OSX VULNERABILITY RESEARCH AND WHY WE WROTE OUR OWN DEBUGGER

Tyler Bohan
Brandon Edwards

WHO WE ARE

- Security researchers for BAE Systems

- Exploitation prevention detection and creation

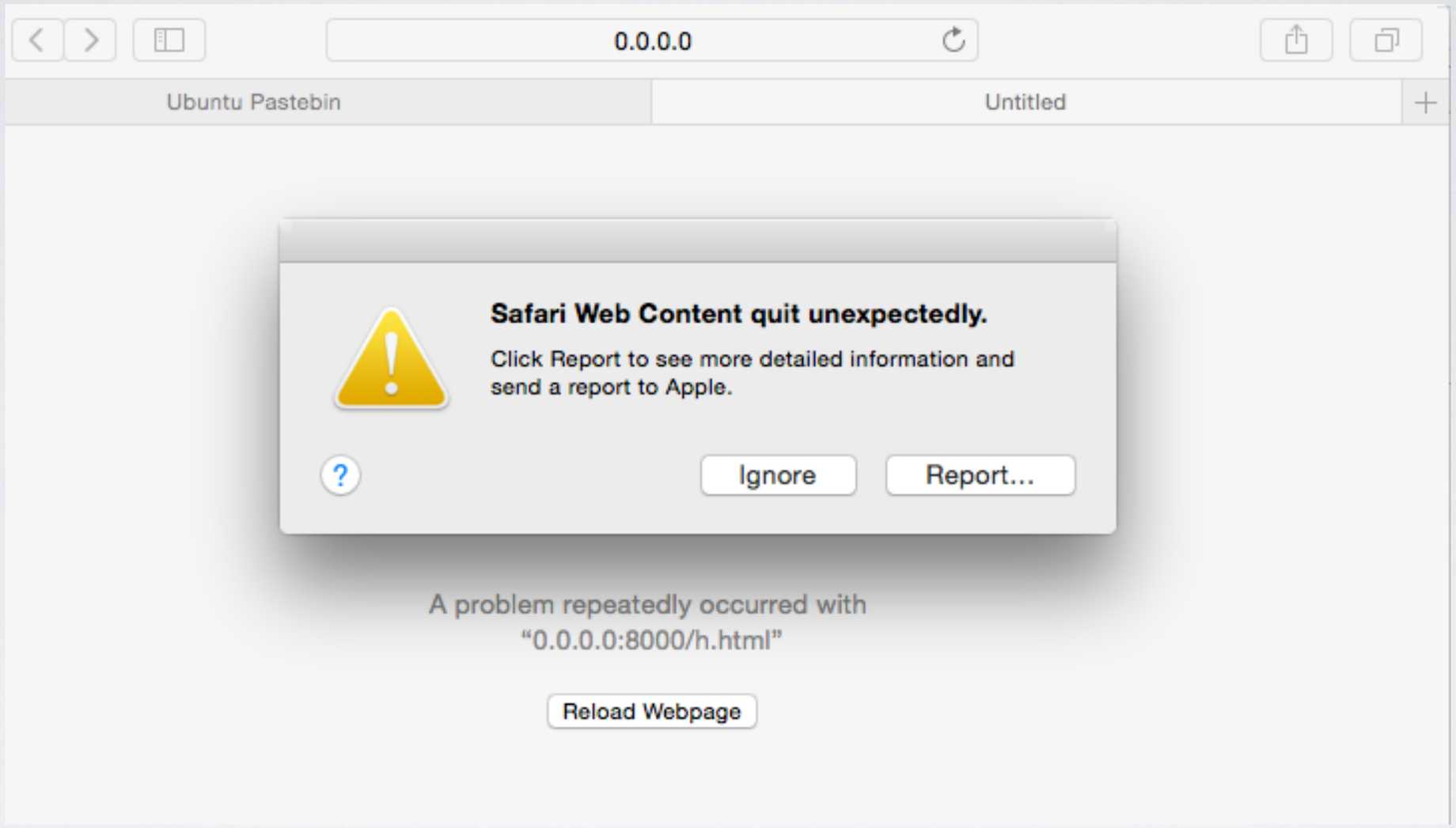
- Bug hunting and reverse engineering

HOW THIS BEGAN

- Write awesome fuzzer
- Find lots of bugs
- Profit

```
try{e[2].suspendRedraw(0);}catch(x){}
try{e[2].setCurrentTime(0);}catch(x){}
try{e[2].unsuspendRedrawAll();}catch(x){}
garbage_collect();
try{e[2].forceRedraw();}catch(x){}
try{e[2].animationsPaused();}catch(x){}
try{e[2].unsuspendRedraw(0);}catch(x){}
try{e[2].pauseAnimations();}catch(x){}
try{e[2].deselectAll();}catch(x){}
try{e[2].checkEnclosure(svgns, svgRect);}catch(x){}
console.log(200)
console.log(error)
try{e[2].suspendRedraw(0);}catch(x){}
try{e[2].xmllang;}catch(x){}
try{v17 = e[2].height;}catch(x){}
try{v18 = e[2].requiredExtensions;}catch(x){}
try{e[2].setCurrentTime(0);}catch(x){}
try{e[2].unsuspendRedrawAll();}catch(x){}
try{e[2].hasExtension(unescape("obediant"));}catch(x){}
try{e[2].forceRedraw();}catch(x){}
try{e[2].unpauseAnimations();}catch(x){}
try{e[2].animationsPaused();}catch(x){}
try{e[2].unsuspendRedraw(0);}catch(x){}
try{e[2].checkIntersection(svgns, svgRect);}catch(x){}
try{e[2].deselectAll();}catch(x){}
try{e[2].checkEnclosure(svgns, svgRect);}catch(x){}
try{e[2].suspendRedraw(0);}catch(x){}
garbage_collect();
```

WHOOPS


0.0.0.0
Ubuntu Pastebin
Untitled
Safari Web Content quit unexpectedly.
Click Report to see more detailed information and send a report to Apple.
Ignore
Report...
A problem repeatedly occurred with "0.0.0.0:8000/h.html"
Reload Webpage

HOW THIS BEGAN

- Quickly find first crash
- Well versed in Windows and Linux exploitation
- How hard can it be?

```
<html>
    <style>
    svg {
        padding-top: 2000%;
        box-sizing: border-box;
    }
    </style>
    <svg viewBox="1 2 500 500" width="900" height="900">
        <polyline points="1 1,2 2"></polyline>
    </svg>
</html>
```

HOW THIS BEGAN

- Different command layout
- Verbose and cumbersome documentation

Show all registers in all register sets for the current thread.	
(gdb) info all-registers	**(lldb)** register read --all **(lldb)** re r -a
Show the values for the registers named "rax", "rsp" and "rbp" in the current thread.	
(gdb) info all-registers rax rsp rbp	**(lldb)** register read rax rsp rbp
Show the values for the register named "rax" in the current thread formatted as binary.	
(gdb) p/t $rax	**(lldb)** register read --format binary rax **(lldb)** re r -f b rax *LLDB now supports the GDB shorthand format syntax but there can't be space after the command:* **(lldb)** register read/t rax **(lldb)** p/t $rax
Read memory from address 0xbffff3c0 and show 4 hex uint32_t values.	
(gdb) x/4xw 0xbffff3c0	**(lldb)** memory read --size 4 --format x --count 4 0xbffff3c0 **(lldb)** me r -s4 -fx -c4 0xbffff3c0 **(lldb)** x -s4 -fx -c4 0xbffff3c0 *LLDB now supports the GDB shorthand format syntax but there can't be space after the command:* **(lldb)** memory read/4xw 0xbffff3c0 **(lldb)** x/4xw 0xbffff3c0 **(lldb)** memory read --gdb-format 4xw 0xbffff3c0

HOW THIS BEGAN

- **gdb:**
  x/4wx 0xbffff3c0
- **lldb**:
  memory read --size 4 --format x --count 4 0xbffff3c0

SCRIPTABILITY

- Complex debugging scenarios
- Reproducible, reusable across projects
- Interoperation with other tools
- Quickly testing analysis ideas

SCRIPTABILITY

- Python based scripting available
- Not entirely intuitive, mostly designed to be used inside of LLDB
- Lacks functionality of fully scriptable debuggers
- More documentation than anyone should have to read

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print “Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print “Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print “Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SO MANY WORDS

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName
    breakpoint.SetScriptCallbackFunction('test.brea
```

```
def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"
```

SO MANY WORDS

```
def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print "Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY

You will need your Python module to contain the script function you want executed, and pass it by qualified name on the command line. For instance, if you have

```
# myfile.py
def callback(wp_no):
   # stuff
# more stuff
mywatchpoint = ...
debugger.HandleCommand("watchpoint command add -F myfile.callback %s" % mywatchpoint.GetID()
```

would be the way to tell LLDB about your callback

Currently, there is no way to pass Python functions directly to LLDB API calls.

SCRIPTABILITY

- We want something more practical
- Ideally closer to Visigoth's VDB / vtrace

```
import vtrace
class MyCallback(vtrace.Breakpoint):
    def notify(self, event, trace):
        print "Hit Breakpoint!"

breakpoint = MyCallback(None, "SSLWrite")
trace.addBreakpoint(breakpoint)
trace.run()
```

SCRIPTABILITY

- Ideally closer to Visigoth's VDB / vtrace

- Stand-alone, independent scripts

```
import vtrace
class MyCallback(vtrace.Breakpoint):
    def notify(self, event, trace):
        print "Hit Breakpoint!"

trace = vtrace.getTrace()
trace.attach(503)
breakpoint = MyCallback(None,"SSLWrite")
trace.addBreakpoint(breakpoint)
trace.run()
```

WHAT ELSE IS AVAILABLE?

Bit Slicer

Download Bit Slicer

Introduction

Bit Slicer is a universal game trainer for OS X, v

It allows you to cheat in video games by searcl
more.

WHAT ELSE IS AVAILABLE?

Ragweed

```
by tduehr, crohlf, and tqbf
http://www.matasano.com/research/ragweed/
```

DESCRIPTION:

- Ragweed is a set of scriptable debugging tools written in native ruby.
- Where required the FFI and Win32API libraries are used to interface th
- There are no third party dependencies

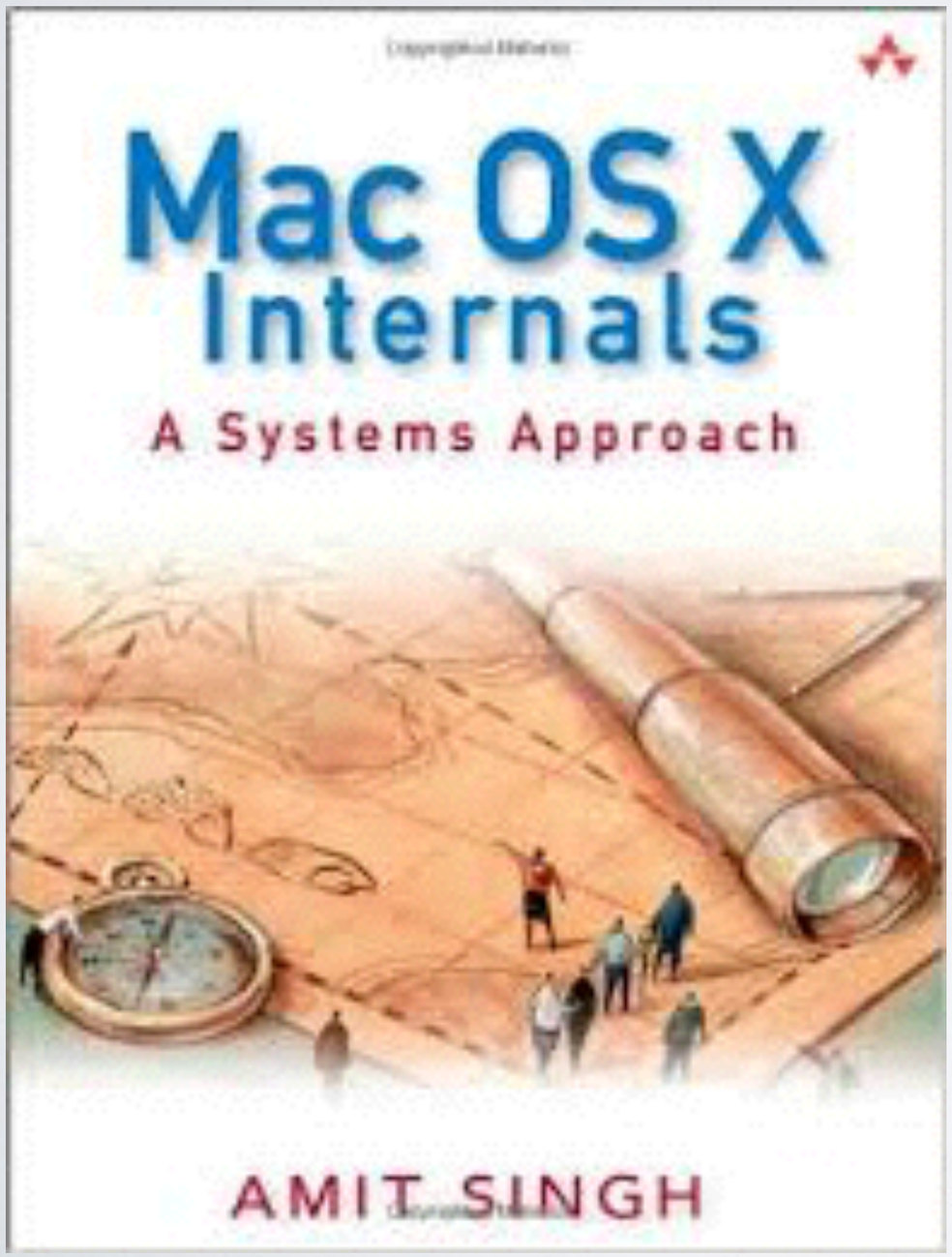
WHAT ELSE IS AVAILABLE?

Vdb

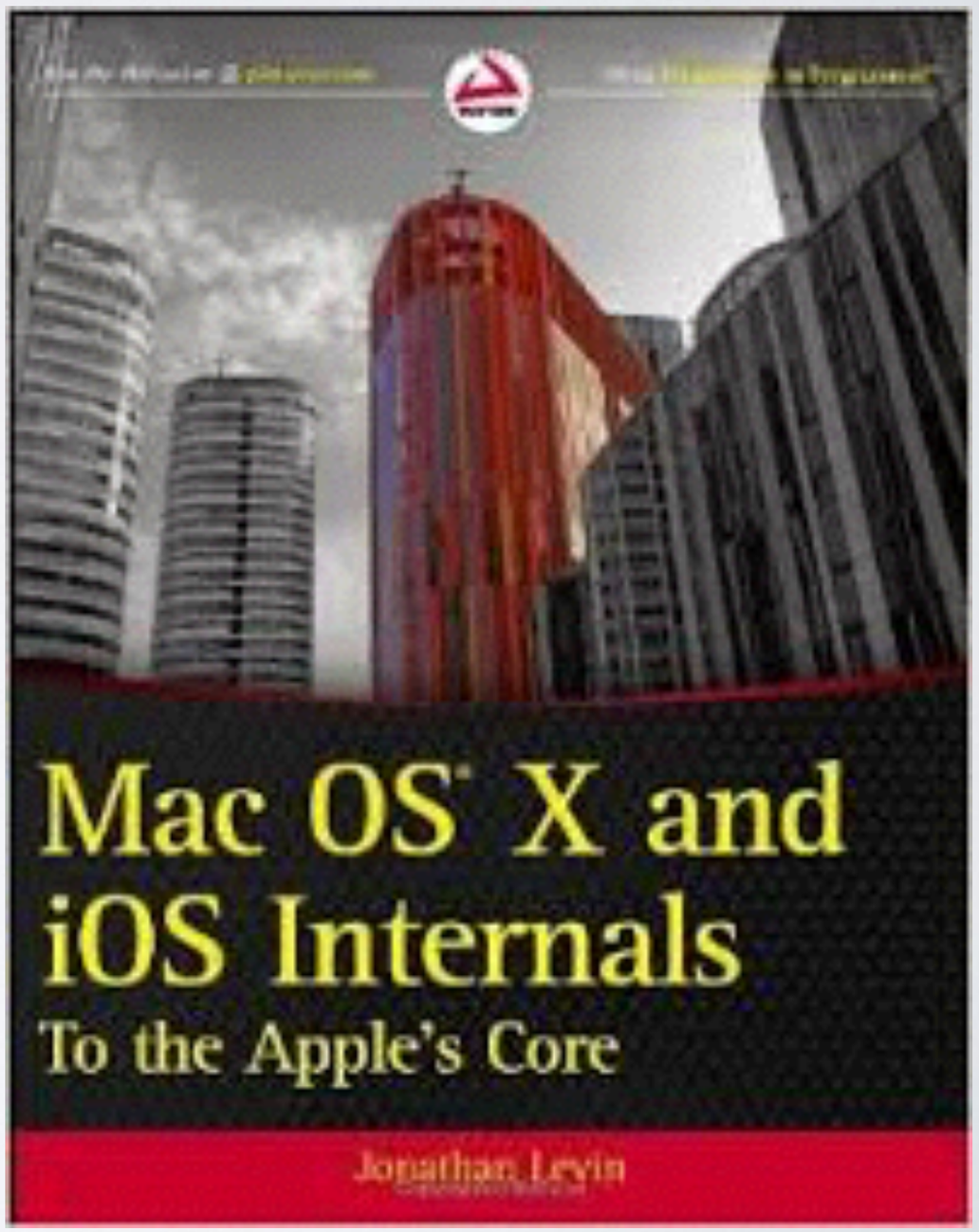
As in previous vdb releases, the command `python vdbbin` from the checkout directory will drop y
prompt on supported platforms. (Windows / Linux / FreeBSD / OSX... kinda?)

HOW'D WE DO IT?

2006



2013



HOW'D WE DO IT?

- Windows -
  - Awesome API and great debugging documentation
- LINUX -
  - Ptrace and other commonly known debugging api's
- OSX -
  - Ptrace kind of…

HOW'D WE DO IT?

- Following slides are all examples in C of the debugging backend

HOW'D WE DO IT?

- Ptrace totally neutered
  - Only allows attaching and detaching - kind of
  - No peak memory - poke memory ??? :(
  - **TL;DR not using ptrace**

HOW'D WE DO IT?

- This awesome blog post written in 2006!!

ATTACH

```
pid_t  pid;
task_t port;

task_for_pid(mach_task_self(), pid, &port);
```

READ & WRITE MEMORY

```
mach_vm_write(vm_map_t map,
              vm_address_t address,
             pointer_t  data,
              unused mach_msg_type_number_t)


mach_vm_read(vm_map_t map,
             mach_vm_address_t addr,
             mach_vm_size_t size,
              pointer_t *data,
             mach_msg_type_number_t*dsize);
```

VIEWING/SETTING REGISTER STATE

```
[thread/act]_[get/set]_state(
                        thread_t thread,
                    int_t flavor,
                    thread_state_t flavor,
            mach_msg_type_number_t  *count)
```

EXCEPTION HANDLING

```
static void exception_server (mach_port_t exceptionPort) {
        ……
      mach_exc_server(msg, reply);
          // Send the now-initialized reply
          rt = mach_msg(reply, MACH_SEND_MSG, reply->msgh_size,
0, MACH_PORT_NULL, 0)
        }
    }
```

EXCEPTION DISPATCH

```
// Handle EXCEPTION_DEFAULT behavior
 kern_return_t catch_mach_exception_raise (
                                          mach_port_t exception_port,
                                          mach_port_t thread,
                                          mach_port_t task,
                                          exception_type_t exception,
                                          mach_exception_data_t code,
                                          mach_msg_type_number_t codeCnt)
```

SIGNALS, EXIT, FORK, OH MY???

- Without ptrace we cant use wait()
- Without wait we cant catch process signals
- Without signals we cant catch process exit or fork
- ?????

KQUEUES

The kqueue() system call provides a generic method of notifying the user when an event happens or a condition holds, based on the results of small pieces of kernel code termed filters.

NOTE_EXIT The process has exited.

NOTE_FORK The process created a child process via fork(2) or similar call.

NOTE_EXEC The process executed a new process via execve(2) or similar call.

NOTE_SIGNAL The process was sent a signal. Status can be checked via waitpid(2) or similar call.

KQUEUES

```
i = kevent(kq, NULL, 0, &ke, 1, NULL);
if (i == -1)
    err(1, "kevent!");

if (ke.fflags & NOTE_FORK)
    printf("pid %d called fork()\n", ke.ident);

if (ke.fflags & NOTE_CHILD)
    printf("pid %d has %d as parent\n", ke.ident,
        ke.data);

if (ke.fflags & NOTE_EXIT)
    printf("pid %d exited\n", ke.ident);

if (ke.fflags & NOTE_EXEC)
    printf("pid %d called exec()\n", ke.ident);

if (ke.fflags & NOTE_TRACKER)
    printf("couldnt attach to child of %d\n", ke.ident);
```

KQUEUES

```
i = kevent(kq, NULL, 0, &ke, 1, NULL);
if (i == -1)
    err(1, "kevent!");

if (ke.fflags & NOTE_FORK)
    printf("pid %d called fork()\n", ke.ident);
if (ke.fflags & NOTE_CHILD)
    printf("pid %d has %d as parent\n", ke.ident,
        ke.data);

if (ke.fflags & NOTE_EXIT)
    printf("pid %d exited\n", ke.ident);

if (ke.fflags & NOTE_EXEC)
    printf("pid %d called exec()\n", ke.ident);

if (ke.fflags & NOTE_TRACKER)
    printf("couldnt attach to child of %d\n", ke.ident);
```

SYSTEM INTEGRITY PROTECTION

- Introduced in El Capitan
- Protected locations cannot be written to by root
- Protected system processes cannot be attached to with a debugger and cannot be subject to code injection
- All kernel extensions must now be signed
- SIP cannot be disabled from within the operating system, only from the OS X Recovery partition

SYSTEM INTEGRITY PROTECTION

- Protected Locations: /bin,/System,/usr,/sbin

```
➜ /usr  echo "hello" > example.text
zsh: operation not permitted: example.text
➜ /usr
```

- And Apple programs in /Applications

```
➜ /usr  pgrep -i Safari
309
415
426
427
435
499
3392
➜ /usr  sudo lldb
Password:
(lldb) attach 309
error: attach failed: lost connection
(lldb)
```

SYSTEM INTEGRITY PROTECTION

Yet we can still use our debugger on them quite easily* :)

```
pid_t  bypass_sip(char *command, char *args[]) {
      execv(command, args);   // run the command
}
```

*Wont work on LLDB :p

WHAT DO WE WANT?

- Easy to use debugger
- Friendly interface
- Lightweight
- Easily Scripted

HOW DO WE GET IT?

- Write our own
- :(
- :)

DESIGN GOALS

- Augment LLDB not replace it
- Easily scripted without overhead
- Offer a lightweight interface not…

All Functions

_lldb'.SBAddress_Clear
_lldb'.SBAddress_GetAddressClass
_lldb'.SBAddress_GetBlock
_lldb'.SBAddress_GetCompileUnit
_lldb'.SBAddress_GetDescription
_lldb'.SBAddress_GetFileAddress
_lldb'.SBAddress_GetFunction
_lldb'.SBAddress_GetLineEntry
_lldb'.SBAddress_GetLoadAddress
_lldb'.SBAddress_GetModule
_lldb'.SBAddress_GetOffset
_lldb'.SBAddress_GetSection
_lldb'.SBAddress_GetSymbol
_lldb'.SBAddress_GetSymbolContext
_lldb'.SBAddress_IsValid
_lldb'.SBAddress_OffsetAddress
_lldb'.SBAddress_SetAddress
_lldb'.SBAddress_SetLoadAddress
_lldb'.SBAddress___str__
_lldb'.SBAddress_swigregister
_lldb'.SBAttachInfo_EffectiveGroupIDIsValid
_lldb'.SBAttachInfo_EffectiveUserIDIsValid
_lldb'.SBAttachInfo_GetEffectiveGroupID
_lldb'.SBAttachInfo_GetEffectiveUserID
_lldb'.SBAttachInfo_GetGroupID
_lldb'.SBAttachInfo_GetIgnoreExisting
_lldb'.SBAttachInfo_GetParentProcessID
_lldb'.SBAttachInfo_GetProcessID
_lldb'.SBAttachInfo_GetProcessPluginName
_lldb'.SBAttachInfo_GetResumeCount
_lldb'.SBAttachInfo_GetUserID
_lldb'.SBAttachInfo_GetWaitForLaunch
_lldb'.SBAttachInfo_GroupIDIsValid
_lldb'.SBAttachInfo_ParentProcessIDIsValid
_lldb'.SBAttachInfo_SetEffectiveGroupID
_lldb'.SBAttachInfo_SetEffectiveUserID
_lldb'.SBAttachInfo_SetExecutable
_lldb'.SBAttachInfo_SetGroupID
_lldb'.SBAttachInfo_SetIgnoreExisting

DESIGN GOALS

All LLDB functions available for python

Continues….

All Functions

_lldb'.SBAddress_Clear
_lldb'.SBAddress_GetAddressClass
_lldb'.SBAddress_GetBlock
_lldb'.SBAddress_GetCompileUnit
_lldb'.SBAddress_GetDescription
_lldb'.SBAddress_GetFileAddress
_lldb'.SBAddress_GetFunction
_lldb'.SBAddress_GetLineEntry
_lldb'.SBAddress_GetLoadAddress
_lldb'.SBAddress_GetModule
_lldb'.SBAddress_GetOffset
_lldb'.SBAddress_GetSection
_lldb'.SBAddress_GetSymbol
_lldb'.SBAddress_GetSymbolContext
_lldb'.SBAddress_IsValid
_lldb'.SBAddress_OffsetAddress
_lldb'.SBAddress_SetAddress
_lldb'.SBAddress_SetLoadAddress
_lldb'.SBAddress___str__
_lldb'.SBAddress_swigregister
_lldb'.SBAttachInfo_EffectiveGroupIDIsValid
_lldb'.SBAttachInfo_EffectiveUserIDIsValid
_lldb'.SBAttachInfo_GetEffectiveGroupID
_lldb'.SBAttachInfo_GetEffectiveUserID
_lldb'.SBAttachInfo_GetGroupID
_lldb'.SBAttachInfo_GetIgnoreExisting
_lldb'.SBAttachInfo_GetParentProcessID
_lldb'.SBAttachInfo_GetProcessID
_lldb'.SBAttachInfo_GetProcessPluginName
_lldb'.SBAttachInfo_GetResumeCount
_lldb'.SBAttachInfo_GetUserID
_lldb'.SBAttachInfo_GetWaitForLaunch
_lldb'.SBAttachInfo_GroupIDIsValid
_lldb'.SBAttachInfo_ParentProcessIDIsValid
_lldb'.SBAttachInfo_SetEffectiveGroupID
_lldb'.SBAttachInfo_SetEffectiveUserID
_lldb'.SBAttachInfo_SetExecutable
_lldb'.SBAttachInfo_SetGroupID
_lldb'.SBAttachInfo_SetIgnoreExisting

DESIGN GOALS

There are

2268 of

these

SCRIPTABILITY - REVISITED :P

```
#test.py
import lldb
def test(debugger, command, result, internal_dict):
    target = debugger.GetSelectedTarget()
    breakpoint = target.BreakpointCreateByName("SSLWrite")
    breakpoint.SetScriptCallbackFunction('test.breakpoint_callback')

def breakpoint_callback(frame, bp_loc, dict):
    print “Hit!"

def __lldb_init_module(debugger, internal_dict):
    debugger.HandleCommand('command script add -f test.test test')
```

SCRIPTABILITY - REVISITED :P

```
#test.py
from libs import MacDbg
from libs.const import *

def breakpoint_callback(info_struct):
    print "Hit!"


dbg = Macdbg()
dbg.add_breakpoint("test", PERSISTENT, breakpoint_callback)
```

INTERESTING IDEAS

- Only debugger (we know of) using Kqueue rather than ptrace
- Lazy exception server registering
- Allows us to be non invasive to the debugged process and debug more things!

INTERESTING IDEAS

- Lateral Movement - code injection
  - Easily create code caves
  - R-X + allocation all in one step

```
code_address = dbg.inject_code("HELLO")
```

INTERESTING IDEAS

Attach and debug multiple processes at once

```
pid_list = [1222,1235,1237,1238,1245]
debuggers = []
for i in pid_list:
    tmp = MacDbg(i)
    debuggers.append(tmp)
```

INTERESTING IDEAS

- While written with Python bindings this is a debugging framework
- Easily extended to any language of your choosing
- Even easy and ready to use in everyones favorite language, C !
- Examples included.

RECAP

- Found a crash in Safari wrote a debugger
- Test debugger find crash in kernel
- Write presentation for Shmoo crash keynote
- Debug keynote crash Ida

DEMOS!

SEARCH MEMORY

- Searching multiple programs memory at once
- Attach to separate programs scan there memory space
- Programmatically determine which one is the correct process

DUMP BINARY

- Easy way to bypass packing
- Or Apples encrypted binaries!(as long as you can get around SIP)

```
program = dbg.dump_binary()
```

APPLE-PROTECTED BINARIES

- Apple encrypts some of its binaries
  - More difficult to run on non Apple hardware
    - Proprietary design - intellectual property
- Finder, Dock, LoginWindow, SystemUIServer etc…
- Easily dumped at run time :)

ASLR

- OSX has the ability to opt out of PIE(Position Independent Executable)
- Wonder which programs chose to do that?

EXAMPLES

- Show the codebase on github or bitbucket or wherever and show the directory structures the examples

TODO

- Still under development
- Needs a few things such as watchpoints, more testing, more functionality
- Some rushed code to get in before this talk
- Stop crashing the kernel and Keynote and Safari

WHAT WE USED TO GET HERE

- credit where credit is due

CREDITS

- Mac OS X Internals - To The Apple's Core - Johnathan Levin
- OSX Interals - Amit Singh
- http://opensource.apple.com
- http://lldb.llvm.org
- http://doc.geoffgarside.co.uk/kqueue/proc.html
- https://github.com/secretsquirrel/the-backdoor-factory

CREDITS

- vdb - Philosophical debugger reference
- readmem - Command line tool to read memory
- m3u - Disabling m3u in iTunes
- cmu - mach exception handling paper
- vm_read - test code of vm_allocate, vm_read, and vm_deallocate
- exception_handlers - blog post on understanding mach-o exception handlers
- exception - stackoverflow to register mach_port for exception handling in 64-bit
- base_address - Getting base address in Mac

CREDITS

- Other people that helped:
  - Gaya Thiru
  - Kenny Yee
  - Tyler Bohan
  - Chris Thompson

QUESTIONS

- More info follow us @:
  - @drraid
  - @1blankwall1

- Code will be released at
  - https://github.com/blankwall/MacDBG

PS

Hopper sucks (0 day)

GOODBYE

Keynote 0 day :p