Intro to x86 Binaries

From ASM to exploit
Intro to x86 Binaries

I lied lets do a quick ctf team thing
Organization

- Ideas? Do we need to a real structure right now?
- Mailing list is OTW
- How do we get more people?
- Suggestions for next week’s CTF?
Intro to x86 Binaries

From ASM to exploit
Notes

- This should be INTERACTIVE
  - Ask questions, poke at things in a terminal
  - This is not just going to be an info dump

- If you have absolutely no clue what's going on at some point ASK
  - I made some guesses as to people's background, but really I have no idea
- If you want, grab the vm now at: http://cseweb.ucsd.edu/classes/sp13/cse127-a/Boxes-2.20.ova
Outline

- Expected background
- Assembly Primer
  - x86 Basics
  - x86 Examples/Syntax
  - x86 64 and beyond
- Program Runtime
  - The Stack
  - Calling conventions
- Exploits
  - What is an exploit
  - Stack Overflows
- Reverse Engineering
  - Examples
- Other Exploits
Background

- **CS Education/Experience**
  - You’ve written some code, hopefully C/C++
  - You’ve had to compile a program before, maybe used GDB

- **CSE 30 or equivalent**
  - You’ve seen assembly, even if it was SPARC/MIPS
  - If not, that’s ok, parts of this may just be super new

- **Security**
  - You’ve heard of exploits, hopefully read about some
Assembly Primer -- What is assembly

- Assembly code is the human readable version of the instructions being performed.

- Example: hello world
# Assembly Primer -- x86 Syntax(es)

**Intel**
- INST Dest, Src
- Register: eax
- Constants: 0ffh
- Dereference: [eax+3]
- Complex: [ebx+ecx*4h-20h]

**AT&T**
- INST Src, Dest
- Register: %eax
- Constants: $0xff
- Dereference: 3(%eax)
- Complex: -0x20(%ebx,%ecx,0x4)

Used by most everyone, somewhat sane

Used by GAS (and thus objdump/gdb by default)

```
set disassembly-flavor intel
-M intel
```

And forget this ever happened
Assembly Primer -- x86 Registers

- Addressing Registers
  - 8 bits, AL, AH
  - 16 bits, AX
  - 32 bits, EAX

- General Register Usage
  - Required
    - EIP: Instruction pointer
    - ESP: Stack pointer
    - EFLAGS: Flags
    - ECX: Counter
  - Common
    - EAX: Return value
    - EDX: Data pointer
    - EBP: Base Pointer
Assembly Primer -- x86 Example

Let's write and look at ‘hello world+’

- Compile to assembly
- Compile to executable and RE
  - First statically
  - Then with gdb
Assembly Primer -- x64 and beyond

Instructions worth knowing (some)

- **REP {instruction} -- Repeat**
  - Runs the given instruction ECX times
  - Also REPNE -- Repeat not equal
- **LEA {dest} {src} -- Load Effective Address**
  - Just like a MOV {dest} [maths]
  - Except it doesn’t dereference
  - LEA EAX, [ECX + 8]
    - EAX = ECX + 8
  - Often used to do math
- **TEST / CMP**
  - TEST: {arg1} & {arg2} only flags stored
  - CMP: SUB, but only flags stored

x64

- RAX, 64bits version
- Pointers!
- R8-R15, some new registers

Extensions SSE, MMX, AVX, NSASPY

- Tons and tons of new bizarre instructions
- Hopefully you won’t see these soon

Floating Point

- FDIV, FMUL, etc
- Again, not that common unless its MATHS
Program Runtime -- The Stack

The stack grows towards LOWER memory addresses.
When a function is called, it creates a new stack FRAME
Local variables are stored on the stack
When a function returns, its frame is popped
Writes to a local array go from lower->higher

http://eli.thegreenplace.net/2011/09/06/stack-frame-layout-on-x86-64/
Program Runtime -- Calling Conventions

- cdecl
  - Classic
  - Push ALL arguments onto the stack
  - Some registers (A,D,C) are CALLER saved, others callee
-fastcall
  - ECX and EDX used to pass arguments, then stack
-stdcall
  - cdecl, but callee cleans up stack
- Many others, usually some variant
  - Of note is the way that member function calls appear, with the THIS ptr in ECX
Exploits -- What is an exploit

- Control Flow Highjack
  - You take control of IP
  - This is usually what we care about

- ‘Logic bug’
  - You take advantage of poor design, but don’t take control
  - Covers SQL injections, clever input values, etc
Exploits -- What is an exploit -- How?

- Control return address
  - Go to buffer of code
  - ROP
- Control function pointer
  - Go to buffer of code
  - ROP
  - Change it to something else
- Unexpected control flow
  - Give it an input the developer didn’t anticipate
  - Disguise your inputs
Exploits -- Stack Overflow

- Most basic attack, overwrite the return pointer
- We will show on a machine with no protections
- Modern machines make this much trickier
- Still a real bug

DEMO!

Grab this file from [http://cseweb.ucsd.edu/~dkohlbre/examples1.tar](http://cseweb.ucsd.edu/~dkohlbre/examples1.tar)
Reverse Engineering -- How

Tools

- file, strings, hexedit/hexdump, etc
  - Astonishingly effective
- GDB / windbg
  - If you can debug the binary, everything gets vastly easier
  - Try changing values during runtime to what you want before you figure out how to make the program do it!
- objdump / other disassembly
  - Get some ASM, go to work
- google
  - Probably the most effective tool
- IDA
  - The real tool
  - I’ll be giving a talk (someday) on this

Techniques

- Manual static analysis
  - Most time consuming, least helpful
  - Always available
- Manual debugging
  - Faster, better than manual
  - Not always available, dangerous?
- Automated tools
  - ie: CMU’s David Brumley
  - Tons of other tools out there
  - Better on ‘real programs’ than CTF often
- Blind
  - Sometimes you only have observed behavior
  - Guess and check!
Reverse Engineering -- Examples

Sorry, I don’t have much right now
Kinda ran short on time
I’ll link some useful pages

Look at Prof. Hovav Shacham’s course page for 127
Or read http://www.devttys0.com/2013/10/reverse-engineering-a-d-link-backdoor/
Other Exploits -- Format String

The new classic

- Of the form: `printf (user_influenced_format);`
- You exploit it using the `%n` operator
  - `%s` takes a string, `%i` takes an integer
  - `%n` takes a POINTER and WRITES to that memory location!
  - It's the only format operator that performs a write
- Honestly, these are a huge pain to write

Basic Idea

- Figure out what's going on in the stack via `%s` or `%x`, and writing an address
- Try to write some useful values in the stack ahead of time
- Use those arguments as pointers to overwrite return pointer/etc piece by piece

```c
address = 0x08480110
address (encoded as 32 bit le string): 
"\x10\x01\x48\x08"
printf ("\x10\x01\x48\x08 %08x.%08x.%08x.%08x.%08x|\%s|");```


Other Exploits -- ROP

- More of a technique than an exploit
- Don’t run your own binary code, use the existing code
- Create a chain of return pointers to useful bits of code
- See Hovav Shacham’s presentation/paper

**Fig. 2. Layout of an ordinary program**

**Fig. 3. Layout of a return-oriented program**
Other Exploits -- Heap Attacks

Heap Spray
- A trick for getting your code somewhere you can use it on a victim
- Cause target to allocate and fill large amounts of heap memory with a given input.
- NOP Sleds

Use-after-free
- Heap allocated objects don’t just disappear after being freed
- If something can write over that objects previous location (say, because that space got reallocated) it can change the ‘old’ object

Heap Corruption
- As in stack overflow, just write to other heap objects
Finale

- This was a ton of stuff really fast
- RE is practice practice and practice
- Exploit writing is practice, practice, and patience

Lets go win a ctf