南開大學
Nankai University

网络空间安全学院

恶意代码分析与防治技术课程实验报告

实验一：恶意代码分析与 Yara 检测

姓名：周钰宸

学号：2111408

专业：信息安全

2024 年 1 月 14 日

1 实验目的

1. 锻炼我们恶意代码基本静态分析的技术能力；
2. 为了能够模拟真实的恶意代码分析过程，关于恶意代码程序的几乎任何信息都没有被给予，具体体现为：
   - 通用名称 (generic names)；
   - 几乎毫无意义甚至带有误导性的相关名称 (meaningless or misleading names)；
3. 提高我们对 Yara 的熟悉程度以及对 Yara 检测规则的实践编写能力；
4. 进一步探索并讨论如何编写更加快速的 Yara 规则。

2 实验原理

2.1 File Signature

在恶意代码分析和计算机安全领域，"file signature"（文件签名）通常指的是文件的特定标识或标志，用于唯一地标识文件的类型、格式或内容。这些签名通常以二进制形式表示，是文件的前几个字节或特定的字节序列。它们用于检测和识别文件，以确定文件是否与已知的恶意软件、病毒、特定文件格式或其他有害的内容相关联。

2.2 Antivirus Scanning

对于一直到恶意代码可以采用文件签名和启发式算法的方式，对于位置的恶意代码吗可以采用混淆处理，结合多态性语法混淆和结合变形性的语义混淆。相关工具方面可以使用 VirusTotal。

2.3 Hashing

是一种**唯一识别**恶意代码的方法。输入为一个具有任意长度的文件或字符串，输出为固定长度的哈希值，在实践中可以唯一地识别一个文件。除此以外，它的作用还包括与其它分析者一起公用哈希值来识别恶意代码以及在网上搜索哈希值来看是否已经有人提前识别了该种文件。常用的相关工具为 HashCalc。

2.4 Finding Strings

1. 字符串：一序列可以打印的字符，使用计算机能够理解的 0 和 1 代表字母，主要分为 ASCII 和 UNICODE 两种。
2. 字符串命令：通过寻找 ASCII 或者 Unicode 字符串来寻找二进制可执行文件。经常是由三个或更多字符序列构成，伴随着终结符。

2.5 Packed and Obfuscated Malware

这是两种都带着使得恶意代码更难进行逆向工程和检测共同目的的行为。与带有较多字符串的合法项目相比，进行了混淆或者加壳处理的恶意代码往往具有更少的字符串。其中混淆可以隐藏可执行

信息；加壳压缩二进制文件的大小，本质上是混淆的其中一个方法。加壳后会使得相关字符串和指令无法被阅读。唯一可见的信息是外面的包装程序，一小段当程序运行会把文件脱壳。相关分析工具为 PEiD。

2.6 Portable Executable File Format

PE 文件格式是 Windows 操作系统下的可执行文件格式，用于存储和加载可执行代码、数据、资源和动态链接库等。PE 文件结构由 PE 文件头和节表组成。包含了基本的能够帮助 Windows 加载二进制可执行文件的必备信息。几乎所有的在 Windows 上能够被执行的文件都是 PE 文件格式。

PE 文件头是 PE 文件的第一个部分，包含了 PE 文件的基本信息，比如需要被加载的库和函数信息。可以使用 LordPE Demo 查看相关 PE 文件的信息。

2.7 Linked Libraries and Functions

一个程序所使用的函数都会被存储在不同的程序中，比如一些库。它们通过链接的方式串到 exe 中。通常有三种链接的方式：静态链接，动态链接和运行链接。其中第一种最少见，第二种最常见，而第三种在一些善意的程序中并不常见，但是在恶意软件中很常见，特别是那些使用了加壳和混淆技术的恶意程序。当这些函数被调用和需要的时候，相关的库才会被链接，而并不是在程序启动时。这在 **LoadLibrary 和 GetProcAddress** 两个函数上体现的尤为明显。基于上述事实，我们可以利用其中的库中的一些信息来帮助我们进行恶意代码分析：

- PE 头的列出了每个将要被加载的库和函数；
- 每个库和函数的名字揭示了这个程序要做什么的目标；
- URLDownloadToFile 这个库函数会指示程序想要下载的东西。

2.8 Dependency Walker

它是一种实用的解决模块依赖性问题的工具，也可以用来进行恶意代码的分析。它的作用包括：显示动态链接的函数，正常的程序往往需要较多的 DLLs 的导入，而恶意代码往往仅涉及较少的 DLLs；显示导入导出关系，DLLs 会导出许多函数，EXEs 会导入许多函数。而导入和导出都会被记录在 PE 文件头上。

2.9 The PE File Headers and Sections

2.9.1 重要的 PE 节

1. .text,.rdata,.data 和.rsrc。虚拟大小往往存储在 RAM 中，Raw Data 往往存储在磁盘中，对于.text 段来说，它们通常是一样大的。但是如果对于某个加壳的可执行文件，则它的.text 段会显示的虚拟文件大小远大于 Raw Data。
2. IMAGE SECTION HEADER: 这是一个结构体，它代表了 PE 文件上上述区段的头部信息。可以看作.text、.rdara、.dara、rsrc 它们这些区段的元数据。
3. Time Date Stamp: 显示了可执行文件被编译的时间。由于往往一些反病毒的软件都通常是较早期的程序。但是有时候显示的日期也可以被修改，甚至出现错误。

2.9.2 常用的查看 PE 结构的工具

1. PEview：一款使用 C/C++ 开发实现的命令行交互式 Windows PE 文件解析器，普遍应用于病毒木马等样本的解包分析工作。
2. Resource Hacker：能够让恶意代码分析者看到.rsrc 段内容的工具，包括其中的字符串、标志、菜单等。

2.10 Yara

YARA 是一种用于恶意软件分析、恶意代码检测和规则匹配的开源工具。它允许安全研究人员和分析师创建自定义规则，用于检测恶意代码、病毒、恶意软件家族等，并在大规模样本集合中进行自动化扫描。

2.10.1 Yara 的作用

1. 恶意代码检测：YARA 主要用于检测计算机病毒、恶意软件、恶意脚本等恶意代码。
2. 规则匹配：它通过用户定义的规则匹配模式来查找文件、内存或其他数据源中的特定模式。
3. 恶意代码分类：YARA 可以帮助分析师将恶意代码样本分类到特定的家族或变种中。

2.10.2 Yara 规则

YARA 规则是用于定义恶意代码检测条件的文本文件。规则通常包含两部分：元数据和规则体。元数据提供规则的名称、作者、描述等信息，规则体包含条件和操作。

1. 规则条件：YARA 使用强大的规则语言，允许用户定义恶意代码的特征、字符串、正则表达式等。
2. 规则操作：当规则条件匹配时，可以执行自定义操作，如生成警报、拦截文件等。

3 实验过程

3.1 实验环境

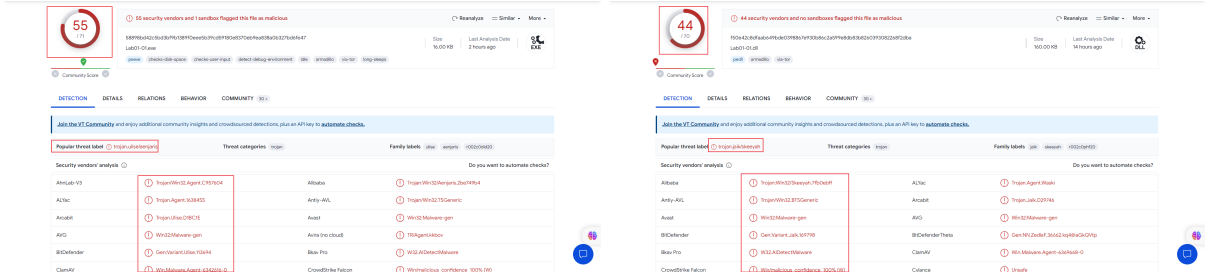
为了保证实验的安全性，选择使用 Sandbox 虚拟机的隔离环境进行实验，所选择的系统为 Windows XP Professional，实验所用虚拟机软件为 VMware Workstation Pro。

3.2 Lab01-01

3.2.1 实验问题

1. **Q1: 将文件上传至 http://www.VirusTotal.com/进行分析并查看报告。文件匹配到了已有的反病毒软件特征吗**?

   回答：将 lab01-01.exe 和 lab01-01.dll 放入 VirusTotal 官网中检测，检测结果如下图所示：



lab01-01.exe　　lab01-01.dll

图 3.1: VirusTotal 检测结果

可以明显地发现两个文件都有较高的威胁值得分，并且都具有流行的病毒标签，比如 trojan 即木马。除此之外，下面的安全公司的分析结果中也可以看到例如阿里巴巴个 Antiy-AVL 公司都找到了对应的病毒签名。**因此两个文件都能够和一些已知反病毒的签名吻合，基本可以确定就是病毒**。

2. **Q2: 它们是在何时被编译的？**

回答：先在虚拟机中使用 StudyPE 打开 lab01-01.exe，找到其 TimeDateStamp 所在的地址位置，如下图所示；

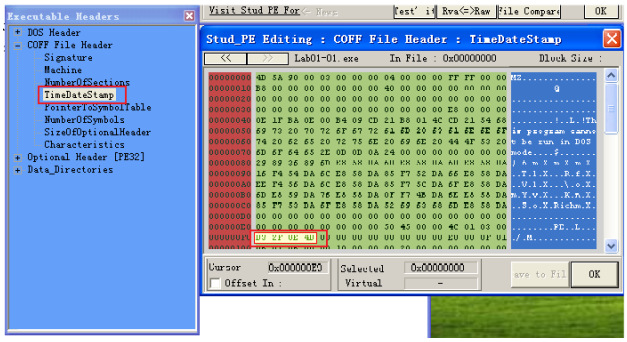


图 3.2: StudyPE 的 lab01-01.exe 的 TimeDateStamp 分析

由于 Windows 系统一般使用小端序，这意味着低位字节存储在低地址，高位字节存储在高地址。所以实际的时间戳为 **4D0E 2FD3**。编写如下代码将其转换为 UTC 时间格式：

```python
import datetime
def hex_timestamp_to_utc(hex_timestamp):
    # 解析十六进制字符串，转换为十进制数值
    decimal_timestamp = int(hex_timestamp.replace(' ', ''), 16)
    # 转换为UTC时间
    utc_time = datetime.datetime.utcfromtimestamp(decimal_timestamp)
    return utc_time
# 十六进制时间戳
hex_timestamp = "4D0E 2FD3"
# 转换为UTC时间
utc_time = hex_timestamp_to_utc(hex_timestamp)
print("UTC时间:", utc_time)
```

```
UTC时间: 2010-12-19 16:16:19
```

图 3.3: Python 程序进行时间戳转换

如上图所示，时间戳转换结果为:**UTC 时间: 2010-12-19 16:16:19**。再在 VirusTotal 中查看报告对比：

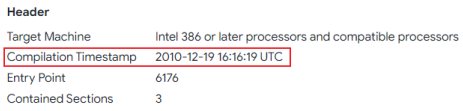
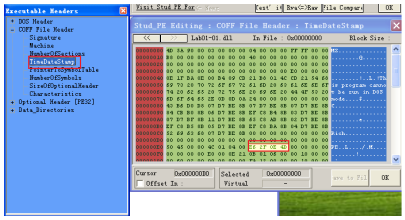


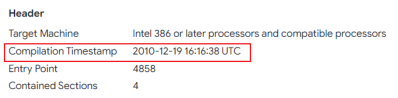
图 3.4: 与 VirusTotal 分析结果验证

发现与 VirusTotal 分析结果相同，验证了静态分析的正确性。同理，对 lab01-01.dll 进行相同操作，也可验证实验的正确性。



TimeDateStamp 分析

时间戳转换

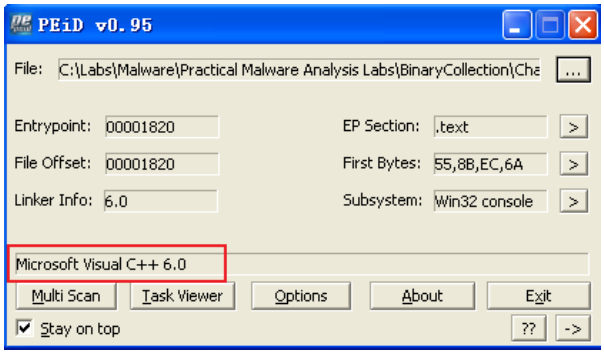


与 VirusTotal 分析结果验证

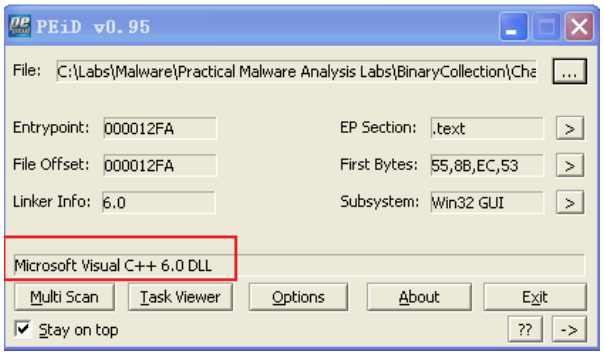
图 3.5: 对 lab01-01.dll 的编译时间分析

3. **Q3: 这两个文件中是否存在迹象说明它们是否被加壳或混淆了？如果是，这些迹象在哪里？**

   回答：在虚拟机中使用 PEiD 依次对 lab01-01 进行检测，检测结果如下：



lab01-01.exe



lab01-01.dll

图 3.6: PEiD 对 lab010-01.exe 和 lab01-01.dll 的分析结果

如上两张检验结果显示，看 PEiDdSCAN 的结果，可以看到该 exe 是用 VC++6.0 编写的，dll 是用 VC6++DLL 编写的。假设程序被加壳，PEiDdSCAN 要么显示壳的名称，要么显示”nothing found”。由于没有显示具体壳的名称，故**不存在迹象能够说明它们二者被加壳即混淆了**。

4. **Q4: 是否有导入函数显示出了这个恶意代码是做什么的？如果是，是哪些导入函数?**

   回答：正常来说应该使用 Dependency walker 完成这项任务，不过这里由于没有配备该工具，改为使用 PEiD 完成，以 Lab01-01.exe 为例，将待测文件拖入 PEiD，查看” 子系统” 和导入表：

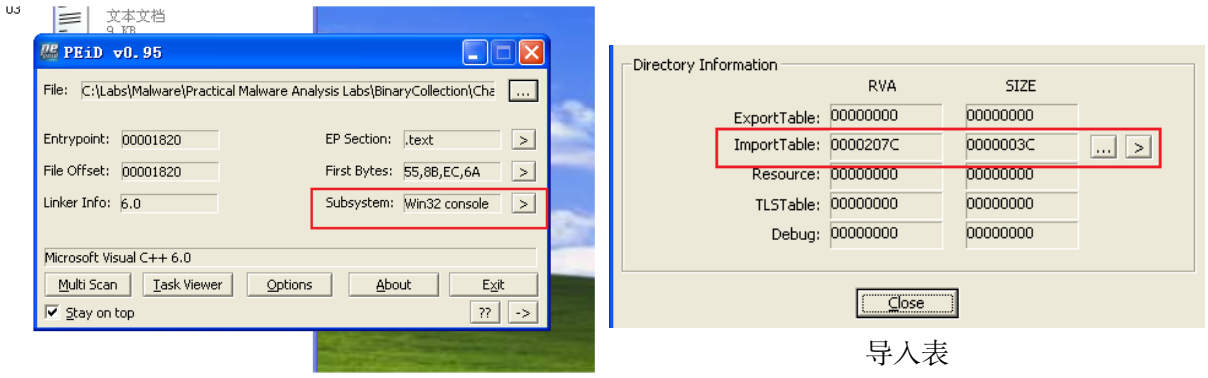


图 3.7: PEiD 对 lab01-01.exe 的导入关系分析

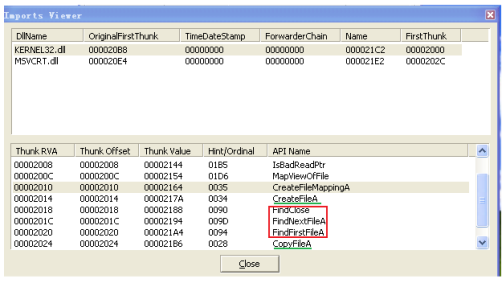


图 3.8: lab01-01.exe 调用的 API 函数 1

该病毒使用了 Kernel32.dll 中的一些 API 函数，其中可以重点发现：

- 使用 CreateFileA、CopyFileA 可能用于创建新文件或复制文件，病毒可能会在系统中创建新的恶意文件或复制自身以传播到其他目录或系统；
- 使用 FindFirstFileA、FindNextFileA、FindClose 可能用于搜索特定目录下的文件，这可能是为了寻找目标文件或目录；

由以上两点推断其可能进行文件操作和传播。

除此之外：

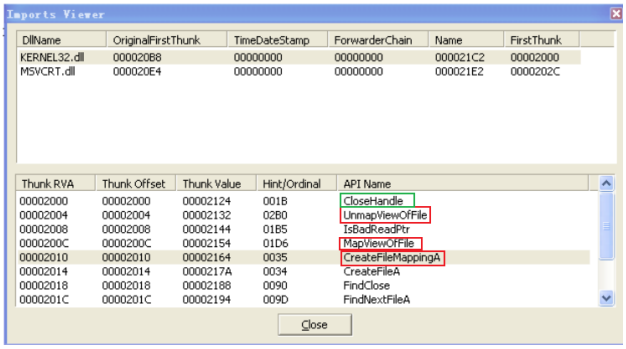


图 3.9: lab01-01.exe 调用的 API 函数 2

其中也可以重点发现:

- 使用 MapViewOfFile、UnmapViewOfFile、CreateFileMappingA 可能是为了操作内存映射文件，这可能与病毒的植入和执行恶意代码相关。
- 使用 CloseHandle 可能用于关闭文件、内存映射或其他系统资源。

由以上两点推断其可能进行系统资源访问和操作。

该病毒还使用了 MSVCRT.dll 中的一些 API 函数:

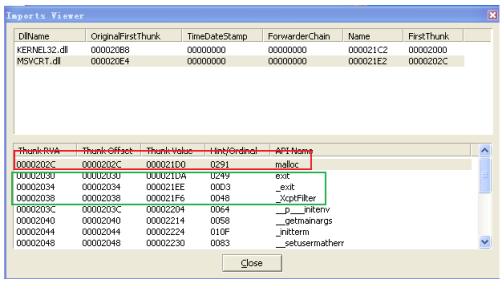


图 3.10: lab01-01.exe 调用的 API 函数 3

其中可以重点发现：使用 malloc 可能用于动态分配内存，这可能与病毒的运行时内存需求有关。其它基本每一个可执行文件都有。

同样的，查看 lab01-01.dll 中调用的 API 函数:

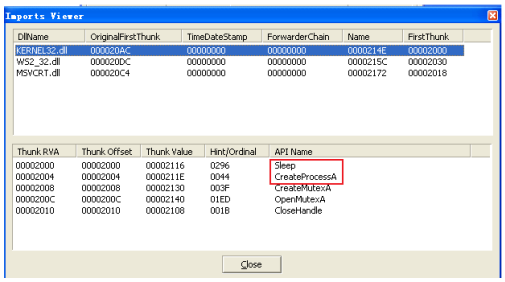


图 3.11: lab01-01.dll 调用的 API 函数

其中可以重点发现:

- CreateProcess 是用于创建新进程的函数。病毒可能会利用此函数创建并执行恶意进程，从而传播和执行恶意代码。
- Sleep 函数用于使当前线程休眠一段时间。病毒可能利用此函数来控制恶意行为的激活时间，以避免被轻易检测到。
- WS_32.dll 是为网络所用的库，导入了其中的函数可以用于联网。可能用于与远程服务器或其他恶意节点通信。病毒可以通过网络通信获取命令、下载更新、上传数据等，或者用于传播自身。

从这里我们可以猜测这个恶意代码会对文件系统进行搜索、打开、修改、创建等操作。具体的来说可能是把系统中某个关键位置的 dll 文件删除，并且替换换成这个程序中某段 dll 代码。

5. **Q5：是否有任何其他文件或基于主机的迹象，让你可以在受感染系统上查找?**

   回答：将 lab01-01.exe 拖入 PEView 中查看，如下图所示:

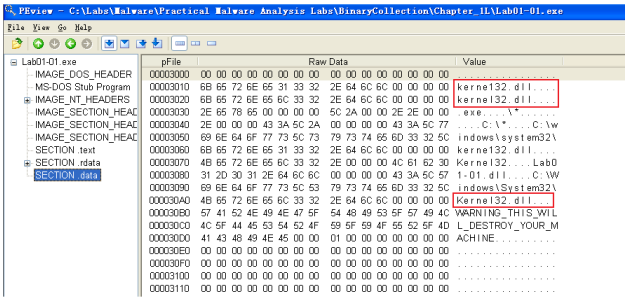


图 3.12: PEView 分析 lab01-01.exe 的字符串

可以发现出现了 exe，那么结合导入函数可以猜测是在文件系统中搜索 exe 文件，然后会有打开、修改等操作。

可以看到 kernel32.dll 被修改成了 kerne132.dll。把 l 换成了 1。结合上面的分析，可以判断是在 system32 中查找到 kernel32.dll，然后把它复制成新的文件 kerne132.dll 中。**这个变换可以作为线索在被感染的计算机中搜索**。用同样的方式查看 lab01-01.dll：

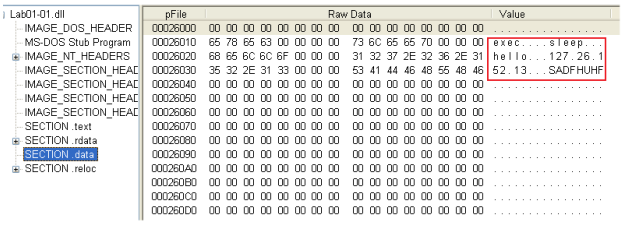


图 3.13: PEView 分析 lab01-01.dll 的字符串

可以看到一个 127.26.152.13 的 IP 地址，有可能是该恶意代码让主机与 127.26.152.13 这个 IP 建立一个连接。

综上，我们猜测这个恶意代码会获取系统权限，对文件系统进行搜索、打开、修改、创建等操作，且感染的主机给 127.26.152.13 发送了一个已连接的套接字数据。

6. **Q6: 是否有基于网络的迹象，可以用来发现受感染机器上的这个恶意代码?**

   回答： 同样将 lab01-01.dll 拖入 PEiD 中：

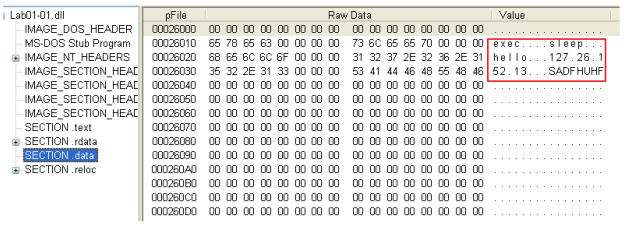


图 3.14: PEView 分析 lab01-01.dll 的字符串

可以看到一个 127.26.152.13 的 IP 地址，有可能是该恶意代码让主机与 127.26.152.13 这个 IP 建立一个连接。**如果是真正的恶意代码的话，这应该是一个可以访问的公网 ip，是一个很好的基于网络的恶意代码感染迹象，可以用来识别这个恶意代码**

7. **Q7: 你猜这些文件的目的是什么?**

   回答: 在之前提到的 dll 导入函数中, 可以注意到了 CreateProcess 和 Sleep, 这些函数在后门程序中被广泛使用。通常这些函数会结合 exec 和 sleep 等操作。这与之前在 PEView 查看 dll 文件时观察到的内容一致。可能利用 Exec 通过网络向后门程序发送传输命令, 而接着使用 CreateProcessA 启动程序。Sleep 可能被用于使后门程序进入休眠模式。

   综上分析, 我们怀疑该 dll 文件可能是一个后门, 而 exe 文件可能用于安装和运行 dll 文件。该恶意代码可能会尝试获取系统权限, 并对文件系统进行搜索、打开、修改和创建等操作。且受感染的主机向 IP 地址 127.26.152.13 发送一个已连接的套接字数据。

3.3 Lab01-02

3.3.1 实验问题

1. **Q1: 将文件上传至 http://www.VirusTotal.com/进行分析并查看报告。文件匹配到了已有的反病毒软件特征吗?**

   回答: 与 lab01-01 同理进行 VirusTotal 分析, 结果如下:

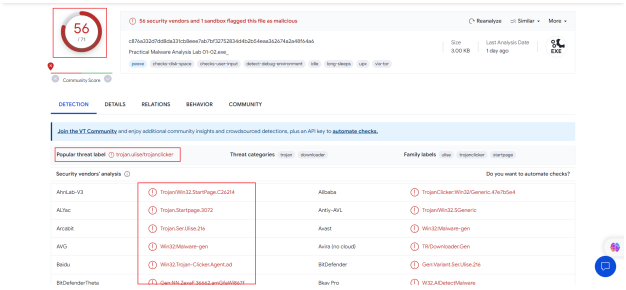


图 3.15: lab01-02.exe 的 VirusTotal 分析结果

   由上图可以发现反病毒引擎的报警率较高, 几乎可以确定其为一个计算机病毒。并且有些公司检测结果将其分为了木马或者木马点击器。**所以可以能够匹配到很多已有的反病毒软件特征。**

2. **Q2: 是否有这个文件被加壳或混淆的任何迹象? 如果是这样, 这些迹象是什么? 如果该文件被加壳, 如果可能的话进行脱壳。**

   回答: 先将 lab01-02.exe 拖入 PEiD 中:

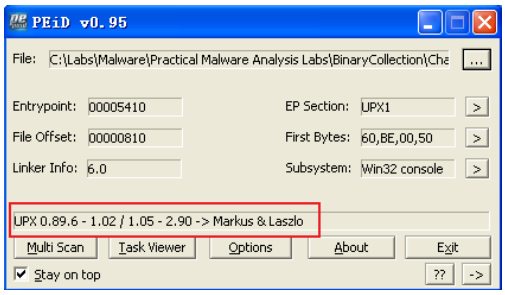


图 3.16: lab01-02.exe 的 PEiD 分析结果

   可以发现, 和 lab01-01 的两个文件结果显示不一样, **根据提示信息有可能用 UPX 加壳了。**进一步使用 PEView 进行分析:

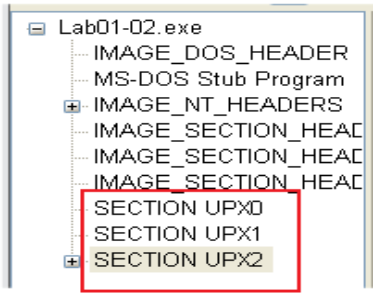


图 3.17: lab01-02.exe 的 PEView 分析结果

可以看到名为 upx0,upx1,upx2 的节，这是由 upx 加壳后恶意代码程序的节名称。**这些便是可能的迹象**。那么可以使用 upx 进行脱壳。

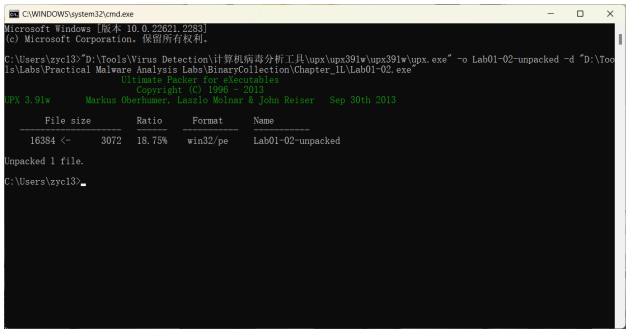


图 3.18: 采用 UPX 脱壳结果

由于我这里的 xp 系统不知道什么原因没办法正常使用命令控制 upx，因此选择在宿主机进行脱壳。使用 upx.exe–o 新文件名–d 加壳文件名命令进行脱壳后结果如上图所示，将脱壳后结果重新放入虚拟机隔离环境中。

3. **Q3: 是否有导入函数显示出了这个恶意代码是做什么的？如果是，是哪些导入函数?**

   回答：和 3.2.1Q4 相同，将 lab01-02 导入 PEiD 中然后顺序查找其子系统与导入表，

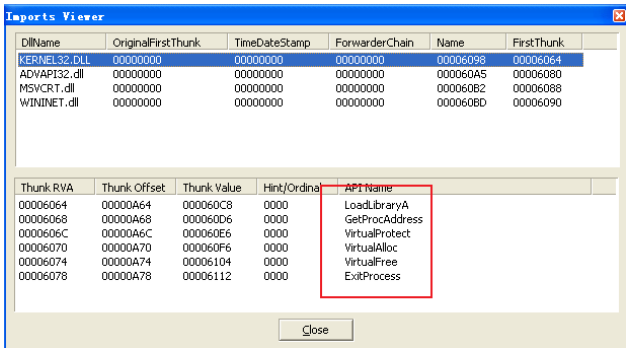


图 3.19: lab01-02 的 PEiD 分析结果——动态链接库

从上图可以发现除了之前出现的 kernel32.dll 和 MSVCRT.dll 之外，又多出现了 ADVAAPI32.dll 和 WININET.dll 两个动态链接库，其中 ADVAAPI32.dll 与注册表的操作以及事件日志有关；WININET.dll 是 Windows 应用程序网络相关模块，与网络的连接有关。**由此可以进行初步的猜**

测：该恶意代码可能存在修改注册表、修改内存信息、建立网络连接等行为。接下来进一步看一下导入函数：

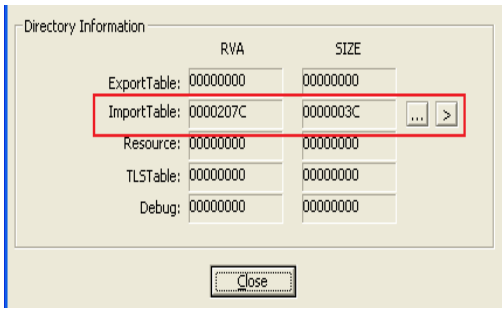


图 3.20: lab01-02 的 PEiD 分析结果——导入函数

可以重点发现：LoadLibraryA 调取动态链接库，由 GetprocAddress 获取动态链接库内的输出库函数地址。这两个函数结合起来可以调用某个动态链接库中的某个特定函数的。而 Virtualprotect 用来变更对程序的内存保护，Virtualalloc 用来申请内存空间，Virtualfree 用来释放内存空间，Exitprocess 用来结束某个进程。

故确实存在上述导入函数能够指示其恶意行为，或者暗示其调用了某个动态链接库中的函数，并对内存进行了申请、修改、释放等操作。

4. **Q4: 是否有基于网络的迹象，可以用来发现受感染机器上的这个恶意代码?**

   回答：是的，除了上述说的发现了的 WININET.dll 动态链接库可能会建立网络的连接以外。我将卸壳后的 lab01-02-unpacked 放入 PEView 中在 Section.data 节发现了如下的字符串：

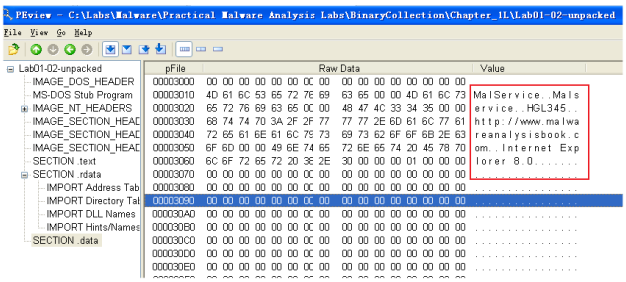


图 3.21: 彩蛋

可以发现其试图使用 Internet Explorer 连接 https://www.malwareanalysisbook.com 这个网址，**因此这也是一个可以用来发现受感染机器上的这个恶意代码的基于网络的迹象。另外我觉得这个可能是一个小彩蛋，但我尝试打开这个网站发现不成功。可能是因为比较久远了。但我还是对发现这个彩蛋好自豪（骄傲脸）**。

3.4 Lab01-03

3.4.1 实验问题

1. **Q1: 将文件上传至 http://www.VirusTotal.com/进行分析并查看报告。文件匹配到了已有的反病毒软件特征吗？**

回答：与之前同理将 lab01-03 拖入 VirusTotal 中分析，结果如下：

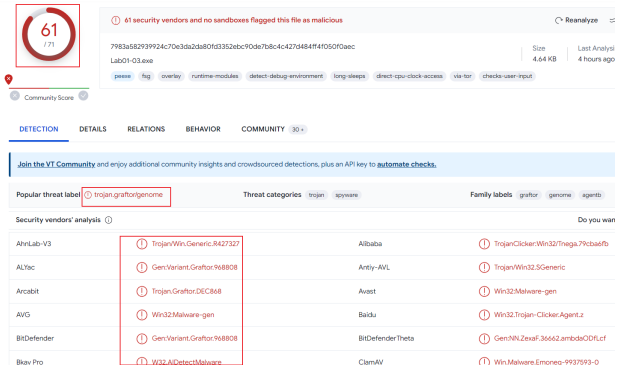


图 3.22: lab01-03.exe 的 VirusTotal 分析结果

可以发现与 lab01-01 和 lab01-02 的这两次的结果几乎相同，但是发现此时的病毒威胁系数更高了，也是匹配到了例如木马等已有的反病毒软件特征。

2. **Q2: 这个文件中是否存在迹象说明它是否被加壳或混淆了？如果是，这些迹象在哪里？如果该文件被加壳，如果可能的话进行脱壳。**

   回答：同理使用 PEiD 分析 lab01-03.exe

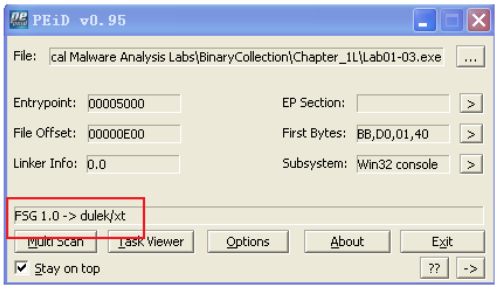


图 3.23: PEiD 对 lab01-03.exe 的加壳结果分析

可以看到提示了 FSG1.0，经过查阅资料：这是一种病毒壳或可执行文件壳，全称为"Fast Small Good"（FSG）的壳，版本为 1.0。但这次在 PEView 中没有发现像是 upx 的加壳现象。另外 uke/xt 可能是病毒作者或黑客使用的定制或特定壳的标识。**这些都可以作为被加壳的迹象**。

但此次使用 PEView 没有发现相关迹象，**关于 FSG 壳由于考虑到其并不是 upx 壳，无法使用 upx 壳脱壳。再加上手动脱去 FSG 壳的难度，这里无法再继续使用 upx 进行脱壳**。

通过继续探索，发现了一个名为 **LinxerUnpacker** 的工具，使用该工具在宿主机上成功地对其进行了脱壳：

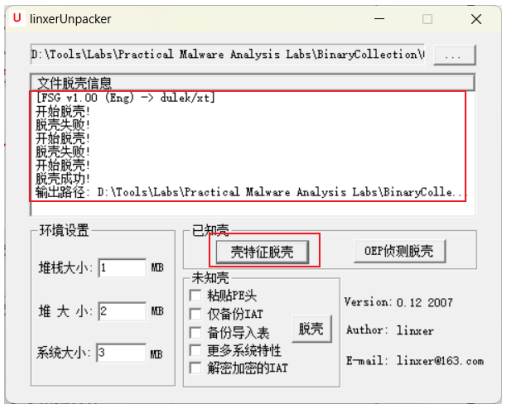


图 3.24: LinxerUnpacker 脱壳 lab01-03.exe

脱壳成功后，将其放回隔离环境 xp 系统中，使用 PEiD 查看脱壳结果，可以发现显示了和 Lab01-01.exe 相同的内容，即 VC6 编译的信息，**这证明了脱壳的成功**。

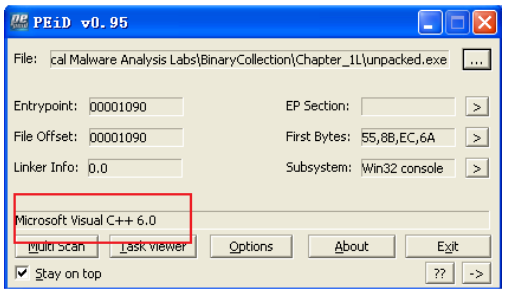


图 3.25: lab01-03.exe 脱壳结果验证

3. **Q3: 是否有导入函数显示出了这个恶意代码是做什么的？如果是，是哪些导入函数?**

   回答： 同样使用 PEiD 查看脱壳后的 lab01-03 的子系统和导入表，结果如下图所示：

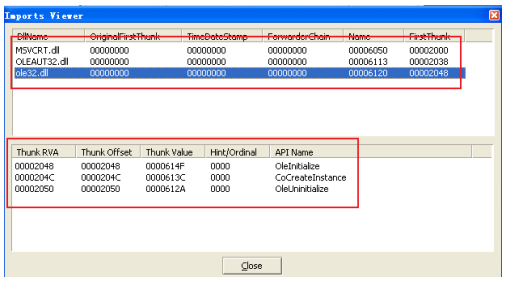


图 3.26: unpacked.exe 的导入函数

可以看到脱壳后的恶意代码主要会用到三个动态链接库，其中 MSVCRT.dll 和之前的两个病毒一样，大部分调用的函数都较常见，可能进行一些基本的内存分配等操作。重点分析 OLEAUT32.dll 和 ole32.dll:

- OLEAUT32.dll: 是 Microsoft 的 COM 组件对象模型库。COM 是一种用于构建组件化、可重用和可扩展的应用程序的技术，它允许程序通过接口与其他程序或组件进行交互。

- ole32.dll: 也与 COM 相关，提供 COM 的核心功能和接口。其中 OleInitialize 和 OleUninitialize 用于初始化和释放 COM 库的资源。CoCreateInstance 用于创建 COM 对象的实例。

综上所述：确实存在上述导入函数能够显示出病毒是做什么的，初步猜测该恶意代码可能存在创建、操纵或利用 COM 对象，用于执行特定任务、操纵系统组件与资源或隐藏自身并执行恶意操作。

4. **Q4: 是否有基于网络的迹象，可以用来发现受感染机器上的这个恶意代码?**

   回答：继续使用 PEView 进行分析脱壳后的 lab01-03.exe 即 unpacked.exe：

图 3.27: unpacked.exe 的相关字符串分析

可以发现脱壳后的恶意病毒具有着和 lab01-02 同样类似的网址字符串可以作为基于网络的迹象,https://www.malwareanaysisbook.com/ad.html 也算是彩蛋吧，可以用来发现受感染机器上的这个恶意代码。

3.5 Lab01-04

3.5.1 实验问题

1. **Q1：将文件上传至 http://www.VirusTotal.com/进行分析并查看报告。文件匹配到了已有的反病毒软件特征吗？**

   回答：继续将 lab01-04 拖入 VirusTotal 进行分析：

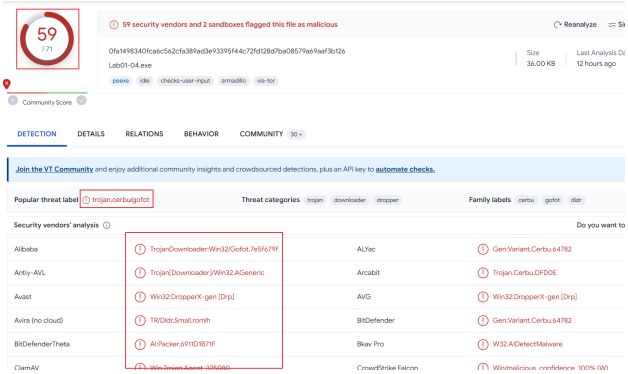


图 3.28: lab01-04.exe 的 VirusTotal 分析

可以发现这个病毒的威胁指数依然很高，足以证明其就是计算机病毒。**也同样匹配到了不少的反病毒的软件特征。**

2. **Q2：这个文件中是否存在迹象说明它是否被加壳或混淆了？如果是，这些迹象在哪里？如果该文件被加壳，如果可能的话进行脱壳。**

回答：同理使用 PEiD 对其进行分析：

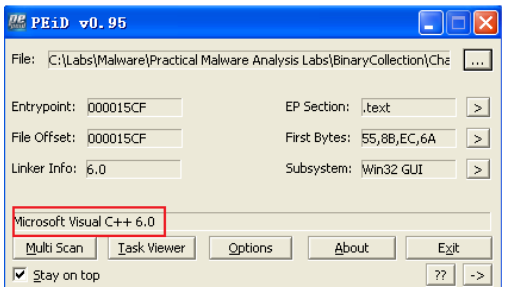
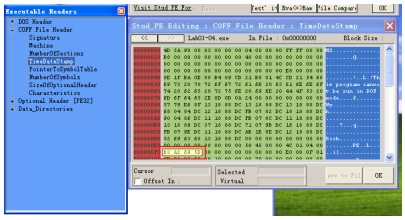


图 3.29: lab01-04.exe 的 PEiD 加壳分析

可以看到其并没有显示加壳的名称，和 lab01-01.exe 一样只显示了其是通过 VC6 编写的，**因此不存在迹象它被加壳了**。

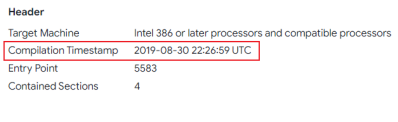
3. **Q3：这些文件是什么时候编译的?**

   回答：与对 lab01-01 中文件相同的处理方式，先通过 StudyPE 进行分析时间戳，发现其对应的十六进制表达为：5D69 A2B3，使用 python 程序将其转换为 UTC 时间为：**2019-08-30 22:26:59**。**最后将其与 VirusTotal 的结果进行对比验证了其正确性**。



TimeDateStamp 分析

时间戳转换

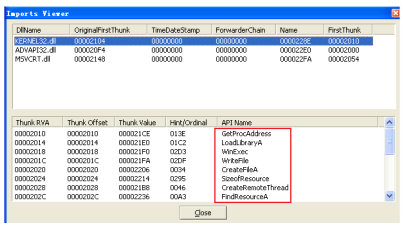


与 VirusTotal 分析结果验证

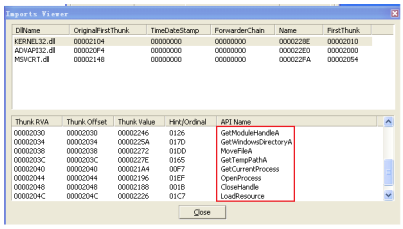
图 3.30: 对 lab01-04.dll 的编译时间分析

4. **Q4: 是否有导入函数显示出了这个恶意代码是做什么的？如果是，是哪些导入函数?**

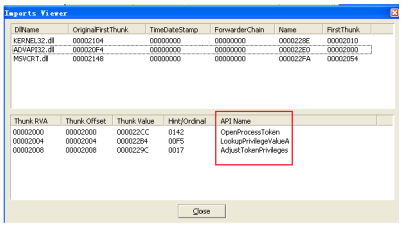
   回答：将 lab01-04 导入 PEiD 中分析其导入函数，结果如下图所示：



kernel32.dll-1



kernel32.dll-2



ADAVAPI32.dll

图 3.31: 对 lab01-04.dll 的导入函数分析

由上图可以看出，导入的动态链接库主要有 Kernel32.dll，MSVCRT.dll 还有 ADVAPI32.dll。但是 MSVCRT.dll 几乎和之前的病毒样本相同，都是一些很常见的导入函数，于是重点分析 Kernel32.dll 和 ADVAPI32.dll 两个动态链接库中函数：

- ADVAPI32.DLL：OpenProcessToken（获取指定进程的访问令牌）和 LookupPrivilegeValueA（根据权限名查找其对应的 LUID 本地唯一标识符）以及 AdjustTokenPrivileges（修改访问令牌的特权）这些导入函数。恶意代码可以尝试提升其运行的进程权限，以执行系统或敏感操作，绕过系统安全限制，或者在系统上产生更大的影响。
- Kernel32.dll：通过 FindResource 可以锁定宿主机的资源，之后使用 LoadResource 把找到的资源装载到全局存储器如磁盘等。与此同时 GetWindowsDirectory 函数可以获取 Windows 目录的完整路径名。结合之前的两个函数，这可能是获取到相应资源并存放在系统目录中了。最后调用 WinExec 函数指令相应的二进制文件。

上述这些导出函数均可以显示出恶意代码的可能行为。

5. **Q5: 是否有基于网络的迹象，可以用来发现受感染机器上的这个恶意代码?**

   回答：将 lab01-04.exe 拖入 PEView 中分析结果，如下图所示：

图 3.32: PEView 的 lab01-04.exe 的 PEView 分析结果

能发现三个字符串，winup.exe，system32 下的 wupdmgrd.exe 还有 http://www.practicalmalwareanlysis.com/updater.exe。通过查阅资料发现：wupdmgrd.exe 是 Windows 自动升级程序。判断可能利用该 exe 从该网站地址中下载了一个恶意代码，使其运行后再在 system32 的目录下创建或覆盖一个 wupdmgrd.exe 文件。

这些都是于网络的迹象，可以用来发现受感染机器上的这个恶意代码。

6. **Q6: 这个文件在资源段中包含一个资源，使用 Resource Hacker 工具来检测资源，然后抽取资源。从资源中你能发现什么？**

   回答：由于之前并没配备该工具，因此从官网下载后将其放入隔离环境虚拟机中后，将 lab01-04.exe 放入进去检测：

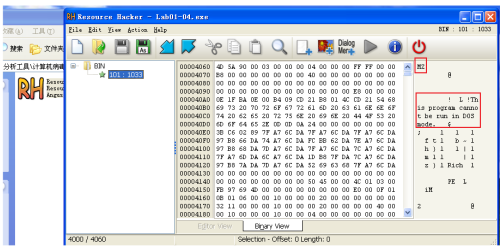
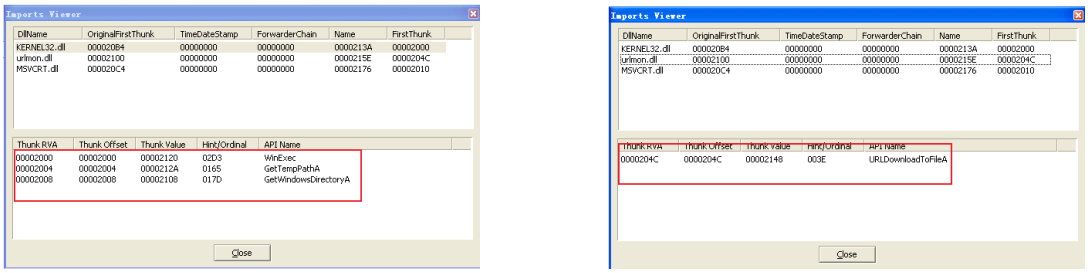


图 3.33: Resource Hacker 对 lab01-04 的检测结果

能够发现一个 MZ 头，证明其为 PE 文件，同时还有一串意思为” 该程序不能在 Windows 的 DOS 模式下运行” 的提示，应该是对使用该病毒练习者的提示。进一步提取其资源 bin 文件后，使用 PEiD 对其 bin 的导入函数进行分析：



Kernel32.dll　　urlmon

图 3.34: PEiD 对 lab01-04 的资源文件的导入关系分析

可以神奇地发现病毒资源此时的导入库中的导入的 dll 和相应的 API 函数都发生了变化。MSVCRT.dll 几乎没变化外，多出了 kernel32.dll 和 urlmon.dll 两个动态链接库，对这二者进行重点分析：

- Kernel32.dll：GetWindowsDirectoryA 和 GetTempPathA 分别获取 Windows 目录的路径和临时文件夹的路径。WinExec 作为老式的 API 函数用于运行可执行文件。
- Urlmon.dll：URLDownloadToFileA 函数用于从指定的 URL 下载文件到本地。

由这些导入函数，再结合之前的发现的网址等基于网络的迹象，可以进一步推断病毒的行为可能是通过 URL 下载其他恶意文件或更新自身的版本，然后在 Windows 中寻找到合适的目录和路径存储恶意文件或数据并执行。

3.6 Yara Detection

3.6.1 Yara 检测

基于之前的对 lab1 的四个病毒的基本静态分析，已经发现了它们都依次带有一些特殊的迹象（如基于网络的迹象），可以用来发现受感染机器上的这个恶意代码。于是编写 Yara 规则如下：

```
rule Lab1
{
meta:
   description = "Lab1:Yara Rules"
   date = "2023/9/17"
   author = "ErwinZhou"
strings:
   $clue1 = "kerne132.dll" wide ascii nocase // feature:lab01-01.exe
   $clue2 = "127.26.152.13" wide ascii // feature:lab01-01.dll
   $clue3 = "http://www.malwareanalysisbook.com" wide ascii //
       feature:lab01-02-unpacked.exe & lab01-03-unpacked.exe
   $clue4 = "wupdmgrd.exe" wide ascii // feature:lab01-04.exe
condition:
   any of them and // matching those strings
   filesize > 0KB and // To avoid error scaning zero length files
   uint16(0) == 0x5A4D and //MZ header
```

```
    uint32(uint32(0x3C)) == 0x00004550 // PE header

}
```

其中重点解释一下几个部分：

1. meta：这里是我对本次实验 Yara 规则的一些编写说明，包括编写时间和我本人的版权说明。
2. strings: 这是匹配字符串的重点部分，我使用了如下四个提示字符串，它们分别对应之前实验过程中分析得到的一些病毒签名特征：
   - lab01-01.exe：在 SECTION.data 区段其使用 kerne132.dll 替换 kernel32.dll。借由**"kerne132.dll"** 作为特征进行搜索。同时为了匹配 2 种类型的字符串与可能出现的大小写情况，添加**修饰符 wide、ascii 和 nocase**。
   - lab01-01.dll：同样在其 SECTION.data 出现了 127.26.152.13 这个 IP 地址，目的是让主机与这个 IP 地址建立连接。借由**"127.26.152.13"** 作为特征进行搜索，同时也添加修饰符 wide 与 ascii。
   - lab01-02.exe 和 lab01-03.exe：这两个病毒都分别进行了加壳，其加壳后的文件根本没有任何可以作为唯一标识的特征字符串，故这里我只找到了对脱壳后的两个病毒进行搜索的方法。也在 SECTION.data 段发现了**" https://www.malwareanalysisbook.com "**这一特征字符串，用来试图与网络连接。借由其作为特征进行搜索，也添加修饰符。
   - lab01-04.exe：由于其在 Section.data 段不存在网址迹象了这次，有一个 update.exe 的网址字符串但是在 rsrc 的 BIN 资源段，经过反复尝试没有办法通过网址字符串匹配搜索到，因此借由**"wupdmgrd.exe"** 作为特征进行搜索，同样添加了修饰符。
3. condition：条件匹配，我这里主要有两个条件，都是**为了能够更好地拟合实际 yara 规则扫描的应用场景**。
   - any of them：只要匹配到上面任何一个 clue 字符串便可以作为病毒进行识别。
   - filesize：这是由于我考虑到实际应用场景中往往从较大盘符和目录下进行扫描，而通过我的实验，经常会出现有的文件显示" zero length file "即文件大小为 0 的特殊情况，由于这种情况肯定不是病毒，因此将其排除。**故对文件大小加以限制**。
   - 由于我的 yara 规则 Lab1 文件也出现了上述字符串并且满足文件大小，并且和目标病毒在同一目录下。**故为了将类似的情况排除，我考虑目标的病毒文件通常都尉.exe 和.dll 两种格式，便通过 MZ 头和 PE 头通过定位将 PE 格式的文件锁定，排除可能出现的其他情况**。

最后，为了能够方便进行后续的优化时间性能的对比，我使用宿主机的 python 借助 jupyter notebook 的 yara 包来进行规则匹配搜索，目标的病毒们所在路径为 D 盘的 Labs 目录下，故以其作为起点，相关代码如下：

```
import os
import yara
import time

rules = yara.compile('D:\\Tools\\Virus Detection\\Yara\\yara64\\Lab1')
```

```python
def scan_folder(folder_path):
    if os.path.exists(folder_path) and os.path.isdir(folder_path):
        for root, dirs, files in os.walk(folder_path):
            for filename in files:
                file_path = os.path.join(root, filename)
                with open(file_path, 'rb') as file:
                    data = file.read()
                    matches = rules.match(data=data)
                    if matches:
                        print(f"File '{filename}' in path '{root}' matched YARA
                            rule(s):")
                        for match in matches:
                            print(f"Rule: {match.rule}")
    else:
        print(f'The folder at {folder_path} does not exist or is not a folder.')

start_time = time.time()
folder_path = 'D:\\Tools'
scan_folder(folder_path)
end_time = time.time()
runtime = end_time - start_time
print(f"Program runtime: {runtime} seconds.")
```

结果如下图所示，可以看到成功地找到了目标的 Chapter1 中的病毒，同时有一些别的章的病毒由于具有类似的特征也被检测了出现，**同时也看到了对应的时间来体现效率，证明了 Yara 检测的成功**。

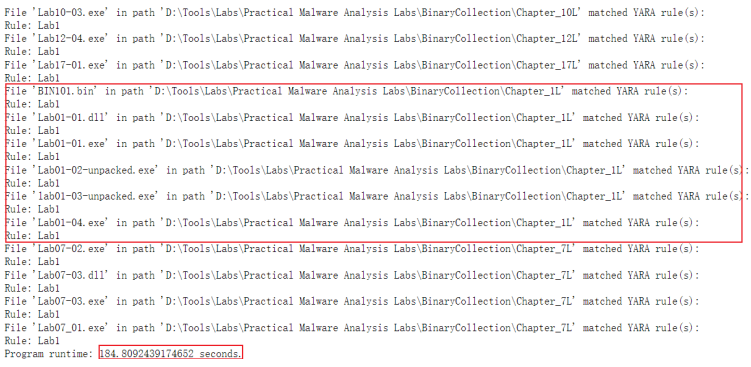


图 3.35: Yara 匹配结果

3.6.2 Yara 规则优化探究与讨论

通常来说有着如下的技巧关于如何编写更加快速的 yara 规则：

1. **精确定位目标字符串：**尽量使用特定的、唯一的标识字符串进行匹配，以避免误报和提高匹配速度。避免使用过于通用的字符串，可以通过特定的文件格式、结构或功能来精确定位目标；避免使用通配符和正则表达式。**比如本次规则编写就没有使用通配符等，而是通过唯一的字符串**。

2. **避免不必要的字符串比较:** 可以通过文件头、特定位置或唯一标识来快速排除不匹配的文件，减少不必要的字符串比较。**这提示我由于所有字符串几乎都出现在 SECTION.data 字段，故可以提前对该位置进行定位来提高性能，有待我进一步探究**。。

3. **优化规则结构:** 将最可能匹配的条件放在规则的前面，以便尽早匹配成功并减少后续比较。使用 all of 或 any of 组合条件，以避免不必要的比较。**比如本次规则就使用了 any 并把最重要的字符串匹配放在了最前面**。

4. **利用元数据和规则修饰符:** 使用 nocase、ascii 等规则修饰符，适当情况下可以提高匹配速度。添加元数据以提供规则的描述和作者信息。**本次也使用了这个技巧**。

5. **限制字符串长度:** 尽量避免在规则中使用过长的字符串，以减少匹配时的比较长度。**这提示我可以匹配较长的网址迹象的标志性部分，比如 https://和 www. 等减少匹配时的比较长度，有待我进一步探究**。。

6. **分割规则:** 将大规模规则拆分为多个小规模规则，分别针对不同的目标或特征。这样可以提高规则的匹配效率。**本次通过匹配 PE 文件格式用到了这个技巧**。

更多的规则技巧相信随着我的不断锻炼和提升能力，会发掘出更多有效的技巧。

4 实验结论及心得体会

4.1 实验结论

本次实验，我使用了 VirusTotal、PEView，PEiD,Study_PE，upx，LinxerUnpacker 等常用病毒分析工具，依次对第一章的四组病毒样本进行了静态分析，利用字符串、时间戳、导入函数关系、加壳与混淆等上课学到的静态分析技巧成功地对这些病毒进行了刨析并回答了对应的问题。与此同时，我还编写了相应的 Yara 规则对这些病毒进行匹配识别，思考并讨论如何更进一步地提高 Yara 规则匹配的性能。**总的来说，实验非常成功**。

4.2 心得体会

本次实验，由于我是第一次接触本门课程和计算机病毒分析领域，当我将上课王老师传授的静态分析技巧和 Yara 规则编写方式运用在实践中，我收获了无比多的经验，也有着非常充实和满足的体验。总的来说：

1. 我掌握了在这个过程中通过自己搭建隔离的虚拟机 SandBox 进行病毒分析环境的搭建，为我未来学习和工作打下了基础；

2. 我学到了如何使用借助那些基本的静态病毒分析工具结合我们的知识进行对病毒的一些特征进行分析，脱壳，并借助 dll 和导入函数等信息来分析恶意代码的目的；

3. 我巩固了很多关于动态链接库、PE 文件格式等基本的静态分析技巧和知识以及一些系统函数，这让我对上课学到的知识掌握更加熟练。

4. 我熟练了如何使用 yara 规则进行匹配和更进一步对其性能进行优化，这也有助于我后续的学习。

总的来说，本次通过亲自实验让我收获颇丰，也收获了对安全领域计算机病毒分析领域的浓厚兴趣，之后我会更加努力学习，精进自己的能力，丰富自己的知识。