



Advanced Static Analysis

with 64 bit code too!



DawgCTF

March 11th! Teams of 4! (or less)

<http://umbccd.umbc.edu/dawgctf>

Lunch and dinner provided

Prizes for top teams

SIGN UP



Recap of last week

- Binary Ninja!
- The 4 steps of RE
- Reverse engineering basic programs
 - Dealing with numbers, bitwise math, etc



4 steps of RE

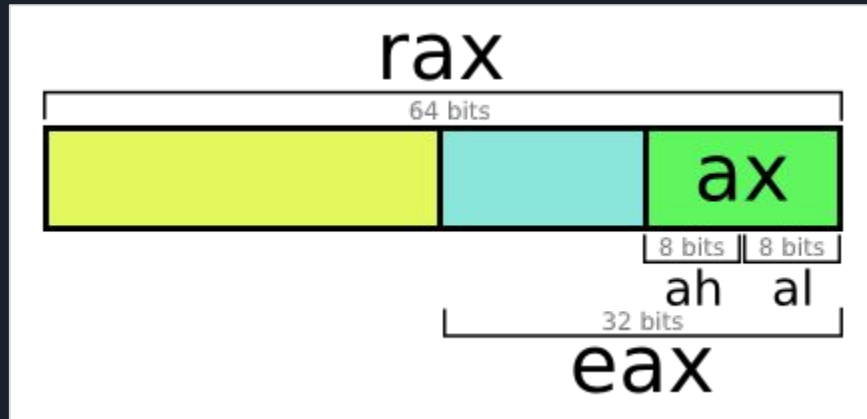
- Identify Code
 - “Start at the bottom, stop when you see math”
 - Work your way back from the goal to identify what you need to seriously reverse
- Identify Input
 - Find out what we can do to influence execution flow through the program
 - Work your way down from the start of the program
- Analysis!
 - Figure out what inputs we need to supply to get through the blocks we identified in step 1
- Implementation
 - Do the thing to get to the goal state, reap rewards



64 bit assembly

- Instead of 32 bits, now all our registers are 64 bits wide now!
 - Prefix changes from “e” to “r”: eg rax, rbx, rcx instead of eax, ebx, and ecx
 - Can still access eax, ebx etc to look at the lower 32 bits of a register
- Have a few extra registers: r8, r9, r10....
- Memory addresses are 48 bit now, instead of 32 bit
 - Not 64 bit!
- Pretty much all the instructions work as you would expect them too

x64 registers





64 bit calling convention

- In 32 bit assembly, we pass arguments to functions on the stack
- In 64 bit assembly, we pass them in registers:
 - Argument 1 in rdi (edi from 32 bit)
 - Argument 2 in rsi
 - Argument 3 in rdx
 - Argument 4 in rcx
 - Argument 5 in r8
 - Argument 6 in r9
 - Rest of the arguments on the stack
- This is faster than pushing stuff onto the stack
- Return value is still in rax (eax)



IDA Pro

Sadly, the free version of Binary Ninja that we used last week doesn't support disassembling 64 bit code

Thankfully the new free version of IDA Pro does, so we can use that

If you own the paid version of Binary Ninja (only \$150!) you can use that instead

IDA Pro (the paid version) does have some advantages over Binary Ninja:

- Handles malware better
- Has a debugger built in
- Supports more architectures



This is the old code we looked at..

```
int main(int argc, char** argv)
{
    printf("gimme num1: ");
    unsigned int n3;
    unsigned int n2;
    unsigned int n1;
    scanf("%u", &n1);
    printf("gimme num2: ");
    scanf("%u", &n2);
    printf("gimme num3: ");
    scanf("%u", &n3);
    unsigned int t1 = n1 ^ 0xcafebabe;
    unsigned int t2 = n2 ^ 0x13371337;
    unsigned int t3 = n3 ^ 0xdeadbeef;
    if((t1) != 2053139493 || (t2) != 3937125628 || (t3) != 3449478826)
    {
        puts("Sorry, I didn't like those numbers");
        return 0;
    }

    n1 = n1 ^ 0xdeadbeef;
    n2 = n2 ^ 0xcafebabe;
    n3 = n3 ^ 0x13371337;
    printf("The flag is: %s\n", &n1);
}
```



New code:

```
struct UserInfo
{
    char* username;
    char* password;
    int age;
};
```

```
class Song
{
    public:
        Note* getNotes();
    private:
        Note* notes;
        int songlen;
};
```



How do we deal with this?

Binary Ninja and IDA both let us specify types for things, and apply those types to variables in the disassembly

This is the big way to make complex, object oriented code readable

You have to do it manually, but it makes a world of difference

```
struct Elf32_ProgramHeader
{
0000    uint32_t type;
0004    uint32_t offset;
0008    uint32_t virtual_address;
000c    uint32_t physical_address;
0010    uint32_t file_size;
0014    uint32_t memory_size;
0018    uint32_t flags;
001c    uint32_t align;
0020 };

struct Elf32_SectionHeader
{
0000    uint32_t name;
0004    uint32_t type;
0008    uint32_t flags;
000c    uint32_t address;
0010    uint32_t offset;
0014    uint32_t size;
0018    uint32_t link;
001c    uint32_t info;
0020    uint32_t align;
0024    uint32_t entry_size;
0028 };

typedef void* va_list;
```


Create Type

Type name: UserInfo

Type (C syntax):

struct
{
 char* username;
 char* password;
 int age;
};

Close Create



```
00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/* : create structure member (data/ascii/array)
00000000 ; N : rename structure or structure member
00000000 ; U : delete structure member
00000000 ; -----
00000000
00000000 marimo      struc ; (sizeof=0x18, mappedto_1)
00000000 birth_time    dd ?
00000004 size?       dd ?
00000008 name        dq ? ; offset
00000010 profile     dq ? ; offset
00000018 marimo      ends
00000018
```



Structures in IDA cheat sheet

Insert in the structures window: create a new structure

Hit 'd' on the start or end to add a new member

Either hit 'd' again on the member or right click to change the size

Rename with 'n'

Continue until you've fully defined the structure

Use 'Set Type' (hotkey: 'y') on a variable to set the type to whatever you did (may have to declare as a pointer)



Structures in Binja cheat sheet

Just type in the struct as a C struct and it will auto make it

Then you can right click a variable and change it's type (hotkey: 'y')

Sometimes it gets propagated to MLIL/LLIL



Reversing C++

It's pretty easy until you throw virtual functions in the mix

The names are really weird (eg, wikipedia::article::format becomes `_ZN9Wikipedia7article6formatE`)

All member function calls go from this:

```
object.function(arg1, arg2...)
```

To this:

```
class::function(object, arg1, arg2...)
```




Chris puts around in IDA for a bit

Just so you aren't completely lost when starting it



Lab!

I have a mix of 32 bit and 64 bit challenges for you at:

<http://toomanybananas.com> (<http://52.91.53.202/>)

You can use either the free version of IDA Pro or the paid version of Binary Ninja, if you have it

As usual I'll be walking around helping people

These ones should be a bit easier than last time