**恶意代码分析与防治技术实验报告**

**Lab6**

1. **实验环境**

Windows7，VMWARE，Windows11

1. **实验工具**

STRINGS, IDAPro,PEVIEW，EXEInfo，YARA

1. **实验内容**

Lab 6-1

1. 由 main 函数调用的唯一子过程中发现的主要代码结构是什么？

主要代码结构是 if 语句。

分析过程：

先进行基础静态分析。使用 PEView 打开 Lab06-01.exe，查看导入表，可以看到导入了

WININET.dll 文件，该库提供了通过网络使用 HTTP 的一组简单的 API,而导入的其中的

InternetGetConnectedState 函数可以返回本地系统的网络连接状态，推测可能有联网行为。

用 strings 工具查看该程序，看到一些和网络连接状况有关的可疑字符串：

分别在联网与没联网的情况下，用命令行运行该可执行文件，进行基础的动态分析，发现该

程序可以判断网络连接状况，并打印相应的语句（与前面查看到的字符串相匹配）。

下面用 IDA Pro 打开该程序，从 main 函数（.text:00401040）开始高级静态分析。

Main 函数调用的唯一子过程在 0x401000 处，双击进入，转到图形模式，如下：

该函数会在函数 InternetGetConnectedState 调用后，使用 cmp 指令比较函数返回结果与 0，

根据比较结果，运用 jz 指令控制执行流，出现两条不同的路径。当网络成功连接时，

InternetGetConnectedState 函数返回 1，ZF 位被清除，jz 指令会进入错误分支（红色）；否

则函数返回 0，进入正确分支（绿色），这是典型的 if 语句的代码结构。

2. 位于 0x40105F 的子过程是什么？

位于 0x40105F 的子过程是 printf。

分析过程：

尝试使用如下 IDA Python 指令获取该地址出子过程的名称，并未得到有效信息。

通过 sub\_401000 处的子过程会在 push 字符串后调用这个函数，推测这是个 printf 函数。

跳转到 0x40105F 处分析代码，可以看到如下信息：

其中\_\_stbuf和\_\_ftbuf都与C库中的printf函数相关，且FILE <0, 0, 0, 2, 1, 0, 0, 0>对应Windows

中的文件描述符，更加肯定了这是 printf 函数。

3. 这个程序的目的是什么？

综上分析，该程序帮助恶意代码在执行一些联网功能之前检查本地系统的网络连接状态。

如果找到可用连接，就打印“Success：Internet Connect”；否则，就打印“Error 1.1：No

Internet”。

Lab 6-2

1. Main 函数调用的第一个子过程执行了什么操作？

该子过程与 Lab6-1 中的 if 语句一样，可以检查本地系统的网络连接状态并打印相应字符串。

分析过程：

首先进行基础静态分析。使用 PEView 打开 Lab06-02.exe，查看导入表，可以看到导入了

WININET.dll 文件以及其中的五个函数。WININET.dll 库和 InternetGetConnectedState 函数的功

能在 Lab6-1 中已经介绍过。其他函数的具体功能如下：

InternetOpenUrlA 使用一个完整的 FTP 或者 HTTP 的 URL 来打开一个句柄。

InternetCloseHandle 用于关闭这些已打开文件的句柄。

InternetReadFile 用于从 InternetOpenUrlA 打开的句柄中读取数据。

InternetOpenA 用于初始化对 WinINet 库的使用，并设置用于 HTTP 通信的 User-Agent 字段。

不难发现，这些函数都与网络有关。

用 strings 工具查看该程序，发现如下可疑字符串：

前两条与 Lab6-1 相同，接下来的三条出错信息 2.3、2.2、2.1 告诉我么该程序可能会打开一

个网页，并解析一条指令。此外，还有一些域名字符串。

接下来，进行基础的动态分析。由于 WinINet 库通常都使用 HTTP，且我们在字符串中也发

现了一个 URL，于是我们使用 Netcat（监听 80 端口）和 ApateDNS 配置出一个虚拟网络，运

行该程序，捕捉到以下网络行为：

该程序对 www.practicalmalwareanalysis.com 有一条 DNS 请求，并且在监听 80 端口时发现，

该程序会使用一个 URL 请求网页。

下面用 IDA Pro 打开该程序，从 main 函数（.text:00401130）开始高级静态分析。

Main 函数调用的第一个子过程是 sub\_401000，双击进入，发现与 Lab6-1 中位于 0x401000

处的方法相同。

2. 位于 0x40117F 的子过程是什么？

位于 0x40117F 的子过程是 printf。

分析过程：

再次尝试用如下 Ida Python 语句获得子过程的名称，同样未得到有效信息。

于是跳转到 0x40117F 处查看代码进行分析，发现与 Lab6-1 中推测过的函数相同，是 printf

函数，于是给它重命名。

再次使用 IDA Python，重命名成功。

3. 被 main 函数调用的第二个子过程做了什么？

被 main 函数调用地第二个子过程位于 0x401040 处，它用于打开网页

http://www.practicalmalwareanalysis.com/cc.htm、读取其中的数据，并从页面开始处解析

注释。

分析过程：

Main 函数调用的第二个子进程是 sub\_401040，双击进入：

该函数首先调用了 InternetOpenA，用于初始化对 WinINet 库的使用，查看参数，可以发现

它将 User-Agent 字段设置为”Internet Explorer 7.5/pma”，与动态分析时在 80 端口监听到的结

果一致。

接着，该函数调用了 InternetOpenUrlA，用于打开一个句柄，通过查看参数，发现这里要打

开的是静态网页 www.practicalmalwareanalysis.com。

接下来，该函数将 InternetOpenUrlA 的返回结果与 0 进行比较，如果等于 0，该函数会打印

出错信息，调用 InternetCloseHandle 函数关闭打开的句柄并返回。否则，该函数会将

InternetOpenUrlA 的返回结果作为参数传给 InternetReadFile，并调用这个函数，再结合它的

其它参数，我们可以知道该函数用于从打开的 HTML 网页中读取内容，并将数据保存在 Buffer

中。

调用结束后，该函数将 InternetReadFile 的返回结果与 0 进行比较，如果等于 0，该函数会打

印出错信息，调用 InternetCloseHandle 函数关闭打开的句柄并返回。否则，将 Buffer 中的数

据逐一地与一个字符进行比较，若不相等，则立即打印出错信息。关闭句柄并返回，否则，

完成比较，函数返回。

通过查询 ASCII 码对照表，我们可以发现这些比较的字符合并起来是<!--，即 HTML 中注释的

开始部分，于是这四条 cmp 指令地作用就是检查被读取的网页是否是由一条注释开头的。

在指令 movsx edx, [ebp+var\_20F]中，由于 IDA pro 没有识别出 BUffer 这个局部变量的大小是

512 字节，因此显示为一个局部变量 var\_20F，实质上相等于一个 Buffer+1。后面的局部变量

类似，于是我们需要填充函数栈，显示一个 512 字节的数组，使得 Buffer 在整个函数中被正

确标记。调整结果如下：

4. 在这个子过程中使用了什么类型的代码结构？

结合上述分析和图形模式的结果，该过程主要使用了多个 if 结构来实现功能。调用 WinINet

的库函数，将返回数据填充到字符数组中，然后每次一个字节地对这个数组进行比较，以

解析一个 HTML 注释。

5. 在这个程序中有任何基于网络的指示吗？

根据前面的分析，我们已经知道了该程序主要有两个网络行为：

第一个是 InternetExplorer7.5/pma 作为 http 的 user-agent；

第二个是下载了 http://www.practicalmalwareanalysis.com/cc.htm 这个网页。

6. 这个恶意代码的目的是什么？

首先，该程序会检查本地系统的网络连接状态，如果不存在可用的 Internet 连接就终止运行。

否则，程序会使用独特的用户代理下载一个网页，该网页包含一段由<!--开始的 HTML 注释，

程序解析其后的那个字符并输出到屏幕，输出格式是 Success: Parsed command is X，其中 X

就是从该 HTML 注释解析出来的字符，解析成功，程序会休眠一分钟，然后终止运行。

分析过程：

当 sub\_401040 函数返回一个非 0 值时，会调用 printf 函数打印"Success: Parsed command

is %c\n"，其中%c 是格式字符串，是从 HTML 中解析出来的字符。

再调用 sleep 函数（0EA60h 表示 60000ms=60s）休眠一分钟。

Lab 6-3

1. 比较在 main 函数与 6-2 的 main 的函数调用。从 main 中调用的新的函数是什么？

调用的新函数是 sub\_401130。

分析过程：

我们仍从基础的静态分析开始。

发现导入表中，除了上述与网络行为相关的函数和库以外，还新增了一些操作注册表、文件

和目录表的函数，推测恶意代码可能会将自身或其他程序加入注册表中，以实现开机自启动。

相较之前，多了如下字符串：

前两条错误信息说明程序可能会修改注册表，而后面则是一些注册表位置、目录和文件名。

Software\Microsoft\Windows\CurrentVersion\Run 常用于恶意代码自启动。

下面用 IDA Pro 打开该程序，从 main 函数（.text:00401210）开始高级静态分析。

可以看到，main 函数中共调用了五个函数。

其中 sub\_401000（检查网络连接）和 sub\_401040（下载网页并解析 HTML 注释）、sub\_401271

（printf）和 sleep 函数都相同，新函数是 sub\_401130。

2. 这个新的函数使用的参数是什么？

该函数的两个参数从右往左依次是程序名和从 HTML 注释中解析出的指令字符。

分析过程：

可以看到，调用这个函数之前，eax 和 ecx 被压入了栈，分别存的是 argv 和 var\_8。其中，

argv 是标准 main 函数参数的 argv[0]，程序名本身；

var\_8 用 AL 设置，eax 是函数 sub\_401040 的返回值，AL 包含在 eax 中，因此 var\_8 包含的

是从 HTML 注释中解析出的指令字符。

3. 这个函数包含的主要代码结构是什么？

一个 switch 语句和一个跳转表。

分析过程：

查看该函数的代码，可以看到，IDA Pro 已经将其识别成有 5 个 cases 的 switch 语句，下面

开始分析具体结构。

其中，arg\_0 是由 IDA Pro 自动生成的标签，表示最后被压栈的参数（也就是第一个参数），

即从网络上获取并解析得到的指令字符，将其赋给 var\_8，再被加载到 ECX 中，下一条指令

sub ecx, 61h 从 ECX 中减去 0x61（ASCII a）。如果 arg\_0 等于 a，执行指令后 ECX 为 0。然后，

再将 ECX 与 4 进行比较，判断指令字符 arg\_0 是否为 a、b、c、d 或 e，其他则跳转离开，否

则，将通过指令 jmp ds:off\_4011F2[edx\*4] ;由指令字符 arg\_0 作为索引，跳转到相应的地址。

该 switch 语句在图形视图中如下：

4. 这个函数能够做什么？

该函数可以创建一个文件夹、复制一个文件、删除一个文件、设置一个注册表项的值、休

眠 100 秒或打印出错信息。

分析过程：

根据跳转表，我们可以分析 switch 语句中的每一个选项：

a 选项：调用 CreateDirectoryA 函数，判断文件路径是否存在"C:\\Temp"，不存在则创建它；

b 选项：调用 CopyFileA 函数，将源文件（当前程序名：Lab06-03.exe）复制到目的文件

（C:\\Temp\\cc.exe）中。

c 选项：调用 DeleteFileA 函数，当存在文件 C:\\Temp\\cc.exe 时，删除它。

d 选项：在注册表中将 Software\Microsoft\Windows\CurrentVersion\Run\Malware 值设置为

C:\Temp\cc.exe，实现跟随系统启动实现自启动，以持久化运行。

e 选项：将 Software\Microsoft\Windows\CurrentVersion\Run\Malware 值设置为 C:\Temp\cc. exe，实现跟随系统启动实现自启动。

休眠 100 秒；

Default 选项：打印“Error 3.2: Not a valid command provided”。

综上，可以得到该函数的功能。

5. 在这个恶意代码中有什么本地特征吗？

注册表键：Software\Microsoft\Windows\CurrentVersion\Run\Malware

文件路径：C:\Temp\cc.exe

6. 这个恶意代码的目的是什么？

首先，该程序会检查本地系统的网络连接状态，如果不存在可用的 Internet 连接就终止运行。

否则，程序会使用独特的用户代理下载一个网页，该网页包含一段由<!--开始的 HTML 注释，

该注释的第一个字符用于 switch 语句，决定该程序在本地系统中的下一步行为，包括创建

一个文件夹、复制一个文件、删除一个文件、设置一个注册表项的值、休眠 100 秒或打印

出错信息。

Lab 6-4

1. 在实验 6-3 和 6-4 的 main 函数中的调用之间的区别是什么？

sub\_401000 用于检查网络连接，sub\_401040 用于解析 HTML，sub\_401150 是 switch 语句，

sub\_4012B5 是 printf 函数。

分析过程：

我们仍从基础的静态分析开始。

相较 Lab6-3，发现导入表中并没有增加什么 Windows API 函数

相较之前，多了如下字符串：

进行基础动态分析，在 80 端口监听到一下信息，User-Agent 变成了 Internet Explorer

7.50/pma0。

下面用 IDA Pro 打开该程序，从 main 函数（.text:00401230）开始高级静态分析。

可以发现，与 Lab6-3 的 main 函数相比，虽然调用的五个函数都是相同的，但其中的结构有

所不同。

其中，sub\_401000 用于检查网络连接，sub\_401040 用于解析 HTML，sub\_401150 是 switch

语句，sub\_4012B5 是 printf 函数。

2. 什么新的代码结构已经被添加到 main 中？

For 循环。

分析过程：

切换到图形模式，可以看到一条向上指的箭头，很显然是循环语句。

下面详细分析该循环语句的代码：

其中，局部变量 var\_C 用于循环计数，在初始化时被赋值为 0，每执行到 loc\_401251 处就加

1，每次到了 0x004012AD 就返回到 loc\_401251 处递增，因此这是一个循环结构。

当 var\_C 大于或者等于 5A0h 时，循环停止。

在 0x00401267 处调用函数 sub\_401040 之前，var\_C 作为参数被压入栈。

且循环到 0x004012AD 处之前会休眠一分钟，最后将计数器加 1，因此持续 1440 分钟，24

小时。

3. 这个实验的解析 HTML 的函数和前面实验中的那些有什么区别？

函数 sub\_401040 使用一个参数，并且使用格式化字符串来调用 sprintf 函数，用传入的参数

创建用于 HTTP 通信的 User-Agent 字段。

分析过程：

刚刚看到，在调用函数 sub\_401040 之前，var\_C 作为参数被压入栈，而在前面的实验中，

sub\_401040 并没有参数，于是进一步查看代码。

其中，arg\_0 是唯一的参数，且只有 main 函数中调用了该函数，因此可以肯定该参数就是

var\_C。arg\_0被赋给eax，与一个格式化字符串以及一个目标地址一起被压入栈，调用了sprintf

函数，后者创建一个字符串，并将其存储在目的缓冲区，即 szAgent。

在 0x00401066 处，szAgent 作为参数被传给 InternetOpenA 函数，每次计数器递增生成的

User-Agent 会随之改变。这个机制可以被管理和监控 web 服务器的攻击者用于跟踪恶意代码

运行时间。

4. 这个程序会运行多久？（假设它已经连接到互联网。）

根据上面的分析，该程序共运行 1440 分钟，即 24 小时。

5. 在这个恶意代码中有什么新的基于网络的迹象吗？

该恶意代码使用了一个新的 User-Agent。其形式为 Internet Explorer 7.50/pma%d，其中，%d

是程序已经运行的分钟数。

6. 这个恶意代码的目的是什么？

在 Lab6-3 的基础上，该恶意代码会在运行 24 小时后终止。

Yara 规则的编写

1. Yara 规则

根据上述实验，总结 Lab6 中恶意代码的特征，写出如下 yara 规则：

2. 扫描

使用如下命令用 Lab6\_rule 扫描 C 盘文件，并查看运行前后的系统时间，约花费 3′20″

1. **实验心得**

**本次实验，我熟悉了对恶意代码分析工具有了更深入的理解。同时得到了充分的锻炼，并开始对本门课程的实验开始得心应手，做实验的速度越来越快了。**