Reverse Engineering

A UMass Cybersecurity Club Workshop

UMASS CYBERSEC CLUB

Census

https://forms.gle/ffWgsHyfCyPn5tiw6

UMASS CYBERSEC CLUB

What is Reversing Engineering?

- Understanding how a...
  - device
  - process
  - system
  - ...or software
  - *accomplishes* a task with *reduced insight*
- We figure out how it works by opening it up and *dissecting* it

UMASS CYBERSEC CLUB

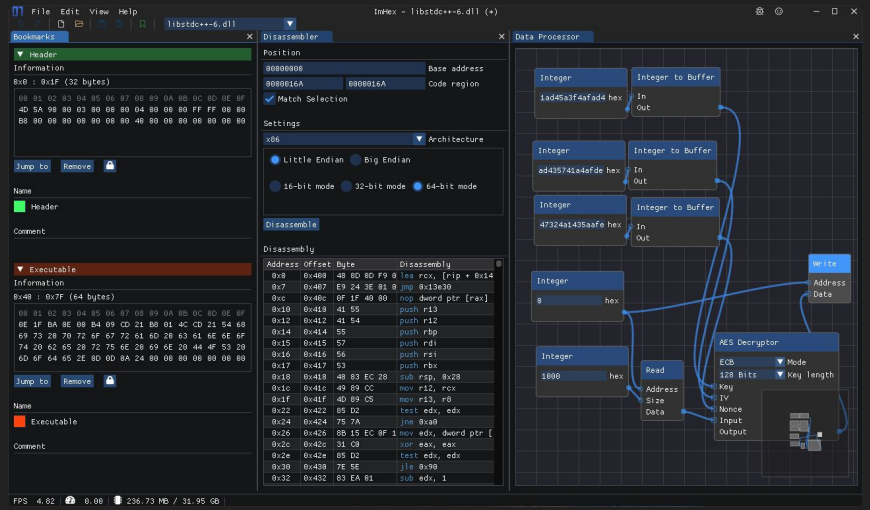
Why Would I Want to Reverse Engineer Something?

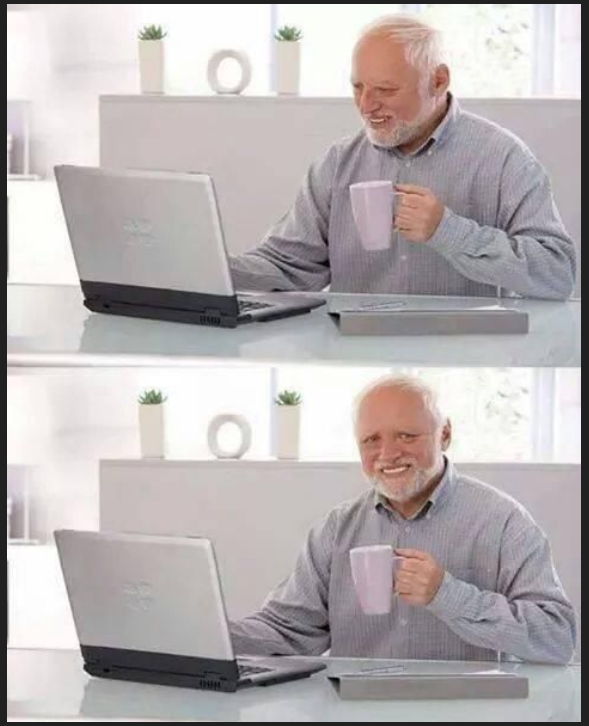
- We can use Reverse Engineering to *ethically* find out the following:
  - Password checks on an executable
  - How does your favorite videogame implement all of its features (and how can you mod it)
- On the corporate side:
  - Did another company illegally use our patented code?!

UMASS CYBERSEC CLUB

How Do I Reverse Engineer a Software?

1. Get the binary file
2. Disassemble it
3. Analyze it



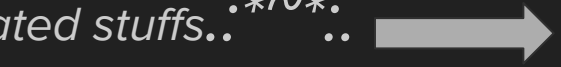


UMASS CYBERSEC CLUB

Huh?

UMASS CYBERSEC CLUB

How is Software Made?

1. Write code (inside text editor/IDE)
2. Build it with ..:*~*:.. *Complicated stuffs* ..:*~*:..  **Compilation**
3. Get an executable
4. Debug/Test
5. Release

Reverse Engineering Strategies

- **Static analysis**
  - Examining assembly code
  - Use disassembler/decompiler on binary
- **Dynamic analysis**
  - Introspecting at run-time
  - Attach debugger when running
  - Read memory of an in-execution process

How To Make Software: Compiler view

- Translate source code to machine instructions
  - Runs directly with your operating system or on hardware
    - Intel, AMD: **x86-64**
    - Qualcomm, Apple: ARM
- Write instructions into binary file

UMASS CYBERSEC CLUB

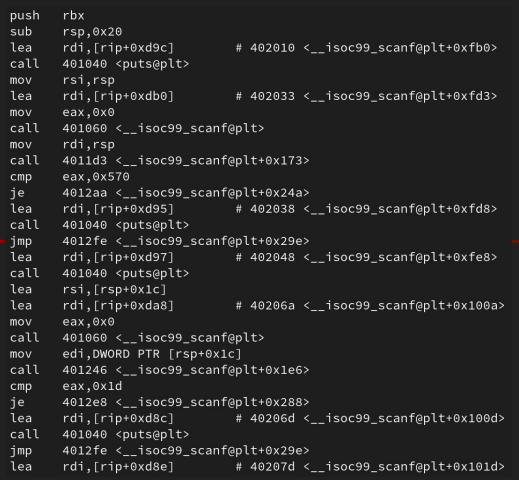
Demo: godbolt.org

Turn and talk to the people around you...

- Think of a simple program (any language) that does arithmetic (addition, subtraction, multiplication, etc...) operations
- Explain how you would write code that achieves this to the people at your table

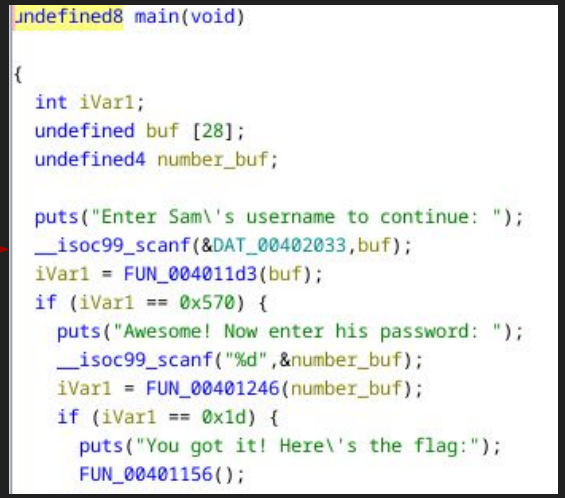
How to Reverse Software: Decompiler View

Binary File (Machine Code)



```
push   rbx
sub    rsp,0x20
lea    rdi,[rip+0xd9c]        # 402010 <__isoc99_scanf@plt+0xfb0>
call   401040 <puts@plt>
mov    rsi,rsp
lea    rdi,[rip+0xdb0]        # 402033 <__isoc99_scanf@plt+0xfd3>
mov    eax,0x0
call   401060 <__isoc99_scanf@plt>
mov    rdi,rsp
call   4011d3 <__isoc99_scanf@plt+0x173>
cmp    eax,0x570
je     4012aa <__isoc99_scanf@plt+0x24a>
lea    rdi,[rip+0xd95]        # 402038 <__isoc99_scanf@plt+0xfd8>
call   401040 <puts@plt>
jmp    4012fe <__isoc99_scanf@plt+0x29e>
lea    rdi,[rip+0xd97]        # 402048 <__isoc99_scanf@plt+0xfe8>
call   401040 <puts@plt>
lea    rsi,[rsp+0x1c]
lea    rdi,[rip+0xda8]        # 40206a <__isoc99_scanf@plt+0x100a>
mov    eax,0x0
call   401060 <__isoc99_scanf@plt>
mov    edi,DWORD PTR [rsp+0x1c]
call   401246 <__isoc99_scanf@plt+0x1e6>
cmp    eax,0x1d
je     4012e8 <__isoc99_scanf@plt+0x288>
lea    rdi,[rip+0xd8c]        # 40206d <__isoc99_scanf@plt+0x100d>
call   401040 <puts@plt>
jmp    4012fe <__isoc99_scanf@plt+0x29e>
lea    rdi,[rip+0xd8e]        # 40207d <__isoc99_scanf@plt+0x101d>
```

Assembly Code



```
undefined8 main(void)

{
  int iVar1;
  undefined buf [28];
  undefined4 number_buf;

  puts("Enter Sam\'s username to continue: ");
  __isoc99_scanf(&DAT_00402033,buf);
  iVar1 = FUN_004011d3(buf);
  if (iVar1 == 0x570) {
    puts("Awesome! Now enter his password: ");
    __isoc99_scanf("%d",&number_buf);
    iVar1 = FUN_00401246(number_buf);
    if (iVar1 == 0x1d) {
      puts("You got it! Here\'s the flag:");
      FUN_00401156();
```

Decompiled “Source”

UMASS CYBERSEC CLUB

What is a binary file?

- Binary file is a sequence of 0s and 1s encoded in a file containing executable “machine code”
    - This type of file is called an ELF file
    - Similar to EXE files on Windows

Instruction Set Architecture

- Defines a set of instructions and their machine code encoding to work with specific hardware design
- CISC: x86, RISC: ARM

UMASS CYBERSEC CLUB

How to Reverse Software: Decompiler View

```
e70f 01f8 0fb6 c0c3 5348 83ec 2048 8d3d
9c0d 0000 e8c7 fdff ff48 89e6 488d 3db0
0d00 00b8 0000 0000 e8d3 fdff ff48 89e7
e83e ffff ff3d 7005 0000 740e 488d 3d95
0d00 00e8 98fd ffff eb54 488d 3d97 0d00
00e8 8afd ffff 488d 7424 1c48 8d3d a80d
0000 b800 0000 00e8 94fd ffff 8b7c 241c
e871 ffff ff83 f81d 740e 488d 3d8c 0d00
00e8 5afd ffff eb16 488d 3d8e 0d00 00e8
4cfd ffff b800 0000 00e8 58fe ffff b800
0000 0048 83c4 205b c300 0000 f30f 1efa
4883 ec08 4883 c408 c300 0000 0000 0000
```

```
push   rbx
sub    rsp,0x20
lea    rdi,[rip+0xd9c]        # 402010 <__isoc99_scanf@plt+0xfb0>
call   401040 <puts@plt>
mov    rsi,rsp
lea    rdi,[rip+0xdb0]        # 402033 <__isoc99_scanf@plt+0xfd3>
mov    eax,0x0
call   401060 <__isoc99_scanf@plt>
mov    rdi,rsp
call   4011d3 <__isoc99_scanf@plt+0x173>
cmp    eax,0x570
je     4012aa <__isoc99_scanf@plt+0x24a>
lea    rdi,[rip+0xd95]        # 402038 <__isoc99_scanf@plt+0xfd8>
call   401040 <puts@plt>
jmp    4012fe <__isoc99_scanf@plt+0x29e>
lea    rdi,[rip+0xd97]        # 402048 <__isoc99_scanf@plt+0xfe8>
call   401040 <puts@plt>
lea    rsi,[rsp+0x1c]
lea    rdi,[rip+0xda8]        # 40206a <__isoc99_scanf@plt+0x100a>
mov    eax,0x0
call   401060 <__isoc99_scanf@plt>
mov    edi,DWORD PTR [rsp+0x1c]
call   401246 <__isoc99_scanf@plt+0x1e6>
cmp    eax,0x1d
je     4012e8 <__isoc99_scanf@plt+0x288>
lea    rdi,[rip+0xd8c]        # 40206d <__isoc99_scanf@plt+0x100d>
call   401040 <puts@plt>
jmp    4012fe <__isoc99_scanf@plt+0x29e>
lea    rdi,[rip+0xd8e]        # 40207d <__isoc99_scanf@plt+0x101d>
```

```
undefined8 main(void)

{
  int iVar1;
  undefined buf [28];
  undefined4 number_buf;

  puts("Enter Sam\'s username to continue: ");
  __isoc99_scanf(&DAT_00402033,buf);
  iVar1 = FUN_004011d3(buf);
  if (iVar1 == 0x570) {
    puts("Awesome! Now enter his password: ");
    __isoc99_scanf("%d",&number_buf);
    iVar1 = FUN_00401246(number_buf);
    if (iVar1 == 0x1d) {
      puts("You got it! Here\'s the flag:");
      FUN_00401156();
```

Binary File (Machine Code)

Assembly Code

Decompiled “Source”

UMASS CYBERSEC CLUB

What is assembly?

- The lowest level *human readable* form of machine instructions
- What does it do?
  - Arithmetic operations:
    - +, -, *, /
  - Memory operations:
    - mov data around
  - Control program flow: **Jump**
    - Jmp to a different location
    - If-else, for while loops

```
sum:
        mov     rdx, rdi
        mov     rax, rsi
        add     rax, rdx
        ret
square:
        mov     edx, edi
        mov     eax, esi
        imul    eax, edx
        ret
```

UMASS CYBERSEC CLUB

Sidenote: AT&T vs. Intel Syntax

AT&T

inst source, destination

```
movl $1, %ecx

movl 3(%eax), %ebx
```

Intel

inst destination, source

```
mov ecx, 1

mov ebx, [eax + 3]
```

CS230 uses AT&T, we use Intel

UMASS CYBERSEC CLUB

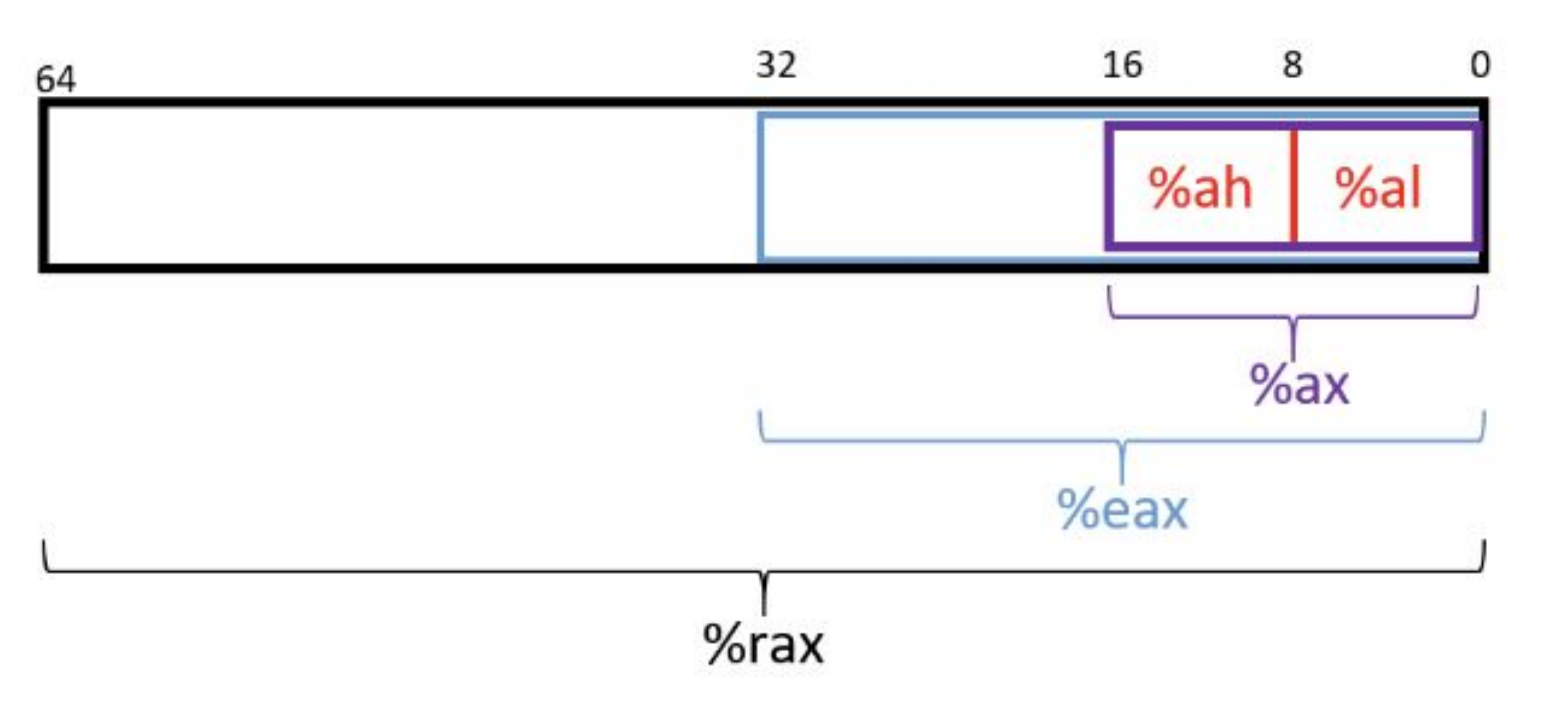
Assembly (x86-64) - Registers

- Registers store values, they are stored physically closest to the CPU so they are fast to work with
- They are required for most assembly instructions
- In x86, there are 6 commonly used registers, 3 special registers and other less commonly used ones eg: r8-r15

Size (in Bits)			
64	32	16	8
RAX	EAX	AX	AH/AL
RBX	EBX	BX	BH/BL
RCX	ECX	CX	CH/CL
RDX	EDX	DX	DH/DL
RDI	EDI	DI	DIL
RSI	ESI	SI	SIL
RBP	EBP	BP	BPL
RSP	ESP	SP	SPL
R8∼R15	R8D∼R15D	R8W∼R15W	R8L∼R15L

UMASS CYBERSEC CLUB

Assembly (x86-64) - Register sizes



UMASS CYBERSEC CLUB

Assembly (x86) - Basic Instructions

- **Perform** operations that we do on registers, values, and memory addresses
- There are thousands of instructions; no need to memorize all of them
  - Intel x86 documentation is >5000 pages long.

Instruction Name	p1	p2	Description
add/sub/imul	register	register OR value	Adds/subtracts/multiplies the two values stored in registers together and stores the solution into the first register
mov	register	register OR value	Moves the value stored in right register into the left register

Assembly (x86) - Basic Instructions

```
mov eax, 1          <=

mov ebx, 3

add ebx, eax

xor eax, eax
```

UMASS CYBERSEC CLUB

Assembly (x86) - Basic Instructions

```
mov eax, 1

mov ebx, 3          <= eax = 1

add ebx, eax

xor eax, eax
```

UMASS CYBERSEC CLUB

Assembly (x86) - Basic Instructions

```
mov eax, 1

mov ebx, 3

add ebx, eax        <= eax = 1, ebx = 3

xor eax, eax
```

UMASS CYBERSEC CLUB

Assembly (x86) - Basic Instructions

```
mov eax, 1

mov ebx, 3

add ebx, eax

xor eax, eax        <= eax = ?, ebx = ?
```

UMASS CYBERSEC CLUB

Assembly (x86) - Basic Instructions

```
mov eax, 1

mov ebx, 3

add ebx, eax

xor eax, eax        <= eax = 1, ebx = 4
```

UMASS CYBERSEC CLUB

Assembly (x86) - Basic Instructions

```
mov eax, 1

mov ebx, 3

add ebx, eax

xor eax, eax
```

<= eax = ?

UMASS CYBERSEC CLUB

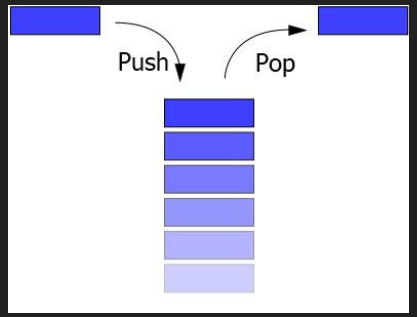
Turn and talk to the people around you.

- Write a function in C that will output assembly that uses all of add, sub, and imul instructions
- Put that code into https://godbolt.org/ and see if it outputs what you think
- Did it work? Discuss why or why not.

UMASS CYBERSEC CLUB

Assembly (x86) - Stack

- **Stack:** The location where a program stores memory
- To make space on the stack we subtract



Instruction Name	p1	p2	Description
push	register		Add new value to the “top” of the stack
pop	register		Pop the “top” of the stack and put the value in the register

UMASS CYBERSEC CLUB

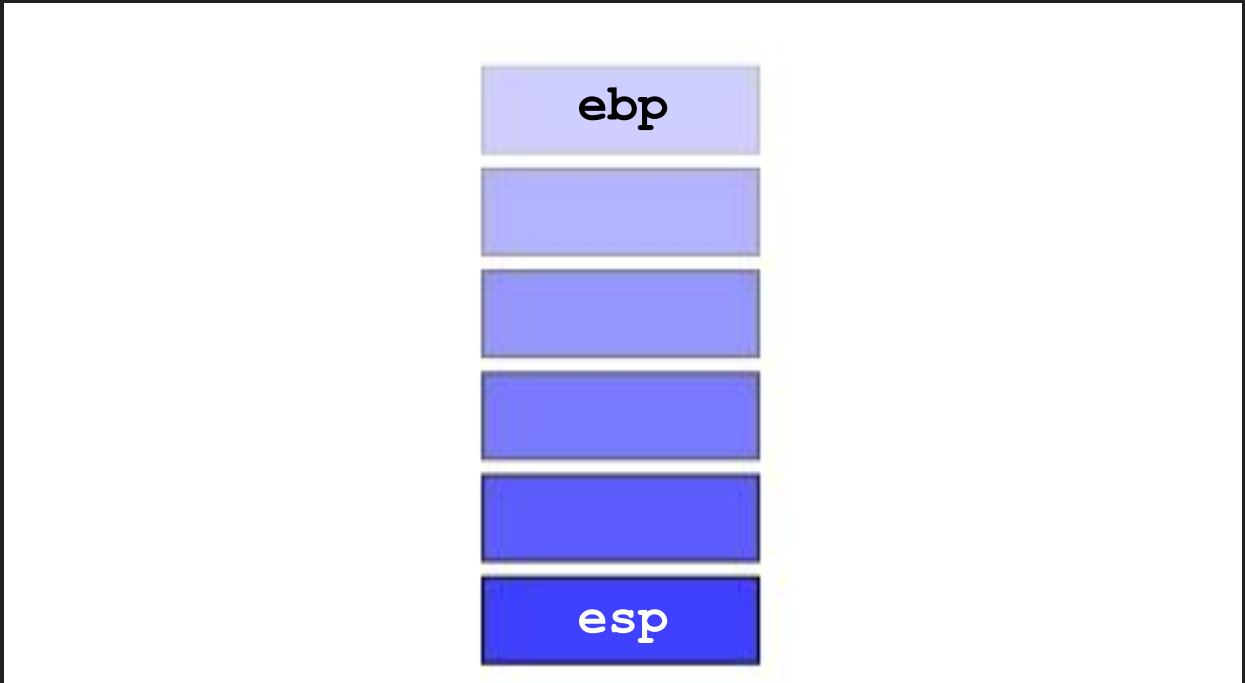
Assembly (x86) - Stack

`push eax`	`sub esp, 4` `mov DWORD [esp], eax`

`pop eax`	`mov eax, DWORD [esp]` `add esp, 4`

Assembly (x86) - Stack

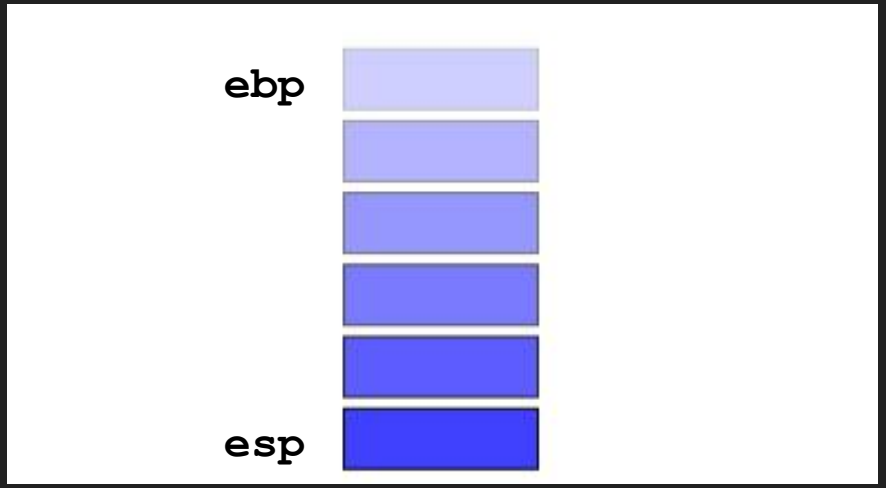
0xFFFFFFFF



0x00000000

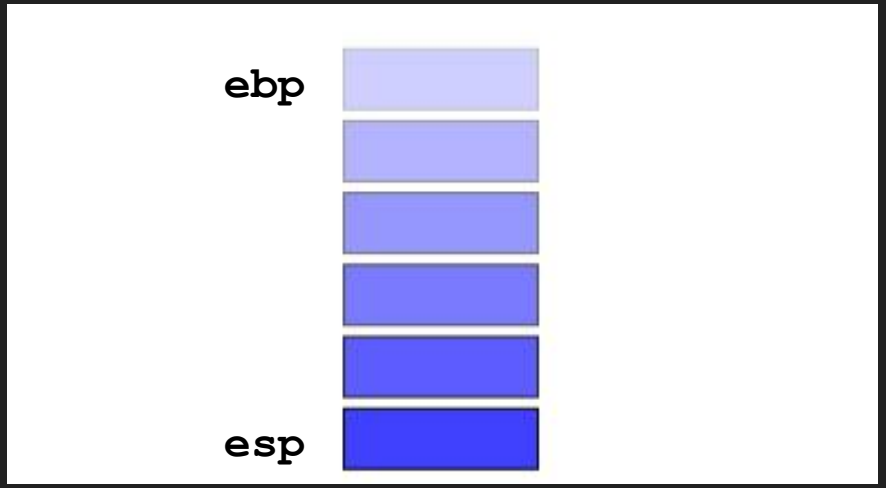
Assembly (x86) - Stack

```
mov eax, 1      <=
push eax
pop ebx
sub eax, ebx
```



Assembly (x86) - Stack

```
mov eax, 1
push eax        <= eax = 1
pop ebx
sub eax, ebx
```



UMASS CYBERSEC CLUB

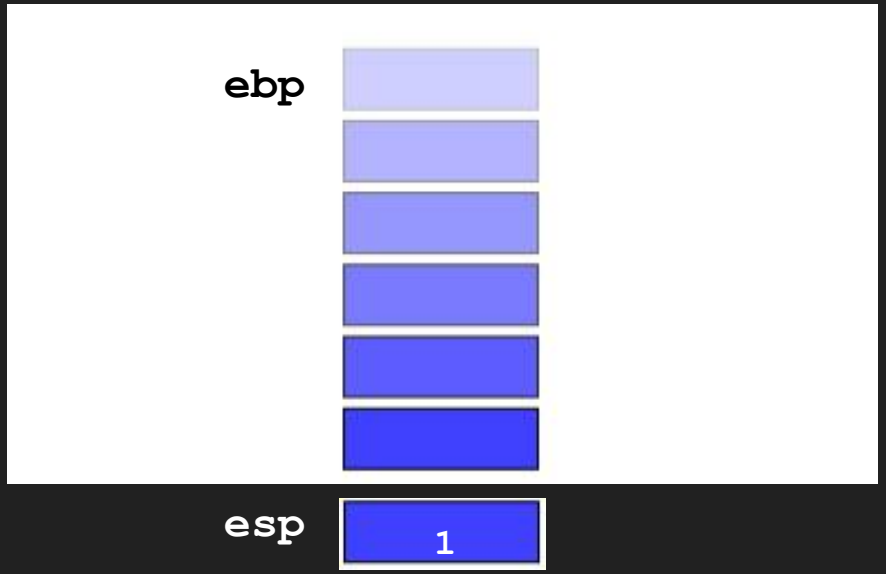
Assembly (x86) - Stack

```
mov eax, 1

push eax

pop ebx          <=

sub eax, ebx
```



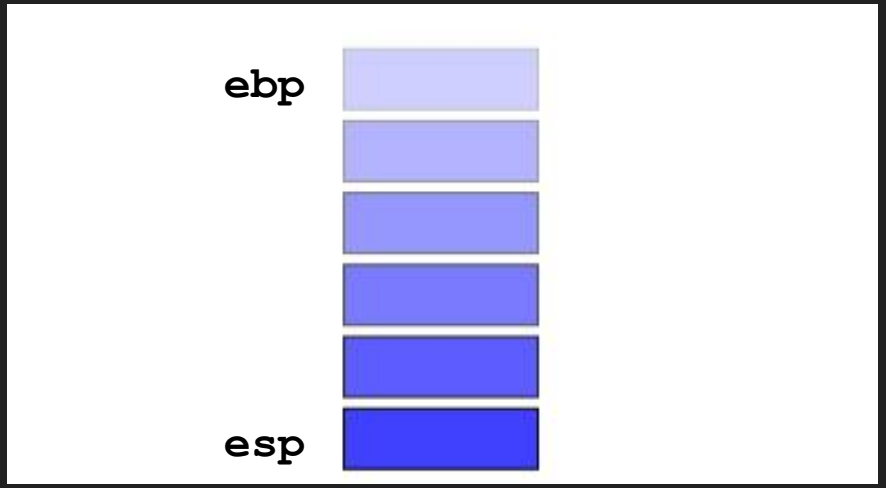
Assembly (x86) - Stack

```
mov eax, 1

push eax

pop ebx

sub eax, ebx  <= ebx = 1
```



Assembly (x86) - Stack

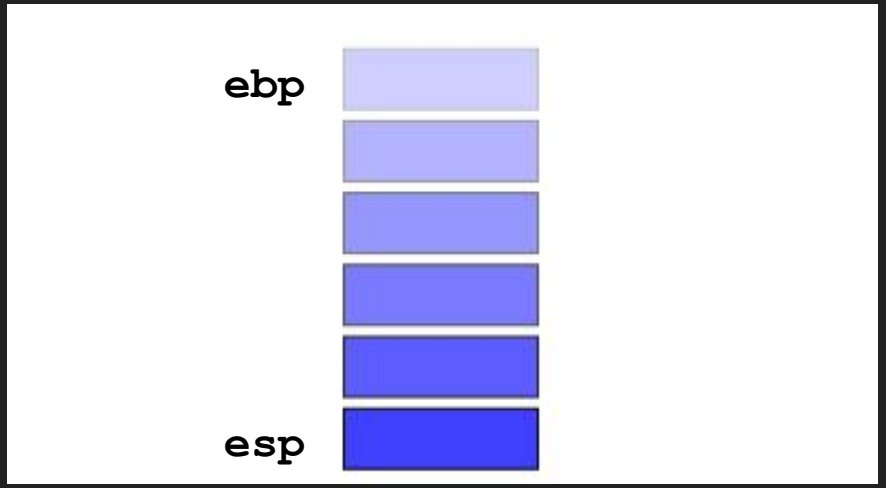
```
mov eax, 1

push eax

pop ebx

sub eax, ebx
```

<= eax = 0



Assembly (x86) - Control Flow

- Instructions can also be used to control the flow of the program
- **Label:** A section of assembly code with an identifier so we can keep track of it

Instruction Name	p1	p2	Description
jmp/jle/jge	label		Goes to and begins executing code from wherever the label is either unconditionally or a condition
cmp	register	register OR value	Compares two registers or a registers and a value, usually following by a jump conditional which uses the output of this instruction

Turn and talk to the people around you.

- Think of a simple program in C that will use control flow with labels
- Put that code into https://godbolt.org/ and see if it outputs what you think
- Did it work? Discuss why or why not.

UMASS CYBERSEC CLUB

Assembly (x86-64) - Calling Convention

Standard calling convention when calling a function in C, on Linux:

- rax: used as return value
- rdi: first parameter
- rsi: second parameter
- rdx: third parameter

https://syscalls.mebeim.net/

```
int sum(int a, int b) {
    return a+b;
}

sum:
    mov     rdx, rdi
    mov     rax, rsi
    add     rax, rdx
    ret
```

UMASS CYBERSEC CLUB

How does a program output to terminal?

In general, libc functions from stdio.h such as:

- printf, puts, putchar

However, we can also use:

- write (syscall)

Writing to where?

Standard output: file descriptor 1

```
; ------------------------------------------------------------------------
; Writes "Hello, World" to the console using only system calls. Runs on 64-bit Linux only.
; To assemble and run:
;
;     nasm -felf64 hello.asm && ld hello.o && ./a.out
; ------------------------------------------------------------------------

          global    _start

          section   .text
_start:   mov       rax, 1                ; system call for write
          mov       rdi, 1                ; file handle 1 is stdout
          mov       rsi, message          ; address of string to output
          mov       rdx, 13               ; number of bytes
          syscall                         ; invoke operating system to do the write
          mov       rax, 60               ; system call for exit
          xor       rdi, rdi              ; exit code 0
          syscall                         ; invoke operating system to exit

          section   .data
message:  db        "Hello, World", 10      ; note the newline at the end
```

UMASS CYBERSEC CLUB

Dive into our Challenge

Download from training platform, rev-basic, rev-stack

Use **ssh hacker@34.75.164.48 if you do not have Linux VM**

Password is “revf2024”

Use **gdb <filename>** to launch gdb debugger

Then type **break main** to stop at the main function

Type **run <args>** to start the program

UMASS CYBERSEC CLUB

How to Reverse Software: Decompiler View

```
e70f 01f8 0fb6 c0c3 5348 83ec 2048 8d3d
9c0d 0000 e8c7 fdff ff48 89e6 488d 3db0
0d00 00b8 0000 0000 e8d3 fdff ff48 89e7
e83e ffff ff3d 7005 0000 740e 488d 3d95
0d00 00e8 98fd ffff eb54 488d 3d97 0d00
00e8 8afd ffff 488d 7424 1c48 8d3d a80d
0000 b800 0000 00e8 94fd ffff 8b7c 241c
e871 ffff ff83 f81d 740e 488d 3d8c 0d00
00e8 5afd ffff eb16 488d 3d8e 0d00 00e8
4cfd ffff b800 0000 00e8 58fe ffff b800
0000 0048 83c4 205b c300 0000 f30f 1efa
4883 ec08 4883 c408 c300 0000 0000 0000
```

```
push   rbx
sub    rsp,0x20
lea    rdi,[rip+0xd9c]        # 402010 <__isoc99_scanf@plt+0xfb0>
call   401040 <puts@plt>
mov    rsi,rsp
lea    rdi,[rip+0xdb0]        # 402033 <__isoc99_scanf@plt+0xfd3>
mov    eax,0x0
call   401060 <__isoc99_scanf@plt>
mov    rdi,rsp
call   4011d3 <__isoc99_scanf@plt+0x173>
cmp    eax,0x570
je     4012aa <__isoc99_scanf@plt+0x24a>
lea    rdi,[rip+0xd95]        # 402038 <__isoc99_scanf@plt+0xfd8>
call   401040 <puts@plt>
jmp    4012fe <__isoc99_scanf@plt+0x29e>
lea    rdi,[rip+0xd97]        # 402048 <__isoc99_scanf@plt+0xfe8>
call   401040 <puts@plt>
lea    rsi,[rsp+0x1c]
lea    rdi,[rip+0xda8]        # 40206a <__isoc99_scanf@plt+0x100a>
mov    eax,0x0
call   401060 <__isoc99_scanf@plt>
mov    edi,DWORD PTR [rsp+0x1c]
call   401246 <__isoc99_scanf@plt+0x1e6>
cmp    eax,0x1d
je     4012e8 <__isoc99_scanf@plt+0x288>
lea    rdi,[rip+0xd8c]        # 40206d <__isoc99_scanf@plt+0x100d>
call   401040 <puts@plt>
jmp    4012fe <__isoc99_scanf@plt+0x29e>
lea    rdi,[rip+0xd8e]        # 40207d <__isoc99_scanf@plt+0x101d>
```

```
undefined8 main(void)

{
  int iVar1;
  undefined buf [28];
  undefined4 number_buf;

  puts("Enter Sam\'s username to continue: ");
  __isoc99_scanf(&DAT_00402033,buf);
  iVar1 = FUN_004011d3(buf);
  if (iVar1 == 0x570) {
    puts("Awesome! Now enter his password: ");
    __isoc99_scanf("%d",&number_buf);
    iVar1 = FUN_00401246(number_buf);
    if (iVar1 == 0x1d) {
      puts("You got it! Here\'s the flag:");
      FUN_00401156();
```

Binary File (Machine Code)

Assembly Code

Decompiled “Source”

UMASS CYBERSEC CLUB

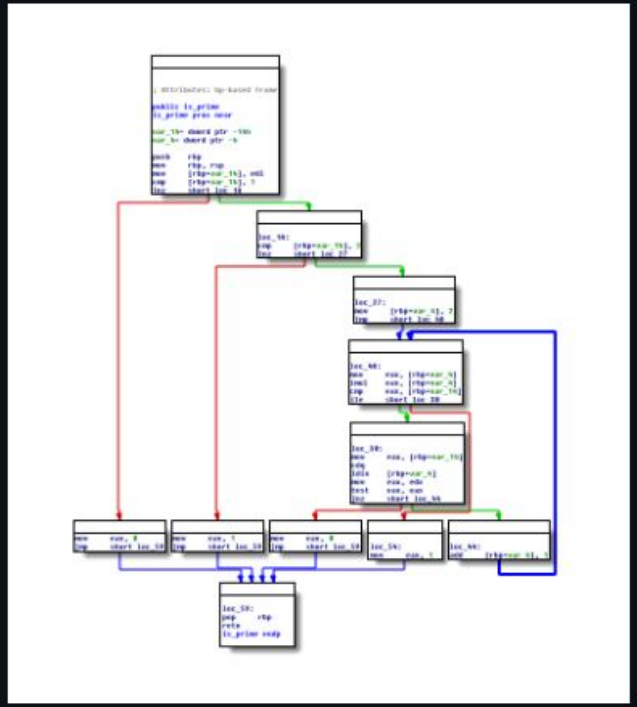
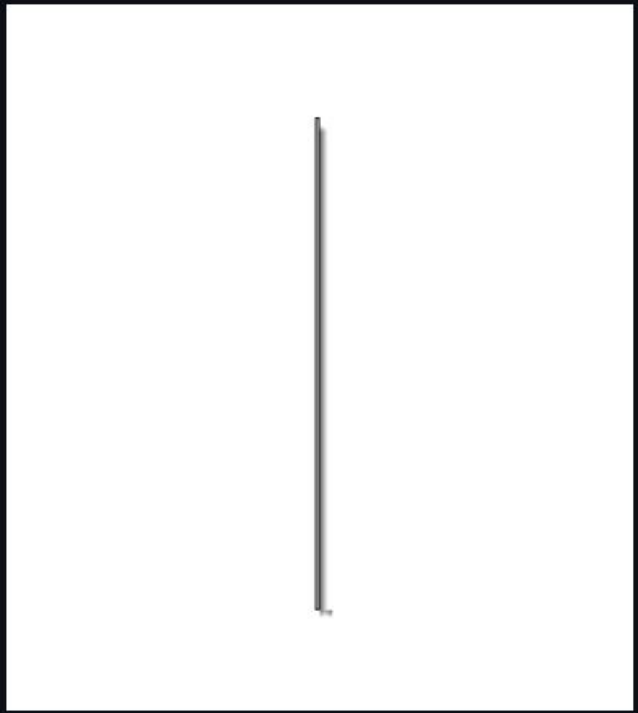
Tools

- **Disassembler**: *Translate* binary into assembly code
  - objdump
  - X86dbg
  - GDB
  - radare2
  - Rizin
  - ...
- **Decompiler**: *Guess* assembly code in higher-level code
  - Ghidra
  - IDA
  - Binary Ninja
  - Snowman
  - RetDec

UMASS CYBERSEC CLUB

Demo: dogbolt.org

Anti-reversing Measures

GCC	M/o/Vfuscator
	

UMASS CYBERSEC CLUB

What's next?

- This Friday we'll learn more about decompilation process and other tools
- Some challenges are hosted on the training platform for some great practice!

https://training.umasscybersec.org

UMASS CYBERSEC CLUB

Intro to Ghidra

Ghidra is a reverse engineering tool developed by the National Security Agency. It is now available for free as an open-source software that is used by security researchers.

Download Ghidra from https://ghidra-sre.org/

You may choose to use Ghidra on your host OS, however, it is recommended to use on Linux since you can run the binary easily

ELF and libc

ELF stands for Executable Linkable Format

Libc, as in “standard C library” is what provides all the functions we use

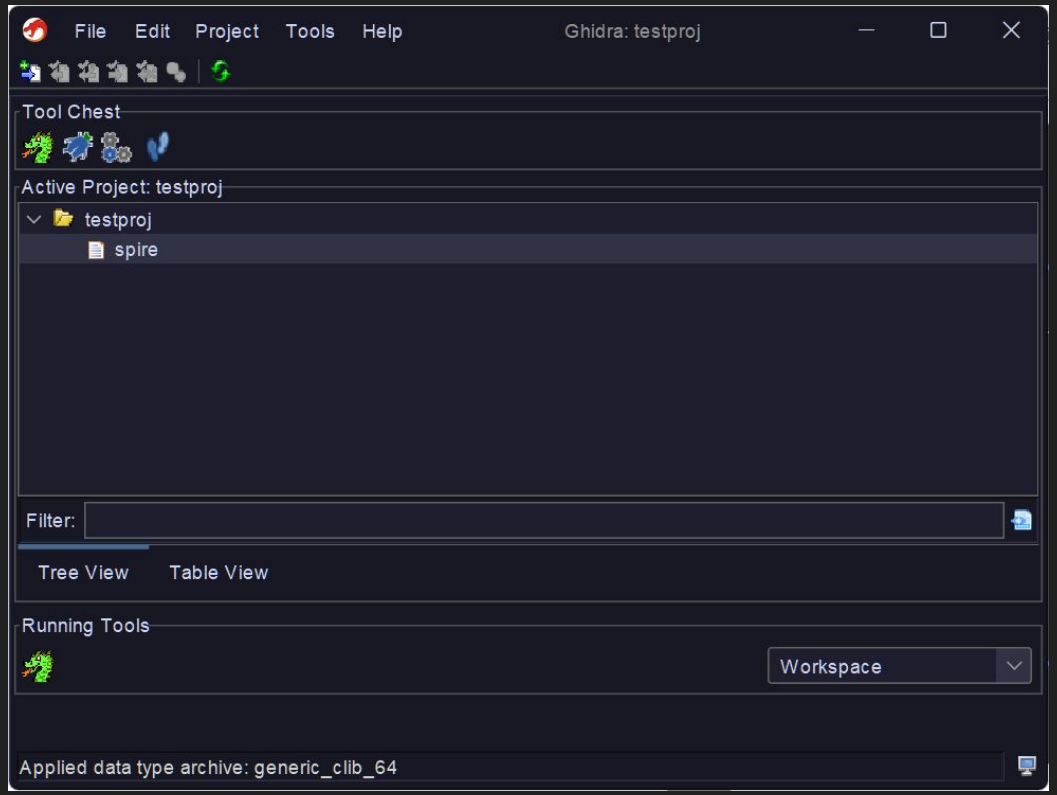
Stack vs Heap

YAP ABOUT STACK

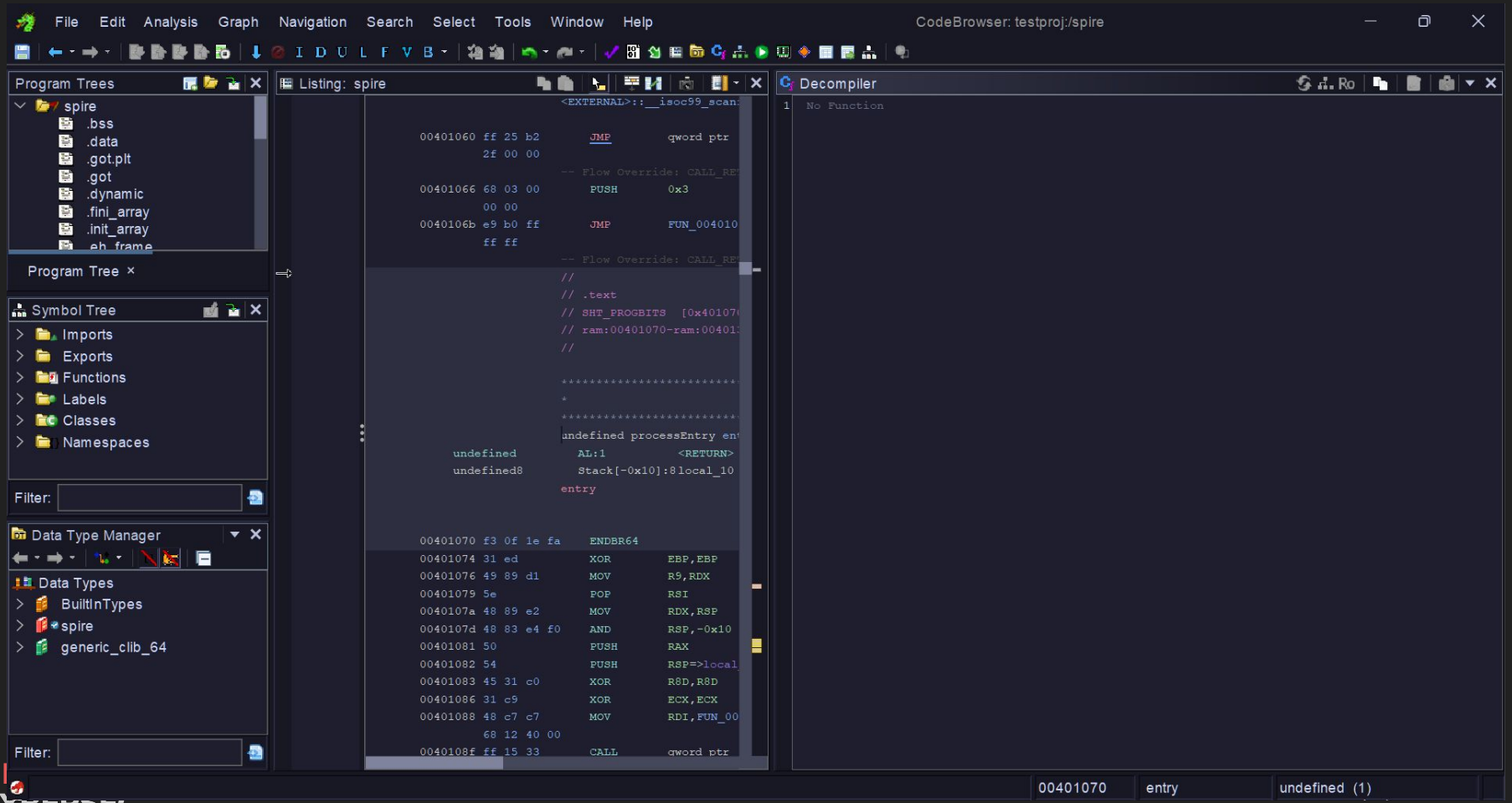
YAP ABOUT MALLOC

Digging into spire-but-ez

- Download the binary from https://training.umasscybersec.org
- Create a new project, import the binary, open in CodeBrowser (the dragon)



UMASS CYBERSEC CLUB


File Edit Analysis Graph Navigation Search Select Tools Window Help
CodeBrowser: testproj:/spire
Program Trees
spire
.bss
.data
.got.plt
.got
.dynamic
.fini_array
.init_array
.eh_frame
Program Tree ×
Symbol Tree
Imports
Exports
Functions
Labels
Classes
Namespaces
Filter:
Data Type Manager
Data Types
BuiltInTypes
spire
generic_clib_64
Filter:
Listing: spire
<EXTERNAL>::__isoc99_scan
00401060 ff 25 b2 2f 00 00 JMP qword ptr
-- Flow Override: CALL_RE
00401066 68 03 00 00 00 PUSH 0x3
0040106b e9 b0 ff ff ff JMP FUN_004010
-- Flow Override: CALL_RE
//
// .text
// SHT_PROGBITS [0x401070
// ram:00401070-ram:00401
//
undefined processEntry en
undefined AL:1 <RETURN>
undefined8 Stack[-0x10]:8 local_10
entry
00401070 f3 0f 1e fa ENDBR64
00401074 31 ed XOR EBP,EBP
00401076 49 89 d1 MOV R9,RDX
00401079 5e POP RSI
0040107a 48 89 e2 MOV RDX,RSP
0040107d 48 83 e4 f0 AND RSP,-0x10
00401081 50 PUSH RAX
00401082 54 PUSH RSP=>local
00401083 45 31 c0 XOR R8D,R8D
00401086 31 c9 XOR ECX,ECX
00401088 48 c7 c7 68 12 40 00 MOV RDI,FUN_00
0040108f ff 15 33 CALL qword ptr
Decompiler
1 No Function
00401070
entry
undefined (1)

Navigate to the left, Symbol Tree, Functions

Look for `entry`

This is the entry point of the program, which is where it starts executing

```
1
2 void processEntry entry(undefined8 param_1,undefined8 param_2)
3
4 {
5   undefined auStack_8 [8];
6
7   __libc_start_main(FUN_00401268,param_2,&stack0x00000008,0,0,param_1,auStack_8);
8   do {
9                     /* WARNING: Do nothing block with infinite loop */
10  } while( true );
11 }
```

Mouse on fun_00401268, press l, rename to main

UMASS CYBERSEC CLUB

```
1
2 undefined8 main(void)
3
4 {
5   int iVar1;
6   undefined auStack_28 [28];
7   undefined4 local_c;
8
9   puts("Enter Sam\'s username to continue: ");
10  __isoc99_scanf(&DAT_00402033,auStack_28);
11  iVar1 = FUN_004011d3(auStack_28);
12  if (iVar1 == 0x570) {
13    puts("Awesome! Now enter his password: ");
14    __isoc99_scanf(&DAT_0040206a,&local_c);
15    iVar1 = FUN_00401246(local_c);
16    if (iVar1 == 0x1d) {
17      puts("You got it! Here\'s the flag:");
18      FUN_00401156();
19    }
20    else {
21      puts("Wrong password!");
22    }
23  }
24  else {
25    puts("Wrong username!");
26  }
27  return 0;
28 }
29
```

Go thru function for username and password: 10 mins each

```
1
2 int FUN_004011d3(byte *param_1)
3
4 {
5   return (param_1[0xc] & 0x6c) +
6          (param_1[0xb] & 0x6f) +
7          (param_1[10] & 0x6f) +
8          (param_1[9] & 99) +
9          (param_1[8] & 0x5f) +
10         (param_1[7] & 0x73) +
11         (param_1[6] & 0x69) +
12         (param_1[5] & 0x5f) +
13         (param_1[4] & 0x73) +
14         (param_1[3] & 0x73) + (param_1[2] & 0x61) + (*param_1 & 0x75) + (param_1[1] & 0x6d);
15 }
16
```

UMASS CYBERSEC CLUB

```
1
2 int FUN_00401246(uint param_1)
3
4 {
5   return (param_1 >> 8 & 0xf) + (param_1 & 0xf) + (param_1 >> 4 & 0xf) + (param_1 >> 0xc & 0xf);
6 }
7
```

UMASS CYBERSEC CLUB