c0ntextomy

Let's debug together

Presented by Nikias Bassen

September 10, 2022 - @nullcon Goa

About the Authors

- Nikias Bassen, Zimperium Inc., VP of Product Security — @pimskeks
  - iOS jailbreak co-author (Absinthe, evasi0n, checkra1n)
  - libimobiledevice maintainer
  - handling research and implementation of next-generation threat detections on iOS in Zimperium zLabs
- Dany Lisiansky, Independent Vulnerability Researcher — @DanyL931
  - iOS & macOS vulnerability research
  - iOS jailbreak co-author (checkra1n)

Agenda

- Introduction
  - lockdownd & services
  - Past vulnerabilities
- Discovery & Research
- Exploitation
- Vulnerability Disclosure and Fix
- Q & A

Introduction

Introduction: lockdownd

- lockdownd is the main service daemon, always running
  - Device <--> host pairings & device info
  - Service startup
  - Originally USB only, later WiFi support

Introduction: Services

- Several device services for different purposes, e.g.
  - Backup, file transfer ("AFC")
  - App and Profile management
  - System logs, Diagnostics
  - Debugging and Instrumentation (after mounting Developer Disk Image)
- Support for wireless connections from iOS 5 (requiring SSL)
  - Wireless Debugging with iOS 11+

Past vulnerabilities

Past vulnerabilities #1

AFC+MobileBackup2: directory traversal

- Create a symlink via AFC (link target was not checked)
  - e.g. `/Books/mylink -> ../../../root/Library/Lockdown`
- Restore a crafted backup via MobileBackup2
  - file would be restored where the link points to instead
- Used in Absinthe 2 jailbreak (iOS 5.1)

Past vulnerabilities #2

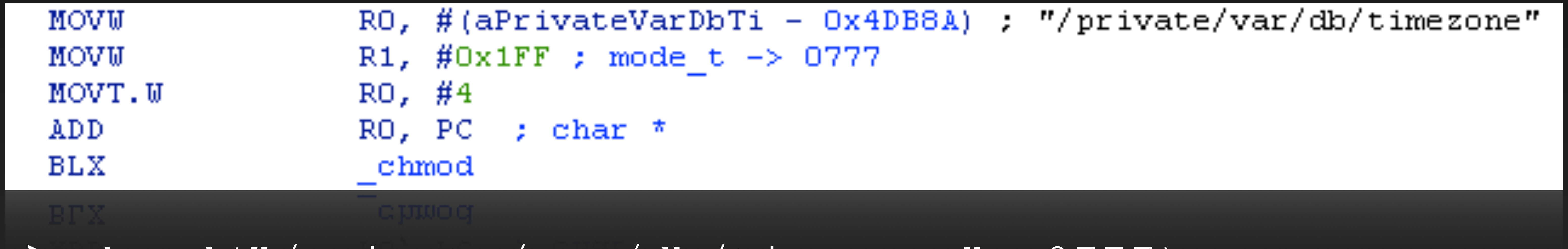
BackupAgent2: directory traversal

- Create crafted backup with symlink pointing to different location
- Needs to be created in allowed path ("Backup domain"), e.g.
  ```
  Media/Recordings/haxx -> /var/mobile
  ```
- Add additional files/directories to backup at symlink location
  ```
  Media/Recordings/haxx/DemoApp.app
  ...
  ```
- Restoring backup then creates files/directories at symlink target location
  ```
  /var/mobile/DemoApp.app
  ```
- Used in evasi0n jailbreak (iOS 6.0 - 6.1.2)

Past vulnerabilities #3

lockdownd: chmod with no path checks - CVE-2013-0979

- lockdownd performs certain tasks every time it launches
- One of them was changing permissions of a certain file:



```
MOVW        R0, #(aPrivateVarDbTi - 0x4DB8A) ; "/private/var/db/timezone"
MOVW        R1, #0x1FF ; mode_t -> 0777
MOVT.W      R0, #4
ADD         R0, PC  ; char *
BLX         _chmod
```

-> `chmod("/private/var/db/timezone", 0777);`

-> no further checks on path (e.g. if it contains a symlink)

- MB2 directory traversal + symlink: setting rwx permission on ANY file
- Just required crashing lockdownd at will (by sending malformed request 🤓)
- Used in evasi0n jailbreak (iOS 6.0 - 6.1.2)

Past vulnerabilities #4

"Launchdown" by Dany Lisiansky - CVE-2019-8637

- When mounting a Developer disk image, a notification is sent by MobileStorageMounter
- lockdownd and installd load resources based on path in notification
- Notification can be spoofed by 3rd party app with controllable path
- This way, lockdownd can be fooled to load custom services agents, essentially allowing arbitrary command execution as any user
- Fixed in iOS 12.3
- More info: https://github.com/DanyL/lockdownd_playground

Discovery & Research

Discovery

- iOS 13+ enforces SSL for most service connections via USB
  - Wireless connections were using SSL already
- libimobiledevice bug report: debugserver communication via USB fails
  - Service startup performs SSL handshake, then drops to plain text
- Also, libimobiledevice couldn't handle network connections yet
- Network sniffing revealed: It is the same over WiFi - it's all plain text!
  - Can this be exploited?! And also, how to fix libimobiledevice?

Research

What exactly is happening here?!

- libimobiledevice was already handling SSL connections correctly
- Experimenting showed that by ignoring the switch to SSL mode the debugserver connection would fail
- And network (or USB) dump clearly showed a SSL Handshake
- Shutting down SSL properly also didn't work
- Turns on, after the handshake, SSL just needs to be ignored COMPLETELY
- This sounds like something is not right here...

Research

Fixing libimobiledevice

- Besides the possible exploitability, we also wanted to fix libimobiledevice
- Several contributors looked into it and suggested changes
- Eventually, a function was added to ignore SSL mode after successful service startup and called in the debug service implementation

Research

Understanding the underlying issue (1)

- MobileDevice.framework on macOS encapsulates the (private) API required to start services on iOS/iPad/tvOS devices
- After starting a service, lockdownd on the device sends back to the host:
  - a port number the service has been made available on
  - a key called EnableServiceSSL with a value of either True or False*
- The code in MobileDevice.framework then performs the SSL handshake based on returned value

* Up to iOS 12.4.x, all services started via USB had it set to False, but all services started via WiFi, or on a device with iOS/iPadOS/tvOS 13.x or higher via USB, will have this set to True.

Research

Understanding the underlying issue (2)

- What happens next on the device side?
  - lockdownd handles the SSL handshake initiated by the host
  - service process is launched and it "checks in" with lockdownd via
    `secure_lockdown_checkin()`
  - Then, usually, it takes over the socket connection and gets SSL context via
    `lockdown_get_socket()`
    `lockdown_get_securecontext()`
  - However, debugserver never calls `lockdown_get_securecontext()`
    - so it does not even expect encrypted communication

Research

Understanding the underlying issue (3)

- So what happens on the host side?
  - The developer tools services are implemented outside of MobileDevice.framework
  - Service startup is initiated from inside Xcode's plugin IDEiOSSupportCore
  - On success it will request the socket file descriptor and pass it to lldb-rpc-server:

    ```
    /Applications/Xcode.app/Contents/SharedFrameworks/LLDBRPC.framework/Resources/
    lldb-rpc-server --unix-fd 68 --fd-passing-socket 70
    ```
  - Unaware of missing SSL, lldb-rpc-server just uses the connection (but then again, debugserver can't handle it anyway)

Research

Summary of the design flaw

- Architectural design flaw that evolved over time
- Initially no need for SSL, since USB only
  - "OK" to expose the raw file descriptor and use it externally
- But then Wireless Development was added to Xcode a few years later
- MobileDevice.framework handles SSL handshake, but still allows to get the raw file descriptor, that is used across processes by Xcode
- Basically, this breaks the abstraction of the service communication code from the consumer application (unlike libimobiledevice 😎)

Exploitation

Exploitation

Is this actually exploitable?

- Exploitation is restricted by a few things:
  - Attacker needs to be in the same network
  - Attacker needs to be able to redirect and manipulate network traffic
  - Victim needs to start a wireless debugging session
- Let's assume the following scenario:
  - intruded local network inside a company developing an AR/Fitness themed app - using wireless debugging

Exploitation

Gaining control over the debug session

- ARP spoofing* to redirect the victim's traffic through a machine we control
- Man-in-the-Middle (MITM) to gain control over the session
- However, we can't predict the debugserver port since it's dynamically assigned and the service startup uses SSL so we can't just see it
- So we have to look at all TCP sessions and pick the right one

* Note: While this specifically targets IPv4 networks, it is possible to use other kinds of attacks to target IPv6 networks as well (e.g. by attacking the NDP protocol or by propagating from other network positions).

Exploitation

Detecting a `gdb-remote` session

- Thankfully the protocol defines an easy to identify handshake that initializes the session:

```
Client -> Server: $QStartNoAckMode#b0
Client <- Server: +$OK#9a
Client -> Server: +
Client -> Server: $qSupported:xmlRegisters=i386,arm,mips#12
Client <- Server: $NqXfer:features:read+;PacketSize=20000;qEcho+;
  SupportedCompressions=lzfse,zlib-deflate,lz4,lzma;DefaultCompressionMinSize=384#00
Client -> Server: $QEnableCompression:type:lzfse;#bf
```

- This allows us to find the right TCP session
- At this stage, the client sends various configuration packets, sets breakpoints, etc.
- If we will try to manipulate right away we are likely to break the session (and make the victim notice)

Exploitation

The right point in time to manipulate the session

- How can we make sure the session is fully initialized?
- The same way as Xcode does:
  - During initialization it enables the "async process profiling" feature
  - Once it receives the first profiling packet, it will present the debugging UI
- So once we receive the first profiling packet, we can begin

Exploitation

Preparing the session for a second client

- We want to add another client to the session
- But we also want to keep Xcode unaware, so the victim doesn't notice
- This is how we do it:
    - Wait for the profiling packet, and save it for replaying, but don't relay it
    - Send a "process interrupt" packet (because lldb expects the process to be stopped when it tries to connect)
    - Now we separate the original session, and start replaying the profile packet we saved earlier on it - Xcode just thinks the process is running

Exploitation

Attacker joins the party (1)

- We are in control of the session and the process is stopped
- How do we add a second client?
- Let's just try it:

```
(lldb) gdb-remote 10.11.0.1:31337
error: failed to get reply to handshake packet
```

- It doesn't work because the client is already initialized 😭
- Hmm... what if we don't relay the session initialization packets to debugserver, and reply to the client with the expected replies...

Exploitation

Attacker joins the party (2)

- Let's see...

```
(lldb) gdb-remote 10.11.0.1:31337
(lldb) th ba
* thread #1, queue = 'com.apple.main-thread', stop reason = signal SIGSTOP
  * frame #0: 0x00000001896bb5f4 libsystem_kernel.dylib`mach_msg_trap + 8
    frame #1: 0x00000001896baa60 libsystem_kernel.dylib`mach_msg + 72
    frame #2: 0x0000000189862068 CoreFoundation`__CFRunLoopServiceMachPort + 216
    frame #3: 0x000000018985d188 CoreFoundation`__CFRunLoopRun + 1444
    frame #4: 0x000000018985c8bc CoreFoundation`CFRunLoopRunSpecific + 464
    frame #5: 0x00000001936c8328 GraphicsServices`GSEventRunModal + 104
    frame #6: 0x000000018d8f26d4 UIKitCore`UIApplicationMain + 1936
    frame #7: 0x000000010008e2e4 project`main + 132
    frame #8: 0x00000001896e7460 libdyld.dylib`start + 4
```

  PARTY 🎉

- We have a full session takeover in place, with working lldb shell

Exploitation

A more practical way to join a second client

- After some further research, we figured out a more practical way
- gdb-remote is implemented as "process connect" plugin, and it has a fd:// argument!

  ```
  process connect -p gdb-remote fd://12
  ```
- Way simpler and cleaner approach
- Allowed us to implement a "reverse lldb client" that accepts connections and passes the socket file descriptors straight to lldb

Exploitation

Remote Code Execution

- Since we now control lldb, we can use the expression feature to execute code in the context of the process
- We over-engineered a PoC that ended up being a complete toolset with modules allowing exfiltration of user data from the victim's device
- Basically, bootstrap lldb with shellcode payload running in a new thread
- Connects back to "exfiltration server"
- Execute modules with additional shellcode

Vulnerability Disclosure & Fix

Disclosure Timeline

June 14, 2019	Vulnerability discovered
March 12, 2020	Apple notified
April 14, 2020	Apple asked to extend the disclosure period until "later this summer"
August 6, 2020	Apple notified us the patches are present in the new beta versions and are planned to be published in a future security update
August 14, 2020	We found a downgrade attack bypassing the new patch, affecting iOS 13.6 and 13.7 😱

Downgrade Attack?

- Vulnerability supposedly fixed with Xcode 12 beta 3
- Developer Disk Images (DDI) for iOS < 13.6 - no mitigations
- DDI for 13.6 and 14.0: new secure service variants, but also old variant
- With Xcode 12 beta 4, DDI for 14.0 does not have old variant anymore
- iOS 13.6 and 13.7 stays vulnerable to downgrade attack
- Apple told us only iOS 14+ will be fully addressed

Downgrade Attack!!

- iOS 13.6 and 13.7 is vulnerable
- Turns out Xcode implementation has a fallback:
  if secure variant isn't starting, try the insecure one
- With MITM and packet inspection, attacker could find the encrypted StartService packet based on size or other heuristics
- Then drop the connection
- Xcode thinks service start failed, and will launch insecure one
- Party again - but yes, only for older versions

The Fix

- New secure service variants where added to DDI: com.apple.debugserver.DVTSecureSocketProxy
- Creates a tunnel with proper SSL connection
- Device side was updated to reflect that
- Socket file descriptor of tunnel endpoint is passed to lldb-rpc-server
- lldb can now transparently communicate over a secure connection

(Full) Disclosure Timeline

June 14, 2019	Vulnerability discovered
March 12, 2020	Apple notified
April 14, 2020	Apple asked to extend the disclosure period until "later this summer"
August 6, 2020	Apple notified us the patches are present in the new beta versions and are planned to be published in a future security update
August 14, 2020	We found a downgrade attack bypassing the new patch, affecting iOS 13.6 and 13.7
September 5, 2020	Apple explained only the upcoming iOS/iPadOS/tvOS 14 and watchOS 7 are fully addressed
September 16, 2020	Vulnerability patched in Xcode 12.0 release - CVE-2020-9992

Q & A

Advisory and PoC at:

https://github.com/c0ntextomy/c0ntextomy