Binary

Reverse and Exploitation

By Kyle

Prerequisite

- C language
- Basic Linux shell command line
- Basic python codes

Outline

- C code to binary code
  - Hello World demo
- Assembly on x86
  - Overview
  - Registers
  - Common Instructions
  - Stack Structure
  - Calling Convention
  - Hello World demo explanation

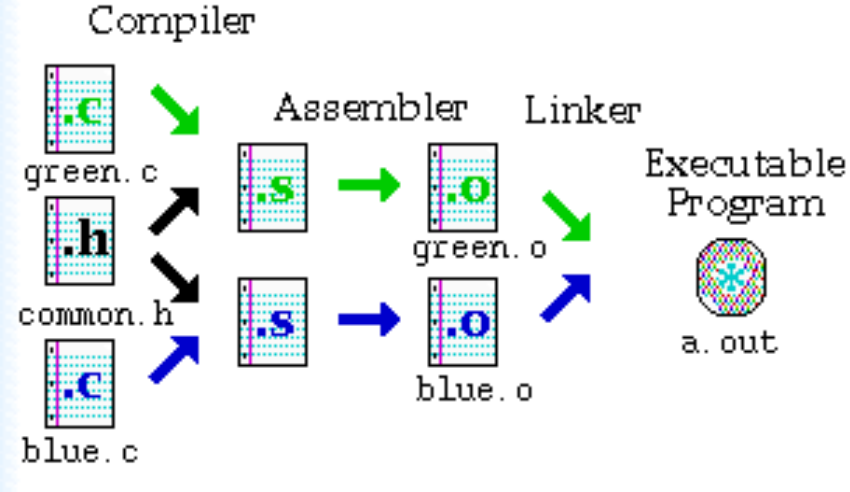
Outline--Continued

- Reverse Engineering
  - Overview
  - Tools
  - Demos
- Exploitation
  - Overview
  - Tools
  - Old School Shellcode Injection
  - ROP
  - ret2libc
  - Demos

C code to binary code

C code to binary code

- Compile?
  - Compile + Assemble + Link



  - We mainly focus on assembly and c language
  - Demo?

Assembly on x86

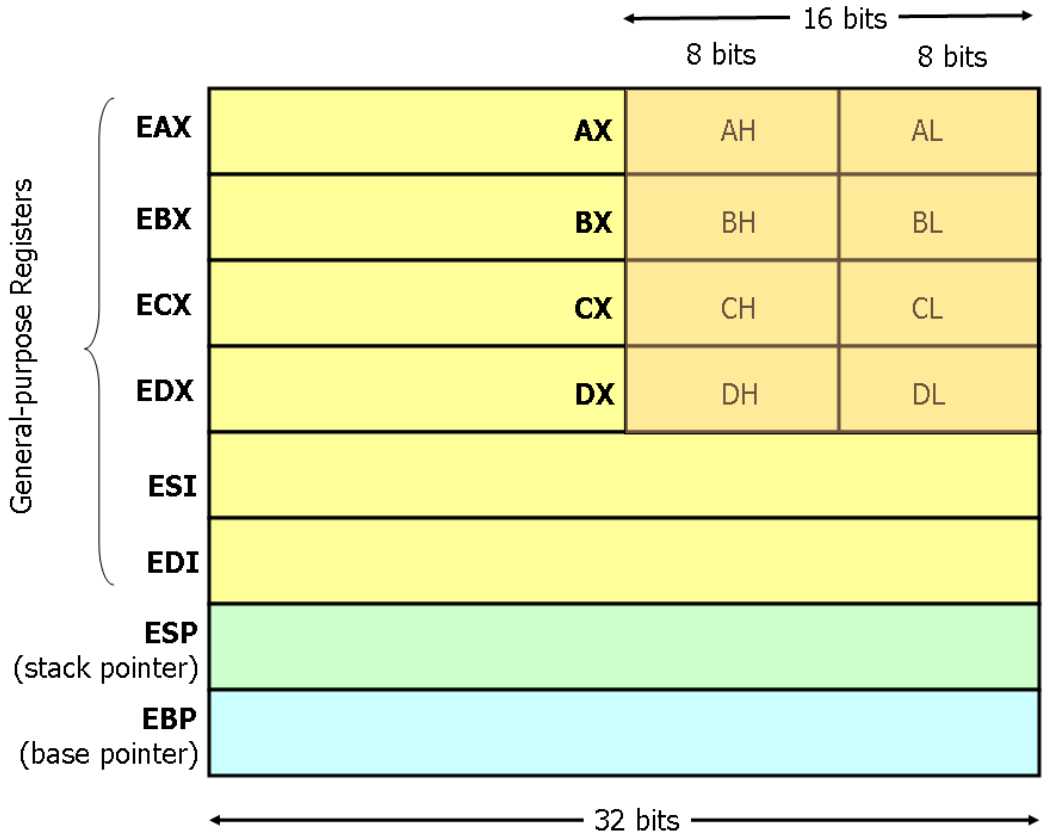
Overview

- What is assembly language – low level code for human, language for CPU
- Why assembly language – almost the same to binary code, show hidden details of a program

```
00000000        push    ebp
00000001        mov     ebp, esp
00000003        movzx   ecx, [ebp+arg_0]
00000007        pop     ebp
00000008        movzx   dx, cl
0000000C        lea     eax, [edx+edx]
0000000F        add     eax, edx
00000011        shl     eax, 2
00000014        add     eax, edx
00000016        shr     eax, 8
00000019        sub     cl, al
0000001B        shr     cl, 1
0000001D        add     al, cl
0000001F        shr     al, 5
00000022        movzx   eax, al
00000025        retn
```

Registers

- Registers are variables for CPU



Registers

- eax, ebx, ecx, edx, esi, edi: general-purpose registers(not exactly)
- esp: stack pointer
- ebp: stack base pointer

- For further information, please refer to:
- http://www.eecg.toronto.edu/~amza/www.mindsec.com/files/x86regs.html

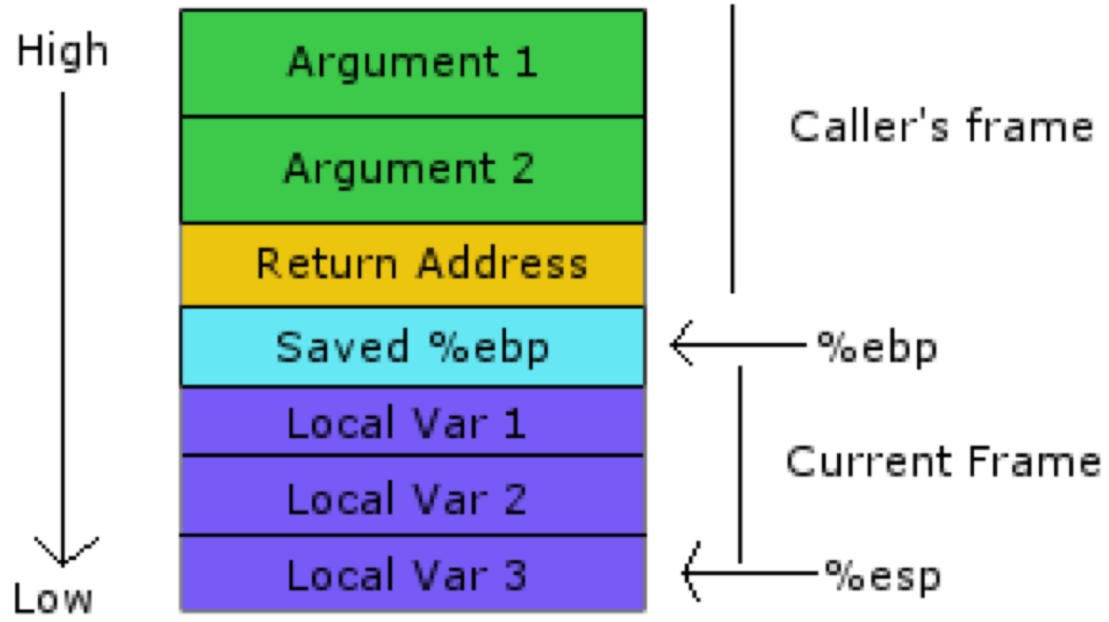
Common Instructions

- mov:
  - mov eax, ebx// ebx → eax
  - mov eax, 10// 10 → eax
- add:
  - add eax, ebx// eax+ebx → eax
- sub:
  - sub eax, ebx// eax-ebx → eax

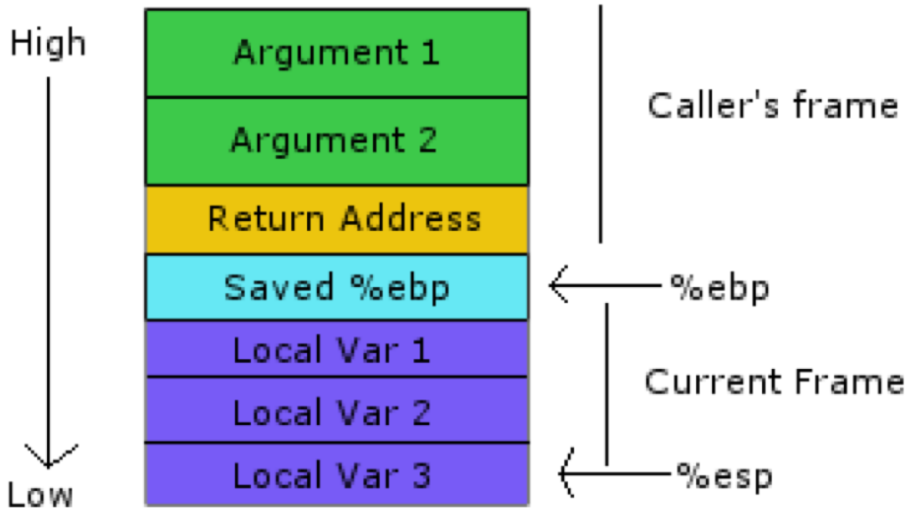
- Other similar instructions like: xor, div, mul
- Other instructions like: call, leave, ret

Stack Structure

- push eax// esp-4 → esp
- pop eax// esp+4 → esp



Calling Conventions




```
printf("%d\n", 10);
----------------------
push ebx  ;; ebx = 10
push eax  ;; eax -> "%d\n"
call printf

printf:
push ebp
mov ebp, esp
push eax
push ebx
push ecx
...
leave  ;; mov esp, ebp
       ;; pop ebp
ret
```

Hello World demo explanation

Reverse Engineering

Overview

- What is RE?
  - ~~the process of analyzing a subject system to identify the system's components and their interrelationships and to create representations of the system in another form or at a higher level of abstraction~~
  - Translate binary code into human readable code to understand the internal logic of a program
- Why RE?
  - Emmm. ~~To crack license-required software, like games.~~ To investigate malicious programs, like virus, trojan horse

Tools

- Decompiler or Disassembler
  - IDA Pro//This is the king!
  - Hopper
- Debugger
  - gdb and its derivatives: pwndbg, gef, (never ever use peda)
  - windbg
  - x64dbg
  - ollydbg
- Other
  - angr//brilliant tool

Demos

- Hopper demo
- IDA Pro demo
- angr demo
- reverse demo

Binary Exploitation

Overview

- What is binary exploitation
  - Break through some trust boundaries by passing unexpected payload to a compiled program
- Why binary exploitation
  - ~~Smash stack for fun and profit~~
  - Attack is the best defense
- Applications of binary exploitation
  - Dirtycow
  - Wannacry(eternal blue)
- Demo

Tools

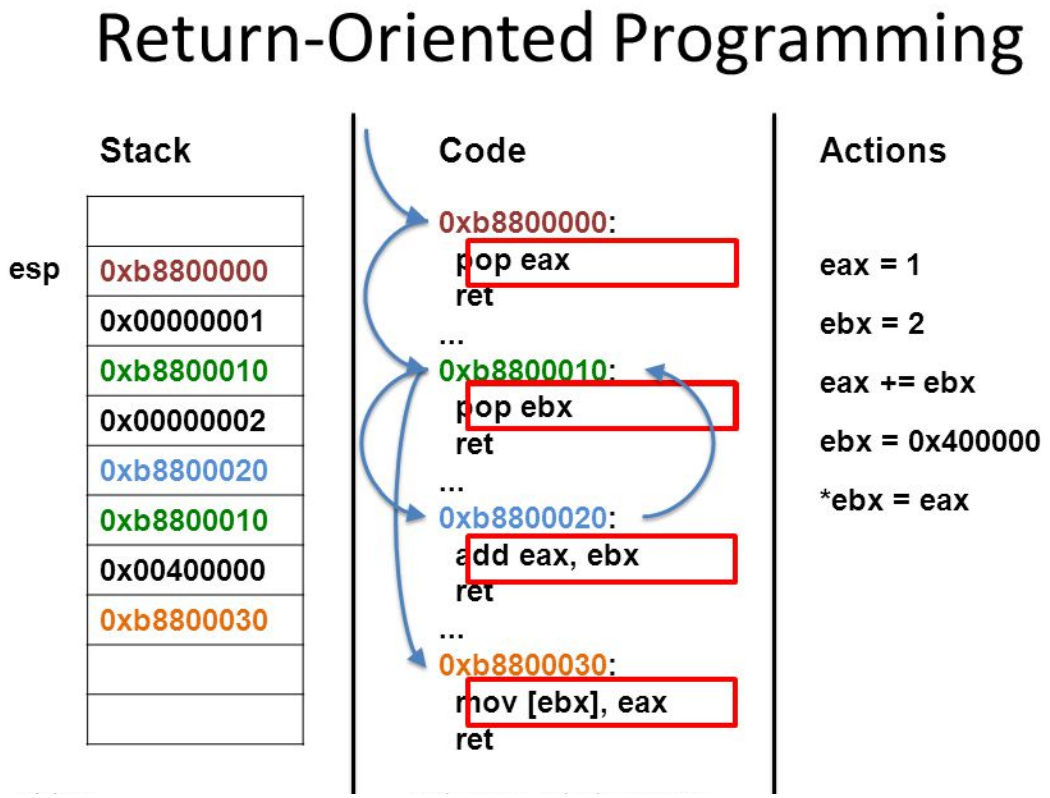
- RE tools
  - Mentioned in RE section
- Utility
  - ROPgadget
  - one_gadget
  - cyclic
  - checksec
- Framework
  - pwntools//This is the king!
  - zio
- Binary analysis
  - angr
  - radare2

Old School Shellcode Injection

- No longer seen in modern operating systems
- Defeated by introducing NX(or DEP)
- Too demanding in modern operating systems: (call user input function) || (stack leak && control flow hijacking)

- demo

ROP(Return Oriented Programming)



ret2libc(Return into libc)

- Key idea:
  - Dynamic linking
  - PLT and GOT(lazy binding)
  - libc is mapped in a whole
- libc is the best place to find gadgets
- libc is not as invulnerable as you may think
- This is how you usually use ROP

Study & Practice

- https://trailofbits.github.io/ctf/exploits/binary1.html
- https://github.com/shellphish/how2heap
- https://github.com/Kyle-Kyle/Pwn

Q&A

Thank you