@rantyben
So, like, I’m in Majuro…
... sailing
DRINKING...
Then Dave is all...
LONG STORY SHORT...
FUZZING OSX

AT SCALE
I PROPOSED…

- 8 weeks
- Write better instrumentation than CrashWrangler
- Run OSX on commodity hardware
- Run “shiny new tools” like AFL
- Centralised C&C
GROUND RULES

• This was new work, didn’t know how it would go
• I knew ~nothing about OSX
• I never release bugs in fuzzing talks
• Userland only
FUZZING IS A PROCESS!

1. Acquire Knowledge
2. Instrumentation
3. Delivery
4. Generation
5. Scale
ACQUIRE KNOWLEDGE
OSX

- Processes ➡ Tasks (Threads ➡ Threads)
- IPC ➡ Mach Ports
- System Calls ➡ Traps
- Libraries ➡ Frameworks
OSX

- gdb ➡ lldb
- gcc ➡ clang
- ldd, objdump, many others ➡ otool
OSX

http://uninformed.org/?v=4&a=3&t=sumry

http://web.mit.edu/darwin/src/modules/xnu/osfmk/man/

https://github.com/shoumikhin/Mach-O-Hook
ACQUIRE KNOWLEDGE
INSTRUMENTATION
INSTRUMENTATION

• Mach Exception Ports?

• “Normal” Unix tools?

• CrashWrangler?

• LLDB - C++ / SWIG API (🐍💩)?

• GDB?
CRASHWRANGLER 😢

- exc_handler ~ 1300LOC ObjC
- CrashLog.rb ~ 800LOC (reimplements half of it?)
- Moar Ruby scripts to do bucketing
- bash scripts for instrumentation
if [ -n "${CW_TIMEOUT+x}" ]; then
    echo Using timeout $CW_TIMEOUT
else
    CW_TIMEOUT=3
fi

./exc_handler "/Applications/Preview.app/Contents/MacOS/Preview" "$1" &
EXCPID=$! # get the PID for the last added background process

ruby -e "begin ; sleep $CW_TIMEOUT; Process.kill("USR1", $EXCPID) if no
# wait for exc_handler to exit, and get exit value
wait $EXCPID
EXIT_VALUE=$?

# return exit value of exc_handler
exit $EXIT_VALUE
CRASHWRANGLER 😊

- About 30 C tests to repro common crashes
- Includes OSX specific faults
  - ObjC, CoreFoundation …
GDB EXPLOITABLE

• Initially developed at CERT

• https://github.com/jfoote/exploitable

• Runs as a GDB plugin

• Another ~20 tests

• 🐍💩
LLDB API

• Actually really good! And has samples!

• http://lldb.llvm.org/python_reference/index.html

• Complete, unsweetened, 🐍 wrapper

• Tools runs AS a debugger, not IN a debugger
EXPLOITABEN*

- Steals design from Exploitable
- Steals OSX heuristics from CrashWrangler
- Steals tests from both
- More or less ground-up rewrite (🐍💩)
EXPLOITABEN

• Uses indicators, not classifications

• Tweaks to hash bucketing

• Assorted Frills

• timeouts, attach-wait, env, command triggers …
INDICATORS

• Memory patterns linked to UAF etc (0xbadbeef)

• ~50 suspicious OSX stack functions

• Access types: read, write, exec, recursion

• Heuristics on $pc, $sp

• Flags block moves
#define SIZE (1 << 30)
int main() {
    char buf[SIZE];
    char c = buf[0];
}

StopDesc:          EXC_BAD_ACCESS (code=1, address=0x7fff1fbffe30)
AvNearNull:       False
AvNearSP:         True
BadBeef:          False
Access Type:      read
Registers:        dl=0x0000000000000078
BlockMov:         False
Weird PC:         False
Weird SP:         True
Suspicious Funcs:
Illegal Insn:      False
Huge Stack:       False
```c
#include <string.h>
int main(int argc, char** argv)
{
    char buf[16];
    memset(buf, 'A', 65);
}
```
<table>
<thead>
<tr>
<th>Description</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>StopDesc:</td>
<td>signal SIGABRT</td>
</tr>
<tr>
<td>AvNearNull:</td>
<td>False</td>
</tr>
<tr>
<td>AvNearSP:</td>
<td>False</td>
</tr>
<tr>
<td>BadBeef:</td>
<td>False</td>
</tr>
<tr>
<td>Access Type:</td>
<td>&lt;not an access violation&gt;</td>
</tr>
<tr>
<td>Registers:</td>
<td></td>
</tr>
<tr>
<td>BlockMov:</td>
<td>False</td>
</tr>
<tr>
<td>Weird PC:</td>
<td>False</td>
</tr>
<tr>
<td>Weird SP:</td>
<td>False</td>
</tr>
<tr>
<td>Suspicious Funcs:</td>
<td>__stack_chk_fail</td>
</tr>
<tr>
<td>Illegal Insn:</td>
<td>False</td>
</tr>
<tr>
<td>Huge Stack:</td>
<td>False</td>
</tr>
</tbody>
</table>
FRANCIS

• Simple Go package to parse exploitaben reports
• Seems pretty reliable so far
• Used as the OSX plugin for my afl-triage tool
  • Multicore, -every mode, Cache DB …
AFL-TRIAGE DEMO
TERRY

- Very slow/lame fuzzer in Go
- Forks a radamsa server for Generation
- Invokes target under exploitaben
- Finds bugs 😭

- [https://github.com/bnagy/terry](https://github.com/bnagy/terry)
AFL (FORESHADOWING)

- AFL has its own Instrumentation
- `fork()` server + signal handler
- polices memory limits and timeouts
SUMMARY

• Good enough instrumentation

• MUCH better crash triage than I’m used to

• OMGSYMBOLS!!! SOURCE CODE!! ❤

• No dialog clickies etc

• No CPU monitoring, timeouts only
HOW WE DOIN?

• I’m in Palau
• Not drinking
• ~4 weeks into the project
• Dave is sending me motivational messages
MOTIVATION

Ben Nagy @rantyben · Feb 5
#febfast holy crap eating out is super cheap now o_0

@daveaitel
@rantyben Why are you eating when you have os x fuzzing to be doing for infiltrate?????????????
AFL

http://prelkia.deviantart.com/art/Furious-Rabbit-157796019

A FRACTAL OF GOOD DESIGN
AFL

• “American Fuzzy Lop” (it’s a rabbit)

• Revolutionary Fuzzing Tools (in historical order)
  • SPIKE
  • AFL
SRSLY?

- Compile-time instrumentation (or dynamic)
- Collect Coverage
- Dumb mutation
- Evolve files that create paths
ALL ABOUT EXECUTION!

• Coverage “bitmap”

• Forkserver

• Mutation algorithm curation

• Tokens - discovered / user supplied
cur = <COMPILE_TIME_RANDOM>;
shared_mem[cur ^ prev]++;
prev = cur >> 1;
ALL ABOUT EXECUTION!

• Effector maps

• Greedy time allocation

• Support tools

  • cmin, tmin, showmap, peruvian-were-rabbit
CMIN

- Shrink corpus to approximate minimum set
- Prefers smaller files
TMIN

- Shrink one test to minimum size and clean
- Remove blocks, see if we lose coverage
- Alphabet normalisation
- Kinda slow (obviously)
PERUVIAN WERE-RABBIT

- Take one or more crashing inputs
- Run “normal” AFL, discard non-crashes
- Great for exploring promising null derefs etc
BUT... BUT... SAGE!

- AFL makes “generation” fuzzing practical
- For virtually any target*
- With no modifications*
- With no input limits*
- With no need to write code*
AFL WEAKNESSES

• Windows / Closed source in general
• Scale / Triage
• Complex Formats / Grammars*
• Effectiveness is exponential
Fuzzers alive : 47
Total run time : 119 days, 3 hours
Total execs : 22 million
Cumulative speed : 102 execs/sec
Pending paths : 27678 faves, 292486 total
Pending per fuzzer : 588 faves, 6223 total
Crashes found : 3 locally unique
Fuzzers alive : 47
Total run time : 875 days, 4 hours
   Total execs : 167 million
Cumulative speed : 104 execs/sec
Pending paths : 41987 faves, 800020 total
Pending per fuzzer : 893 faves, 17021 total
Crashes found : 3647 locally unique
TIME TO EXPERIMENT
• Day 1: AFL supports qemu DBI for coverage 🎉😍

• Day 3: Except it doesn’t work on OSX 😢

• Day 14: Even if it did it would be too slow 😭
CORPUS DRIVEN FUZZING

- Great Corpora ➡ Much Bug
- How to acquire?
BUILDING CORPORA

• “Prospector”

• Download files from Internet

• Trace for coverage

• Select minimum files with maximum cover
PROSPECTOR

• Many people have had this idea
• I got it from Charlie
• Peach had a minset tool (but it was broken)
• AFAIK mine is the only public, “working” code 😳
PROSPECTOR ISSUES

• Real World files are not a good way to get cov
• Scraping is hard / slow / annoying
• Files are bloated with useless user data
EXISTING CORPORA

- Great if you can get them!
- http://acroeng.adobe.com/wp/?page_id=10
- https://code.google.com/p/imagetestsuite/
- ...?
AFL CORPORAE

- Generate against fast targets
- OSS parsers - different versions, options
- Minimise with cmin / tmin
- (Complex Grammars need help)
SYNTHESIS
CDF VS PDF

• AFL side - wrote a lexer to tokenise PDF
  • https://github.com/bnagy/pdfllex
  • Extracted ~1500 tokens
  • Curated by hand
CDF VS PDF

• Bugged Icamtuf to add a “fixup” feature
• AFL can now invoke a custom .so for each test
• Wrote a shim .so that talks to a unix socket
• https://github.com/bnagy/aflfix
PDF FIXUPS

• Wrote a Go fixer to patch \texttt{startxref} offsets

• Deeper coverage, moar* bugs
<table>
<thead>
<tr>
<th>Process Timing</th>
<th>Overall Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run time: 9 days, 4 hrs, 3 min, 48 sec</td>
<td>Cycles done: 3</td>
</tr>
<tr>
<td>Last new path: 0 days, 0 hrs, 8 min, 48 sec</td>
<td>Total paths: 7272</td>
</tr>
<tr>
<td>Last uniq crash: 0 days, 11 hrs, 42 min, 55 sec</td>
<td>Uniq crashes: 161</td>
</tr>
<tr>
<td>Last uniq hang: 4 days, 17 hrs, 15 min, 39 sec</td>
<td>Uniq hangs: 500+</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cycle Progress</th>
<th>Map Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now processing: 7264 (99.89%)</td>
<td>Map density: 10.9k (16.59%)</td>
</tr>
<tr>
<td>Paths timed out: 0 (0.00%)</td>
<td>Count coverage: 5.44 bits/tuple</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Stage Progress</th>
<th>Findings in Depth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Now trying: splice 13</td>
<td>Favorred paths: 550 (7.56%)</td>
</tr>
<tr>
<td>Stage execs: 1355/3360 (40.33%)</td>
<td>New edges on: 1070 (14.71%)</td>
</tr>
<tr>
<td>Total execs: 83.7M</td>
<td>Total crashes: 1330 (161 unique)</td>
</tr>
<tr>
<td>Exec speed: 93.33/sec (slow!)</td>
<td>Total hangs: 72.0k (500+ unique)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fuzzing Strategy Yields</th>
<th>Path Geometry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit flips: n/a, n/a, n/a</td>
<td>Levels: 29</td>
</tr>
<tr>
<td>Byte flips: n/a, n/a, n/a</td>
<td>Pending: 5845</td>
</tr>
<tr>
<td>Arithmetics: n/a, n/a, n/a</td>
<td>Pend fav: 3</td>
</tr>
<tr>
<td>Known ints: n/a, n/a, n/a</td>
<td>Own finds: 7271</td>
</tr>
<tr>
<td>Dictionary: n/a, n/a, n/a</td>
<td>Imported: n/a</td>
</tr>
<tr>
<td>Havoc: 4936/44.1M, 2496/38.8M</td>
<td>Variable: 34</td>
</tr>
<tr>
<td>Trim: 0.39%/723k, n/a</td>
<td>[cpu: 78%]</td>
</tr>
<tr>
<td>Process Timing</td>
<td>Overall Results</td>
</tr>
<tr>
<td>-------------------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td><strong>run time</strong></td>
<td><strong>cycles done</strong>: 3</td>
</tr>
<tr>
<td>9 days, 4 hrs, 3 min, 31 sec</td>
<td><strong>total paths</strong>: 7101</td>
</tr>
<tr>
<td><strong>last new path</strong></td>
<td><strong>uniq crashes</strong>: 156</td>
</tr>
<tr>
<td>0 days, 0 hrs, 14 min, 22 sec</td>
<td><strong>uniq hangs</strong>: 500+</td>
</tr>
<tr>
<td><strong>last uniq crash</strong></td>
<td></td>
</tr>
<tr>
<td>0 days, 2 hrs, 53 min, 29 sec</td>
<td></td>
</tr>
<tr>
<td><strong>last uniq hang</strong></td>
<td></td>
</tr>
<tr>
<td>4 days, 15 hrs, 48 min, 13 sec</td>
<td></td>
</tr>
<tr>
<td>Cycle Progress</td>
<td>Map Coverage</td>
</tr>
<tr>
<td><strong>now processing</strong></td>
<td><strong>map density</strong>: 10.9k (16.68%)</td>
</tr>
<tr>
<td>5238* (73.76%)</td>
<td><strong>count coverage</strong>: 5.22 bits/tuple</td>
</tr>
<tr>
<td>paths timed out</td>
<td><strong>findings in depth</strong></td>
</tr>
<tr>
<td>0 (0.00%)</td>
<td><strong>favored paths</strong>: 584 (8.22%)</td>
</tr>
<tr>
<td>Stage Progress</td>
<td><strong>new edges on</strong>: 1088 (15.32%)</td>
</tr>
<tr>
<td><strong>now trying</strong></td>
<td><strong>total crashes</strong>: 3480 (156 unique)</td>
</tr>
<tr>
<td>splice 17</td>
<td>61.7k (500+ unique)</td>
</tr>
<tr>
<td>Stage execs</td>
<td><strong>path geometry</strong></td>
</tr>
<tr>
<td>900/1680 (53.57%)</td>
<td><strong>levels</strong>: 48</td>
</tr>
<tr>
<td>Total execs</td>
<td><strong>pending</strong>: 5605</td>
</tr>
<tr>
<td>87.4M</td>
<td><strong>pend fav</strong>: 1</td>
</tr>
<tr>
<td>Exec speed</td>
<td><strong>own finds</strong>: 7100</td>
</tr>
<tr>
<td>124.4/sec</td>
<td><strong>imported</strong>: n/a</td>
</tr>
<tr>
<td>Fuzzing Strategy Yields</td>
<td><strong>variable</strong>: 144</td>
</tr>
<tr>
<td>Bit flips</td>
<td></td>
</tr>
<tr>
<td>n/a, n/a, n/a</td>
<td></td>
</tr>
<tr>
<td>Byte flips</td>
<td></td>
</tr>
<tr>
<td>n/a, n/a, n/a</td>
<td></td>
</tr>
<tr>
<td>Arithmetics</td>
<td></td>
</tr>
<tr>
<td>n/a, n/a, n/a</td>
<td></td>
</tr>
<tr>
<td>Known Ints</td>
<td></td>
</tr>
<tr>
<td>n/a, n/a, n/a</td>
<td></td>
</tr>
<tr>
<td>Dictionary</td>
<td></td>
</tr>
<tr>
<td>n/a, n/a, n/a</td>
<td></td>
</tr>
<tr>
<td>Havoc</td>
<td></td>
</tr>
<tr>
<td>4800/46.0M, 2456/40.8M</td>
<td></td>
</tr>
<tr>
<td>Trim</td>
<td></td>
</tr>
<tr>
<td>0.30%/544k, n/a</td>
<td></td>
</tr>
</tbody>
</table>
PDF SHRINKIES

• Partial parser, built on top of my lexer
• Blindly truncate PDF stream objects
• Fix up xref section so indirect refs all work
• “Real Enough” so parsers get deep coverage
CDF RESULTS

• AFL corpus, after many revisions

• Acroeng samples run through pdfshrink

• Feed to terry

• Fuzz qlmanage (actual Apple software)
CDF RESULTS

• Terry results > dumb AFL results
• AFL 50% faster than Terry
• The concept works!
OSX DELIVERY TIPS

• `xattr -c` can clear quarantine metadata

• If that fails, use `open -a Preview foo.pdf`

• Mount ramdisks like:

  ```bash
diskutil erasevolume HFS+ 'ramdisk'
  \`
  \`
  \`
  `hdiutil attach -nomount ram://1048576``
OSX DELIVERY TIPS

• Try `MallocScribble/MallocGuardEdges`

• Disable CrashReporter (AFL will remind you)
OSX DELIVERY TIPS

- Magic compatibility mode for the C++ linker
  
  export CFLAGS="-mmacosx-version-min=10.4"

- Force content type for qlmanage
  
  -c com.adobe.pdf
OSX DELIVERY TIPS

- run `DevToolsSecurity -enable`
I AM TOTALLY CALM

Ben Nagy @rantyben · Mar 14
(I can feel @daveaitel judging me for spending four hours writing lexers instead of Applescript dialog clickers)
MOTIVATION INTENSIFIES

@daveaitel @daveaitel · Mar 16
We are really pretty hard on the dry runs. I spend a lot of time imagining the font choices from the back of the room.

@Ben Nagy
@rantyben
@daveaitel YOU ARE NOT HELPING SHUT UP SHUT UP

RETWEET
1
HOW WE DOIN?

• I’m in Adelaide now.

• 6 weeks in. Total panic.

• Biggest time sinks:
  • Writing solid instrumentation
  • Doing multi-day benchmarks
  • Tooling to handle PDF
VIRTUALIZING OSX

• No leetness required!

• http://www.contrib.andrew.cmu.edu/~somlo/OSXKVM/

• http://blog.ostanin.org/2014/02/11/playing-with-mac-os-x-on-kvm/
HOW IT WAS DONE
(By Smarter People)

• Upstream KVM patches
  • ACPI issues, unsupported instructions
• Upstream QEMU patches
  • Patch SeaBIOS
• Add SMC chip emulation
HOW IT WAS DONE
(BY SMARTER PEOPLE)

- Chameleon bootloader is the final missing piece!
  - This is real OSX from real install media
  - (the bootloader is probably still backdoored)
OSX VIRT TIPS

• Use OSX VNC Client
  • (Finder ⌘K ➡ vnc://ip.addr.of.server:5900)

• Use OSX VNC Server
  • System Preferences ➡ Screen Sharing
OSX VIRT TIPS

• Use Chimera

• DO NOT INSTALL on your Macbook! 😳

• Assorted Useful Features (autostart, better res…)

• org.chameleon.Boot.plist ➡ CASE SENSITIVE
OSX VIRT TIPS

• Download a real Installer from App Store
  • Recovery Partition image didn’t work for me

• Convert to an ISO:
  • ~somlo/OSXKVM/MakelInstallDVD.sh

• Easy VNC port forwarding using -net user
  hostfwd=tcp:172.16.216.135:5901-:5900
Create a Computer Account

Fill out the following information to create your computer account.

Full name: David Aitel
Account name: zappy
This will be the name of your home folder.
Password: ************
Hint: near Indonesia
Require password to unlock screen

Options:
- Set time zone based on current location
- Send Diagnostics & Usage data to Apple

About Diagnostics and Privacy...
All virtualisation research was carried out on genuine Apple® hardware using properly licensed software.

This research was undertaken for educational purposes only.

This presentation does not constitute legal advice.

The adjective “pavonine” meaning “of or like a peacock” is from the Latin *pavoninus*. 
SCALING OSX?

• Planned on building a massive OSX Cloud
• Decided it was a bad idea
• Scale by CDF instead
WHY CDF?

- **pdftoppm** is “slow” for AFL - 160 tests/sec
- **ghostscript** even slower ~6
- Preview is about 0.6
- **CDF** is a force multiplier
WHY CDF?

- 10 image parsing bugs in IE
- WITHOUT EVER FUZZING IE
Corpus Driven Fuzzing! so ethical!

GNU
- Linux cluster
  - AFL Park
    - Poppler / gs / pdfinfo
  - cmin / tmin

WORKING CORPUS
- Continually evolving
- pdfshrink

- Samples / Tests / Ange

Mostly uses OSS crashes. Try IBB for pennies

OSX Crashes

- terry
  - manage / Preview

- OS Repro / Sandbox Team
  - Organically Drawn! Gluten Free!
HOW WE DOIN?  

• I’m back in Majuro now
• Liver hurts from SySCAN
• Biggest failure was lack of C&C tooling
• Biggest success was solidifying CDF approach
• Approach seems viable
FUTURE WORK

http://glitched9700.deviantart.com/art/Broken-Robot-II-82773491
FUTURE WORK

• AFL Clouds

• Designed lots of stuff, figured it was niche

• NSQ, Docker, Kubernetes blah blah blah?

• Ansible, Puppet blah blah?
FUTURE WORK

• Finish full triage framework (queues, DB, etc)
  • Unify indicators with GDB plugin
• Write ObjC apps that use Apple frameworks?
  • PrivateFrameworks/CorePDF
FUTURE WORK

• OSX Instrumentation ( OSS )
  • New LLVM passes being discussed RIGHT NOW

• OSX Instrumentation ( Closed )
  • New results with dyninst

• Intel PT on Broadwell CPUs?
SOFTWARE
• aflfix

• crashwalk (includes afl-triage)

• francis (includes exploitaben.py)

• gootool (toy Mach-O disassembler based on Capstone engine)

• pdflex (includes pdftok, pdfshrink)

• terry
SOme links

http://web.mit.edu/darwin/src/modules/xnu/osfmk/man/ Mach IPC Interface
https://github.com/shoumikhin/Mach-O-Hook Mach-O Hooking
http://en.wikipedia.org/wiki/Mach_%28kernel%29
http://en.wikipedia.org/wiki/XNU
http://brendanzagaeski.appspot.com/0004.html minimal pdf
http://www.ghostscript.com/doc/9.15/Use.htm how to use ghostscript
http://events.linuxfoundation.org/sites/events/files/slides/lcna13_kleen.pdf Intel PT
https://github.com/jfoote/exploitable gdb exploitable
http://lldb.llvm.org/python_reference/index.html
http://ho.ax/tag/lldb/
http://www.opensource.apple.com/source/xnu/xnu-1456.1.26/osfmk/mach/exception_types.h?txt # exception types
https://code.google.com/p/honggfuzz/source/browse/trunk/mac/arch_mac.c
http://llvm.org/svn/llvm-project/lldb/trunk/examples/python/disasm.py
http://lldb.llvm.org/python_reference/index.html
https://developer.apple.com/library/mac/documentation/Performance/Conceptual/ManagingMemory/Articles/MallocDebug.html
MallocScribble
http://blog.ostanin.org/2014/02/11/playing-with-mac-os-x-on-kvm/
http://www.contrib.andrew.cmu.edu/~somlo/OSXKVM/