

Exam 2 Extra Credit

sdh140430

Swapnil Hasabe

Motivation of learning aspects of CTF:

1. CTF introduces the real tools and real types of problems without breaking the law. It provides opportunity to learn O.S, applications, cryptography, architecture, attack-defense techniques.

2. It comes into following categories:

Reverse Engineering	Cryptography	Exploitation
Programming	Debugging	Shell Coding

3. Following are the important elements of Reverse Engineering:

3.1. Difference between Static and Dynamic

Points	Static	Dynamic
Description	Looking at the code, figure things out	Examine the process during execution
Analysis	Disassemblers are used for static analysis e.g. IDA Pro, objdump	Debuggers are used for dynamic analysis. E.g. WinDBG, IDA, GDB

3.2. Tools used to analyze following two file format:

PE (Portable Executable): CFF Explorer, IDA, pefile (python library)

ELF (Executable and Linkable Format): readelf, objdump, file

3.3. Assembly (More focused on Intel Instructions): This section demonstrates how the instructions like mov, cmp, jmp, call works. It also describes the data types and sizes in assembly. It provides process memory layout for windows, Linux and virtual memory.

3.4.1 Registers: This section describes the all types of registers e.g. EIP, ESP, EBP, EAX, EBX, ECX, EDX, ESI and EDI

3.4.1 Three stack frames Prolog, Epilog and Epilog 2 have been explained in this section.

3.5. IDA

3.5.1: We can do many things, including grouping a set of nodes, color coding them, and renaming them

3.5.2: Knowing that all these checks error out on failure we can simplify the graph

3.6 Debugging (Dynamic Analysis through Debugging):

A good debugger will have several useful features

- Set breakpoints
- Step into / over
- Show loaded modules, SEH chain, etc.
- Memory searching

Dynamic Analysis: You cannot control everything

- If you want to skip over an instruction, or a function call, do it!
- If you want to bypass the “authentication” method or make it return true... you can!
- You can change register contents and memory values, whatever you want.
- You can even patch programs (make changes and save it to a new executable).

Exam 2 Extra Credit

sdh140430

Swapnil Hasabe

Overview of Dynamic analysis using GDB:

The screenshot shows a GDB session with the following steps and disassembly:

1. Run the program
(gdb) run
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/nomnom/FSU_Reversing/a.out
2. Where are we? Check out EIP
(gdb) x/5i \$eip
0x08048437 <main+3>: and esp,0xffffffff
0x0804843a <main+6>: sub esp,0x50
0x0804843d <main+9>: cmp DWORD PTR [ebp+0x8],0x3
0x08048441 <main+13>: je 0x08048456 <main+34>
0x08048443 <main+15>: mov DWORD PTR [esp],0x08048590
3. Continue until we hit an instruction of interest
(gdb) ni
0x0804843a in main ()
(gdb) ni
0x0804843d in main ()
(gdb) x/5i \$eip
0x0804843d <main+9>: cmp DWORD PTR [ebp+0x8],0x3
0x08048441 <main+13>: je 0x08048456 <main+34>
0x08048443 <main+15>: mov DWORD PTR [esp],0x08048590
0x0804844a <main+22>: call 0x08048364 <puts@plt>
0x0804844f <main+27>: mov eax,0xffffffff
4. Let's see what's being compared – we can see this jump is not taken
(gdb) x/xw \$ebp+0x8
0xbffffcd0: 0x00000001
(gdb) ni
0x08048441 in main ()
(gdb) ni
0x08048443 in main ()
(gdb) x/i \$eip
0x08048443 <main+15>: mov DWORD PTR [esp],0x08048590
(gdb) x/s 0x08048590
0x08048590: "Missing something?"
(gdb) ni
0x0804844a in main ()
(gdb) ni
Missing something?
0x0804844f in main ()
5. Check out the argument passed to puts

A red box highlights the instruction: `0x08048443 <main+15>: mov DWORD PTR [esp],0x08048590` with the text: "Aha! We don't satisfy the compare (1 != 3), and call puts, then exit!"

Overview of Dynamic analysis using IDA:

The screenshot shows IDA Pro with the following information:

- Dynamic Analysis - IDA**
 - The breakpoint has been hit, execution is stopped
- args:**
 - 013011FD loc_13011FD: ; Comperand
 - 013011FD push ebx ; Exchange
 - 013011FE push esi ; Destination
 - 013011FF push edi ; Destination
 - 01301200 call imp__InterlockedCompareExchange@12 ; InterlockedCompareExchange(x,x,x)
 - 01301206 cmp eax, ebx
 - 01301208 jz short loc_1301223
- The stack:**
 - Stack view: 0046FA46 01303380 .data: native_s
 - 0046FA48 00470000
 - 0046FA4A 00000000
 - 0046FA4C 00000000
 - 0046FA4E 00000000
 - 0046FA50 00000000
 - 0046FA52 00000000
 - 0046FA54 00000000
 - 0046FA56 7EFDE000 debug@14:7EFDE000
- The registers:**
 - General registers
 - EAX 7EFDD000 TIB[00000000]:7EFDD000
 - EBX 00000000
 - ECX 0046FA68 Stack[00000008]:0046FA68
 - EDX 00000001
 - ESI 00470000
 - EDI 01303380 .data: __native_startup_lock
 - EBP 0046FA7C Stack[00000008]:0046FA7C
 - ESP 0046FA40 Stack[00000008]:0046FA40
 - EIP 01301200 __tmainCRTStartup+38
 - EFL 00000202

Everyone has their own preferences

But combination of the IDA and GDB will undoubtedly yield the best results.

Conclusion:

Following learning goals achieved:

- Opening up and examining a binary and looking at its sections to get a feel for it
- Working with and simplifying the disassembly
- Converting back to source code where needed
- Using a debugger to fill in the gaps or manipulate program execution