Dumping Memory on iOS 8

Back in January of 2015 NetSPI published a blog on extracting memory from an iOS device. Even though NetSPI provided a script to make it easy, it required iOS 7 (or less) and GDB; but GDB is currently no longer on iOS 8.

Fortunately, there are other options to GDB and extracting memory from an Apple iPhone running iOS 8+ could not be easier. It requires the following pieces of software.

- LLDB (http://lldb.llvm.org/)
- Debugserver (part of Xcode)
- Tcprelay.py (https://code.google.com/p/iphonetunnel-mac/source/browse/trunk/gui/tcprelay.py?r=5)

Of course you will need a jailbroken iPhone or iPad. I will not cover that part of the operation here.

Start tcprelay so you can connect to the device over a USB connection:

```
$ ./tcprelay.py -t 22:2222 1234:1234

Forwarding local port 2222 to remote port 22

Forwarding local port 1234 to remote port 1234

Incoming connection to 2222

Waiting for devices…

Connecting to device <MuxDevice: ID 17 ProdID 0x12a8 Serial
‘0ea150b00ba3deeacb42f399492b7990416a0c87’ Location 0x14120000>

Connection established, relaying data

Incoming connection to 1234

Waiting for devices…

Connecting to device <MuxDevice: ID 17 ProdID 0x12a8 Serial
‘0ea150b00ba3deeacb42f399492b7990416a0c87’ Location 0x14120000>

Connection established, relaying data
```

The command “tcprelay.py -t 22:2222 1234:1234” is redirecting two local ports to the device. The first one is used to SSH to the device over port 2222. The second one is the port the debugserver will be using.

Then you will need to connect to the iOS device and start the debug server (I am assuming you have

already copied the software to the device). If not, you can use scp to copy the binary.)

```
$ ssh root@127.0.0.1 -p 2222

root@127.0.0.1's password:
```

Then, if the application is already running, verify its name using 'ps aux | grep <appname>' and connect to the application with debugserver (using the name of the application not the PID):

```
root# ./debugserver *:1234 -a appname

debugserver-@(#)PROGRAM:debugserver  PROJECT:debugserver-320.2.89

for arm64.

Attaching to process appname...

Listening to port 1234 for a connection from *...

Waiting for debugger instructions for process 0.
```

The command './debugserver *:1234 -a appname' is telling the software to startup on port 1234 and hook into the application named 'appname'. It will take a little time, so be patient.

On the MAC, startup LLDB and connect to the debugserver software running on the iOS device. Remember, we have relayed the device port 1234 that the debugserver is listening on to the local port 1234.

```
$ lldb

(lldb) process connect connect://127.0.0.1:1234

Process 2017 stopped

* thread #1: tid = 0x517f9, 0x380f54f0 libsystem_kernel.dylib mach_msg_trap + 20, queue
= 'com.apple.main-thread', stop reason = signal SIGSTOP

frame #0: 0x380f54f0 libsystem_kernel.dylib mach_msg_trap + 20

libsystem_kernel.dylib mach_msg_trap:

->  0x380f54f0 <+20>: pop    {r4, r5, r6, r8}

0x380f54f4 <+24>: bx     lr

libsystem_kernel.dylib mach_msg_overwrite_trap:
```

0x380f54f8 <+0>: mov   r12, sp

0x380f54fc <+4>: push   {r4, r5, r6, r8}

Now you can dump the information about the memory sections of the application.

(lldb) image dump sections appname

Sections for '/private/var/mobile/Containers/Bundle/Application/F3CFF345-71FC-47C4-B1FB-3DAC523C7627/appname.app/appname(0x0000000000047000)' (armv7):

SectID Type Load Address File Off. File Size Flags Section Name

———- ———— ——————————— ———- ———- ———- ——————————-

0x00000100 container [0x0000000000000000-0x0000000000004000)* 0x00000000 0x00000000 0x00000000 appname.__PAGEZERO

0x00000200 container [0x0000000000047000-0x00000000001af000) 0x00000000 0x00168000 0x00000000 appname.__TEXT

0x00000001 code [0x000000000004e6e8-0x000000000016d794) 0x000076e8 0x0011f0ac 0x80000400 appname.__TEXT.__text

0x00000002 code [0x000000000016d794-0x000000000016e5e0) 0x00126794 0x00000e4c 0x80000400 appname.__TEXT.__stub_helper

0x00000003 data-cstr [0x000000000016e5e0-0x0000000000189067) 0x001275e0 0x0001aa87 0x00000002 appname.__TEXT.__cstring

0x00000004 data-cstr [0x0000000000189067-0x00000000001a5017) 0x00142067 0x0001bfb0 0x00000002 appname.__TEXT.__objc_methname

0x00000005 data-cstr [0x00000000001a5017-0x00000000001a767a) 0x0015e017 0x00002663 0x00000002 appname.__TEXT.__objc_classname

0x00000006 data-cstr [0x00000000001a767a-0x00000000001abe0c) 0x0016067a 0x00004792 0x00000002 appname.__TEXT.__objc_methtype

0x00000007 regular [0x00000000001abe10-0x00000000001ac1b8) 0x00164e10 0x000003a8 0x00000000 appname.__TEXT.__const

0x00000008 regular [0x00000000001ac1b8-0x00000000001aeb20) 0x001651b8 0x00002968 0x00000000 appname.__TEXT.__gcc_except_tab

0x00000009 regular [0x00000000001aeb20-0x00000000001aeb46) 0x00167b20 0x00000026 0x00000000 appname.__TEXT.__ustring

0x0000000a code [0x00000000001aeb48-0x00000000001af000) 0x00167b48 0x000004b8 0x80000408 appname.__TEXT.__symbolstub1

0x00000300 container [0x00000000001af000-0x00000000001ef000) 0x00168000 0x00040000 0x00000000 appname.__DATA

0x0000000b data-ptrs [0x00000000001af000-0x00000000001af4b8) 0x00168000 0x000004b8 0x00000007 appname.__DATA.__lazy_symbol

0x0000000c data-ptrs [0x00000000001af4b8-0x00000000001af810) 0x001684b8 0x00000358 0x00000006 appname.__DATA.__nl_symbol_ptr

0x0000000d regular [0x00000000001af810-0x00000000001b2918) 0x00168810 0x00003108 0x00000000 appname.__DATA.__const

0x0000000e objc-cfstrings [0x00000000001b2918-0x00000000001ba8d8) 0x0016b918 0x00007fc0 0x00000000 appname.__DATA.__cfstring

0x0000000f data-ptrs [0x00000000001ba8d8-0x00000000001baf1c) 0x001738d8 0x00000644 0x10000000 appname.__DATA.__objc_classlist

0x00000010 regular [0x00000000001baf1c-0x00000000001baf4c) 0x00173f1c 0x00000030 0x10000000 appname.__DATA.__objc_nlclslist

0x00000011 regular [0x00000000001baf4c-0x00000000001bafa0) 0x00173f4c 0x00000054 0x10000000 appname.__DATA.__objc_catlist

0x00000012 regular [0x00000000001bafa0-0x00000000001bafa4) 0x00173fa0 0x00000004 0x10000000 appname.__DATA.__objc_nlcatlist

0x00000013 regular [0x00000000001bafa4-0x00000000001bb078) 0x00173fa4 0x000000d4 0x00000000 appname.__DATA.__objc_protolist

0x00000014 regular [0x00000000001bb078-0x00000000001bb080) 0x00174078 0x00000008 0x00000000 appname.__DATA.__objc_imageinfo

0x00000015 data-ptrs [0x00000000001bb080-0x00000000001e0d40) 0x00174080 0x00025cc0 0x00000000 appname.__DATA.__objc_const

0x00000016 data-cstr-ptr [0x00000000001e0d40-0x00000000001e4420) 0x00199d40 0x000036e0 0x10000005 appname.__DATA.__objc_selrefs

0x00000017 regular [0x00000000001e4420-0x00000000001e442c) 0x0019d420 0x0000000c 0x00000000 appname.__DATA.__objc_protorefs

0x00000018 data-ptrs [0x00000000001e442c-0x00000000001e4ab8) 0x0019d42c 0x0000068c 0x10000000 appname.__DATA.__objc_classrefs

0x00000019 data-ptrs [0x00000000001e4ab8-0x00000000001e4e48) 0x0019dab8 0x00000390 0x10000000 appname.__DATA.__objc_superrefs

0x0000001a regular [0x00000000001e4e48-0x00000000001e6184) 0x0019de48 0x0000133c 0x00000000 appname.__DATA.__objc_ivar

0x0000001b data-ptrs [0x00000000001e6184-0x00000000001ea02c) 0x0019f184 0x00003ea8 0x00000000 appname.__DATA.__objc_data

0x0000001c data [0x00000000001ea030-0x00000000001ed978) 0x001a3030 0x00003948 0x00000000 appname.__DATA.__data

0x0000001d zero-fill [0x00000000001ed980-0x00000000001edce0) 0x00000000 0x00000000 0x00000001 appname.__DATA.__bss

0x0000001e zero-fill [0x00000000001edce0-0x00000000001edce8) 0x00000000 0x00000000 0x00000001 appname.__DATA.__common

0x00000400 container [0x00000000001ef000-0x0000000000207000) 0x001a8000 0x00015bf0 0x00000000 appname.__LINKEDIT

The next step is to convert that output into LLDB commands to actually dump the data in those memory sections. You can probably skip the sections named zero-fill or code. For example, the take the following output:

0x00000003 data-cstr [0x000000000016e5e0-0x0000000000189067) 0x001275e0 0x0001aa87 0x00000002 appname.__TEXT.__cstring

Into the LLDB command:

Memory read –outfile ~/0x00000003data-cstr 0x000000000016e5e0 0x0000000000189067 –force

This command is telling LLDB to dump the memory from address 0x000000000016e5e0 to 0x0000000000189067 and put it into the file 0x00000003data-cstr.

(lldb) memory read –outfile ~/0x00000003data-cstr 0x000000000016e5e0 0x0000000000189067 –force

You will (or should) not see any output from this command other that the file being created. Once you have all of the files, search them using your favorite search tool or even a text editor. Search for sensitive data (i.e. credit card number, passwords, etc. The files will contain information similar to the following:

0x0016e5e0: 3f 3d 26 2b 00 3a 2f 3d 2c 21 24 26 27 28 29 2a ?=&+.:/=,!$&'()*

```
0x0016e5f0: 2b 3b 5b 5d 40 23 3f 00 00 62 72 61 6e 64 4c 6f  +;[]@#?..brandLo
0x0016e600: 67 6f 2e 70 6e 67 00 54 72 61 64 65 47 6f 74 68  go.png.TradeGoth
0x0016e610: 69 63 4c 54 2d 42 6f 6c 64 43 6f 6e 64 54 77 65  icLT-BoldCondTwe
0x0016e620: 6e 74 79 00 4c 6f 61 64 69 6e 67 2e 2e 2e 00 4c  nty.Loading....L
0x0016e630: 6f 61 64 69 6e 67 00 76 31 32 40 3f 30 40 22 4e  oading.v12@?0@"N
0x0016e640: 53 44 61 74 61 22 34 40 22 45 70 73 45 72 72 6f  SData"4@"EpsErro
0x0016e650: 72 22 38 00 6c 6f 61 64 69 6e 67 50 61 67 65 54  r"8.loadingPageT
0x0016e660: 79 70 65 00 54 69 2c 4e 2c 56 5f 6c 6f 61 64 69  ype.Ti,N,V_loadi
0x0016e670: 6e 67 50 61 67 65 54 79 70 65 00 6f 76 65 72 76  ngPageType.overv
0x0016e680: 69 65 77 52 65 71 52 65 73 48 61 6e 64 6c 65 72  iewReqResHandler
0x0016e690: 00 54 40 22 45 70 73 4f 76 65 72 76 69 65 77 52  .T@"EpsOverviewR
0x0016e6a0: 65 71 52 65 73 48 61 6e 64 6c 65 72 22 2c 26 2c  eqResHandler",&,
0x0016e6b0: 4e 2c 56 5f 6f 76 65 72 76 69 65 77 52 65 71 52  N,V_overviewReqR
0x0016e6c0: 65 73 48 61 6e 64 6c 65 72 00 41 50 49 43 61 6c  esHandler.APICal
```

Have fun looking at the iOS application memory and use this process for only good intentions. As stated in the previously mentioned blog:

This technique can be used to determine if the application is not removing sensitive information from memory once the instantiated classes are done with the data. All applications should de-allocate spaces in memory that deal with classes and methods that were used to handle sensitive information, otherwise you run the risk of the information sitting available in memory for an attacker to see.