

# THE SH3LLC0D3R'S BLOG

HOME CONTACT CTF WALKTHROUGHS EXPLOIT DEVELOPMENT MOBILE SECURITY NETWORK

SECURITYTUBE - LINUX ASSEMBLY EXPERT 32-BIT SECURITYTUBE - OFFENSIVE IOT EXPLOITATION SECURITYTUBE EXAMS

CISCO EMBEDDED

Home / Main / How to debug an iOS application with Appmon and LLDB

## How to debug an iOS application with Appmon and LLDB

📅 June 13, 2017 👤 elcapitan 🏷️ Main

How to install Appmon [here](#).

LLDB reference can be found [here](#).

More on ASLR and debugging with LLDB [here](#).

This blog is dedicated to my research and experimentation on ethical hacking. The methods and techniques published on this site should not be used to do illegal things.

If we install Appmon, then we have an LLDB window, where we can debug the IOS application. Debugging is one way to examine/manipulate an application.

The best way to debug the application is to load it into IdaPro first. Then let us find an address, where we want to set breakpoint. For this blog post, I am using the [DamnVulnerableIOSApp](#). Here is the `[JailbreakDetectionVC jailbreakTest1Tapped:]` method in IdaPro.

```
text:000000010001974C
text:000000010001974C ; ===== SUBROUTINE =====
text:000000010001974C ; JailbreakDetectionVC - (void)jailbreakTest1Tapped:(id)
text:000000010001974C ; Attributes: bp-based Frame
text:000000010001974C
text:000000010001974C ; void __cdecl -[JailbreakDetectionVC jailbreakTest1Tapped:](struct JailbreakDetectionVC *self, SEL, id)
text:000000010001974C ; JailbreakDetectionVC_jailbreakTest1Tapped_
text:000000010001974C ; DATA XREF: __objc_const:00000001001FA1A8↓o
text:000000010001974C
text:000000010001974C var_10      = -0x10
text:000000010001974C var_s0      = 0
text:000000010001974C
text:000000010001974C STP        X20, X19, [SP, #-0x10+var_10]!
text:0000000100019750 STP        X29, X30, [SP, #-0x10+var_s0]
text:0000000100019754 ADD        X29, SP, #0x10
text:0000000100019758 ADRP      X8, #classRef_DamnVulnerableAppUtilities@PAGE
text:000000010001975C LDR        X19, [X8, #classRef_DamnVulnerableAppUtilities@PAGEOFF]
text:0000000100019760 ADRP      X8, #selRef_isJailbroken@PAGE
text:0000000100019764 NOP
text:0000000100019768 LDR        X1, [X8, #selRef_isJailbroken@PAGEOFF]
text:000000010001976C BL         _objc_msgSend
text:0000000100019770 MOV        X2, X0
text:0000000100019774 ADRP      X8, #selRef_showAlertForJailbreakTestIsJailbroken_@PAGE
text:0000000100019778 NOP
text:000000010001977C LDR        X1, [X8, #selRef_showAlertForJailbreakTestIsJailbroken_@PAGEOFF]
text:0000000100019780 MOV        X0, X19
text:0000000100019784 LDP        X29, X30, [SP, #-0x10+var_s0]
text:0000000100019788 LDP        X20, X19, [SP, #-0x10+var_10], #0x20
text:000000010001978C B         _objc_msgSend
text:000000010001978C ; End of function -[JailbreakDetectionVC jailbreakTest1Tapped:]
text:000000010001978C
text:0000000100019790
```

The address is 0x10001974C. However the real address in memory is different, because the whole TEXT segment is shifted with a certain value (ASLR). In order to determine the shift value, execute the following command in LLDB:

**(lldb) image dump sections <APP\_NAME>**

```
(lldb) image dump sections DamnVulnerableIOSApp
Sections for '/Users/horvathviktor/git/appmon/ipa_installer/apps/DamnVulnerableIOSApp-injected-resigned/Payload/DamnVulnerableIOSApp.app/DamnVulnerableIOSApp' (arm64)
-----
SecID   Type      Load Address          Perm File Off.   File Size   Flags   Section Name
-----
0x00000100 container [0x0000000000000000-0x0000000100000000)+ --- 0x00000000 0x00000000 0x00000000 DamnVulnerableIOSApp.__PAGEZERO
0x00000200 container [0x0000000100000000-0x000000010028c000) r-x 0x00000000 0x001e4000 0x00000000 DamnVulnerableIOSApp.__TEXT
0x00000001 code       [0x000000010000ac734-0x0000000100130ad8) r-x 0x00004734 0x0000843d4 0x80000400 DamnVulnerableIOSApp.__TEXT.__text
0x00000002 code       [0x0000000100130ad8-0x0000000100131438) r-x 0x000085ad8 0x000000960 0x80000408 DamnVulnerableIOSApp.__TEXT.__stubs
0x00000003 code       [0x0000000100131438-0x0000000100131d00) r-x 0x000085ad8 0x000000278 0x80000400 DamnVulnerableIOSApp.__TEXT.__stub_helper
```

I do not take responsibility for acts of other people.

## RECENT POSTS

Androguard usage

How to debug an iOS application with Appmon and LLDB

OWASP Uncrackable – Android Level3

OWASP Uncrackable – Android Level2

How to install Appmon and Frida on a Mac

## CATEGORIES

Android (5)

Fusion (2)

IoT (13)

Main (3)

Mobile (6)

Protostar (24)

SLAE32 (8)

The TEXT segment starts at 0x1000a8000, so the shift value is 0xa8000. We have to add this value to every address, that comes from IdaPro. The following command disassemble a few bytes from the passed address:

VulnServer (6)

Windows Reverse Shell (2)

**(lldb) disassemble --start-address 0xa8000+0x10001974**

```
(lldb) disassemble --start-address 0xa8000+0x10001974c
DamnVulnerableIOApp`___lldb_unnamed_symbol1578$$DamnVulnerableIOApp:
0x1000c174c <+0>: stp    x20, x19, [sp, #-0x20]!
0x1000c1750 <+4>: stp    x29, x30, [sp, #0x10]
0x1000c1754 <+8>: add    x29, sp, #0x10          ; =0x10
0x1000c1758 <+12>: adrp   x8, 510
0x1000c175c <+16>: ldr    x19, [x8, #0xcf8]
0x1000c1760 <+20>: adrp   x8, 507
0x1000c1764 <+24>: nop
0x1000c1768 <+28>: ldr    x1, [x8, #0xdc0]
(lldb) █
```

I set a breakpoint at 0x100019770.

**(lldb) breakpoint set -a 0xa8000+0x100019770**

This is the next instruction after the `[jailbreakDetectionVC jailbreakTest1Tapped:]` call in `jailbreakTestTapped1` method. The return value is in X0 register. 0x1 means, that the device is jailbroken.

```
(lldb) register read
General Purpose Registers:
x0 = 0x0000000000000001
x1 = 0x000000001702ca330
x2 = 0x0000000000000008
x3 = 0x0000000000000000
x4 = 0x0000000000000000
x5 = 0x0000000000000000
x6 = 0x0000000000000000
x7 = 0x0000000000000000
x8 = 0x0000000000000000
x9 = 0x0000000000000000
x10 = 0x0000000000000000
x11 = 0x0000000000000000
x12 = 0x0000000000000000
x13 = 0x0000000000000000
x14 = 0x0000000000000000
x15 = 0x0000000000000000
x16 = 0x0000000000000000
x17 = 0x0000000000000000
x18 = 0x0000000000000000
x19 = 0x0000000000000000
x20 = 0x0000000000000000
x21 = 0x0000000000000000
x22 = 0x0000000000000000
x23 = 0x0000000000000000
x24 = 0x0000000000000000
x25 = 0x0000000000000000
x26 = 0x0000000000000000
x27 = 0x0000000000000000
x28 = 0x0000000000000000
x29 = 0x0000000000000000
x30 = 0x0000000000000000
```

This can be modified with ...

```
(lldb) register write x0 0
```

```
(lldb) continue
```

```
(lldb) register write x0 0  
(lldb) c  
Process 1040 resuming  
(lldb) █
```

The application does not recognize the device as jailbroken one.

Locked SIM 14:23 99%

## Menu Jailbreak Detection

Some developers do a check for a jailbroken device and allow the application to function only if it isn't. Your task is to run this application on a jailbroken device and fool the application into thinking it is not jailbroken.

Device is Not Jailbroken

Ok

Jailbreak Test 2

« PREVIOUS POST

NEXT POST »

Copyright © 2019, The sh3llc0d3r's blog. Proudly powered by  
[WordPress](#). Blackoot design by [Iceable Themes](#).

[Home](#) [Contact](#) [CTF walkthroughs](#) [Exploit development](#)  
[Mobile Security](#) [Network](#)  
[SecurityTube – Linux Assembly Expert 32-bit](#)  
[SecurityTube – Offensive IoT Exploitation](#) [SecurityTube exams](#)  
[CISCO](#) [Embedded](#)