Forensics II

Static and dynamic analysis [repetition]

IDA Pro and OllyDbg

Obfuscated code analysis

De-obfuscation of binaries

Forensic Analysis of unknown files

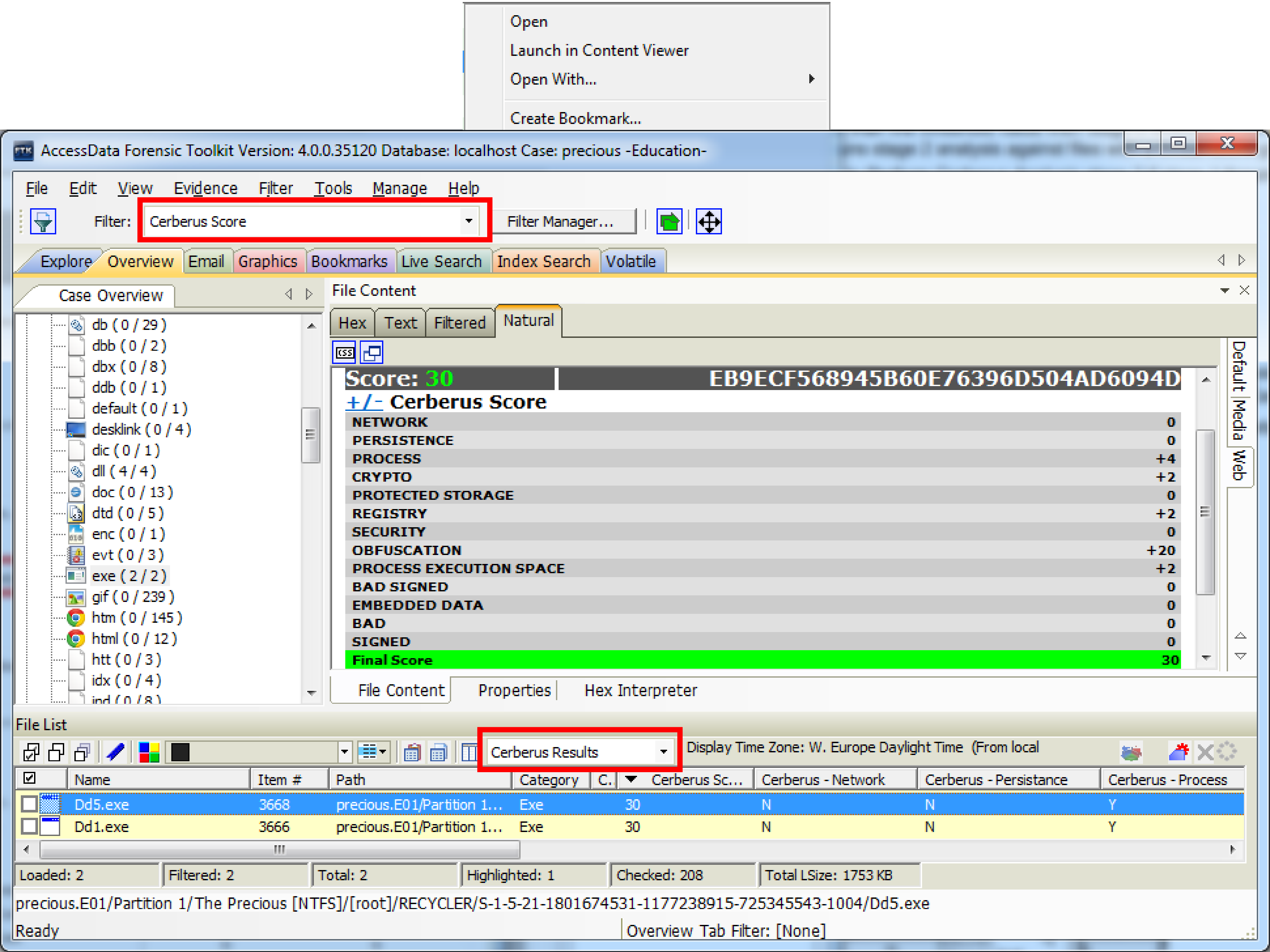
- Before you begin check if you are allowed to examine!
- Question to answer - what are the true functions and capabilities of the file/program?
- Deep knowledge about the program may give additional benefits as
  - Anti-... methods
  - Damage control know how
  - Info about the creator

Two analysis methods

- Before you begin with any deeper analyse
  - Document as much as you know about the file
    - System location, OS, full path to file, etc.
    - Who found it?
  - Run a malware scan (can give quick result!)
- Static analysis
  - No execution
  - Extensive search in the binary with various tools
- Dynamic analysis
  - Execution
  - Extensive monitoring
  - Alter the execution and program flow

Automatic malware analysis

- Scan malware with different AntiVirus agents
  - If there is an alert, research AV manufacturers websites
  - If analysis is already done – 90% of your job <u>may</u> be done ☺
    - AV report can be faulty, malcode may be of a new variant etc.
- Web based static and dynamic analyze
  - http://www.virustotal.com
  - http://www.sunbeltsecurity.com - ThreatTrack Security
  - http://metascan-online.com/
- Indicators of Compromise (IOCs)
  - Mandiant IOC Editor and Finder
  - iDefense MAP (Malcode Analyst Pac)
  - FTK – Cerberus
- Many other various solutions – Search! Landscape is changing constantly
- ethical-hacker.net > Blog (Tools and Techniques)
  - http://ethicalhackernet.blogspot.com/2008_04_01_archive.html



AccessData Forensic Toolkit Version: 4.0.0.35120 Database: localhost Case: precious -Education-

Filter: Cerberus Score

- db (0 / 29)
- dbb (0 / 2)
- dbx (0 / 8)
- ddb (0 / 1)
- default (0 / 1)
- desklink (0 / 4)
- dic (0 / 1)
- dll (4 / 4)
- doc (0 / 13)
- dtd (0 / 5)
- enc (0 / 1)
- evt (0 / 3)
- exe (2 / 2)
- gif (0 / 239)
- htm (0 / 145)
- html (0 / 12)
- htt (0 / 3)
- idx (0 / 4)
- ind (0 / 8)

Score: 30 EB9ECF568945B60E76396D504AD6094D

+/- Cerberus Score

NETWORK	0
PERSISTENCE	0
PROCESS	+4
CRYPTO	+2
PROTECTED STORAGE	0
REGISTRY	+2
SECURITY	0
OBFUSCATION	+20
PROCESS EXECUTION SPACE	+2
BAD SIGNED	0
EMBEDDED DATA	0
BAD	0
SIGNED	0
Final Score	30

Cerberus Results

Display Time Zone: W. Europe Daylight Time (From local

☑	Name	Item #	Path	Category	C.	Cerberus Sc...	Cerberus - Network	Cerberus - Persistance	Cerberus - Process
□	Dd5.exe	3668	precious.E01/Partition 1...	Exe		30	N	N	Y
□	Dd1.exe	3666	precious.E01/Partition 1...	Exe		30	N	N	Y

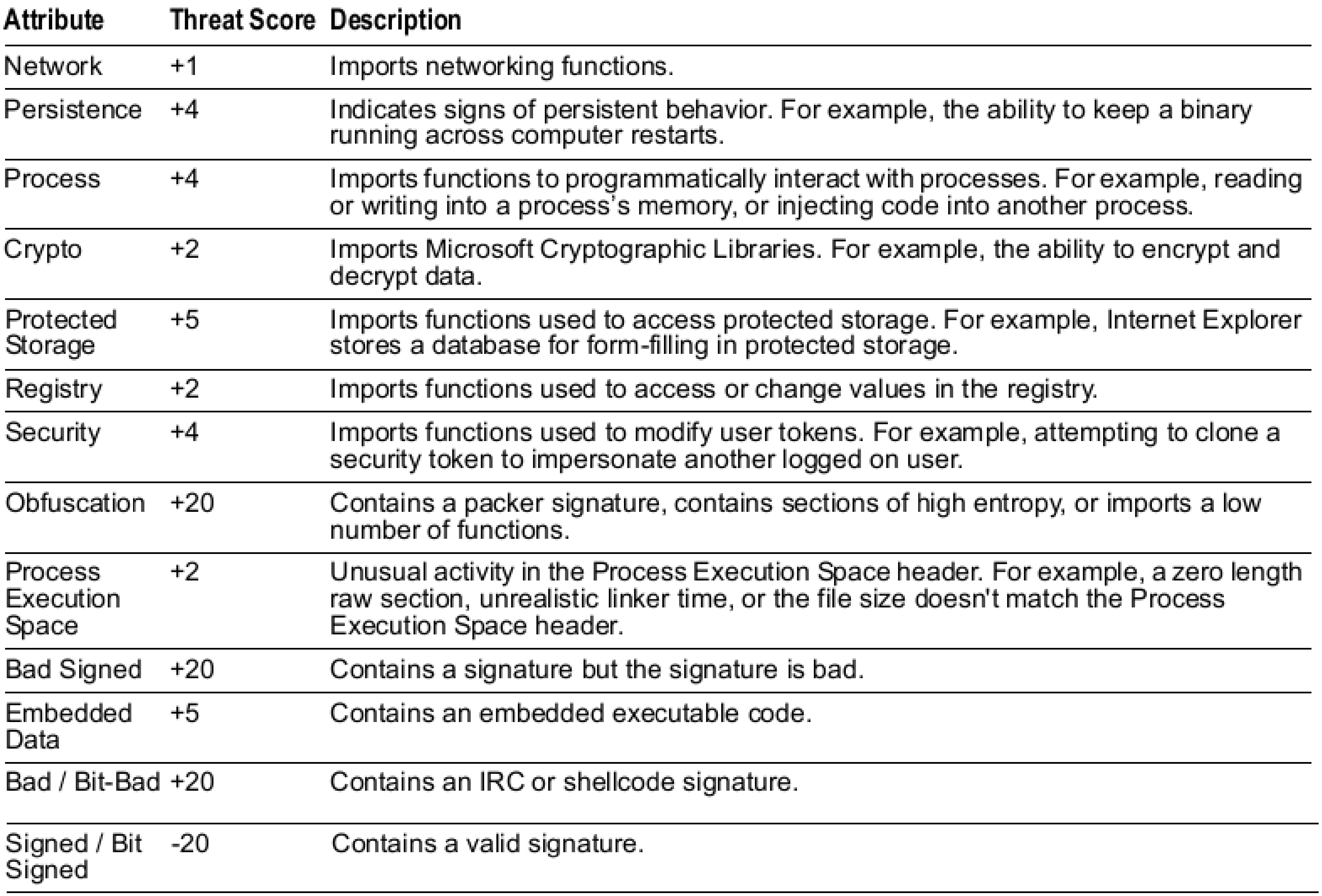
Loaded: 2 | Filtered: 2 | Total: 2 | Highlighted: 1 | Checked: 208 | Total LSize: 1753 KB

precious.E01/Partition 1/The Precious [NTFS]/[root]/RECYCLER/S-1-5-21-1801674531-1177238915-725345543-1004/Dd5.exe

Ready

Overview Tab Filter: [None]

Cerberus Stage 1 Score



Attribute	Threat Score	Description
Network	+1	Imports networking functions.
Persistence	+4	Indicates signs of persistent behavior. For example, the ability to keep a binary running across computer restarts.
Process	+4	Imports functions to programmatically interact with processes. For example, reading or writing into a process's memory, or injecting code into another process.
Crypto	+2	Imports Microsoft Cryptographic Libraries. For example, the ability to encrypt and decrypt data.
Protected Storage	+5	Imports functions used to access protected storage. For example, Internet Explorer stores a database for form-filling in protected storage.
Registry	+2	Imports functions used to access or change values in the registry.
Security	+4	Imports functions used to modify user tokens. For example, attempting to clone a security token to impersonate another logged on user.
Obfuscation	+20	Contains a packer signature, contains sections of high entropy, or imports a low number of functions.
Process Execution Space	+2	Unusual activity in the Process Execution Space header. For example, a zero length raw section, unrealistic linker time, or the file size doesn't match the Process Execution Space header.
Bad Signed	+20	Contains a signature but the signature is bad.
Embedded Data	+5	Contains an embedded executable code.
Bad / Bit-Bad	+20	Contains an IRC or shellcode signature.
Signed / Bit Signed	-20	Contains a valid signature.

Static analysis methods (Linux)

- Hash the file
- File
  - Properties and type of file etc.
- Strings
- Hexdump
- Nm
  - List symbol info
- Ldd
  - View shared objects which is linked in at runtime
  - Listed in the .interp section
- Readelf, elfdump, objdump

```
hjo@lnx:~/$ file winkill
winkill: ELF 32-bit LSB executable, Intel
80386, version 1 (SYSV), for GNU/Linux
2.0.0, dynamically linked (uses shared libs),
for GNU/Linux 2.0.0, not stripped
```

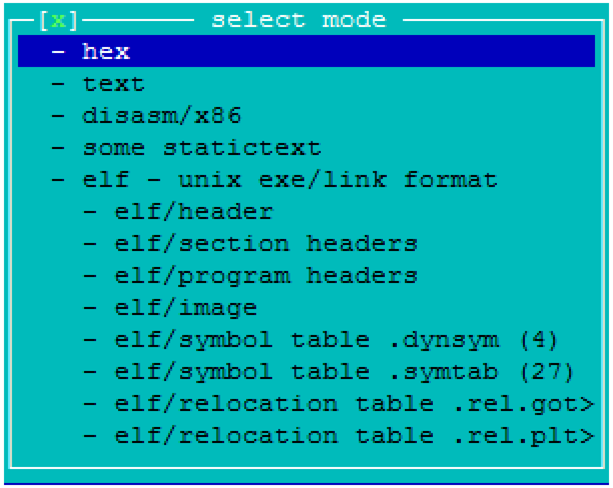
```
hjo@lnx:~/$ nm winkill
...
08048784 T parse_args
08049c78 D port
         U printf@@GLIBC_2.0
08048760 T usage
         U usleep@@GLIBC_2.0
…
D  The symbol is in the initialized .data section
T  The symbol is in the .text (code) section
U  The symbol is unknown
…
```

```
hjo@lnx:~/$ ldd winkill
        linux-gate.so.1 =>  (0xffffe000)
        libc.so.6 => /lib/tls/i686/cmov/libc.so.6
(0xb7e36000)
        /lib/ld-linux.so.2 (0xb7f70000)
```

Readelf

```
hjo@lnx:~/$ readelf
Usage: readelf <option(s)> elf-file(s)
 Display information about the contents of ELF format files
 Options are:
  -a --all               Equivalent to: -h -l -S -s -r -d -V -A -I
  -h --file-header       Display the ELF file header
  -l --program-headers   Display the program headers
     --segments          An alias for --program-headers
  -S --section-headers   Display the sections' header
     --sections          An alias for --section-headers
  -g --section-groups    Display the section groups
  -t --section-details   Display the section details
  -e --headers           Equivalent to: -h -l -S
  -s --syms              Display the symbol table
      --symbols          An alias for --syms
  -n --notes             Display the core notes (if present)
  -r --relocs            Display the relocations (if present)
  -u --unwind            Display the unwind info (if present)
  -d --dynamic           Display the dynamic section (if present)
  -V --version-info      Display the version sections (if present)
  -A --arch-specific     Display architecture specific information (if any).
  -D --use-dynamic       Use the dynamic section info when displaying symbols
  -x --hex-dump=<number> Dump the contents of section <number>
  -w[liaprmfFsoR] or
  --debug-dump[=line,=info,=abbrev,=pubnames,=aranges,=macro,=frames,=str,=loc,=Ranges]
                         Display the contents of DWARF2 debug sections
  -I --histogram         Display histogram of bucket list lengths
  -W --wide              Allow output width to exceed 80 characters
  @<file>                Read options from <file>
  -H --help              Display this information
  -v --version           Display the version number of readelf
Report bugs to <URL:http://www.sourceware.org/bugzilla/>
```

Objdump and HT Editor


[x] select mode
- hex
- text
- disasm/x86
- some statictext
- elf - unix exe/link format
- elf/header
- elf/section headers
- elf/program headers
- elf/image
- elf/symbol table .dynsym (4)
- elf/symbol table .symtab (27)
- elf/relocation table .rel.got>
- elf/relocation table .rel.plt>

HT Editor - http://hte.sourceforge.net/

- Provides readelf functions and further probing of contents

- Disassemble

```
-d, --disassemble       Display assembler contents of executable sections
-D, --disassemble-all    Display assembler contents of all sections
```

  - Convert from binary to assembly code
    - Dead listing
  - hjo@lnx:~/$ objdump -d winkill

```
08048874 <main>:
 8048874:       55                      push   %ebp
 8048875:       89 e5                   mov    %esp,%ebp
 8048877:       81 ec b8 3a 00 00       sub    $0x3ab8,%esp
 804887d:       c7 45 e8 98 3a 00 00    movl
$0x3a98,0xffffffe8(%ebp)
 8048884:       83 7d 08 01             cmpl   $0x1,0x8(%ebp)
 8048888:       7f 0e                   jg     8048898 <main+0x24>
 804888a:       8b 45 0c                mov    0xc(%ebp),%eax
 804888d:       8b 10                   mov    (%eax),%edx
 804888f:       52                      push   %edx
 8048890:       e8 cb fe ff ff          call   8048760 <usage>
 8048895:       83 c4 04                add    $0x4,%esp
...
```

```
08048760 <usage>:
 8048760:       55                      push   %ebp
 8048761:       89 e5                   mov    %esp,%ebp
 8048763:       8b 45 08                mov    0x8(%ebp),%eax
 8048766:       50                      push   %eax
 8048767:       68 80 8b 04 08          push   $0x8048b80
 804876c:       e8 97 fe ff ff          call   8048608
<printf@plt>
...
```

This is an excerpt from the output!

Dynamic analysis methods

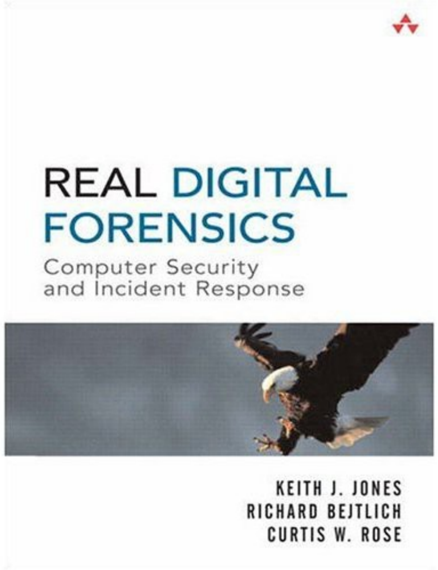
- Safe controlled isolated lab
  - Assume the worst!
- System Call Trace (strace)
  - hjo@lnx:~/$ strace -d ./winkill
  - Library Call Trace (ltrace)
  - *trace got similar options

```
hjo@lnx:~/$ ltrace ./winkill
__libc_start_main(0x8048874, 1, 0xbfd3d314, 0x8048528,
0x8048b2c <unfinished ...>
__register_frame_info(0x8049c7c, 0x8049d90,
0xbfd3d298, 0x804854d, 0xb7faaff4)    = 0
printf("Usage: %s <host> -p port -t hits"...,
"./winkill"Usage: ./winkill <host> -p port -t hits
)                        = 40
exit(1 <unfinished ...>
__deregister_frame_info(0x8049c7c, 0xbfd397a8,
0x8048b41, 0xb7faaff4, 0xbfd397c8) = 0
+++ exited (status 1) +++
```

- The GNU debugger, http://www.gnu.org/software/gdb/
  - Huge subject
    - Google on ”gnu debugger gdb tutorial”
  - Stop program execution
  - Control program flow
  - Examine data structures
  - Disassemble etc. etc. etc. ...

Further analysis!

- RDF chapters 13, 14 and 15 are elite!
- Ch 14 deals with
  - Advanced static options
  - Advanced dynamic options
  - Unlink an unpacked tmp file
    - Open and execve the deleted tmp file
  - Generate core file (process dump)
    - ulimit -c unlimited (to enable core)
    - kill -s SIGSEV <PID> (from another console) other signals which action is core should do aswell, SIGSEV = Invalid memory reference
    - Check out the Linux manual: man signal
  - Examine core files with gdb
  - Packers
  - RCE etc. ...



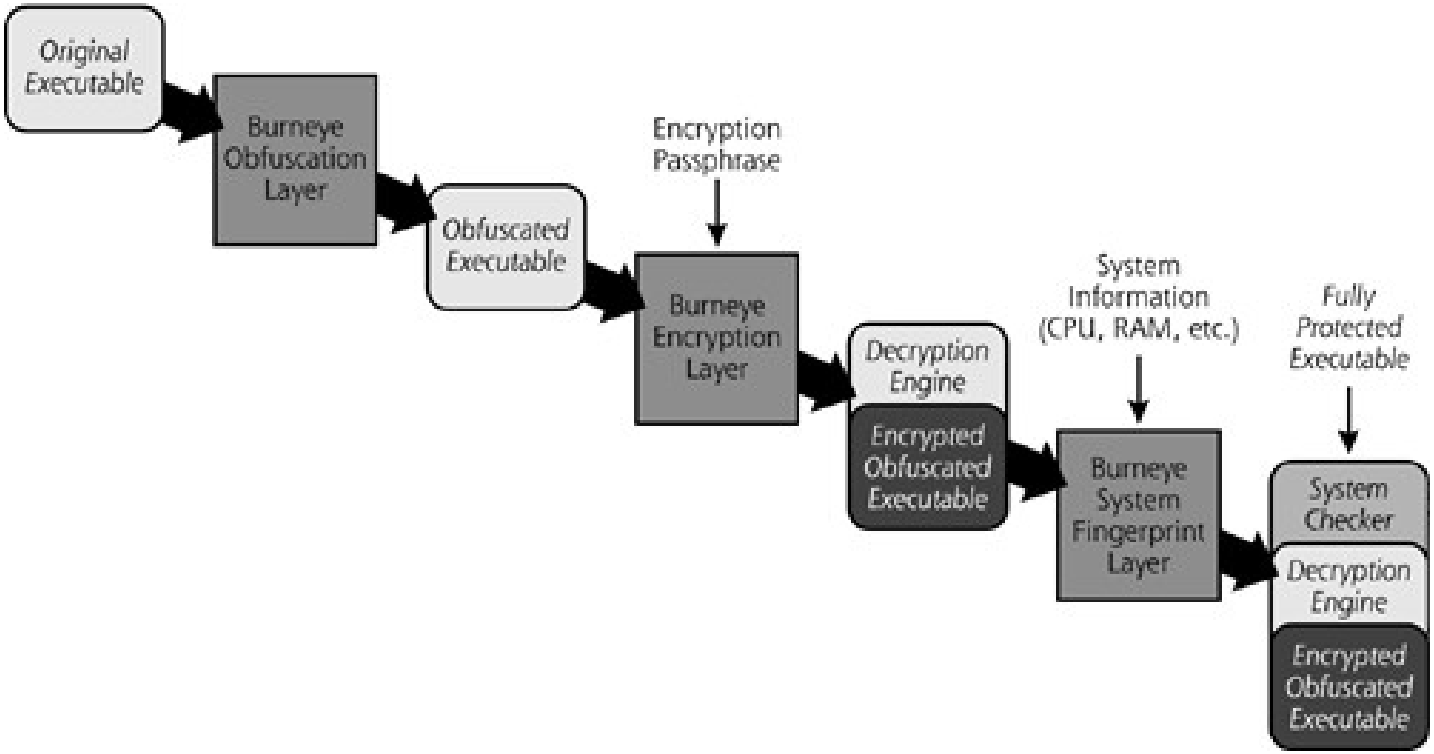
Further analysis...

- Different methods to recover a unpackable packed binary...
  - Debugfs
    - ext2/ext3 file system debugger
    - Similar to ifind and icat as in SITIC/CERT course exercise but on a deleted file
  - Strace hexdump – output all
    - In combination with hexeditor (cut and paste) rebuild binary
  - /proc pseudo file system
    - ls -al /proc/<PID>/
    - # man proc
    - Copy the exe link
  - Packers as UPX(nrv/ucl)
    - First try to unpack with packer versions
    - Note that programmer may have ”edited” away traces of used packers with a hexeditor
  - Crypt-packers as Burneye

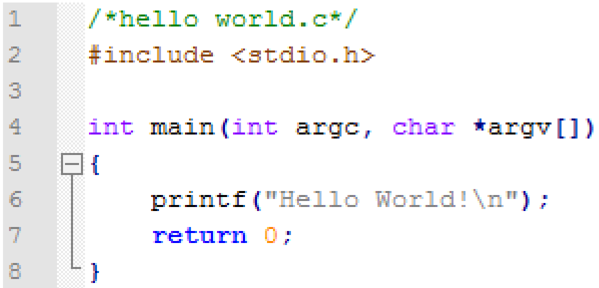
```
hjo@lnx:~/$ ls -al /proc/29279/
dr-xr-xr-x  5 hjo  hjo  0 Feb  6 12:56 .
dr-xr-xr-x 82 root root 0 Nov  7 11:49 ..
-r--------  1 root root 0 Feb  6 12:57 auxv
--w-------  1 root root 0 Feb  6 12:57 clear_refs
-r--r--r--  1 root root 0 Feb  6 12:56 cmdline
-rw-r--r--  1 root root 0 Feb  6 12:57 coredump_filter
lrwxrwxrwx  1 root root 0 Feb  6 12:57 cwd -> /
-r--------  1 root root 0 Feb  6 12:57 environ
lrwxrwxrwx  1 root root 0 Feb  6 12:57 exe ->
/tmp/upxRandName (deleted)
dr-x------  2 root root 0 Feb  6 12:57 fd
dr-x------  2 root root 0 Feb  6 12:57 fdinfo
-r--------  1 root root 0 Feb  6 12:57 limits
-r--r--r--  1 root root 0 Feb  6 12:57 maps
-rw-------  1 root root 0 Feb  6 12:57 mem
-r--r--r--  1 root root 0 Feb  6 12:57 mounts
-r--------  1 root root 0 Feb  6 12:57 mountstats
-rw-r--r--  1 root root 0 Feb  6 12:57 oom_adj
-r--r--r--  1 root root 0 Feb  6 12:57 oom_score
lrwxrwxrwx  1 root root 0 Feb  6 12:57 root -> /
-r--r--r--  1 root root 0 Feb  6 12:57 smaps
-r--r--r--  1 root root 0 Feb  6 12:56 stat
-r--r--r--  1 root root 0 Feb  6 12:57 statm
-r--r--r--  1 root root 0 Feb  6 12:56 status
dr-xr-xr-x  3 hjo  hjo  0 Feb  6 12:57 task
-r--r--r--  1 root root 0 Feb  6 12:57 wchan
```

Burneye's three layers of executable protection

- Scrambles the code in the executable thru **obfuscated** instructions
- **Encryption** of the binary program
- **System fingerprint** – will only run on certain computers


Original Executable
Burneye Obfuscation Layer
Obfuscated Executable
Encryption Passphrase
Burneye Encryption Layer
Decryption Engine
Encrypted Obfuscated Executable
System Information (CPU, RAM, etc.)
Burneye System Fingerprint Layer
Fully Protected Executable
System Checker
Decryption Engine
Encrypted Obfuscated Executable

Windows dev. tools



```
1  /*hello world.c*/
2  #include <stdio.h>
3
4  int main(int argc, char *argv[])
5  {
6      printf("Hello World!\n");
7      return 0;
8  }
```

- Enable a C/C++ compiler (since it not is bundled with OS)
  - Goal is to run cl.exe (as gcc/g++ in Linux)
- Visual C++ 20xx Express Edition or other free download
- Visual Studio 20xx
  - Run the ”Developer Command Prompt for VS20xx” cmd in program menu if you want cmd line enabled
- Generate an ASM listing
  - cl.exe /?
    - /Fa[file] name assembly listing file
- cl.exe /Fahello.asm hello.c
  - hello.asm, hello.exe, hello.obj

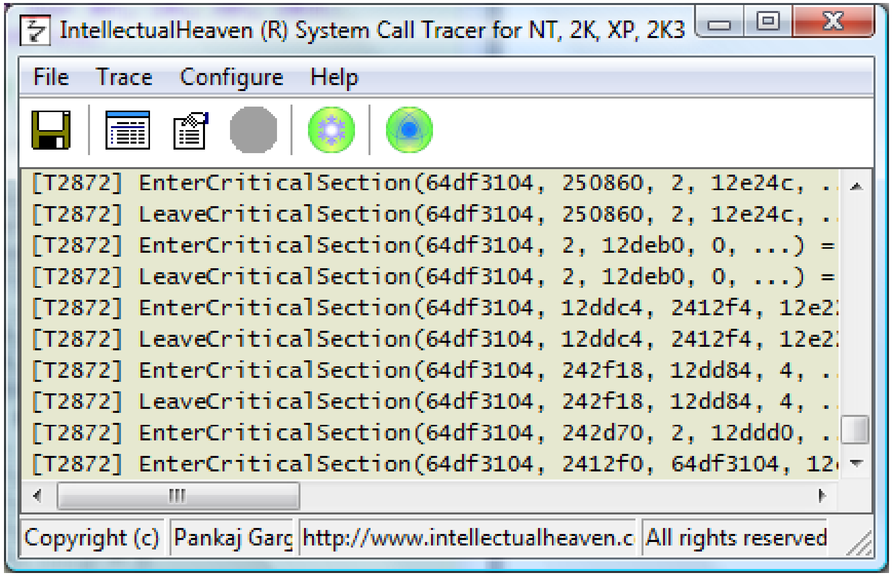
```
; Listing generated by Microsoft (R) Optimizing
Compiler Version 14.00.50727.762
       TITLE  C:\data\ppt\hello\hello.c
       .686P
       .XMM
       include listing.inc
       .model flat

INCLUDELIB LIBCMT
INCLUDELIB OLDNAMES

_DATA  SEGMENT
$SG2245       DB     'Hello World!', 0aH, 00H
_DATA  ENDS
PUBLIC _main
EXTRN  _printf:PROC
; Function compile flags: /Odtp
_TEXT  SEGMENT
_argc$ = 8                    ; size = 4
_argv$ = 12                   ; size = 4
_main  PROC
; File c:\data\hello\hello.c
; Line 5
       push   ebp
       mov    ebp, esp
; Line 6
       push   OFFSET $SG2245
       call   _printf
       add    esp, 4
; Line 7
       xor    eax, eax
; Line 8
       pop    ebp
       ret    0
_main  ENDP
_TEXT  ENDS
END
```

Static analysis methods (Windows)

- Option 1
  - Install Cygwin (Unix shell in windows)
  - Use same Unix/Linux programs/commands as earlier described
    - Some may not be ported and some may be ported to native Windows
  - md5, file, strings, ldd, hexedit, nm, objdump
  - Examining PE structure
    - pe_map (almost as objdump)
- Option 2
  - dumpbin.exe or link.exe, need the VsDevCmd.bat to be set as well
    - They got similar output (as nm and objdump) link.exe -dump -all <file>
  - Use Windows third party tools for above as
    - Hex editors, Sysinternals tools, etc.
    - Earlier mentioned PE/COFF tools
    - Disassemble with IDA Pro etc. (Google search for others)

Dynamic analysis methods (Windows)

- System call trace (ported)
  - Strace NT 0.8 beta
- Dependency Walker (profile runtime dependencies)
- Debug exe (open or attach)
  - MS Visual Studio IDE
    - http://msdn.microsoft.com/en-us/library/0bxe8ytt.aspx
  - Gdb, ddd, OllyDbg, IDA Pro, etc...


IntellectualHeaven (R) System Call Tracer for NT, 2K, XP, 2K3
File
Trace
Configure
Help

```
[T2872] EnterCriticalSection(64df3104, 250860, 2, 12e24c, ..
[T2872] LeaveCriticalSection(64df3104, 250860, 2, 12e24c, ..
[T2872] EnterCriticalSection(64df3104, 2, 12deb0, 0, ...) =
[T2872] LeaveCriticalSection(64df3104, 2, 12deb0, 0, ...) =
[T2872] EnterCriticalSection(64df3104, 12ddc4, 2412f4, 12e2
[T2872] LeaveCriticalSection(64df3104, 12ddc4, 2412f4, 12e2
[T2872] EnterCriticalSection(64df3104, 242f18, 12dd84, 4, ..
[T2872] LeaveCriticalSection(64df3104, 242f18, 12dd84, 4, ..
[T2872] EnterCriticalSection(64df3104, 242d70, 2, 12ddd0, ..
[T2872] EnterCriticalSection(64df3104, 2412f0, 64df3104, 12
```

- Sysinternals Process Monitor (or the separate tools)
  - File, registry, network and process monitor in the same package!
- Reverse Code Engineer the file
  - Debug the generated assembly list files
  - Tracking down how a program works can take weeks or more!

Dynamic analysis methods (Windows)

- Virtualization
- Ghost images for quick reload of OS
- Other tools as
  - Wireshark, ...
  - Port Reporter tool
    - A service logging all TCP and UDP traffic
      - http://support.microsoft.com/kb/837243
    - Parser tool for Port Reporter
      - http://support.microsoft.com/kb/884289
    - Webcast – usage
      - http://support.microsoft.com/kb/840832/

- The ports that are used
- The processes that use the port
- Whether a process is a service
- The modules that a process loaded
- The user accounts that run a process

- Microsoft Windows GUI to review the logs
- Identifying suspicious data or data that you are interested in
- Analyzing the logs and generating data

Dynamic analysis methods (Windows)

- Enable auditing for process tracking in event log (failure and success events)
  - auditpol.exe /enable /process:all
- Non real-time registry or file snapshot tools
  - As RegShot and Incontrol5
  - Not to be used for longer time since you don’t see
    - Keys or files that have been searched for
    - Timeline when keys or files were accessed
- Dump the process – depending on demands
  - .dmp format tools as ProcDump (Sysinternals) etc.
  - PE format tools as ProcDump32 v1.6.2, LordPE dlx b v1.41
- Dump the RAM and examine as we did earlier in the course and analyze

Zero Wine: Malware Behavior Analysis

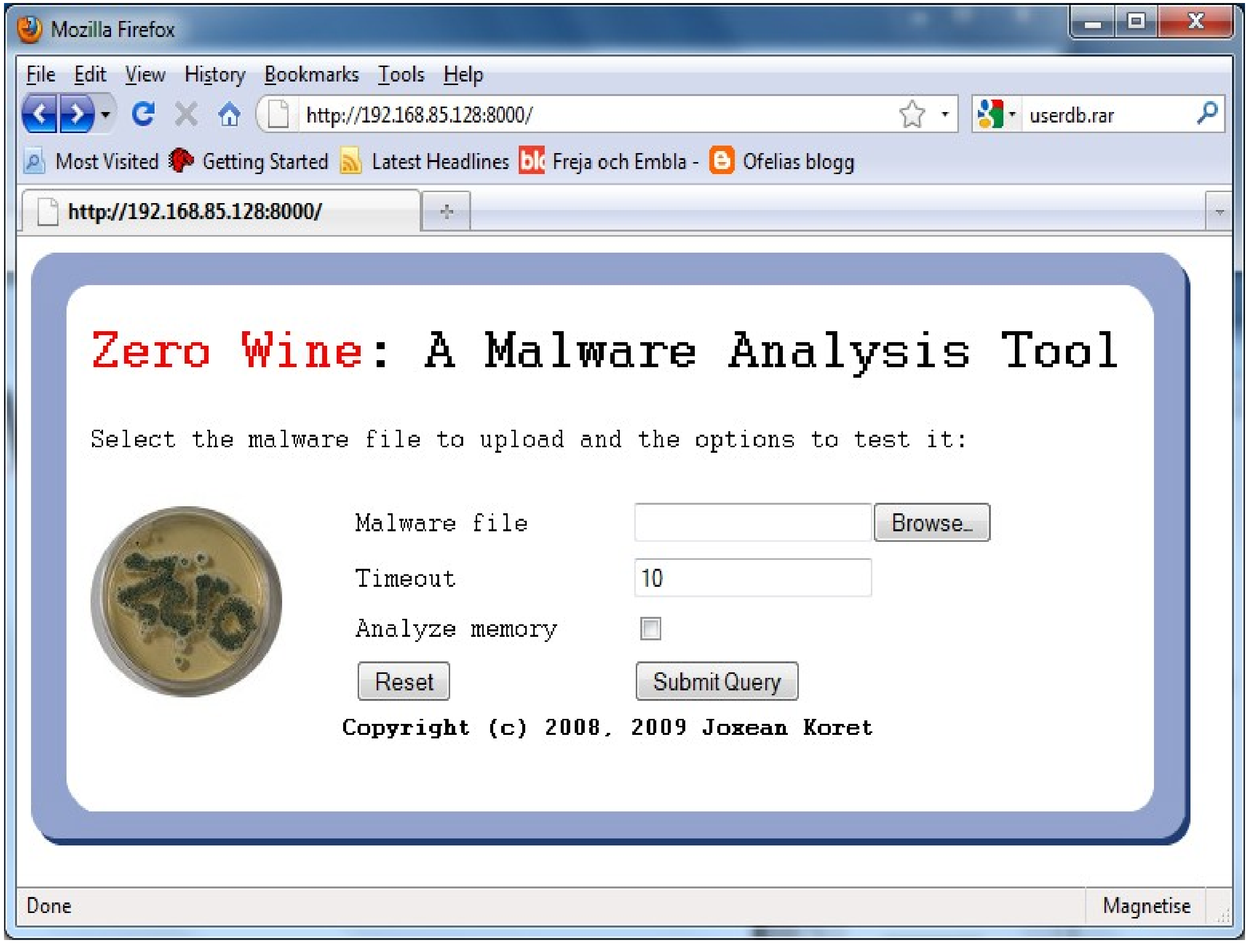
Upload malware perform static and dynamic analyze

Same userdb.txt (signatures) as PEiD

Virtual machine using Qemu or VMware and Linux/Wine

Output:

- Raw trace (Report)
- Strings
- PE headers
- Signature (API calls)



Zero Wine: A Malware Analysis Tool

Select the malware file to upload and the options to test it:

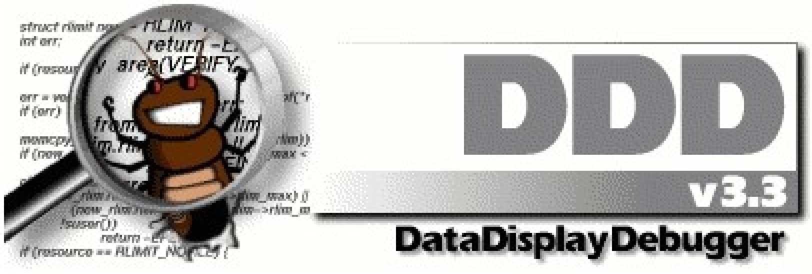
Malware file	Browse...
Timeout	10
Analyze memory	□

Copyright (c) 2008, 2009 Joxean Koret

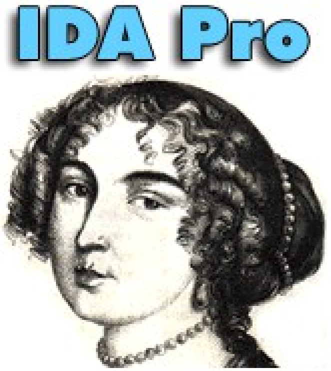
http://zerowine.sourceforge.net/

Static and dynamic verfication

- Verify diffrence/similarity between examined file and assumed source code/binary in ”the wild”
- Compare output
  - With diff or other line by line tool
  - Functions with nm
  - Strings
  - Assembly code side by side
  - Ssdeep, nwdiff, bindiff (binary)
- strace, ltrace
- Gdb/ddd or other tools as IDA Pro, OllyDbg
  - http://www.gnu.org/software/ddd/
  - http://www.hex-rays.com/idapro/
  - http://www.ollydbg.de/
- Practical usage testing and monitoring
  - lsof, netstat, wireshark etc. (live response methods)

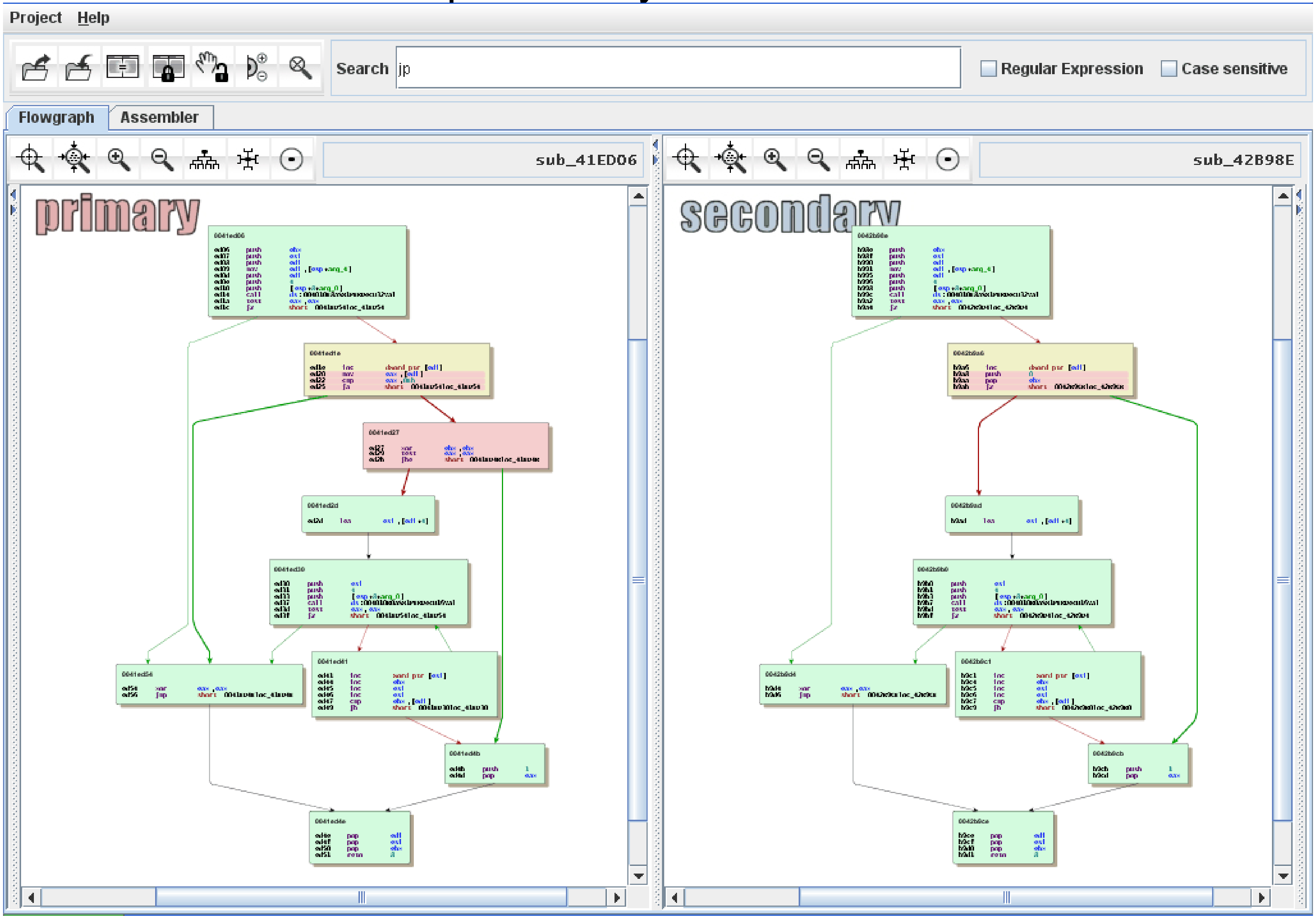


OllyDbg
Win32 Symbolic Debugger



BinDiff and BinNavi (IDA Pro addon)

http://www.zynamics.com



Search: jp

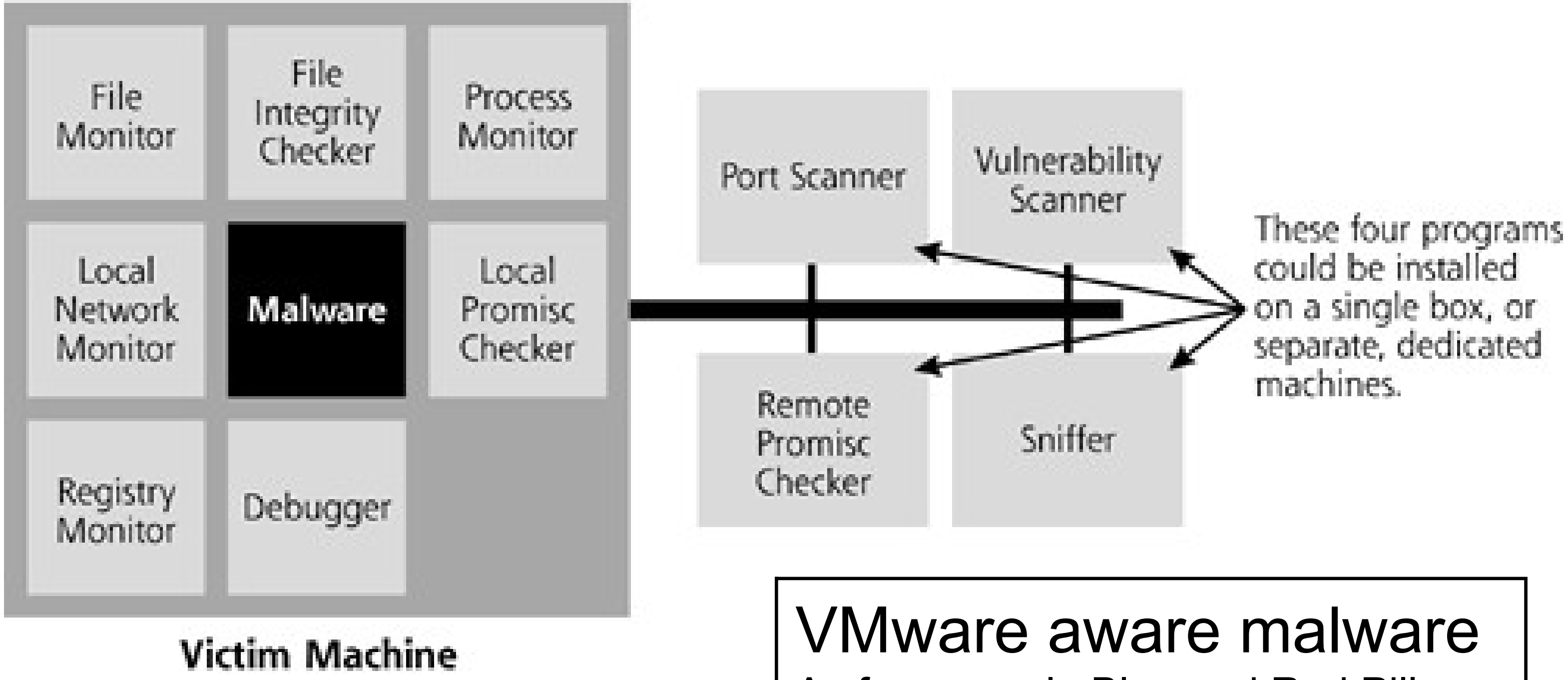
□ Regular Expression

□ Case sensitive

sub_41ED06

sub_42B98E

Dynamic analysis



These four programs could be installed on a single box, or separate, dedicated machines.

VMware aware malware

As for example Blue and Red Pill

http://www.invisiblethings.org

http://bluepillproject.org

Debugger aware malware

PEB (Process Environment Block) struct got a member variable:

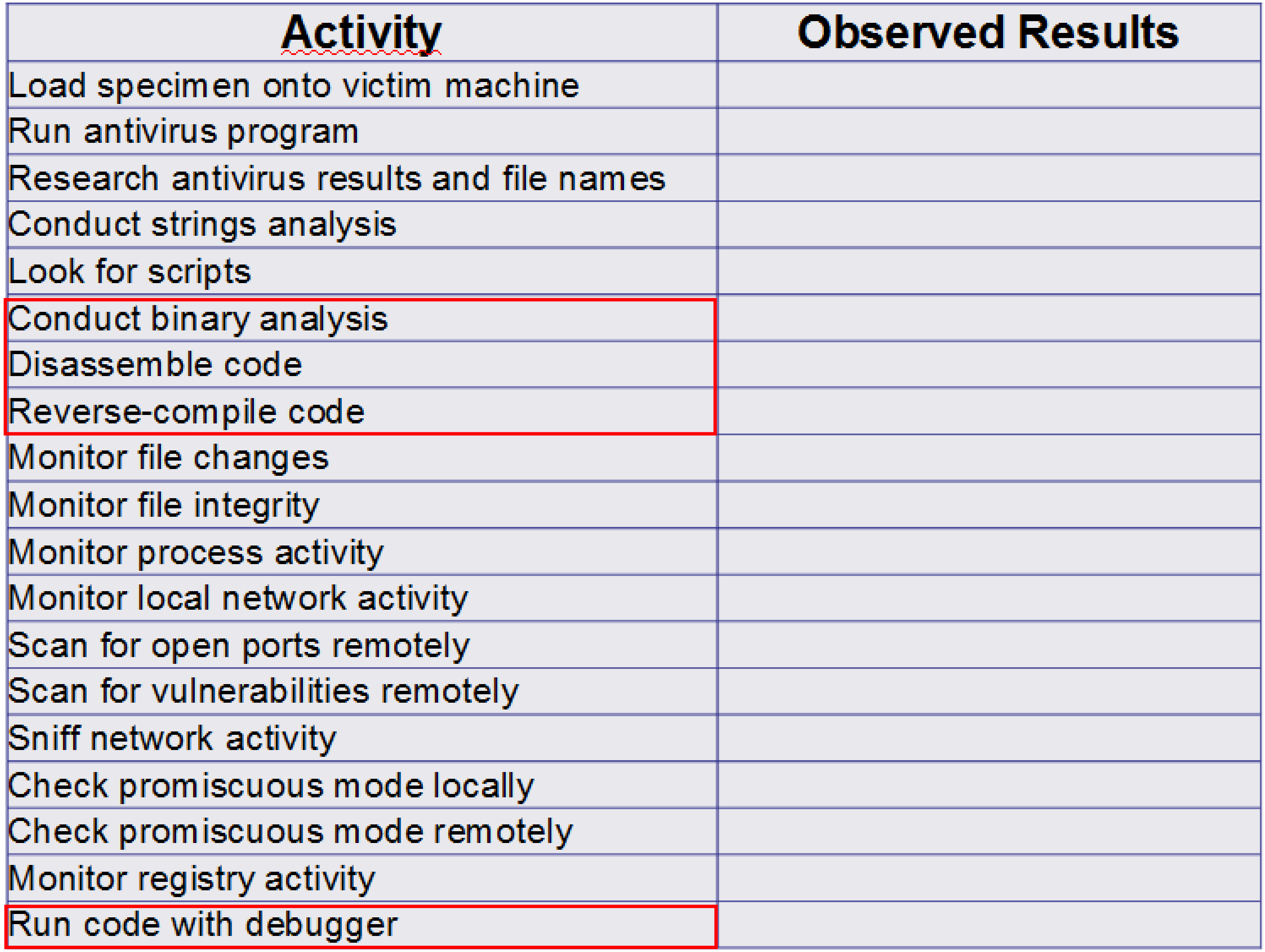
UCHAR BeingDebugged;

Malware check itself if being debugged!

```
int swallow_redpill () {
  unsigned char m[2+4], rpill[] = "\x0f\x01\x0d\x00\x00\x00\x00\xc3";
  *((unsigned*)&rpill[3]) = (unsigned)m;
  ((void(*)())&rpill)();
  return (m[5]>0xd0) ? 1 : 0;
}
```

Malware analysis template

http://www.counterhack.net/malware_template.html



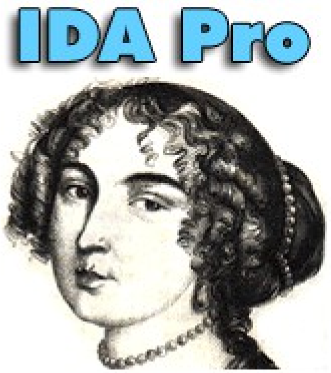
Activity	Observed Results
Load specimen onto victim machine	
Run antivirus program	
Research antivirus results and file names	
Conduct strings analysis	
Look for scripts	
Conduct binary analysis	
Disassemble code	
Reverse-compile code	
Monitor file changes	
Monitor file integrity	
Monitor process activity	
Monitor local network activity	
Scan for open ports remotely	
Scan for vulnerabilities remotely	
Sniff network activity	
Check promiscuous mode locally	
Check promiscuous mode remotely	
Monitor registry activity	
Run code with debugger	

LAB

Static analysis

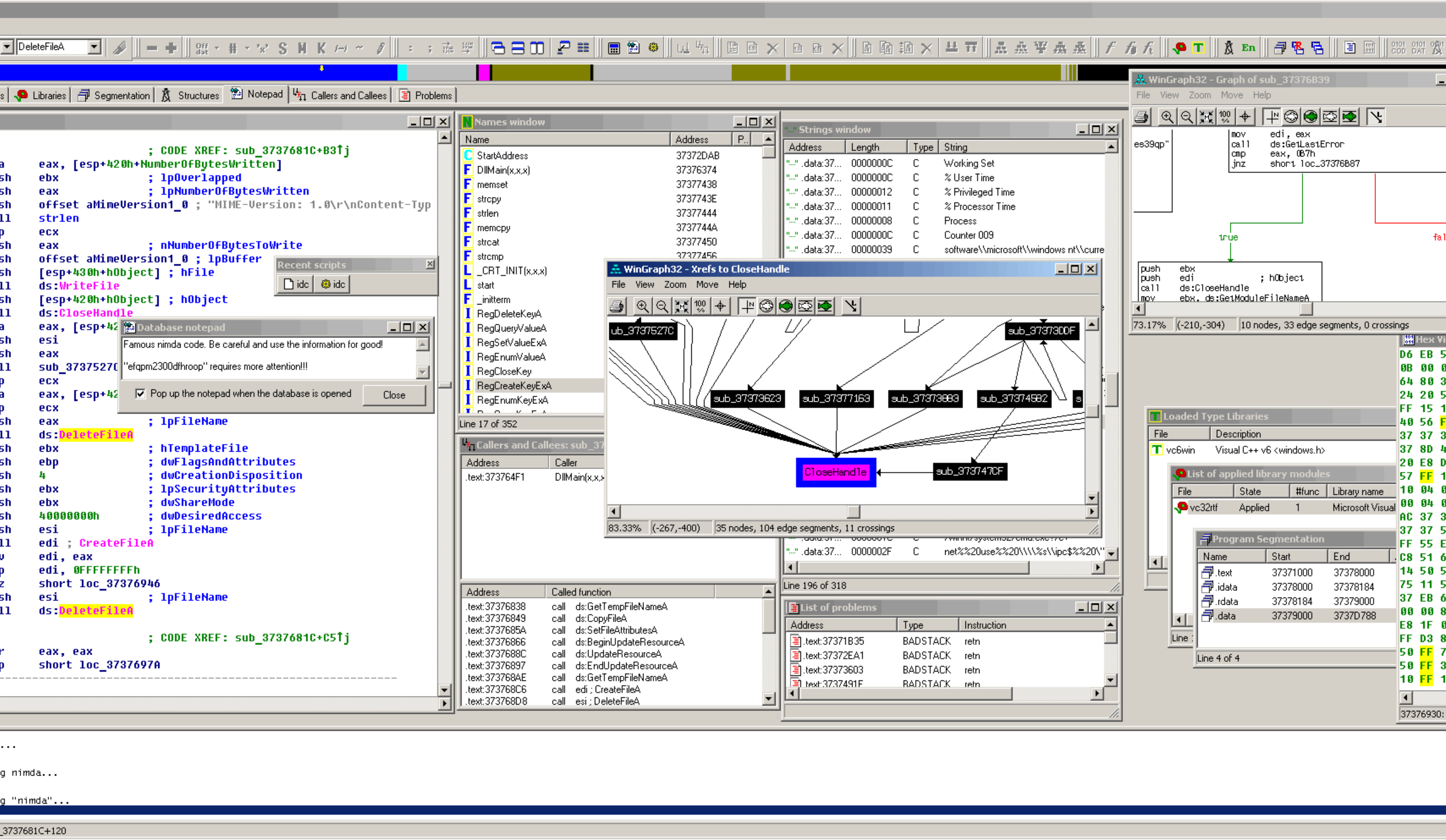
Dynamic analysis

IDA Pro



- What is IDA Pro?
  - Disassembler and debugger (also remote debugging)
  - Interactive (change app state) and programmable (scripts and plugins)
- How is IDA Pro useful?
  - Hostile Code analysis and vulnerability research
  - COTS validation, privacy protection and patent infringements
- Who are the IDA Pro users?
  - Anti-virus companies and vulnerability research companies
  - Large software development companies and forensic software companies
  - Three letter agencies, military organizations and patent trolls
- Highlights
  - FLIRT (Fast Library Identification and Recognition Technology)
    - Recognize and annotate (data information about) library functions
    - Saving significant amounts of time tracing through machine code
  - Obfuscated code analysis (bypass obfuscation) and many other things

IDA Pro basics



```
                              ; CODE XREF: sub_3737681C+B3↑j
a      eax, [esp+420h+NumberOfBytesWritten]
sh     ebx                    ; lpOverlapped
sh     eax                    ; lpNumberOfBytesWritten
sh     offset aMimeVersion1_0 ; "MIME-Version: 1.0\r\nContent-Typ
ll     strlen
p      ecx
sh     eax                    ; nNumberOfBytesToWrite
sh     offset aMimeVersion1_0 ; lpBuffer
sh     [esp+430h+hObject] ; hFile
ll     ds:WriteFile
sh     [esp+420h+hObject] ; hObject
ll     ds:CloseHandle
a      eax, [esp+42
sh     esi
sh     eax
ll     sub_3737527C
p      ecx
a      eax, [esp+42
p      ecx
sh     eax                    ; lpFileName
ll     ds:DeleteFileA
sh     ebx                    ; hTemplateFile
sh     ebp                    ; dwFlagsAndAttributes
sh     4                      ; dwCreationDisposition
sh     ebx                    ; lpSecurityAttributes
sh     ebx                    ; dwShareMode
sh     40000000h              ; dwDesiredAccess
sh     esi                    ; lpFileName
ll     edi ; CreateFileA
v      edi, eax
p      edi, 0FFFFFFFFh
z      short loc_37376946
sh     esi                    ; lpFileName
ll     ds:DeleteFileA

                              ; CODE XREF: sub_3737681C+C5↑j
r      eax, eax
p      short loc_3737697A
```

Database notepad

Famous nimda code. Be careful and use the information for good!

"efqpm2300dfhroop" requires more attention!!!

☑ Pop up the notepad when the database is opened   Close

Names window

Name	Address	P..
C StartAddress	37372DAB	
F DllMain(x,x,x)	37376374	
F memset	37377438	
F strcpy	3737743E	
F strlen	37377444	
F memcpy	3737744A	
F strcat	37377450	
F strcmp	37377456	
L _CRT_INIT(x,x,x)		
L start		
F _initterm		
I RegDeleteKeyA		
I RegQueryValueA		
I RegSetValueExA		
I RegEnumValueA		
I RegCloseKey		
I RegCreateKeyExA		
I RegEnumKeyExA		

Line 17 of 352

Strings window

Address	Length	Type	String
.data:37...	0000000C	C	Working Set
.data:37...	0000000C	C	% User Time
.data:37...	00000012	C	% Privileged Time
.data:37...	00000011	C	% Processor Time
.data:37...	00000008	C	Process
.data:37...	0000000C	C	Counter 009
.data:37...	00000039	C	software\\microsoft\\windows nt\\curre
.data:37...	0000002F	C	net%%20use%%20\\\\%s\\ipc$%%20\"

Line 196 of 318

83.33%   (-267,-400)   35 nodes, 104 edge segments, 11 crossings

Callers and Callees: sub_37

Address	Caller
.text:373764F1	DllMain(x,x,x

Address	Called function
.text:37376838	call ds:GetTempFileNameA
.text:37376849	call ds:CopyFileA
.text:3737685A	call ds:SetFileAttributesA
.text:37376866	call ds:BeginUpdateResourceA
.text:3737688C	call ds:UpdateResourceA
.text:37376897	call ds:EndUpdateResourceA
.text:373768AE	call ds:GetTempFileNameA
.text:373768C6	call edi ; CreateFileA
.text:373768D8	call esi ; DeleteFileA

List of problems

Address	Type	Instruction
.text:37371B35	BADSTACK	retn
.text:37372EA1	BADSTACK	retn
.text:37373603	BADSTACK	retn
.text:3737491F	BADSTACK	retn

```
es39qp"

mov     edi, eax
call    ds:GetLastError
cmp     eax, 0B7h
jnz     short loc_37376B87

push    ebx
push    edi                     ; hObject
call    ds:CloseHandle
mov     ebx, ds:GetModuleFileNameA
```

73.17%   (-210,-304)   10 nodes, 33 edge segments, 0 crossings

Loaded Type Libraries

File	Description
vc6win	Visual C++ v6 <windows.h>

List of applied library modules

File	State	#func	Library name
vc32rtf	Applied	1	Microsoft Visual

Program Segmentation

Name	Start	End
.text	37371000	37378000
.idata	37378000	37378184
.rdata	37378184	37379000
.data	37379000	3737D788

Line 4 of 4

```
D6 EB 5
0B 00 0
64 80 3
24 20 5
FF 15 1
40 56 F
37 37 3
37 8D 4
20 E8 D
57 FF 1
10 04 0
00 04 0
AC 37 3
37 37 5
FF 55 E
C8 51 6
14 50 5
75 11 5
37 EB 6
00 00 8
E8 1F 0
FF D3 8
50 FF 7
50 FF 3
10 FF 1
37376930:
```

```
...
g nimda...
g "nimda"...
```

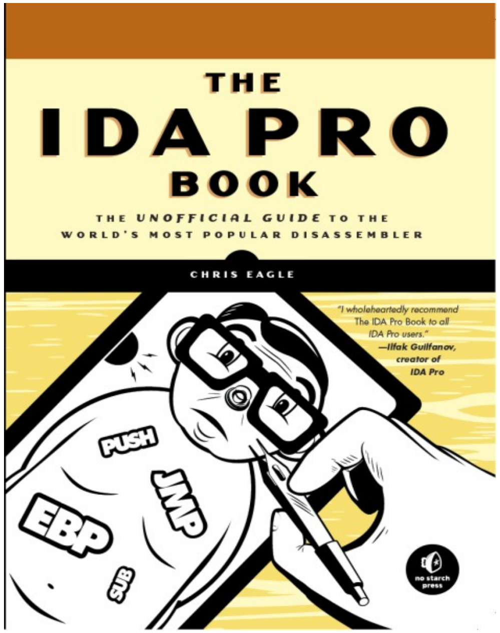
_3737681C+120

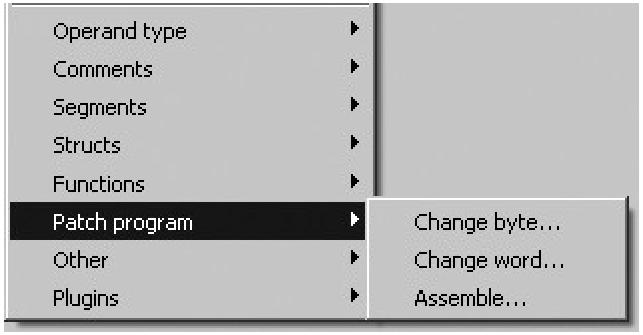
Consult the excellent tutorial IdaTut.pdf file for full coverage – included in lab

The IDA Pro Disassembler and Debugger

The most advanced tool for Hostile Code Analysis, Vulnerability Research and Software Reverse Engineering

- http://en.wikipedia.org/wiki/Interactive_Disassembler
- 5.x series have a very nice graph based interface and support for things as iPhone etc. (IDA Pro is at v6.5 now)
- Hex-Rays Decompiler plugin
  - Converts executable programs into a human readable C-like pseudo code text
- The IDA Pro Book (5 * at Amazon)
  - http://www.idabook.com/ (Chris Eagle)
- Enable the ”Edit > Patch program” menu
  - IDA is not primarly for patching binaries
  - In idagui.cfg, "DISPLAY_PATCH_SUBMENU = YES"


THE IDA PRO BOOK
THE UNOFFICIAL GUIDE TO THE WORLD'S MOST POPULAR DISASSEMBLER
CHRIS EAGLE
"I wholeheartedly recommend The IDA Pro Book to all IDA Pro users." —Ilfak Guilfanov, creator of IDA Pro
PUSH
JMP
EBP
SUB
no starch press

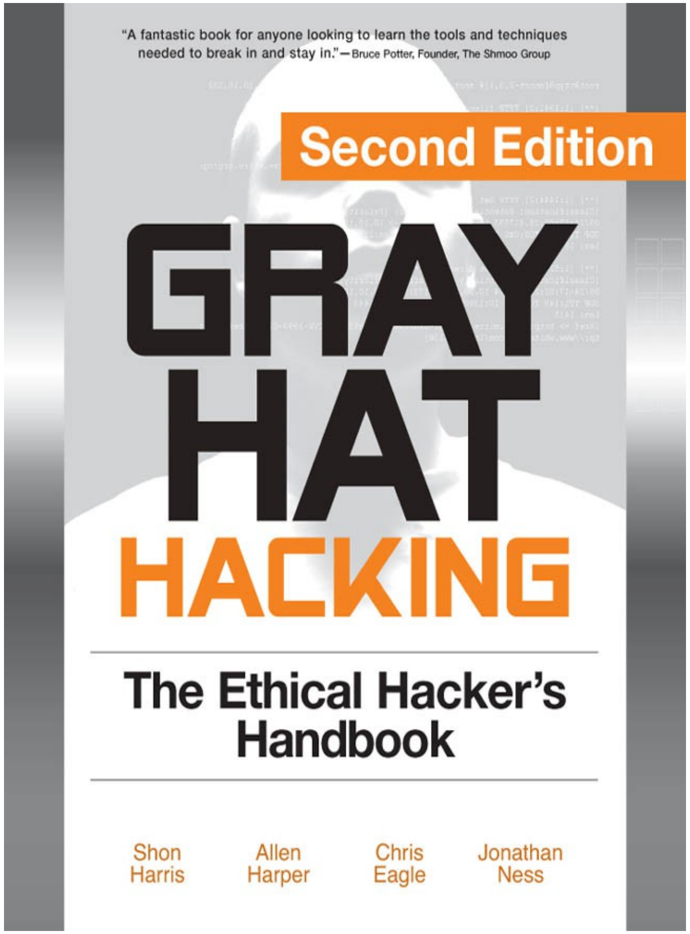


- Fix problem with .hlp files in Windows Vista/7 etc. http://support.microsoft.com/kb/917607
- Right click the .hlp file and press unblock under the General tab

The IDA Pro Disassembler and Debugger

The most advanced tool for Hostile Code Analysis, Vulnerability Research and Software Reverse Engineering

- This book have several chapters dealing with IDA Pro and RCE
- IDA does not work against the binary direct, instead an IDB file is used
  - Database with each byte flagged
  - May not be a full PE image
- FLAIR (Fast Library Acquisition for Identification and Recognition)
  - As FLIRT but create your own signatures for library functions
  - http://users.du.se/~hjo/cs/common/books/The%20IDA%20Pro%20Book/idaPro_ch12.pdf
- Write your own plugins (SDK) and scripts
- Lots of other functions in IDA Pro



"A fantastic book for anyone looking to learn the tools and techniques needed to break in and stay in." —Bruce Potter, Founder, The Shmoo Group

Second Edition

GRAY HAT HACKING

The Ethical Hacker's Handbook

Shon Harris Allen Harper Chris Eagle Jonathan Ness

- IDA Pro 5.x graph based interface
- IDB to exe instructions in lab

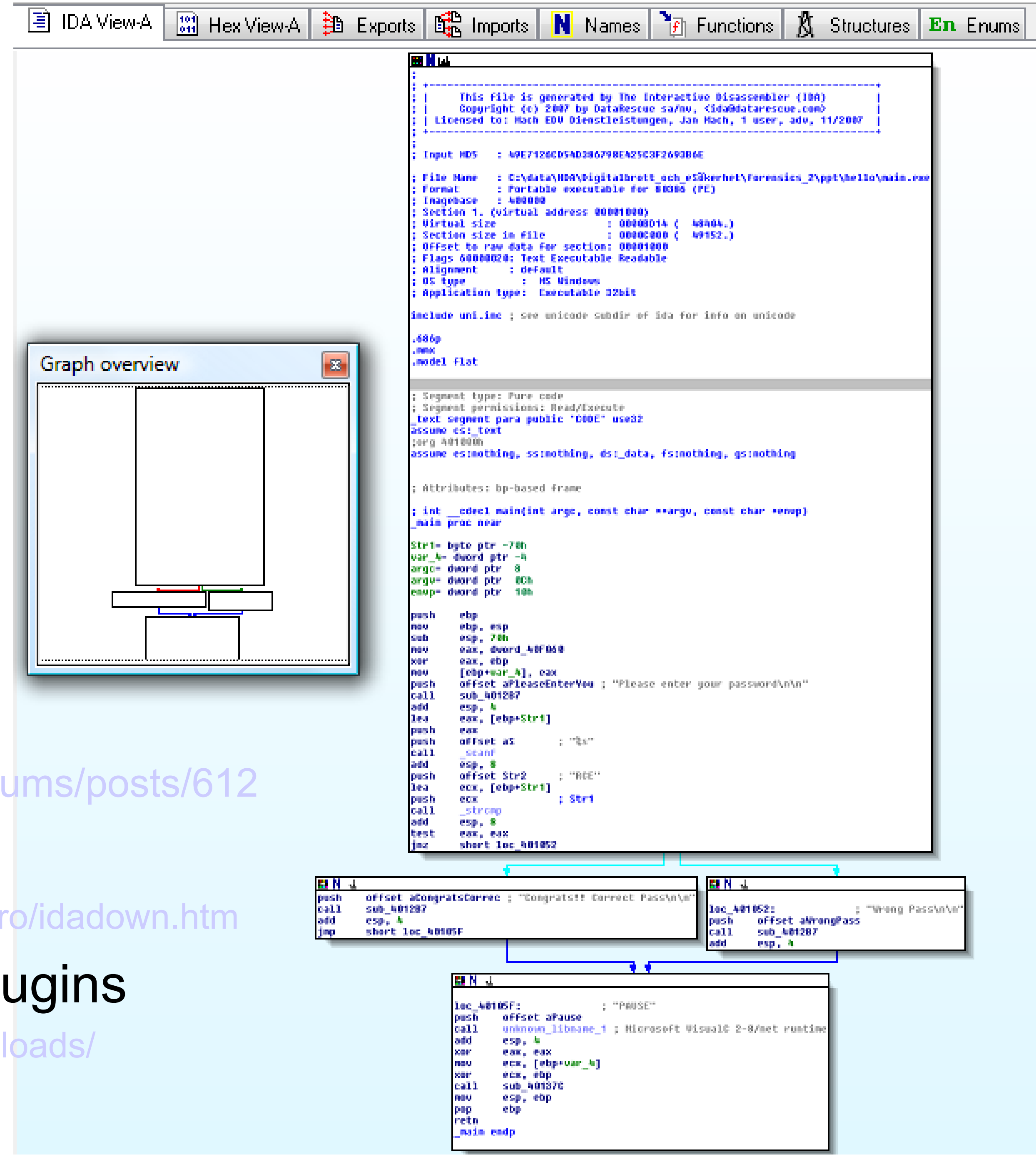
http://www.openrce.org/forums/posts/612

- PE utilities

http://www.hex-rays.com/idapro/idadown.htm

- IDA scripts and plugins

http://www.openrce.org/downloads/



```
;
; +-------------------------------------------------------------------------+
; |      This file is generated by The Interactive Disassembler (IDA)      |
; |      Copyright (c) 2007 by DataRescue sa/nv, <ida@datarescue.com>      |
; | Licensed to: Mach EDV Dienstleistungen, Jan Mach, 1 user, adv, 11/2007 |
; +-------------------------------------------------------------------------+
;
; Input MD5    : 49E7126CD54D386798EA25C3F2693B6E

; File Name    : C:\data\HDA\Digitalbrott_och_eSäkerhet\Forensics_2\ppt\hello\main.exe
; Format       : Portable executable for 80386 (PE)
; Imagebase    : 400000
; Section 1. (virtual address 00001000)
; Virtual size                  : 0000BD14 (  48404.)
; Section size in file          : 0000C000 (  49152.)
; Offset to raw data for section: 00001000
; Flags 60000020: Text Executable Readable
; Alignment     : default
; OS type         :  MS Windows
; Application type:  Executable 32bit

include uni.inc ; see unicode subdir of ida for info on unicode

.686p
.mmx
.model flat

; Segment type: Pure code
; Segment permissions: Read/Execute
_text segment para public 'CODE' use32
assume cs:_text
;org 401000h
assume es:nothing, ss:nothing, ds:_data, fs:nothing, gs:nothing


; Attributes: bp-based frame

; int __cdecl main(int argc, const char **argv, const char *envp)
_main proc near

Str1= byte ptr -70h
var_4= dword ptr -4
argc= dword ptr  8
argv= dword ptr  0Ch
envp= dword ptr  10h

push    ebp
mov     ebp, esp
sub     esp, 70h
mov     eax, dword_40F060
xor     eax, ebp
mov     [ebp+var_4], eax
push    offset aPleaseEnterYou ; "Please enter your password\n\n"
call    sub_401287
add     esp, 4
lea     eax, [ebp+Str1]
push    eax
push    offset aS       ; "%s"
call    _scanf
add     esp, 8
push    offset Str2     ; "RCE"
lea     ecx, [ebp+Str1]
push    ecx             ; Str1
call    _strcmp
add     esp, 8
test    eax, eax
jnz     short loc_401052
```

```
push    offset aCongratsCorrec ; "Congrats!! Correct Pass\n\n"
call    sub_401287
add     esp, 4
jmp     short loc_40105F
```

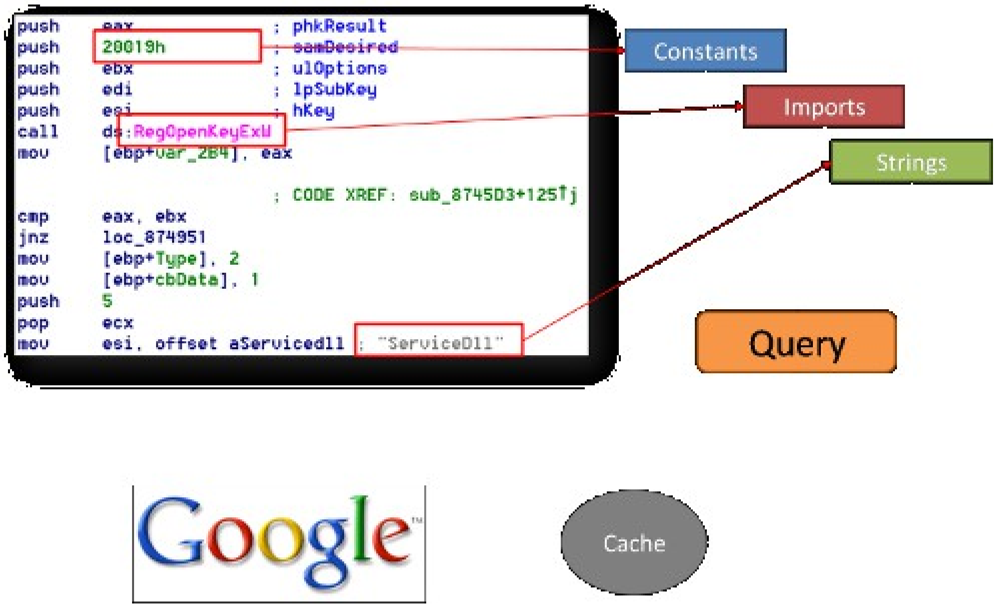
```
loc_401052:             ; "Wrong Pass\n\n"
push    offset aWrongPass
call    sub_401287
add     esp, 4
```

```
loc_40105F:             ; "PAUSE"
push    offset aPause
call    unknown_libname_1 ; Microsoft VisualC 2-8/net runtime
add     esp, 4
xor     eax, eax
mov     ecx, [ebp+var_4]
xor     ecx, ebp
call    sub_40137C
mov     esp, ebp
pop     ebp
retn
_main endp
```

RE-Google

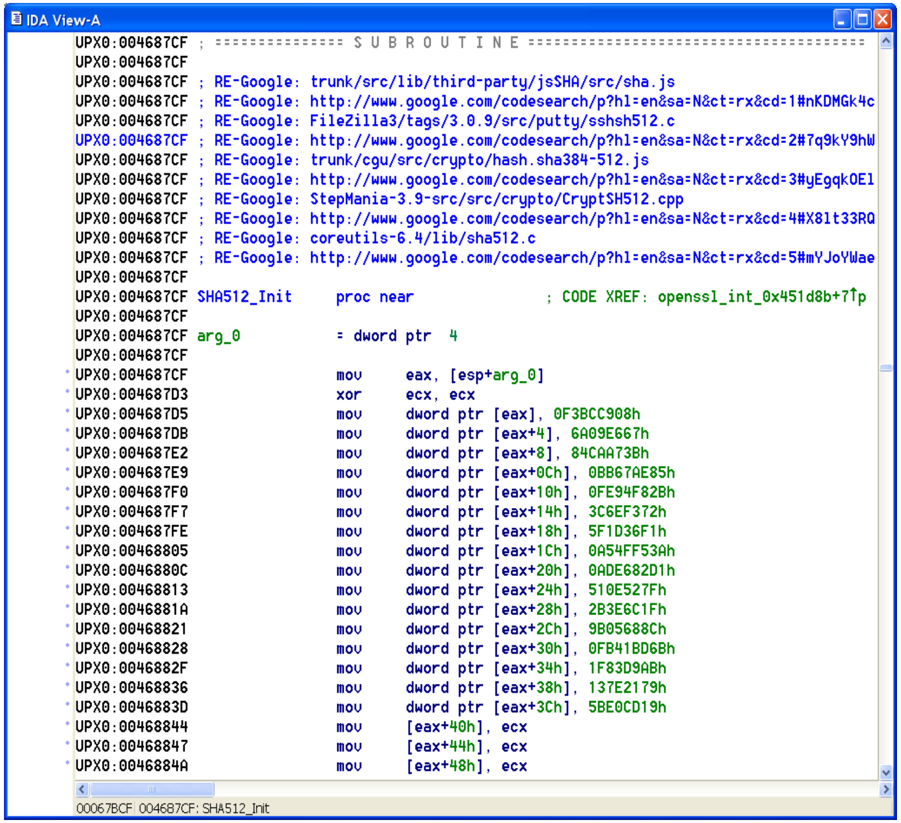
http://regoogle.carnivore.it/

- RE-Google
  - A plugin for the IDA Pro that queries Google Code for information about the functions etc. contained in a disassembled binary
  - High probability that Google finds the code parts ”in the wild”



```
push    eax                ; phkResult
push    20019h             ; samDesired
push    ebx                ; ulOptions
push    edi                ; lpSubKey
push    esi                ; hKey
call    ds:RegOpenKeyExW
mov     [ebp+var_2B4], eax

                           ; CODE XREF: sub_8745D3+125↑j
cmp     eax, ebx
jnz     loc_874951
mov     [ebp+Type], 2
mov     [ebp+cbData], 1
push    5
pop     ecx
mov     esi, offset aServicedll ; "ServiceDll"
```

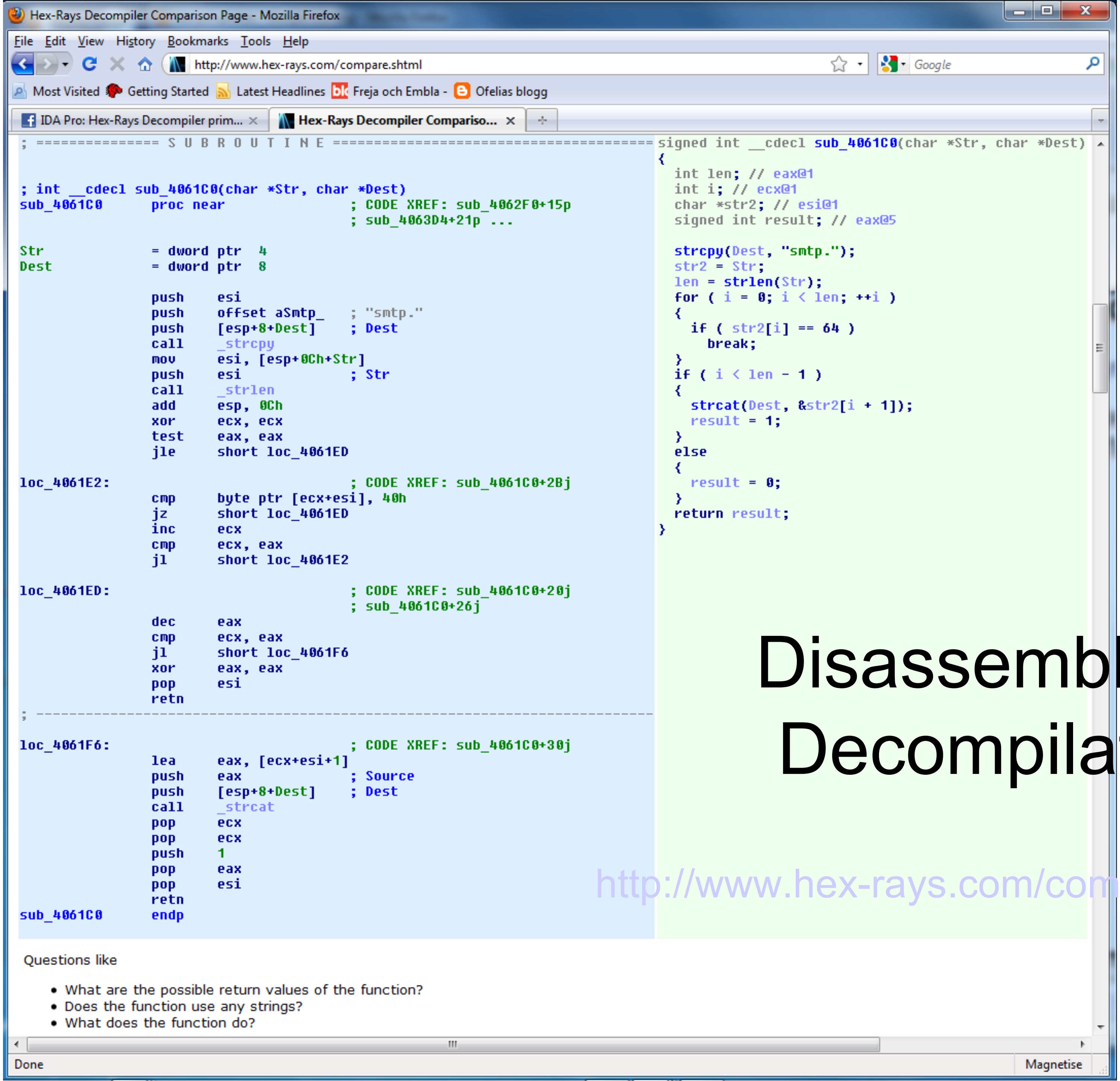
Uses the IDAPython plugin
http://d-dome.net/idapython



```
UPX0:004687CF ; =============== S U B R O U T I N E =======================================
UPX0:004687CF
UPX0:004687CF ; RE-Google: trunk/src/lib/third-party/jsSHA/src/sha.js
UPX0:004687CF ; RE-Google: http://www.google.com/codesearch/p?hl=en&sa=N&ct=rx&cd=1#nKDMGk4c
UPX0:004687CF ; RE-Google: FileZilla3/tags/3.0.9/src/putty/sshsh512.c
UPX0:004687CF ; RE-Google: http://www.google.com/codesearch/p?hl=en&sa=N&ct=rx&cd=2#7q9kY9hW
UPX0:004687CF ; RE-Google: trunk/cgu/src/crypto/hash.sha384-512.js
UPX0:004687CF ; RE-Google: http://www.google.com/codesearch/p?hl=en&sa=N&ct=rx&cd=3#yEgqkOEl
UPX0:004687CF ; RE-Google: StepMania-3.9-src/src/crypto/CryptSH512.cpp
UPX0:004687CF ; RE-Google: http://www.google.com/codesearch/p?hl=en&sa=N&ct=rx&cd=4#X8lt33RQ
UPX0:004687CF ; RE-Google: coreutils-6.4/lib/sha512.c
UPX0:004687CF ; RE-Google: http://www.google.com/codesearch/p?hl=en&sa=N&ct=rx&cd=5#mYJoYWae
UPX0:004687CF
UPX0:004687CF SHA512_Init     proc near               ; CODE XREF: openssl_int_0x451d8b+7↑p
UPX0:004687CF
UPX0:004687CF arg_0           = dword ptr  4
UPX0:004687CF
UPX0:004687CF                 mov     eax, [esp+arg_0]
UPX0:004687D3                 xor     ecx, ecx
UPX0:004687D5                 mov     dword ptr [eax], 0F3BCC908h
UPX0:004687DB                 mov     dword ptr [eax+4], 6A09E667h
UPX0:004687E2                 mov     dword ptr [eax+8], 84CAA73Bh
UPX0:004687E9                 mov     dword ptr [eax+0Ch], 0BB67AE85h
UPX0:004687F0                 mov     dword ptr [eax+10h], 0FE94F82Bh
UPX0:004687F7                 mov     dword ptr [eax+14h], 3C6EF372h
UPX0:004687FE                 mov     dword ptr [eax+18h], 5F1D36F1h
UPX0:00468805                 mov     dword ptr [eax+1Ch], 0A54FF53Ah
UPX0:0046880C                 mov     dword ptr [eax+20h], 0ADE682D1h
UPX0:00468813                 mov     dword ptr [eax+24h], 510E527Fh
UPX0:0046881A                 mov     dword ptr [eax+28h], 2B3E6C1Fh
UPX0:00468821                 mov     dword ptr [eax+2Ch], 9B05688Ch
UPX0:00468828                 mov     dword ptr [eax+30h], 0FB41BD6Bh
UPX0:0046882F                 mov     dword ptr [eax+34h], 1F83D9ABh
UPX0:00468836                 mov     dword ptr [eax+38h], 137E2179h
UPX0:0046883D                 mov     dword ptr [eax+3Ch], 5BE0CD19h
UPX0:00468844                 mov     [eax+40h], ecx
UPX0:00468847                 mov     [eax+44h], ecx
UPX0:0046884A                 mov     [eax+48h], ecx
```

00067BCF 004687CF: SHA512_Init

Disassembly vs. Decompilation


Hex-Rays Decompiler Comparison Page - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.hex-rays.com/compare.shtml
Google
Most Visited
Getting Started
Latest Headlines
Freja och Embla -
Ofelias blogg
IDA Pro: Hex-Rays Decompiler prim...
Hex-Rays Decompiler Compariso...
Done
Magnetise

```
; =============== S U B R O U T I N E =======================================


; int __cdecl sub_4061C0(char *Str, char *Dest)
sub_4061C0      proc near               ; CODE XREF: sub_4062F0+15p
                                        ; sub_4063D4+21p ...

Str             = dword ptr  4
Dest            = dword ptr  8

                push    esi
                push    offset aSmtp_   ; "smtp."
                push    [esp+8+Dest]    ; Dest
                call    _strcpy
                mov     esi, [esp+0Ch+Str]
                push    esi             ; Str
                call    _strlen
                add     esp, 0Ch
                xor     ecx, ecx
                test    eax, eax
                jle     short loc_4061ED

loc_4061E2:                             ; CODE XREF: sub_4061C0+2Bj
                cmp     byte ptr [ecx+esi], 40h
                jz      short loc_4061ED
                inc     ecx
                cmp     ecx, eax
                jl      short loc_4061E2

loc_4061ED:                             ; CODE XREF: sub_4061C0+20j
                                        ; sub_4061C0+26j
                dec     eax
                cmp     ecx, eax
                jl      short loc_4061F6
                xor     eax, eax
                pop     esi
                retn
; ---------------------------------------------------------------------------

loc_4061F6:                             ; CODE XREF: sub_4061C0+30j
                lea     eax, [ecx+esi+1]
                push    eax             ; Source
                push    [esp+8+Dest]    ; Dest
                call    _strcat
                pop     ecx
                pop     ecx
                push    1
                pop     eax
                pop     esi
                retn
sub_4061C0      endp
```

```
signed int __cdecl sub_4061C0(char *Str, char *Dest)
{
  int len; // eax@1
  int i; // ecx@1
  char *str2; // esi@1
  signed int result; // eax@5

  strcpy(Dest, "smtp.");
  str2 = Str;
  len = strlen(Str);
  for ( i = 0; i < len; ++i )
  {
    if ( str2[i] == 64 )
      break;
  }
  if ( i < len - 1 )
  {
    strcat(Dest, &str2[i + 1]);
    result = 1;
  }
  else
  {
    result = 0;
  }
  return result;
}
```

Questions like

- What are the possible return values of the function?
- Does the function use any strings?
- What does the function do?

http://www.hex-rays.com/compare.shtml

Decompilation with IDA Pro - for free!

- Desquirr IDA Pro plugin
- http://www.iitac.org/2008/01/decompilation-with-ida-pro-for-free/
- Thesis
  - Designing an object-oriented decompiler : Decompilation support for Interactive Disassembler Pro, David Eriksson
- Plugin and scripts installation
  - \IDA\Plugins\ and \IDA\idc directory
  - Edit plugins.cfg if neccessary
- Exerpt from translation
  - Before comp. after decomp.

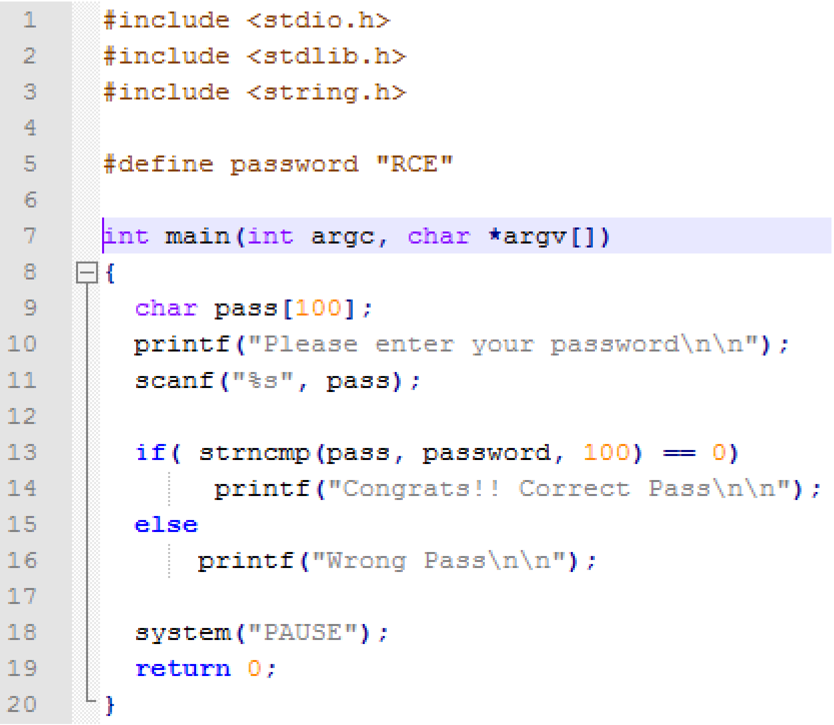
Patched v4.9 SDK included in lab

```
{
  if (x >= 2)
    return (fib(x - 1) + fib(x - 2));
  else
    return (1);
}
```

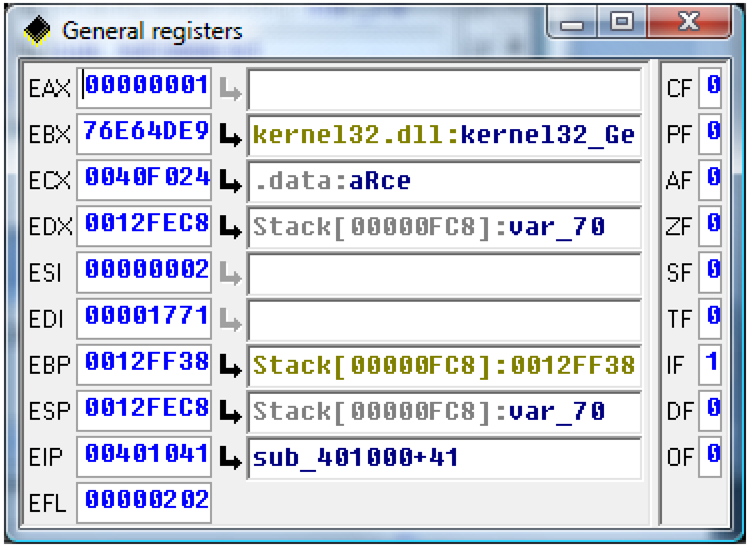
```
{
  if (arg 0 >= 2) goto loc 10313;
  dx = sub 102EB(arg 0 - 1);
  ax = sub 102EB(arg 0 + 0xfffe);
  ax = dx + ax;
  goto loc 10318;
  goto loc 10318;
  loc 10313:
  ax = 1;
  goto loc 10318;
  loc 10318:
  return ax;
}
```

Demo!

- Looking at the program ->
- How to enforce access?
- At least two ways
  - Find password
  - Break thru password protection
- Strings/hexeditor
- Examine and debug using IDA Pro free v5.0 (and/or OllyDbg)
  - Change the ASM code (binary)
  - Change CPU registers
    - ASM instruction ”test”, performs a non destructive logic AND on operands
    - If strcmp result == 0 : ZF = 1
    - If result is anything else : ZF = 0



```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define password "RCE"

int main(int argc, char *argv[])
{
  char pass[100];
  printf("Please enter your password\n\n");
  scanf("%s", pass);

  if( strncmp(pass, password, 100) == 0)
      printf("Congrats!! Correct Pass\n\n");
  else
      printf("Wrong Pass\n\n");

  system("PAUSE");
  return 0;
}
```



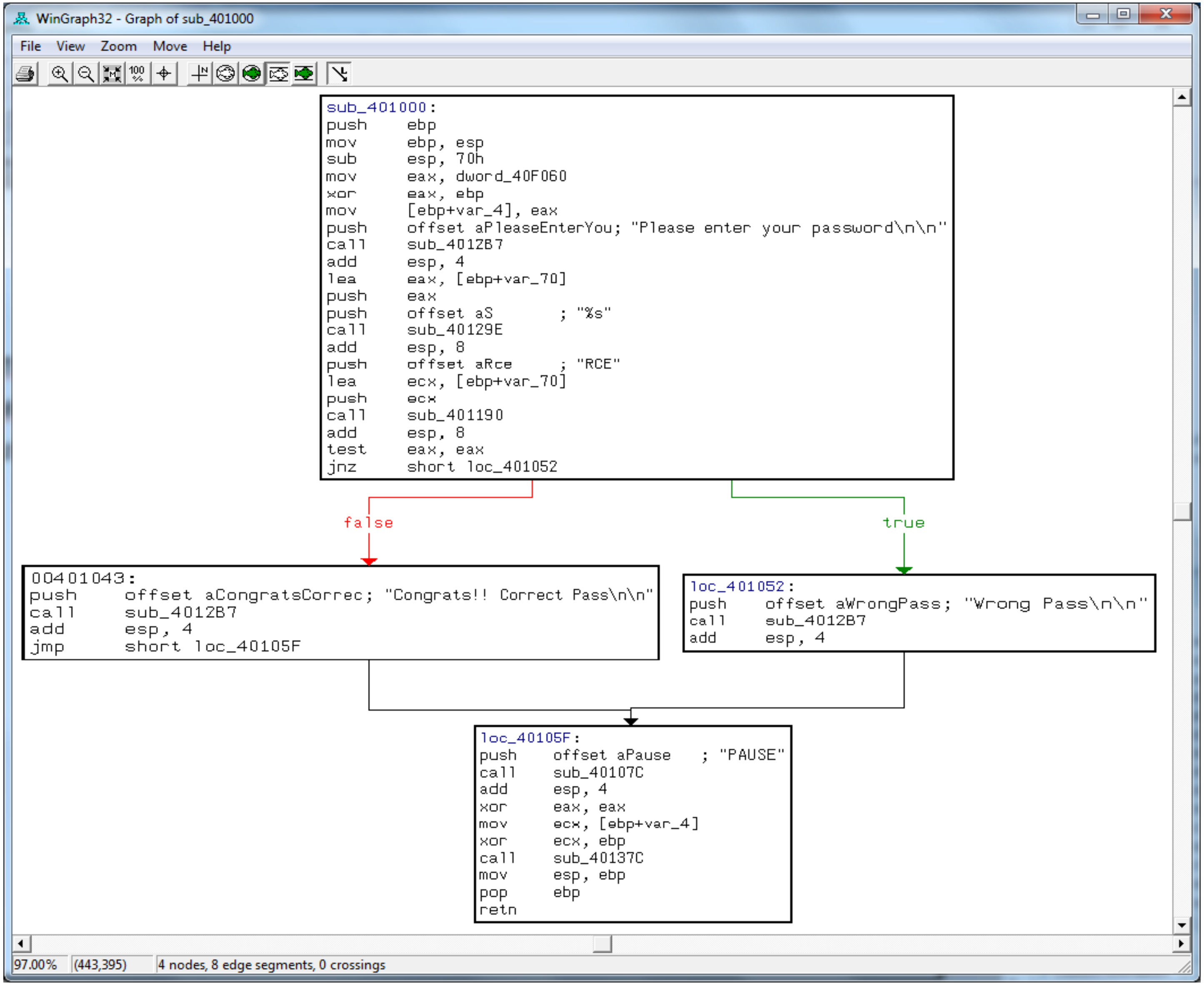
General registers

```
EAX 00000001                              CF 0
EBX 76E64DE9 kernel32.dll:kernel32_Ge     PF 0
ECX 0040F024 .data:aRce                   AF 0
EDX 0012FEC8 Stack[00000FC8]:var_70       ZF 0
ESI 00000002                              SF 0
EDI 00001771                              TF 0
EBP 0012FF38 Stack[00000FC8]:0012FF38     IF 1
ESP 0012FEC8 Stack[00000FC8]:var_70       DF 0
EIP 00401041 sub_401000+41                OF 0
EFL 00000202
```

IDA Pro free – graphs

- F12
- Solution flow chart

- Ctrl-F12
- Graph of function calls
- Can be very large and hard to view!


WinGraph32 - Graph of sub_401000
File View Zoom Move Help
false
true
97.00%
(443,395)
4 nodes, 8 edge segments, 0 crossings

```
sub_401000:
push    ebp
mov     ebp, esp
sub     esp, 70h
mov     eax, dword_40F060
xor     eax, ebp
mov     [ebp+var_4], eax
push    offset aPleaseEnterYou; "Please enter your password\n\n"
call    sub_4012B7
add     esp, 4
lea     eax, [ebp+var_70]
push    eax
push    offset aS       ; "%s"
call    sub_40129E
add     esp, 8
push    offset aRce     ; "RCE"
lea     ecx, [ebp+var_70]
push    ecx
call    sub_401190
add     esp, 8
test    eax, eax
jnz     short loc_401052
```

```
00401043:
push    offset aCongratsCorrec; "Congrats!! Correct Pass\n\n"
call    sub_4012B7
add     esp, 4
jmp     short loc_40105F
```

```
loc_401052:
push    offset aWrongPass; "Wrong Pass\n\n"
call    sub_4012B7
add     esp, 4
```

```
loc_40105F:
push    offset aPause   ; "PAUSE"
call    sub_40107C
add     esp, 4
xor     eax, eax
mov     ecx, [ebp+var_4]
xor     ecx, ebp
call    sub_40137C
mov     esp, ebp
pop     ebp
retn
```

IDA Pro decompiled code

- Main function
- Hex-Rays decompiler
- sub_4012B7
  - printf
- char Str1
  - entered password
- unknown_libname_1
  - system

The password program

```
int __cdecl main(int argc, const char **argv, const char *envp)
{
  char ST08_1_0; // ST08_1@0
  char v4; // ST08_1@1
  int  s; // [sp+70h] [bp+0h]@1
  unsigned int v7; // [sp+6Ch] [bp-4h]@1
  char Str1; // [sp+0h] [bp-70h]@1

  v7 = (unsigned int)& s ^ dword_40F060;
  sub_4012B7((int)"Please enter your password\n\n", ST08_1_0);
  scanf("%s", &Str1);
  if ( strcmp(&Str1, "RCE") )
    sub_4012B7((int)"Wrong Pass\n\n", v4);
  else
    sub_4012B7((int)"Congrats!! Correct Pass\n\n", v4);
  unknown_libname_1("PAUSE");
  return 0;
}
```

Restrictions on IDA free

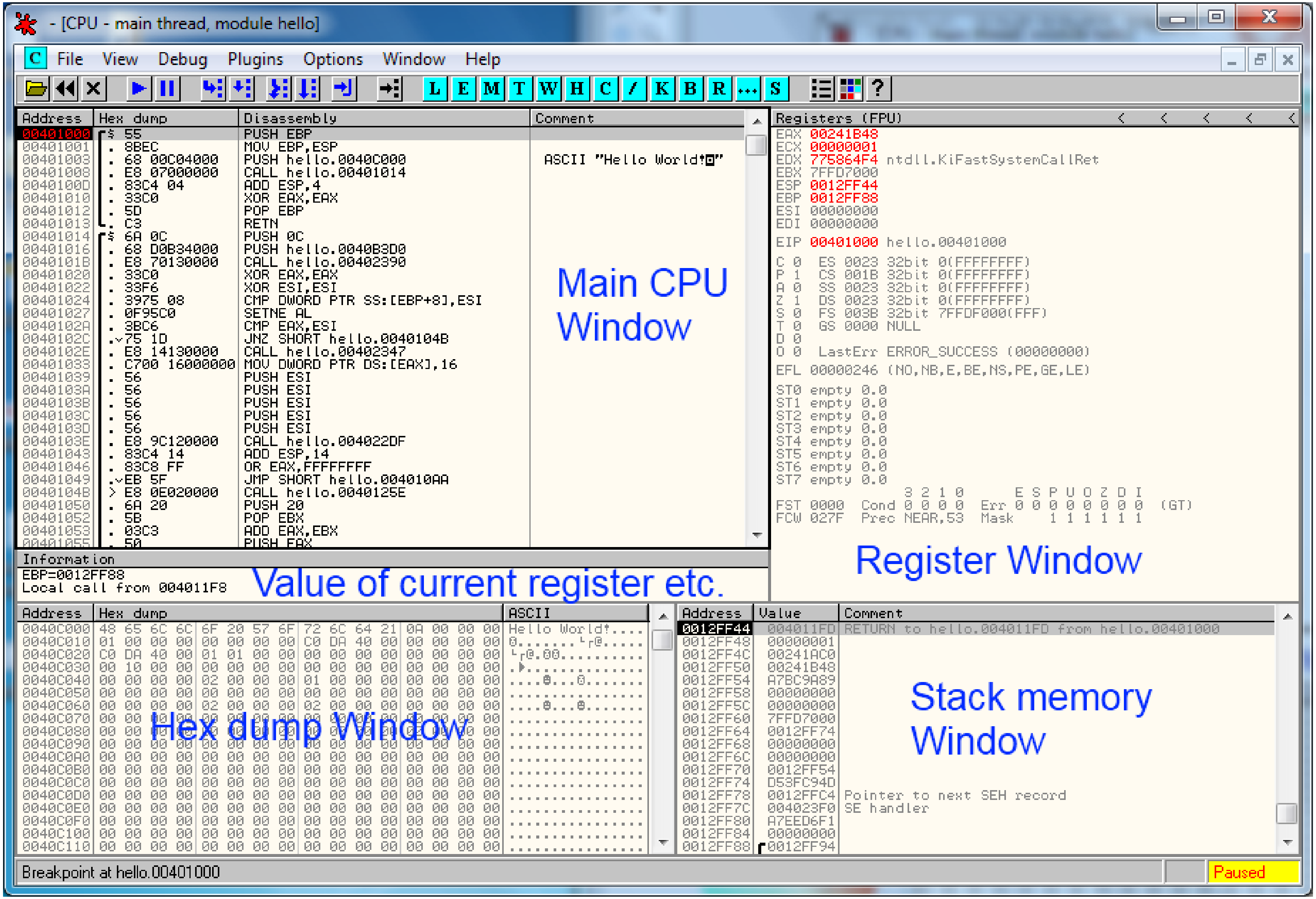
If you wish to use the freeware version of IDA, you must abide by (and, perhaps, put up with) the following restrictions and reduced functionality:

- The freeware version is for non-commercial use only and only as a Windows GUI
- The freeware version lacks all features introduced in later versions of IDA, including all SDK and scripting features that were introduced in versions 5.0 and later
- The freeware version ships with substantially fewer plug-ins and IDC scripts than the commercial versions
- The freeware version can disassemble only x86 code (it has only one processor module)
- The freeware version ships with only seven loader modules that cover common x86 file types, including PE, ELF, MS-DOS, COFF and a.out. Loading files in binary format is also supported
- The freeware version includes only a few type libraries common to x86 binaries, including those for GNU, Microsoft, and Borland
- Add-ons such as the FLAIR tools and the SDK are not included
- Debugging is allowed only for local Windows processes/binaries. No remote debugging capability is available

Source: The IDA Pro Book

OllyDbg, quick tutorial

OllyDbg
Win32 Symbolic Debugger


- [CPU - main thread, module hello]
C File View Debug Plugins Options Window Help
L E M T W H C / K B R ... S
?

```
Address   Hex dump         Disassembly                        Comment
00401000 ┌$ 55              PUSH EBP
00401001 │. 8BEC            MOV EBP,ESP
00401003 │. 68 00C04000     PUSH hello.0040C000                  ASCII "Hello World!◙"
00401008 │. E8 07000000     CALL hello.00401014
0040100D │. 83C4 04         ADD ESP,4
00401010 │. 33C0            XOR EAX,EAX
00401012 │. 5D              POP EBP
00401013 └. C3              RETN
00401014 ┌$ 6A 0C           PUSH 0C
00401016 │. 68 D0B34000     PUSH hello.0040B3D0
0040101B │. E8 70130000     CALL hello.00402390
00401020 │. 33C0            XOR EAX,EAX
00401022 │. 33F6            XOR ESI,ESI
00401024 │. 3975 08         CMP DWORD PTR SS:[EBP+8],ESI
00401027 │. 0F95C0          SETNE AL
0040102A │. 3BC6            CMP EAX,ESI
0040102C │.v75 1D           JNZ SHORT hello.0040104B
0040102E │. E8 14130000     CALL hello.00402347
00401033 │. C700 16000000   MOV DWORD PTR DS:[EAX],16
00401039 │. 56              PUSH ESI
0040103A │. 56              PUSH ESI
0040103B │. 56              PUSH ESI
0040103C │. 56              PUSH ESI
0040103D │. 56              PUSH ESI
0040103E │. E8 9C120000     CALL hello.004022DF
00401043 │. 83C4 14         ADD ESP,14
00401046 │. 83C8 FF         OR EAX,FFFFFFFF
00401049 │.vEB 5F           JMP SHORT hello.004010AA
0040104B │> E8 0E020000     CALL hello.0040125E
00401050 │. 6A 20           PUSH 20
00401052 │. 5B              POP EBX
00401053 │. 03C3            ADD EAX,EBX
00401055 │. 50              PUSH EAX
```

Main CPU Window

```
Registers (FPU)
EAX 00241B48
ECX 00000001
EDX 775864F4 ntdll.KiFastSystemCallRet
EBX 7FFD7000
ESP 0012FF44
EBP 0012FF88
ESI 00000000
EDI 00000000

EIP 00401000 hello.00401000

C 0  ES 0023 32bit 0(FFFFFFFF)
P 1  CS 001B 32bit 0(FFFFFFFF)
A 0  SS 0023 32bit 0(FFFFFFFF)
Z 1  DS 0023 32bit 0(FFFFFFFF)
S 0  FS 003B 32bit 7FFDF000(FFF)
T 0  GS 0000 NULL
D 0
O 0  LastErr ERROR_SUCCESS (00000000)

EFL 00000246 (NO,NB,E,BE,NS,PE,GE,LE)

ST0 empty 0.0
ST1 empty 0.0
ST2 empty 0.0
ST3 empty 0.0
ST4 empty 0.0
ST5 empty 0.0
ST6 empty 0.0
ST7 empty 0.0
               3 2 1 0      E S P U O Z D I
FST 0000  Cond 0 0 0 0  Err 0 0 0 0 0 0 0 0  (GT)
FCW 027F  Prec NEAR,53  Mask    1 1 1 1 1 1
```

Register Window

```
Information
EBP=0012FF88
Local call from 004011F8
```

Value of current register etc.

```
Address   Hex dump                                         ASCII
0040C000  48 65 6C 6C 6F 20 57 6F 72 6C 64 21 0A 00 00 00  Hello World!....
0040C010  01 00 00 00 00 00 00 00 C0 DA 40 00 00 00 00 00  ☺.......└┌@.....
0040C020  C0 DA 40 00 01 01 00 00 00 00 00 00 00 00 00 00  └┌@.☺☺..........
0040C030  00 10 00 00 00 00 00 00 00 00 00 00 00 00 00 00  .►..............
0040C040  00 00 00 00 02 00 00 00 01 00 00 00 00 00 00 00  ....☻...☺.......
0040C050  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C060  00 00 00 00 02 00 00 00 02 00 00 00 00 00 00 00  ....☻...☻.......
0040C070  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C080  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C090  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0A0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0B0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0C0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0D0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0E0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C0F0  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C100  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0040C110  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Hex dump Window

```
Address   Value      Comment
0012FF44  004011FD   RETURN to hello.004011FD from hello.00401000
0012FF48  00000001
0012FF4C  00241AC0
0012FF50  00241B48
0012FF54  A7BC9A89
0012FF58  00000000
0012FF5C  00000000
0012FF60  7FFD7000
0012FF64  0012FF74
0012FF68  00000000
0012FF6C  00000000
0012FF70  0012FF54
0012FF74  D53FC94D
0012FF78  0012FFC4   Pointer to next SEH record
0012FF7C  004023F0   SE handler
0012FF80  A7EED6F1
0012FF84  00000000
0012FF88 ┌0012FF94
```

Stack memory Window

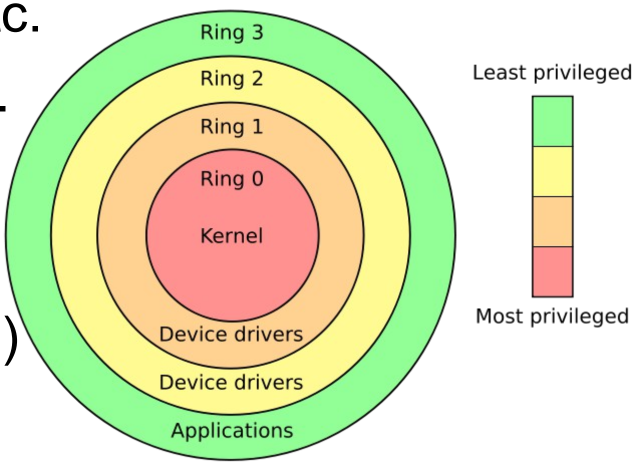
```
Breakpoint at hello.00401000                                    Paused
```

OllyDbg, quick tutorial

- Drag-and-drop a file or click File->Open (F3), select a file, wait for the analysis to finish, and then, well, watch and analyze the code
- Right after the file has been loaded, the pointer is at the first instruction to be executed in the program (OEP). The program is stopped right now.
- F9 runs the program from the current position
- Use F7 and F8 to step instruction by instruction, F7 enters inside any function calls, while F8 steps “over” them – (highly recommended for WINAPI calls)
- F2 puts a breakpoint (the program will stop if it reaches a breakpoint)
  - It can be resumed with F9, or you can press F7 and F8 to step further on
- With Shift+F7/F8/F9 one can pass exceptions to debugged program instead of the debugger taking care of it
- Ctrl+A re-analyzes the code
- Right-click in the windows and on objects in the windows to get pop-up menus enabling you to do a multitude of different things
- OllyDbg saves data in .udd files (v2.x can save analysis data as well)
- Internet is full of OllyDbg tutorials...

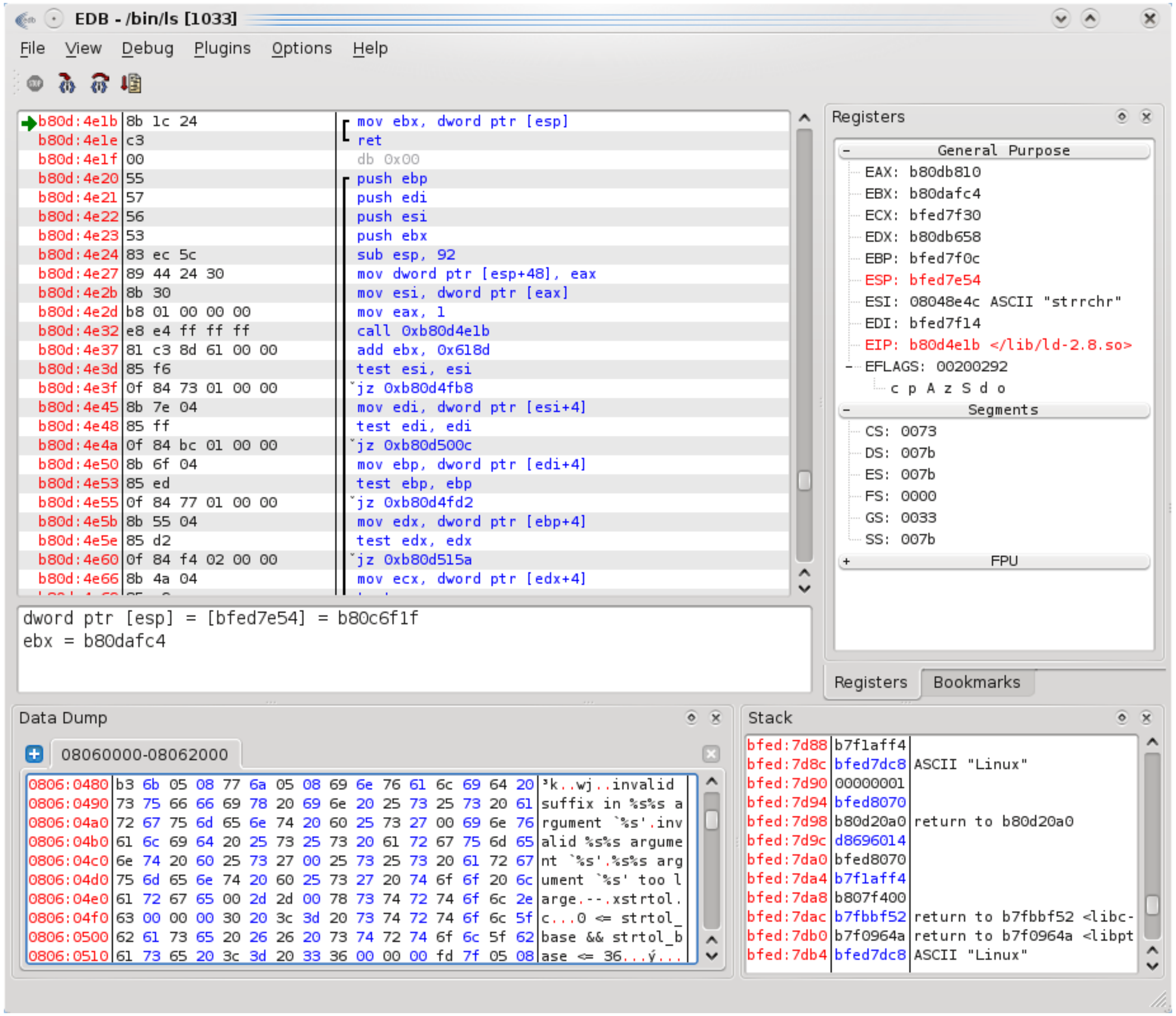
OllyDbg 2.x and other debuggers

- Plugins is not supported yet in OllyDbg 2.x branch, development is slow, x64 version in the works
- Immunity Debugger (free based on OllyDbg 1.x) with Python API, function graphing, heap analysis and more
  - Focused more on the security industry (exploits)
  - http://www.immunityinc.com/products-immdbg.shtml
- Ring 0 (kernel space) vs. Ring 3 (user space) debuggers
  - Debug drivers, kernel, executive services etc.
  - Syser, MS WinDbg (user space as well) etc.
- The Windows OS runs processes in one of two modes
  - User Mode (ring 3) and Kernel Mode (ring 0)



EDB (Evan's Debugger)

- http://www.codef00.com/projects
- OllyDbg in Linux?
- Developers goal is to have features on par with OllyDbg



```
b80d:4e1b 8b 1c 24            mov ebx, dword ptr [esp]
b80d:4e1e c3                  ret
b80d:4e1f 00                  db 0x00
b80d:4e20 55                  push ebp
b80d:4e21 57                  push edi
b80d:4e22 56                  push esi
b80d:4e23 53                  push ebx
b80d:4e24 83 ec 5c            sub esp, 92
b80d:4e27 89 44 24 30         mov dword ptr [esp+48], eax
b80d:4e2b 8b 30               mov esi, dword ptr [eax]
b80d:4e2d b8 01 00 00 00      mov eax, 1
b80d:4e32 e8 e4 ff ff ff      call 0xb80d4e1b
b80d:4e37 81 c3 8d 61 00 00   add ebx, 0x618d
b80d:4e3d 85 f6               test esi, esi
b80d:4e3f 0f 84 73 01 00 00   jz 0xb80d4fb8
b80d:4e45 8b 7e 04            mov edi, dword ptr [esi+4]
b80d:4e48 85 ff               test edi, edi
b80d:4e4a 0f 84 bc 01 00 00   jz 0xb80d500c
b80d:4e50 8b 6f 04            mov ebp, dword ptr [edi+4]
b80d:4e53 85 ed               test ebp, ebp
b80d:4e55 0f 84 77 01 00 00   jz 0xb80d4fd2
b80d:4e5b 8b 55 04            mov edx, dword ptr [ebp+4]
b80d:4e5e 85 d2               test edx, edx
b80d:4e60 0f 84 f4 02 00 00   jz 0xb80d515a
b80d:4e66 8b 4a 04            mov ecx, dword ptr [edx+4]
```

```
dword ptr [esp] = [bfed7e54] = b80c6f1f
ebx = b80dafc4
```

```
EAX: b80db810
EBX: b80dafc4
ECX: bfed7f30
EDX: b80db658
EBP: bfed7f0c
ESP: bfed7e54
ESI: 08048e4c ASCII "strrchr"
EDI: bfed7f14
EIP: b80d4e1b </lib/ld-2.8.so>
EFLAGS: 00200292
  c p A z S d o
CS: 0073
DS: 007b
ES: 007b
FS: 0000
GS: 0033
SS: 007b
```

```
0806:0480 b3 6b 05 08 77 6a 05 08 69 6e 76 61 6c 69 64 20 ³k..wj..invalid 
0806:0490 73 75 66 66 69 78 20 69 6e 20 25 73 25 73 20 61 suffix in %s%s a
0806:04a0 72 67 75 6d 65 6e 74 20 60 25 73 27 00 69 6e 76 rgument `%s'.inv
0806:04b0 61 6c 69 64 20 25 73 25 73 20 61 72 67 75 6d 65 alid %s%s argume
0806:04c0 6e 74 20 60 25 73 27 00 25 73 25 73 20 61 72 67 nt `%s'.%s%s arg
0806:04d0 75 6d 65 6e 74 20 60 25 73 27 20 74 6f 6f 20 6c ument `%s' too l
0806:04e0 61 72 67 65 00 2d 2d 00 78 73 74 72 74 6f 6c 2e arge.--.xstrtol.
0806:04f0 63 00 00 00 30 20 3c 3d 20 73 74 72 74 6f 6c 5f c...0 <= strtol_
0806:0500 62 61 73 65 20 26 26 20 73 74 72 74 6f 6c 5f 62 base && strtol_b
0806:0510 61 73 65 20 3c 3d 20 33 36 00 00 00 fd 7f 05 08 ase <= 36...ý...
```

```
bfed:7d88 b7f1aff4
bfed:7d8c bfed7dc8 ASCII "Linux"
bfed:7d90 00000001
bfed:7d94 bfed8070
bfed:7d98 b80d20a0 return to b80d20a0
bfed:7d9c d8696014
bfed:7da0 bfed8070
bfed:7da4 b7f1aff4
bfed:7da8 b807f400
bfed:7dac b7fbbf52 return to b7fbbf52 <libc-
bfed:7db0 b7f0964a return to b7f0964a <libpt
bfed:7db4 bfed7dc8 ASCII "Linux"
```

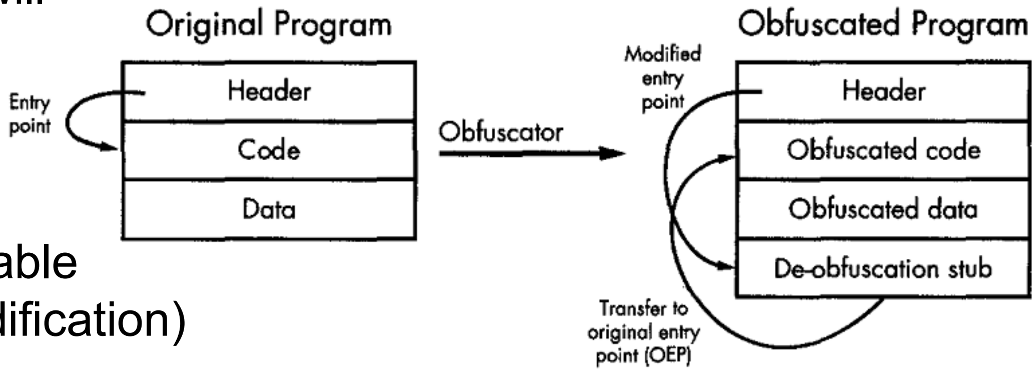
Obfuscated code analysis

- Even under ideal circumstances, comprehending a disassembly listing is a difficult task at best
- Over the last several years, an arms race of sorts has been taking place between reverse engineers and programmers who wish to keep their code secret
- Software Protection through Anti-Debugging
  - http://people.seas.harvard.edu/~mgagnon/software_protection_through_anti_debugging.pdf
- Honynet project challenges at: honeynet.org
  - Scan 32 - Analyze a Malware binary
    - http://old.honeynet.org/scans/scan32/sotm32.pdf
    - Heading 6 – Code analysis, good in detail description of unpacking/dumping malware and analysis of the binary with OllyDbg, OllyDump, ImpREC etc.
  - Scan 33 - Advanced reverse engineering challenge
  - The reverse challenge

Obfuscated code analysis

Anti-Static Analysis Techniques

- Disassembly Desynchronization
  - Prevent the disassembly from finding the correct starting address for one or more instructions. Forcing the disassembler to lose track of itself
- Dynamically Computed Target Addresses
  - Address to which execution will flow is computed at run-time
- Opcode Obfuscation
  - Encode or encrypt the actual instructions when the executable file is being created (self modification)



- Imported Function Obfuscation
  - In order to avoid leaking information about potential actions that a binary may perform, aimed at making it difficult for the static analysts to determine which shared libraries and library functions are used within an obfuscated binary
- Targeted Attacks on Analysis Tools

Obfuscated code analysis

Anti-Dynamic Analysis Techniques

- Detecting Virtualization
  - Detection of virtualization-specific software and hardware
  - Detection of virtual machine-specific behaviors
  - Detection of processor-specific behavioral changes (blue/red pill etc.)
- Detecting Instrumentation (Sysinternals tools, WireShark etc.)
  - Check loaded drivers, scan active process list or windows title texts etc.
- Detecting Debuggers
  - API functions such as the Windows IsDebuggerPresent(), NtQueryInformationProcess() or OutputDebugStringA()
  - Lower-level checks for memory or processor artifacts resulting from the use of a debugger
    - Detecting that a processor's trace (single step) Trap Flag (TF) is set.
  - SoftIce, a Windows kernel debugger, can be detected through the presence of the "\\.\NTICE" device (named pipe), which is used to communicate with the debugger

Obfuscated code analysis

Anti-Dynamic Analysis Techniques

- Preventing Debugging
  - Intentionally generating various exceptions when a SEH (Structured Exception Handler) is set
    - Attached debugger will catch the exception and the debug user must analyze why the exception occurred and decide if to or not to pass the exception along to the program being debugged
      - Standard software breakpoint such as opcode 0xCC (INT3)
      - Functions as CloseHandle(HANDLE invalid_handle)
  - Perplex the debugger by introducing spurious breakpoints, clearing hardware breakpoints, hindering selection of breakpoint addresses or preventing the debugger from attaching to a process
    - WriteProcessMemory() to remove/add INT3
    - Encoded programs makes placing your own breakpoints difficult
  - Calling GetTickCount() at regular intervals, detecting slow execution
  - Suspend thread - if the process is not a child of explorer.exe
- Many more exists as TLS callback, Hardware BPs, etc. see paper next slide

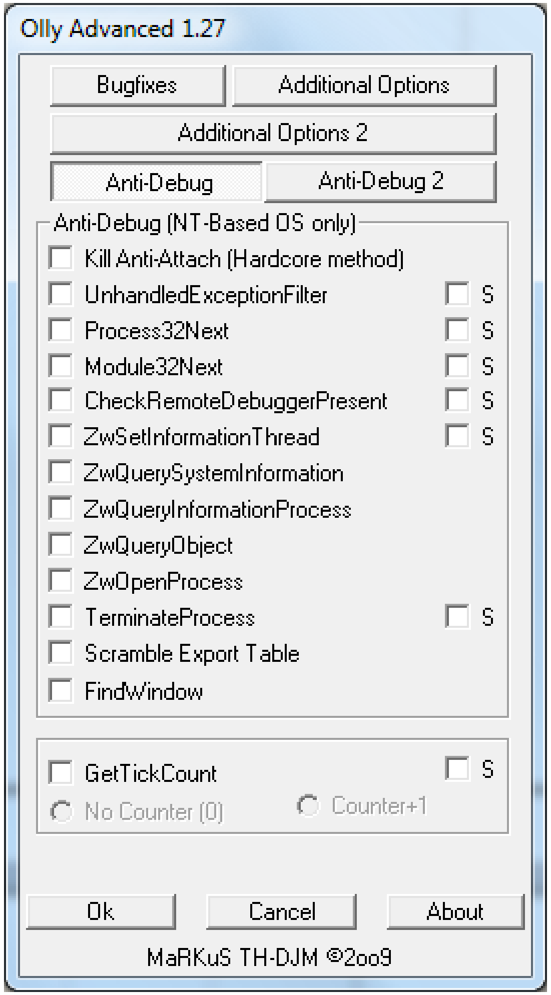
Obfuscated code analysis

Anti-Dumping/Anti-MUP

- Code Splicing or stolen bytes
  - Place code in blocks outside image (what PE loader have allocated)
- Debug-Blocker (Armadillo)
  - Creates two processes, the parent acts as a debugger and protects the child from other debuggers!
- Nanomites (Armadillo)
  - Extension of debug-blocker - parent have a special table (obfuscated) which child is fetching assembly codes from when INT3 occurs!
- CopyMem I and II
  - Code is not fully decrypted, only if a required memory page is needed
- Hide Import tables and Original Entry Point (OEP)
- Very good paper covering anti-everything - **included in lab**
  - Cracking,TheAnti
  - https://secure.um.edu.mt/__data/assets/pdf_file/0008/51767/wict08_submission_32.pdf

IDA and Olly debug stealth

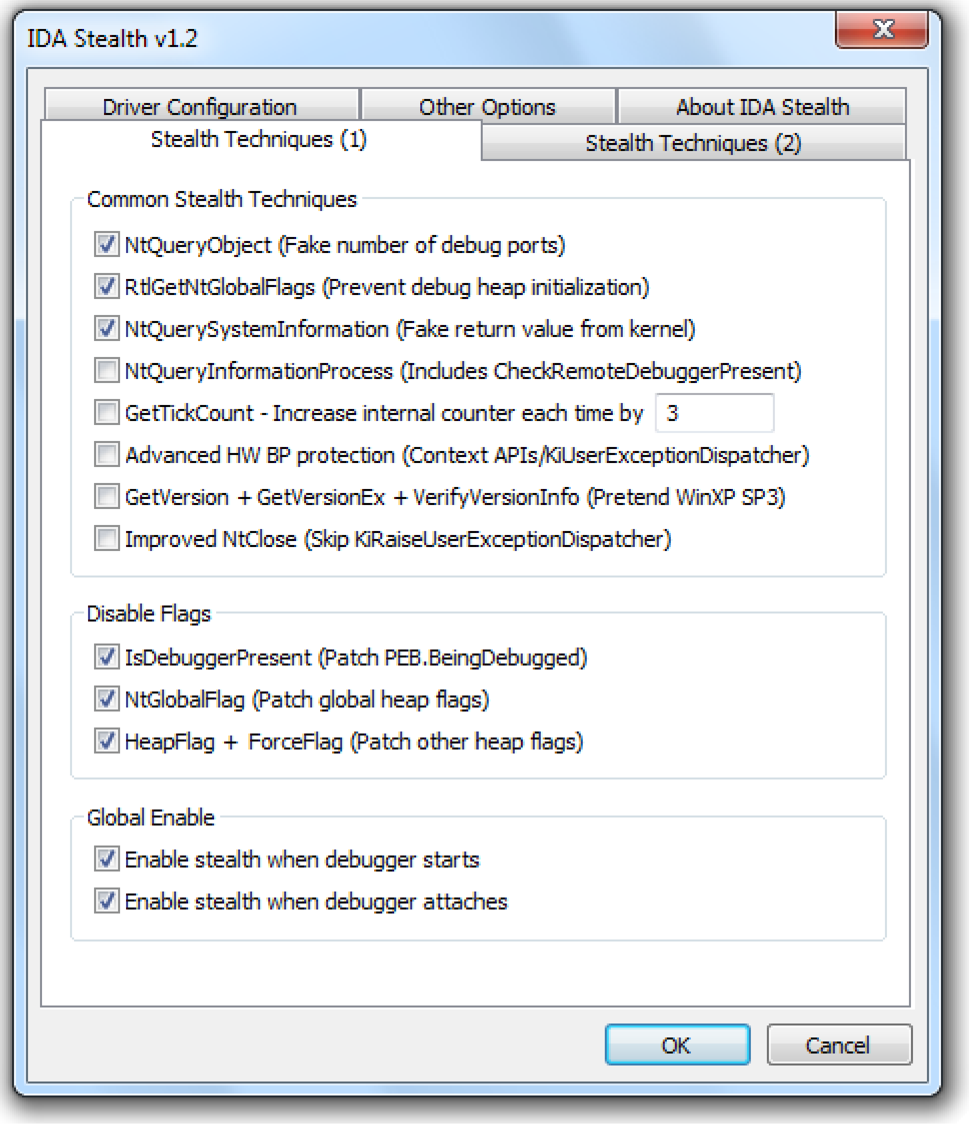
- Anti-anti-debug plugins



Anti-Debug (NT-Based OS only)

□ Kill Anti-Attach (Hardcore method)
□ UnhandledExceptionFilter □ S
□ Process32Next □ S
□ Module32Next □ S
□ CheckRemoteDebuggerPresent □ S
□ ZwSetInformationThread □ S
□ ZwQuerySystemInformation
□ ZwQueryInformationProcess
□ ZwQueryObject
□ ZwOpenProcess
□ TerminateProcess □ S
□ Scramble Export Table
□ FindWindow

□ GetTickCount □ S
○ No Counter (0) ○ Counter+1



Common Stealth Techniques

☑ NtQueryObject (Fake number of debug ports)
☑ RtlGetNtGlobalFlags (Prevent debug heap initialization)
☑ NtQuerySystemInformation (Fake return value from kernel)
□ NtQueryInformationProcess (Includes CheckRemoteDebuggerPresent)
□ GetTickCount - Increase internal counter each time by 3
□ Advanced HW BP protection (Context APIs/KiUserExceptionDispatcher)
□ GetVersion + GetVersionEx + VerifyVersionInfo (Pretend WinXP SP3)
□ Improved NtClose (Skip KiRaiseUserExceptionDispatcher)

Disable Flags

☑ IsDebuggerPresent (Patch PEB.BeingDebugged)
☑ NtGlobalFlag (Patch global heap flags)
☑ HeapFlag + ForceFlag (Patch other heap flags)

Global Enable

☑ Enable stealth when debugger starts
☑ Enable stealth when debugger attaches

Dynamic de-obfuscation of binaries

- If the import table is weird just showing KERNEL32.DLL and just a few imported functions from the DLL as LoadLibraryA and GetProcAddress it’s probably an indicator of obfuscation
- The following steps provide a basic and somewhat simplistic guide for dynamic de-obfuscation of binaries – MUP (Manually UnPack)
  1. Open an obfuscated program with a debugger (OllyDbg is a popular choice)
  2. Search for and set a breakpoint on the end of the de-obfuscation routine (hard!), what you search for is the OEP (Original Entry Point) before obfuscation
  3. Launch the program from the debugger, and wait for your breakpoint to trigger
  4. Utilize the debugger's memory-dumping features to capture the current state of the process to a file (OllyDump etc.)
  5. Terminate the process before it can do anything malicious. Note if IAT is going to be restored (for dynamic analysis) you must **leave the process on in paused mode!**
  6. Perform static analysis on the captured process image (and dynamic analysis)
- IDA Pro tutorials - Using IDA to deal with packed executables
  1. Difficult example via normal user interface
  2. Using the universal PE unpacker plug-in

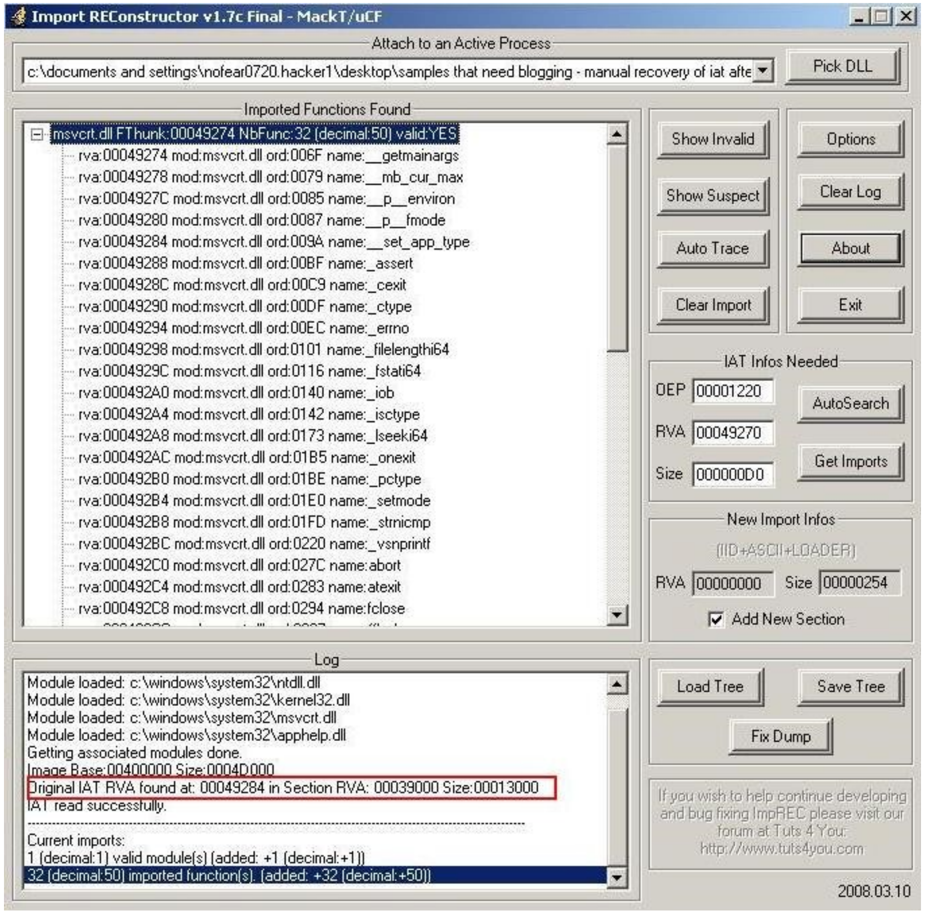
http://www.hex-rays.com/idapro/unpack_pe/index.htm

IDT/IAT restoration

- One of the trickiest parts of reconstructing a binary image from an obfuscated process is restoration of the program's imported function table
- The de-obfuscation process must also take care of linking the newly de-obfuscated process to all of the shared libraries and functions the process requires in order to execute properly
  - The only trace of this process is usually a table of imported function addresses somewhere within the process's memory image
- When dumping a de-obfuscated process image to a file, steps are often taken to attempt to reconstruct a valid import table in the dumped process image
  - In order to do this, the headers of the dumped image need to be modified to point to a **new import table structure** that must properly reflect all of the shared library dependencies of the original de-obfuscated program
- A popular tool to either automate this process or do it by hand is the ImpREC (Import REConstruction) utility
  - http://www.woodmann.com/collaborative/tools/index.php/ImpREC
- x9090's Blog
  - http://x9090.blogspot.com/2009/04/manual-iat-recovery-using-imprec.html

ImpREC (Import REConstructor) utility

- Attach to an Active Process, Correct OEP, AutoSearch, Get Imports
- Show Invalid, Double Click rva:..., Load/Save Tree, Fix Dump
- ImpRec adds a new section with correct IDT/IAT with name .mackt



Import REConstructor v1.7c Final - MackT/uCF

Attach to an Active Process: c:\documents and settings\nofear0720.hacker1\desktop\samples that need blogging - manual recovery of iat afte

Imported Functions Found

```
⊟ msvcrt.dll FThunk:00049274 NbFunc:32 (decimal:50) valid:YES
    rva:00049274 mod:msvcrt.dll ord:006F name:__getmainargs
    rva:00049278 mod:msvcrt.dll ord:0079 name:__mb_cur_max
    rva:0004927C mod:msvcrt.dll ord:0085 name:__p__environ
    rva:00049280 mod:msvcrt.dll ord:0087 name:__p__fmode
    rva:00049284 mod:msvcrt.dll ord:009A name:__set_app_type
    rva:00049288 mod:msvcrt.dll ord:00BF name:_assert
    rva:0004928C mod:msvcrt.dll ord:00C9 name:_cexit
    rva:00049290 mod:msvcrt.dll ord:00DF name:_ctype
    rva:00049294 mod:msvcrt.dll ord:00EC name:_errno
    rva:00049298 mod:msvcrt.dll ord:0101 name:_filelengthi64
    rva:0004929C mod:msvcrt.dll ord:0116 name:_fstati64
    rva:000492A0 mod:msvcrt.dll ord:0140 name:_iob
    rva:000492A4 mod:msvcrt.dll ord:0142 name:_isctype
    rva:000492A8 mod:msvcrt.dll ord:0173 name:_lseeki64
    rva:000492AC mod:msvcrt.dll ord:01B5 name:_onexit
    rva:000492B0 mod:msvcrt.dll ord:01BE name:_pctype
    rva:000492B4 mod:msvcrt.dll ord:01E0 name:_setmode
    rva:000492B8 mod:msvcrt.dll ord:01FD name:_strnicmp
    rva:000492BC mod:msvcrt.dll ord:0220 name:_vsnprintf
    rva:000492C0 mod:msvcrt.dll ord:027C name:abort
    rva:000492C4 mod:msvcrt.dll ord:0283 name:atexit
    rva:000492C8 mod:msvcrt.dll ord:0294 name:fclose
```

IAT Infos Needed

OEP 00001220
RVA 00049270
Size 000000D0

New Import Infos (IID+ASCII+LOADER)

RVA 00000000 Size 00000254

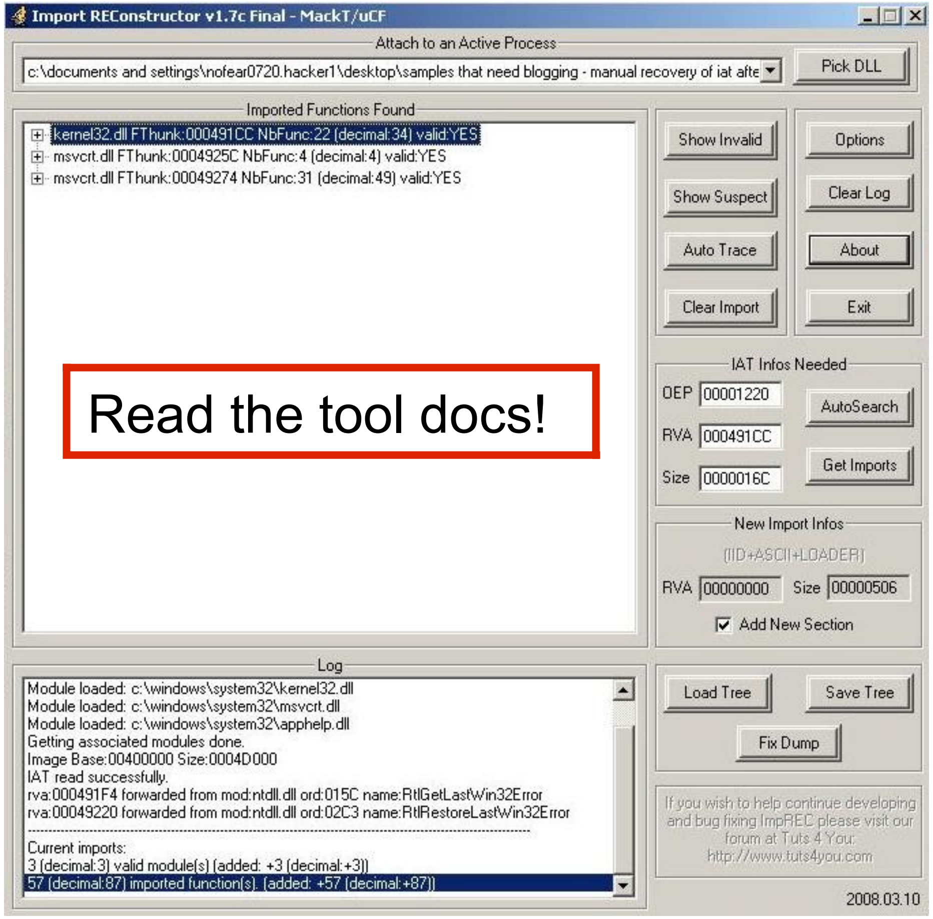
☑ Add New Section

Log

```
Module loaded: c:\windows\system32\ntdll.dll
Module loaded: c:\windows\system32\kernel32.dll
Module loaded: c:\windows\system32\msvcrt.dll
Module loaded: c:\windows\system32\apphelp.dll
Getting associated modules done.
Image Base:00400000 Size:0004D000
Original IAT RVA found at: 00049284 in Section RVA: 00039000 Size:00013000
IAT read successfully.
------------------------------------------------------------------------------------------
Current imports:
1 (decimal:1) valid module(s) (added: +1 (decimal:+1))
32 (decimal:50) imported function(s). (added: +32 (decimal:+50))
```

If you wish to help continue developing and bug fixing ImpREC please visit our forum at Tuts 4 You: http://www.tuts4you.com

2008.03.10



Import REConstructor v1.7c Final - MackT/uCF

Attach to an Active Process: c:\documents and settings\nofear0720.hacker1\desktop\samples that need blogging - manual recovery of iat afte

Imported Functions Found

```
⊞ kernel32.dll FThunk:000491CC NbFunc:22 (decimal:34) valid:YES
⊞ msvcrt.dll FThunk:0004925C NbFunc:4 (decimal:4) valid:YES
⊞ msvcrt.dll FThunk:00049274 NbFunc:31 (decimal:49) valid:YES
```

Read the tool docs!

IAT Infos Needed

OEP 00001220
RVA 000491CC
Size 0000016C

New Import Infos (IID+ASCII+LOADER)

RVA 00000000 Size 00000506

☑ Add New Section

Log

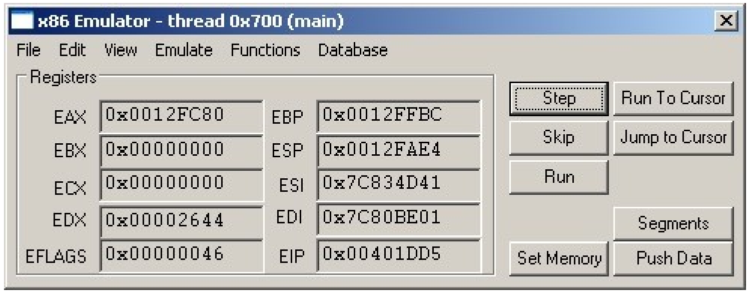
```
Module loaded: c:\windows\system32\kernel32.dll
Module loaded: c:\windows\system32\msvcrt.dll
Module loaded: c:\windows\system32\apphelp.dll
Getting associated modules done.
Image Base:00400000 Size:0004D000
IAT read successfully.
rva:000491F4 forwarded from mod:ntdll.dll ord:015C name:RtlGetLastWin32Error
rva:00049220 forwarded from mod:ntdll.dll ord:02C3 name:RtlRestoreLastWin32Error
------------------------------------------------------------------------------------------
Current imports:
3 (decimal:3) valid module(s) (added: +3 (decimal:+3))
57 (decimal:87) imported function(s). (added: +57 (decimal:+87))
```

If you wish to help continue developing and bug fixing ImpREC please visit our forum at Tuts 4 You: http://www.tuts4you.com

2008.03.10

Defeating HyperUnpackMe2 With an IDA Processor Module

- Not as executable protectors of yesteryear wherein the contents of the executable in memory, minus the import information are restored eventually
- Breaking a modern protector (2007)
  - This article is an exercise in overkill...
- Active measures to frustrate attempts to dump the process
  - Convert code to proprietary byte code
  - Use virtual machines operating upon polymorphic byte code
  - Copying portions of the code elsewhere in the process address space but not in PE image (stolen bytes)
- Very interesting read!
  - http://www.openrce.org/articles/full_view/28
- The x86 Emulator plugin for IDA Pro
  - Useful for obfuscated code
  - http://www.idabook.com/x86emu/

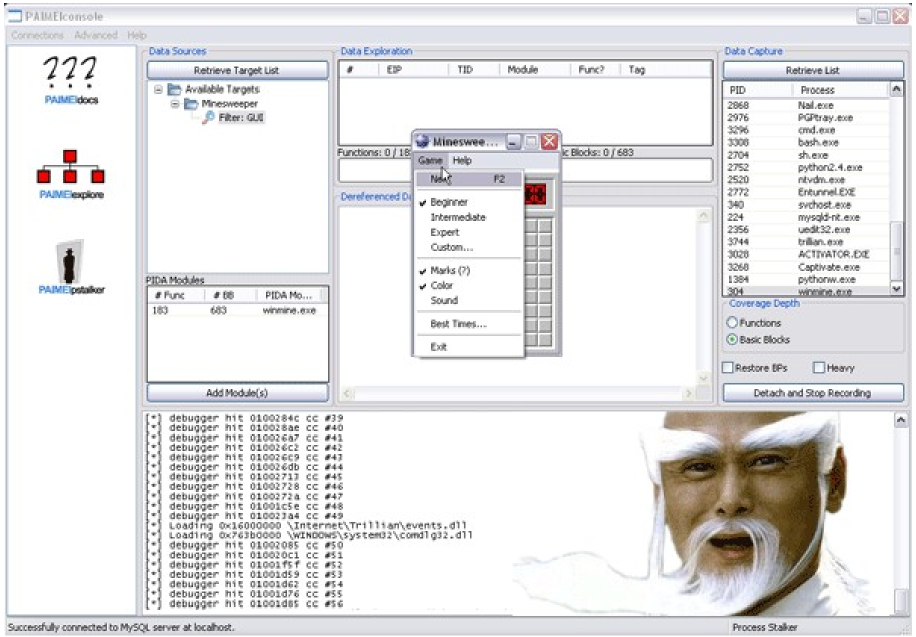

x86 Emulator - thread 0x700 (main)

Registers

EAX	0x0012FC80	EBP	0x0012FFBC
EBX	0x00000000	ESP	0x0012FAE4
ECX	0x00000000	ESI	0x7C834D41
EDX	0x00002644	EDI	0x7C80BE01
EFLAGS	0x00000046	EIP	0x00401DD5

RCE frameworks

- The goal of frameworks is to reduce the time from "idea" to prototype to a matter of minutes, instead of days
- Frameworks can essentially be thought of as a reverse engineer's swiss army knife - can also be plugins to OllyDbg, IdaPro etc.
- Used for static and dynamic tasks such as: fuzzer assistance, code coverage tracking, data flow tracking, malcode analysis etc.
- http://www.woodmann.com/collaborative/tools/index.php/Category:Reverse_Engineering_Frameworks
- Examples
  - Radare (book included) and ariadne
  - ERESI (The Reverse Engineering Software Interface)
  - Security Research and Development Framework (SRDF)
  - Malcode Analysis Pack (iDefense Labs)



PIDA Modules

# Func	# BB	PIDA Mo...
183	683	winmine.exe

PID	Process
2868	Nail.exe
2976	PGPtray.exe
3296	cmd.exe
3308	bash.exe
2704	sh.exe
2752	python2.4.exe
2520	ntvdm.exe
2772	Entunnel.EXE
340	svchost.exe
224	mysqld-nt.exe
2356	uedit32.exe
3744	trillian.exe
3028	ACTIVATOR.EXE
3268	Captivate.exe
1384	pythonw.exe
304	winmine.exe

```
[*] debugger hit 0100284c cc #39
[*] debugger hit 010028ae cc #40
[*] debugger hit 010026a7 cc #41
[*] debugger hit 010026c2 cc #42
[*] debugger hit 010026c9 cc #43
[*] debugger hit 010026db cc #44
[*] debugger hit 01002713 cc #45
[*] debugger hit 01002728 cc #46
[*] debugger hit 0100272a cc #47
[*] debugger hit 01001c5e cc #48
[*] debugger hit 010023a4 cc #49
[*] Loading 0x16000000 \Internet\Trillian\events.dll
[*] Loading 0x763b0000 \WINDOWS\system32\comdlg32.dll
[*] debugger hit 01002085 cc #50
[*] debugger hit 010020c1 cc #51
[*] debugger hit 01001f5f cc #52
[*] debugger hit 01001d59 cc #53
[*] debugger hit 01001d62 cc #54
[*] debugger hit 01001d76 cc #55
[*] debugger hit 01001d85 cc #56
```

Successfully connected to MySQL server at localhost.

Malware analysis

Boken har ett stort antal verktyg på en DVD (vilken kan laddas ner) som är mycket intressanta!

Jag har lagt DVDn på [server]\malware\malwarecookbook.com

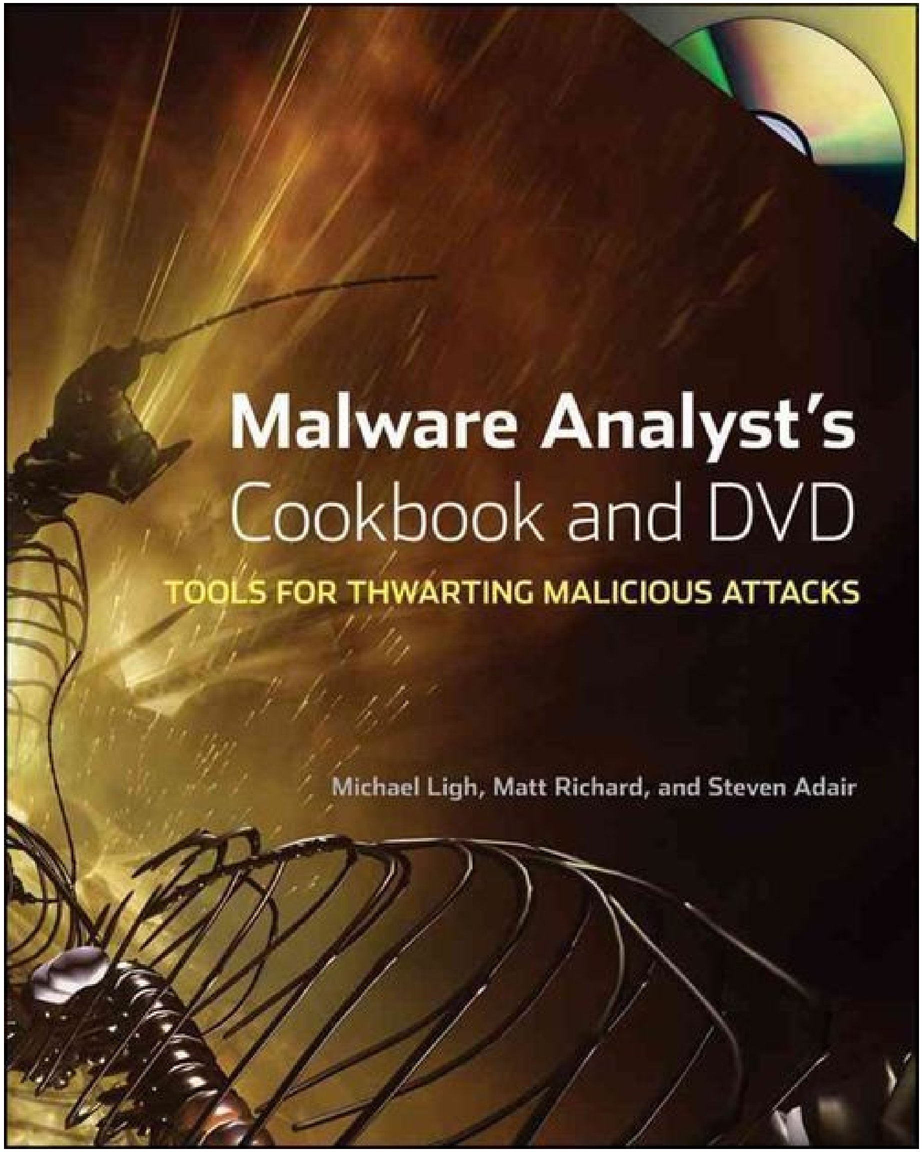
Password “infected”

På internet

http://www.malwarecookbook.com/

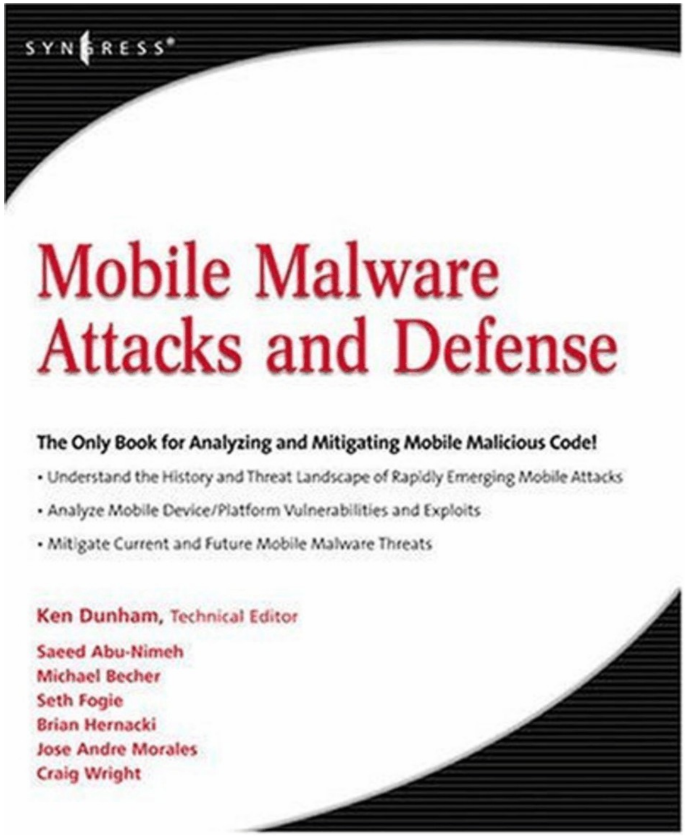
Full pott på Amazon, läs recensionerna för att veta mer

Helt enkelt bästa boken i ämnet!


Malware Analyst’s
Cookbook and DVD
TOOLS FOR THWARTING MALICIOUS ATTACKS
Michael Ligh, Matt Richard, and Steven Adair

Malware analysis

- Malware Forensics Investigating and Analyzing Malicious Code
  - http://www.elsevier.com/wps/find/bookdescription.cws_home/714697/description
- Mobile malware attacks and defense
  - http://www.elsevier.com/wps/find/bookdescription.cws_home/715445/description



SYNGRESS®

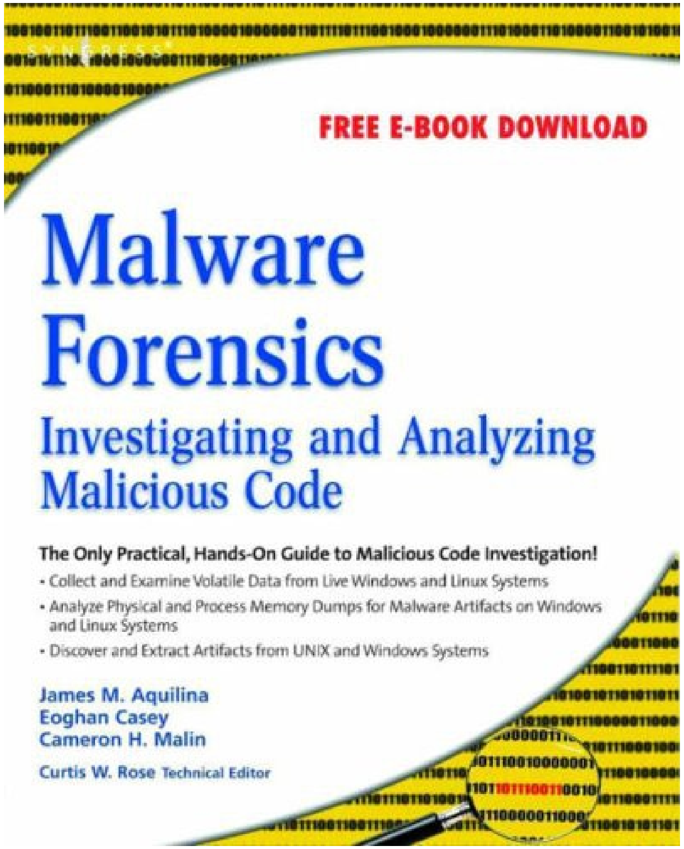
Mobile Malware Attacks and Defense

The Only Book for Analyzing and Mitigating Mobile Malicious Code!

- Understand the History and Threat Landscape of Rapidly Emerging Mobile Attacks
- Analyze Mobile Device/Platform Vulnerabilities and Exploits
- Mitigate Current and Future Mobile Malware Threats

Ken Dunham, Technical Editor

Saeed Abu-Nimeh
Michael Becher
Seth Fogie
Brian Hernacki
Jose Andre Morales
Craig Wright



SYNGRESS®

FREE E-BOOK DOWNLOAD

Malware Forensics

Investigating and Analyzing Malicious Code

The Only Practical, Hands-On Guide to Malicious Code Investigation!

- Collect and Examine Volatile Data from Live Windows and Linux Systems
- Analyze Physical and Process Memory Dumps for Malware Artifacts on Windows and Linux Systems
- Discover and Extract Artifacts from UNIX and Windows Systems

James M. Aquilina
Eoghan Casey
Cameron H. Malin

Curtis W. Rose Technical Editor

Malware analysis links

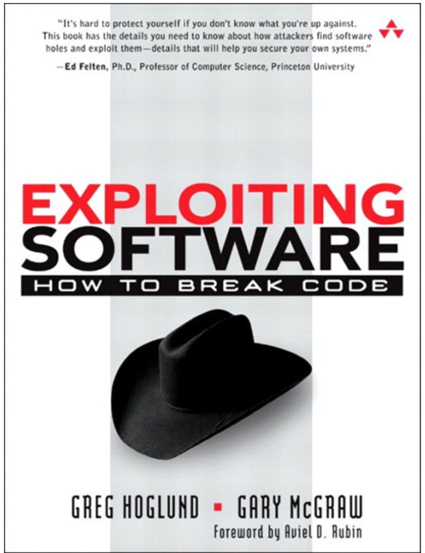
- Reverse Engineering Wiki books (**mycket bra**)
  - http://en.wikibooks.org/wiki/Reverse_Engineering
- Reverse Engineering Malware, article in 5 parts
  - http://www.windowsecurity.com/articles/Reverse-Engineering-Malware-Part1.html
- Reverse Engineering Hostile Code
  - http://www.securityfocus.com/infocus/1637
- Fifteen Minute Malware Analysis
  - http://forensiczone.blogspot.com/2008/02/fifteen-minute-malaware-analysis.html
- Fifteen Minute Virus Analysis
  - http://forensiczone.blogspot.com/2008/03/practical-of-15-minute-virus-analysis.html
- Basic tutorial about how to dump a process and update the IAT using Immunity Debug, LordPE, and ImpRec
  - https://www.openrce.org/blog/view/1135/Basic_tutorial_about_how_to_dump_a_process_and_update_the_IAT_using_Immunity_Debug,_LordPE,_and_ImpRec
- REVERSE CODE ENGINEERING: AN IN-DEPTH ANALYSIS OF THE BAGLE VIRUS
  - Paper
    - http://rozinov.sfs.poly.edu/papers/bagle_analysis_v.1.0.pdf
  - Presentation
    - http://rozinov.sfs.poly.edu/presentations/bagle_analysis_v.1.0-presentation.pdf

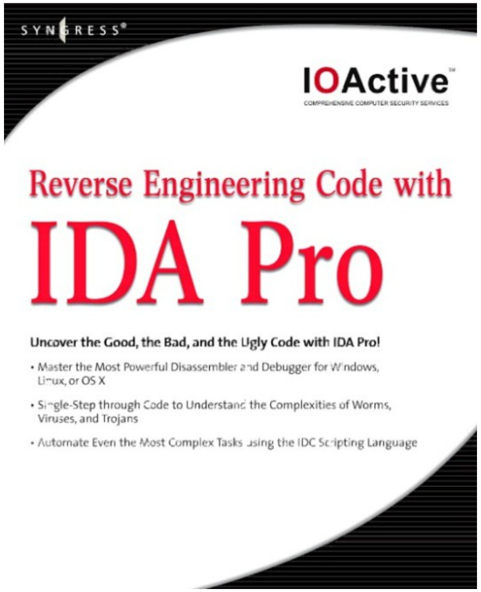
Learning IDA Pro and OllyDbg

- The best video packages learning IDA and OllyDbg
  - TiGa’s Video Tutorials - Reverse Engineering Using IDA Pro
  - Learn how to do Reverse Code Engineering for newbies by Lena (OllyDbg)
  - [server]\training_forensics_networkanalysis_pen-test\Reverse.Code.Engineering
- Reverse Engineering Code with IDA Pro

http://www.elsevierdirect.com/companion.jsp?ISBN=9781597492379

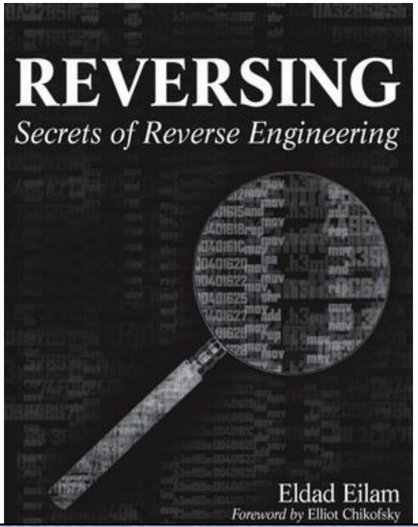
  - 1,5 * on amazon ☺
- Exploiting Software: How to Break Code
  - 4,5 * on amazon


"It's hard to protect yourself if you don't know what you're up against. This book has the details you need to know about how attackers find software holes and exploit them—details that will help you secure your own systems."
—Ed Felten, Ph.D., Professor of Computer Science, Princeton University
EXPLOITING
SOFTWARE
HOW TO BREAK CODE
GREG HOGLUND ▪ GARY McGRAW
Foreword by Aviel D. Rubin


SYNGRESS®
IOActive
COMPREHENSIVE COMPUTER SECURITY SERVICES
Reverse Engineering Code with
IDA Pro
Uncover the Good, the Bad, and the Ugly Code with IDA Pro!
• Master the Most Powerful Disassembler and Debugger for Windows, Linux, or OS X
• Single-Step through Code to Understand the Complexities of Worms, Viruses, and Trojans
• Automate Even the Most Complex Tasks using the IDC Scripting Language

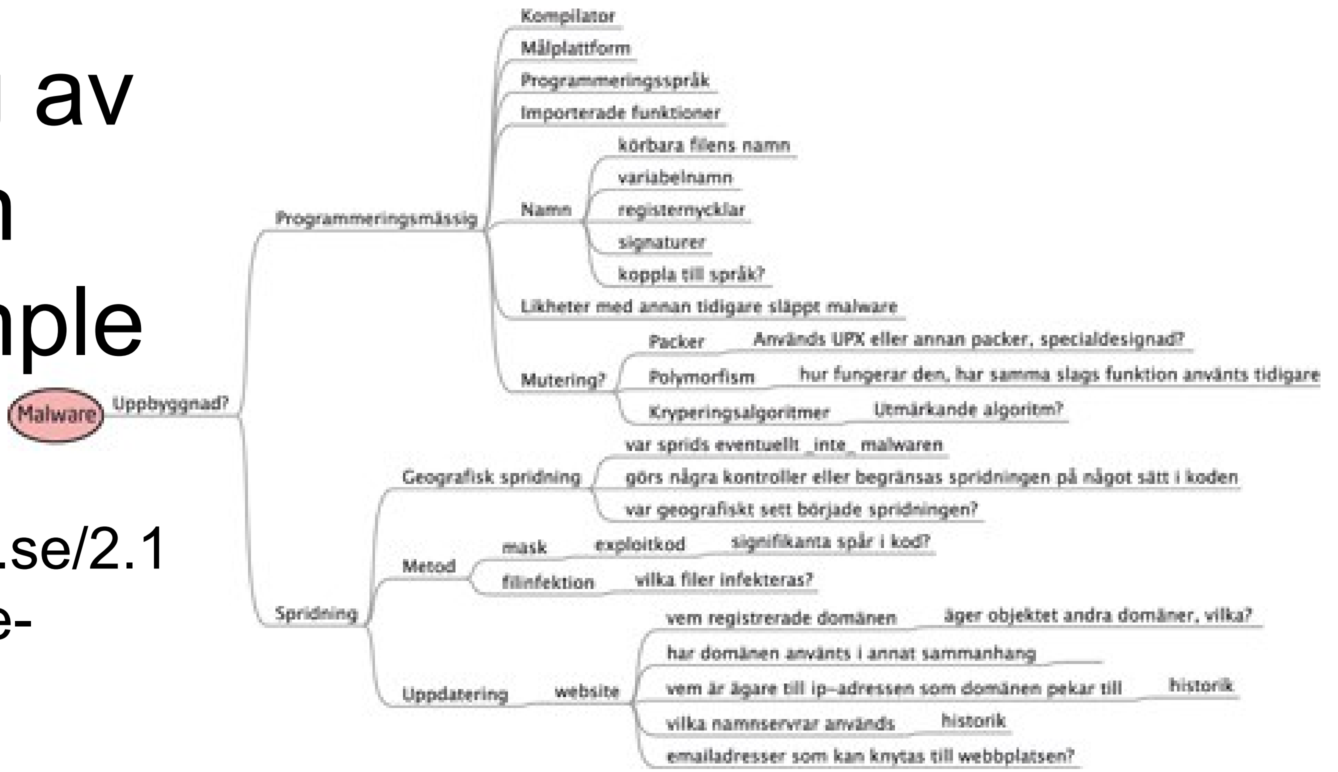
Readings and RCE sites etc.

- Introduction to Reverse Engineering Software
  - http://www.acm.uiuc.edu/sigmil/RevEng/index.html
- Open Reverse Code Engineering community
  - Great site! Loads of plugins for IDA Pro and OllyDbg
  - http://www.openrce.org
- The book Reversing: Secrets of Reverse Engineering, 4,5 *
  - http://en.wikipedia.org/wiki/Reversing:_Secrets_of_Reverse_Engineering
- Woodmann Reverse Engineering
  - http://www.woodmann.com
- TUTS4YOU
  - http://tuts4you.com
- The Reverse Code Engineering Community
  - http://www.reverse-engineering.net/
- Crackmes.de – reversers' playground
  - http://crackmes.de/
- Blogs
  - http://x9090.blogspot.com/
  - http://reversengineering.wordpress.com/

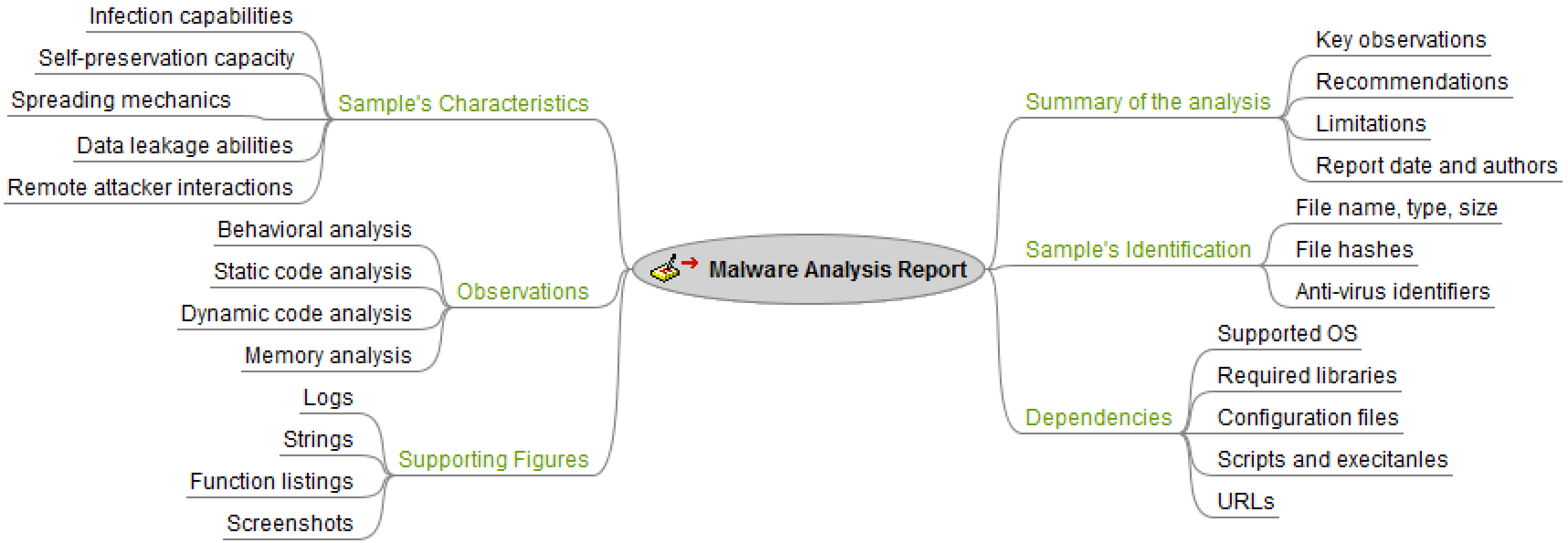


End!
and
Backups

Kartläggning av upphovsman Report example


Malware
Uppbyggnad?
Programmeringsmässig
Kompilator
Målplattform
Programmeringsspråk
Importerade funktioner
Namn
körbara filens namn
variabelnamn
registernycklar
signaturer
koppla till språk?
Likheter med annan tidigare släppt malware
Mutering?
Packer
Används UPX eller annan packer, specialdesignad?
Polymorfism
hur fungerar den, har samma slags funktion använts tidigare
Kryperingsalgoritmer
Utmärkande algoritm?
Spridning
Geografisk spridning
var sprids eventuellt _inte_ malwaren
görs några kontroller eller begränsas spridningen på något sätt i koden
var geografiskt sett började spridningen?
Metod
mask
exploitkod
signifikanta spår i kod?
filinfektion
vilka filer infekteras?
Uppdatering
website
vem registrerade domänen
äger objektet andra domäner, vilka?
har domänen använts i annat sammanhang
vem är ägare till ip-adressen som domänen pekar till
historik
vilka namnservrar används
historik
emailadresser som kan knytas till webbplatsen?

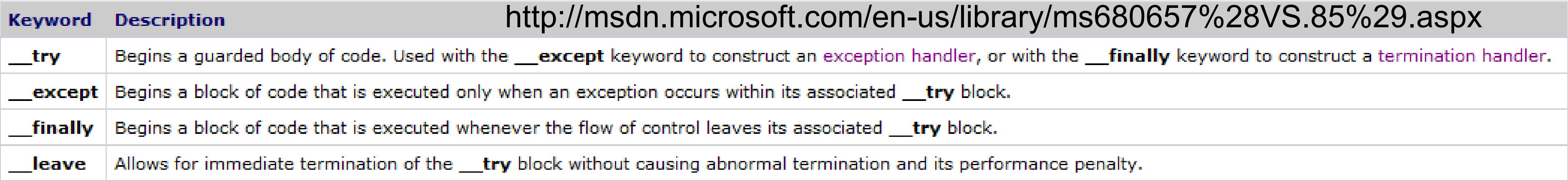
http://sakerhet.idg.se/2.1070/1.272980/yrke-virusjagare



Structured Exception Handling (SEH)

- Structured exception handling is a mechanism for handling both hardware and software exceptions in Windows (Windows API)
- One of Microsoft's main motivations for adding SEH to Windows was to ease the development of the operating system and make the system more robust
- Mainly used in C and works like C++/Java/.NET exception handling
- RaiseException Function works like throw

http://msdn.microsoft.com/en-us/library/ms680657%28VS.85%29.aspx



Keyword	Description
__try	Begins a guarded body of code. Used with the **__except** keyword to construct an exception handler, or with the **__finally** keyword to construct a termination handler.
__except	Begins a block of code that is executed only when an exception occurs within its associated **__try** block.
__finally	Begins a block of code that is executed whenever the flow of control leaves its associated **__try** block.
__leave	Allows for immediate termination of the **__try** block without causing abnormal termination and its performance penalty.

```
// Termination-Handler Syntax
__try {
    // guarded body of code
}
__finally {
    // __finally block
}
```

```
// Exception-Handler Syntax
__try {
    // guarded body of code
}
__except (filter-expression) {
    // exception-handler block
}
```

SEH nested example

```
DWORD FilterFunction()
{
    printf("1 ");                     // printed first
    return EXCEPTION_EXECUTE_HANDLER;
}

VOID main(VOID)
{
    __try {
        __try {
            RaiseException(
                1,                    // exception code
                0,                    // continuable exception
                0, NULL);             // no arguments
        }
        __finally {
            printf("2 ");             // this is printed second
        }
    }
    __except ( FilterFunction() ) {
        printf("3\n");                // this is printed last
    }
}
```

IA-32 (x86) Hardware breakpoints

- The IA-32 family of processors provides support for 4 hardware breakpoints
  - The hardware breakpoints use special debug registers
  - These registers contain the breakpoint addresses as well as control information and breakpoint type
  - Breakpoint addresses are stored in debug registers D0 to D3
  - In order to set breakpoints a size field is needed. The possible sizes are 1, 2, or 4 bytes. Breaks on execution use a size of 1 byte
  - The possible sizes have been expanded to include 8 bytes for 64-bit CPUs
- There are various conditions to trigger the breakpoints.
  - Break on execution
  - Break on memory access (reads and writes)
  - Break on memory write only
  - Break on I/O port access (rarely used, most debuggers do not have this as an option)