

Part 5 – Reverse Engineering Tools Basics

Swapnil Pathak



www.SecurityXploded.com

Disclaimer

The Content, Demonstration, Source Code and Programs presented here is "AS IS" without any warranty or conditions of any kind. Also the views/ideas/knowledge expressed here are solely of the trainer's only and nothing to do with the company or the organization in which the trainer is currently working.

However in no circumstances neither the trainer nor SecurityXploded is responsible for any damage or loss caused due to use or misuse of the information presented here.

Acknowledgement

- Special thanks to **null & Garage4Hackers** community for their extended support and cooperation.
- Thanks to all the **Trainers** who have devoted their precious time and countless hours to make it happen.

Reversing & Malware Analysis Training

This presentation is part of our **Reverse Engineering & Malware Analysis** Training program. Currently it is delivered only during our local meet for FREE of cost.



For complete details of this course, visit our [Security Training page](#).

Who am I

Swapnil Pathak

- Member SecurityXploded
- Security Researcher @ McAfee Labs
- RE, Malware Analysis, Network Security
- Email: swapnilpathak101@gmail.com

Course Q&A

- ⦿ Keep yourself up to date with latest security news
 - <http://www.securityphresh.com>

- ⦿ For Q&A, join our mailing list.
 - <http://groups.google.com/group/securityxploded>

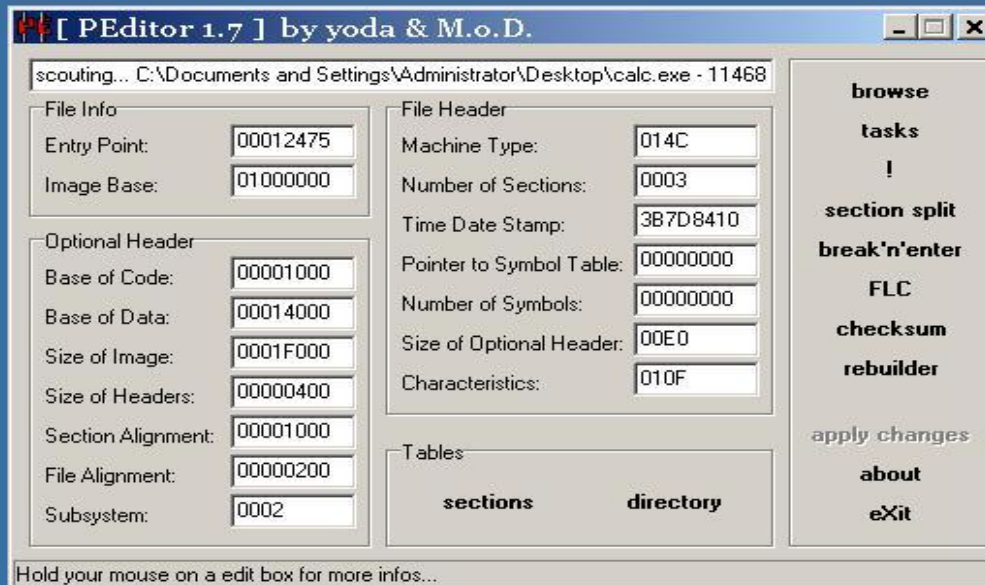
Presentation Outline

- PE Tools
 - PE Editor
- Disassemblers
 - IDA Pro
- Debuggers
 - OllyDbg
- Other Useful Tools

PE Tools

- ⦿ Portable Executable Editor
 - Allow user to view, edit data structures present in PE file format
- ⦿ Some Common Use Cases:
 - Change entry point of the executable
 - View Import/Export/Sections within EXE
 - Change characteristics of a file i.e. Dll to Exe
 - Fix anomalies of PE files
- ⦿ PE editors
 - Hiew, PE Editor, CFF Explorer, StudPE, LordPE etc

PE Editor



[PEEditor 1.7] by yoda & M.o.D.

scouting... C:\Documents and Settings\Administrator\Desktop\calc.exe - 11468

File Info

Entry Point: 00012475
Image Base: 01000000

Optional Header

Base of Code: 00001000
Base of Data: 00014000
Size of Image: 0001F000
Size of Headers: 00000400
Section Alignment: 00001000
File Alignment: 00000200
Subsystem: 0002

File Header

Machine Type: 014C
Number of Sections: 0003
Time Date Stamp: 3B7D8410
Pointer to Symbol Table: 00000000
Number of Symbols: 00000000
Size of Optional Header: 00E0
Characteristics: 010F

Tables

sections directory

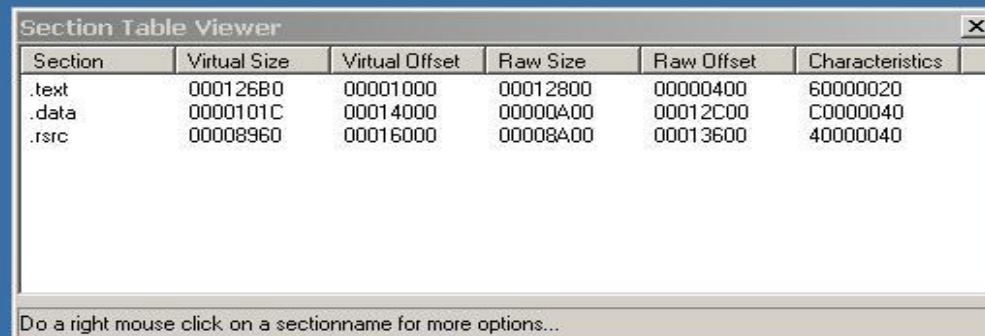
tasks

!
section split
break'n'enter
FLC
checksum
rebuilder

apply changes

about
eXit

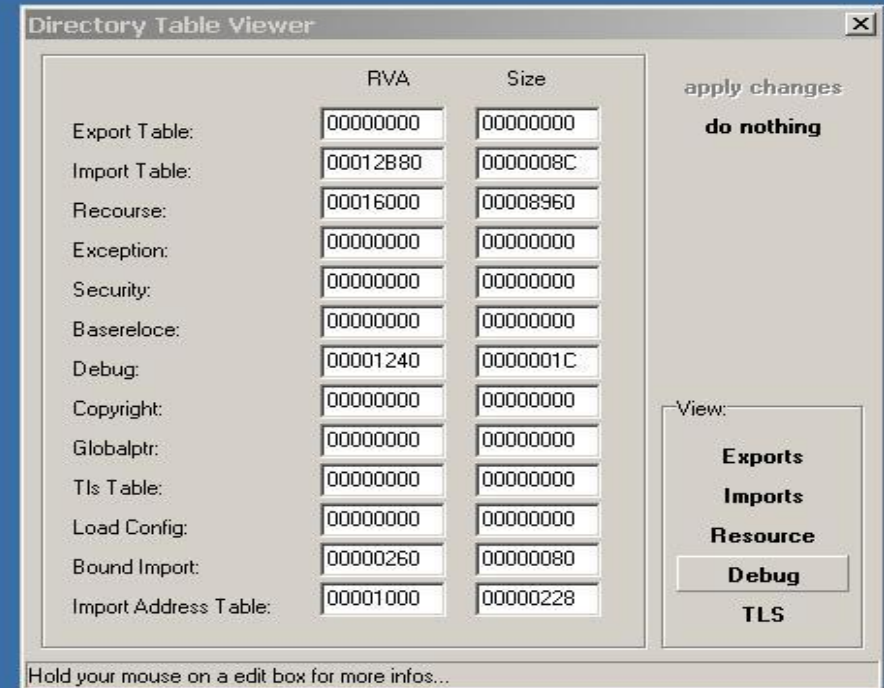
Hold your mouse on a edit box for more infos...



Section Table Viewer

Section	Virtual Size	Virtual Offset	Raw Size	Raw Offset	Characteristics
.text	000126B0	00001000	00012800	00000400	60000020
.data	0000101C	00014000	00000A00	00012C00	C0000040
.rsrc	00008960	00016000	00008A00	00013600	40000040

Do a right mouse click on a sectionname for more options...



Directory Table Viewer

	RVA	Size
Export Table:	00000000	00000000
Import Table:	00012B80	0000008C
Recourse:	00016000	00008960
Exception:	00000000	00000000
Security:	00000000	00000000
Baserebase:	00000000	00000000
Debug:	00001240	0000001C
Copyright:	00000000	00000000
Globalptr:	00000000	00000000
Tls Table:	00000000	00000000
Load Config:	00000000	00000000
Bound Import:	00000260	00000080
Import Address Table:	00001000	00000228

apply changes

do nothing

View:

Exports
Imports
Resource
Debug
TLS

Hold your mouse on a edit box for more infos...

IDA Pro

- ⦿ Popular commercial software used for Reverse Engineering.
- ⦿ Disassembler and Debugger in one tool
- ⦿ Supports variety of executable formats for different processors and OS
- ⦿ Provides both Text & Graphical view of the code flow.
- ⦿ View strings, Imports, Exports referenced in the executable
- ⦿ Support Plugins
- ⦿ Some of the useful plugins
 - X86 emulator
 - IDAPython
 - IDARub
 - IDA Scripts
 - Windbg

IDA Pro Functions

- ⦿ IDA Windows & purpose – IDA View, Hex View, Imports, Strings, Functions windows etc.
- ⦿ Graphs & Text view (use “space” to switch between views)
- ⦿ Xref To & Xref From – powerful feature
- ⦿ Jump & Search
- ⦿ Edit function names (N), Add comments (;)

IDA Pro in Action

The screenshot displays the IDA Pro interface for a file named 'arch - Copy.exe'. The main window shows assembly code with a control flow graph overlaid. The assembly code includes instructions like 'test', 'jle', 'mov', 'xor', 'lea', and 'sub'. The control flow graph on the left shows the flow between different code blocks. The 'Exports' window on the right lists various symbols and their addresses. The 'Imports' window shows the addresses and names of imported functions. The 'Output window' at the bottom displays the results of a sample IDC plugin call.

IDA - C:\Users\Daniel\Desktop\arch.idb (arch - Copy.exe)

File Edit Jump Search View Debugger Options Windows Help

Windbg debugger

Functions window

Function name

- start
- sub_0_401100
- sub_0_401140
- sub_0_401730
- sub_0_4017C0
- sub_0_4018A0
- sub_0_402520
- sub_0_4026D0

Line 5 of 103

Graph overview

IDA View-A, Exports, Imports

Hex View-A

Structures

Enums

Exports

Name

- start

Line 1 of 1

Imports

Address	Ord	Name
0040F128		__errno
0040F12C		__getreent
0040F130		__main
	F134	__mb_cur_max
	F138	__ctype_
	F13C	__dll_crt0@0
	F140	__exit
		__impure_ptr
0040F144		abort
0040F148		atexit
0040F14C		calloc
0040F150		cygwin_internal
0040F154		exit

To: sub_0_4017C0:loc_0_401895

loc_0_401895:

```
sub [ebp+var_10], edi
jmp loc_0_4017F2
sub_0_4017C0 endp
```

loc_0_401843:

```
mov edx, [ebx]
add esi, 1
mov eax, [ecx]
mov [ebx], eax
add ebx, 4
mov [ecx], edx
add ecx, 4
```

loc_0_401880:

```
mov edx, [ebx]
add esi, 1
mov eax, [ecx]
mov [ebx], eax
add ebx, 4
mov [ecx], edx
add ecx, 4
```

loc_0_401859:

```
test edx, edx
jle short loc_0_401859
```

loc_0_401861:

```
test edi, edi
jle short loc_0_401895
```

loc_0_401895:

```
mov ecx, [ebp+var_18]
xor esi, esi
mov edx, [ebp+var_20]
mov eax, [ebp+var_10]
add eax, [ebp+var_18]
sub eax, [ebp+var_14]
lea ebx, [edx+ecx*4]
lea ecx, [edx+eax*4]
lea esi, [esi+0]
```

loc_0_401895:

```
mov ecx, [ebp+var_18]
xor esi, esi
mov edx, [ebp+var_20]
mov eax, [ebp+var_10]
add eax, [ebp+var_18]
sub eax, [ebp+var_14]
lea ebx, [edx+ecx*4]
lea ecx, [edx+eax*4]
lea esi, [esi+0]
```

100.00% (27, 1157) (546, 191) 00000C98 00401898: sub 0 4017C0+D8

Output window

Sample IDC plugin: term() has been called
init() called!
term() called!

Python

AU: idle Down Disk: 21GB

Ollydbg

- ⦿ Most Popular Ring 3 Debugger used in Reversing
- ⦿ Provides the below mentioned functionalities
 - Debugging program step by step (Single Stepping)
 - Software, Hardware and Memory based Breakpoints
 - Examine the current state of the program wrt variables, memory etc.
 - Change the flow or state of Program by directly editing Instructions, Registers or Memory.
- ⦿ Support Plugins, Here are popular ones,
 - OllyDump
 - OllyAdvanced
 - OllyScript

Ollydbg Cont.

- ⦿ Different Windows – CPU, Registers, Dump, Stack screens
- ⦿ Trace Into, Trace Over, Exceptions
- ⦿ Integrate windows API help file in ollydbg
- ⦿ Shortcuts (imp. Only)
 - F7 – Step into [call]
 - F8 – Single step [execute call]
 - F9 – Run
 - F2 – Breakpoint

Ollydbg in Action

The screenshot displays the Ollydbg interface with the following components:

- Assembly View:** Shows assembly instructions for the CPU's main thread in module ntdll. The instruction at address 77CD15EE is highlighted in red: `ADD ESP, 4`. Other instructions include `RETN 18`, `MOV EAX, 12F`, `XOR ECK, ECK`, `LEA EDX, DWORD PTR SS:[ESP+4]`, and `CALL DWORD PTR FS:[C0]`.
- Registers (FPU):** Lists the state of various registers. EAX is 00000000, ECX is 00000000, EDX is 00000000, EBX is 0018FB3C, ESP is 0018FB28, EBP is 00000000, ESI is 00000000, EDI is 00000000, and EIP is 77CD15EE. Control flags (C, P, A, Z, S, T, D, O) and the error flag (EFL) are also shown.
- Memory Dump:** A table showing memory addresses, hex dumps, and ASCII representations. The address 0042D050 is highlighted in red, showing the hex value `41 70` and the ASCII characters `lp`.
- Command Line:** Shows the command `Process terminated, exit code 4000001E (1073741854).`
- Status Bar:** Indicates the process is **Terminated**.

UPX Unpacking Ollyscript

Here is example for Unpacking UPX based Malwares using OllyScript in OllyDbg

```
var hwdBP // Local variable to store hardware breakpoint
var softBP // Local variable to store software breakpoint
sti // Step into F7 command
findop eip, #61# // find next POPAD
mov hwdBP, $RESULT // Store $RESULT to hardware breakpoint local variable
bphws hwdBP, "x" // Set hardware breakpoint (execute) on the next POPAD
run // Run F9 command
findop eip, #E9????????# // Find the next JMP
mov softBP, $RESULT // Store $RESULT to software breakpoint local variable
bp softBP
run // Run to JMP instruction
sti // Step into the OEP
cmt eip, "<<>>"
msg "OEP found, you can dump the file starting from this address"
ret
```

Source : <http://x9090.blogspot.in/2009/07/ollyscript-tutorial-unpack-upx.html>

Useful Tools

- ⦿ Packer Identifier Tools
 - RDG packer detector
 - PEID
 - ExeScan

PEiD – PE Packer Identifier Tool



Reference

- [Complete Reference Guide for Reversing & Malware Analysis Training](#)

Thank You !



www.SecurityXploded.com