iOS Swift Anti-Jailbreak Bypass with Frida

May 13, 2020 | Posted by **Raffaele Sabato**

Link: https://syrion.me/blog/ios-swift-antijailbreak-bypass-frida/

References:

- https://syrion.me/
- https://twitter.com/syrion89
- https://www.linkedin.com/in/raffaelesabato/

Frida

Frida is a dynamic binary instrumentation framework that has been around for a while. In a nutshell, Frida allows reverse engineers to perform activities such as function hooking/trancing and runtime code modification. If your target is an iOS application, Frida provides you with powerful Objective-C API, making painless reverse engineering tasks. Unfortunately, out of the box, Frida lacks of Swift API, and some community contributions are outdated. Analysing Swift iOS applications with Frida hasn't been an easy task for me so far. Fortunately, my friend r3dx0f provided me with some suggestions about how to approach the problem, and I will share with you what I learnt during this journey.

Swift Security Checks

In this blogpost, I will describe how to bypass a number of jailbreak and reverse engineering detection mechanisms implemented within the **IOSSecuritySuite**. The aforementioned project is written in Swift and is hosted on Github. However, we will perform the analysis against a non-stripped iOS application which implements such library. The analysis will be conducted without the support of the source code in order to simulate a real-life scenario. When the application is executed, it shows the message below, telling us that our iPhone is jailbroken. Some suspicious files are found and the application is considered “reversed”.

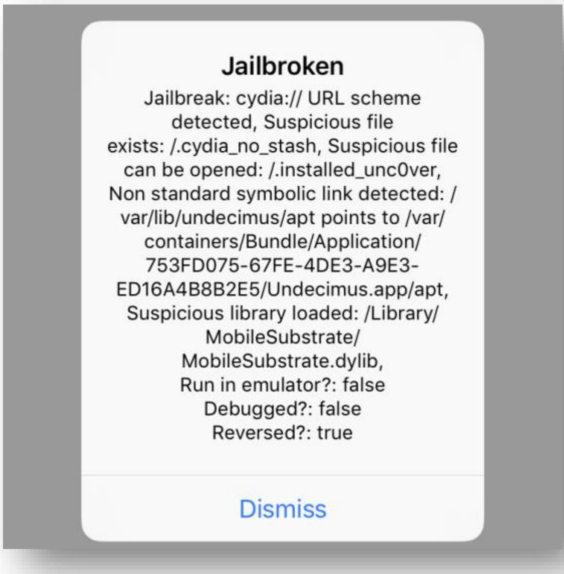


Figure 1 - Security Checks

We have two modules defined within the binary:

- ***FrameworkClientApp***
- ***IOSSecuritySuite***

Because we know that the **IOSSecuritySuite** modules contains the jailbreak detection mechanism logic, we will reverse it first.

Reverse Engineering

When I’m looking for jailbreak detection mechanisms, I usually start searching for strings and functions containing the word “**jailbr**” (jail, jailbreak or jailbroken) or “**root**”. We have a lot of matches as shown below:

jailbr

Tag Scope

Address	Type	Name
0xacac	P	static IOSSecuritySuite.IOSSecuritySuite.amIJailbroken() -> Swift.Bool
0xacf4	P	static IOSSecuritySuite.IOSSecuritySuite.amIJailbrokenWithFailMessage() -> (jailbroken: S...
0xad38	P	static IOSSecuritySuite.IOSSecuritySuite.amIJailbrokenWithFailedChecks() -> (jailbroken:...
0xe628	P	static IOSSecuritySuite.JailbreakCheck.__derived_enum_equals(IOSSecuritySuite.Jailbreak...
0xe798	P	IOSSecuritySuite.JailbreakCheck.hashValue.getter : Swift.Int
0xe80c	P	lazy protocol witness table accessor for type IOSSecuritySuite.JailbreakCheck and confor...
0xe870	P	IOSSecuritySuite.JailbreakCheck.hash(into: inout Swift.Hasher) -> ()
0xe968	P	static IOSSecuritySuite.JailbreakCheck.allCases.getter : [IOSSecuritySuite.JailbreakCheck]
0xe9c0	P	protocol witness for static Swift.Equatable.== infix(A, A) -> Swift.Bool in conformance IOS...
0xe9e4	P	protocol witness for Swift.Hashable.hashValue.getter : Swift.Int in conformance IOSSecuri...
0xea00	P	protocol witness for Swift.Hashable.hash(into: inout Swift.Hasher) -> () in conformance IO...
0xea1c	P	protocol witness for Swift.Hashable._rawHashValue(seed: Swift.Int) -> Swift.Int in conform...
0xea38	P	protocol witness for static Swift.CaseIterable.allCases.getter : A.AllCases in conformance I...
0xea60	P	IOSSecuritySuite.JailbreakChecker.JailbreakStatus.passed.getter : Swift.Bool
0xea68	P	IOSSecuritySuite.JailbreakChecker.JailbreakStatus.failMessage.getter : Swift.String
0xeaa0	P	IOSSecuritySuite.JailbreakChecker.JailbreakStatus.failedChecks.getter : [(check: IOSSecu...
0xeadc	P	IOSSecuritySuite.JailbreakChecker.JailbreakStatus.init(passed: Swift.Bool, failMessage: S...
0xeae4	P	static IOSSecuritySuite.JailbreakChecker.amIJailbroken() -> Swift.Bool
0xeb38	P	static IOSSecuritySuite.JailbreakChecker.(performChecks in _F8E503CD913F87B6FC3E9...
0xf198	P	static IOSSecuritySuite.JailbreakChecker.amIJailbrokenWithFailMessage() -> (jailbroken:...
0xf21c	P	static IOSSecuritySuite.JailbreakChecker.amIJailbrokenWithFailedChecks() -> (jailbroken:...
0xf298	P	static IOSSecuritySuite.JailbreakChecker.(checkURLSchemes in _F8E503CD913F87B6FC...
0xf718	P	static IOSSecuritySuite.JailbreakChecker.(checkExistenceOfSuspiciousFiles in _F8E503C...
0x10280	P	static IOSSecuritySuite.JailbreakChecker.(checkSuspiciousFilesCanBeOpened in _F8E503...
0x10824	P	static IOSSecuritySuite.JailbreakChecker.(checkRestrictedDirectoriesWriteable in _F8E50...
0x112a8	P	static IOSSecuritySuite.JailbreakChecker.(checkFork in _F8E503CD913F87B6FC3E966D6...
0x11574	P	static IOSSecuritySuite.JailbreakChecker.(checkSymbolicLinks in _F8E503CD913F87B6F...
0x11c0c	P	static IOSSecuritySuite.JailbreakChecker.(checkDYLD in _F8E503CD913F87B6FC3E966D...
0x123a4	P	static IOSSecuritySuite.JailbreakChecker.(canOpenUrlFromList in _F8E503CD913F87B6F...

Figure 2 - Functions containing the string "Jailbr"

My first goal is to hook each function and see if it does affect the alert message that we saw before.

Bypass canOpenUrlFromList

The first function we are going to analyse is ***_$s16IOSSecuritySuite16JailbreakCheckerC18canOpenUrlFromList33_F8E503CD913F87B6F3E966D69D 813ABLL10urlSchemesSb6passed_SS11failMessagetSaySSG_tFZ.***

By reversing it, we can see the canOpenURL method call at the address offset **0x126bc**. According with the AArch64 ABI, the **x0** register will contain the value returned from the function call. This function usually tries to open suspicious URL scheme such as “**cydia://**” in order to identify jailbreak artefact. When it founds any suspicious file, the function returns “true”. Because the value returned is a Boolean it should contain the values **0x01** (**true**) or **0x00** (**false**), therefore we have to change it to **0x00** in order to bypass this check.

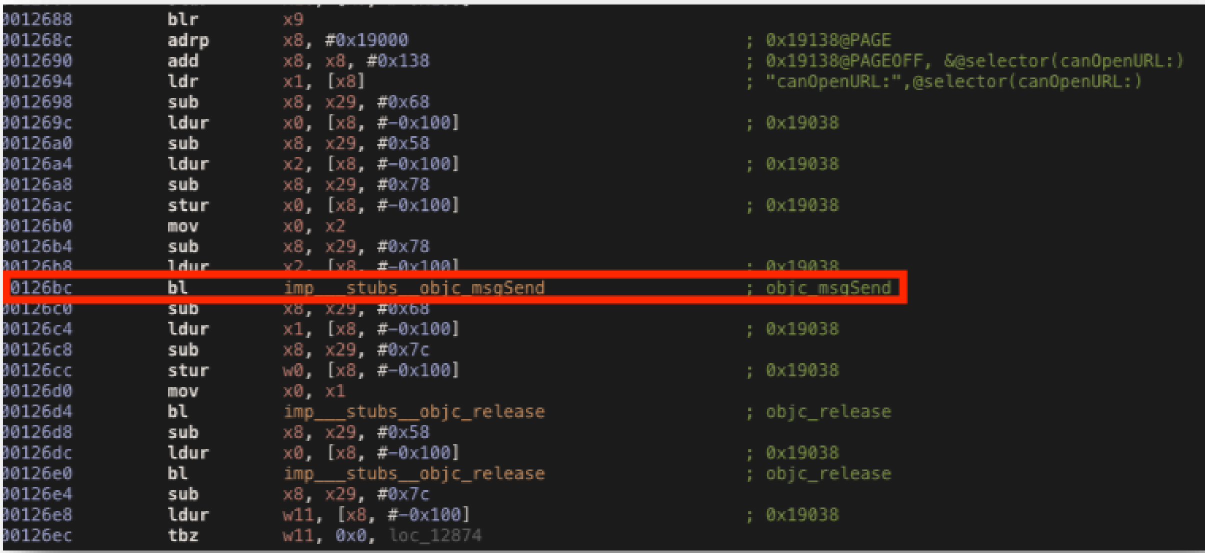


Figure 3 - canOpenURL

We can use Frida’s Module.getBaseAddress() function to obtain the base address where the library is loaded in memory (Frida calculate the ASLR Shift for us) and then we have to add the **0x126c** offset to it. Then we ask Frida to hook and replace the code at that specific address (baseAddress+offset). Our implementations will change the **x0** register value from **0x01** (**true**) to **0x00** (**false**). In Frida we can access register’s values by using the this.context object.

Frida Code

```
var targetModule = 'IOSSecuritySuite';
var addr = ptr(0x126c0);
var moduleBase = Module.getBaseAddress(targetModule);
var targetAddress = moduleBase.add(addr);
   Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                    this.context.x0=0x00
```

```
                        console.log("Bypass canOpenUrlFromList");
                }
            },
        });
```

Running the script, we can see that the target instruction is reached and the check is bypassed as shown in the images below.

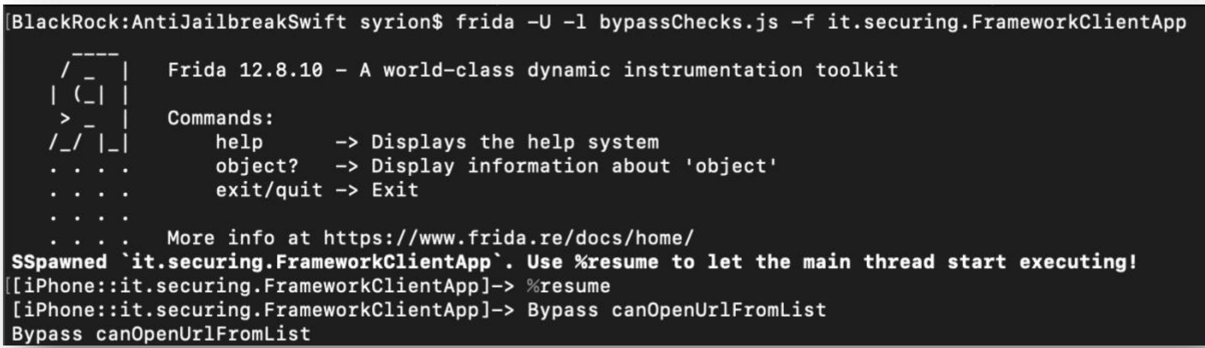


Figure 4 - Frida Script

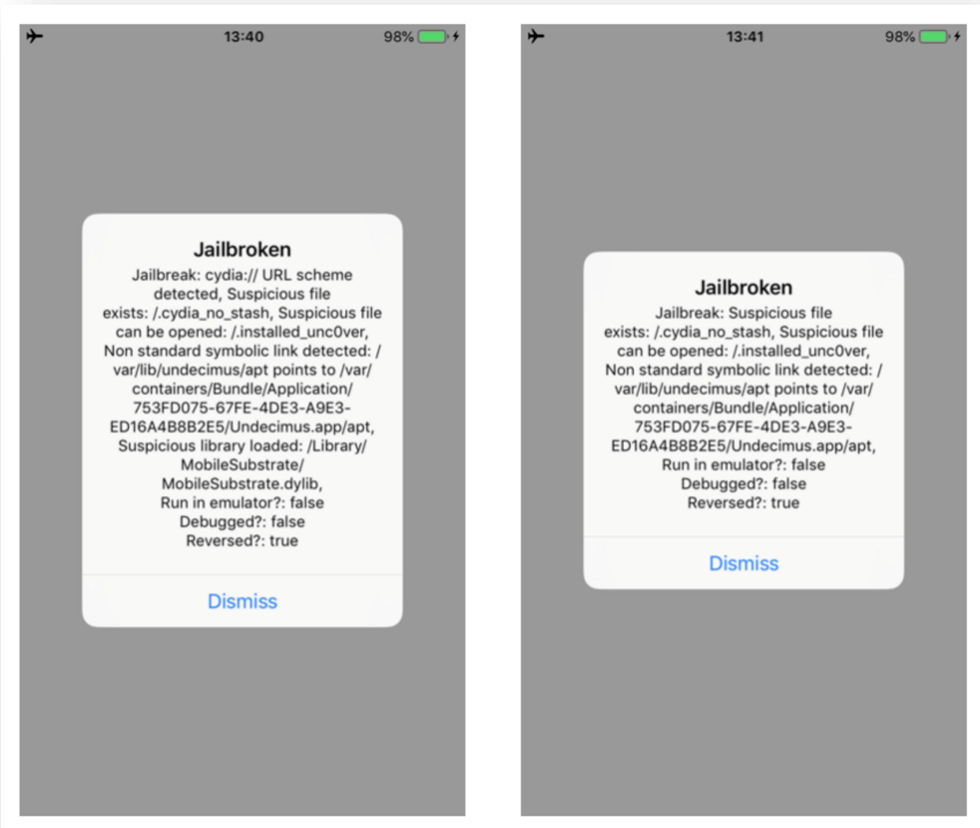


Figure 5 – Security Checks

N.B. During the analysis it was observed that some Swift functions defined within the **IOSSecuritySuite** library, executed **Objective-C** code by “bridging” them. It should be noted that in such circumstances, controls can also be bypassed by using the “**ObjC**” object defined within Frida’s API.

Bypass checkExistenceOfSuspiciousFiles

Function:
_$s16IOSSecuritySuite16JailbreakCheckerC31checkExistenceOfSuspiciousFiles33_F8E503C913F87 B6FC3E966D69D813ABLLSb6passed_SS11failMessagetyFZ

In the disassembled code below we can see that the function calls the **fileExistsAtPath** method at the offset address **0x100a8**, so we need to change the return value as we did before.

Figure 6 - fileExistsAtPath

Using the same Frida code, we can set the new target address, and bypass the check.

Frida Code

```
addr = ptr(0x100ac);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                    this.context.x0=0x00
                    console.log("Bypass checkExistenceOfSuspiciousFiles");
                }
        },
    });
```

Running the updated script, we can bypass the two methods. As shown in the images below. Moreover, the message in the alert box will change as well.

```
[BlackRock:AntiJailbreakSwift syrion$ frida -U -l bypassChecks.js -f it.securing.FrameworkClientApp
     ____
    / _  |   Frida 12.8.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `it.securing.FrameworkClientApp`...
Spawned `it.securing.FrameworkClientApp`. Use %resume to let the main thread start executing!
[iPhone::it.securing.FrameworkClientApp]-> %resume
[iPhone::it.securing.FrameworkClientApp]-> Bypass canOpenUrlFromList
Bypass checkExistenceOfSuspiciousFiles
```

Figure 7 - Frida Script

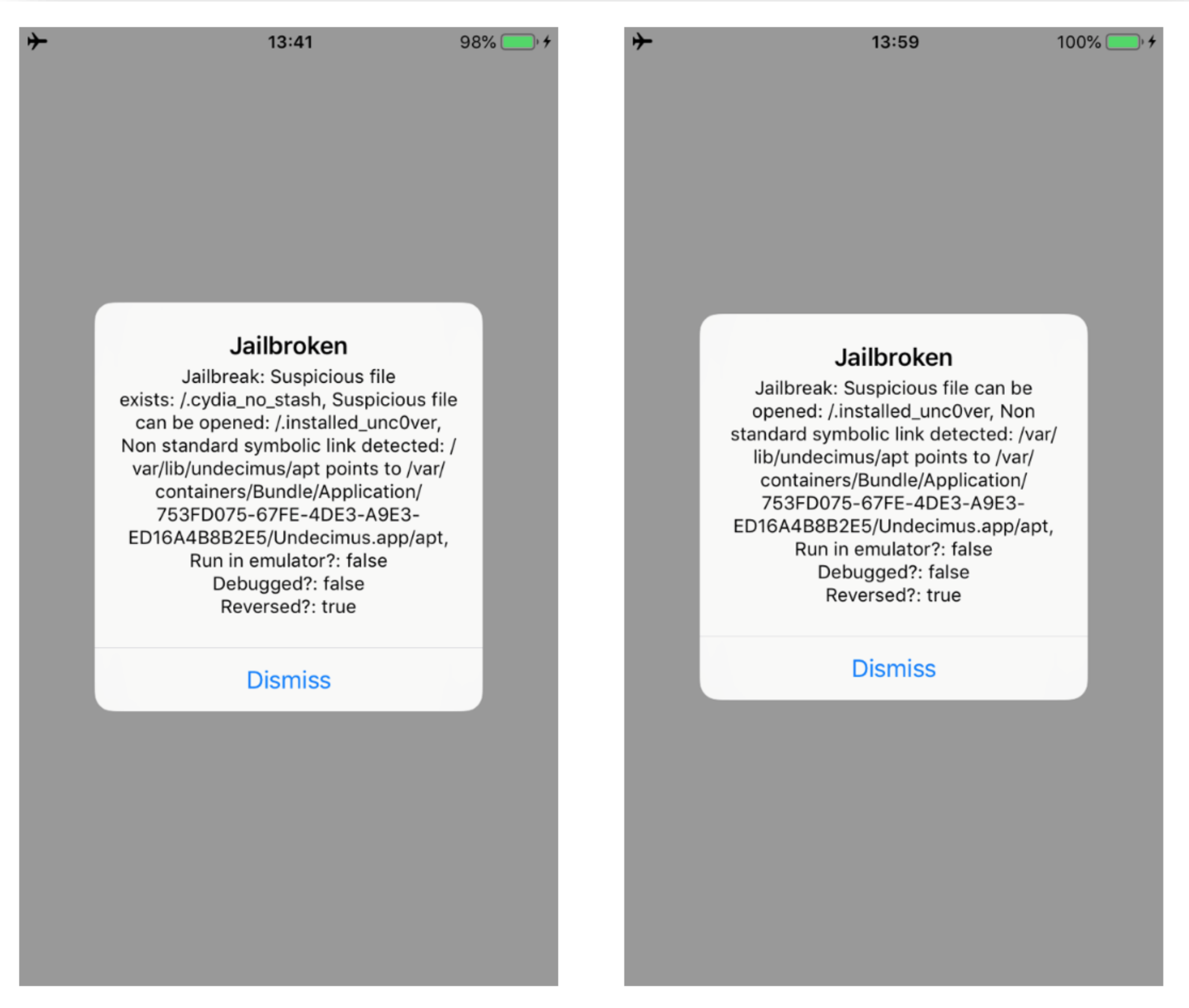


Figure 8 - Security Checks

Bypass checkSuspiciousFilesCanBeOpened

Function:
_$s16IOSSecuritySuite16JailbreakCheckerC31checkSuspiciousFilesCanBeOpened33_F8E50 3CD913F87 B6FC3E966D69D813ABLLSb6passed_SS11failMessagetyFZ

As we can see in the disassembled code below, the **isReadableFileAtPath** method is called at the offset address **0x1064c** and the return value is stored in the **x0** register as usual.

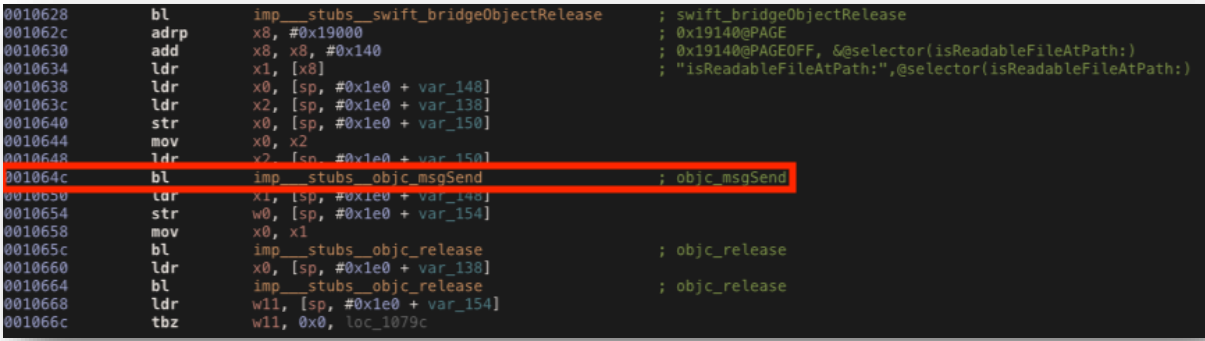


Figure 9 - isReadableFileAtPath

Using the same script with the new address we can bypass the check.

Frida Code

```
addr = ptr(0x10650);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                    this.context.x0=0x00
                    console.log("Bypass checkSuspiciousFilesCanBeOpened");
                }
        },
    });
```

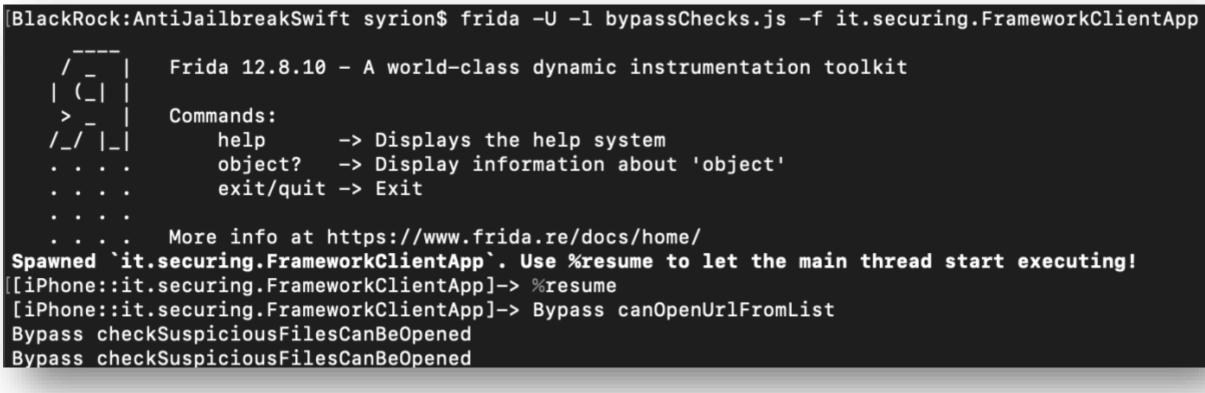


Figure 10 - Frida Script

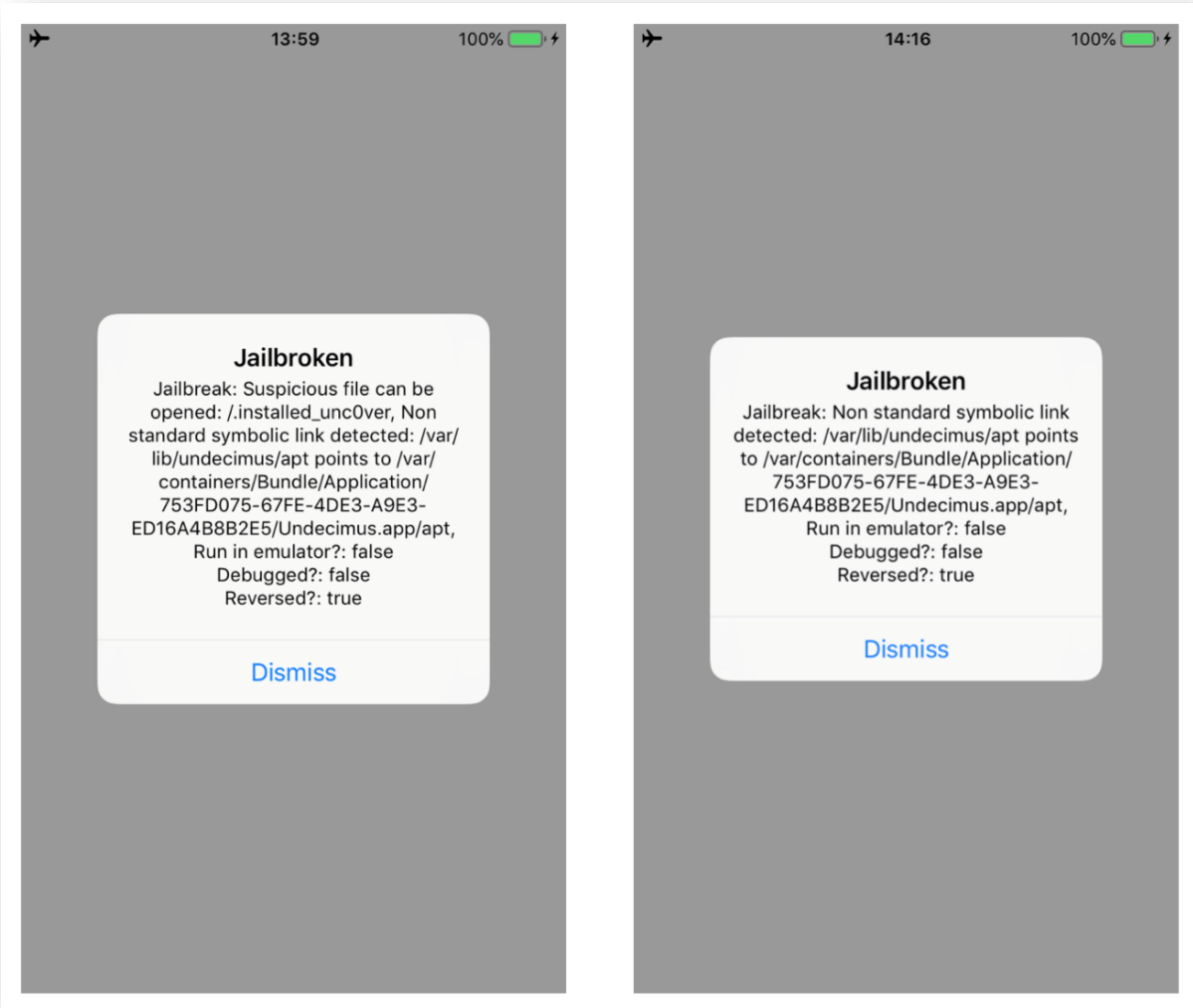


Figure 11 - Security Checks

Bypass checkSymbolicLinks

Function:
_$s16IOSSecuritySuite16JailbreakCheckerC18checkSymbolicLinks33_F8E503CD913F87B6F C3E966 D69D813ABLLSb6passed_SS11failMessagetyFZ

Our iPhone is still recognized as "**jailbroken**", so we need to bypass other methods. As we can see below, the **destinationOfSymbolicLinkAtPath** method is called at the offset address **0x118ac**, so we can modify the **x0** register by replacing the its value with **0x00** as we did before.

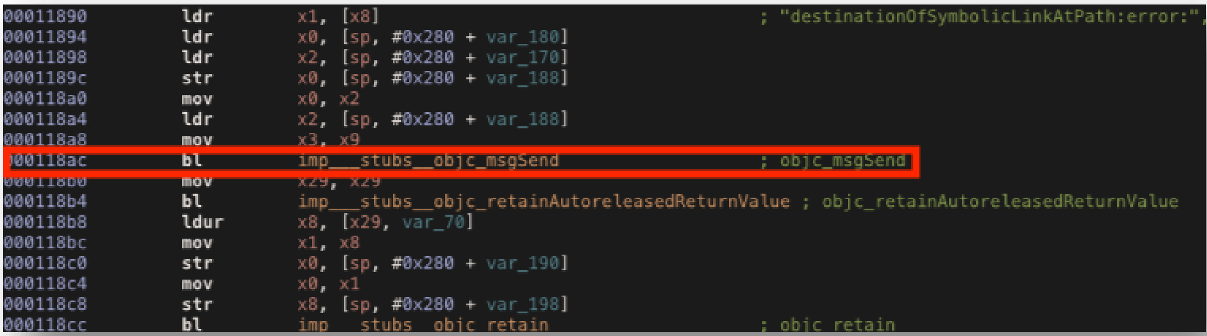


Figure 12 - destinationOfSymbolicLinkAtPath

Using the same script we can change the return value contained in the **x0** register.

Frida Code

```
addr = ptr(0x118b0);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 != 0x00){
                    this.context.x0 = 0x00
                    console.log("Bypass checkSymbolicLinks");
                }
        },
    })
```

```
BlackRock:AntiJailbreakSwift syrion$ frida -U -l bypassChecks.js -f it.securing.FrameworkClientApp
     ____
    / _  |   Frida 12.8.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `it.securing.FrameworkClientApp`...
Spawned `it.securing.FrameworkClientApp`. Use %resume to let the main thread start executing!
[iPhone::it.securing.FrameworkClientApp]-> %resume
[iPhone::it.securing.FrameworkClientApp]-> Bypass checkSymbolicLinks
Bypass checkSymbolicLinks
```

Figure 13 - Frida Script

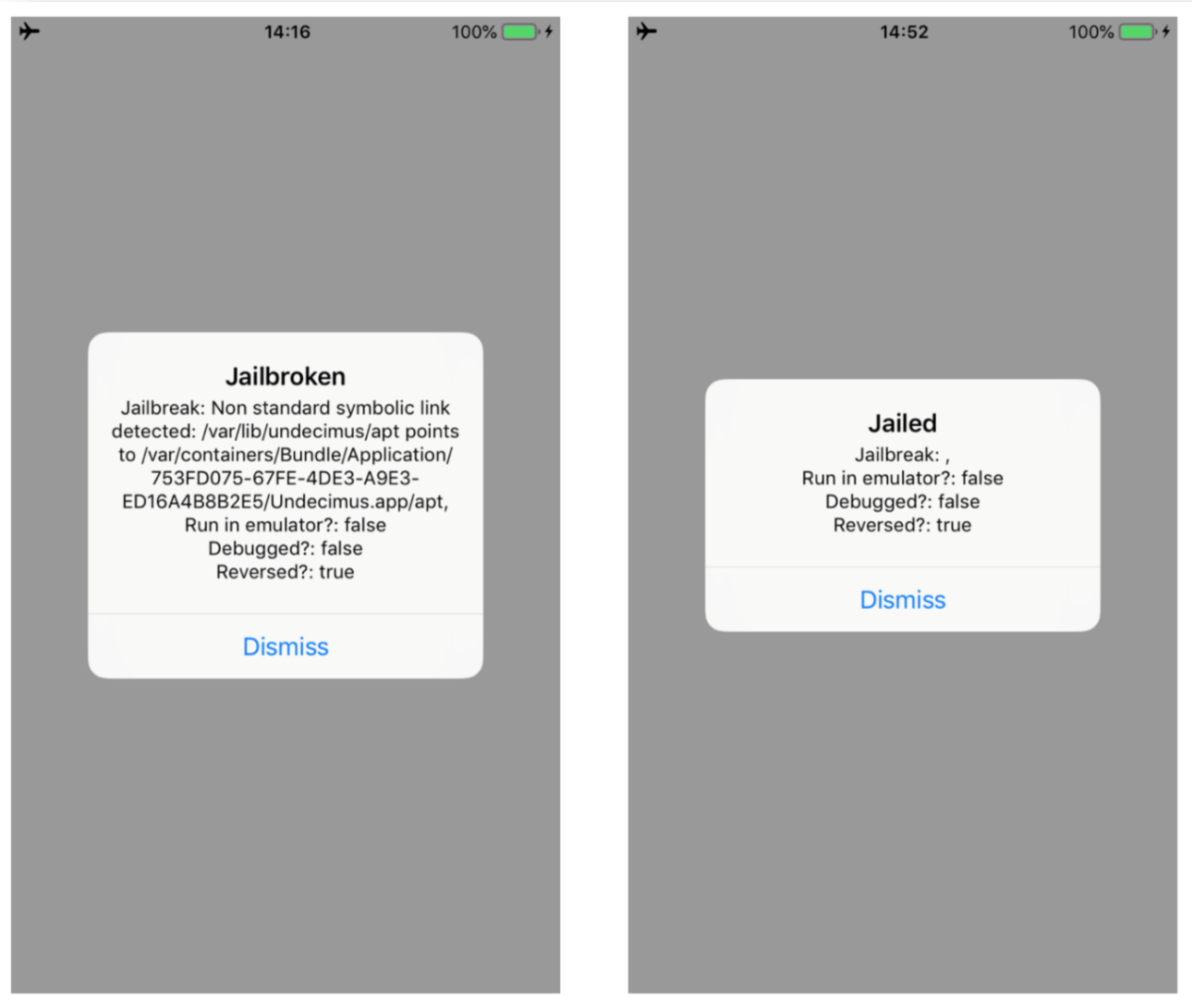


Figure 14 - Security Checks

We successfully bypassed all the Anti-Jailbreak checks.

Bypass amIReversed

Function: **_$s16IOSSecuritySuiteAAC20amIReverseEngineeredSbyFZ**

The method **amIReversed** checks if the application is debugged or tampered using instrumentation tools like Frida. Again, we can change its return value contained in the **x0** register, before the **RET** instruction is executed at the offset address **0xaea8**.

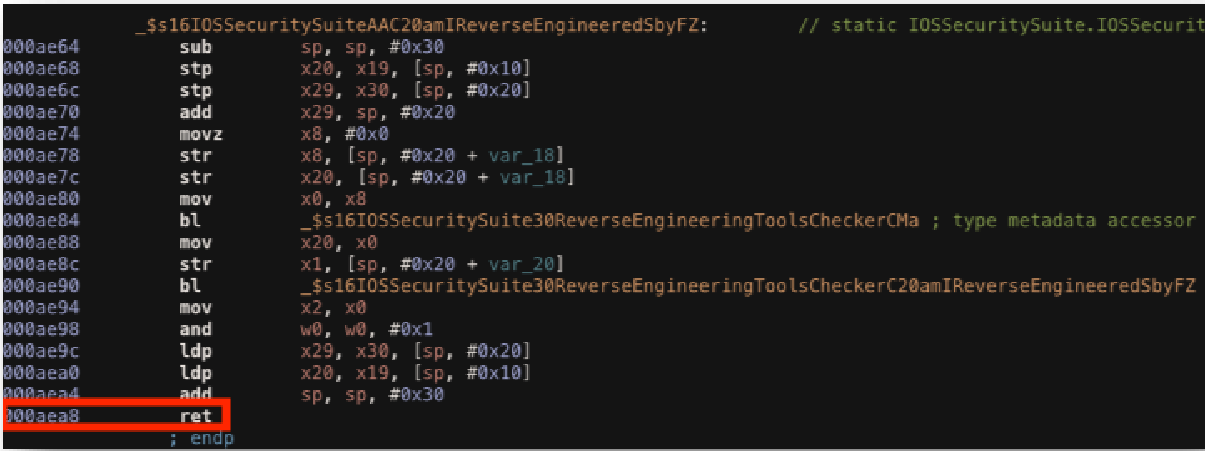


Figure 15 - amIReverseEngineered

Frida Code

```
addr = ptr(0xaea8);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
  Interceptor.attach(targetAddress, {
    onEnter: function(args) {
        if(this.context.x0 == 0x01){
          this.context.x0=0x00
           console.log("Bypass amIReverseEngineered");
        }
    },
  });
```

```
[BlackRock:AntiJailbreakSwift syrion$ frida -U -l bypassChecks.js -f it.securing.FrameworkClientApp
     ____
    / _  |   Frida 12.8.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawned `it.securing.FrameworkClientApp`. Use %resume to let the main thread start executing!
[iPhone::it.securing.FrameworkClientApp]-> %resume
[iPhone::it.securing.FrameworkClientApp]-> Bypass amIReverseEngineered
```

Figure 16 - Frida Script

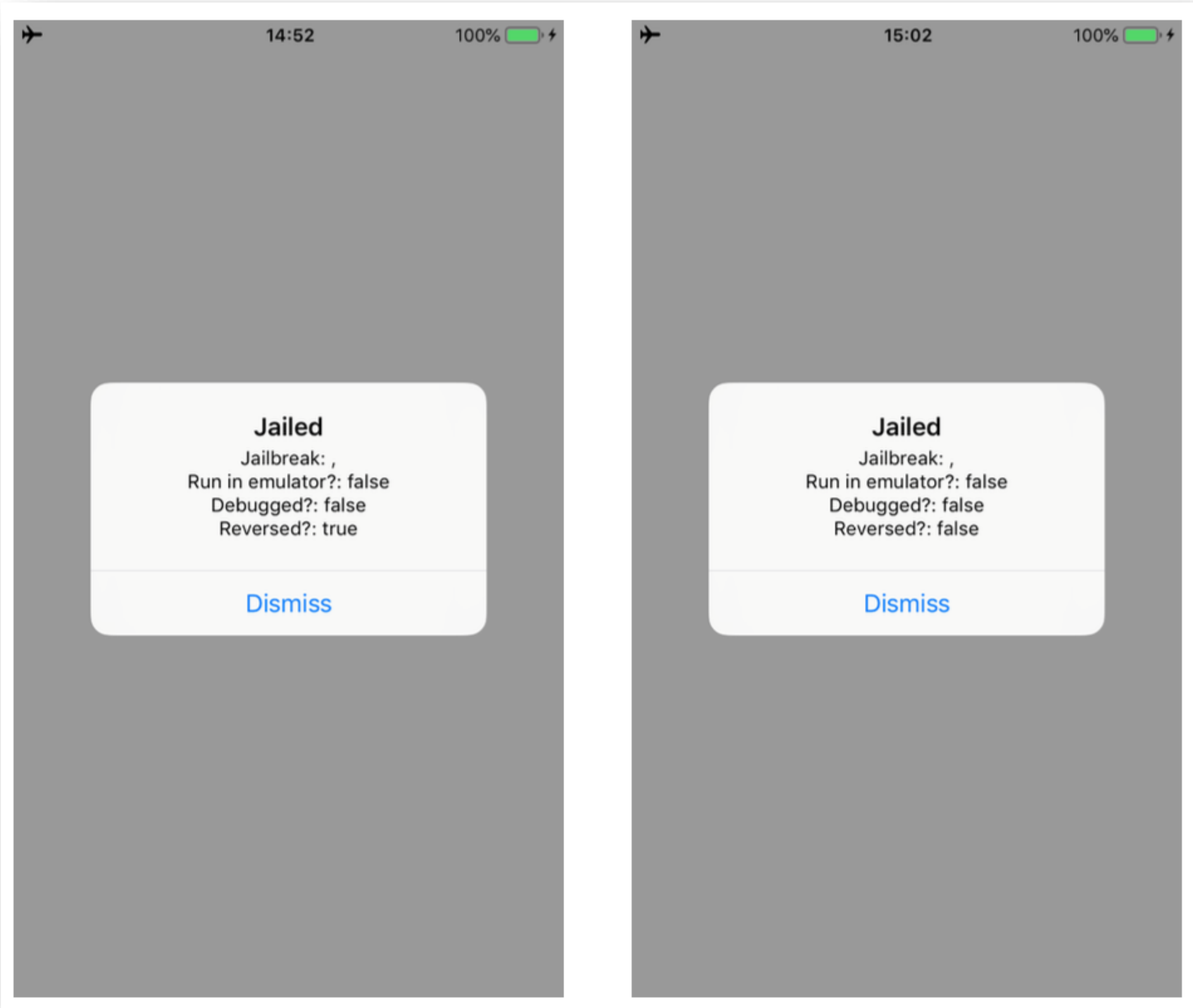
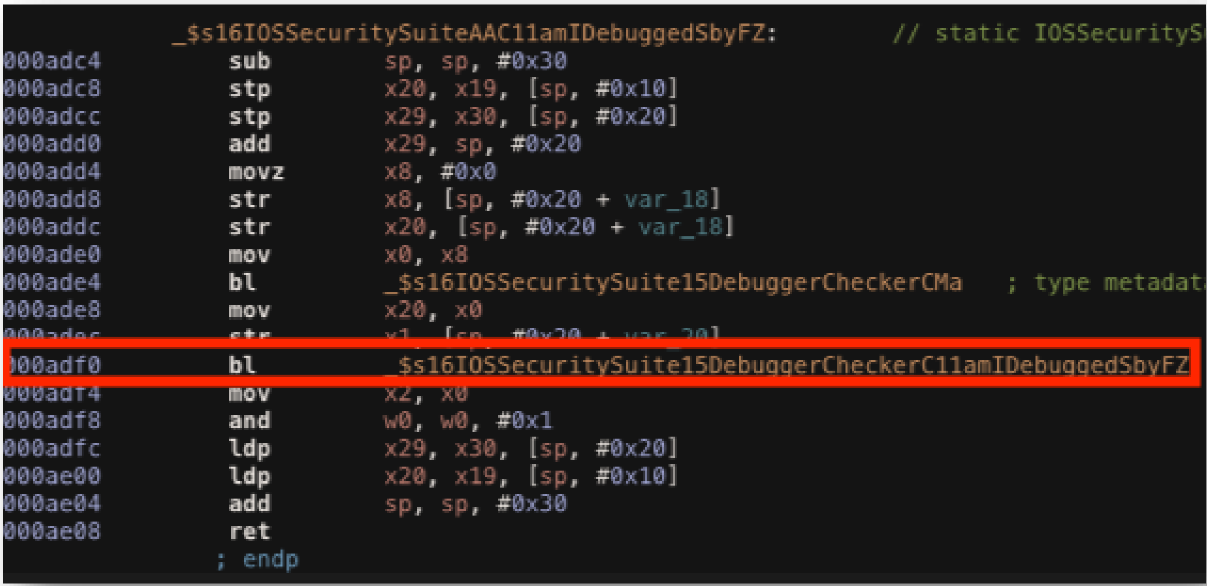


Figure 17 - Security Checks

Bypass AmIDebugged

Function: **_$s16IOSSecuritySuiteAAC11amIDebuggedSbyFZ**



```
            _$s16IOSSecuritySuiteAAC11amIDebuggedSbyFZ:        // static IOSSecurityS
000adc4         sub         sp, sp, #0x30
000adc8         stp         x20, x19, [sp, #0x10]
000adcc         stp         x29, x30, [sp, #0x20]
000add0         add         x29, sp, #0x20
000add4         movz        x8, #0x0
000add8         str         x8, [sp, #0x20 + var_18]
000addc         str         x20, [sp, #0x20 + var_18]
000ade0         mov         x0, x8
000ade4         bl          _$s16IOSSecuritySuite15DebuggerCheckerCMa    ; type metadat
000ade8         mov         x20, x0
000adec         str         x1, [sp, #0x20 + var_20]
000adf0         bl          _$s16IOSSecuritySuite15DebuggerCheckerC11amIDebuggedSbyFZ
000adf4         mov         x2, x0
000adf8         and         w0, w0, #0x1
000adfc         ldp         x29, x30, [sp, #0x20]
000ae00         ldp         x20, x19, [sp, #0x10]
000ae04         add         sp, sp, #0x30
000ae08         ret
                ; endp
```

Figure 18 - AmIDebugged

The function AmIDebugged checks if the application is debugged, following the method call, we can see there is another method **_$s16IOSSecuritySuite15DebuggerCheckerC11amIDebuggedSbyFZ** looking for a debugger using sysctl and getPid.

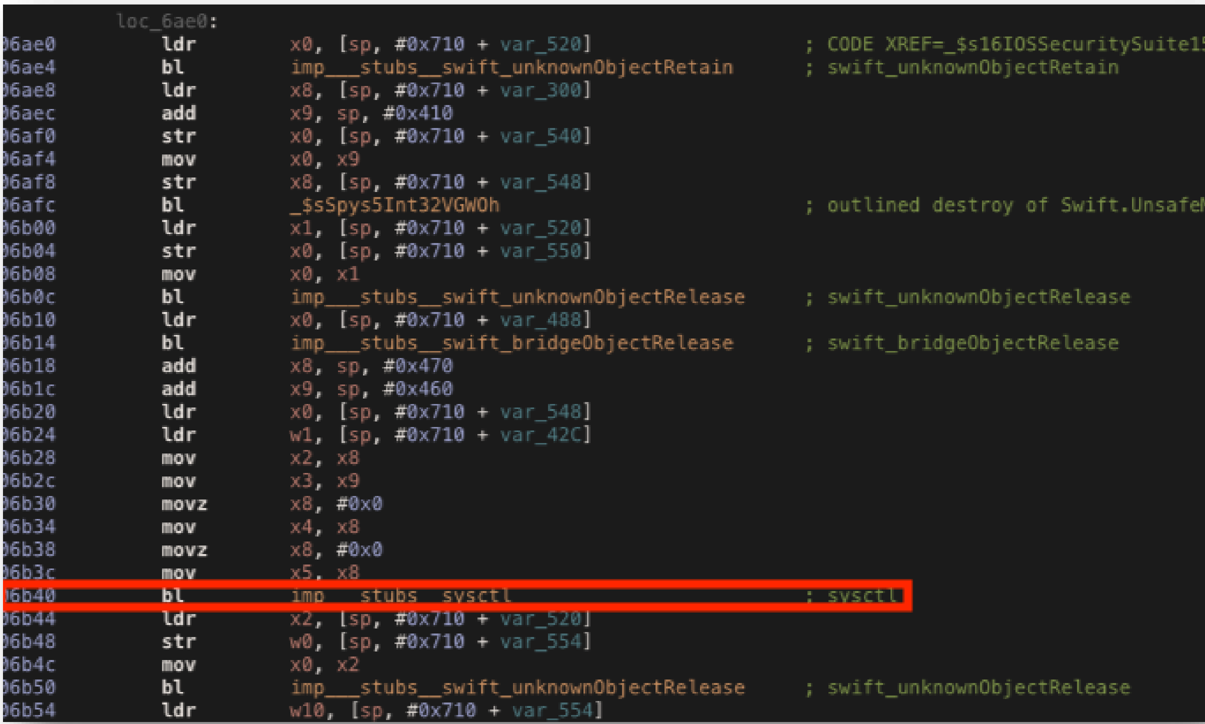


Figure 19 - sysctl

If we try to debug the application, the check will return true but we can hook the ret instruction at address **0xae08** and change the return value contained in **x0** to **0x00** with the following Frida script.

Frida Code

```
addr = ptr(0xae08);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                   this.context.x0=0x00
                    console.log("Bypass amIDebugged");
                }
        },
    });
```

```
[iPhone::it.securing.FrameworkClientApp]->

Thank you for using Frida!
BlackRock:AntiJailbreakSwift syrion$ frida -U -l bypassChecks.js -f it.securing.FrameworkClientApp
     ____
    / _  |   Frida 12.8.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawned `it.securing.FrameworkClientApp`. Use %resume to let the main thread start executing!
[iPhone::it.securing.FrameworkClientApp]-> %resume
[iPhone::it.securing.FrameworkClientApp]-> Bypass amIDebugged
```

Figure 20 - Frida Script

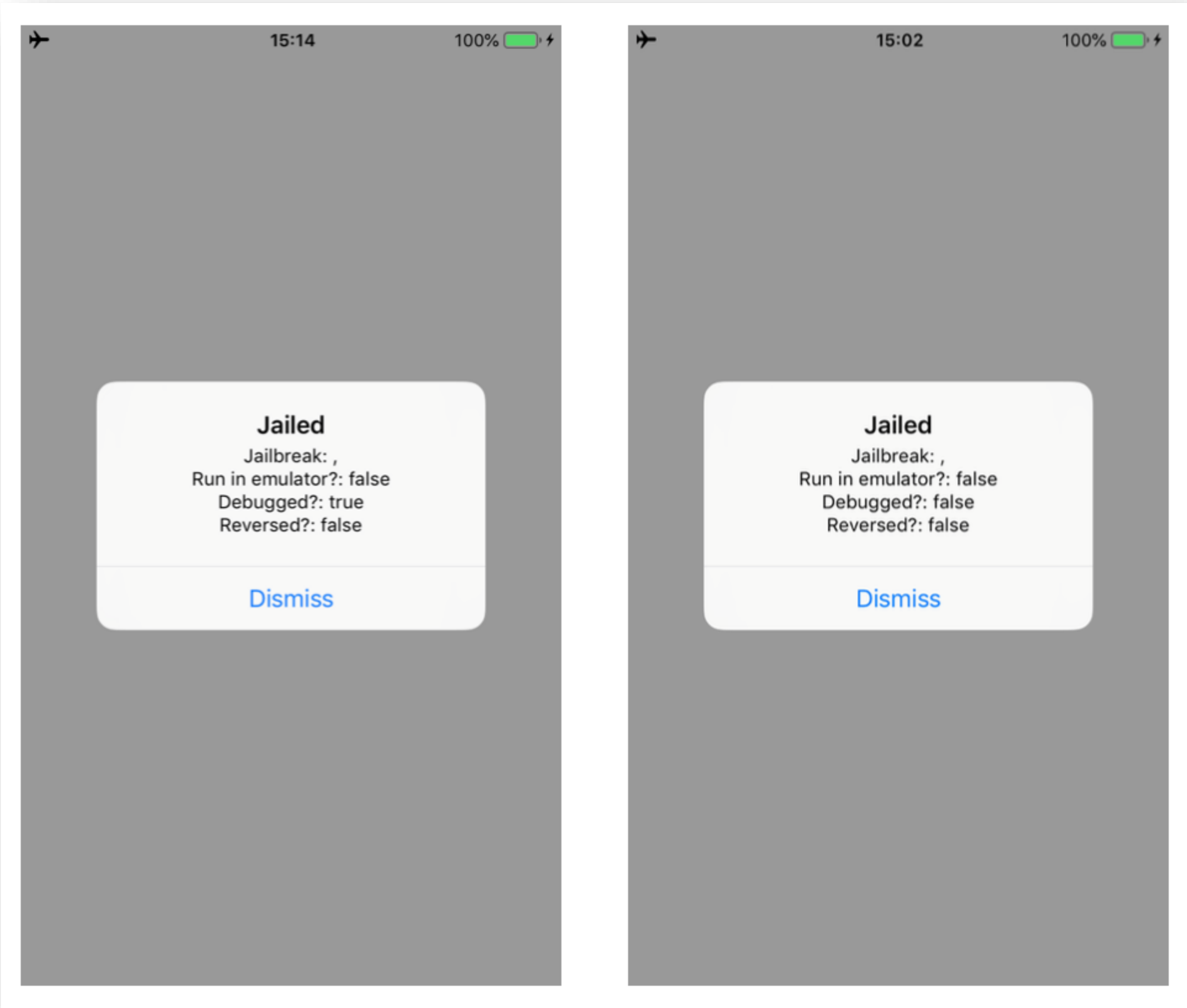


Figure 21 - Security Checks

Bypass amIRunInEmulator

Function: **_$s16IOSSecuritySuite15EmulatorCheckerC08amIRunInC0SbyFZ**

Because the application is running on a real iPhone, the **amIrunInEmulator** check can't be triggered, therefore what we can do is to inject a true value and let the application believe that is running in an emulator. The image below shows the method. As always, we have to change the **x0** register value before the **RET** instruction at the offset address **0xa880** is executed.

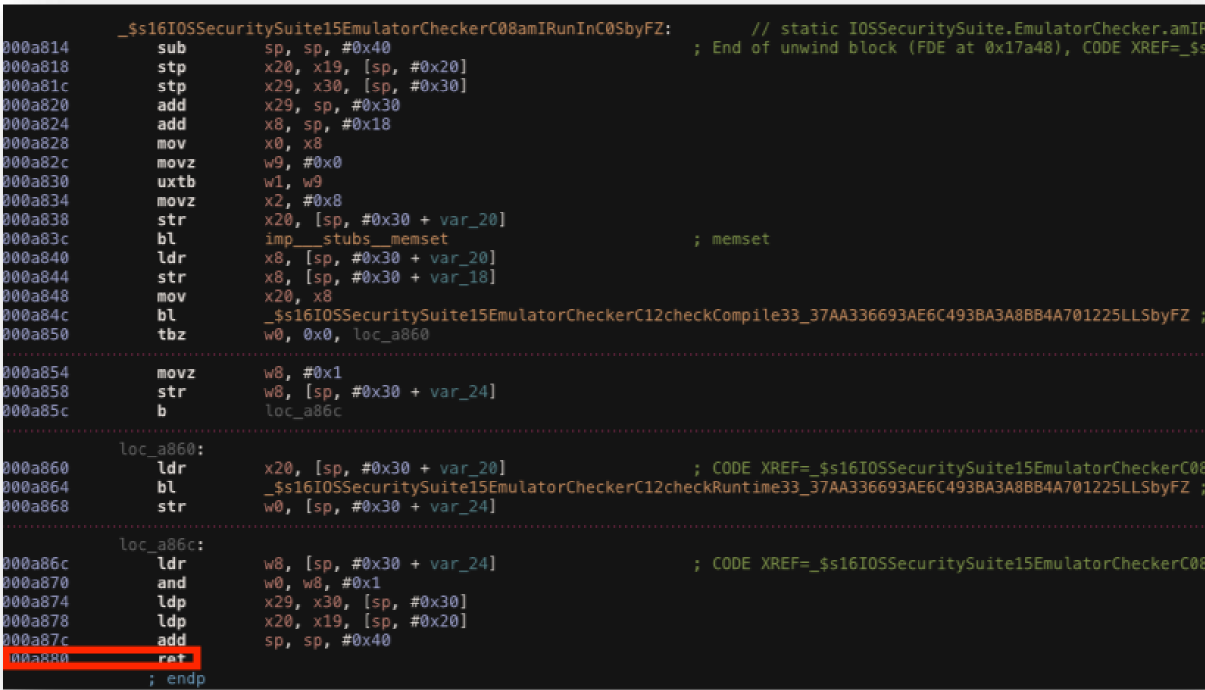


Figure 22 - amIRunInEmulator

We can "enable" the emulator with the following Frida script.

Frida Code

```
addr = ptr(0xa880);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x00){
                   this.context.x0=0x01
                    console.log("Enable amIRunInEmulator");
                }
        },
    });
```

```
BlackRock:AntiJailbreakSwift syrion$ frida -U -l bypassChecks.js -f it.securing.FrameworkClientApp
     ____
    / _  |   Frida 12.8.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://www.frida.re/docs/home/
Spawning `it.securing.FrameworkClientApp`...
Spawned `it.securing.FrameworkClientApp`. Use %resume to let the main thread start executing!
[iPhone::it.securing.FrameworkClientApp]-> %resume
[iPhone::it.securing.FrameworkClientApp]-> Enable amIRunInEmulator
Enable amIRunInEmulator
Enable amIRunInEmulator
Enable amIRunInEmulator
Enable amIRunInEmulator
Enable amIRunInEmulator
Enable amIRunInEmulator
```

Figure 23 - Frida Script

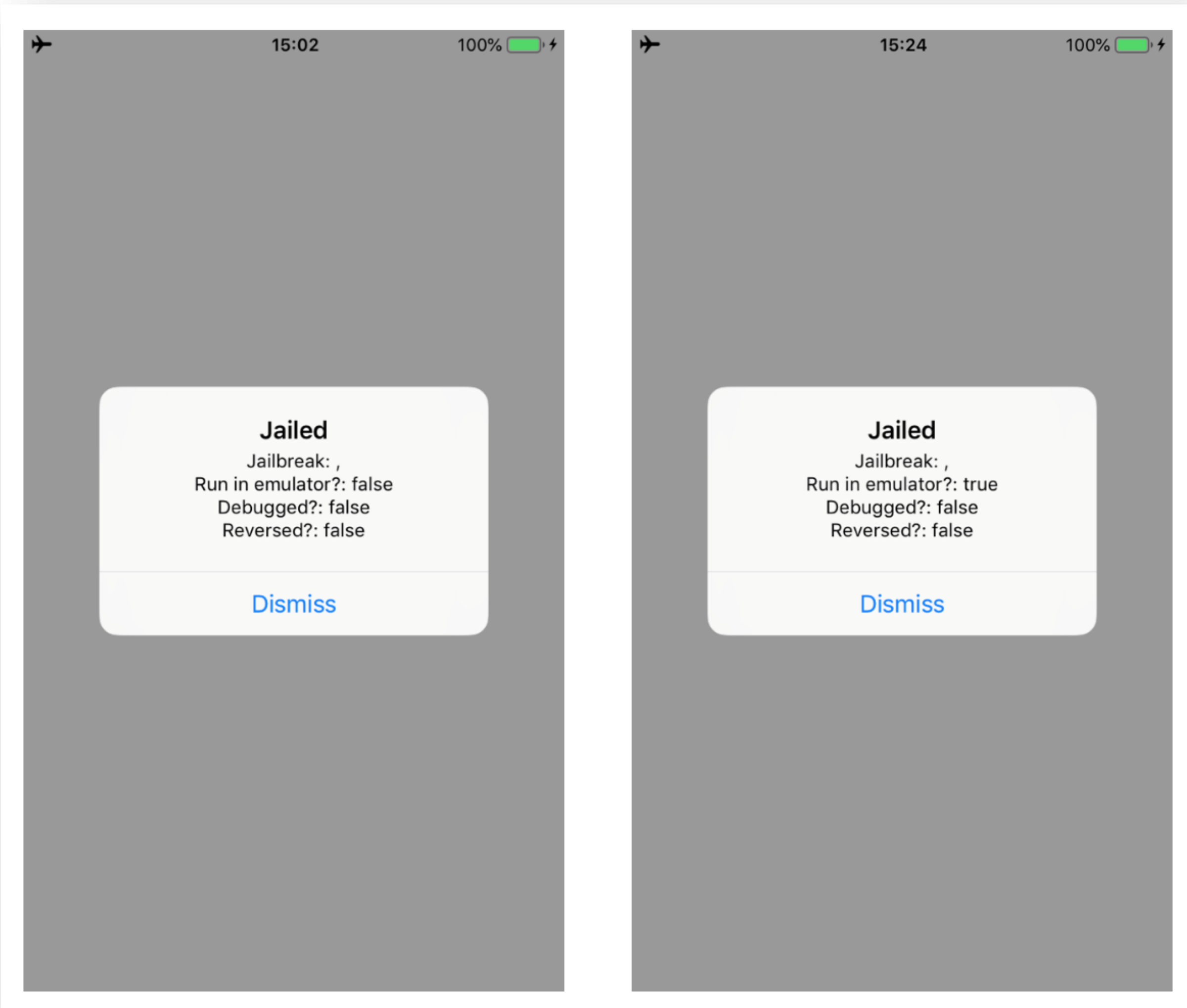


Figure 24 - Security Checks

Change the “Jailed” string

Finally, I wanted to change “Jailed” string for fun. However, its address is no writable so we can’t modify its value. What we can do, is creating our own string and put its address in the register that point to the string message. In order to do so, we need to reverse the FrameworkClientApp module. In the **_$s18FrameworkClientApp14ViewControllerC13viewDidAppearyySbF** methods, we can find the instruction where the “Jailed” string address is put into the **x0** register. We can see it at address **0x4348**.

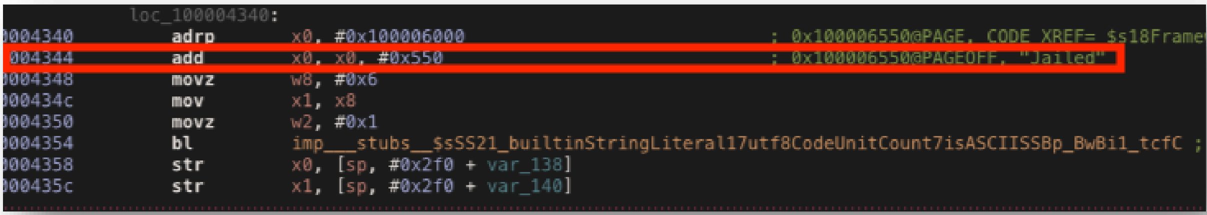


Figure 25 - Jailed String

We can create our own string and put its address into the **x0** register after the instruction at address **0x4348**.

Frida Code

```
targetModule = 'FrameworkClientApp';
addr = ptr(0x04348);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
var myMessage = Memory.allocUtf8String("Br0k3n")
    Interceptor.attach(targetAddress, {
         onEnter: function(args) {
                    this.context.x0 = myMessage;
        },
    });
```

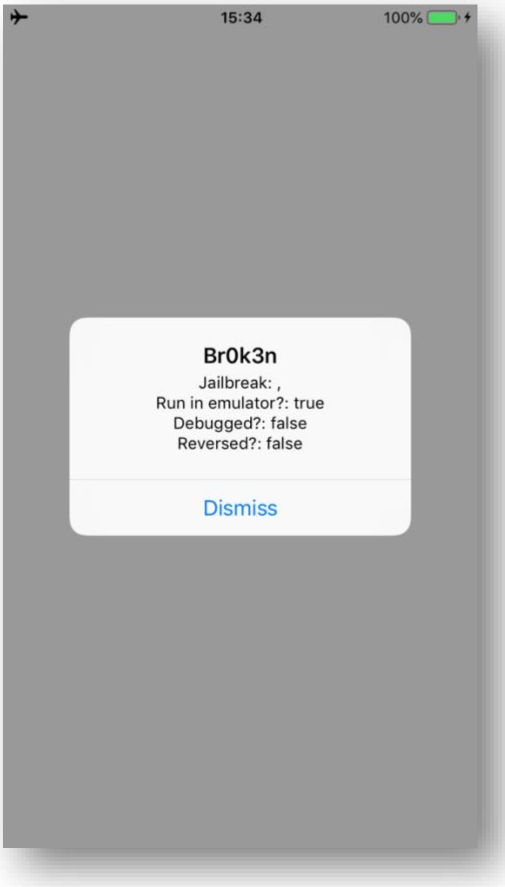


Figure 26 - Security Checks

This is the complete Frida script (which potentially can be written in a better way, so I will let this to you as exercise).

Complete Frida Script

```
var targetModule = 'IOSSecuritySuite';
var addr = ptr(0x126c0);
var moduleBase = Module.getBaseAddress(targetModule);
var targetAddress = moduleBase.add(addr);
   Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                    this.context.x0=0x00
                    console.log("Bypass canOpenUrlFromList");
            }
        },
    });

addr = ptr(0x100ac);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                    this.context.x0=0x00
                    console.log("Bypass checkExistenceOfSuspiciousFiles");
```

```
                }
        },
    });

addr = ptr(0x10650);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                    this.context.x0=0x00
                    console.log("Bypass checkSuspiciousFilesCanBeOpened");
                }
        },
    });

addr = ptr(0x118b0);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 != 0x00){
                    this.context.x0 = 0x00
                    console.log("Bypass checkSymbolicLinks");
                }

        },
    });

addr = ptr(0xaea8);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                   this.context.x0=0x00
                    console.log("Bypass amIReverseEngineered");
                }

        },
    });

addr = ptr(0xae08);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x01){
                   this.context.x0=0x00
                    console.log("Bypass amIDebugged");
                }

        },
    });

addr = ptr(0xa880);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
    Interceptor.attach(targetAddress, {
        onEnter: function(args) {
                if(this.context.x0 == 0x00){
```

```
                   this.context.x0=0x01
                    console.log("Enable amIRunInEmulator");
                }
        },
    });

targetModule = 'FrameworkClientApp';
addr = ptr(0x04348);
moduleBase = Module.getBaseAddress(targetModule);
targetAddress = moduleBase.add(addr);
var myMessage = Memory.allocUtf8String("Br0k3n")
    Interceptor.attach(targetAddress, {
         onEnter: function(args) {
                    this.context.x0 = myMessage;
        },
    });
```

Conclusion

As we have seen, Frida provides a very powerful way to attach each arm instruction and interact with register and memory. Its documentation is very clear and full of examples. I hope you enjoined it. If you find any mistakes in my write-up, please contact me and I will be more than happy to fix them.