Understanding Malware

2015/08/13 Security Camp 2015 10-D
JPCERT/CC Analysis Center
You NAKATSURU
Notice

These training materials are used for "Security Camp 2015" in Japan
- Security training program for students to discover & nurture young talent
- https://www.ipa.go.jp/jinzai/camp/ (Japanese only)

The training course consists of the following 2 parts
- Malware, Malware analysis basics, Static analysis basics
  - Learning basic knowledge for malware analysis
  - Malware analysis
    - Understanding details of malware samples using static analysis method

The training mainly focuses on 32bit Windows malware
- Some slides have display problems due to animation
- Any questions and comments are welcome
  - Please contact us at aa-info@jpcert.or.jp
Agenda

- Malware Basics
- Malware Analysis Basics
- Static Analysis Basics
Objectives of This Session

Understanding malware

• What malware is
• What malware does
• Malware trends
• Typical prevention/response methods

Understanding malware analysis

• What malware analysis is
• Malware analysis methods
• Static analysis techniques
Malware Basics
Definition

Malicious Software

Broader in concept than a computer virus

- Virus, Worm, Trojan Horse, Rootkit, Bot, DoS Tool, Exploit kit, Spyware
Malware Purpose

Mischief
- Crashing a system
- DoS

For Profit
- Havoc via DDoS
- Sending Spam
- Visiting affiliate sites

Others
- Stalking
- Self-assertion
For Profit

Selling
- Sensitive information
- Malware, malware builder

Sending spam emails
- Rental business

DDoS
- Blackmail

Affiliate
- Let user access to the site using malware
Growth of Malware

- **Virus**
  - Mischief, Concept code
- **Worm, Trojan, Backdoor**
- **Bot**
  - For Profit (underground)
Infection Method

Software

• Attack software vulnerabilities
  • OS, Office, Browser
• Make machines to execute malware

Human

• Trick users to execute malware
  • provide a line about software contents
  • camouflage an "icon"
Exploiting Software Vulnerability

- Buffer overflow, etc.
- Take control and execute arbitrary code

- Shellcode for malware execution
- Malware
Malware Attack Vector

- Vulnerability, Social engineering

USB, Network, Mail, Download, Browser, P2P
Malware Behavior

- Can do anything on the infected machine
  - Within the limits of infected user's privilege
Malware Behavior: Installation

Create main module
- download, creation

Copy / Delete itself
- copy to the system folder

Run after reboot
- registry entry related to Autorun
- Startup folder
- register as a service
Malware Behavior: Modifying System

Disable security features

- Windows Firewall
- Windows Update

Avoid security programs

- Anti-Virus software
- Analysis tool

Hide itself

- modifying other processes
Malware Behavior: Main Behavior

**Steal information**
- read registry entries / config files
- key logging, packet capture

**Bot**
- connect to C&C servers
- execute commands

**Spread**
- attack other machines
Important Points

Network activity is important

• Attackers need to take out information for profit
• Able to recognize damage by analyzing packets

Do not trust infected machines

• Possibility of data falsification
  • Such as anti-virus software results on infected machines
    • Recommended to re-install Windows
• Preventing malware infection is the most important
MALWARE EXAMPLE
Targeted Attack

Looks like normal Excel file

Exploits vulnerability in Excel

From: © Subject: Personnel reshuffle
There was a personnel reshuffle. See the attachment file.
RAT

Remote Administration Tool/Trojan—Often used for targeted attack
Banking Trojan

- Attempts to steal users credential of the Internet banking
  - Inject additional input form on the web page
Web-based Attack

Attacking web browsers or add-ons

1. Inject malicious JavaScript into web contents
2. Redirects to an attack site in background
3. Attacks vulnerabilities
4. Information theft
"All your important files are encrypted!"

Intimidation message

Encryption key, URL to lead (Tor)

Delete after encryption
Android Malware

- Re-package popular legitimate apps
  - Distributed through
    - 3rd party market
    - Android Market

Steal contact information in background
MALWARE TRENDS IN JAPAN
Banking Trojan

- ZeuS, Citadel, Gameover were over

- Vawtrak
- Dyre
- Tsukuba
- Tinba
- Dridex
- Chthonic
- KINS
Ransomware

Spread through Drive-by-Download attack

Targeted Attack: Watering Hole Attack

A case of compromised site for a media player update

IP address check

Update Server

Player

Another site

Automatically install malware
CloudyOmega/Blue Termite

Attack infrastructure (affected sites, affected VPS servers)

Japan

Outside of Japan

Victim (government related organizations, private companies)

Targeted email

Mass email
ANTI-MALWARE
# Typical Malware Prevention

## Fix vulnerabilities
- Update OS & software
- Configure security options for OS & software

## Use anti-virus software
- Possibly false results

## Do not open a file without confirming
- Beware of social engineering
Typical Response

Disconnect network connection first

• To stop information leakage & attack to outside

Re-install OS

• Basically malware can do anything on infected machines

Recurrence prevention

• Identify & fix up the cause of infection
Worldwide Activity

Botnet takedown
- Microsoft, FBI, anti-virus vendors, etc.
- Major activities
  - Rustock takedown
  - ZeroAccess takedown
  - Citadel takedown

Convention on Cybercrime
- Drawn up by the Council of Europe
- Convention for co-investigation of cyber criminals
Malware Analysis Basics
Who Analyzes Malware?

- CSIRTs
- Security product developers
- Security service providers
- Anti-malware researchers
- Software developers
- Law enforcement
Why Analyze Malware?

- Incident response
- Product development/improvement
- Signature creation
- Cutting-edge countermeasure
- Vulnerability analysis
- Criminal arrest
IMPORTANT POINT
Security is a Key for Success

Analyze malware with great care
- If you make a mistake, it may bring serious consequences

Develop environment with great care
- Pay great attention to environment for malware download and analysis

Publish results with great care
- Take great care in publishing details of malware
  - e.g. 0-day vulnerability
Sample Analysis Environment

- Download Environment
  - Change file extension
  - Linux
  - Real time filtering
  - Dedicated network
    - Be able to restore to a clean system

- Write-protected

Copyright © 2015 JPCERT/CC All rights reserved.
SURFACE ANALYSIS
Surface Analysis

- Retrieve surface information from targets without execution

- Target
  - Hash value
    - VirusTotal
  - File type
    - Analyst's blog
  - Strings
    - Tool information
  - Anti-virus software results
    - Vender's database
Runtime Analysis

- Execute malware and monitor its behavior
  - Difficult to reveal "all" of malware’s behavior

Manual Analysis
- Use monitoring tools on analysis environment
  - Sysinternals suite, etc.

Automation
- Use sandbox system
  - Free / Commercial products
  - Can reduce analysis time
Runtime Analysis Environment

- Dedicated and isolated network/system
  - Native or Virtual (VMware, etc.)
What Static Analysis is

- Reading code in binary file and understanding its functionality
  - Takes a long time
  - Requires deep and broad knowledge
ANALYSIS PROCESS
## Analysis Process Comparison

<table>
<thead>
<tr>
<th>Overview</th>
<th>Surface analysis</th>
<th>Runtime analysis</th>
<th>Static analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Retrieve surface information from targets without execution</strong></td>
<td><strong>Execute samples</strong> and monitor its behavior</td>
<td><strong>Read codes</strong> in binary files and understand its functionality</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Output</th>
<th>- Hash values</th>
<th>- Strings</th>
<th>- File attributes</th>
<th>- Packer info</th>
<th>- Anti-virus detection info</th>
<th>Activity of - File system</th>
<th>- Registry</th>
<th>- Process</th>
<th>- Network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Security risk</td>
<td>Low</td>
<td>High</td>
<td>Moderate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis coverage</td>
<td>Low</td>
<td>Moderate</td>
<td>High</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Malware Analysis Flow

1. Start
   - Surface analysis
     - enough?
       - Yes: Summarize Result, End
       - No: Runtime analysis
2. Runtime analysis
   - enough?
     - Yes: Summarize Result, End
     - No: Static analysis
3. Static analysis
   - Summarize Result, End
Static Analysis Basics
Important Points

No need to know all of malware

• You need much time for static analysis

Need much knowledge/experiences

• Need
  • OS knowledge
  • Assembly basics
  • Efficient reading techniques
  • Anti-analysis techniques
• We have to continue studying
Disassemble & Decompile

Binary -> source code

Disassemble

```c
BOOL __stdcall D1Main(HINSTANCE hinstDLL, LONG lngCmd, HINSTANCE hPrevInstance, LPVOID lpvReserved)

var_100 = dword ptr -10h
Filename = dword ptr 0Ch
hinstDLL = dword ptr 8
fReason = dword ptr 0Ch
InvResed = dword ptr 18h

push ebf
push eax
mov eax, [ebp+0Ch]
push 0
call d SmashModuFileNamesA
lea eax, [esp-1Ch]
call d GetKeyboardLayout

if ( fReason == 1 )
{
    v5 = _mbstowcs(v4);
    if ( _mbstowcs(v5, (const unsigned __int8 *)
    {
        v6 = (unsigned int)GetKeyboardLayout;
        if ( v6 == 1033 ) [sub_1000126C();]
    }
```

Decompile
Read assembly code while executing step by step
Static Analysis Flow

1. Manual Unpacking

2. Decompile

3. Disassemble, Debugging

Packed?

Can decompile?

Analyze unpacked target

Start

Result

End
## Static Analysis Tools

<table>
<thead>
<tr>
<th>Category</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Disassembler</td>
<td>IDA</td>
<td>Disassembles more than 50 architectures</td>
</tr>
<tr>
<td>Decompiler</td>
<td>Hex-rays</td>
<td>x86/ARM binary to C source code</td>
</tr>
<tr>
<td></td>
<td>Decompiler</td>
<td></td>
</tr>
<tr>
<td></td>
<td>VB</td>
<td>Visual Basic binary to Visual Basic source code</td>
</tr>
<tr>
<td></td>
<td>.NET Reflector</td>
<td>.NET binary to .NET source code</td>
</tr>
<tr>
<td>Debugger</td>
<td>OllyDbg</td>
<td>World famous x86 debugger</td>
</tr>
<tr>
<td></td>
<td>Immunity Debugger</td>
<td>Python familiar x86 debugger</td>
</tr>
</tbody>
</table>
Static Analysis Basics

1. Manual unpacking
2. Decompile
3. Disassemble, Debugging

Focused on "Disassemble" only

Start

Packed?

Can decompile?

Analyze unpacked target

Result

End

With kind permission from JPCERT/CC

Copyright © 2015 JPCERT/CC All rights reserved.
## Interactive DisAssembler

- [http://www.hex-rays.com/idapro/](http://www.hex-rays.com/idapro/)
  - 3 versions are available

<table>
<thead>
<tr>
<th>Version</th>
<th>Free</th>
<th>Starter</th>
<th>Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Ver. 5.0 Ver. 6.8 demo</td>
<td>Ver. 6.8</td>
<td>Ver. 6.8</td>
</tr>
<tr>
<td>Cost</td>
<td>Free</td>
<td>• 589USD/User • 879USD/Computer</td>
<td>• 1129USD/User • 1689USD/Computer</td>
</tr>
<tr>
<td>Features</td>
<td>• Old or limited</td>
<td>• Supports up to 20 processes</td>
<td>• Supports up to 50 processes • Can analyse files for 64 bit platforms</td>
</tr>
</tbody>
</table>
ASSEMBLY BASICS
Components of Computer System

- Control Unit
- Arithmetic Logic Unit
- Main Memory (DRAM...)
- Storage (HDD...)
- Input Device (Keyboard...)
- Output Device (Display...)
- CPU
- Memory Unit
- I/O
CPU Operation

Fetch & Decode
read CPU instruction from Memory

Execute
execute CPU instruction using Registers

next instruction address
int main(){
    int a, b, c;
    a = 1;
    b = 2;
    c = a + b;
    printf("Answer is \%d\n", c);
    return 0;
}
Disassemble

Machine code to assembly code (human readable)

```
55
8B EC
81 EC 04 01 00 00
83 7D 0C 01
74 06

33 C0
C9
C2 0C 00

68 04 01 00 00
8D 85 FC FF FF FF
50
6A 00
FF 15 2C 60 00 10
8D 85 FC FF FF FF
6A 5C
50
E8 83 01 00 00
```
Format of Assembly Code

OpCode

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>push</td>
<td></td>
</tr>
<tr>
<td>mov</td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td></td>
</tr>
<tr>
<td>mov</td>
<td></td>
</tr>
<tr>
<td>mov</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td></td>
</tr>
<tr>
<td>mov</td>
<td></td>
</tr>
<tr>
<td>push</td>
<td></td>
</tr>
<tr>
<td>push</td>
<td></td>
</tr>
<tr>
<td>call</td>
<td></td>
</tr>
<tr>
<td>add</td>
<td></td>
</tr>
<tr>
<td>xor</td>
<td></td>
</tr>
<tr>
<td>mov</td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td></td>
</tr>
<tr>
<td>ret</td>
<td></td>
</tr>
</tbody>
</table>

Operand (arguments)

<table>
<thead>
<tr>
<th>Operand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ebp</td>
<td></td>
</tr>
<tr>
<td>ebp, esp</td>
<td></td>
</tr>
<tr>
<td>esp, 0Ch</td>
<td></td>
</tr>
<tr>
<td>[ebp- 4], 1</td>
<td></td>
</tr>
<tr>
<td>[ebp- 8], 2</td>
<td></td>
</tr>
<tr>
<td>eax, [ebp- 4]</td>
<td></td>
</tr>
<tr>
<td>eax, [ebp- 8]</td>
<td></td>
</tr>
<tr>
<td>[ebp- 0Ch], eax</td>
<td></td>
</tr>
<tr>
<td>ecx, [ebp- 0Ch]</td>
<td></td>
</tr>
<tr>
<td>ecx</td>
<td></td>
</tr>
<tr>
<td>0040C000h</td>
<td></td>
</tr>
<tr>
<td>00401034h</td>
<td></td>
</tr>
<tr>
<td>esp, 8</td>
<td></td>
</tr>
<tr>
<td>eax, eax</td>
<td></td>
</tr>
<tr>
<td>esp, ebp</td>
<td></td>
</tr>
<tr>
<td>ebp</td>
<td></td>
</tr>
</tbody>
</table>
## Register

- Memory inside CPU
  - Can use them as variables for calculations
  - Address that indicates next instruction (Program Counter)
  - Pointers related stack

<table>
<thead>
<tr>
<th>Register name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAX, EBX, EDX</td>
<td>General purpose register</td>
</tr>
<tr>
<td>ECX</td>
<td>General purpose register especially used for counter</td>
</tr>
<tr>
<td>ESI, EDI</td>
<td>General purpose register especially used for &quot;source&quot; and &quot;destination&quot;</td>
</tr>
<tr>
<td>EIP (Instruction Pointer)</td>
<td>Address that indicates next instruction</td>
</tr>
<tr>
<td>ESP (Stack Pointer)</td>
<td>Current stack address</td>
</tr>
<tr>
<td>EBP (Base Pointer)</td>
<td>Bottom of stack for current function</td>
</tr>
</tbody>
</table>
Register Size

Several registers' names are changed according to the data size
— EAX, EBX, ECX...

EAX (32bit)

AX (lower 16bit)
AH (higher 8bit)
AL (lower 8bit)

“mov eax, 0” ≠ “mov ax, 0”
## Major Instructions

<table>
<thead>
<tr>
<th>Category</th>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assignment</td>
<td>mov</td>
<td>Copy value</td>
</tr>
<tr>
<td></td>
<td>lea</td>
<td>Load address</td>
</tr>
<tr>
<td>Calculation</td>
<td>add &amp; sub</td>
<td>+ / -</td>
</tr>
<tr>
<td></td>
<td>and &amp; or &amp; xor &amp; not</td>
<td>Logical operation</td>
</tr>
<tr>
<td></td>
<td>inc &amp; dec</td>
<td>++1 / --1</td>
</tr>
<tr>
<td>Jump</td>
<td>jmp</td>
<td>Jump to specified address</td>
</tr>
<tr>
<td></td>
<td>jz, jnz, ja, ...</td>
<td>For branch on condition</td>
</tr>
<tr>
<td></td>
<td>call</td>
<td>Call subroutine (function)</td>
</tr>
</tbody>
</table>
Stack

- Store temporary values to "stack" on memory
  - Due to the limits of registers

- Stack management
  - Use PUSH/POP
  - Stack related addresses are stored in EBP & ESP

```
PUSH 1
POP  EAX
POP  EBX
POP  ECX
POP  EDX
```
Function Call using Stack

- call = push + jmp
- retn = pop + jmp

```
main:
push 440100h ;arg1
push 440000h ;arg2
call foo
mov [ebp+result],eax
:
foo:
push ebp
mov ebp, esp
xor eax, eax
:
mov esp, ebp
pop ebp
retn 10h
```
Local Variables

- Local variables are allocated on stack
  - Normally referred using offset from ebp

```c
void func(){
    char Name[] = "abcdefg";
    int i;
    :
}
```
Branch on Conditions

Basic flow

1. Comparison operation
   - `cmp a b`
     - "sub a b" and discard result
   - `test a b`
     - "and a b" and discard result test

2. Jump on condition
   - `jz`
   - `jnb`
   - `ja`
   - `etc.`

<table>
<thead>
<tr>
<th>Char</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>not</td>
</tr>
<tr>
<td>z/e</td>
<td>zero/equal Previous result is 0 (Both values are same)</td>
</tr>
<tr>
<td>a</td>
<td>above Operand 1 is higher than operand 2</td>
</tr>
<tr>
<td>b</td>
<td>below Operand 1 is smaller than operand 2</td>
</tr>
</tbody>
</table>
Exercise 1. Static Analysis Basic

i. Analyze the following function and explain what it does

```assembly
sub_401000 proc near

; CODE XREF: _main+8↓p

var_8     = dword ptr -8
arg_0     = dword ptr  8
arg_4     = dword ptr 0Ch

push ebp
mov ebp, esp
sub esp, 8
mov eax, [ebp+arg_0]
add eax, [ebp+arg_4]
mov [ebp+var_8], eax
mov eax, [ebp+var_8]
mov esp, ebp
pop ebp
ret

sub_401000 endp
```
Exercise 1. Answer

i. Analyze the following function and explain what it does

```
sub_401000 proc near
    ; CODE XREF: _main+8↓p

var_8    = dword ptr -8
arg_0    = dword ptr 8
arg_4    = dword

push      
mov      ebp, esp
sub      esp, 8
mov      eax, [ebp+arg_0]
add      eax, [ebp+arg_4]
mov      [ebp+var_8], eax
mov      eax, [ebp+var_8]
mov      esp, ebp
pop      ebp
retn     

sub_401000 endp
```

```
arg_0 + arg_4
```
Exercise 1. Static Analysis Basic

ii. Find “branch on condition” and explain each condition and corresponding result

```c
_sub_401000 is "add_value" function
```
Exercise 1. Answer

ii. Find “branch on condition” and explain each condition and corresponding result

stored result of "add_value" func

result is 0 or not?
## MOV—Move

<table>
<thead>
<tr>
<th>Opcode</th>
<th>Instruction</th>
<th>Op/En</th>
<th>64-Bit Mode</th>
<th>Compat/Leg Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>88 /r</td>
<td>MOV r/m8,r8</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r8 to r/m8.</td>
</tr>
<tr>
<td>REX + 88 /r</td>
<td>MOV r/m8***,r8***</td>
<td>MR</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r8 to r/m8.</td>
</tr>
<tr>
<td>89 /r</td>
<td>MOV r/m16,r16</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r16 to r/m16.</td>
</tr>
<tr>
<td>89 /r</td>
<td>MOV r/m32,r32</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r32 to r/m32.</td>
</tr>
<tr>
<td>REX.W + 89 /r</td>
<td>MOV r/m64,r64</td>
<td>MR</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r64 to r/m64.</td>
</tr>
<tr>
<td>8A /r</td>
<td>MOV r8,r/m8</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m8 to r8.</td>
</tr>
<tr>
<td>REX + 8A /r</td>
<td>MOV r8***,r/m8***</td>
<td>RM</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r/m8 to r8.</td>
</tr>
<tr>
<td>8B /r</td>
<td>MOV r16,r/m16</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m16 to r16.</td>
</tr>
<tr>
<td>8B /r</td>
<td>MOV r32,r/m32</td>
<td>RM</td>
<td>Valid</td>
<td>Valid</td>
<td>Move r/m32 to r32.</td>
</tr>
<tr>
<td>REX.W + 8B /r</td>
<td>MOV r/m64,r64</td>
<td>RM</td>
<td>Valid</td>
<td>N.E.</td>
<td>Move r/m64 to r64.</td>
</tr>
<tr>
<td>8C /r</td>
<td>MOV r/m16,Sreg**</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move segment register to r/m16.</td>
</tr>
<tr>
<td>REX.W + 8C /r</td>
<td>MOV r/m64,Sreg**</td>
<td>MR</td>
<td>Valid</td>
<td>Valid</td>
<td>Move zero extended 16-bit segment register</td>
</tr>
</tbody>
</table>
EFFICIENT CODE ANALYSIS
In which order should be read the following source code?

```c
int send_data(){
    HANDLE hInternet, hConnect, hRequest;

    hInternet = InternetOpen(NULL, INTERNET_OPEN_TYPE_PRECONFIG, NULL, NULL, 0);
    if(hInternet == NULL)
        return 1;

    hConnect = InternetConnect(hInternet, SERVER_NAME, INTERNET_DEFAULT_HTTP_PORT, ...
    if(hConnect == NULL){
        InternetCloseHandle(hInternet);
        return 2;
    }

    hRequest = HttpOpenRequest(hConnect, __T("GET"), NULL, NULL, ...
Reading Steps

1. Check Windows API
2. Check arguments
3. Check brunch on condition

```assembly
lea    edx, [esp+4DUh+FileName]  ; IpBuffer
push   edx                      ; nBufferLength
push   104h
call   ds:GetTempPathW
test   eax, eax
jnz    short loc_4010B1
lea    eax, [esi+48h]
```
## Learning Windows API

**Use MSDN Library**

- [http://msdn.microsoft.com/library](http://msdn.microsoft.com/library)

<table>
<thead>
<tr>
<th></th>
<th><strong>Online</strong></th>
<th><strong>Offline</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cost</strong></td>
<td>Free</td>
<td>MSDN Subscription needed to download</td>
</tr>
</tbody>
</table>
| **Features** | • Always up to date  
• Need Internet connection  
• Can download necessary sections for offline use  
• Depends on connection speed | • Can be used in offline environment  
• Fast |
Reading Arguments

- **Assembly code**

```
push 0 ; bFailIfExists
mov eax, [ebp+C]
push eax ; lpNewFileName
mov ecx, [ebp+8]
push ecx ; lpExistingFileName
call ds: CopyFileA
```

- **C++ syntax from MSDN Library**

```cpp
BOOL CopyFile(
    LPCTSTR lpExistingFileName,
    LPCTSTR lpNewFileName,
    BOOL bFailIfExists
);```

- **Return value is stored in EAX**
Exercise 2. Efficient Code Analysis

Read the following function efficiently and guess what it does.

```
and [ebp+hKey], 0
lea eax, [ebp+hKey]
push eax ; phkResult
push 0F003Fh ; sam Desired
push 0 ; ulOptions
push offset SubKey ; "SOFTWARE\Microsoft\Windows\CurrentVe"
push 8000001h ; hKey
call ds:RegOpenKeyExA
test eax, eax
jz short loc_401382
xor eax, eax
jmp short locret_4013BB
```

```
loc_401382:
push 105h ; cbData
push offset Data ; lpData
push 1 ; dwType
push 0 ; Reserved
push offset ValueName ; "TRAINING"
push [ebp+hKey] ; hKey
call ds:RegSetValueExA
```
Exercise 2. Answer

Read the following function efficiently and guess what it does.

- Target entry is "Run" key
- Register itself to be executed automatically after rebooting
- Write value in registry entry
USING IDA
(recap) **Interactive DisAssembler**

- [http://www.hex-rays.com/idapro/](http://www.hex-rays.com/idapro/)
  - 3 versions are available

<table>
<thead>
<tr>
<th></th>
<th>Free</th>
<th>Starter</th>
<th>Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Version</strong></td>
<td>Ver. 5.0</td>
<td>Ver. 6.5</td>
<td>Ver. 6.5</td>
</tr>
<tr>
<td></td>
<td>Ver. 6.5 demo</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Cost</strong></td>
<td>Free</td>
<td>• 539USD/User</td>
<td>• 1059USD/User</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 819USD/Computer</td>
<td>• 1589USD/Computer</td>
</tr>
<tr>
<td><strong>Features</strong></td>
<td>• Old or limited</td>
<td>• Supports up to 20 processes</td>
<td>• Supports up to 50 processes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Can analyze files for 64 bit platforms</td>
<td></td>
</tr>
</tbody>
</table>
You Have to Talk with

Madame de Maintenon
(see Wikipedia)
Important Points for IDA

1. Make it right
   • **Instruction** or **data**?
   • **Malicious function** or **library function**?

2. Use as a high functionality notepad
   • Name analyzed function / variables
   • Write your comments
   • Put function type-declaration
   • Change display format for easier reading
   • Hex / binary / ASCII / offset
Main Windows (IDA view)

Graph view / Text view
Main Windows (IDA view)

Reading Code

```
00000000 55 MYUNK  db  55h ; Regular unknown name
00000001 02 MYDATA  db offset MYCODE+1 ; Regular data name
00000001 1D+ MYCODE:  ; CODE XREF: MyFunc::MyClass(int,ulong)
00000001 2B+      ; MyFunc::MyClass(int,ulong)+Cp
00000001 02+      ; DATA XREF: seg000:0000002C1o
00000001 00+      ; seg000:0000000010
00000007 53+      ; Regular code name
00000025 9A+      ; Macro name & string constant
0000002C 55 unk_0_28 db  5Eh ; Dummy unknown name
0000002D 01 HIDNAME db '1', '1', MYCODE
0000002E 49 off_1  db  0 ; Hidden name
00000030 63 d      ; Dummy data name and libfunc name
00000031 55 db  55h ; Character data constant
00000031 ; Below are an assembler directive and a segment name
00000031 asumse ds:seg000 ; Numeric data constant
00000031 SUBROUTINE ; MyFunc::MyClass(int,unsigned long)
00000031 ; A demangled name is above
00000031 MYFunc::MyClass::call near
00000031 arg_4 = dword ptr 8
00000033 80+        ; Character constant in instruction
00000033 8B+        ; A void operand
00000033 88h+       ; Call Procedure
00000033 <void>     ; Call Procedure
00000038 8B+        ; Stack var and alternative operand
00000038 call MYCODE
0000003D 8B+        ; Dummy code name
0000003D call near ptr MYCODE+1
00000042 C7+        ; mov [ebp+arg_4], ALTOP
00000042 mov  [ebp+arg_4], ALTOP
00000049 locret_2:  ; mov [ebp+arg_4], ALTOP
```

- **Name**: 
- **XREF**: 
- **Calling API**: 
- **Comments**: 

Copyright © 2015 JPCERT/CC All rights reserved.
Strings Window

Like BinText
Functions Window

A list of functions
Imports Windows

A list of APIs required by the target

### Table of Imports

<table>
<thead>
<tr>
<th>Address</th>
<th>Ordinal</th>
<th>Name</th>
<th>Library</th>
</tr>
</thead>
<tbody>
<tr>
<td>00406000</td>
<td></td>
<td>ExitProcess</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406004</td>
<td></td>
<td>GetEnvironmentVariableW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406008</td>
<td></td>
<td>FindResourceW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>0040600C</td>
<td></td>
<td>LoadResource</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406010</td>
<td></td>
<td>CreateProcessW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406014</td>
<td></td>
<td>HeapAlloc</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406018</td>
<td></td>
<td>HeapFree</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>0040601C</td>
<td></td>
<td>GetProcessHeap</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406020</td>
<td></td>
<td>WriteFile</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406024</td>
<td></td>
<td>LoadLibraryW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406028</td>
<td></td>
<td>SizeOfResource</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>0040602C</td>
<td></td>
<td>GetModuleFileNameW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406030</td>
<td></td>
<td>CreateFileW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406034</td>
<td></td>
<td>GetTempPathW</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>00406038</td>
<td></td>
<td>GetLastError</td>
<td>KERNEL32</td>
</tr>
<tr>
<td>0040603C</td>
<td></td>
<td>GetProcAddress</td>
<td>KERNEL32</td>
</tr>
</tbody>
</table>

Line 1 of 65
Hex View Window

- Imports (list of APIs required by the target)
## Recommended Configuration

- **IDA config files you can edit**
  - `C:\Program Files (x86)\IDA Free\cfg\`

- **You can create user settings file**
  - `idauser.cfg, idauserg.cfg`

<table>
<thead>
<tr>
<th>Config File</th>
<th>Name</th>
<th>Meaning</th>
<th>Recommended Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ida.cfg /</td>
<td>_OPCODE_BYTES</td>
<td>Display binary data</td>
<td>8</td>
</tr>
<tr>
<td>idauser.cfg</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>SHOW_SP</td>
<td>Display stack pointer</td>
<td>YES</td>
</tr>
<tr>
<td></td>
<td>SHOW_XREFS</td>
<td>Display cross references</td>
<td>8</td>
</tr>
<tr>
<td>idagui.cfg /</td>
<td>DISPLAY_PATCH_SUBMENU</td>
<td>Display patch submenu</td>
<td>YES</td>
</tr>
<tr>
<td>idauserg.cfg</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Basic Instruction

#### Move

<table>
<thead>
<tr>
<th>Key assign</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Jump to address</td>
</tr>
<tr>
<td>Esc</td>
<td>Back</td>
</tr>
</tbody>
</table>

#### Changing data type

<table>
<thead>
<tr>
<th>Key assign</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>U</td>
<td>Change selection to &quot;Unknown&quot;</td>
</tr>
<tr>
<td>C</td>
<td>Change selection to &quot;Code&quot;</td>
</tr>
<tr>
<td>D</td>
<td>Change selection to &quot;Data&quot;</td>
</tr>
<tr>
<td></td>
<td>• Byte, Word, Double Word</td>
</tr>
<tr>
<td>P</td>
<td>Change selection to &quot;Function&quot;</td>
</tr>
<tr>
<td>A</td>
<td>Change selection to &quot;ASCII string&quot;</td>
</tr>
<tr>
<td>*</td>
<td>Change selection to &quot;Array&quot;</td>
</tr>
</tbody>
</table>
# Basic Instruction

## Note

<table>
<thead>
<tr>
<th>Key assign</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Name function/variables/etc.</td>
</tr>
<tr>
<td>:</td>
<td>Insert comment</td>
</tr>
<tr>
<td>;</td>
<td>Insert repeatable comment</td>
</tr>
<tr>
<td>Y</td>
<td>Put type-declaration</td>
</tr>
<tr>
<td></td>
<td>• void __cdecl Func(int num1, int num2);</td>
</tr>
</tbody>
</table>

## Display format

| H          | Decimal <-> Hexadecimal |
| R          | ASCII <-> Hexadecimal |
| Right-click Symbolic constant | Symbolic constant <-> Hexadecimal |
|           | • ERROR_ALREADY_EXISTS |
|           | • ACCESS_ALL |
|           | • KEY_WRITE |
Basic Analysis Process in IDA

1. **Read instructions**
2. **Rename & Comment**
   - Variables
   - Functions
   - Data
   - etc.
3. **Understand meaning**
Example of Renaming

Exercise 1

```
sub_401000 proc near ; COD
var_8 = dword ptr -8
arg_0 = dword ptr 8
arg_4 = dword ptr 0Ch

push    ebp
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+arg_1] arg1
add     eax, [ebp+arg_2] arg2
mov     [ebp+var_8], eax
mov     eax, [ebp+var_8]
mov     esp, ebp
pop     ebp
retn

sub_401000 endp
```

```
decl add_func(int arg1, int arg2)
proc near ; COD

push    ebp
mov     ebp, esp
sub     esp, 8
mov     eax, [ebp+arg1]
add     eax, [ebp+arg2]
mov     [ebp+answer], eax
mov     eax, [ebp+answer]
mov     esp, ebp
pop     ebp
retn

add_func endp
```
Example of Analysis

### Analyzed IDB sample

```assembly
push offset rclsid ; rclsid
call ds:CoCreateInstance
mov esi, eax
cmp esi, edi
jle loc_10016896
mov eax, [ebp+ppv]
 mov ecx, [eax]
lea edx, [ebp+pProxy]
push edx
push eax, [ecx+0Ch]
call eax
mov esi, eax
cmp esi, edi
jle loc_10016896
mov ecx, [ebp+pProxy]
push edi ; dwCapabilities
push edi ; pAuthInfo
push 3 ; dwImpLevel
push 3 ; dwAuthnLevel
push edi ; pServerPrincName
push edi ; dwAuthzSvc
push 0Ah ; dwAuthzSvc
call ds:CoSetProxyBlanket
```
Exercise 3. Using IDA

i. Analyze the following functions in "static_sample3.idb" and rename functions/variables or insert your comments
   — sub_4012DD
   — sub_401303
   — sub_401357
Exercise 3. Answer

i. Analyze the following functions in "static_sample3.idb" and rename functions/variables or insert your comments
   — sub_4012DD
   — sub_401303
   — sub_401357
   — See static_sample3_ans.idb
FYI: IDC Scripting

If you want to change background color on "jmp" & "call" instructions

```
auto head, op;
head = NextHead(0x00000000, 0xFFFFFFFF);
while ( head != 0xFFFFFFFF ){
    op = GetMnem(head);
    if (op == "jmp" || op == "call")
        SetColor(head, CIC_ITEM, 0xFFFFCF8);
    head = NextHead(head, 0xFFFFFFFF);
}
```
Questions?