PESU Center for
Information Security,
Forensics and
Cyber Resilience

Reverse Engineering

About me

- My name is Gagan Chandan. I’m a third semester CSE student here at PES RR Campus.
- I have been into cybersecurity for a little over 2 years now. I have mostly focussed on reverse engineering, vulnerability research and software fuzzing.
- I have lots of CTF experience having played with two different teams, inlcuding our team at ISFCR, mainly working on Reverse Engineering and Binary Exploitation challenges. Both these teams have had multiple top 3 finishes in competitions and have regularly featured among India’s top teams during my time with them

What is Reverse Engineering

- Reverse Engineering is the process of understanding the functionality of a program or application, usually one without the source code available, in order to discover deeper flaws within it.
- It is heavily used in vulnerability research and malware analysis.
- It is also used for modifying closed source software. It is often used for modding games and even for cracking commercial software.

x64 Assembly

Assembly, or ASM, is a term used to refer to any low level language with a strong, direct correspondence between instructions in the language and the architecture's machine code. It is used to directly communicate with a computer's hardware.

Different CPU architectures have different assembly languages. They are also called Instruction Set Architectures (ISA).

x64 or x86-64 ISA is used by 64-bit Intel and AMD processors.

Registers

- *Small, storage areas built into the CPU.*
- *In x86-64, there are 16 general purpose registers(although two are not actually general purpose as we will come to see), one instruction pointer and the RFLAGS register.*
- *Registers in x86-64 are 64 bits wide. In x86, they are 32 bits wide.*

Registers

- General purpose registers: rax, rbx, rcx, rdx ,rsi, rdi, rbp, rsp, r8, r9, r10, r11, r12, r13, r14 ,r15
- rbp : stores the address of the base of the current stack frame.
- rsp : stores the address of the top of the stack.
- Instruction pointer: rip. Stores the address of the next instruction.

The Stack

- Conceptual area in RAM. Allocated by the OS when the program starts.
- Last In First Out (LIFO) data structure. Push and pop operations are used.
- Grows from higher addresses to lower addresses.
- Local variables, temporary values and sometimes function arguments are stored on the stack.
- Each function is alloted a segment of the stack and is referred to as that function's *stack frame*.

Instructions

- The x86 and x64 families of processors use a kind of architecure called CISC architecture.
- CISC stands for Complex Instruction Set Computer. It means that a single instruction can perform multiple low level operations or multi-step operations.
- According to Intel’s official documentation, there are 3,684 instruction variants int the x64 ISA.
- We will only be looking at 21 of the most commonly seen ones.

NOP

- Syntax: *nop*
- Does nothing!!!

MOV

- Syntax: *mov destination, source*
- Moves the value stored in *source* into *destination*.
- Examples:
- *mov rax, 16*
- *mov rax, rbx*
- *mov rax, [rbx]*

PUSH

- Syntax: *push operand*
- Pushes the operand onto the top of the stack
- Examples:
- *push rax*
- *push 1337*
- *push DWORD PTR [rax]*

POP

- Syntax: *pop destination*
- Pops off the top element of the stack into the destination
- Examples:
- *pop rax*

LEA

- Syntax: *lea destination, source*
- Loads the effective address of source into destination.
- Examples:
- *lea rax, [rsp]*
- *lea rdi, [rip+0x2000]*

Add

- Syntax: *add destination, source*
- Add the value of source to the value at destination.
- Examples:
- *add rax, rbx*
- *add rax, 20*
- *add rax, [rbp - 0x04]*

SUB

- Syntax: sub *destination, source*
- Subtract the value of source from destination.
- Examples:
- *sub rax, rbx*
- *sub rax, 20*
- *sub rax, [rbp - 0x04]*

NOT

- Syntax: *not argument*
- Inverts each bit of the argument.
- Examples:
- *not rax*

AND

- Syntax: *and operand1, operand2*
- Performs bitwise and on the operands and stores the value at operand1.
- Examples:
- *and rax, rax*
- *and rax, 0*

OR

- Syntax: or *operand1, operand2*
- Performs bitwise or on the operands and stores the value at operand1.
- Examples:
- *or rax, rax*
- *or rax, 1*

XOR

- Syntax: xor *operand1, operand2*
- Performs bitwise xor on the operands and stores the value at operand1.
- Examples:
- *xor rax, rax*
- *xor rax, 23*

CMP

- Syntax: cmp *operand1, operand2*
- Compares the operands by subtraction and sets necessary flags.
- Examples:
- *cmp rax, rbx*
- *cmp rcx, 0x10*

TEST

- Syntax: test *operand1, operand2*
- Performs bitwise and on the operands and sets the necessary flags.
- Examples:
- *test rax, rax*

JMP

- Syntax: jmp addr
- Jump to given address and resume execution there
- Examples:
- *jmp 0x1337*

JE/JZ

- Syntax: je/jz addr
- Jump to given address only if zero flag is set to 1.
- Examples:
- *je 0x1337*
- *jz 0xf00d*

JNE/JNZ

- Syntax: jne/jnz addr
- Jump to given address only if zero flag is set to 0.
- Examples:
- *jne 0x1337*
- *jnz 0xf00d*

Other Conditional Jumps

- *Signed: js, jns, jg/jnle, jge/jnl, jl/jnge, jle/jng*

- *Unsigned: ja/jnbe, jae/jnb, jb/jnae, jbe/jna*

CALL

- Syntax: *call func*
- Push the address of the next instruction onto the stack and jump to function.
- Examples:
- *call 0x1234*

LEAVE

- Syntax: *leave*
- Moves the value of rbp into rsp and pops off the top value of th stack into rbp.

RET

- Syntax: *ret*
- Pops off the top value of the stack into rip.

Calling conventions

- When calling functions, arguments need to be placed in designated registers.
- The first argument is stored in rdi, the second in rsi, the third in rdi and so on.
- https://syscall.sh is a good reference for this.

Basics of Reverse Engineering

What is a binary

- On Linux, an executable file is known as a binaries.
- They are of a file format known as ELF (Executable Linkable Format.
- Reverse engineering mainly deals with examining and modifying binaries.

Debuggers

- Debuggers are tools that allow you to step through the execution of a binary interactively.
- They can be used to set breakpoints at different parts of the program and examine values in registers and variables.
- GDB is the standard debugger for compiled C binaries on Linux.

Disassembly

- Disassembly is the process of generating assembly code from machine code.
- Tools which receive a binary and output assembly are known as disassemblers.
- Popular disassemblers include Capstone and Hopper.
- Tools like GDB and objdump can also be used for disassembly.

Decompilation

- Decompilation is the process of converting assembly code into a higher level language, usually either C or some psuedo-code.
- Decompilation is rarely perfect.
- Popular decompilers include Hex-Rays, Snowman and RetDec.

Reverse Engineering Frameworks

- Reverse engineering frameworks are tools which combine disassembly and decompilation along with other tools such as a debugger.
- They are singular tools which can be used for the entire reverse engineering process.
- Popular RE frameworks are Ghidra, IDA, Binary Ninja and radare2.

Tools

GDB

- GDB stands for GNU debugger
- Primary debugger on most Unix systems
- It is largely used for C, C++ and Assembly programs.

Ghidra

- Ghidra is a comprehensive software reverse engineering framework developed by the USA's National Security Agency.
- It is fully free and open source.
- Among other things, it can be used for debugging, disassembly and decompilation.

Other utilities

- file – used to print information about files such as their format.
- strings – used to print all human-readable strings present in a file.
- nm – used to dump symbols from ELF binaries.
- objdump – used for inspecting object and binary files.
- We will also use a little bit of Python for basic scripting.

PESU Center for
Information Security,
Forensics and
Cyber Resilience

Let’s solve some challenges!