6 July 2023

Breaking Mobile App Defenses with Frida and Reverse Engineering

CyberCX

!whoami

Shofe Miraz

@shmi012

Security Consultant at CyberCX

- Working in cyber security for 4 years.
- I like playing cricket, photography and presenting.
- You may recognize me from.. **HackAndLearn** monthly meetup.

user@pc:~$ sudo whoami

Why this talk?

Today’s Agenda

Popularity of cross-platform frameworks

Anti-Tampering Libraries

Challenges

How it works?

Bypassing Checks

Benefits of added friction

Cross-Platform frameworks

But.. what is Jailbreak?

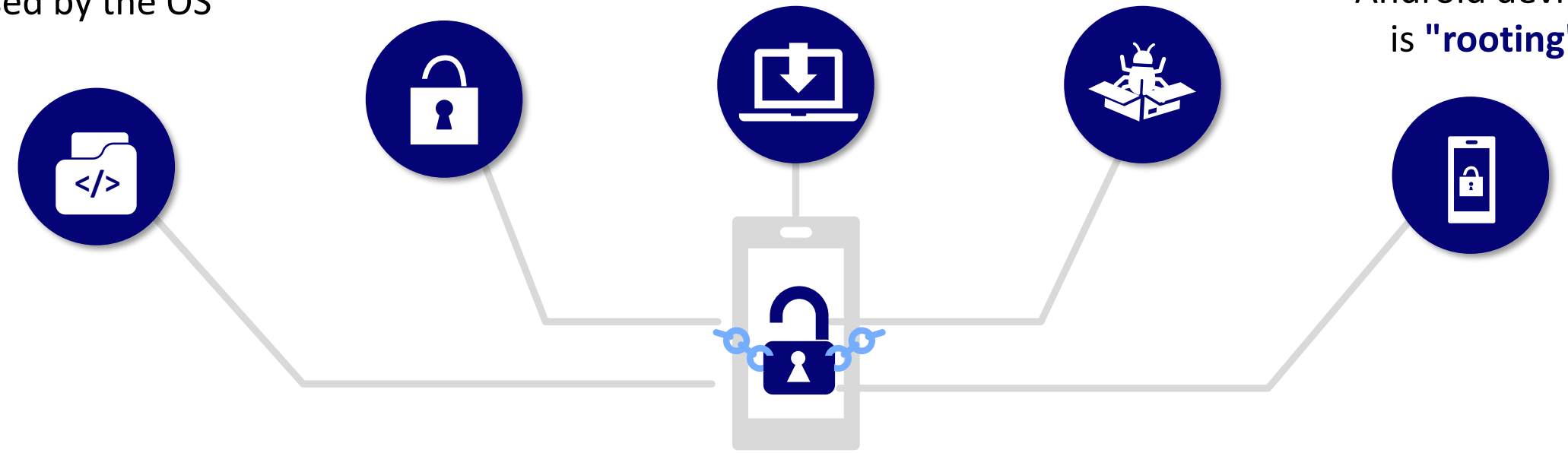
Remove software restrictions imposed by the OS

Offers **privileged access and customization** beyond manufacturer limitations

Install **unauthorized apps** and tweaks

Bypass security mechanisms and sandboxing restrictions

Equivalent process on Android devices is **"rooting"**



Anti-Tampering Libraries

iOS Security Suite
Open Source Anti-tampering Swift Library

Prevent running on **modified** or higher-privilege devices

Detect **tampering** or **modifications**

Block **reverse engineering** attempts

Block **anti-debugging** techniques

Usage

Gaming apps

Financial Apps

Challenges

Dealing with Anti-Tampering Libraries

Time Consuming

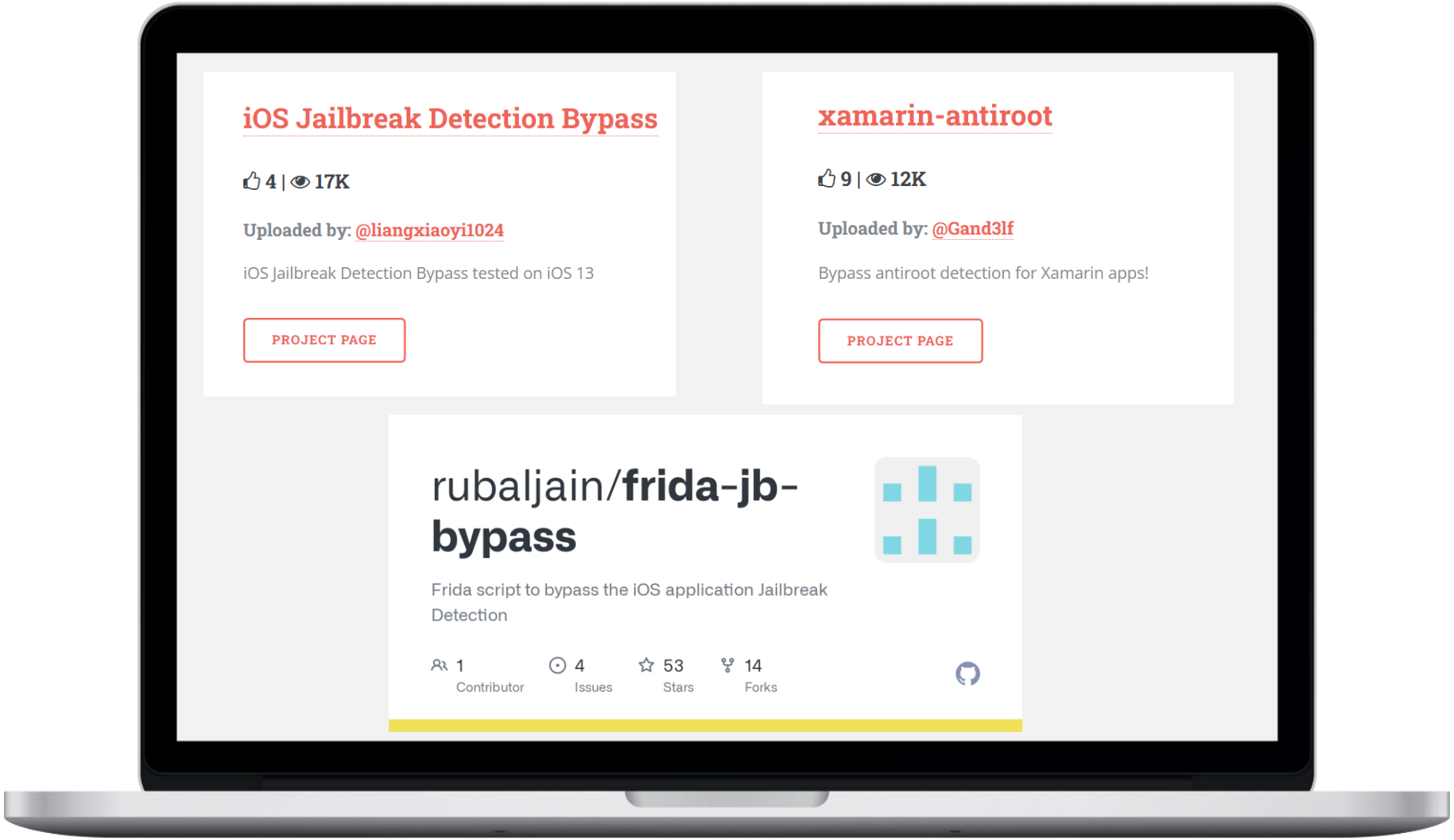
Extensive checks performed by popular frameworks

Limited visibility into the inner workings

Lack of **access to source code** for precompiled applications

Need for **reverse engineering techniques**.

Challenges | So many resources…



How it works?

Let’s build an app


```swift
public class SwiftFlutterJailbreakDetectionPlugin: NSObject, FlutterPlugin {
  public static func register(with registrar: FlutterPluginRegistrar) {
    let channel = FlutterMethodChannel(name: "flutter_jailbreak_detection", binaryMessenger: registrar.messenger())
    let instance = SwiftFlutterJailbreakDetectionPlugin()
    registrar.addMethodCallDelegate(instance, channel: channel)
  }

  public func handle(_ call: FlutterMethodCall, result: @escaping FlutterResult) {
        switch call.method {
        case "jailbroken":

            let check2 = IOSSecuritySuite.amIJailbroken()
            result(check2)
            break
        case "developerMode":
            result(IOSSecuritySuite.amIRunInEmulator())
            break
```

How it works?

```
25      static func amIJailbroken() -> Bool {
26          return !performChecks().passed
27      }
28
29      static func amIJailbrokenWithFailMessage() -> (jailbroken: Bool, failMessage: String) {
30          let status = performChecks()
31          return (!status.passed, status.failMessage)
32      }
33
34      static func amIJailbrokenWithFailedChecks() -> (jailbroken: Bool, failedChecks: [Failed
35          let status = performChecks()
36          return (!status.passed, status.failedChecks)
37      }
```

IOSSecuritySuite is Open Source!

```
39      private static func performChecks() -> JailbreakStatus {
40          var passed = true
41          var failMessage = ""
42          var result: CheckResult = (true, "")
43          var failedChecks: [FailedCheckType] = []
44
45          for check in FailedCheck.allCases {
46              switch check {
47              case .urlSchemes:
48                  result = checkURLSchemes()
49              case .existenceOfSuspiciousFiles:
50                  result = checkExistenceOfSuspiciousFiles()
51              case .suspiciousFilesCanBeOpened:
52                  result = checkSuspiciousFilesCanBeOpened()
53              case .restrictedDirectoriesWriteable:
54                  result = checkRestrictedDirectoriesWriteable()
55              case .fork:
56                  if !EmulatorChecker.amIRunInEmulator() {
57                      result = checkFork()
```

How it works?

```
124     private static func checkExistenceOfSuspiciousFiles() -> CheckResult {
125         var paths = [
126             "/var/mobile/Library/Preferences/ABPattern", // A-Bypass
127             "/usr/lib/ABDYLD.dylib", // A-Bypass,
128             "/usr/lib/ABSubLoader.dylib", // A-Bypass
129             "/usr/sbin/frida-server", // frida
130             "/etc/apt/sources.list.d/electra.list", // electra
131             "/etc/apt/sources.list.d/sileo.sources", // electra
132             "/.bootstrapped_electra", // electra
133             "/usr/lib/libjailbreak.dylib", // electra
134             "/jb/lzma", // electra
135             "/.cydia_no_stash", // unc0ver
136             "/.installed_unc0ver", // unc0ver
137             "/jb/offsets.plist", // unc0ver
138             "/usr/share/jailbreak/injectme.plist", // unc0ver
139             "/etc/apt/undecimus/undecimus.list", // unc0ver
140             "/var/lib/dpkg/info/mobilesubstrate.md5sums", // unc0ver
141             "/Library/MobileSubstrate/MobileSubstrate.dylib",
142             "/jb/jailbreakd.plist", // unc0ver
143             "/jb/amfid_payload.dylib", // unc0ver
144             "/jb/libjailbreak.dylib", // unc0ver
145             "/usr/libexec/cydia/firmware.sh",
146             "/var/lib/cydia",
```

Hold on…

Do we need to do all that?

```
25      static func amIJailbroken() -> Bool {
26          return !performChecks().passed
27      }
28
29      static func amIJailbrokenWithFailMessage() -> (jailbroken: Bool, failMessage: String) {
30          let status = performChecks()
31          return (!status.passed, status.failMessage)
32      }
33
34      static func amIJailbrokenWithFailedChecks() -> (jailbroken: Bool, failedChecks: [FailedCheckType]) {
35          let status = performChecks()
36          return (!status.passed, status.failedChecks)
37      }
```

NOT QUITE

Reverse Engineering Techniques

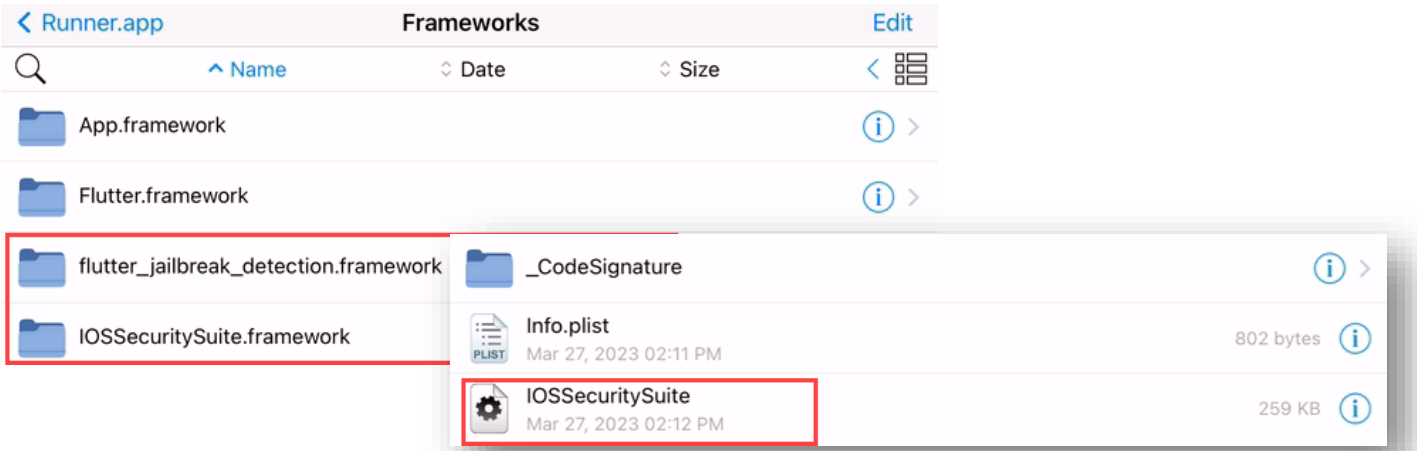
Entering assembly..

```
                                ; endp

        ; =============== B E G I N N I N G   O F   P R O C E D U R E ===============

        ; Variables:
        ;    saved_fp: 0
        ;    var_10: -16

                                                                                                  2
                                1
                    _$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ:      // static IOSSecuritySuite.IOSSecuritySuite.amIJailbroken() -> Swift.Bool
000000000000c704          stp         x20, x19, [sp, #-0x20]!
000000000000c708          stp         x29, x30, [sp, #0x10]
000000000000c70c          add         x29, sp, #0x10
000000000000c710          movz        x0, #0x0
000000000000c714          bl          _$s16IOSSecuritySuite16JailbreakCheckerCMa  ; type metadata accessor for IOSSecuritySuite.JailbreakChecker
000000000000c718          mov         x20, x0          3
000000000000c71c          bl          _$s16IOSSecuritySuite16JailbreakCheckerC13performChecks33_F8E503CD913F87B6FC3E966D69D813ABLLAC0C6StatusVyFZ ; static
000000000000c720          mov         x19, x0
000000000000c724          mov         x0, x2
000000000000c728          mov         x20, x3
000000000000c72c          bl          imp___stubs__swift_bridgeObjectRelease      ; swift_bridgeObjectRelease
000000000000c730          mov         x0, x20
000000000000c734          bl          imp___stubs__swift_bridgeObjectRelease      ; swift_bridgeObjectRelease
000000000000c738          mvn         w8, w19   4
000000000000c73c          and         w0, w8, #0x1
000000000000c740          ldp         x29, x30, [sp, #0x10]
000000000000c744          ldp         x20, x19, [sp], #0x20
000000000000c748          ret
                        ; endp
```

Reverse Engineering Techniques

```
                    _$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ:        // static IOSSecuritySuite.IOSSecuritySuite.amIJailbroken() -> Swift.Bool
000000000000c704        stp         x20, x19, [sp, #-0x20]!
000000000000c708        stp         x29, x30, [sp, #0x10]
000000000000c70c        add         x29, sp, #0x10
000000000000c710        movz        x0, #0x0
000000000000c714        bl          _$s16IOSSecuritySuite16JailbreakCheckerCMa  ; type metadata accessor for IOSSecuritySuite.JailbreakChecker
000000000000c718        mov         x20, x0
000000000000c71c        bl          _$s16IOSSecuritySuite16JailbreakCheckerC13performChecks33_F8E503CD913F87B6FC3E966D69D813ABLLAC0C6StatusVyFZ ; s
000000000000c720        mov         x19, x0
000000000000c724        mov         x0, x2
000000000000c728        mov         x20, x3
000000000000c72c        bl          imp___stubs__swift_bridgeObjectRelease      ; swift_bridgeObjectRelease
000000000000c730        mov         x0, x20
000000000000c734        bl          imp___stubs__swift_bridgeObjectRelease      ; swift_bridgeObjectRelease
000000000000c738        mvn         w8, w19
000000000000c73c        nop
000000000000c740        ldp         x29, x30, [sp, #0x10]
000000000000c744        ldp         x20, x19, [sp], #0x20
000000000000c748        ret
```

Modifying assembly instructions directly



JB check bypassed!

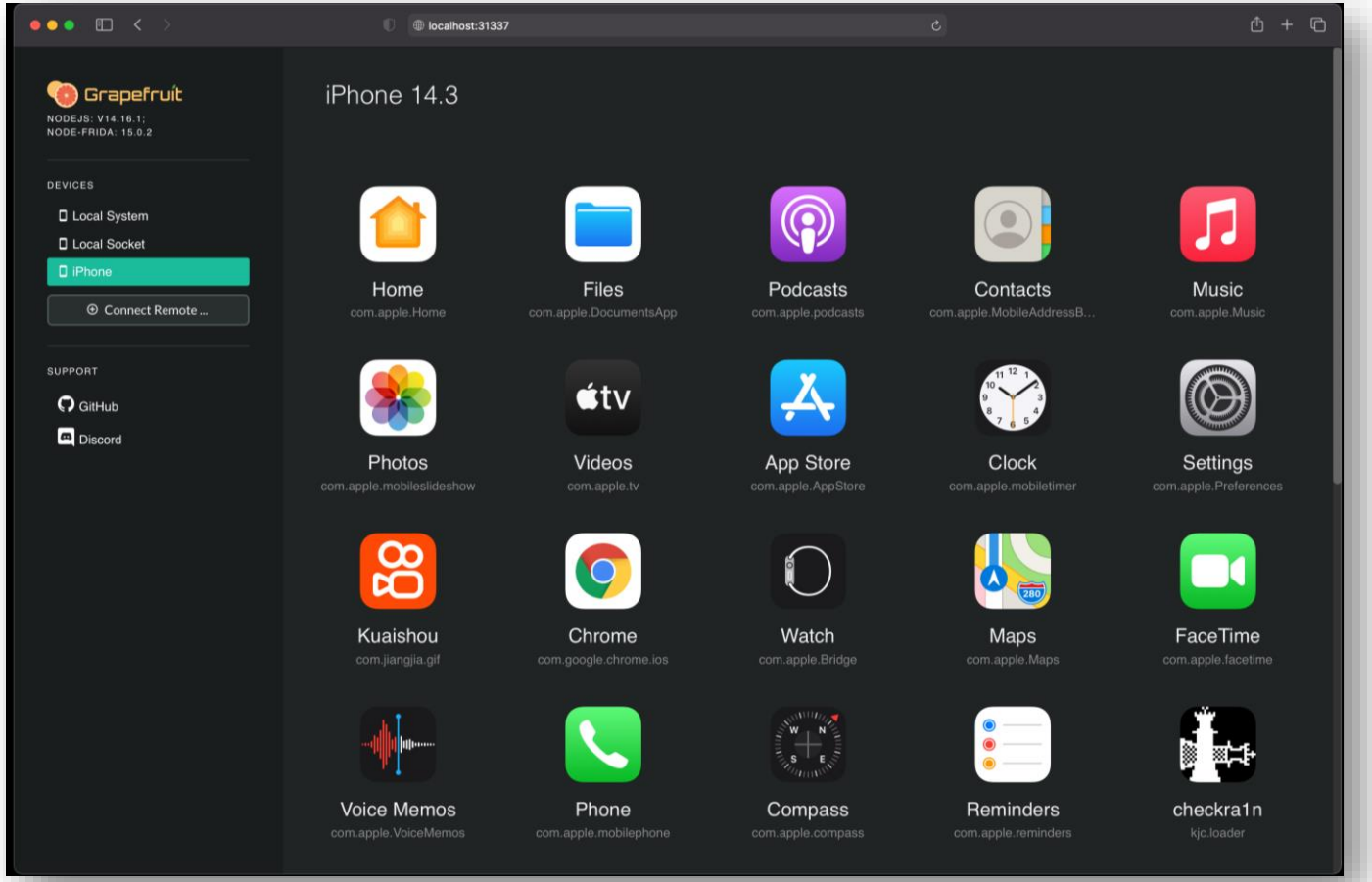
What if…
there is more reliable way

Dynamic Instrumentation

FRIDA
DYNAMIC INSTRUMENTATION TOOLKIT

OBJECTION
RUNTIME
MOBILE
EXPLORATION
GIT.IO/OBJECTION

Bypassing Anti-Tampering Checks..



Bypassing Anti-Tampering Checks..

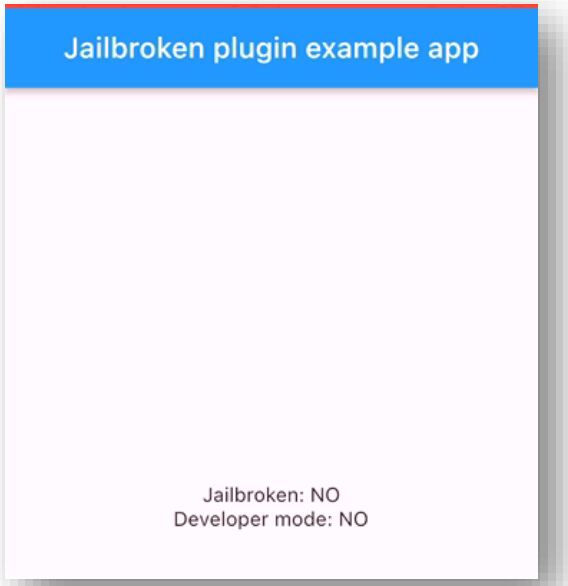
```
f 0x100ea8be8 static IOSSecuritySuite.IOSSecuritySuite.amITampered([IOSSecuritySuite.FileIntegrityCheck]) -> (result: Swift....
f 0x100ea8be4 static IOSSecuritySuite.IOSSecuritySuite.denyDebugger() -> ()
f 0x100ea8a2c static IOSSecuritySuite.IOSSecuritySuite.amIJailbroken() -> Swift.Bool
f 0x100eab024 static IOSSecuritySuite.IOSSecuritySuite.hasWatchpoint() -> Swift.Bool
f 0x100eaa160 static IOSSecuritySuite.IOSSecuritySuite.denySymbolHook(_: Swift.String, at: Swift.UnsafePointer<__C.mach_head...
f 0x100ea9504 static IOSSecuritySuite.IOSSecuritySuite.denySymbolHook(Swift.String) -> ()
f 0x100eab01c static IOSSecuritySuite.IOSSecuritySuite.hasBreakpointAt(_: Swift.UnsafeRawPointer, functionSize: Swift.UInt?)...
```

```
Interceptor.attach(Module.findExportByName("IOSSecuritySuite", "$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ"), {
  onEnter(args) {
    // todo: add code here
    console.log("$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ has been called");
    // console.log('$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ called from:\n' +
    //     Thread.backtrace(this.context, Backtracer.ACCURATE)
    //     .map(DebugSymbol.fromAddress).join('\n') + '\n');
  },
  onLeave(retval) {},
});
```

Template to use with Frida

Bypassing Anti-Tampering Checks.. with Frida!

```
Spawned `test.lula.test`. Resuming main thread!
[iOS Device::test.lula.test ]-> $s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ has been called with arguments:
arg0: 0x100962a38 (context)
$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ called from:
0x10098e518 flutter_jailbreak_detection!specialized SwiftFlutterJailbreakDetectionPlugin.handle(_:result:)
0x10098e210 flutter_jailbreak_detection!@objc SwiftFlutterJailbreakDetectionPlugin.handle(_:result:)
0x1012722bc Flutter!0x58e2bc (0x58e2bc)
0x100d27840 Flutter!0x43840 (0x43840)
0x19612a2b0 libdispatch.dylib!_dispatch_call_block_and_release
0x19612b298 libdispatch.dylib!_dispatch_client_callout
0x19610d430 libdispatch.dylib!_dispatch_main_queue_callback_4CF$VARIANT$armv81
0x1964722e0 CoreFoundation!__CFRUNLOOP_IS_SERVICING_THE_MAIN_DISPATCH_QUEUE__
0x19646c740 CoreFoundation!__CFRunLoopRun
0x19646b818 CoreFoundation!CFRunLoopRunSpecific
0x1acb71570 GraphicsServices!GSEventRunModal
0x198d970e8 UIKitCore!-[UIApplication _run]
0x198d9c664 UIKitCore!UIApplicationMain
0x1008f7224 Runner!0x7224 (0x100007224)
0x19614a140 libdyld.dylib!start

$s16IOSSecuritySuiteAAC13amIJailbrokenSbyFZ returned: 0x1
Setting JB check results to False
```



JB check bypassed!

Demo

Add Friction, but how?

- Code obfuscation
- Runtime application self-protection (RASP)
- Logging user's activity on a JB device
- Restricting functionalities inside the app
- Anti-debugging techniques to deter RE attempts

Code Obfuscation Examples

There are multiple ways to skin the cat..

Original Source Code Before Rename Obfuscation	Reverse-Engineered Source Code After Rename Obfuscation

private void
CalculatePayroll (SpecialList employee-
Group) {
while (employeeGroup.HasMore()) {
employee=
employeeGroup.GetNext(true);
employee.UpdateSalary();
Distribute Check(employee);
}
}
	private void a(a b) {
	while (b.a()) {
	a=b.a(true);
	a.a () ;
	a.(a) ;
	}
	}

Original Source Code Before Control Obfuscation	Reverse-Engineered Source Code After Control Flow Obfuscation

public int CompareTo (Object o) {
int n = occurrences -
((WordOccurrence)o) .occurrences;
if (n=0) {
n=String.Compare
(word, ((WordOccurrence)0) .word) ;
}
return (n) ;
}
	private virtual int_a(Object A+0) {
	int local0;
	int local1;
	local 10 =this.a - (c) A_0.a;
	if (local10 !=0) goto i0;
	while (true) {
	return local1:
	}
	i1:local10=
	System.String.Compare(this.b, (c)
	A_0.b) ;
	goto i0;
	}

Recap and Key Takeaways

Detection libraries provide some assurance however.. . .

Importance of code obfuscation, RASP, monitoring etc. to make RE and tampering more difficult.

Can discourage attackers by increasing the time and effort required.

Resources

CyberCX Blog

- https://cybercx.com.au/flutter-restrictions-bypass/

Tooling

- https://github.com/CyberCX-STA/flutter-jailbreak-root-detection-bypass

Thank You to Our Sponsors and Hosts!

CyberCX

OWASP NEW ZEALAND DAY 2023