

0×00 简介

YARA是一款旨在帮助恶意软件研究人员识别和分类恶意软件样本的开源工具，使用YARA可以基于文本或二进制模式创建恶意软件家族描述信息，当然也可以是其他匹配信息。

这款工具配备一个短小精悍的命令行搜索引擎，它由纯C语言编写，优化了执行的效率。该引擎可以跨平台使用，能够运行Windows,Linux及Mac OS X系统上。同时，这款工具提供python扩展，允许通过python脚本访问搜索引擎。最后，也是最重要的一点，该引擎也可以扫描正在运行的进程。YARA规则类似于C代码，通常由两部分组成：脚本定义和布尔表达式（condition）。我们可以参看如下的规则示例：

```
rule evil_executable
{
    strings:
        $ascii_01 = "mozart.pdb"
        $byte_01  = { 44 65 6d 6f 63 72 61 63 79 }
    condition:
        uint16(0) == 0x5A4D and
        1 of ( $ascii_01, $byte_01 )
}
```

规则语法的简单明了和布尔逻辑使YARA成为一个完美的IOC（Indicator of Compromise, 攻陷指标）。事实上，从2011年开始支持YARA规则的安全厂商数量不断地增加，意味着这款工具不再局限于分析人员的笔记本上。它已经集成到恶意软件沙箱，蜜罐客户端，取证工具以及网络安全工具中。而且，随着越来越多的安全社区采取YARA格式来分享IOC，我们可以预见到这个格式在网络防御领域的广泛应用前景。

维基百科给攻陷指标（IOC）所下的定义是“网络上或操作系统中所观察到的异常。高可信度的异常意味着计算机入侵。”

与此同时，YARA成为一款功能丰富的扫描器，尤其是与模块的集成。模块可以在保持规则可读性的基础上开启了非常精细的扫描。比如PE模块，专门处理Windows可执行程序，我们可以创建一条匹配特定PE区块(Section)名称。同情况类似，我们使用Hash模块基于文件的某部分创建哈希值，比方说是PE文件的某个区块（Section）。

0×01 事件响应中的应用

究竟类似YARA的工具是如何整合到事件响应（Incident Response，缩写为IR）团队？可能最明显的答案是在执行恶意代码静态分析过程中开发和使用YARA规则。这个过程让你有机会交叉索引样本和之前的分析结果，如果样本和之前分析的结果相匹配的话可以为我们节省不少时间。否则，我们需要基于样本的分析结果来创建新的规则。虽然这种方法并没有什么不妥之处，但是此过程仅仅关注事件响应的某个具体阶段。进一步讲，如果不进行恶意代码分析，你最终很可能不会把YARA纳入自己的工具集。

0×11 垃圾邮件分析

让我们看一下垃圾邮件分析的应用场景。如果你的团队需要在事件响应过程中分析可疑的邮件消息，你极有可能会发现携带恶意宏的文件或重定向至漏洞利用工具的站点。olevba.py是一款流行的分析可疑微软office文档的工具，它属于oletools工具包的一部分。当分析嵌入的OLE对象来识别恶意活动时，它会使用YARA功能（更多内容可参看）。在应对漏洞利用工具时，thug一款流行的低交互式蜜罐客户端，模拟成web浏览器，也会使用YARA来识别漏洞利用工具家族。在上述两种场景中，事件响应团队之间交换YARA规则可以大大增强垃圾邮件的分类和分析的能力。

0×12 取证分析

另一种值得一提的应用场景是取证。Volatility一款非常流行的内存取证工具，可以支持YARA扫描来查明可疑的对象，比如进程、文件、注册表键值或互斥体（mutex）。相对于静态文件的规则，因为它需要应对加壳器和加密器，分析内存对象的YARA规则通常可以获得更广的观察范围。在网络取证领域，yara Pcap使用YARA扫描网络数据包文件（PCAP）。类似于垃圾邮件分析的应用场景，使用YARA规则进行取证可以起到事半功倍的作用。

0×13 终端扫描

最终，还有值得留意的应用场景是端点扫描。不错，在客户端计算机上进行YARA扫描。由于YARA扫描引擎是跨平台的，我们完全可以在Windows系统上使用Linux系统上开发的特征规则。唯一需要解决的问题是如何分发扫描引擎，下发规则，以及将扫描结果发送到某个中心位置。Hipara，一款C语言开发的主机入侵防御系统，可以实现基于YARA规则文件的实时扫描，并将报告结果发回到某个中心服务器。另一种解决方案是自己编写python脚本来调用YARA模块，同时使用REST库实现推拉（pull/push）的操作。此过程及其概念验证的代码已在SANS文章“ntelligence-Driven Incident Response with YARA”中详细描述（[打开此链接查看](#)）。

提升事件响应团队能力的关键是引入YARA规则创建和使用的流程。首先是如何使用YARA来加强事件触发机制，并提供规则的反馈，重点关注误报，然后进行规则的微调。此外，应该创建一个资料库来集中存储规则，并保持规则的更新。最后，团队应该统一规则的命名方案，最好能反应事件响应团队所用的分类。以上这些是一些将YARA整合到事件响应流程的关键步骤，并为团队接下来实现IOC共享流程做好准备。

0×02 实战环节

0×21 环境准备

在实战之前，我们需要一个Linux系统环境和下列工具：

1. [YARA 3.4.0](#)
2. [pescanner.py](#)

此外你需要一个段恶意代码来分析，你可以从Malwr.com网站上获取样本：

1. 样本地址：[下载链接](#)

2. 样本MD5值：f28b0f046046c861175426f6b3081061

2. 样本MD5值：f38b0f94694ae861175436fcb3981061

警告：该样本是一个真实的恶意软件，确保分析是在可控、隔离和安全环境中进行，比如临时性的虚拟机。

0×12 场景模拟

在周三下午4点，你的邮箱接受到一份事件报告的通知邮件。它似乎是一个可疑的HTTP文件下载（文件哈希值为**f38b0f94694ae861175436fcb3981061**）命中了网络IPS的特征库。你迅速检查IPS报警的详情，查看它是否把样本存入待深入分析的临时仓库中。你可以发现文件已被成功的保存下来，且文件类型是PE（可执行文件），绝对值得一看。下载文件之后，你需要进行初始的静态分析：利用Google和VirusTotal查询这个哈希值，分析PE文件头来寻找[恶意的企图](#)。

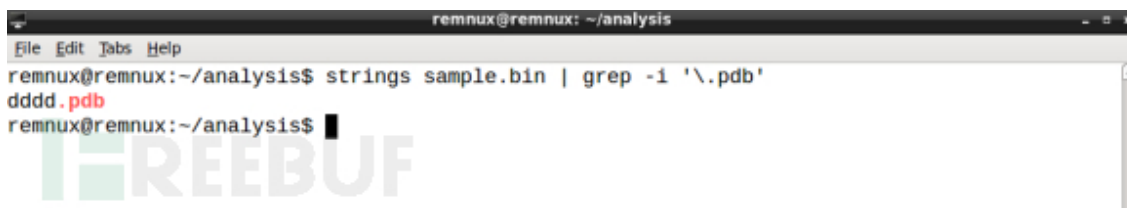
0×13挑战

创建匹配下述条件的YARA规则：

1. 与调试信息相关的可疑字符串
2. .text区块的MD5哈希值
3. 高熵值的.rsrc区块
4. GetTickCount导入符号
5. Rich签名的XOR密钥
6. 必须是Windows可执行文件

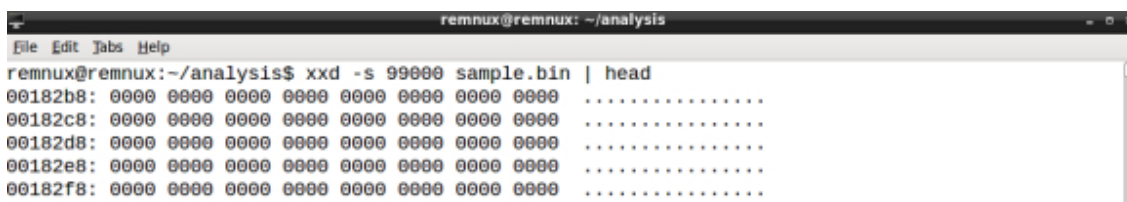
0×14静态分析

有关PE文件头的详细介绍，你可以参考这篇[Windows平台shellcode开发入门（二）](#)。第1个挑战是寻找与调试信息相关的字符串，尤其是我们可以搜索pdb文件（程序数据库文件）的路径。我们使用strings命令输出文件中的ASCII字符串。



```
remnux@remnux: ~/analysis
File Edit Tabs Help
remnux@remnux:~/analysis$ strings sample.bin | grep -i '\.pdb'
dddd.pdb
remnux@remnux:~/analysis$
```

在大量输出中，**dddd.pdb**字符串显得格外显眼，这个字符串可能正是我们要找的。记住，如果字符串确实与调试信息有关，它应该属于RSDS头的一部分。让我们使用**99136**偏移作为中心输出样本的部分字节。



```
remnux@remnux:~/analysis
File Edit Tabs Help
remnux@remnux:~/analysis$ xxd -s 99000 sample.bin | head
00182b8: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00182c8: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00182d8: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00182e8: 0000 0000 0000 0000 0000 0000 0000 0000 .....
00182f8: 0000 0000 0000 0000 0000 0000 0000 0000 .....
```

```

0018308: 0000 0000 0000 0000 0000 0000 0000 0000 .....
0018318: 0000 0000 2100 0000 1c00 0000 2883 0100 ....!.....(...
0018328: 5253 4453 97ac 0100 996e 0500 cb2e 0700 RSDS.....n.....
0018338: 1806 0100 0100 0000 6464 6464 2e70 6462 .....dddd.pdb
0018348: 00c1 8301 0000 0000 0000 0000 007b 8601 .....{..
remnux@remnux:~/analysis$

```

RSDS字符串的出现让我们确信dddd.pdb是与调试信息相关的字符串。接下来，我们需要计算.text区块的哈希值，为此我们需要使用hiddenillusion版本的pescanner.py，并把样本的名称作为参数。

```

remnux@remnux:~/analysis$ pescanner.py sample.bin
[0] File: sample.bin
=====
Meta-data
=====
Size           : 335872 bytes
Type           : PE32 executable (GUI) Intel 80386, for MS Windows
Architecture   : 32 Bits binary
MD5            : f38b0f94694ae861175436fcb3981061
SHA1           : 3e2eb77e74681f130a81bd9a05bdb7ad7451336
ssdeep         : 6144:hKwYi5hr6D061DdffCDvxbAZ9GC97DZNos5mWL949V0151:hKwUD5rfkvxbA9BX3153KV015
lmphash        : e724b5c43dc9c61d740a9b21f30eb96f
Date           : 0x50E21FCD [Mon Dec 31 23:29:17 2012 UTC]
Language       : NEUTRAL
CRC: (Claimed) : 0x54b2f, (Actual): 0x54b2f
Entry Point    : 0x406444 .text 0/4

Sections
=====
Name      VirtAddr  VirtSize  RawSize  MD5                                Entropy
-----
.text     0x1000    0x16414   0x17000   2a7865468f9de73a531f0ce00750ed17 5.427415
.rdata    0x18000   0x945     0x1000    c663fda3d7b936dfc47c996cdb0fbe57 3.023484
.data     0x19000   0xc92     0x1000    5bd6727d52ce29a93bdec4b619bc282c 4.931830
.rsrc     0x1a000   0x375a0   0x38000   746376ccec9b9f357c62cea98995c126 7.983700 [SUSPICIOUS]

Resource entries
=====
Resource type  Total
-----
RT_FONT       : 2

Imports
=====
[1] kernel32.dll
[2] ctl3d32.dll
[3] crypt32.dll
[4] user32.dll
[5] dbnmpntw.dll

Suspicious IAT alerts
=====
[1] CopyFileA
[2] CreateDirectoryW
[3] FindNextFileW
[4] GetDriveTypeW
[5] GetProcAddress
[6] GetStartupInfoA
[7] GetTickCount
[8] LoadLibraryA
remnux@remnux:~/analysis$

```

pescanner.py输出一个有关PE头结构的扩展报告，其中包含区块（Section）列表及相应哈希值。记录下来.text区块的哈希值（2a7865468f9de73a531f0ce00750ed17），接下来我们需要利用该值创建YARA规则。

同时在pescanner.py的报告中，我们发现.rsrc具有非常高的熵值。这是一个非常可疑的指标，表明代码经过高度的混淆。创建规则时一定要记住这条信息，因为它帮助我们应答挑战的第3项。最后，报告也列出了导入符号，其中我们可以看到GetTickCount，一个非常有名的反调试计时函数。这可以帮助解决挑战的第4项。顺便提一下，报告也提到了文件类型，表明是它一个PE32文件，满足挑战的最后一项。

最后，我们需要着手处理用于编码Rich签名的XOR密钥，有关Rich签名的更多信息参看[此文](#)。你可以通过两种方式来检查密钥的存在与否：通常你可以转储样本前面的字节，足以覆盖PE文件的DOS头即可，Rich签名开始于文件的0x80偏移处，而XOR密钥位于紧随Rich字符串之后双字（Dword）

המחלוקת בין אלו המאמינים כי יש להעביר את המערכת לידי הממשלה, לאלו המאמינים כי יש להעבירה לידי הוועד, נבעה מן העובדה כי הוועד לא היה מוכן להעביר את המערכת לידי הממשלה (עמ' 10).

```
remnux@remnux: ~/analysis
file Edit Tabs Help
remnux@remnux:~/analysis$ xxd sample.bin | head -n 15
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ.....
00000010: b000 0000 0000 0000 4000 0000 0000 0000  .....@.....
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .....
00000030: 0000 0000 0000 0000 0000 0000 c000 0000  .....
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  ....!..!..L..!Th
00000050: 6973 2070 726f 6772 616d 2963 616e 6e6f  is program canno
00000060: 7420 6265 2072 756e 2069 6e20 444f 5320  t be run in DOS
00000070: 6d6f 6465 2e0d 0d0a 2400 0000 0000 0000  mode....$.
00000080: e3e2 11db a783 7f88 a783 7f88 a783 7f88  .....
00000090: a783 7f88 b883 7f88 5ba3 6d88 a683 7f88  ....[.m.
000000a0: 6085 7988 a683 7f88 5269 6368 a783 7f88  `y.....Rich...
000000b0: 0000 0000 0000 0000 0000 0000 0000 0000  .....
000000c0: 5045 0000 4c01 0400 cd1f e250 0000 0000  PE..L.....P...
000000d0: 0000 0000 e000 0f01 0b01 0600 0070 0100  .....p.
000000e0: 00a0 0300 0000 0000 4464 0000 0010 0000  ....Dd.....
remnux@remnux:~/analysis$
```

记住x86字节顺序是**little-endian**，因此你需要调整双字(Dword)的字节顺序，所以XOR密钥的值为**0x887f83a7**或十进制**2290058151**。

[illegible]

接下来就简单了，让我们一起使用如下内容创建名为**rule.yar**的YARA规则文件。通过使用**-print-module-data**参数，YARA将会输出PE模块的报告，其中包括**rich_signature**区块和十进制形式的XOR密钥值。

现在，我们已经搜集创建YARA规则所需的全部信息，并完成这次的挑战。在接下来的部分，我们会继续深入介绍YARA其他的高级功能，敬请期待。

参考文献：

1. [Unleashing YARA – Part 1](#)
2. [Unleashing YARA – Part 2](#)
3. [YARA——恶意软件模式匹配利器](#)

* FB资深作者Rabbit_Run整理报道，转载请注明来自FreeBuf黑客与极客 (FreeBuf.COM)