

南开大学

恶意代码分析与防治技术实验报告

实验一：基本静态分析技术与 Yara 规则编写



学 院 网络空间安全学院
专 业 信息安全、法学
学 号 2112514
姓 名 辛浩然
班 级 信息安全、法学

一、实验目的

1. 加深对静态分析方法的认识，通过对恶意代码进行静态分析，将静态分析基础技术的理论付诸实践，提高不同恶意代码文件的静态分析能力；
2. 检查文件是否具有反病毒软件特征；
3. 掌握恶意代码文件是否被加壳或混淆的判断方法，及掌握基本的脱壳方法；
4. 学会通过导入函数或其他迹象分析恶意代码的功能；
5. 熟悉 PEiD、Strings、IDA Pro、PEView 等基本分析工具的使用；
6. 掌握并熟练 Yara 规则的编写原理、方法。

二、实验原理

本次实验基于静态分析基础技术对恶意代码文件进行分析。

在分析一个可疑的恶意代码样本时，第一步是**反病毒引擎扫描**。类似 VirusTotal 的网站调用多个反病毒引擎来进行扫描用户上传的文件，并会生成一份报告，其中提供了所有引擎对这个样本的识别情况、标识这个样本是否恶意、恶意代码名称，以及其他额外信息。

从字符串中进行搜索是获得程序功能提示的一种简单方法。比如，如果程序访问了一个 URL，访问的 URL 就存储为程序中的一个字符串。可以使用 Strings 程序来搜索可执行文件中的可打印字符串。这些字符串通常是以 ASCII 或 Unicode 格式进行存储的。

恶意代码编写者经常使用**加壳或混淆技术**，让恶意代码文件更难被检测或分析。混淆程序是恶意代码编写者尝试去隐藏其执行过程的代码。而加壳程序则是混淆程序中的一类，加壳后的恶意程序会被压缩，并且难以分析。合法程序大多总是会包含很多字符串，而由被加壳或者混淆的恶意代码直接分析获取到的可打印字符串则很少。加壳和混淆代码通常至少会包含 LoadLibrary 和 GetProcAddress 函数，它们是用来加载和使用其他函数功能的。当加壳的程序运行时，会首先运行一小段脱壳代码，来解压缩加壳的文件，然后再运行脱壳后的文件。当对一个加壳程序进行静态分析时，只有这一小段脱壳代码可以被解析。

检测加壳软件的一种方法是使用 PEiD 工具。可以使用 PEiD 来检测加壳器的类型，或是用来链接应用程序的编译器类型，这使得分析加壳软件变得更加容易一些。

当一个程序被加壳后，必须对它进行**脱壳**，才能够执行进一步分析。脱壳过程往往是很复杂的。但是 UPX 加壳工具非常流行，也非常容易进行脱壳处理。

文件格式可以揭示出很多关于程序功能的信息。可移植执行(PE)文件格式是 Windows 可执行文件、对象代码和 DLL 所使用的标准格式。PE 文件以一个文件头开始，其中包括代码信息、应用程序类型、所需的库函数与空间要求。**PE 头**中的信息对于恶意代码分析而言，是非常有价

值的。

PE 文件头中存储了每个将被装载的库文件，以及每个会被程序使用的函数信息。**程序所使用的库与调用的函数**，经常是一个程序中最重要的一部分，识别它们尤为重要，因为这些信息允许我们来猜测这个恶意代码样本到底干了些什么事情。

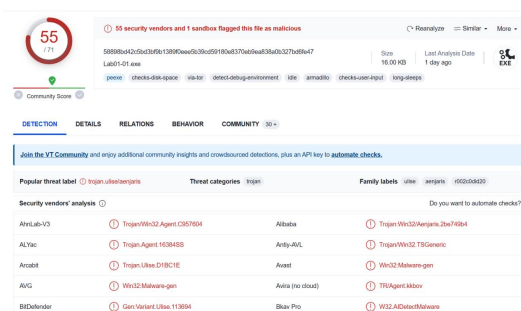
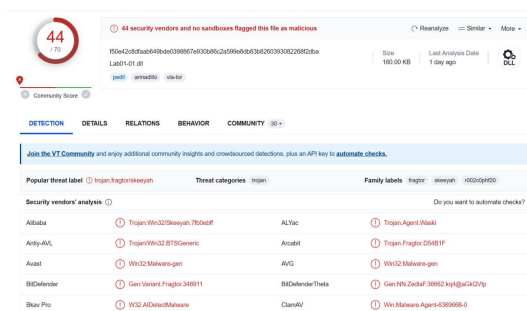
在对恶意代码样本分析后，可以编写匹配的 Yara 规则。Yara 规则是一种用于检测和分类恶意软件样本的强大工具，它可以根据特定的规则来匹配文件或内存中的数据。

三、实验过程

Lab 1-1

1.1.1 Upload the files to <http://www.VirusTotal.com/> and view the reports. Does either file match any existing antivirus signatures?

上传 Lab01-01.dll，检测结果如下图左图所示：



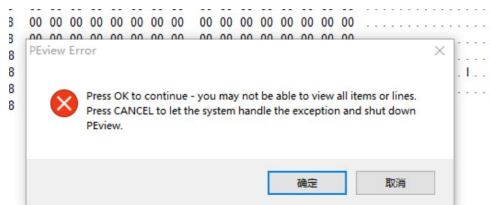
可以发现，匹配到了已有的反病毒特征，已经有 44 个安全公司将该文件标记为病毒。在报告提供的具体的分析中也列举了热门的威胁标签及威胁种类。

上传 Lab01-01.exe，检测结果如上图右图所示：

同样可以发现，匹配到了已有的反病毒特征，已经有 55 个安全公司和 1 个沙箱将该文件标记为病毒。

1.1.2 When were these files compiled?

在 PEView 中打开 exe 文件，解析文件结构时会产生报错，无法查看：

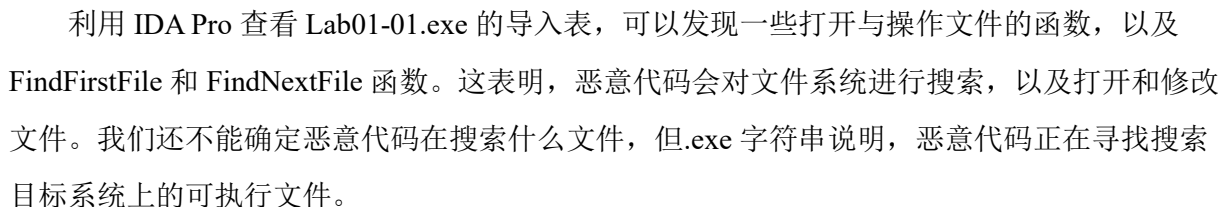


因此，使用补丁版的 PEView 查看文件。文件的 IMAGE_NT_HEADERS -> IMAGE_FILE_HEADER -> Time Date Stamp 字段给出了文件的编译时间。

Lab01-01.exe 的编译时间为 2010/12/19 16:16:19；Lab01-01.dll 的编译时间为：2010/12/19 16:16:38，二者编译时间非常相近。猜测这两个文件是关联的，.exe 文件可能是用以使用或安装.dll

[illegible]

在 PEid 工具中打开.dll 文件和.exe 文件,二者都没有发现加壳信息,且是由 Microsoft Visual C++ 6.0 编译的。



查看 Lab01-01.dll 的导入表：可以发现，从 kernel32.dll 导入 CreateProcess 和 Sleep 函数，这两个函数普遍在后门程序中使用。还导入了 WS2_32.dll 中的一些函数，这提供了联网功能。

1.1.5 Are there any other files or host-based indicators that you could look for on infected systems?

利用 strings 工具打开 Lab01-01.exe，同时发现文件 kernel.dll 和 knernl1.dll，可知恶意代码用数字 1 代替了字母 l 创建了新文件，新文件看起来像是系统文件 kernel32.dll，试图混淆。

```
C:\Windows\System32\cmd.exe
exit
XcptFilter
p__initenv
__getmainargs
__initterm
__setusermatherr
__adjust_fdiv
p__commode
p__fmode
__set_app_type
__except_handler3
__controlfp
__stricmp
kernel32.dll
kernel32.dll
.exe
C:\*
C:\windows\system32\kernel32.dll
Kernel32.
Lab01-01.dll
C:\Windows\System32\Kernel32.dll
WARNING_THIS_WILL_DESTROY_YOUR_MACHINE
C:\Users\XHR\Desktop\Strings>
```

1.1.6 What network-based indicators could be used to find this malware on infected machines?

利用 strings 工具查看 Lab01-01.dll 的字符串信息，文件中包含一个私有子网 IP 地址 127.26.152.13 的字符串。如果这是真正的恶意代码，这个 IP 地址应该是可路由的公网 IP 地址，它会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。

```
__initterm
malloc
__adjust_fdiv
exec
sleep
hello
127.26.152.13
SADFHUHF
4010[0b0b0
```

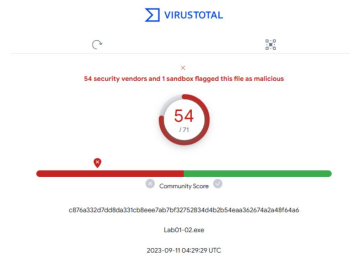
1.1.7 What would you guess is the purpose of these files?

在前面分析 Lab01-01.dll 的导入表时，发现从 kernel32.dll 导入 CreateProcess 和 Sleep 函数，因此猜测 Lab01-01.dll 可能是一个后门，而 Lab01-01.exe 是用来安装与运行 dll 文件的。

Lab 1-2

1.2.1 Upload the Lab01-02.exe file to <http://www.VirusTotal.com/>. Does it match any existing antivirus definitions?

将 Lab01-02.exe 上传到网站，匹配到了已有的反病毒特征，已经有 54 个安全公司和 1 个沙箱将该文件标记为病毒。



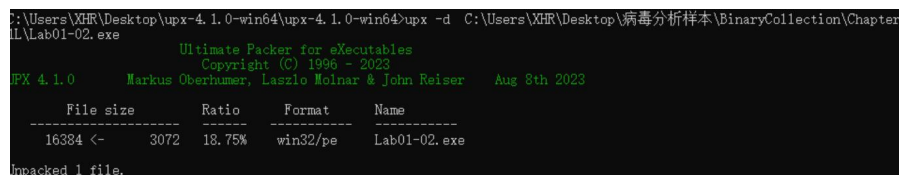
1.2.2 Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.

利用 PEiD 加壳检测工具打开，可以发现已经 UPX 加壳。

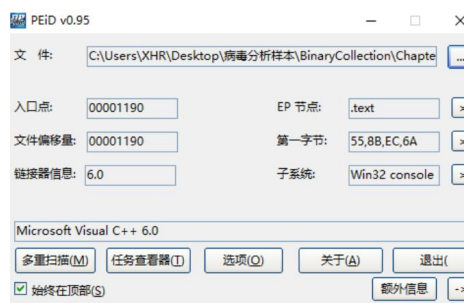


UPX0 段虚拟大小为 0x4000，而原始数据大小却为 0。UPX0 是长度最大的节，标记为可执行的，因此其中很可能包含了原始的未加壳代码。

利用 UPX 工具进行脱壳：

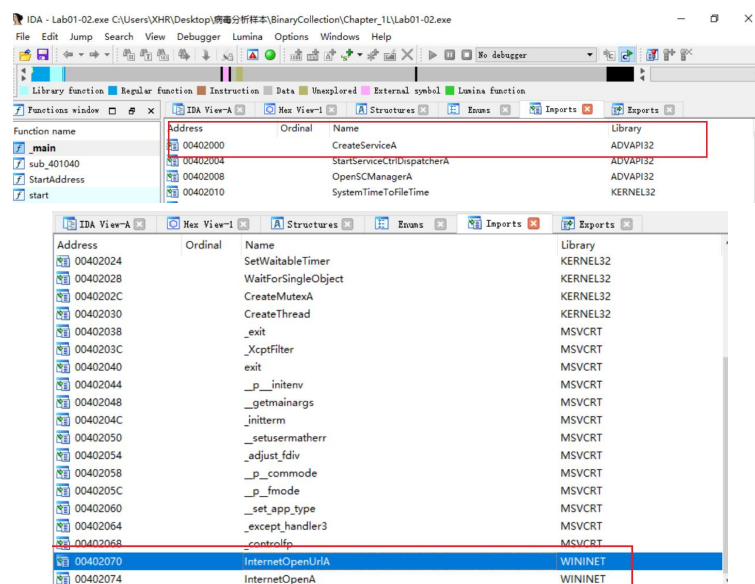


脱壳后，可以发现文件是 Microsoft Visual C++ 6.0 编译的，说明脱壳成功。



1.2.3 Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?

利用 IDA Pro 打开 exe 文件，查看导入表，可以发现，恶意代码在 wininet.dll 中导入了 InternetOpen 和 InternetOpenURL 等函数，恶意代码会进行联网操作；在 advapi32.dll 中导入了 CreateService 等函数，会创建一个服务。



1.2.4 What host- or network-based indicators could be used to identify this malware on infected machines?

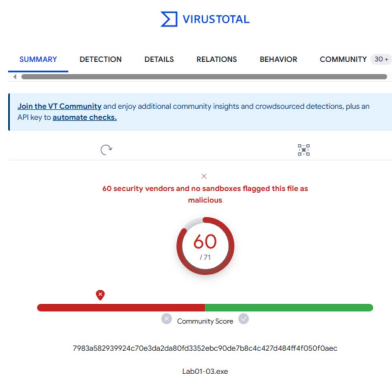
利用 strings 工具查看字符串列表，发现了 Malservice 字符串，猜测创建了 Malservice 服务。还发现了 <http://www.malwareanalysisbook.com> 字符串，这可能是 InternetOpenURL 函数中所打开的 URL。可以通过 Malservice 的服务，并通过到 <http://www.malwareanalysisbook.com> 的网络流量，来检查被恶意代码感染的主机。

```
%d @
%h @
KERNEL32.DLL
ADVAPI32.dll
MSVCRT.dll
WININET.dll
SystemTimeToFileTime
GetModuleFileNameA
CreateWaitableTimerA
ExitProcess
OpenMutexA
SetWaitableTimer
WaitForSingleObject
CreateMutexA
CreateThread
CreateServiceA
StartServiceCtrlDispatcherA
OpenSCManagerA
_exit
_XcptFilter
_exit
_p__initenv
__getmainargs
_initterm
__setusermatherr
_adjust_fdiv
_p__commode
_p__fmode
_set_app_type
_except_handler3
_controlfp
InternetOpenUrlA
InternetOpenA
Malservice
Malservice
HGL345
http://www.malwareanalysisbook.com
Internet Explorer 8.0
```

Lab 1-3

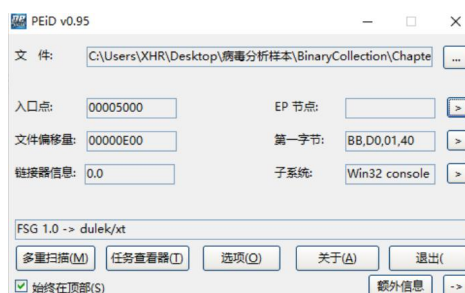
1.3.1 Upload the Lab01-03.exe file to <http://www.VirusTotal.com/>. Does it match any existing antivirus definitions?

将 Lab01-03.exe 上传到网站，匹配到了已有的反病毒特征，已经有 60 个安全公司将该文件标记为病毒。



1.3.2 Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.

在 PEiD 检测工具中打开该文件，发现经过了 FSG 加壳。



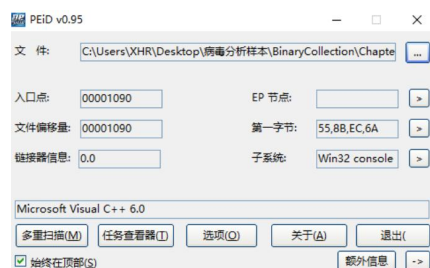
且可以看到它有一个导入表，但其中只有 LoadLibrary 和 GetProcAddress 两个函数。加壳文件往往只有这两个导入函数。

Address	Ordinal	Name	Library
00405128		LoadLibraryA	KERNEL32
0040512C		GetProcAddress	KERNEL32

利用万能脱壳工具对其进行脱壳：



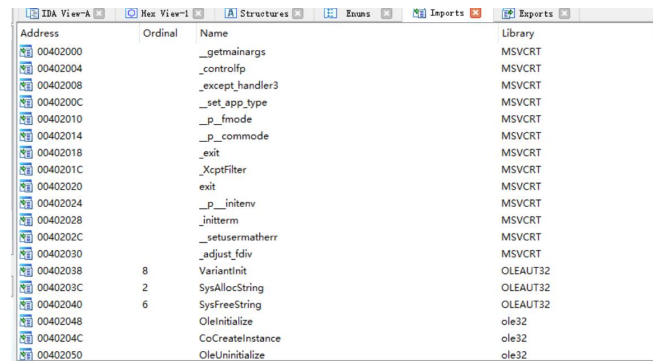
在 PEiD 打开脱壳后的文件，可以发现是由 Microsoft Visual C++ 6.0 编译的。成功脱壳。



1.3.3 Do any imports hint at this program's functionality? If so, which imports are they and

what do they tell you?

在 IDA Pro 中打开脱壳后的文件，查看导入表。

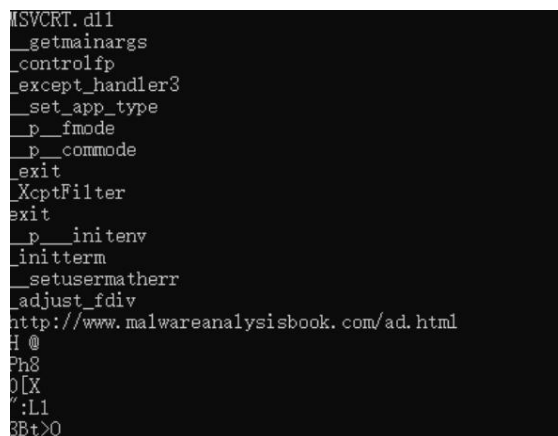


Address	Ordinal	Name	Library
00402000		__getmainargs	MSVCRT
00402004		__controlfp	MSVCRT
00402008		_except_handler3	MSVCRT
0040200C		__set_app_type	MSVCRT
00402010		__p_fmode	MSVCRT
00402014		__p_commode	MSVCRT
00402018		_exit	MSVCRT
0040201C		_XcptFilter	MSVCRT
00402020		exit	MSVCRT
00402024		__p_initenv	MSVCRT
00402028		_initterm	MSVCRT
0040202C		_setusermatherr	MSVCRT
00402030		_adjust_fdiv	MSVCRT
00402038	8	VariantInit	OLEAUT32
0040203C	2	SysAllocString	OLEAUT32
00402040	6	SysFreeString	OLEAUT32
00402048		OleInitialize	ole32
0040204C		CoCreateInstance	ole32
00402050		OleUninitialize	ole32

发现在 ole.dll 中导入了 CoInitialize 和 CoCreateInstance 等函数。Ole.dll 是链接和嵌入在应用程序中的对象的过程文件。它是用于编写和整合来自不同应用程序的数据在 Windows 作业系统的骨干部分。CoInitialize 和 CoCreateInstance 是两个在 Windows 平台上用于初始化 COM (Component Object Model) 环境并创建 COM 对象的重要函数。COM 是一种用于组件化开发的技术，它允许不同的软件组件相互协作。恶意代码可以利用这两个函数来创建和控制 COM 对象实例，这可能导致恶意对象与系统中其他 COM 组件相互交互，从而实施攻击，如数据窃取、篡改或破坏系统功能。

1.3.4 What host- or network-based indicators could be used to identify this malware on infected machines?

查看脱壳后 exe 文件的字符串，发现一个网址，猜测可能是通过网络感染。可以检测该网址的网络流量，来检查恶意代码感染的主机。

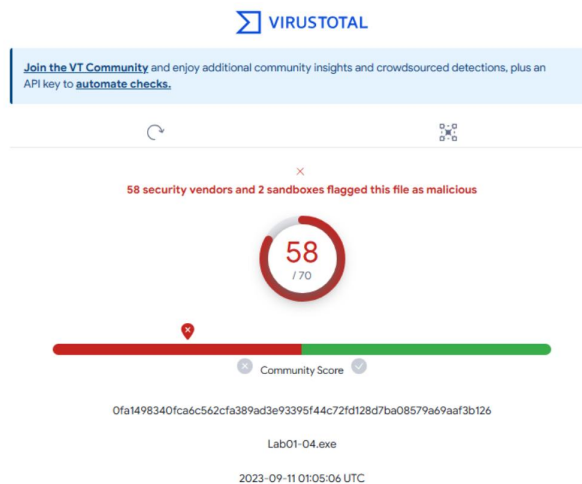


```
MSVCRT.dll
__getmainargs
__controlfp
_except_handler3
__set_app_type
__p_fmode
__p_commode
_exit
_XcptFilter
exit
__p_initenv
_initterm
_setusermatherr
_adjust_fdiv
http://www.malwareanalysisbook.com/ad.html
H@
Ph8
0[X
":L1
3Bt>0
```

Lab 1-4

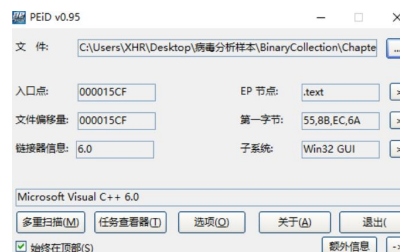
1.4.1 Upload the Lab01-04.exe file to <http://www.VirusTotal.com/>. Does it match any existing antivirus definitions?

将 Lab01-04.exe 上传到网站，匹配到了已有的反病毒特征，已经有 58 个安全公司和 2 个沙箱将该文件标记为病毒。



1.4.2 Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.

在 PEiD 中打开该文件，没有迹象显示该文件是被混淆或加密过的。



1.4.3 When was this program compiled?

在 PEViewPatched 中打开该文件，可以发现，该程序是在 2019/8/30 22:26:59 编译的。

pFile	Data	Description	Value
000000EC	014C	Machine	IMAGE_FILE_MACHINE_I386
000000EE	0004	Number of Sections	
000000F0	5D69A2B3	Time Date Stamp	2019/08/30 22:26:59 UTC
000000F4	00000000	Pointer to Symbol Table	
000000F8	00000000	Number of Symbols	
000000FC	00E0	Size of Optional Header	
000000FE	010F	Characteristics	
	0001		IMAGE_FILE_RELOCS_STRIPPED
	0002		IMAGE_FILE_EXECUTABLE_IMAGE
	0004		IMAGE_FILE_LINE_NUMS_STRIPPED
	0008		IMAGE_FILE_LOCAL_SYMS_STRIPPED
	0100		IMAGE_FILE_32BIT_MACHINE

1.4.4 Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?

利用 IDA Pro 查看文件的导入表，发现从 advapi32.dll 中导入函数，说明程序在做一些与权限有关的事情。导入函数 WinExec 和 WriteFile 说明程序会写一个文件到磁盘上，然后执行它。导入函数还有一些是用于从文件的资源节中读取信息的。

Address	Ordinal	Name	Library
00402000		OpenProcessToken	ADVAPI32
00402004		LookupPrivilegeValueA	ADVAPI32
00402008		AdjustTokenPrivileges	ADVAPI32
00402010		GetProcAddress	KERNEL32
00402014		LoadLibraryA	KERNEL32
00402018		WinExec	KERNEL32
0040201C		WriteFile	KERNEL32
00402020		CreateFileA	KERNEL32
00402024		SizeofResource	KERNEL32
00402028		CreateRemoteThread	KERNEL32
0040202C		FindResourceA	KERNEL32
00402030		GetModuleHandleA	KERNEL32
00402034		GetWindowsDirectoryA	KERNEL32
00402038		MoveFileA	KERNEL32
0040203C		GetTempPathA	KERNEL32
00402040		GetCurrentProcess	KERNEL32
00402044		OpenProcess	KERNEL32
00402048		CloseHandle	KERNEL32
0040204C		LoadResource	KERNEL32

1.4.5 What host- or network-based indicators could be used to identify this malware on infected machines?

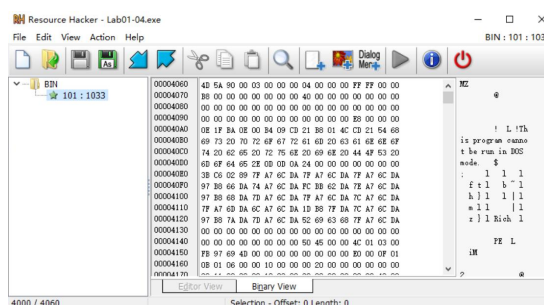
查看文件的字符串，字符串\system32\wupdmgr.exe 表示，这个程序会在这个位置创建或者修改文件。字符串 <http://www.malwareanalysisbook.com/updater.exe> 可能表示额外恶意代码的网络存储位置，用于下载。

```

_controlfp
\winup.exe
%*%
\system32\wupdmgrd.exe
%*%
http://www.practicalmalwareanalysis.com/updater.exe

```

1.4.6 This file has one resource in the resource section. Use Resource Hacker to examine that resource, and then use it to extract the resource. What can you learn from the resource?



使用 Resource Hacker 工具打开这个恶意代码，看到了一个资源。Resource Hacker 工具识别这个资源的类型是二进制，说明是任意的二进制数据。发现字符串 This program cannot be run in DOS mode，这个字符串是在所有 PE 文件开始处的 DOS 头部中都会包含的错误。因此，该资源可能是在 Lab01-04.exe 资源节中存储的另一个可执行文件。

Address	Ordinal	Name	Library
00402000		WinExec	KERNEL32
00402004		GetTempPathA	KERNEL32
00402008		GetWindowsDirectoryA	KERNEL32
00402010		_controlfp	MSVCRT
00402014		_snprintf	MSVCRT
00402018		_exit	MSVCRT
0040201C		_XcptFilter	MSVCRT
00402020		exit	MSVCRT
00402024		_p__initenv	MSVCRT
00402028		_getmainargs	MSVCRT
0040202C		_initterm	MSVCRT
00402030		_setusermatherr	MSVCRT
00402034		_adjust_fdiv	MSVCRT
00402038		_p__commode	MSVCRT
0040203C		_p__fmode	MSVCRT
00402040		_set_app_type	MSVCRT
00402044		_except_handler3	MSVCRT
0040204C		URLDownloadToFileA	urlmon

将资源保存为二进制文件，然后在 IDA Pro 中打开。查看导入表，看到嵌入文件在访问一

些网络函数，它调用了 URLDownloadToFile 函数，一个由恶意下载器普遍使用的函数，还调用了 WinExec 函数，可能还执行了下载到的文件。

Lab 1-5

1.5.1 编写 Yara 规则：

1.5.1.1 针对 Lab01-01.dll

结合前面的分析可知，Lab01-01.dll 包含一个私有子网 IP 地址 127.26.152.13 的字符串。

Lab01-01.dll 从 kernel32.dll 导入 CreateProcess 和 Sleep 函数，因此猜测 Lab01-01.dll 可能是一个后门。

结合利用 strings 工具查看到的包含的字符串，我们制定以下规则：

```
rule Lab01_01_dll {
  meta:
    description = "This is for Lab01-01.dll"
  strings:
    $string1 = "127.26.152.13" fullword ascii
    $string2 = "sleep" fullword ascii
    $string3 = "CreateProcessA" fullword ascii
    $string4 = "exec" fullword ascii
  condition:
    uint16(0) == 0x5A4D and // MZ 头
    uint32(uint32(0x3C)) == 0x00004550 and // PE 头
    filesize < 10MB and
    all of them
}
```

在条件中，首先判断 PE 文件的条件，然后为了增加扫描速度，限制文件大小为小于 10MB，设定匹配所有设定的字符串。

在病毒样本 Chapter_1L 目录下进行扫描，成功扫描出 Lab01-01.dll：

```
C:\Users\XHR\Desktop\yara-4.3.2-2150-win64>yara64 lab01_01_dll.yar Chapter_1L
Lab01_01_dll Chapter_1L\Lab01-01.dll
```

1.5.1.2 针对 Lab01-01.exe

结合前面的分析，Lab01-01.exe 包含 FindFirstFile 和 FindNextFile 等对文件系统操作的函数。查看其包含的字符串，同时发现 kernel.dll 和 knernl1.dll。其字符串也包含 Lab01-01.dll，因为.exe 文件可能是用以使用或安装.dll 文件的。

因此，制定规则如下：

```
rule Lab01_01_exe {
```

```

meta:
  description = "This is for Lab01-01.exe"
strings:
  $string1 = "kernel32.dll" fullword
  $string2 = "kernel32.dll" fullword
  $string3 = "C:\\windows\\system32\\kernel32.dll" fullword
  $string4 = "C:\\Windows\\System32\\Kernel32.dll" fullword
  $string5 = "Lab01-01.dll" fullword
  $string6 = "FindNextFileA" fullword
  $string7 = "FindFirstFileA" fullword
  $string8 = "CopyFileA" fullword
condition:
  uint16(0) == 0x5A4D and
  uint32(uint32(0x3C)) == 0x00004550 and
  filesize < 10MB and
  all of them
}

```

在条件中，同样首先判断 PE 文件的条件，然后限制文件大小为小于 10MB，并设定匹配所有设定的字符串。

在病毒样本 Chapter_1L 目录下进行扫描，成功扫描到 Lab01-01.exe:

```

C:\Users\XHR\Desktop\yara-4.3.2-2150-win64>yara64 lab01_01_exe.yar Chapter_1L
Lab01_01_exe Chapter_1L\Lab01-01.exe

```

1.5.1.3 针对 Lab01-02.exe

首先查看 Lab01-02.exe 的字符串:

01	0P
PQ6	PTj
Yn	XPTPSW
e)	u
(23h	KERNEL32.DLL
MalService	ADVAPI32.dll
sHGL345	MSVCRT.dll
http://w	WININET.dll
warean	LoadLibraryA
ysisbook.co	GetProcAddress
om#Int6net Explo!r 3FEI	VirtualProtect
.0<	VirtualAlloc
SystemTimeToFile	VirtualFree
GetMo	ExitProcess
NaA	CreateServiceA
Cvg	exit
*Waitab'r	InternetOpenA
Process	

在 Lab01-02.exe 的字符串中，可以发现 CreateServiceA，用于创建服务；InternetOpenA，用于联网操作。

由于进行了加壳操作，所以相对于前面对脱壳后的文件的字符串分析，一些字符串发生了一些变化，比如 url 地址。

使用 HexView 打开 Lab01-02.exe，查看到被处理后的 url 地址，得到其 16 进制表示：

00000600	72 76 69 63 65 FF DB 3F A4 73 48 47 4C 33 34 35	ervice ? HGL345
00000610	68 74 74 70 3A 2F 2F 77 FF B7 BF DD 00 2E 6D	.http://w.房?m
00000620	1E 77 61 72 65 61 6E 07 79 73 69 73 62 6F 6F 6B	.warean.ysisbook
00000630	2E 63 6F FF DB DB 6F 6D	.co.坊om#Int6net
00000640	20 45 78 70 6C 6F 21 72 20 38 46 45 49 C7 2E 30	Explo!r 8FEI?0
00000650	3C 01 20 01 A0 C0 09 65 73 14 15 98 10 10 BF 9D	<.犁.es..?.縉
00000660	FF FF 01 53 79 73 74 65 6D 54 69 6D 65 54 6F 46	...SystemTimeToF
00000670	69 6C 65 15 47 65 74 4D 6F C1 B6 FC DA 64 75 0E	ile.GetMo练 du.

因此，编写规则如下：

```
rule Lab01_02_exe {
  meta:
    description = "This is for Lab01-02.exe"
  strings:
    $string1 = "MalService" fullword ascii
    $string2 = "InternetOpenA" fullword ascii
    $string3 = "CreateServiceA" fullword ascii
    $hex_com = {68 74 74 70 3A 2F 2F 77 FF B7 BF DD 00 2E 6D 1E 77 61 72 65 61 6E 07 79 73 69 73 62 6F 6F 6B 2E 63 6F FF DB DB 6F 6D}
  condition:
    uint16(0) == 0x5A4D and
    uint32(uint32(0x3C)) == 0x00004550 and
    filesize < 10MB and
    all of them
}
```

在条件中，同样首先判断 PE 文件的条件，然后限制文件大小为小于 10MB，并设定匹配所有设定的字符串。

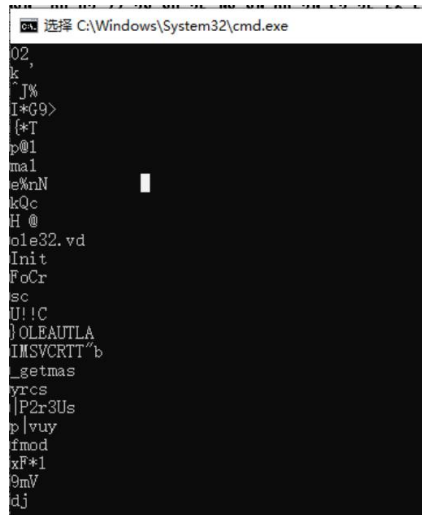
在病毒样本 Chapter_1L 目录下进行扫描，成功扫描到 Lab01-02.exe：

```
C:\Users\XHR\Desktop\yara-4.3.2-2150-win64>yara64 lab01_02_exe.yar Chapter_1L
Lab01_02_exe Chapter_1L\Lab01-02.exe
```

1.5.1.4 针对 Lab01-03.exe

结合前面的分析，Lab01-03.exe 进行了 FSG 加壳处理。在对脱壳后的文件进行了分析，从 ole.dll 中的 CoInitialize 和 CoCreateInstance 等函数，字符串中包含一个网址。

但是对于脱壳前的文件分析，得到一些破碎的字符串，因此选择可能代表其特征的破碎字符串进行匹配。



编写规则如下:

```
rule Lab01_03_exe {
  meta:
    description = "This is for Lab01-03.exe"
  strings:
    $string1 = "ole32.vd" fullword ascii
    $string2 = "}OLEAUTLA" fullword ascii
    $string3 = "_getmas" fullword ascii
    $string4 = "fmod" fullword ascii
    $string5 = "|P2r3Us" fullword ascii
    $string6 = "p|vuy" fullword ascii
  condition:
    uint16(0) == 0x5A4D and
    uint32(uint32(0x3C)) == 0x00004550 and
    filesize < 10MB and
    all of them
}
```

在病毒样本 Chapter_1L 目录下进行扫描, 成功扫描到 Lab01-03.exe:

```
C:\Users\XHR\Desktop\yara-4.3.2-2150-win64>yara64 lab01_03_exe.yar Chapter_1L
Lab01_03_exe Chapter_1L\Lab01-03.exe
```

1.5.1.5 针对 Lab01-04.exe

结合前面的分析, 恶意代码发现从 `advapi32.dll` 中导入函数, 说明程序在做一些与权限有关的事情。导入函数 `WinExec` 和 `WriteFile` 说明程序会写一个文件到磁盘上, 然后执行它。字符串 `\system32\wupdmgr.exe` 表示, 这个程序会在这个位置创建或者修改文件。字符串 `http://www.malwareanalysisbook.com/updater.exe` 可能表示额外恶意代码的网路存储位置, 用于下载。

Lab01-04.exe 资源节中存储了另一个可执行文件，它调用了 URLDownloadToFile 函数，一个由恶意下载器普遍使用的函数，还调用了 WinExec 函数，可能还执行了下载到的文件。

因此，编写规则如下：

```
rule Lab01_04_exe {
  meta:
    description = "This is for Lab01-04.exe"
  strings:
    $string1 = "\\system32\\wupdmgr.exe" fullword ascii
    $string2 = "http://www.practicalmalwareanalysis.com/updater.exe" fullword ascii
    $string3 = "WinExec" fullword ascii
    $string4 = "ADVAPI32.dll" fullword ascii
    $string5 = "URLDownloadToFileA" fullword ascii
  condition:
    uint16(0) == 0x5A4D and
    uint32(uint32(0x3C)) == 0x00004550 and
    filesize < 10MB and
    all of them
}
```

在病毒样本 Chapter_1L 目录下进行扫描，成功扫描出 Lab01-04.exe：

```
C:\Users\XHR\Desktop\yara-4.3.2-2150-win64>yara64 lab01_04_exe.yar Chapter_1L
Lab01_04_exe Chapter_1L\Lab01-04.exe
```

1.5.1.6

对之前编写的规则整合到 Lab01.yar 中，扫描 Chapter_1L 文件，成功扫描出每个规则相应的病毒：

```
C:\Users\XHR\Desktop\yara-4.3.2-2150-win64>yara64 lab01.yar Chapter_1L
Lab01_01_exe Chapter_1L\Lab01-01.exe
Lab01_04_exe Chapter_1L\Lab01-04.exe
Lab01_01_dll Chapter_1L\Lab01-01.dll
Lab01_02_exe Chapter_1L\Lab01-02.exe
Lab01_03_exe Chapter_1L\Lab01-03.exe
```

1.5.2 使用自己编写的规则对自己电脑的 C 盘进行 Yara 引擎的扫描，记录扫描所用时间。

编写 C++ 代码，执行命令 `yara64 -r lab01.yar c: .`

```
#include <iostream>
#include <windows.h>
#include <string>
using namespace std;
```

```

string cmdPopen(const string& cmdLine) {
    char buffer[1024] = { '\0' };
    FILE* pf = NULL;
    pf = _popen(cmdLine.c_str(), "r");
    if (NULL == pf) {
        printf("Open pipe failed\n");
        return string("");
    }
    string ret;
    while (fgets(buffer, sizeof(buffer), pf)) {
        ret += buffer;
    }
    _pclose(pf);
    return ret;
}

int main() {
    // 设置工作目录
    wstring workingDir = L"C:\\Users\\XHR\\Desktop\\yara-4.3.2-2150-win64";
    if (!SetCurrentDirectory(workingDir.c_str())) {
        cout << "Failed to set the working directory" << endl;
        return 1;
    }

    long long start, end, freq;
    string cmdLine = "yara64 -r lab01.yar c:"; // 执行的指令

    QueryPerformanceFrequency((LARGE_INTEGER*)&freq);
    QueryPerformanceCounter((LARGE_INTEGER*)&start);
    string res = cmdPopen(cmdLine);
    QueryPerformanceCounter((LARGE_INTEGER*)&end);
    cout << "扫描到的文件: " << endl;
    cout << res; // 输出 cmd 指令的返回值
    cout << "运行时间为 " << (end - start) / freq << "s" << endl;
    return 0;
}

```

```
}
```

扫描结果如图所示：

```
Microsoft Visual Studio 调试控制台
error scanning c:\Windows\System32\winevt\Logs\Microsoft-Windows-StorageSpaces-Driver%4Operational.evtx: could not open file
error scanning c:\Windows\System32\winevt\Logs\Microsoft-Windows-StorageSpaces-ManagementAgent%4WHC.evtx: could not open file
error scanning c:\Windows\WinSxS\amd64_microsoft-windows-n..n_service_datastore_31bf3856ad364e35_10.0.19041.3208_none_45ee38b1eb36a302\dinary.xsd: could not open file
error scanning c:\Windows\WinSxS\amd64_microsoft-windows-n..n_service_datastore_31bf3856ad364e35_10.0.19041.746_none_af27db7894cefc18\dinary.xsd: could not open file
error scanning c:\Windows\WinSxS\amd64_microsoft-windows-u..userpredictionmodel_31bf3856ad364e35_10.0.19041.1_none_42c9bed4b6bd2e16\SBModel.txt: could not open file
error scanning c:\Windows\WinSxS\amd64_microsoft-windows-u..userpredictionmodel_31bf3856ad364e35_10.0.19041.1_none_42c9bed4b6bd2e16\SBModel.json: could not open file
扫描到的文件:
Lab01_01_exe c:\Users\XHR\Desktop\yara-4.3.2-2150-win64\Chapter_1L\Lab01-01.exe
Lab01_03_exe c:\Users\XHR\Desktop\yara-4.3.2-2150-win64\Chapter_1L\Lab01-03.exe
Lab01_02_exe c:\Users\XHR\Desktop\yara-4.3.2-2150-win64\Chapter_1L\Lab01-02.exe
Lab01_04_exe c:\Users\XHR\Desktop\yara-4.3.2-2150-win64\Chapter_1L\Lab01-04.exe
Lab01_04_exe c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_12L\Lab12-04.exe
Lab01_03_exe c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_18L\Lab18-02.exe
Lab01_01_exe c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_1L\Lab01-01.exe
Lab01_01_dll c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_1L\Lab01-01.dll
Lab01_02_exe c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_1L\Lab01-02.exe
Lab01_03_exe c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_1L\Lab01-03.exe
Lab01_04_exe c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_1L\Lab01-04.exe
Lab01_01_dll c:\Users\XHR\Desktop\请识别该病毒样本\BinaryCollection\Chapter_7L\Lab07-03.dll
Lab01_01_dll c:\Users\XHR\Desktop\yara-4.3.2-2150-win64\Chapter_1L\Lab01-01.dll
运行时间为 1433s
C:\Users\XHR\source\repos\ConsoleApplication1\x64\Debug\ConsoleApplication1.exe (进程 3200)已退出, 代码为 0。
按任意键关闭此窗口。 . . .
```

可以发现，扫描过程中遇到了一些没有权限的文件。整体扫描过程用时 1433s。成功扫描到了存放在桌面的病毒样本文件夹中的相关病毒文件。

1.5.3 讨论哪些 Yara 条件执行效率高，哪些 Yara 条件执行效率低，如何改进那些执行效率低的 Yara 条件？

执行效率高的一些条件：

简单字符串模式匹配通常执行速度较快。

使用字节字符串（hex strings）而不是文本字符串（strings）可以提高匹配速度。这对于二进制数据的匹配特别有效。

在条件中使用 `at` 操作符来限定字符串匹配的位置，以减少搜索范围，提高效率。

`filesize` 条件用于匹配文件的大小。这个条件非常高效，因为只需读取文件的大小信息，而不需要对文件的内容进行扫描。

`entrypoint` 条件用于匹配文件的入口点。这也是一个高效的条件，因为它只涉及到文件的元数据。

`of` 操作符用于部分字符串匹配，指定了字符串的偏移位置。它类似于 `at` 操作符，可以提高匹配效率。

执行效率较低的一些条件：

Yara 支持修饰符，如 `nocase`（不区分大小写）和 `fullword`（全字匹配），可以提高匹配的准确性和速度。

复杂的正则表达式可能会导致性能问题，特别是对于大型样本。尽量避免在条件中过多复

杂的正则表达式。

长字符串匹配会消耗更多的时间，尤其是在样本中的位置多次出现的情况下。

@ 操作符用于获取匹配字符串的偏移位置，但它可能会导致性能开销，特别是当多个条件都使用 @ 操作符时，会导致引擎多次计算偏移位置。

in 操作符用于指定字符串匹配的位置范围，这需要引擎在指定范围内搜索匹配。这可能会导致较低的执行效率，特别是当范围较大时。

文件包含条件用于在文件中查找另一个文件，这通常需要较多的时间，因为它涉及到文件的读取和比对。

要改进执行效率较低的 Yara 条件，可以考虑以下方法：

通过逆向分析目标文件，确定关键字串的出现位置或范围，然后在 Yara 条件中使用适当的偏移位置和长度来缩小匹配范围。这可以减少无用的搜索。

如果知道目标文件的入口点或其他关键信息，可以在 Yara 规则中使用 entrypoint 条件来精确匹配入口点，从而提高匹配的准确性和效率。

尽量减少使用 @ 操作符，特别是在大文件上。通过预先确定关键字串的位置，可以避免在每个条件中使用 @ 来获取偏移位置。

如果知道目标文件的类型，可以使用文件头部或其他特征来进行文件类型识别，然后只对特定类型的文件应用相应的 Yara 规则。这可以减少对不相关文件的匹配尝试。

如果目标文件遵循特定的结构或格式，可以根据文件结构来构建 Yara 条件，以减少不必要的搜索和提高匹配效率。

四、实验结论及心得体会

本次实验利用静态分析基础技术对所给恶意代码文件进行多步骤分析，掌握和熟练了静态分析基本方法，尤其是学会了如何根据导入表、字符串等信息推断恶意代码的功能，熟练了 PEid、IDA Pro 等基本工具的使用。

本次实验还练习了 Yara 规则的编写，探索了如何编写出更加高效的 Yara 规则。本次实验编写的 Yara 规则还比较初级，大多是字符串的简单匹配，未涉及到更复杂的特征匹配和结构，而且是针对单个样本的匹配。今后可以练习一些更复杂病毒样本或是多个同类恶意代码样本的 Yara 规则编写。