

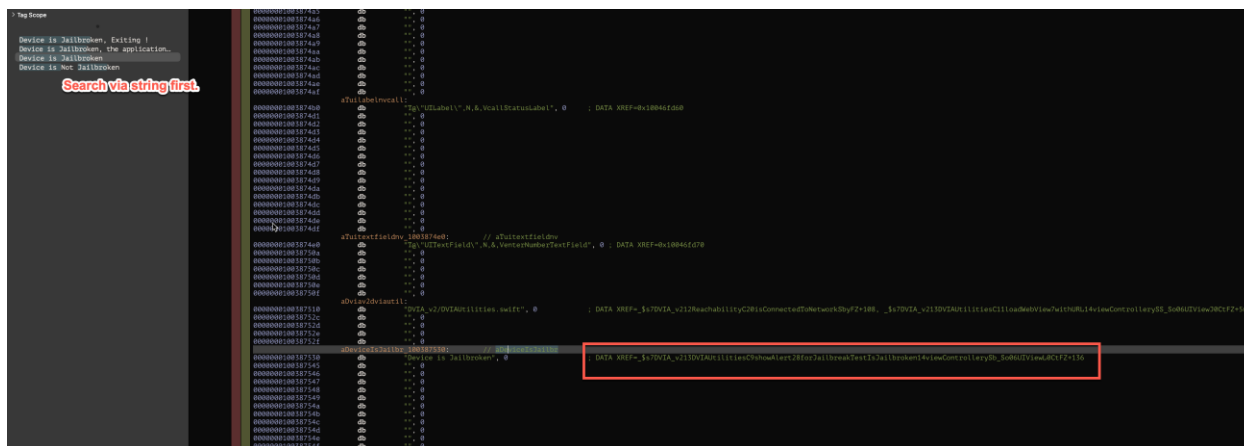
## DVIA-V2

### Jailbreak Test 1

It begins by searching for the string “Device is Jailbroken” as shown from the error message on the app itself.



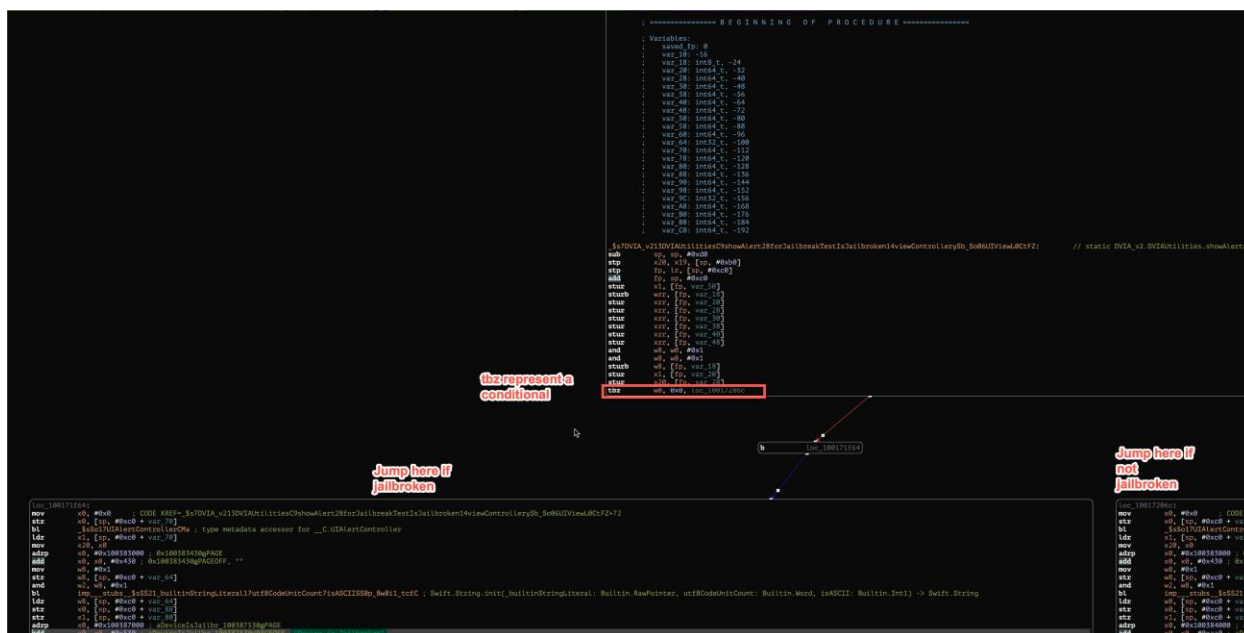
Subsequently, the app needs to be disassembled, and the relevant string(s) need to be searched.



By zeroing on the highlighted swift method and changing the view to code flow graph. There is a preceding code that determines whether a device is jailbroken.

This is the most important code here in arm assembly.

`tbz w0, 0x0, loc_10017206c`



There are two ways to hijack this.

A. You find the offset for “tbz” and modify the register values.

- B. You just hook into the affected swift method and tamper with the method argument so that it appears to the app that the device is not jailbroken.

Enlisting GPT's help to draft up Frida's code. Here it is: [\[08\]](#)

```
1 // Define the name of the target module (app binary)
2 // The module name is the name of the app binary, found using "frida -U -n TargetApp -i"
3 var moduleName = "DVIA-v2";
4
5 // Find the base address of the module in memory
6 var baseAddr = Module.findBaseAddress(moduleName);
7
8 if (baseAddr) {
9     console.log("[*] Found base address of " + moduleName + ": " + baseAddr);
10
11     // Offset of the target function from our static disassembly
12     // This is calculated as: function_address - base_address
13     var functionOffset = 0x171F18; // Offset from our analysis
14
15     // Calculate the actual memory address of the function in runtime
16     var targetFunction = baseAddr.add(functionOffset);
17
18     console.log("[*] Hooking function at: " + targetFunction);
19
20     // Attach an interceptor to the function to monitor or modify its behavior
21     Interceptor.attach(targetFunction, {
22         onEnter: function(args) {
23             console.log("[*] Hooked into showAlert function!");
24
25             // The first argument (args[0]) is a Swift Bool (true/false)
26             // In Swift, a Bool is passed as an integer (1 for true, 0 for false)
27             var jailbreakDetected = args[0].toInt32();
28             console.log("[*] Original Jailbreak Detection Status: " + jailbreakDetected);
29
30             // Modify the jailbreak detection result (force it to false)
31             // Setting args[0] to 0 means we are telling the app "no jailbreak detected"
32             args[0] = ptr(0);
33
34             console.log("[*] Overwritten Jailbreak Detection to: 0 (Not Jailbroken)");
35         },
36         onLeave: function(retval) {
37             // Log the return value (if applicable)
38             console.log("[*] Function finished execution. Return value: " + retval);
39         }
40     });
41
42 } else {
43     // If the module (app binary) isn't found, print an error message
44     console.log("[!] Error: Could not find base address of module " + moduleName);
45 }
```

How the offset is found is via hooking to the first instruction of the affected method. As ASLR is enabled, it means that on every program runtime, address will change so you need offset(s) to get the real address of the method during runtime.

```

***** BEGINNING OF PROCEDURE *****
Variables:
  saved_ip: 0
  var_10: -10
  var_18: int8_t, -24
  var_20: int64_t, -32
  var_28: int64_t, -40
  var_30: int64_t, -48
  var_38: int64_t, -56
  var_40: int64_t, -64
  var_48: int64_t, -72
  var_50: int64_t, -80
  var_58: int64_t, -88
  var_60: int64_t, -96
  var_64: int32_t, -100
  var_70: int64_t, -112
  var_78: int64_t, -120
  var_80: int64_t, -128
  var_88: int64_t, -136
  var_90: int64_t, -144
  var_98: int64_t, -152
  var_9C: int32_t, -156
  var_A0: int64_t, -160
  var_B0: int64_t, -176
  var_80: int64_t, -184
  var_C0: int64_t, -192

- [com.highaltitudehacks.DVIAswiftv2] showAlert28forJailbreakTestIsJailbroken4viewControllerD_ScMJViewACf2: // static DVIA_v2.DVIAUtilities.showAlert(forJailbreakTestIsJailbroken: Swift.Bool, viewController: _C.UIViewController) -> ()
0000000100171f18  stp     [fp, #-40]
0000000100171f1c  stp     [fp, #-32], [fp, #0xb0]
0000000100171f20  and     fp, sp, #0xb0
0000000100171f24  stuz    r1, [fp, var_50]
0000000100171f2c  stuz    r2r, [fp, var_10]
0000000100171f30  stuz    r2r, [fp, var_20]
0000000100171f38  stuz    r2r, [fp, var_30]
0000000100171f3c  stuz    r2r, [fp, var_40]
0000000100171f40  stuz    r2r, [fp, var_40]
0000000100171f44  stuz    r2r, [fp, var_40]
0000000100171f48  and     r0, r0, #0x1
0000000100171f4c  and     r0, r0, #0x1
0000000100171f50  stuz    r0, [fp, var_10]
0000000100171f54  stuz    r1, [fp, var_20]
0000000100171f58  stuz    r2r, [fp, var_20]
0000000100171f5c  stuz    r0, r0

```

When you hook, you need to hook into the beginning of the routine.

So here is the result after running the above code in Frida. Do observe the output.

```

/ _ | Frida 16.6.6 - A world-class dynamic instrumentation toolkit
| C |
> _ | Commands:
/_ | | help      -> Displays the help system
. . . | object?   -> Display information about 'object'
. . . | exit/quit -> Exit
. . . |
. . . | More info at https://frida.re/docs/home/
. . . |
. . . | Connected to iPhone (id=224b4e[REDACTED] 84ea9)
Spawning `com.highaltitudehacks.DVIAswiftv2` ...
[*] Found base address of DVIA-v2: 0x104608000
[*] Hooking function at: 0x104779f18
Spawned `com.highaltitudehacks.DVIAswiftv2`. Resuming main thread!
[iPhone::com.highaltitudehacks.DVIAswiftv2 ]-> [*] Hooked into showAlert function!
[*] Original Jailbreak Detection Status: 1
[*] Overwritten Jailbreak Detection to: 0 (Not Jailbroken)
[*] Function finished execution. Return value: 0x10889a000

```



## 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.

Jailbreak Test 1

Jailbreak Test 2

Device is Not Jailbroken

OK