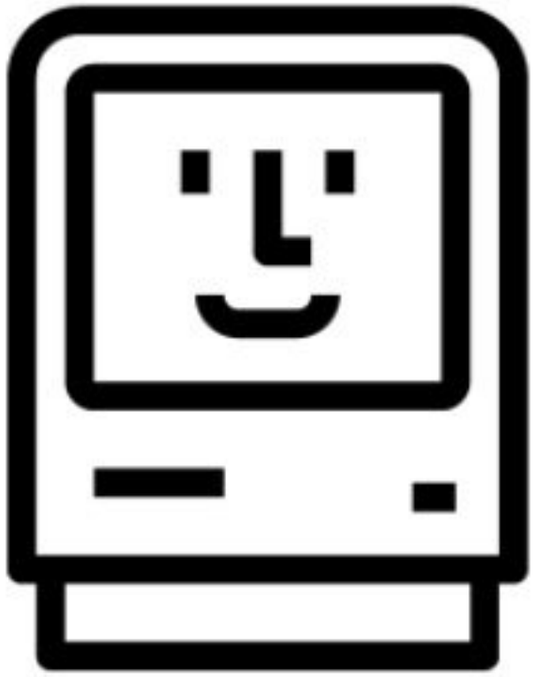
Summer of Fuzz



Jeremy Brown, August 2021
Defcon AppSec Village + HITB Singapore

Agenda

I. Intro
II. Walkthrough
    A. Debugging Tools
    B. SIP and App Sandbox
    C. Crash Reporting
    D. Sleep ‘n SSH
    E. Monitoring Process Execution
    F. Enumerating Handlers
    G. Clients and Network Services
III. Fuzzing
    A. CLI/GUI Applications
    B. Network Clients and Servers
    C. Bugs
IV. Conclusion

whoami

- Mostly interested in bug hunting, fuzzing, offensive security these days
- Previously doing security stuff @ MSFT, AMZN, NVDA, CRM
  - Breaking stuff is the best
  - Native code, web services, cloud, containers, etc
  - Attacking products and services and helping get them in better shape before release

Intro

- Didn’t know much about Mac security before this research

Image Credit
https://www.computerhistory.org/timeline/1984/

Intro

- I’ve been fuzzing for a while now
  - There’s so many advances these days, especially with AFL++ and friends
- But wanted to look at it from a different angle
  - Forget a lot of what I knew, respectfully forgo conventional wisdom, etc
  - Build something from scratch that serves a purpose
- Discussing various tricks and tooling for fuzzing userland on Mac

Intro

- Looking mostly at core, default userland applications on OS X and OS 11
  - CLI / GUI apps
  - Network clients
  - Network servers
- How to setup the environment for debugging, enumerating targets and ./
  - Not for scaling a 1,000,000 iPhone fuzzing farm
  - Most of this stuff you can do at home with a Macbook

Related Prior Work

- Ben Nagy's stuff
  - https://github.com/bnagy/slides/blob/master/OSXScale.pdf
  - https://github.com/bnagy/francis/tree/master/exploitaben
- CrashWrangler
  - https://github.com/ant4g0nist/crashwrangler
- Inspiration to get back into fuzzing
  - https://tmpout.sh/1/5.html

Debugging

- Xcode Tools
  - Headless installation script
  - Enable developer mode
    - `sudo DevToolsSecurity -enable`

References
https://github.com/timsutton/osx-vm-templates/blob/master/scripts/xcode-cli-tools.sh

Debugging

- Guard Malloc
  - `DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib target args`
    - `GuardMalloc[x-82750]: Allocations will be placed on 16 byte boundaries.`
    - `GuardMalloc[x-82750]:  - Some buffer overruns may not be noticed.`
    - `GuardMalloc[x-82750]:  - Applications using vector instructions (e.g., SSE) should work.`
    - `GuardMalloc[x-82750]: version 064544.67.1`
    - `Process 82750 stopped`
    - `* thread #1, queue = 'com.apple.main-thread', stop reason = EXC_BAD_ACCESS)`
    - `libsystem_platform.dylib`_platform_memmove`

"By default, the returned address for the allocation is positioned such that the end of the allocated buffer is at the end of the last page, and the next page after that is kept unallocated. Thus, accesses beyond the end of the buffer cause a bad access error immediately. When memory is freed, libgmalloc deallocates its virtual memory, so reads or writes to the freed buffer cause a bad access error..."

Reference
https://www.unix.com/man-page/osx/3/libgmalloc/

Debugging

- LLDB
  - Automate crash triage and produce bucketing data

```
lldb -o "target create `which some-binary`" -o "settings set target.env-vars DYLD_INSERT_LIBRARIES=/usr/lib/libgmalloc.dylib" -o "run arg1 arg2" -o "bt" -o "reg read" -o "dis -s \$pc-32 -c 24 -m -F intel" -o "quit"
```

SIP

- System Integrity Protection
  - Restricts even the root user
  - Only signed processes can modify /System, default Apps, select a startup disk, etc
- Disable it for “free range” debugging

Reference
https://support.apple.com/en-us/HT204899

SIP

- Physical machine
  - Reboot into recovery mode (CTRL+R)
  - csrutil disable or csrutil enable *--without debug*

Reference
https://developer.apple.com/documentation/security/disabling_and_enabling_system_integrity_protection

SIP

- VMware Fusion
  - OS X 10
    - Reboot and CTRL+R, or….
    - VM Settings -> Startup Disk and hold down the Option key
      - “Restart to Firmware” will appear
    - Boot Manager -> Boot from the secondary added disk
    - Enter Recovery Mode, Utilities -> Terminal -> “csrutil disable” and reboot

References

https://communities.vmware.com/t5/VMware-Fusion-Discussions/Can-t-boot-into-recovery-partition-on-macOS-11-Big-Sur/td-p/2298419
https://apple.stackexchange.com/questions/415086/how-to-disable-sip-when-big-sur-is-installed-in-a-vmware-fusion-player-virtual-m

SIP

- VMware Fusion
  - OS X 11
    - Create dummy VM with Mac Installer aka “Install Mac OS Big Sur”
    - Start the VM and immediately stop it
    - Go to the real VM and add the dummy disk “Temporary Installation Source Disk.vmdk”
    - Delete the dummy VM, then...
    - Go to VM Settings -> Startup Disk and hold down the Option key
      - “Restart to Firmware” will appear
    - Boot Manager -> Boot from the secondary added disk
    - Enter Recovery Mode, Utilities -> Terminal -> “csrutil disable” and reboot

References

https://communities.vmware.com/t5/VMware-Fusion-Discussions/Can-t-boot-into-recovery-partition-on-macOS-11-Big-Sur/td-p/2298419

https://apple.stackexchange.com/questions/415086/how-to-disable-sip-when-big-sur-is-installed-in-a-vmware-fusion-player-virtual-m

App Sandbox

- “App Sandbox is an access control technology provided in macOS, enforced at the kernel level. It is designed to contain damage to the system and the user’s data if an app becomes compromised.”

References

https://developer.apple.com/library/archive/documentation/Security/Conceptual/AppSandboxDesignGuide/AboutAppSandbox/AboutAppSandbox.html

App Sandbox

- Let’s say mutated files placed in /tmp are passed to the app during execution
  - "The file ... couldn't be opened because you don't have permission to view it"
- Check the logs
  - log show --style syslog --last boot --predicate 'process == "kernel" AND eventMessage CONTAINS[c] "Sandbox"' | tail
    - localhost kernel[0]: (Sandbox) Sandbox: com.apple.BKAgen(628) deny(1) file-read-data /private/tmp/fuzz_cyhvkvha.epub

App Sandbox

- So where can we put test cases?
  - ~/Library/Containers/<app-bundle-id>
- Passing files directly to the Books app
  - ~/Library/Containers/com.apple.iBooksX/Data/test.epub
- Can still double click to open an .epub file from ~/Downloads
  - But for fuzzing, store test cases in the local app folder

Crash Reporting

- Disabling ReportCrash
  - launchctl unload -w /System/Library/LaunchAgents/com.apple.ReportCrash.plist
  - sudo launchctl unload -w /System/Library/LaunchDaemons/com.apple.ReportCrash.Root.plist

ReportCrash analyzes crashing processes and saves a crash report to disk. A crash report contains information that can help a developer diagnose the cause of a crash. ReportCrash also records the identity of the crashing process and the location of the saved crash report in the system.log and the ASL log database.

References

https://ss64.com/osx/reportcrash.html

https://discussions.apple.com/thread/2785409?answerId=13350313022#13350313022

Crash Reporting

- But, enabling it can also help when fuzzing with less visibility for some targets
  - launchctl load -w /System/Library/LaunchAgents/com.apple.ReportCrash.plist
- Let's say we're fuzzing an UDP server
  - If we can attach or run it in a debugger and catch crashes, that's fine
  - Otherwise, we can monitor ~/Library/Logs/DiagnosticLogs for crashes files for the process

Reference
https://ss64.com/osx/reportcrash.html

Sleep

- ...is important for humans
  - But when fuzzing on Mac, we need **it** to sleep… less
- Disable sleep modes while fuzzing sessions are running
  - systemsetup -setsleep Never
  - pmset, System Preferences, etc
  - KeepingYouAwake
    - Can set it to automatically activate on execution via URI or `defaults` command
  - These seem to work with local Terminal sessions, but need more adjustments for SSH…

References
https://github.com/newmarcel/KeepingYouAwake

SSH

- Fuzzing GUIs over SSH instead of the physical / desktop session
  - Try to keep SSH from timing out
  - “I’m alive” console pings built into the fuzzer
- Avoid interruptions like `client_loop: send disconnect: Broken pipe`
  - sshd_config
    - TCPKeepAlive Yes
    - ClientAliveInterval 0
    - ClientAliveCountMax 0

Monitoring Process Execution

- sudo newproc.d | grep -Ev ‘stuff-we-dont-care-about’
  - ls
  - csrutil status // checking if SIP is disabled so this can actually work
  - xpcproxy org.cups.cupsd
  - /usr/sbin/cupsd -l // turned on Printer Sharing
  - /bin/launchctl unload -w /System/Library/LaunchDaemons/com.apple.smbd.plist // turned off File Sharing
  - /usr/bin/killall -HUP netbiosd
  - /usr/bin/killall -HUP smbd
  - /System/Library/CoreServices/RemoteManagement/screensharingd.bundle/Contents/MacOS/screensharingd
  - /System/Library/CoreServices/RemoteManagement/ScreensharingAgent.bundle/Contents/MacOS/ScreensharingAgent
  - /System/Library/CoreServices/RemoteManagement/AppleVNCServer.bundle/Contents/Support/VNCPrivilegeProxy
  - /System/Applications/App Store.app/Contents/MacOS/App Store // opened the Appstore

References

https://opensource.apple.com/source/dtrace/dtrace-168/DTTk/Proc/newproc.d.auto.html

Enumerating Handlers

- SwiftDefaultApps
  - Get URI handlers
    - ./swda getSchemes
      - addressbook /System/Applications/Contacts.app
      - afp /System/Library/CoreServices/Finder.app
      - apconfig /System/Applications/Utilities/AirPort Utility.app
      - applefeedback /System/Library/CoreServices/Applications/Feedback Assistant.app
      - applenews /System/Applications/News.app
      - applescript /System/Applications/Utilities/Script Editor.app
      - cifs /System/Library/CoreServices/Finder.app
      - cloudphoto /System/Applications/Photos.app
      - daap /System/Applications/TV.app
      - dict /System/Applications/Dictionary.app
      - facetime /System/Applications/FaceTime.app
      - fb /System/Library/CoreServices/AddressBookUrlForwarder.app
      - file /System/Library/CoreServices/Finder.app
      - …
    - ./swda getSchemes | wc -l
      - **101**

References

https://github.com/Lord-Kamina/SwiftDefaultApps

Enumerating Handlers

- SwiftDefaultApps
  - Get file handlers
    - ./swda getUTIs | grep -Ev "No application set"
      - com.adobe.encapsulated-postscript /System/Applications/Preview.app
      - com.adobe.flash.video /System/Applications/QuickTime Player.app
      - com.adobe.pdf /System/Applications/Preview.app
      - com.adobe.photoshop-image /System/Applications/Preview.app
      - com.adobe.postscript /System/Applications/Preview.app
      - com.adobe.raw-image /System/Applications/Preview.app
      - com.apple.addressbook.group /System/Applications/Contacts.app
      - com.apple.applescript.script /System/Applications/Utilities/Script Editor.app
      - com.apple.archive /System/Library/CoreServices/Applications/Archive Utility.app
      - …
    - ./swda getUTIs | grep -Ev "No application set" | wc -l
      - **420**

References
https://github.com/Lord-Kamina/SwiftDefaultApps

Enumerating Handlers

- But this doesn't necessarily tell you the file extensions
  - For example **com.apple.applescript.script** -> /System/Applications/Utilities/Script Editor.app
  - It's the default application for opening **.scpt** files
- Also check ss64 as it has a great list of potential targets
  - https://ss64.com/osx/

Enumerating Handlers

- Another way to dig around for file attack surface is asking simple questions
  - What files types are already included on the filesystem?
    - `find / -type f -name '*.*' | sed 's|.*\.||' | sort -u | sed '/^.\{4\}./d' > file-types.txt`
    - `find / -name \*.eps -exec cp {} eps-files \;`
  - What mac apps might open file extension .eps?
    - https://www.google.com/search?q=%22mac%22+%22eps+file%22
- Map those to apps by searching or correlating file handler information
  - com.adobe.encapsulated-postscript -> /System/Applications/Preview.app

Enumerating Handlers

- Did you know the “notepad equivalent” on macOS parses doc and rtf files?
  - com.microsoft.word.doc /System/Applications/TextEdit.app
  - public.rtf /System/Applications/TextEdit.app
  - ...

Enumerating Handlers

- Some “files” are directories
  - Calendar -> Export -> Calendar Archive… produces a .icbu file
  - $ file test.icbu
    - test.icbu: directory
  - Double click it on Mac, it opens like a file, acts like a file, but not actually a file
- This makes fuzzing such targets more difficult
  - Not simply mutating a file, but opening a directory and modifying **the right files**

Enumerating Network Processes

- dtrace -n 'syscall::recv*:entry { printf("-> %s (pid=%d)", execname, pid); }' >> recv.log
  - *wait a while, then ctrl+c*
- sort -u recv.log > procs.txt
- head procs.txt
  - recvmsg:entry -> adprivacyd (pid=48835)
  - recvmsg:entry -> amsengagementd (pid=594)
  - recvmsg:entry -> appstoreagent (pid=48874)
  - recvmsg:entry -> apsd (pid=108)
  - recvmsg:entry -> familycircled (pid=48775)
  - recvmsg:entry -> mDNSResponder (pid=185)
  - recvmsg:entry -> remindd (pid=70710)
  - ….

Enumerating Network Services

- “Just turn everything on and check netstat / lsof”
  - netstat -an | grep LISTEN
    - tcp4 0 0 *.631 *.* LISTEN
    - tcp4 0 0 *.56352 *.* LISTEN
    - tcp4 0 0 *.3031 *.* LISTEN
    - tcp4 0 0 *.445 *.* LISTEN
    - tcp4 0 0 *.88 *.* LISTEN
    - tcp46 0 0 *.3283 *.* LISTEN
    - tcp4 0 0 *.5900 *.* LISTEN
    - tcp4 0 0 127.0.0.1.49742 *.* LISTEN
    - tcp4 0 0 *.22 *.* LISTEN
    - tcp4 0 0 127.0.0.1.8021 *.* LISTEN
    - tcp6 0 0 ::1.8021 *.* LISTEN

Enumerating Network Services

- “Just turn everything on and check netstat / lsof”
  - $ lsof -i | grep LISTEN
    - launchd 1 root 7u IPv6 0xa13e8b40a862b75b 0t0 TCP *:ssh
    - launchd 1 root 9u IPv6 0xa13e8b409c221dbb 0t0 TCP *:eppc
    - launchd 1 root 18u IPv6 0xa13e8b409c22241b 0t0 TCP *:rfb
    - launchd 1 root 53u IPv6 0xa13e8b40a862b0fb 0t0 TCP *:microsoft-ds
    - kdc 121 root 5u IPv6 0xa13e8b409c222a7b 0t0 TCP *:kerberos
    - screensha 497 root 4u IPv4 0xa13e8b409c229333 0t0 TCP *:rfb
    - ARDAgent 535 test 9u IPv6 0xa13e8b40a862c41b 0t0 TCP *:net-assistant
    - ODSAgent 44385 root 3u IPv6 0xa13e8b40a862bdbb 0t0 TCP *:51656
    - cupsd 47781 root 5u IPv6 0xa13e8b409c2210fb 0t0 TCP localhost:ipp

Fuzzing

- AFL
  - brew install afl-fuzz
  - afl-fuzz -n -i pdf -o crashes *yolo* @@
    - [-] PROGRAM ABORT : Program *'yolo'* is not a 64-bit Mach-O binary
- AFL++
  - git clone && make distrib
  - afl-fuzz -n -i pdf -o crashes *yolo* @@
    - [+] All set and ready to roll!

Fuzzing

- What about GUI apps?
  - afl-fuzz -n -i ttc -o crashes -d -t 10000 -V 2 "/System/Applications/Font Book.app/Contents/MacOS/Font Book" @@
    - [-] PROGRAM ABORT : All test cases time out, giving up!
- Network fuzzing? Windows?
  - Respect to WinAFL and AFLNet, but they have their limitations too

Fuzzing

- Litefuzz
  - “Just works” across the three major operating systems
    - Linux, Mac, Windows
  - Supports file and network fuzzing, CLI or GUI, even interactive network GUIs (eg. Postman)
  - Automatic crash triage and diffing
  - Most useful for fuzzing closed source applications, clients and servers
  - Built for bug hunters and does some neat things in an unorthodox fashion

Fuzzing

- Breaks a lot of the rules for modern fuzzing
  - Doesn't do instrumentation
  - Not optimized for speed, execs/sec, etc
  - Lacks native devops integration support
  - **But it does find bugs**

Fuzzing

- But then how do I know if the fuzzer is doing a good job?

Fuzzing

- Mac intricacies
  - Some GUI apps like...
    - Unique filenames
    - Files to have the right extension
    - Reading files within the sandbox
      - ~/Library/Containers/com.apple.Safari/Data
      - ~/Library/Containers/com.apple.iBooksX/Data/…
  - Also, some are hard to fuzz directly, eg. passing a file as command line arg
    - But many work just fine this “classic” way

Targeting Applications

- Step 1
  - Select a CLI or GUI target
    - iBooks, Font Book, pkgutil
- Step 2
  - Collect test files
    - Also keep in mind that some apps won't open the file if it's not a known extension and may prefer to only open files from certain locations
- Step 3
  - If the target doesn't exit on it's own, measure a reasonable timeout
- Step 4
  - Start fuzzing

iBooks

- litefuzz
  - -l
  - -c "/System/Applications/Books.app/Contents/MacOS/Books FUZZ"
  - -i files/epub
  - -o crashes/ibooks
  - -t /Users/test/Library/Containers/com.apple.iBooksX/Data/tmp *// use this special temp directory*
  - -x 10 *// max running time*
  - -n 100000
  - -ez

Font Book

- litefuzz
  - -l
  - -c "/System/Applications/Font Book.app/Contents/MacOS/Font Book FUZZ"
  - -i input/fonts
  - -o crashes/font-book
  - -x 2
  - -n 500000
  - -ez *// recycle any found crashes as new fuzzing inputs*

pkgutil

- litefuzz
  - -I
  - -c "pkgutil --expand FUZZ /tmp/test"
  - -i input/pkg
  - -n 1000000
  - -ez

Note: if it's just a console app parsing files, you can use AFL in non-instrumented mode on it too

pkgutil

- cat /tmp/litefuzz/out
  - GuardMalloc[pkgutil-20655]: Allocations will be placed on 16 byte boundaries.
  - GuardMalloc[pkgutil-20655]: - Some buffer overruns may not be noticed.
  - GuardMalloc[pkgutil-20655]: - Applications using vector instructions (e.g., SSE) should work.
  - GuardMalloc[pkgutil-20655]: version 064544.67.1
  - Entity: line 1: parser error : Extra content at the end of the document
  - <?xm
  - ^
  - Could not open package for expansion: .../fuzz_lvfiyzax.pkg

Minimizing Crashes

EXC_BAD_ACCESS_SIGSEGV_7fffxxxxxxxx_2b4e8f57a43e7c77bxxxx

[+] attempting to repro the crash...

[+] repro OK

[+] starting minimization

@ 6276/6276 (0 new crashes, **8618 -> 6277 bytes**, ~0:00:00 remaining)

[+] reduced crash @ pc=7fffxxxxxxxx to 6277 bytes

Minimizing Crashes

EXC_BAD_ACCESS_SIGSEGV_7fffxxxxxxxx_2b4e8f57a43e7c77bxxxx

@ 6276/6276 (0 new crashes, **8618 -> 6277 bytes**….)

[+] *supermin* activated, continuing…

@ 5106/5106 (0 new crashes, **6277 -> 5106 bytes**....)

[+] reduced crash @ pc=7fff203b6588 to 5106 bytes

….

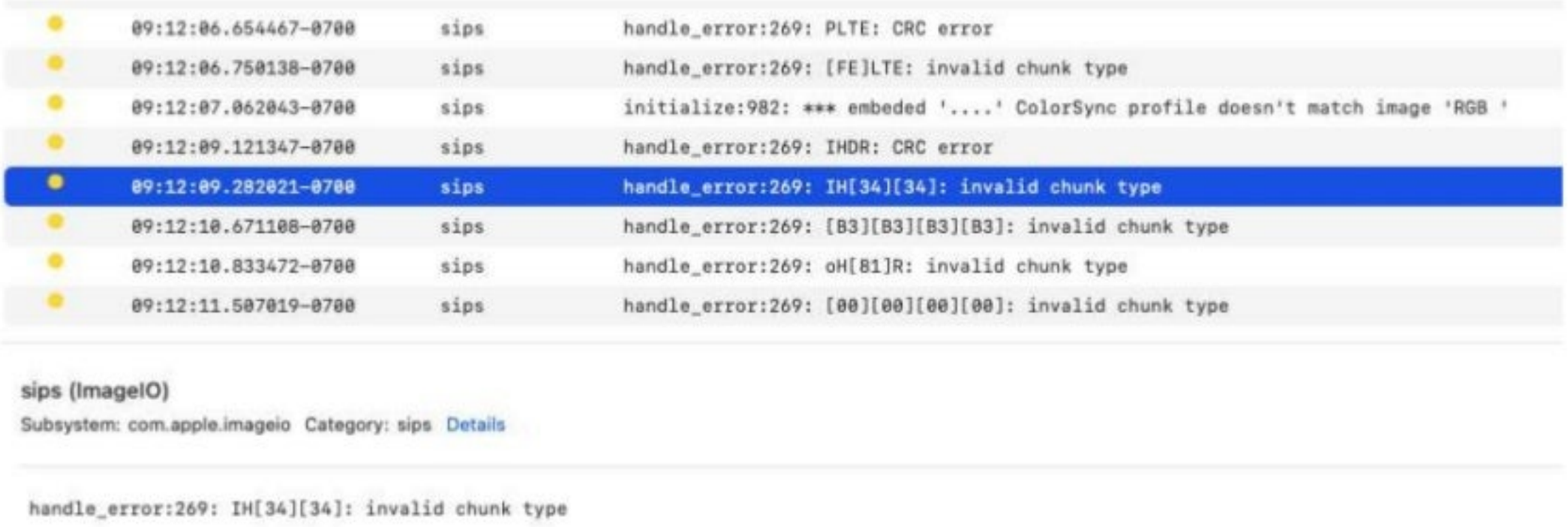
@ 3958/3958 (0 new crashes, **3972 -> 3958 bytes**….)

[+] reduced crash @ pc=7fff203b6588 to 3958 bytes

[+] achieved maximum minimization @ 3958 bytes

Console

- Console app gives you additional visibility on the target application



Targeting Clients

- Step 1
  - Select a network client
    - smbutil
    - CUPS (lpadmin)
- Step 2
  - Capture and export the protocol data from the packets exchanged
    - eg. valid responses from servers
- Step 3
  - Fuzz!

smbutil

- litefuzz
  - -lk
  - -c "smbutil view smb://localhost:4455"
  - -a tcp://localhost:4455
  - -i input/mac-smb-resp
  - -p
  - -n 100000
  - -z *// malloc debugger*

CUPS

- litefuzz
  - -lk
  - -c "lpadmin -h localhost:6631 -p test -E -v ipp://123" *// connect to server listening on port 6631*
  - -a tcp://localhost:6631 *// listen on port 6631 (don't interfere with real CUPS server on port 631)*
  - -i input/cups
  - -o crashes/cups
  - -p
  - -x 2
  - -n 100000
  - -ez

Targeting Servers

- Step 1
  - Select network server(s)
    - Remote Management aka **ARDAgentd** aka VNC'ish server
      - /System/Library/CoreServices/RemoteManagement/ARDAgent.app/Contents/MacOS/ARDAgent
    - Screen Sharing aka **screensharingd** aka VNC server
      - /System/Library/CoreServices/RemoteManagement/screensharingd.bundle/Contents/MacOS/screensharingd
    - CD/DVD Sharing aka **ODSAgent**
      - /System/Library/CoreServices/ODSAgent.app/Contents/MacOS/ODSAgent
- Step 2
  - Capture and export the protocol data flying across the on the network
- Step 3
  - Fuzz!

References

http://lockboxx.blogspot.com/2019/07/macos-red-teaming-206-ard-apple-remote.html

ARDAgent

- lsof -i | grep *:net-assistant
  - ARDAgent 82822 user 6u IPv6 0x93eb6bf24ae78f55 0t0 UDP *:net-assistant // :3283
  - ARDAgent 82822 user 7u IPv4 0x93eb6bf24ae748d5 0t0 UDP *:net-assistant
  - ARDAgent 82822 user 9u IPv6 0x93eb6bf249f575a5 0t0 TCP *:net-assistant (LISTEN)
- Also
  - **-rwsr-xr-x** 1 root wheel 2033200 Jan 1 2020 /System/Library/CoreServices/RemoteManagement/ARDAgent.app/Contents/MacOS/ARDAgent

ARDAgent

- litefuzz
  - -s
  - // -ls -c “/System/Library/CoreServices/RemoteManagement/ARDAgent.app/Contents/MacOS/ARDAgent”
  - -a udp://10.0.0.100:3283
  - -i input/ard-pkt.bin
  - -n 100000
- Catching crashes can be interesting for some targets
  - Enable ReportCrash and monitor /Library/Logs/DiagnosticLogs for crash reports
  - Kill packet sniffing and last few packets when crash is detected
  - Run ARDAgent in a debugger or attach to it and signal when to stop eg. crash is detected

ARDAgent

- dtrace -n 'syscall::recv*:entry { printf("-> %s (pid=%d)", execname, pid); }'
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - 1 215 recvmsg:entry -> ARDAgent (pid=78386)
  - .....

References
https://wiki.freebsd.org/DTrace/One-Liners

screensharingd

- lsof -i :5900 | grep LISTEN
  - launchd 1 root 14u IPv6 0x1bc800334779b03 0t0 TCP *:rfb (LISTEN)
  - launchd 1 root 16u IPv4 0x1bc800334780653 0t0 TCP *:rfb (LISTEN)
  - screensha 48691 root fp.u IPv6 0x1bc800334779b03 0t0 TCP *:rfb (LISTEN)
  - screensha 48691 root fp.u IPv4 0x1bc800334780653 0t0 TCP *:rfb (LISTEN)
  - screensha 48691 root 3u IPv6 0x1bc800334779b03 0t0 TCP *:rfb (LISTEN)
  - screensha 48691 root 4u IPv4 0x1bc800334780653 0t0 TCP *:rfb (LISTEN)
- For some server processes, launchd runs them only upon connection

screensharingd

- litefuzz
  - -s
  - -a tcp://localhost:5900
  - -i input/screenshared-session
  - --reportcrash screensharingd
  - -p
  - -n 1000000
- Came across a potential exhaustion bug, but wasn’t reproducible
  - crashes/screensharingd_XXXXXX-XXXXXX_mac11.cpu_resource.diag

ODSAgent

- ODSAgent is a XML-based web service
  - Default is to prompt on request for access

ODSAgent

- litefuzz
  - -s
  - -a tcp://10.0.0.100:56156 (dynamic port)
  - -i input/cd-dvd-sharing
  - -p
  - --reportcrash ODSAgent *// check for system crash logs*
  - -n 1000000

BONUS

- Fuzzing the classic ‘say’ app
  - $ say -f talk.txt -i
    - how does this work?
- Computer voice reading garbled text++
  - It’s funny for the first 10 seconds, then you’ll want to mute your macbook :’)
  - But is it possible to actually crash this thing?
    - “Input text is not UTF-8 encoded”

Reference
https://ss64.com/osx/say.html

Bugs

- Fuzzed out bugs for various Mac apps and components such as
  - AppleScript
  - ColorSync
  - Syslog
  - ...
- And more upcoming CVEs

Conclusion

- Maybe you're more interested in fuzzing on Mac now
  - There's a learning curve, but lots of core apps and attack surface which means bugs
  - Must workaround / turn some security features off to get started
  - Apple has spent a lot of effort locking down many debugging features by default
- Lots of good tools out there to make vuln research more efficient
  - Applying unconventional techniques in fuzzing can still yield good results

EOF