Augmenting Binary Analysis with Python and Pin

November 14th, 2014

@ancat, @1blankwall1

Who are we?

About Us

- Omar
  - Security Engineer at Etsy
  - Recent graduate of NYU
- Tyler
  - Studies at NYU

What is binary analysis?

What is binary analysis?

- *Binary:* A file containing all the resources and native code needed for a program to execute
- *Analysis:* To make sense of an application when the original intentions are not clear or known

Using a debugger (WinDbg, GDB, Immunity, etc)

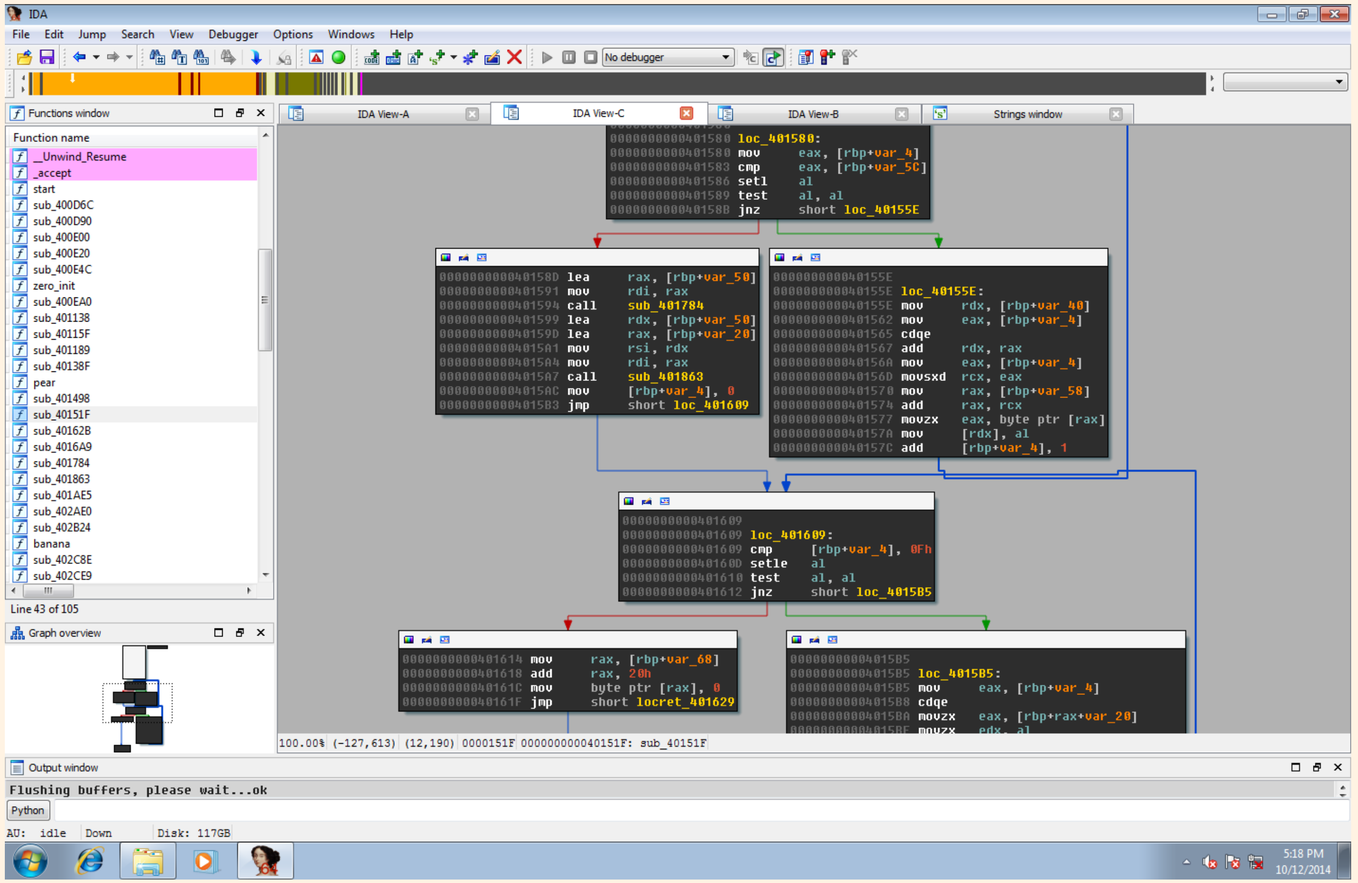
```
   0x7ffff72771f3 <__GI___libc_malloc+35>:      mov    rbx,QWORD PTR fs:[rax]
(gdb) finish
Run till exit from #0  __GI___libc_malloc (bytes=140737488347216) at malloc.c:2876
0x0000000000404128 in ?? ()
1: x/10i $pc
=> 0x404128:    mov    QWORD PTR [rbp-0x20],rax
   0x40412c:    mov    eax,DWORD PTR [rip+0x20140e]        # 0x605540
   0x404132:    mov    edx,DWORD PTR [rbp-0x14]
   0x404135:    mov    edi,edx
   0x404137:    sub    edi,eax
   0x404139:    mov    eax,edi
   0x40413b:    mov    edx,eax
   0x40413d:    shr    edx,0x1f
   0x404140:    add    eax,edx
   0x404142:    sar    eax,1
Value returned is $3 = (void *) 0x6065f0
```

Simply observing the execution of a binary

```
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
hello world

BOOM!!!
The bomb has blown up.
$ ./bomb
Welcome to my fiendish little bomb. You have 6 phases with
which to blow yourself up. Have a nice day!
Public speaking is very easy.
Phase 1 defused. How about the next one?
```

Reading disassembly output (IDA, objdump, etc)



Running /usr/bin/strings on a binary

```
So you think you can stop the bomb with ctrl-c, do you?
Well...
OK. :-)
Invalid phase%s
%d %d %d %d %d %d
Bad host (1).
Bad host (2).
Bad host (3).
Error: Premature EOF on stdin
GRADE_BOMB
Error: Input line too long
ERROR: dup(0) error
ERROR: close error
ERROR: tmpfile error
Subject: Bomb notification
nobody
defused
exploded
bomb-header:%s:%d:%s:%s:%d
bomb-string:%s:%d:%s:%d:%s
bomb
/usr/sbin/sendmail -bm
%s %s@%s
ERROR: notification error
ERROR: fclose(tmp) error
ERROR: dup(tmpstdin) error
ERROR: close(tmpstdin)
BOOM!!!
The bomb has blown up.
%d %s
austinpowers
Curses, you've found the secret phase!
But finding it and solving it are quite different...
Congratulations! You've defused the bomb!
generic
isrveawhobpnutfg
(END)
```

Static Analysis

- Reading disassembly output (IDA, objdump, etc)
- Running /usr/bin/strings on a binary
- Using a debugger (WinDbg, gdb, Immunity, etc)
- Simply observing the execution of a binary

Dynamic Analysis

- Reading disassembly output (IDA, objdump, etc)
- Running /usr/bin/strings on a binary
- Using a debugger (WinDbg, gdb, Immunity, etc)
- Simply observing the execution of a binary

Static vs Dynamic

- Speed
- Level of Understanding
- Code Coverage
  - Static can cover 100% of the code (good or bad?)
  - Dynamic can be accurate due to run time information

Introducing...

Dynamic Binary Instrumentation

Dynamic Binary Instrumentation

- A technique to modify the behavior of programs based on certain conditions during execution
- Generally, DBI involves redirecting execution to your code before or after the target executes
  - Sometimes done by modifying the code before starting the program
  - For example, an INT3 instruction on x86 used by debuggers, or less specifically, trampolines

Debugger Scripting

- GDB & LLDB
  - Scriptable using Python - Unix only (mostly)
- WinDBG
  - Scriptable using Python (somewhat) - Windows only
- VDB
  - Entirely Python API - Windows and and Unix support
  - Lacks documentation, breaks often

Debugger Scripting

```
define structs
    set $target = $root
    set $limit = 0
    while $target
        printf "[0x%x] node.name=0x%x; node.value=0x%x; node.next=0x%x; node.prev=0x%x\n", $target, *($target), *($target+4), *($target+8), *($target+0xc)
        set $old_target = $target
        set $target = *($target+8)

        if $old_target == $target
            set $limit = $limit + 1
        end

        if $limit > 10
            printf "Infinite loop?\n"
            set $target = 0
        end
    end
end
```

DBI Frameworks

- Valgrind
  - GPL'd system for debugging and profiling Linux programs
  - Automatically detects many memory management and threading bugs
  - Works on x86/Linux, AMD64/Linux and PPC32/Linux
  - Focused on Safe and Reliable Code
    - Developer tool used for finding code errors

DBI Frameworks

- Valgrind
  - DEMO

DBI Frameworks

- Address Sanitizer
  - Fast memory error detector
  - The tool consists of a compiler instrumentation module (currently, an LLVM pass) and a run-time library which replaces the malloc function
  - Works on x86 Linux, and Mac, and ARM Android
  - Focused on bugs
    - Heap/Stack Buffer overflows and Use After Free

DBI Frameworks

- Address Sanitizer
  - DEMO????

DBI Frameworks

- DynamoRIO
  - Runtime code manipulation system that supports code transformations on any part of a program at runtime
  - Works on x86/AMD64 Linux Mac, and Windows
  - Transparent, and comprehensive manipulation of unmodified applications running on stock operating systems
  - Direct Competitor to Pin :-!

What *is* Pin?

- Pin allows user to insert arbitrary code into an executable right after it is loaded into memory
- Generates code from a “PinTool” used to “hook” instructions and calls
- Pin is the framework
- PinTools are the interface
  - The mechanism that decides where and what code is inserted
  - The code to execute at insertion points

Why Pin?

Intel's Pin

- Amazing documentation
- Same exact API works for Windows and Unix
- Extremely popular
- Nothing needs to be recompiled to be used with Pin

It's easy to get started

- Large repo of well commented sample tools come with Pin
- Documentation is generally easy to follow
- Installation is a piece of cake

It can be as granular as you need it to be

- Simple hook/callback system
  - instructions
  - basic blocks
  - function calls
  - and so on

Mostly personal preference, though

Why not Pin?

- The Pin API uses C++
  - Not a huge deal, but can be inconvenient during a time crunch (ctf)
  - Harder to prototype
- Slower than other DBI Frameworks
- Not as granular as other solutions
  - Harder to do more advanced binary analysis techniques such as taint tracing

Awesome but what can Pin do?

Popular Uses

- The Pin API has been used extensively in industry
- Most notably Microsoft Blue Hat (2012) Winner kBouncer (Vasilis Pappas)
  - Efficient and fully transparent ROP mitigation technique
  - Very similar to second place ROPGuard (Ivan Fratric)
    - Used in Microsofts EMET protection system
- IDA 6.4 and above includes a pin tool for tracing code in the debugger

Cool... WHERE ARE MY BUGS?!

- Pin can be used to find many different classes of bugs
- Most can be found by using the right kind of instrumentation
  - Format Strings
    - Analyze parameters passed to formatting functions
  - Buffer Overflows
    - Analyze memory read and write instructions
  - Misused Memory Allocation (Double Frees or UAF)
    - Analyze memory allocation functions (malloc/free) and memory writes

Misused Heap Allocations

- How to find these dynamically?
  - Keep track of all malloc calls and the addresses returned
  - Maintain state: Freed or In use and size
  - When a memory read or write happens, if the target is on the heap, verify that the memory is a valid place to be read from or written to

D-d-d-d-d-demo!

- Pin use after free demo

Pin

- Wow, Pin is really cool!
- But, wait! Pin is a mess!
  - Correction, C++ is a mess :P
- Lots of necessary boilerplate code
- Hard to prototype quickly
- Difficult to understand

C++

```
RTN mallocRtn = RTN_FindByName(img, MALLOC);
if (RTN_Valid(mallocRtn))
{
    RTN_Open(mallocRtn);

    // Instrument malloc() to print the input argument value and the return value.
    RTN_InsertCall(mallocRtn, IPOINT_BEFORE, (AFUNPTR) Arg1Before,
                   IARG_ADDRINT, MALLOC,
                   IARG_FUNCARG_ENTRYPOINT_VALUE, 0,
                   IARG_END);
    RTN_InsertCall(mallocRtn, IPOINT_AFTER, (AFUNPTR) MallocAfter,
                   IARG_FUNCRET_EXITPOINT_VALUE, IARG_END);

    RTN_Close(mallocRtn);
}
```

Python

```
rtn = pin.RTN_FindByName(img, "malloc")
if pin.RTN_Valid(rtn):
        pin.RTN_Open(rtn)
        pin.RTN_InsertCall(pin.IPOINT_BEFORE, "malloc", rtn, 1, malloc_before)
        pin.RTN_InsertCall(pin.IPOINT_AFTER, "malloc", rtn, 1, malloc_after)
        pin.RTN_Close(rtn)
```

C++ vs Python

- Python
  - Simpler
  - Cleaner
  - No need for recompilation every time
  - Extensive libraries and support

Python-Pin

- Essentially, a python interpreter embedded within a PinTool
  - “Virtual” pin module exposed to the python script
  - Enables access to most of Pin’s functionality from within python
  - Quick and easy to write PinTools
  - Negligible performance impact
  - Enables seamless integration with other Python modules
    - Z3py, PIL, SciPy, etc

Python-Pin Demo

- Use after free and double free
- Code coverage IDA basic block coloring
- Basic utility demos

The Future of Python-Pin

- Better memory management
- Implement more Pin functionality
- More testing for Windows support

Acknowledgements

Tyler Bohan
Kevin Chung
Dan Guido
Robert Meggs
Jonathan Salwan
Rich Smith
Paolo Soto
Alex Sotirov
Kai Zhong

Thanks for tuning in!

- Slides and pin tools will be posted to twitter, probably
  - @ancat/@1blankwall1