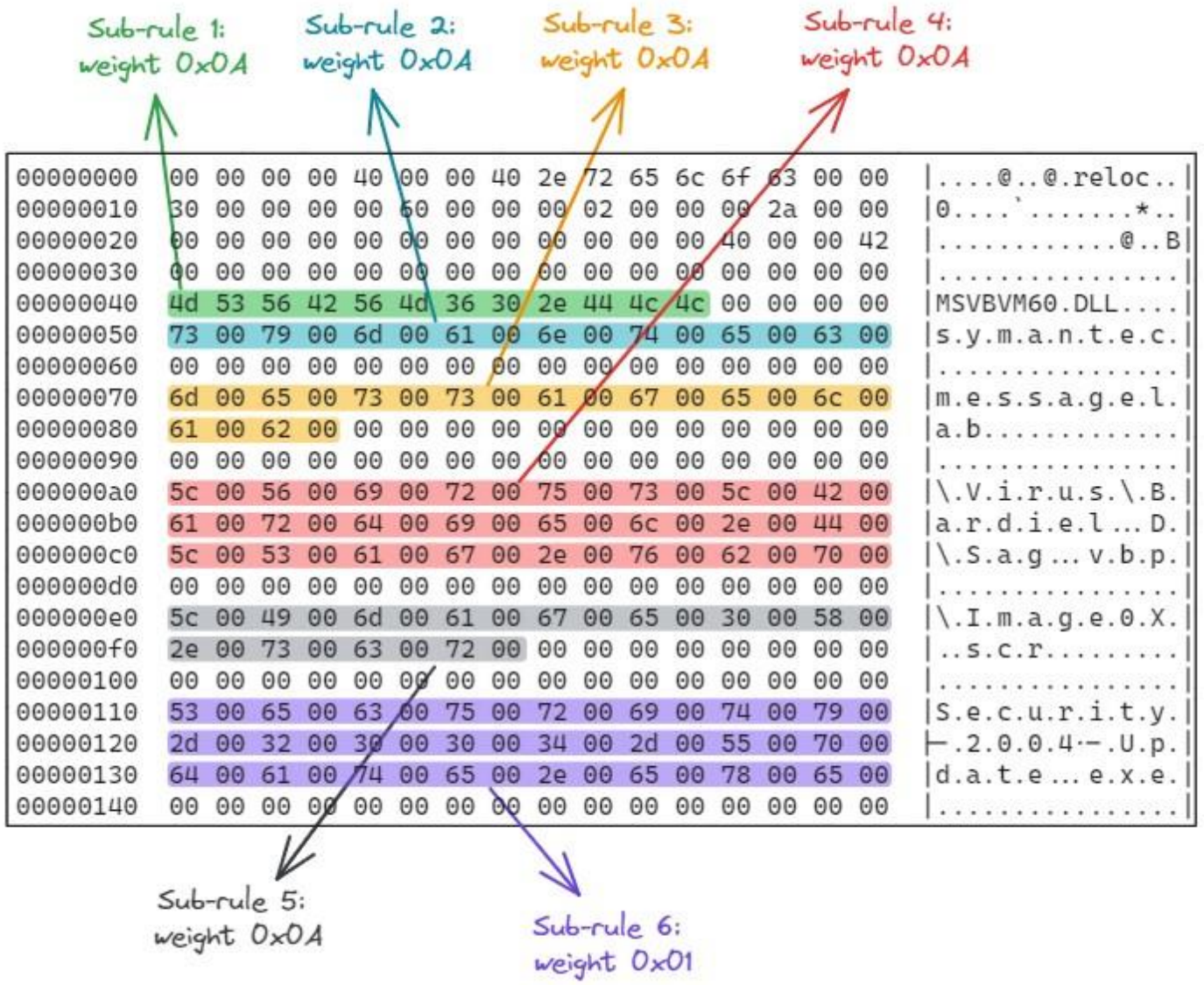
Sub-rule 3: 权重 0x0A.

Sub-rule 4: 权重 0x0A.

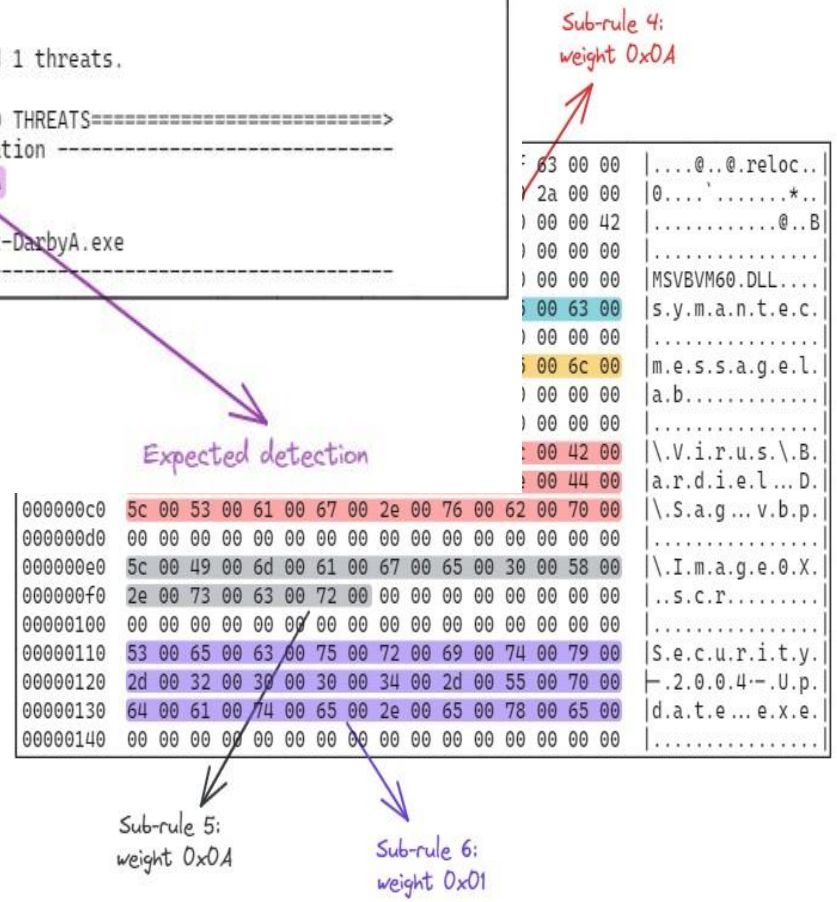
Sub-rule 5: 权重 0x0A.

Sub-rule 6: 权重 0x01.

总计：0x33



```
PS C:\Program Files\Windows Defender> .\MpCmdRun.exe -Scan -ScanType 3 -File
'C:\Users\user\test-DarbyA.exe' -DisableRemediation -Trace -Level 0x10
Scan starting...
Scan finished.
Scanning C:\Users\user\test-DarbyA.exe found 1 threats.

<==========================LIST OF DETECTED THREATS==========================>
---------------------------- Threat information -----------------------------
Threat                  : Worm:Win32/Darby.A
Resources               : 1 total
    file                : C:\Users\user\test-DarbyA.exe
-----------------------------------------------------------------------------
```




```
PS C:\Program Files\Windows Defender> .\MpCmdRun.exe
–Scan –ScanType 3 –File <filepath> -
DisableRemediation –Trace –Level 0x10
```

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

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?

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：

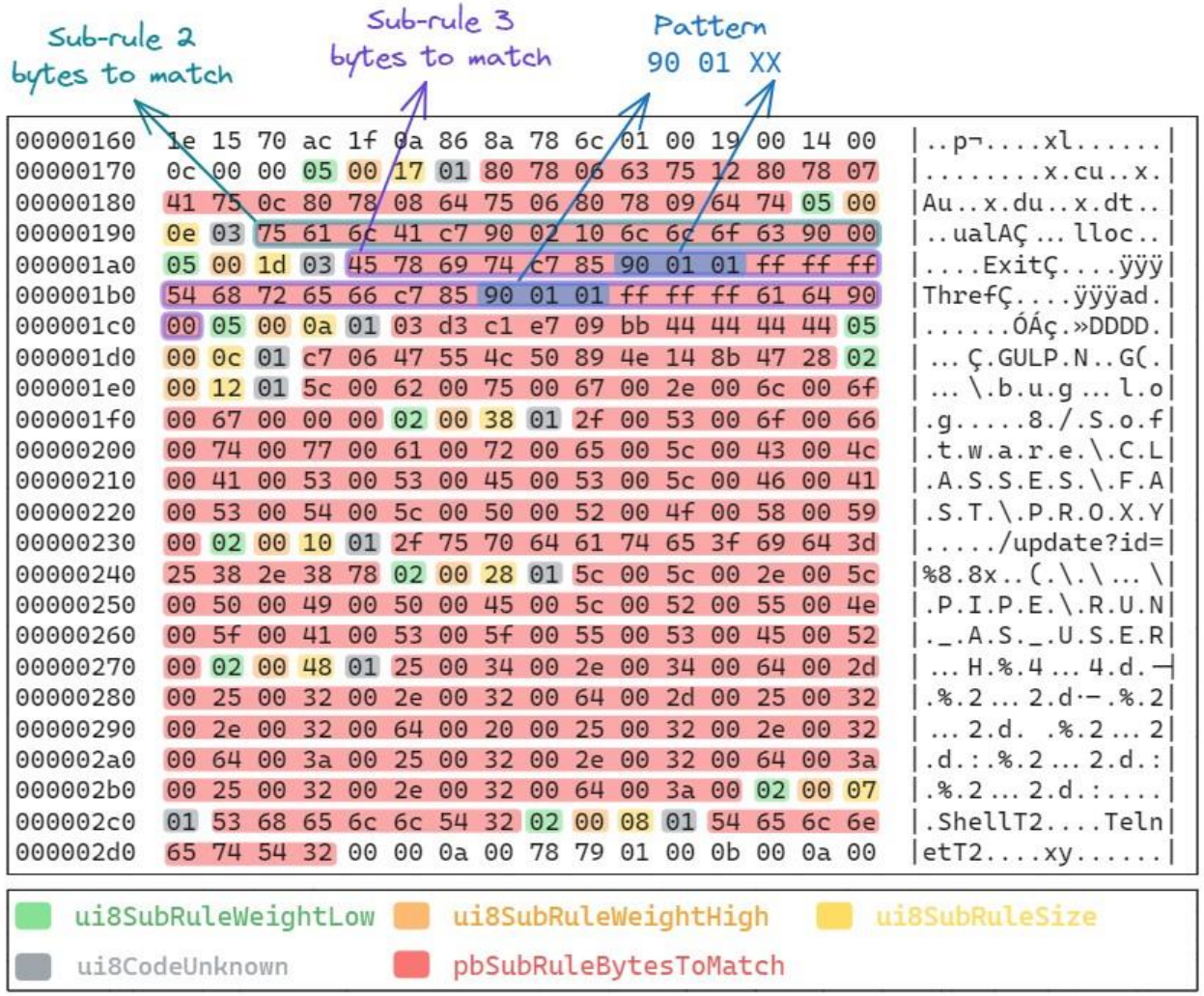
  ```
  90 06 -> 90 20
  ```

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):

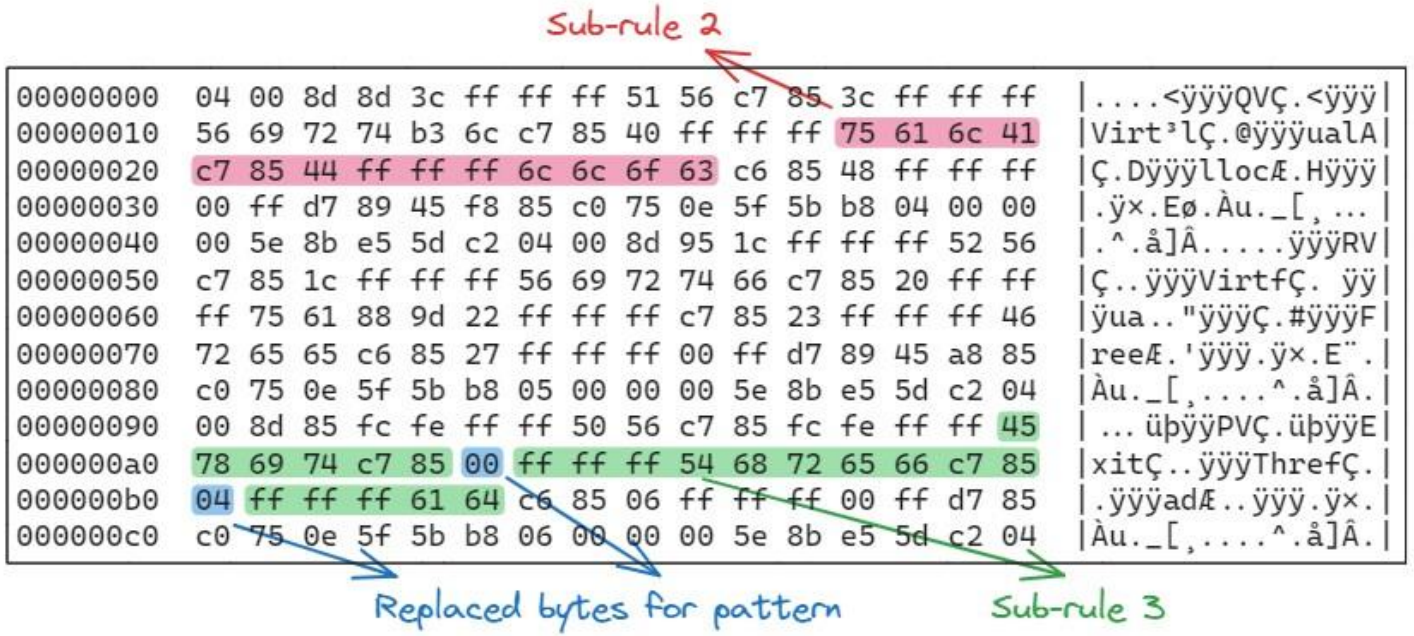
  0x00

  0x04

  在红色子规则 2 中

  在绿色子规则 3 中

- 预计检测为: Plugx.A




```
PS C:\Program Files\Windows Defender> .\MpCmdRun.exe -Scan -ScanType 3 -File
'C:\Users\user\deeac56026f3804968348c8afa5b7aba10900aeabee05751c0fcac2b88cff71e' -DisableRemediation -
Trace -Level 0x10
Scan starting ...
Scan finished.
Scanning C:\Users\user\deeac56026f3804968348c8afa5b7aba10900aeabee05751c0fcac2b88cff71e found 1 threats.

<===========================LIST OF DETECTED THREATS===========================>
---------------------------- Threat information -----------------------------
Threat                  : Backdoor:Win32/Plugx.A
Resources               : 1 total
    file                : C:\Users\user\deeac56026f3804968348c8afa5b7aba10900aeabee05751c0fcac2b88cff71e
-----------------------------------------------------------------------------
```

Expected detection

90 02 XX

模块 90 02 XX:

用作占位符，最多可匹配 XX 个特定字节

示例模块标记为 青色

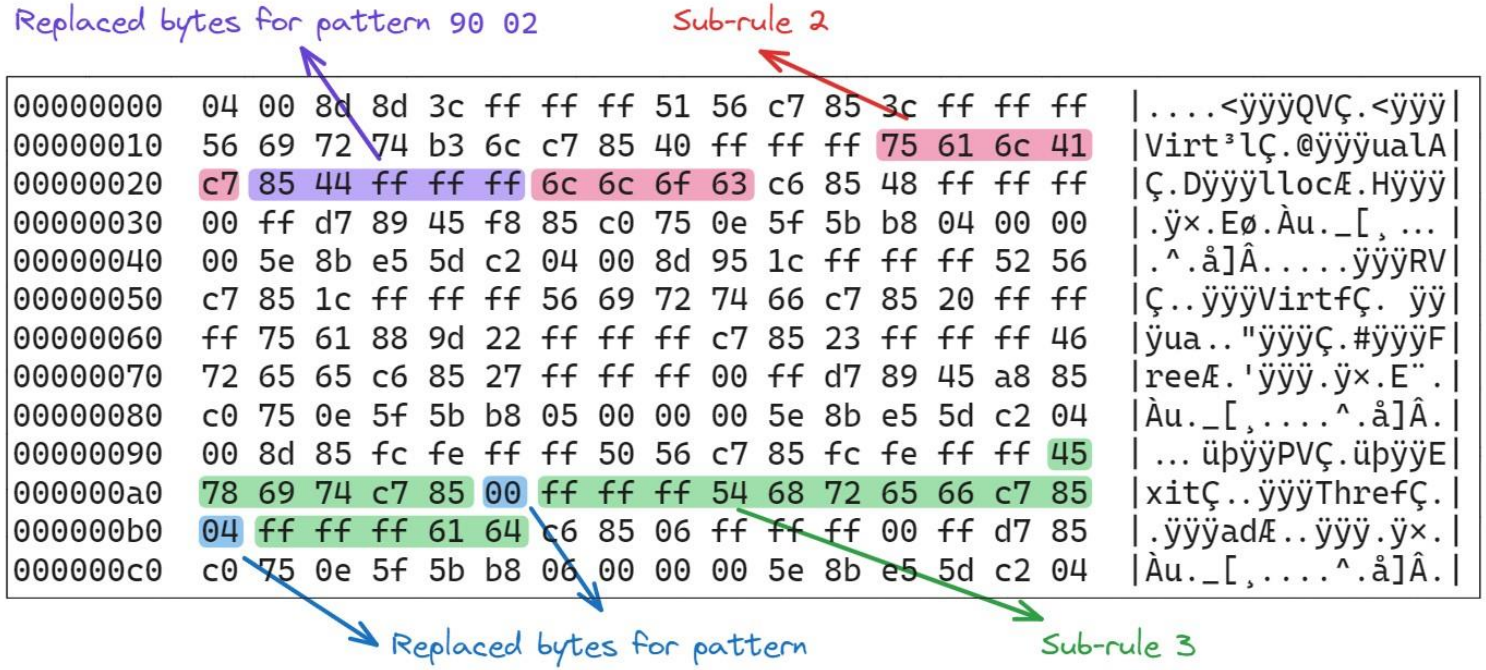
```
rule PlugxA-Sub-Rule2-Example { strings:
            $sub_rule_2_hex = {75 61 6C 41 C7 [0-16] 6C 6C 6F 63 }
condition:
           $sub_rule_2_hex }
```

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



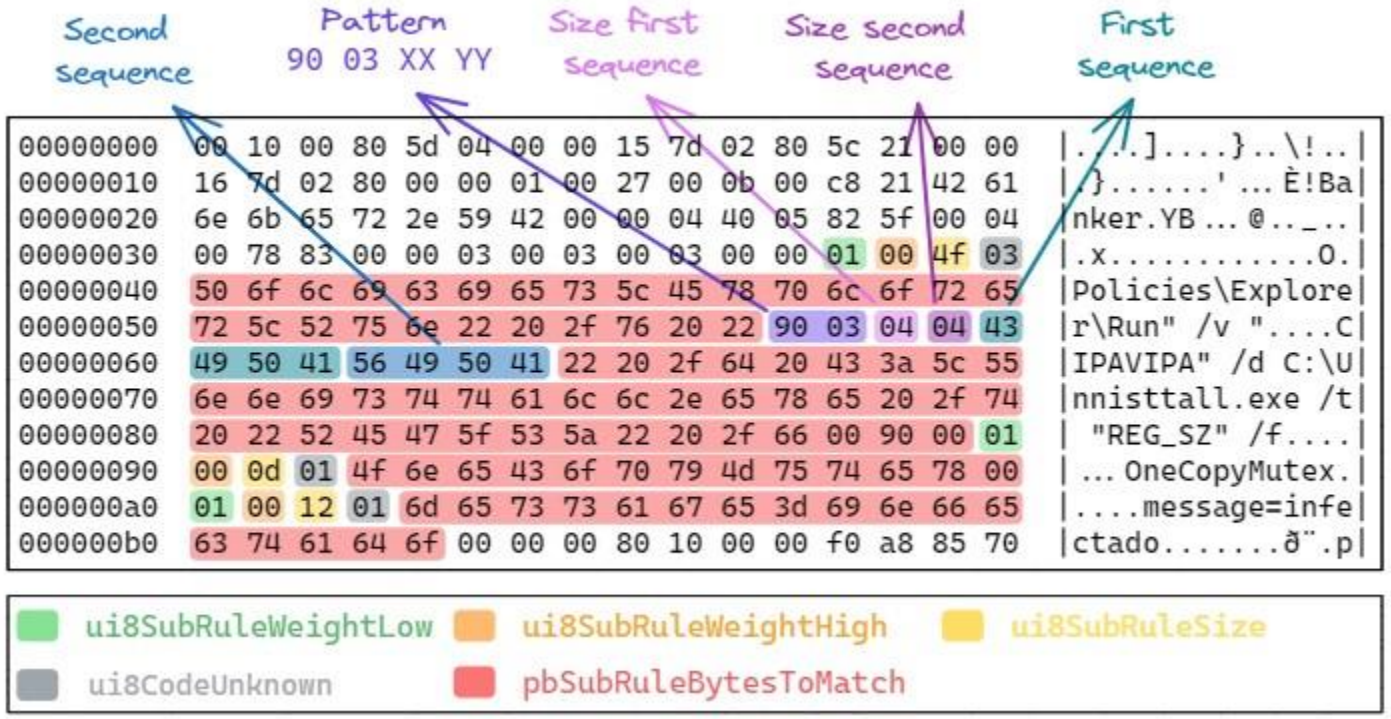

```
PS C:\Program Files\Windows Defender> .\MpCmdRun.exe -Scan -ScanType 3 -File
'C:\Users\user\deeac56026f3804968348c8afa5b7aba10900aeabee05751c0fcac2b88cff71e' -DisableRemediation -
Trace -Level 0x10
Scan starting...
Scan finished.
Scanning C:\Users\user\deeac56026f3804968348c8afa5b7aba10900aeabee05751c0fcac2b88cff71e found 1 threats.

<===========================LIST OF DETECTED THREATS===========================>
---------------------------- Threat information -----------------------------
Threat                  : Backdoor:Win32/Plugx.A
Resources               : 1 total
    file                : C:\Users\user\deeac56026f3804968348c8afa5b7aba10900aeabee05751c0fcac2b88cff71e
-----------------------------------------------------------------------------
```

Expected detection

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

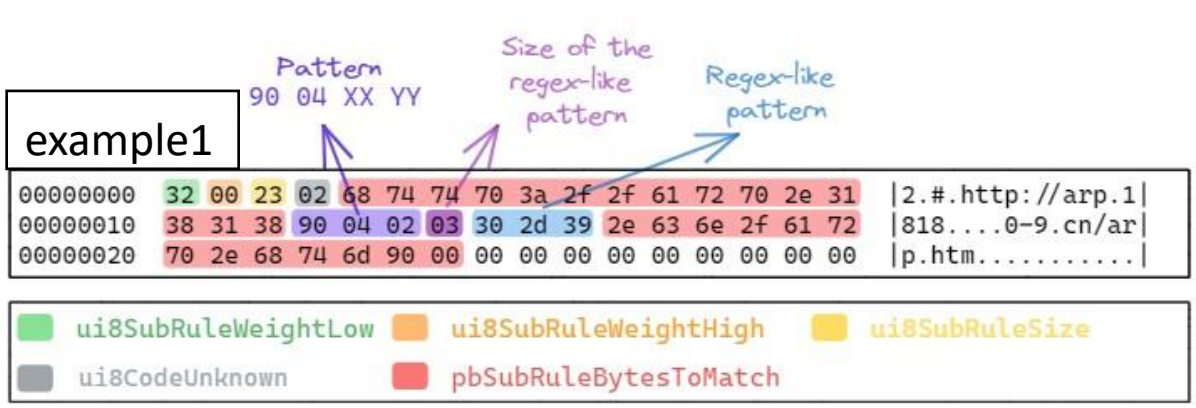


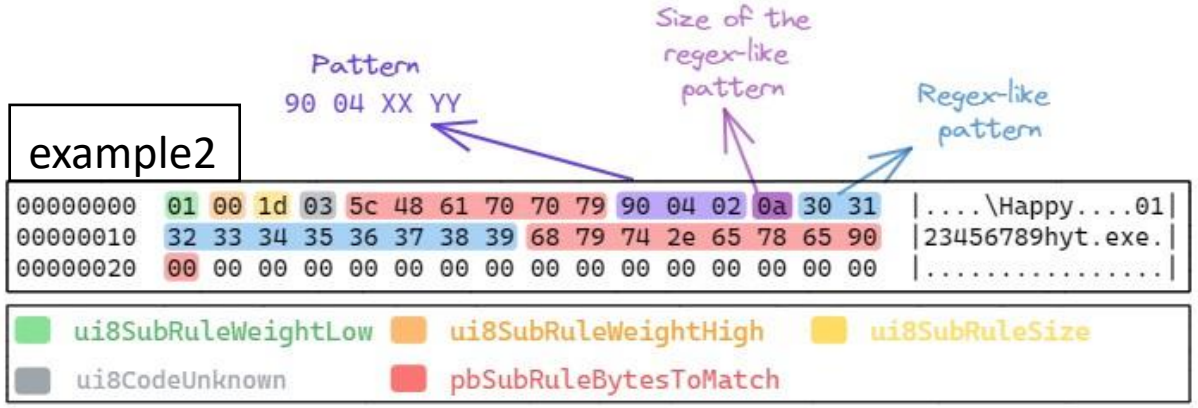

```
rule
    BankerYB_Sub_Rule1_Example{    strings:
        $sub_rule_1_hex = { 50 6f 6c 69 63 69 65 73 5c 45 78 70 6c 6f 72 65 72 5c 52 75 6e 22 20 2f 76 20 22
                            (43 49 50 41|56 49 50 41) 22 20 2f 64 20 43 3a 5c 55 6e 6e 69 73 74 74 61 6c 6c 2e
                             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
}
```

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

```
rule Pattern-90-04-example
   {     strings:

      $example1_90_04 =
{ 68 74 74 70 3a 2f 2f 61 72 70 2e 31 38 31 38 [30-39]
                                                -     [30
 39] 2e 63 6e 2f 61 72 70 2e 68 74 6d 90 00 }

                                          $example2_90_04 =
{ 5c 48 61 70 70 79 [30-39]    -39] 68 79 74 2e 65 78 65 90
00 }                    [30

   condition:
      $ example1_90_04 and
$ example2_90_04 }
```





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
- 红色字段命中子规则

Replaced bytes for pattern 90 04 02 03 30 2d 39

Sub-rule 2

```
00000000  00 00 00 00 00 00 00 00 00 00 00 00 64 72 69 76  |............driv|
00000010  65 72 73 5c 73 79 73 74 65 6d 2e 65 78 65 00 00  |ers\system.exe..|
00000020  00 00 00 00 00 68 74 74 70 3a 2f 2f 61 72 70 2e  |.....http://arp.|
00000030  31 38 31 38 30 39 2e 63 6e 2f 61 72 70 2e 68 74  |181809.cn/arp.ht|
00000040  6d 90 00 00 00 00 00 00 00 00 72 6f 76 65 72 00  |m.........rover.|
00000050  00 00 00 00 00 00 77 70 63 61 70 2e 64 6c 6c 00  |......wpcap.dll.|
00000060  00 00 00 00 00 00 6d 79 65 78 65 00 00 00 00 00  |......myexe.....|
00000070  00 00 64 72 69 76 65 72 73 5c 6e 70 66 2e 73 79  |..drivers\npf.sy|
00000080  73 00 00 00 00 00 00 00 00 50 61 63 6b 65 74 2e  |s.........Packet.|
00000090  64 6c 6c 00 00 00 00 00 00 00 57 61 6e 50 61 63  |dll.......WanPac|
000000a0  65 74 2e 64 6c 6c 00 00 00 00 00 00 00 5f 64 65  |et.dll......._de|
000000b0  6c 65 74 65 6d 65 2e 62 61 74 00 00 00 00 00 00  |leteme.bat......|
000000c0  00 3a 74 72 79 00 00 00 00 00 00 00 69 66 20 20  |.:try.......if  |
000000d0  20 65 78 69 73 74 00 00 00 00 00 00 00 00 00 00  | exist..........|
```

example1

Pattern 90 04 XX YY

Size of the regex-like pattern

Regex-like pattern

```
00000000  32 00 23 02 68 74 74 70 3a 2f 2f 61 72 70 2e 31  |2.#.http://arp.1|
00000010  38 31 38 90 04 02 03 30 2d 39 2e 63 6e 2f 61 72  |818....0-9.cn/ar|
00000020  70 2e 68 74 6d 90 00 00 00 00 00 00 00 00 00 00  |p.htm...........|
```

ui8SubRuleWeightLow ui8SubRuleWeightHigh ui8SubRuleSize

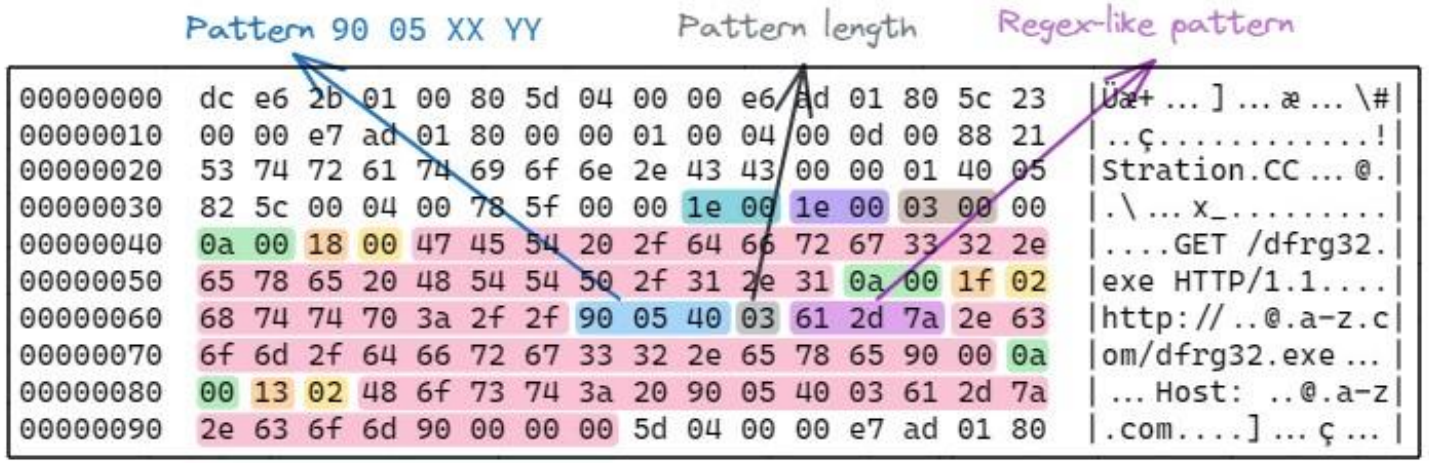
ui8CodeUnknown pbSubRuleBytesToMatch

```
PS C:\Program Files\Windows Defender> .\MpCmdRun.exe -Scan -ScanType 3 -File
'C:\Users\user\threat-small.exe' -DisableRemediation -Trace -Level 0x10
Scan starting ...
Scan finished.
Scanning C:\Users\user\threat-small.exe found 1 threats.

<============================LIST OF DETECTED THREATS============================>
------------------------------ Threat information -------------------------------
Threat                  : Backdoor:Win32/Small
Resources               : 1 total
    file                : C:\Users\user\threat-small.exe
--------------------------------------------------------------------------------
```

Expected detection

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
- 90 05 XX YY 后的字段用正则匹配式描述模块本身




```
rule Pattern-90-05-
      example{  strings:

                                        $example_90_05 =
        "http:// [a-zA-                 \\.com/dfrg32\\.exe“
                 Z]{0,64}
      condition:
            $example_90_05

}
```

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

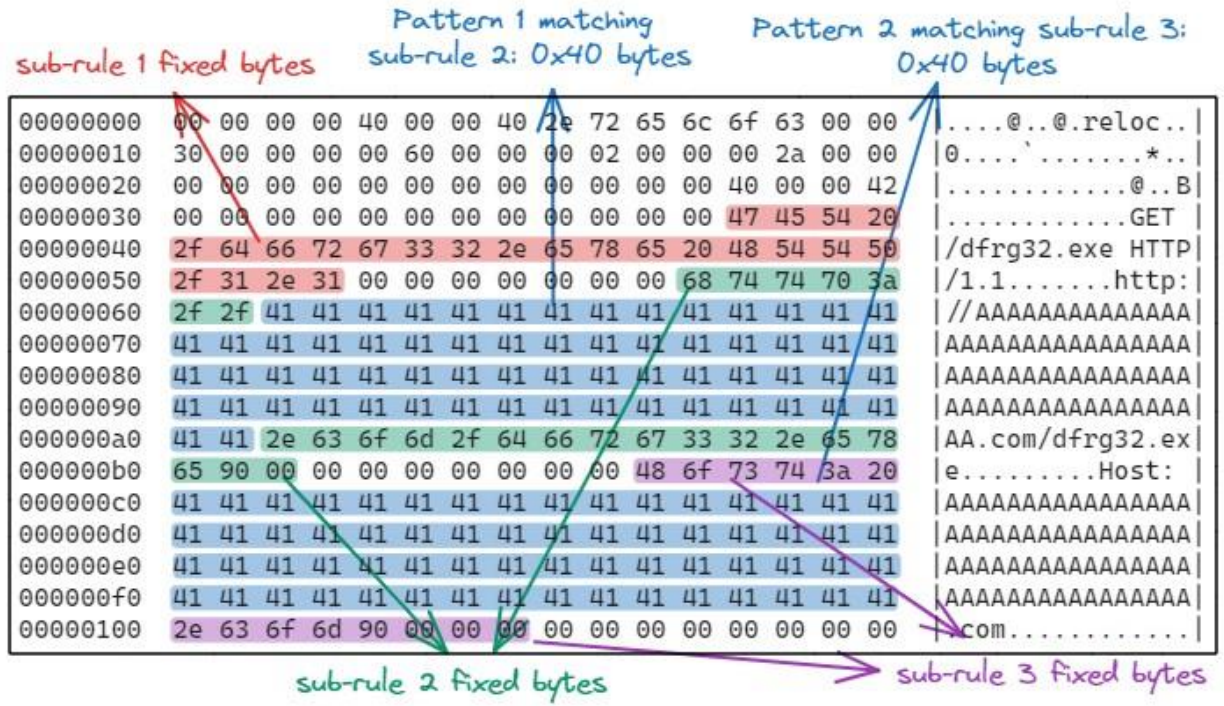
  cd /c/<your_path>/lab3_stration/Exercise
- 分析 Stration.CC PEHSTR 签名

  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




```
PS C:\Program Files\Windows Defender> .\MpCmdRun.exe -Scan -ScanType 3 -File
'C:\Users\user\threat-strationcc.exe' -DisableRemediation -Trace -Level 0x10
Scan starting ...
Scan finished.
Scanning C:\Users\user\threat-strationcc.exe found 1 threats.

<============================LIST OF DETECTED THREATS============================>
------------------------------ Threat information -------------------------------
Threat                    : TrojanDownloader:Win32/Stration.CC
Resources                 : 1 total
    file                  : C:\Users\user\threat-strationcc.exe
--------------------------------------------------------------------------------
```

Expected detection

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

2. 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
3. 修改 `lab4_havoc\Exercise\havoc_emu_asm.S` 以实现与提供的 `XorAlgorithm` 样本相同的功能
4. 使用 `build.sh` 脚本构建

https://retooling.io/blog

That'sAllFolks

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/3af5a472e11b7e135273e71cb5fed866/raw/15be4f2ae75b2d62465cf9faef72a2f61147a393/ExpandDefenderSig.p s1
- https://learn.microsoft.com/en-us/defender-endpoint/command-linearguments-microsoft-defender-antivirus