2021

iOS 테스트 앱 보고서

개 정 이 력

버전	작성일	변경사유[1]	변경내용[2]	작성자	승인자
1.0	2021.11.20	최초작성		정형수	

[1] 변경 사유: 변경 내용이 이전 문서에 대해 최초작성/승인/추가/수정/삭제 중 선택 기입

[2] 변경 내용: 변경이 발생되는 위치와 변경 내용을 자세히 기록(장.절과 변경 내용을 기술한다.)

목 차

1. 개요

1.1 본 문서의 목적

테스트 APP(iOS)에 대한 취약점 진단 점검 결과 보고서로 "모바일점검가이드 iOS"의 항목을 기준으로 테스트 APP 취약점 점검 결과 보고서이다.

1.2 점검의 범위

본 문서에서 정의하는 점검의 수행 범위는 iOS 모바일 단말기에 설치되는 앱으로 한정한다.

2. 점검항목

분류	평가항목	금융위	금감원	금보원	비고
입력 값 검증 및 예외처리	[MA-1] 입력 값 검증	SA-1			
	[MA-2] 예외 처리	SA-2			
사용자 인증	[MB-1] 적절한 인증절차 및 세션관리	SB-1			
	[MB-2] 패스워드 복잡도 검증	SB-2			
	[MB-3] 사용자 식별 정보 관리	SB-3			
정보의 기밀성 및 무결성	[MC-1] 단말기 중요정보 저장 금지	SC-1	FB-5	CC-5	
	[MC-2] 역분석 방지	SC-2	FA-4	CB-3	
	[MC-3] 소스코드 내 중요정보 노출 여부		FB-10		
	[MC-4] 중요정보 화면 미표시	SC-3			
	[MC-5] 메모리 보호	SC-4			
	[MC-6] 입력정보 보호	SC-5	FB-1	CA-2	
	[MC-7] 송·수신 정보 보호	SC-6	FB-2	CA-3	
	[MC-8] 거래전문 무결성 검증기법 적용		FB-3	CA-4	
악성코드 및 프로그램 위변조 대응	[MD-1] 악성코드 방지 (백신프로그램)	SD-1	FB-6	CA-1	
	[MD-2] OS 변조 탐지	SD-2	FA-1	CB-1	
	[MD-3] 프로그램 무결성 검증	SD-3	FA-2	CB-2	
	[MD-4] 적용된 암호기술(알고리즘/키공유 방식)의 적정성 여부		FA-3		
기타	[ME-1] 디버깅 정보 노출 방지	SE-1			
	[ME-2] 멀티로그인(Multi-Login) 차단		FB-9	CC-4	
	[ME-3] 앱 취약점 점검		FB-7	CC-1	인터뷰
	[ME-4] 위·변조 앱 모니터링			CC-2	
	[ME-5] 앱 위·변조 로그기록		FA-2	CC-3	
	[ME-6] 스마트폰 금융거래기록 정보 보관		FB-4	CC-6	
	[ME-7] 스마트폰 정보 보관시 고객 사전 동의 여부				
	[ME-8] 소스코드 개발시 시큐어 코딩 기술 적용 여부		FB-8		

3. 테스트 APP 진단

3.1 [MC-3]소스코드 내 중요정보 노출 여부

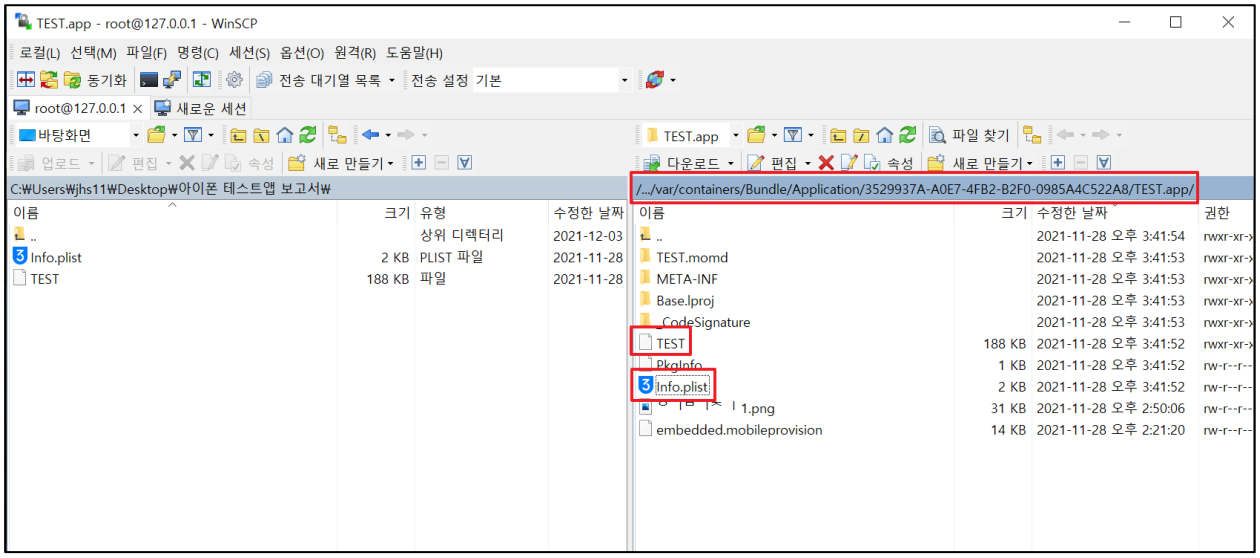
[취약점 설명]

소스 또는 바이너리 내 중요한 정보를 하드 코딩하여 저장할 경우, 정보 노출이 매우 쉬우므로 저장하지 않는 것이 바람직하다. 이로 인해 중요 정보가 노출되어 2 차 피해가 발생할 수 있다.

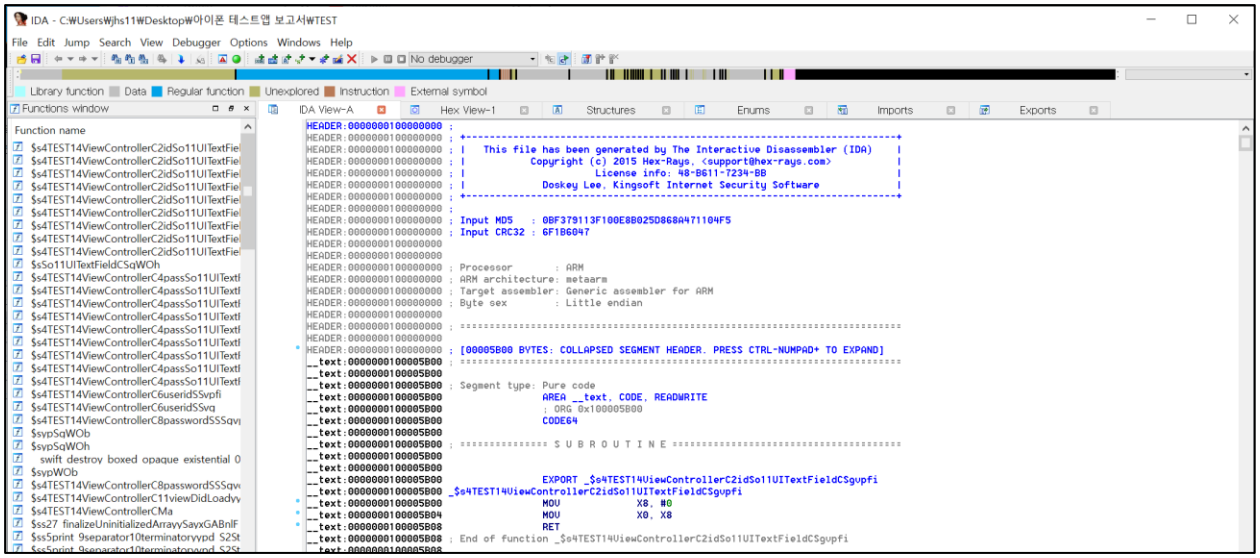
[점검절차]

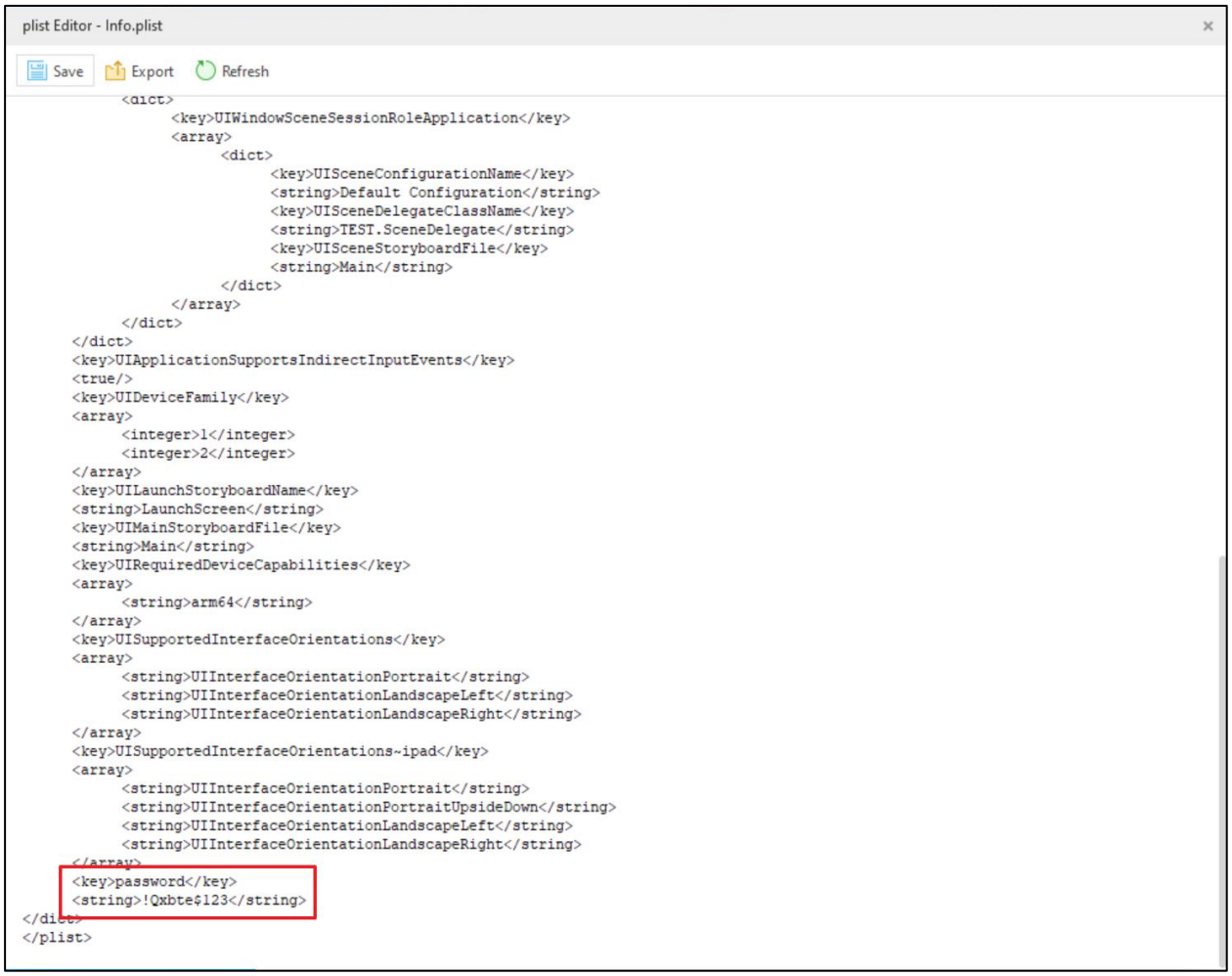
Step 1. WinSCP 를 사용하여 테스트 APP 파일을 추출한다.

[위치 : /private/var/containers/Bundle/Application/3529937A-A0E7-4FB2-B2F0-0985A4C522A8/TEST.APP]

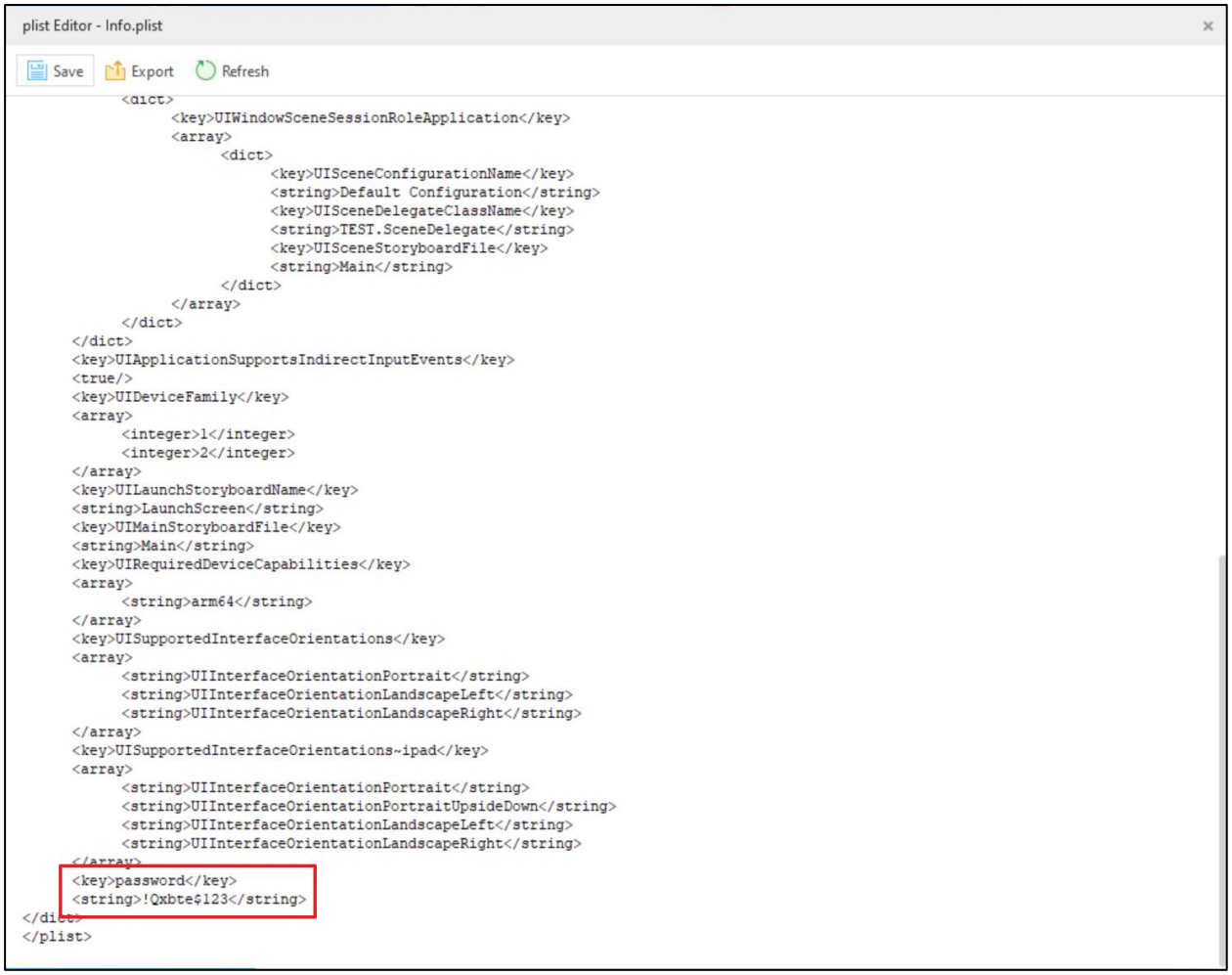


Step 2. 추출한 TEST 파일을 IDA, info.plist 파일을 3utools 를 사용하여 접근한다.




plist Editor - Info.plist
Save
Export
Refresh
<dict>
<key>UIWindowSceneSessionRoleApplication</key>
<array>
<dict>
<key>UISceneConfigurationName</key>
<string>Default Configuration</string>
<key>UISceneDelegateClassName</key>
<string>TEST.SceneDelegate</string>
<key>UISceneStoryboardFile</key>
<string>Main</string>
</dict>
</array>
</dict>
</dict>
<key>UIApplicationSupportsIndirectInputEvents</key>
<true/>
<key>UIDeviceFamily</key>
<array>
<integer>1</integer>
<integer>2</integer>
</array>
<key>UILaunchStoryboardName</key>
<string>LaunchScreen</string>
<key>UIMainStoryboardFile</key>
<string>Main</string>
<key>UIRequiredDeviceCapabilities</key>
<array>
<string>arm64</string>
</array>
<key>UISupportedInterfaceOrientations</key>
<array>
<string>UIInterfaceOrientationPortrait</string>
<string>UIInterfaceOrientationLandscapeLeft</string>
<string>UIInterfaceOrientationLandscapeRight</string>
</array>
<key>UISupportedInterfaceOrientations~ipad</key>
<array>
<string>UIInterfaceOrientationPortrait</string>
<string>UIInterfaceOrientationPortraitUpsideDown</string>
<string>UIInterfaceOrientationLandscapeLeft</string>
<string>UIInterfaceOrientationLandscapeRight</string>
</array>
<key>password</key>
<string>!Qxbte$123</string>
</dict>
</plist>

Step 3. 관리자 password 가 평문으로 저장된 것을 확인 가능하다.



3.2 [MC-5] 메모리 보호

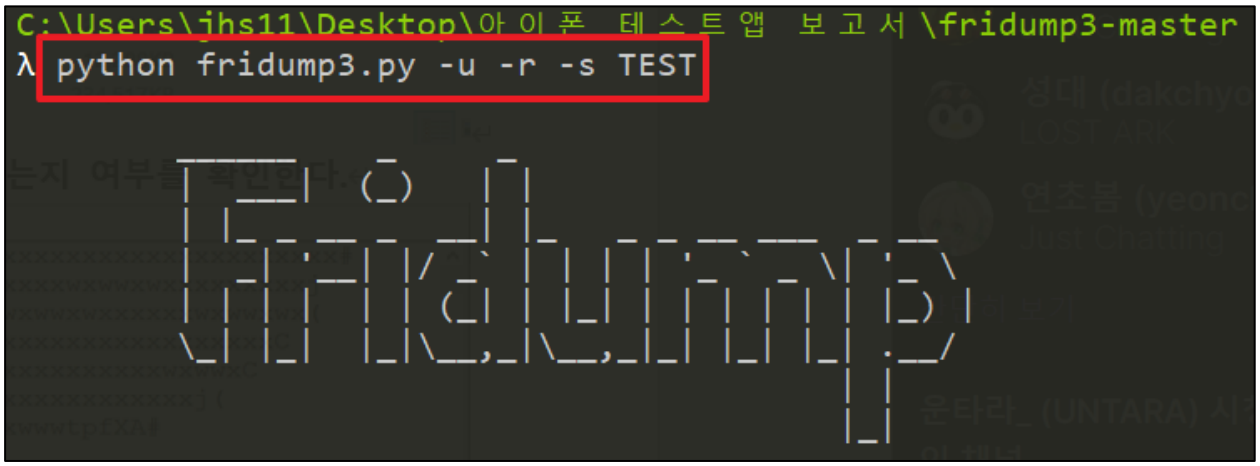
[취약점 설명]

중요한 키 또는 개인정보 등을 암호화해서 저장하여 사용하더라도 메모리 상에서는 이를 해제하여 사용하는 경우가 매우 많다. 이러한 경우, 메모리 덤프로 관련 정보가 유출될 수 있으므로 반드시 최소한의 시간 동안만 암호를 해제하여 유지해야 한다. 취약한 경우 중요 정보 노출로 인해 2 차 피해가 발생할 수 있다.

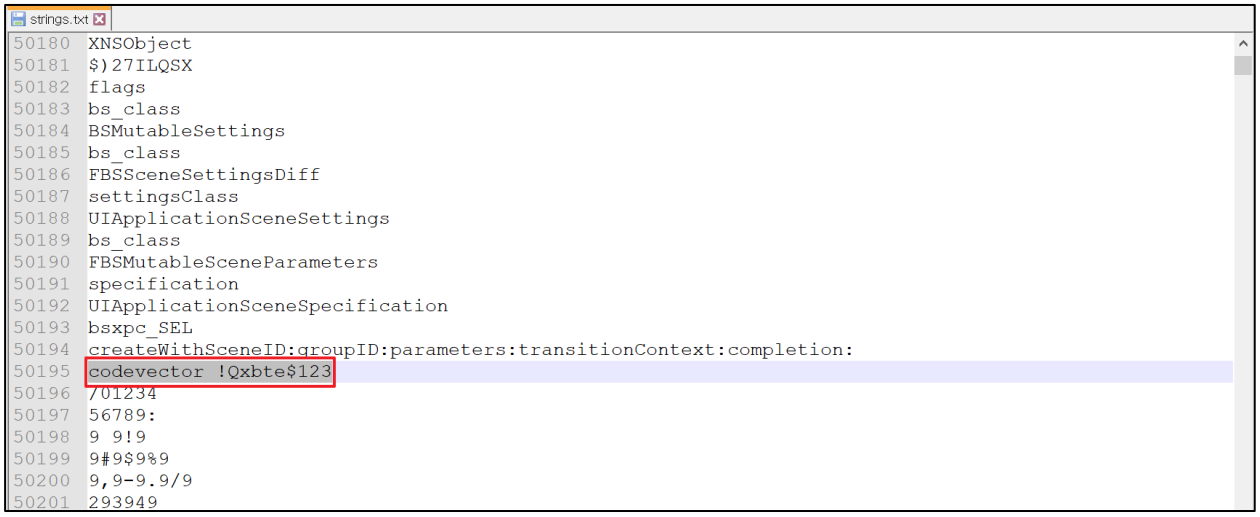
[점검절차]

Step 1. 앱 실행 후 Fridump 명령을 실행하여 타켓 앱의 메모리영역의 덤프를 확보한다.

[명령어 : python fridump3.py -u -r -s 앱이름]



Step 2. strings.txt 파일 상에 개인정보 및 계정정보 등의 중요 정보가 평문으로 메모리상에 저장되어 있는 것을 확인 가능하다.



※ [pyhton fridump3.py -h] 명령어를 사용하여 옵션확인이 가능하다.

- ✓ -h : 도움말 표시.
- ✓ -o dir : 결과물 저장할 디렉토리 지정.
- ✓ -u : USB 를 통해 연결.
- ✓ -H : IP 를 통해 연결.
- ✓ -v : 자세한 정보 출력.
- ✓ -r : 메모리의 읽기 전용 부분 덤프.
- ✓ -s 옵션은 strings.txt 파일을 추가로 생성하여 문자열만 모아서 파일에 저장.

※ [frida-ps -Ua] 명령어를 통해 실행중인 APP 의 PID, NAME, Identifier 확인이 가능하다.

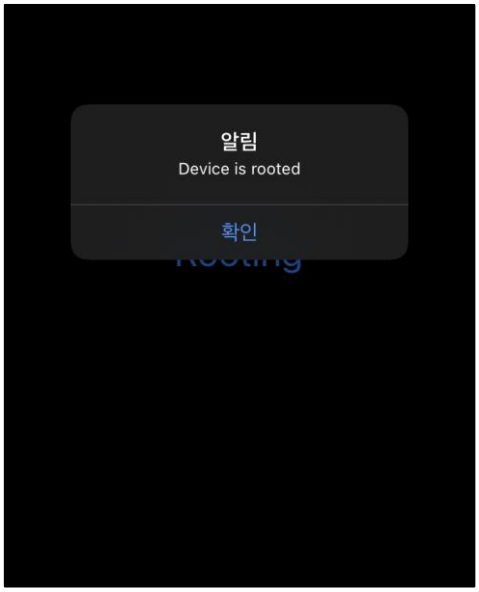
3.3 루팅 우회

[취약점 설명]

모바일 취약점 진단 수행 시 대상 APP 에 루팅 탐지 로직이 존재할 경우 우회해야 진단이 가능하기 때문에 frida 를 사용하여 루팅 우회가 필요하다.

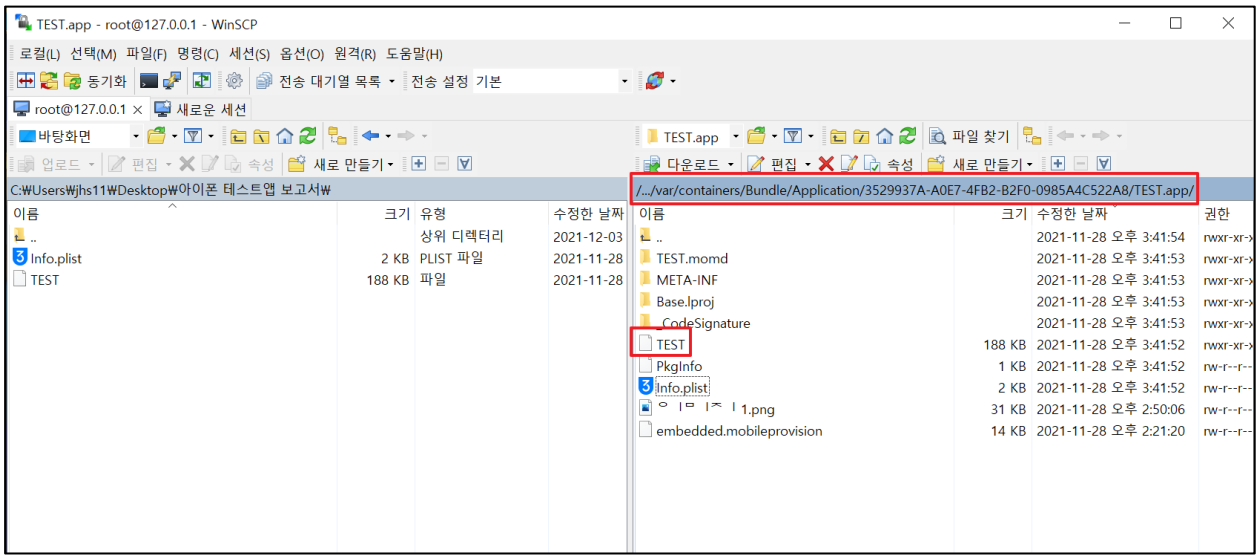
[점검절차]

Step 1. Rooting 버튼을 클릭 시 루팅된 단말기의 경우 'Device is rooted' 메시지가 출력된다.

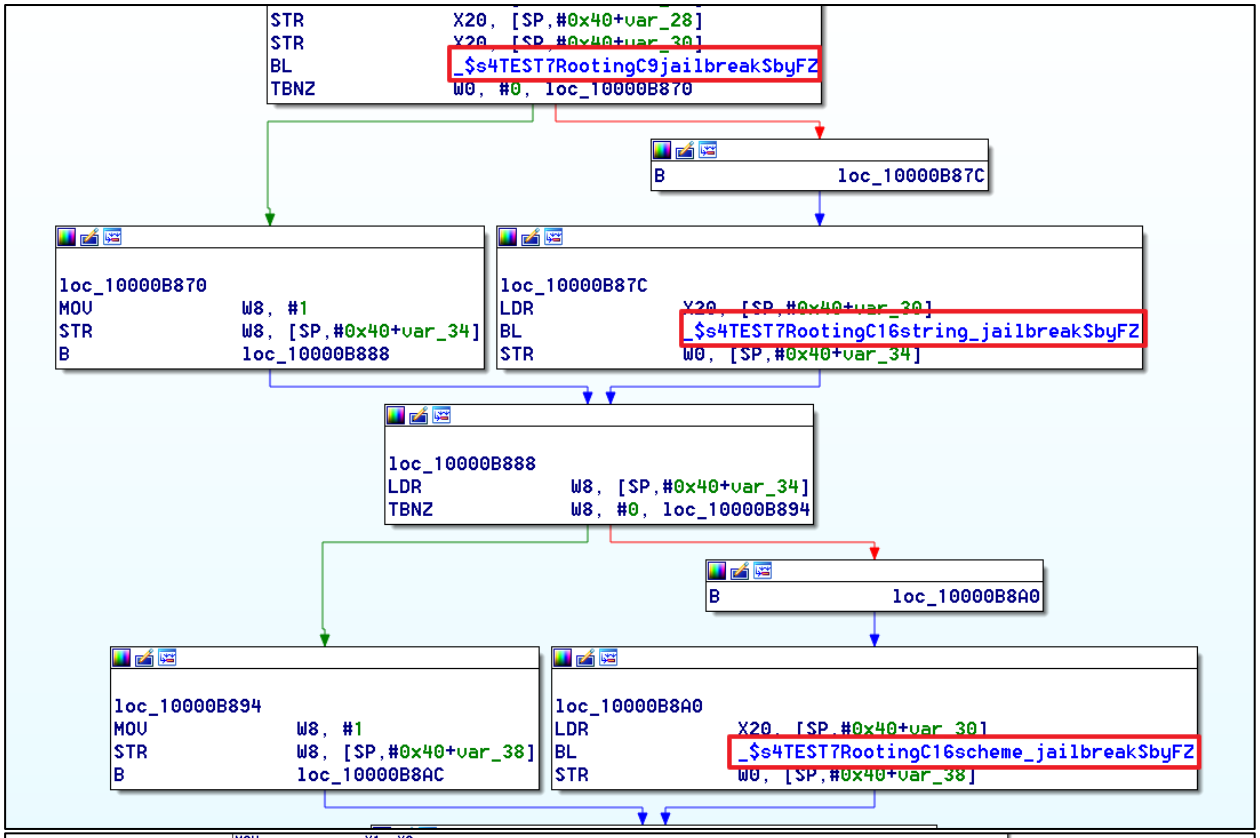


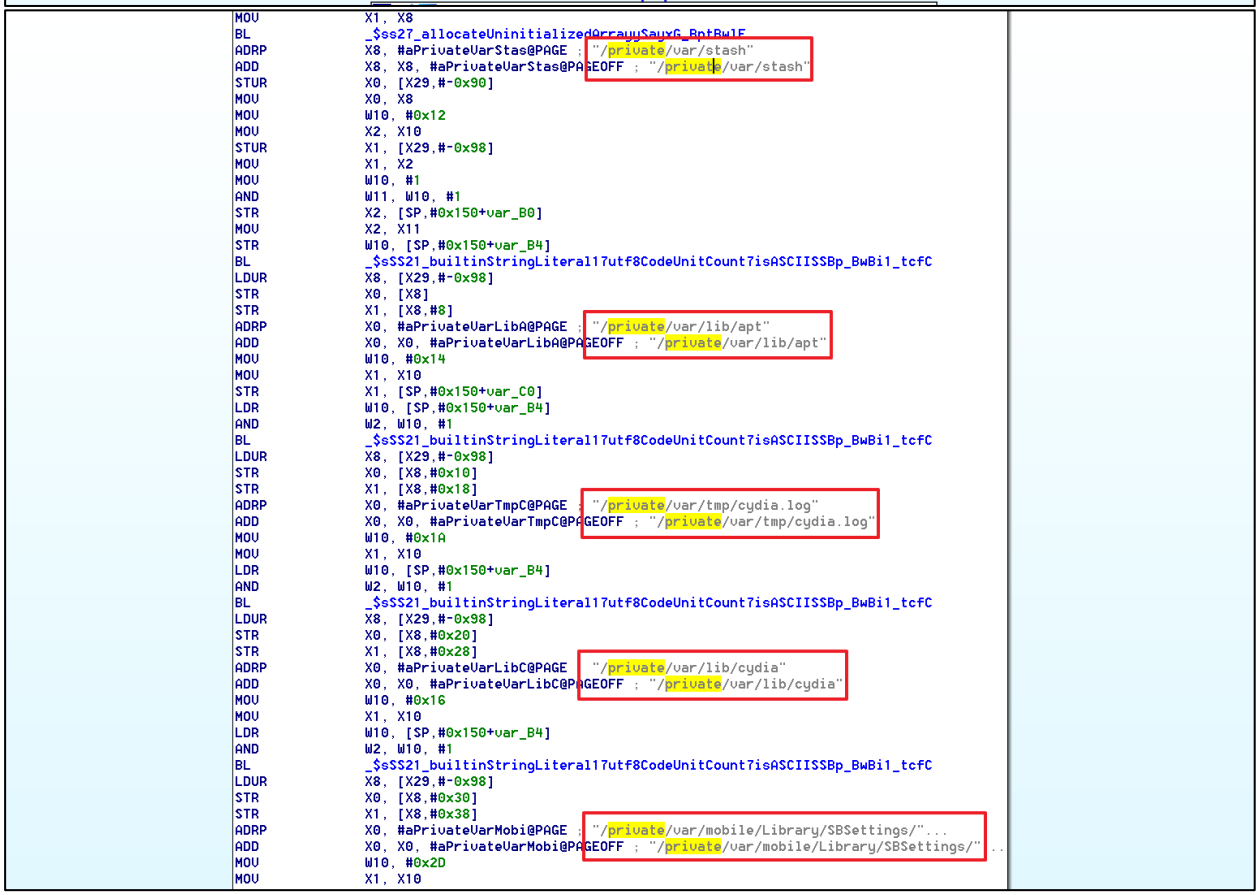
Step 2. WinSCP 를 사용하여 TEST 파일을 추출한다.

[위치 : /private/var/containers/Bundle/Application/3529937A-A0E7-4FB2-B2F0-0985A4C522A8/TEST.APP]



Step 3. 추출한 TEST 파일을 IDA 로 열어 루팅 탐지 로직을 확인한다.





```
STUR            X10, [X29,#-0x28]
ADRP            X0, #aJailbroken@PAGE ; "jailbroken"
ADD             X0, X0, #aJailbroken@PAGEOFF ; "jailbroken"
MOV             W11, #0xA
MOV             X2, X11
MOV             X1, X2
MOV             W11, #1
AND             W2, W11, #1
STUR            X8, [X29,#-0x90]
STUR            W11, [X29,#-0x94]
BL              _$sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC
SUB             X8, X29, #0x38
STUR            X0, [X29,#-0x38]
STR             X1, [X8,#8]
ADRP            X0, #aVarRootJailbre@PAGE ; "/var/root/jailbreak.txt"
ADD             X0, X0, #aVarRootJailbre@PAGEOFF ; "/var/root/jailbreak.txt"
MOV             W11, #0x17
MOV             X1, X11
LDUR            W11, [X29,#-0x94]
AND             W2, W11, #1
STUR            X8, [X29,#-0xA0]
BL              _$sSS21_builtinStringLiteral17utf8CodeUnitCount7isASCIISSBp_BwBi1_tcfC
SUB             X8, X29, #0x48
STUR            X0, [X29,#-0x48]
STR             X1, [X8,#8]
```

```
ADRP            X0, #aFatalError@PAGE ; "Fatal error"
ADD             X0, X0, #aFatalError@PAGEOFF ; "Fatal error"
ADRP            X3, #aUnexpectedlyFo@PAGE ; "Unexpectedly found nil while unwrapping"...
ADD             X3, X3, #aUnexpectedlyFo@PAGEOFF ; "Unexpectedly found nil while unwrapping"...
ADRP            X6, #aTestRootingche@PAGE ; "TEST/RootingCheck.swift"
ADD             X6, X6, #aTestRootingche@PAGEOFF ; "TEST/RootingCheck.swift"
STUR            XZR, [X29,#-0x18]
ADRP            X8, #_$s10Foundation3URLVSgMD@PAGE
ADD             X8, X8, #_$s10Foundation3URLVSgMD@PAGEOFF
STUR            X0, [X29,#-0x20]
MOV             X0, X8
STUR            X20, [X29,#-0x28]
STUR            X3, [X29,#-0x30]
STUR            X6, [X29,#-0x38]
BL              ___swift_instantiateConcreteTypeFromMangledName
LDUR            X8, [X0,#-8]
LDR             X8, [X8,#0x40]
ADD             X8, X8, #0xF
AND             X8, X8, #0xFFFFFFFFFFFFFFF0
MOV             X9, X8
STUR            X8, [X29,#-0x40]
ADRP            X16, #___chkstk_darwin_ptr@PAGE
LDR             X16, [X16,#___chkstk_darwin_ptr@PAGEOFF]
BLR             X16
MOV             X8, SP
LDUR            X9, [X29,#-0x40]
SUBS            X8, X8, X9
MOV             SP, X8
LDUR            X10, [X29,#-0x28]
STUR            X10, [X29,#-0x18]
ADRP            X11, #classRef_UIApplication@PAGE
LDR             X0, [X11,#classRef_UIApplication@PAGEOFF]
STUR            X8, [X29,#-0x48]
BL              _objc_opt_self
ADRP            X8, #selRef_sharedApplication@PAGE
LDR             X1, [X8,#selRef_sharedApplication@PAGEOFF]
BL              _objc_msgSend
MOV             X29, X29
BL              _objc_retainAutoreleasedReturnValue
ADRP            X8, #aCydiaPackageCo@PAGE ; "cydia://package/com.example.package"
ADD             X8, X8, #aCydiaPackageCo@PAGEOFF ; "cydia://package/com.example.package"
STUR            X0, [X29,#-0x50]
```

Step 4. 확인한 로직을 우회하기 위해 코드를 작성한다.

```
var root1 = 'TEST';
var root2 = ptr(0x76C4); //바이너리 파일 내 접근할 함수의 주소
var module1 = Module.getBaseAddress(root1); //바이너리 파일 시작 주소
var target = module1.add(root2); //타겟 주소.


Interceptor.attach(target,{ //특정 함수가 실행될 때 접근하여 레지스터 값 변경
    onEnter: function(args){
        if(this.context.x0 == 0x1){ //this.context를 사용하여 레지스터 값에 접근 가능하며 return 값이
            this.context.x0=0x0;    //true 또는 false이기 때문에 0x0, 0x1로 수정하여 접근한다.
            send("Bypass1");        //0x1인 경우 루팅된 단말기를 의미하기 때문에 0x0으로 수정한다.
        }
    },
});


root2=ptr(0x79E8);
module1 = Module.getBaseAddress(root1);
target = module1.add(root2);

Interceptor.attach(target,{
    onEnter: function(args){
        if(this.context.x0 == 0x1){
            this.context.x0=0x0;
            send("Bypass2");
        }
    },
});

```

Step 5. frida 명령어를 사용하여 루팅 우회를 시도한다.

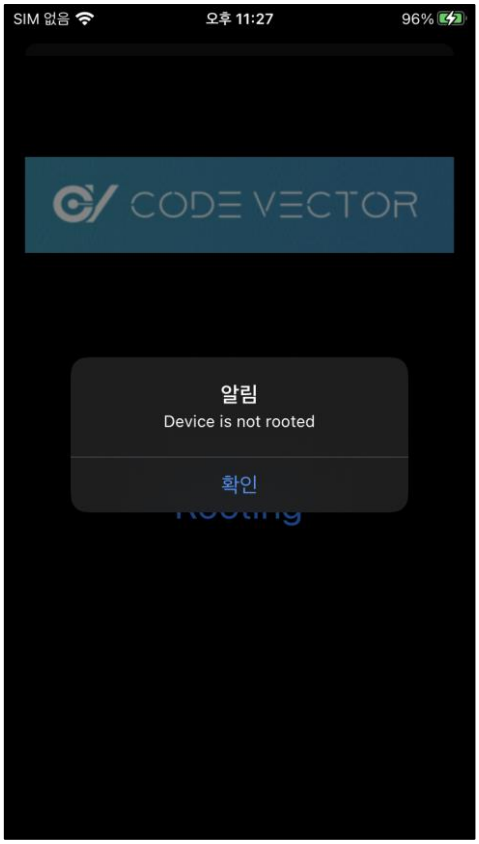
[명령어] : [frida -Uf kr.co.jeong.TEST -l rooting.js]

※ -Uf : -U -f 와 같은 의미. USB 로 연결된 단말기 App 중 kr.co.jeong.TEST 를 실행.

※ -l rooting.js : 사용할 스크립트를 로드하기 위해 -l 명령어를 사용.

```
C:\Users\jhs11\Desktop\아이폰 테스트앱 보고서
λ frida -Uf kr.co.jeong.TEST -l rooting.js
     ____
    / _  |   Frida 15.1.10 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at https://frida.re/docs/home/
Spawned `kr.co.jeong.TEST`. Use %resume to let the main thread start executing!
[Apple iPhone::kr.co.jeong.TEST]-> %resume
[Apple iPhone::kr.co.jeong.TEST]-> message: {'type': 'send', 'payload': 'Bypass1'} data: None
message: {'type': 'send', 'payload': 'Bypass2'} data: None
```

Step 6. 우회가 성공하여 'Device is not rooted' 메시지가 출력되는 것을 확인 가능하다.



Step 7. 우회가 성공하여 다음 화면으로 이동한 것을 확인 가능하다.

4. frida

4.1 frida 란

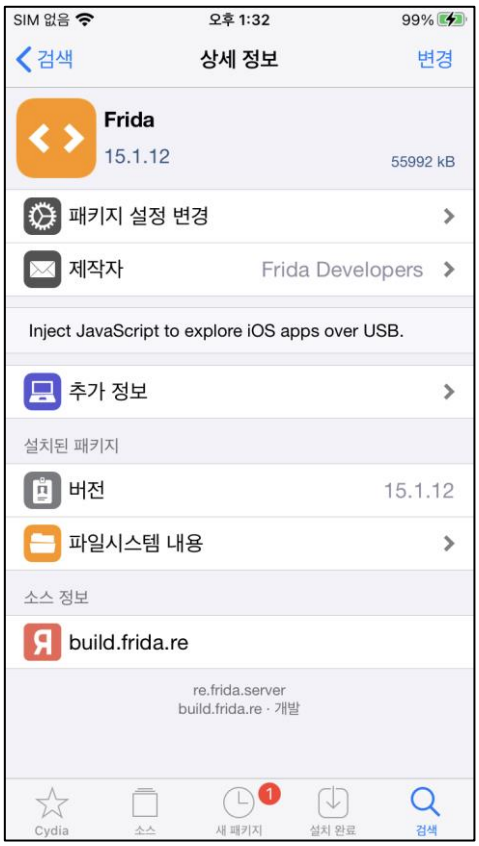
frida 는 바이너리를 동적으로 실행하면서 분석하는 DBI(Dynamic Binary Instrumentation) 도구이다. 이를 통해 Windows, MacOS, GNU/Linux, iOS, Android 및 QNX 의 기본 앱에 JavaScript 또는 자체 라이브러리의 스니펫(재사용 가능한 소스 코드, 기계어, 텍스트의 작은 부분을 일컫는 용어)을 삽입할 수 있다. 또한, Frida API 를 기반으로 구축된 몇 가지 도구를 제공한다. 이것들을 그대로 사용하거나 조정하여 사용할 수 있다.

4.2 frida 설치

frida 를 사용하기 위해서는 기본적으로 python 3.x 가 필요하다.

[설치 절차]

Step 1. 루팅된 단말기 Cydia 앱을 실행하여 frida 를 설치한다.



Step 2. frida 가 정상적으로 동작하기 위해 단말기의 설치된 frida 버전을 로컬 PC 에 설치한다.

```
λ pip install frida==15.1.12
```

Step 3. [frida --version]명령어를 실행하여 설치된 frida 버전을 확인한다.

```
C:\Users\jhs11\Desktop\아이폰 테스트앱 보고서\fridump3-master
λ frida --version
15.1.12
```

Step 4. frida 버전에서 사용하는 frida-tools 버전을 확인하여 설치한다.

```
C:\Users\jhs11
λ pip install frida-tools==10.4.1
```

python3-frida-tools-10.4.1-1.fc34.noarch.rpm

python3-frida-tools_10.4.1-1.ubuntu-focal_all.deb

python3-frida-tools_10.4.1-1.ubuntu-hirsute_all.deb

Step 5. 정상적으로 설치된 것을 확인하기 위해 APP 을 실행한 뒤 [frida-ps -Ua] 명령어를 실행 하여 인식이 되는지 확인한다.

```
C:\Users\jhs11\Desktop\아이폰 테스트앱 보고서\fridump3-master
λ frida-ps -Ua
PID  Name   Identifier
---  -----  ----------------
678  Cydia  com.saurik.Cydia
593  TEST   kr.co.jeong.TEST
```