## Cyber@UC Meeting 85

Battelle goat challenge/IDA

#### If You're New!

- Join our Slack: <u>cyberatuc.slack.com</u>
- Check out our website: <u>cyberatuc.org</u>
- Organization Resources on our Wiki: <u>wiki.cyberatuc.org</u>
- **SIGN IN!** (Slackbot will post the link in #general every Wed@6:30)
- Feel free to get involved with one of our committees:
  - Content Finance Public Affairs Outreach Recruitment Lab
- Ongoing work in our research lab!



#### Announcements

- Looking for lab committee volunteers!
- New bi-weekly lab events!
- Grilled Cheese at Baldwin was lit!
- Executive meeting sunday, all are welcome to come
- Revolution UC coming up!
- Dodgeball!
- Upcoming Loveland outreach march 11th
- Smash after meeting!!!!!!



## Workshop: Goat Disassembly

# The Topics Today Go Something Exactly Like This

- Quick touch on Assembly & Disassembly
- The RE tools in Kali and IDA
- Battelle's Feed the Magical Goat CTF

#### Assembly?!

- Nearest possible human readable version of machine code
- Everything is either stored in registers, which can be compared to variables, or in literals values (ints/strings)
- Functions are called subprocesses
- First years take note

	08048919	movl	%esp,%ebp
	0804891b	subl	\$0x4,%esp
	0804891e	movl	\$0x0,0xfffffffc(%ebp)
	08048925	cmpl	\$0x63,0xfffffffc(%ebp)
	08048929	jle	08048930
man	0804892b	jmp	08048948
	0804892d	nop	
nachine	0804892e	nop	
	0804892f	nop	
According	08048930	movl	<pre>0xfffffffc(%ebp),%eax</pre>
stored in	08048933	pushl	%eax
be	08048934	pushl	\$0x8049418
	08048939	call	080487c0 <printf></printf>
es, or in	0804893e	addl	\$0x8,%esp
strings)	08048941	incl	<pre>0xfffffffc(%ebp)</pre>
90)	08048944	jmp	08048925
	08048946	nop	
	08048947	nop	
	08048948	xorl	%eax, %eax
	0804894a	jmp	0804894c
	0804894c	leave	

ret

pushl

08048918

0804894d

%ebp

#### Registers?!

- Usually prefixed with a "%"
- You only have 8 that you should really be looking at / using
- Basically 32 bit pointers / ints
  - Pointers are ints
- Google the names for x64, there's plenty of tables

08048918	pushl	%ebp
08048919	movl	%esp,%ebp
0804891b	subl	\$0x4,%esp
0804891e	movl	\$0x0,0xfffffffc(%ebp)
08048925	cmpl	\$0x63,0xfffffffc(%ebp)
08048929	jle	08048930
0804892b	jmp	08048948
0804892d	nop	
0804892e	nop	
0804892f	nop	
08048930	movl	<pre>0xfffffffc(%ebp),%eax</pre>
08048933	pushl	%eax
08048934	pushl	\$0x8049418
08048939	call	080487c0 <printf></printf>
0804893e	addl	\$0x8,%esp
08048941	incl	0xffffffc(%ebp)
08048944	jmp	08048925
08048946	nop	
08048947	nop	
08048948	xorl	%eax,%eax
0804894a	jmp	0804894c
0804894c	leave	
0804894d	ret	

### Subprocesses

- Equivalent of functions
- Functions arguments are **pushed** 0804892e onto the stack
- The subprocess is **called**
- Subprocess **return** as functions do

```
08048918
             pushl
                     %ebp
08048919
             movl
                    %esp, %ebp
                    $0x4, %esp
0804891b
             subl
                    $0x0,0xfffffffc(%ebp)
0804891e
             movl
                    $0x63,0xfffffffc(%ebp)
08048925
             cmpl
08048929
             ile
                    08048930
0804892b
                    08048948
             jmp
0804892d
             nop
             nop
0804892f
             nop
08048930
                    0xfffffffc(%ebp), %eax
             movl
08048933
             pushl
                     %eax
08048934
             pushl
                     $0x8049418
08048939
             call
                    080487c0 <printf>
0804893e
             addl
                    $0x8, %esp
             incl
                    0xfffffffc(%ebp)
08048941
08048944
             jmp
                    08048925
08048946
             nop
08048947
             nop
08048948
             xorl
                    %eax, %eax
0804894a
                    0804894c
             jmp
0804894c
             leave
0804894d
             ret
```

#### Conditionals

- Variables can be **compared**
- **Jumps** in execution can be made depending on comparisons
- Jumps can also be unconditional (like goto & break)
- C if statements are typically compares and jumps

08048919 0804891b 0804891e

08048918

- 08048925
- 08048929
- 0804892b 0804892d
- 0804892e
- 0804892f 08048930
- 08048933 08048934
- 08048939 0804893e
- sequentially executed
- 08048946 08048947

08048941

08048944

- 08048948
- 0804894a 0804894c
  - 0804894d

- %esp, %ebp
- \$0x4, %esp

%ebp

- \$0x0,0xfffffffc(%ebp)
- \$0x63,0xfffffffc(%ebp)
- cmpl jle 08048930

08048948

jmp nop

pushl

movl

subl

movl

- nop
- nop movl

call

addl

xorl

qmj

- 0xfffffffc(%ebp), %eax pushl %eax
- pushl \$0x8049418
  - 080487c0 <printf>
  - \$0x8, %esp
- 0xfffffffc(%ebp) incl
- jmp 08048925
- nop nop
  - %eax, %eax
- 0804894c leave
- ret

#### **Other Notes**

- Strings are typically stored as static character arrays then copied later when they are used
- This is basically just C with harder syntax and heavy use of goto
- Every instruction has a position offset value compared to where the program's base memory address is

08048918 pushl %ebp 08048919 movl %esp, %ebp 0804891b subl \$0x4, %esp 0804891e \$0x0,0xfffffffc(%ebp) movl \$0x63,0xfffffffc(%ebp) 08048925 cmpl 08048929 ile 08048930 0804892b 08048948 jmp 0804892d nop 0804892e nop 0804892f nop 08048930 0xfffffffc(%ebp), %eax movl 08048933 pushl %eax 08048934 pushl \$0x8049418 call 080487c0 <printf> 08048939 0804893e addl \$0x8, %esp incl 0xfffffffc(%ebp) 08048941 08048944 jmp 08048925 08048946 nop 08048947 nop 08048948 xorl %eax, %eax 0804894a 0804894c qmj 0804894c leave 0804894d ret

#### Other Notes Cont.

- AT&T vs Intel Format
- **Move** operations just copy paste a register value into another \_\_\_ register

pushl %ebp 08048919 movl %esp, %ebp 0804891b subl \$0x4, %esp 0804891e \$0x0,0xfffffffc(%ebp) movl

\$0x63,0xfffffffc(%ebp) 08048925 cmpl 08048929 ile 08048930 0804892b 08048948 jmp

0804892d nop 0804892e nop

08048918

08048934

08048939

0804893e

08048941

08048944

08048946

08048947

08048948

0804894a

0804894c

0804894d

0804892f nop 08048930 movl 08048933 pushl

%eax pushl \$0x8049418 080487c0 <printf> call addl \$0x8, %esp

0xfffffffc(%ebp), %eax

0xfffffffc(%ebp) incl jmp 08048925 nop

nop

qmj

ret

leave

%eax, %eax xorl 0804894c

#### Disassembly

- All the 1337 HaX0rs do it
- You should too
- Process of taking apart binary programs, which are typically compiled from C/C++
- Static analysis Just reading assembly code
- Dynamic analysis running and debugging the program
- Basically just feed a binary in and assembly code comes out



ComputerHope.com

### Disassembly Tools in Kali Linux (and IDA)

Binary Tools (ELF / PE)	Android / Java Tools
diStorm3 IDA edb-debugger OllyDbg Valgrind YARA strings	apktool dex2jar jad javasnoop jd-gui smali



#### Interactive Disassembler (IDA)

- Download the free version from <a href="https://www.hex-rays.com/">https://www.hex-rays.com/</a>
- Grab the Magical Goat zip file from <a href="https://www.battelle.org/cyber-challenge">https://www.battelle.org/cyber-challenge</a>
- I don't have any slides for IDA itself so we'll just go into it with the binary

#### Alternatives to IDA:

- Radare2 (r2)
- Binary Ninja, which has really nice intermediate language support
- GHIDRA, the NSA made equivalent to be released in March

