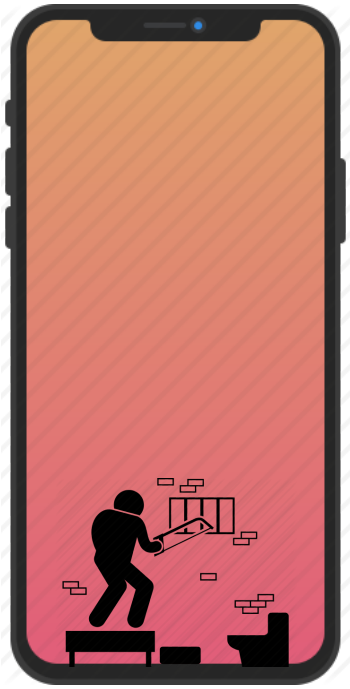
#13

Frida for iOS

FRIDA

Dynamic instrument toolkit for developers, reverse-engineers, and security researchers



2018년 12월 iOS 버전이 12.1.2까지 배포되었다. 반면 iOS Jailbreak(탈옥) 툴은 iOS 11.4 Beta3 까지 지원된다. (iOS 11.4 Beta3 다운그레이드 불가) 현재 iOS 11.4.1 버전에 대한 Exploit을 Ian Beer가 가지고 있는 것으로 알려져 있고 Electra를 통해 배포될 것으로 예상하고 있지만 언제 배포될지 확실하지 않다.

새로운 버전의 iOS가 배포되면 이전 버전 iOS를 위한 Jailbreak 툴이 배포되었던 흐름에 맞추어 볼 때, 최근 iOS Jailbreak가 쉽지 않아졌음을 짐작할 수 있다.

이 문서에서 설명하는 Frida는 Jailbreak 여부와 상관없이 iOS 애플리케이션을 분석하는데 사용할 수 있지만, 제한없는 사용을 위해 탈옥된 아이폰을 대상으로 한다. 따라서 iOS 11.4 Beta3 이하 버전의 탈옥된 아이폰을 사용하는 것이 좋다. 탈온된 아이폰에서 Frida를 이용한 애플리케이션(이하 앱) 분석 및 공격 방법을 알아보자.

Jailbreak iPhone

iOS 버전에 따라 다양한 Jailbreak 툴을 사용할 수 있다. 가장 최신 버전의 iOS를 위한 Jailbreak 툴을 소개하자면, Coolstar의 Electra가 있다. 11.2 부터 11.4 Beta3 까지 지원한다. 가지고 있는 아이폰의 iOS 버전을 확인하고 그에 맞는 Jailbreak 툴을 이용하면 된다. 자신의 아이폰이 iOS 11.4.1 이상 버전이라면 다른 아이폰을 구해보도록 하자.

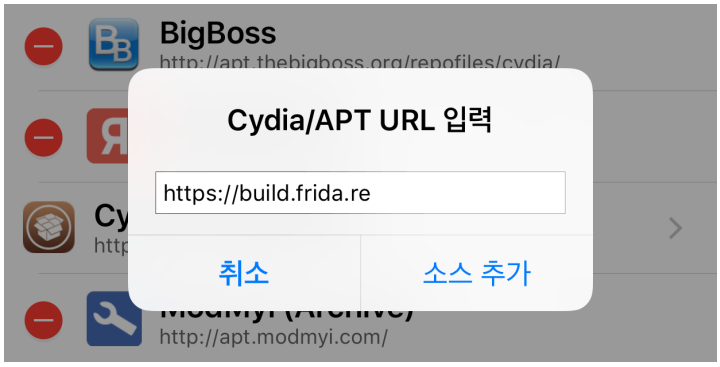
Cydia

Cydia는 탈옥된 iOS 디바이스를 위한 앱 스토어다. 애플의 앱 스토어에서 설치할 수 없는 앱 또는 기능 개선을 위한 트윅(tweak)을 설치할 수 있다. iOS 앱 해킹을 위해 기본적으로 BigBoss Recommended Tools, OpenSSH, Frida와 같은 트윅을 설치한다. Jailbreak 툴을 사용해 탈옥을 하면 보통 자동으로 설치 되지만 간혹 함께 설치 되지 않는 경우도 있어 직접 설치해야 할 때도 있다. (Cydia를 만든 Jay Freeman이 곧 Cydia를 폐쇄 한다고 한다.)

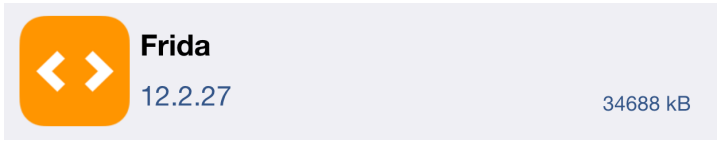
Install Frida Server on iPhone

Cydia에서 Frida를 설치해보자. frida.re에서 간단한 설치 방법을 확인할 수 있다.

Sources(소스) 탭 -> Edit(편집) -> Add(추가) -> https://build.frida.re



그리고 나면 Search(검색) 탭에서 Frida를 찾을 수 있다. 설치하면 된다.



현재 버전은 12.2.27이다. 이어서 설치할 Frida CLI Tools의 Major 버전이 일치해야 제대로 동작한다. Frida tools의 버전은 12.2.x 여야 한다.

Install Frida's CLI Tools on your Mac

Frida 서버에 스크립트를 전달하기 위해 Python을 사용한다.(Command 1) 이 때 사용할 Python용 Frida 라이브러리를 설치한다. Python 3 를 사용하여 전달하기로 한다.(Command 2)
아이폰에 설치된 Frida 서버와 통신하고 간단한 명령을 전달하기위해 사용자의 컴퓨터에 Frida 툴(Frida's CLI tools)을 설치해야 한다. 설치를 위해 Python이 설치되어 있어야 하는데, Python 2.7을 사용하는 것을 추천한다. (Command 3,4)
Commands:

1. brew install python3 (Python 3 설치(brew for Mac))
2. pip3 install frida (Python3 용 Frida 라이브러리 설치)
3. sudo easy_install pip (Python 2.7 패키지 관리자 설치 - 기본적으로 설치되어있지 않다.)
4. sudo -H pip install frida-tools (Python 2.7 버전의 Frida-tools 설치)
   : pip3로 설치하지 않는 이유는 pip3의 frida-tools는 인코딩 버그가 있어 한글이 제대로 표시되지 않는다.
   : Module six 관련 에러가 날 경우 sudo -H pip install frida-tools --ignore-installed six 로 설치

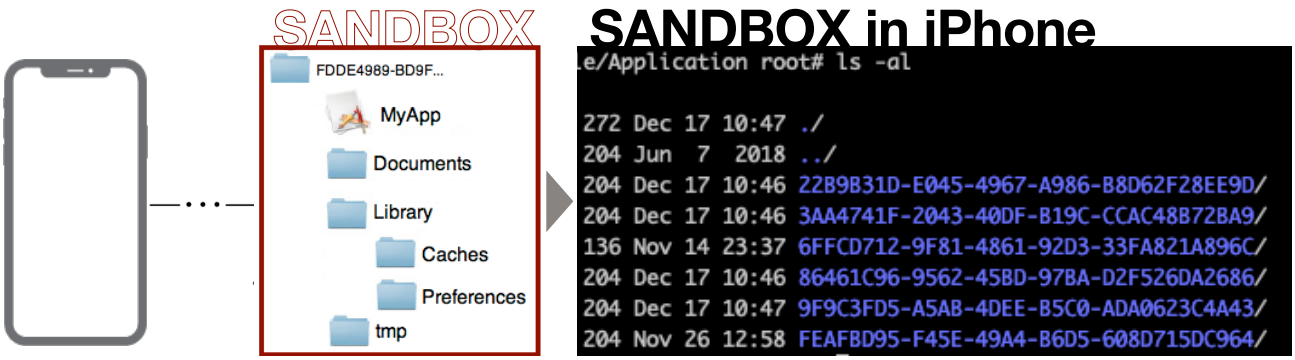
Test Frida

여기까지 Frida를 실행할 준비를 마쳤다. 아이폰을 컴퓨터에 연결하고 frida-ps -U 명령을 실행해보자.
위 명령이 정상적으로 실행되었다면 현재 아이폰에서 동작중인 프로세스의 목록을 볼 수 있을 것이다. 앞으로 자주 사용하게 될 명령 frida-ps -Uai 도 실행해보자. 프로세스의 Identifier를 출력해준다. 주로 앱의 Identifier를 이용해 앱에 접근해 공격을 실행할 것이다.

Sandbox

아이폰은 보안을 위해 Sandbox라는 개념을 가지고 있다. Sandbox는 커널 수준에서 시행되는 **접근 제어 기술**이다. 앱을 실행할 때 격리된 공간(Sandbox)을 제공하고 그곳을 벗어나 허용되지 않은 작업을 하지 못하도록 방지하는 기술이다.

운영체제별로 다양한 형태로 Sandbox를 구현하고 있는데, 아이폰의 경우 아래 그림과 같이 구현되어 있다.

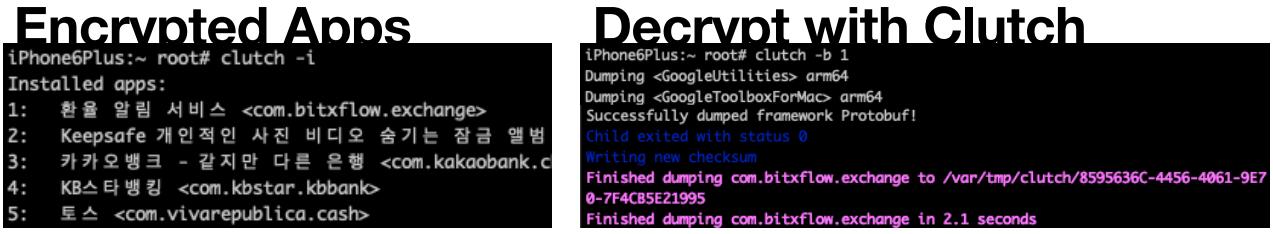


Cydia에서 OpenSSH를 설치하고 root 계정으로 접속해보면 /var/containers/Bundle/Application에서 앱 리스트를 확인할 수 있다. 위 그림에서 총 6개의 앱들을 확인할 수 있는데, 예측하기 힘든 난수로 된 문자열의 폴더명을 가지고 있다. 앱이 설치되기 전까지 디렉터리의 이름을 알 수 없기 때문에 다른 앱이나 리소스가 접근할 수 없다.

Clutch and Get a binary from Application Sandbox

Frida 스크립트를 작성하기 전에 앱 분석을 먼저 해야 한다. 앱이 어떻게 구현되어 있는지 알고 공격하고자 하는 위치를 정확히 알아야 그에 맞는 스크립트를 작성할 수 있다. 아이폰의 실행파일은 앞서 설명한 앱의 샌드박스에 들어있다. 샌드박스 안을 살펴보면 MyApp.app, Documents, Library 등의 폴더와 파일들이 존재하는데, 이 중에 MyApp.app 패키지 폴더에 실행파일이 들어있다. scp 또는 iFunbox 등을 이용해 다운로드 받으면 된다. 아래 예시에선 exchange라는 앱의 실행파일을 다운로드 받았다.

앱 스토어에 배포된 앱을 분석할 경우, 실행파일이 암호화되어 있기 때문에 다운로드 받더라도 분석할 수 없다. 이 경우 Clutch를 이용해 복호화할 수 있다.

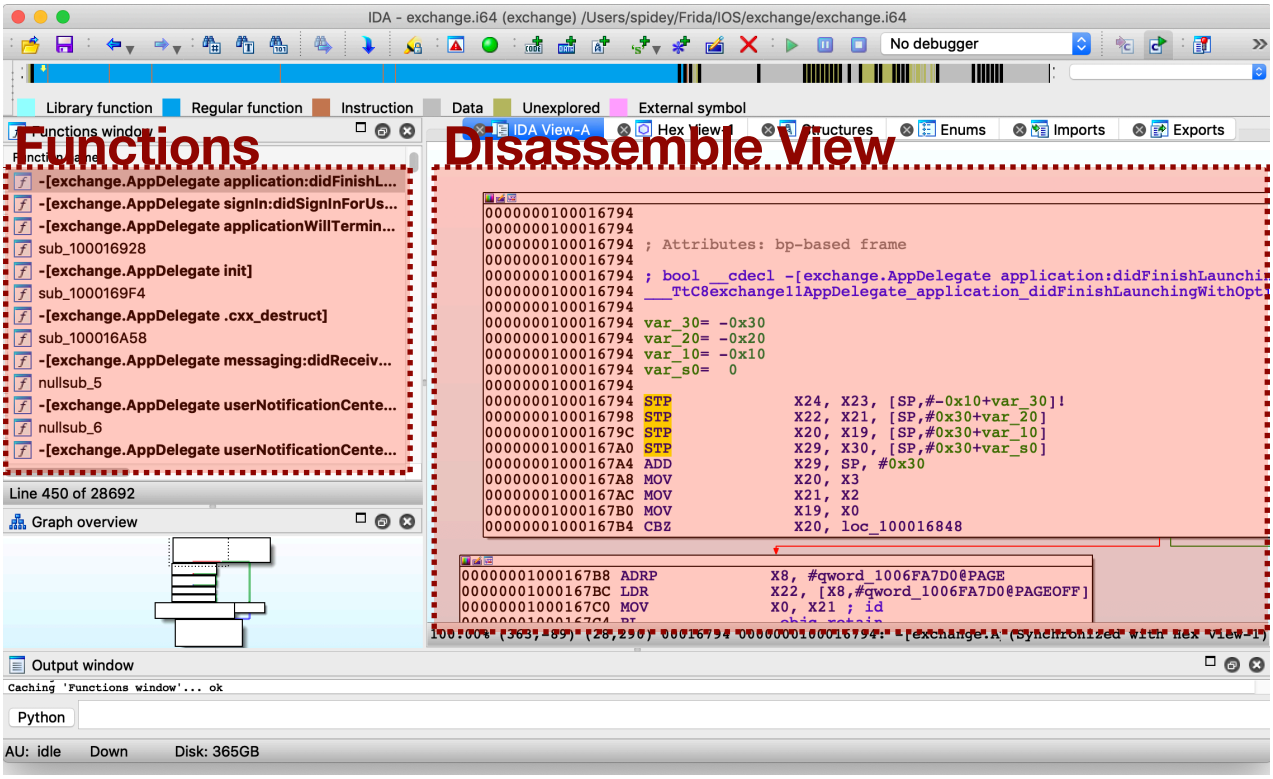


Clutch 를 이용해 앱을 복호화[1]하면 분홍색으로 복호화된 실행파일의 경로를 표시해준다. 위 예시의 경우 /var/tmp/clutch/8595636C-4456-4061-9E70-7F4CB5E21995 에 있는 실행파일을 가져오면 된다.

Open the Binary with IDA

[1] clutch를 이용해 복호화 하면 실행파일과 함께 사용된 라이브러리도 함께 복호화된다. clutch에 표시된 경로에서 실행파일을 찾아보자.

아이폰에서 다운로드한 실행파일을 IDA 를 이용해 열어보자.

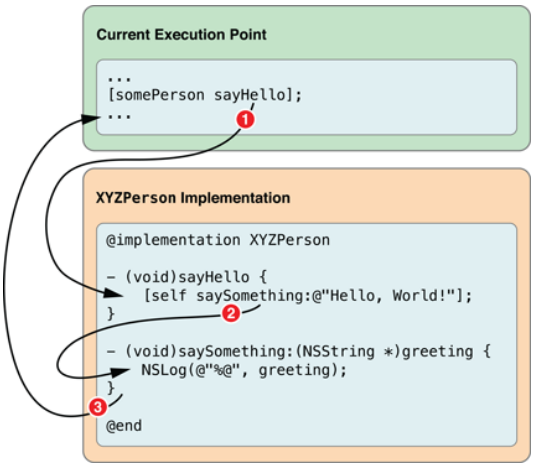


위 그림과 같이 IDA는 함수의 목록을 보여주는 **Functions** 부분과 함수의 내용을 표시해주는 **Disassemble View** 부분으로 나뉜다. Functions 에서 분석하고자 하는 대상의 함수를 먼저 찾고, Disassemble View 에서 함수의 내용을 확인한다. 함수를 분석하기 위해서는 Objective C 의 기본 문법[2]에 대한 이해를 필요하다.

Objective C - Basic #1. Functions

IDA의 Functions에서 확인할 수 있는 Objective C 함수의 특징을 살펴보자.
함수의 모양을 살펴보면 총 3가지 형태의 함수가 있다.

1. -[exchange.AppDelegate application:didFinishLaunch:]
   Objective C의 기본적인 함수의 형태다. (-)로 시작하는 함수는 인스턴스 함수를 의미하며 (+)로 시작하는 함수는 클래스 함수다. [Object Method:Parameter ..] 로 구성되어 있다.
   오른쪽 그림과 같이 somePerson 객체에 존재하는 sayHello 함수를 호출하면 XYZPerson 클래스에 있는 sayHello 함수가 호출되는 방식이다.
   sayHello 함수에서 다시 함수가 호출되는데, 여기서 사용된 self는 C++의 this 포인터와 같은 역할을 한다. 자기 자신의 saySomething 함수를 호출한다. 콜론(:)을 구분자로 파라미터를 전달할 수 있다.
2. sub_100016928



[2] swift로 작성된 앱도 빌드 과정에서 Objective C로 변환되기 때문에 최종적으로 Objective C 문법을 이해하고 있어야 한다.

심볼(Symbol) 정보가 없는 함수다. 함수의 주소값을 이용해 함수에 접근할 수 있다.
Objective C 컴파일러에 의한 코드 최적화 과정에서 코드의 일부가 별도의 함수로 분리된 경우도 이에 속한다. 이 함수와 연관된 Caller 함수는 가까운 주소에 위치하고 있다.

3. nullsub_( )
프로그램에 실제로 존재하는 코드이긴 하나, 최종 버전에는 존재하지 않는 호출이다. 함수의 값이 0으로 설정된 경우 return 과 같은 역할을 한다.

Objective C - Basic #2. objc_msgSend

Message Send는 Objective C를 대표하는 가장 중요한 개념이다. 앞서 살펴본 바와 같이 대괄호 [ ] 사이에 쓰여지는 코드에 의해 Objective C 메시지를 생성한다. Objective C 코드로 **[receiver message]** 와 같이 메시지를 보낸다고 하면 컴파일러는 **objc_msgSend(receiver, selector)**로 코드를 바꾼다. 인자가 있을 경우 **objc_msgSend(receiver, selector, arg1, arg2, ..)** 로 바꾼다.
Objective C가 receiver에게 message를 전달하면 대상 객체(receiver)는 메시지(message)가 누구로부터 왔는지, 클래시 캐시(Classh cache)와 클래스 디스패치 테이블(class dispatch table)을 검색하여 어떤 함수를 실행할지 등의 결정을 한다.

예를 들어, 다음 코드는

```
[self printMessageWithString:@"Hello world!"];
```

컴파일되면 실제로 아래와 같은 코드로 변환되고,

```
objc_msgSend(self, @selector(printMessageWithString:), @"Hello world!");
```

대상 객체의 **isa 포인터**[3]를 이용하여 해당 메시지에 대한 셀렉터의 응답여부를 확인하고 실행한다. 실제로 objc_msgSend 함수는 아무값도 리턴하지 않지만, 이 메시지에 의해 실행되는 함수가 결과를 리턴하기 때문에 objc_msgSend가 리턴하는 것처럼 보인다.

Trace Functions

Frida는 frida-tools에 포함된 명령어를 이용하는 방법과 Frida API를 활용한 스크립트 인젝션(Injection) 두가지 방법으로 사용할 수 있다. 좀 더 자유로운 Frida 사용을 위해 Frida API를 이용해 앱을 후킹(Hooking) 해보자. iOS 앱의 기본이자 앱의 시작인 AppDelegate 클래스를 추적할 것이다. 사용한 스크립트는 0xdea/frida-scripts를 참조하여 작성하였다. 스크립트를 다운로드 받아 raptor_frida_ios_trace.js(이하 trace.js) 파일을 열고 코드 마지막 부분에 아래와 같이 trace("-[exchange.AppDelegate *]") 를 작성하여 저장하여 아래 명령을 실행해보자.

$ frida -U -f com.bitxflow.exchange -l ./raptor_frida_ios_trace.js --no-pause

옵션을 간단히 살펴보면 -U 옵션은 USB 연결, -F 옵션은 실행(spawn)할 앱의 패키지 명, -L 옵션은 불러올 스크립트 파일명, 그리고 --no-pause는 앱을 시작시킨 후 멈춤없이 진행하는 것을 뜻한다.

---

[3] Objective C의 모든 객체는 isa라는 클래스 타입 맴버 변수를 가지고 있다. 다른 클래스를 가리키거나 자기 자신 클래스 객체를 가리키는데 쓰인다.

Trace.js

```
if (ObjC.available) {
    trace("-[exchange.AppDelegate *]");

} else {
    send("error: Objective-C Runtime is not available!");
}
```

Traced AppDelegate

```
*** entered -[exchange.AppDelegate application:didFinishLaunchingWithOptions:]

Caller: 0x18be67100 UIKit!-[UIApplication _handleDelegateCallbacksWithOptions:is
Suspended:restoreState:]

application: <UIApplication: 0x101403150>
didFinishLaunchingWithOptions: nil

retval: 0x1

*** exiting -[exchange.AppDelegate application:didFinishLaunchingWithOptions:]
```

실행된 결과를 보면 exchange.AppDelegate 클래스와 관련된 모든 함수들의 실행 결과를 화면에 출력해준다. 그리고 어떤 함수(Caller)가 호출하였는지, 인자(parameter)는 무엇인지, 그리고 결과값(retval)은 무엇인지 보여준다. 앱이 실행되는 과정에서 어떤 함수들이 호출되는지 살펴볼 때 사용할 수 있다.

Debug Function

Debug라는 제목을 쓰긴 했지만 Frida는 후킹(Hooking)을 이용하기 때문에 정확한 의미에서 Debug는 아니다. 하지만 거시적 관점에서 앱 해킹을 할 때 Debugging을 통해 함수의 입/출력값 조작, 함수의 동작 변경을 수행하고 있기 때문에 Debug라고 해보자. 호출되는 함수들 중에 특정 함수를 살펴보고자 할 땐 다음과 같이 Frida를 사용할 수 있다. 예제 코드는 raptor_frida_ios_autoIntercept.js(이하 autoIntercept.js)를 사용하였다.

autoIntercept.js

```
function autoIntercept(target)
{
    var className = target.match(/^[-+]\[(.*)\s/)[1];
    var methodType = target.match(/^([-+])/)[1];
    var methodName = target.match(/^[-+]\[.*\s(.*)\]/)[1];
    var argCount = (methodName.match(/:/g) || []).length;

    console.log("\ninfo: trying to intercept", target);
    var oldImpl = ObjC.classes[className][methodType + " " + methodName];

    Interceptor.attach(oldImpl.implementation, {

        onEnter: function(args) {
            this.flag = 0;
                this.flag = 1;

            if (this.flag) {

                console.warn("\n*** entered", target, "***");

                // print args
                for (i = 0; i < argCount; i++) {
                    printType("\narg " + (i + 1) + " type:\t", args[i + 2]);
                    printValue("arg " + (i + 1) + " value:\t", args[i + 2]);
                }
            }
        },

        onLeave: function(retval) {

            if (this.flag) {

                // print retval
                printType("\nretval type:\t", retval);
                printValue("retval value:\t", retval);

                console.log("\n*** exiting", target, "***");
            }
        }

    });
}

if (ObjC.available) {

    autoIntercept("-[FIRAuth canHandleURL:]");

}
```

Intercept FIRAuth URL

```
SPIDEY:ios-snippets spidey$ frida -U -f com.bitxflow.exchange -l ./autoIntercept.js

     ____
    / _  |   Frida 12.2.15 - A world-class dynamic instrumentation toolkit
   | (_| |
    > _  |   Commands:
   /_/ |_|       help      -> Displays the help system
   . . . .       object?   -> Display information about 'object'
   . . . .       exit/quit -> Exit
   . . . .
   . . . .   More info at http://www.frida.re/docs/home/
Spawning `com.bitxflow.exchange`...

info: trying to intercept -[FIRAuth canHandleURL:]
Spawned `com.bitxflow.exchange`. Use %resume to let the main thread start executing
!
[iPhone::com.bitxflow.exchange]-> %resume
[iPhone::com.bitxflow.exchange]->
*** entered -[FIRAuth canHandleURL:] ***

arg 1 type:     NSURL NSURL
arg 1 value:    com.googleusercontent.apps. [redacted] -17oukjnck82fl4i9k0p189pu5nr
rfjlf:/oauth2callback?code=4/vwB58vBp7qrJRIdi8B1UlLvsb9XgeRrKMgUP1otNcmViLtZlBnI42a
HuEt-1 [redacted] &scope=email%20profile%20https://www.googleapis.co
m/auth/userinfo.email%20https://www.googleapis.com/auth/userinfo.profile

retval type:    nil nil
retval value:   nil

*** exiting -[FIRAuth canHandleURL:] ***
```

Google Firebase OAuth 인증 요청 URL. 인자(arg)값과 리턴(retval)값을 확인할 수 있고, 원하는 값으로 변경도 가능하다.

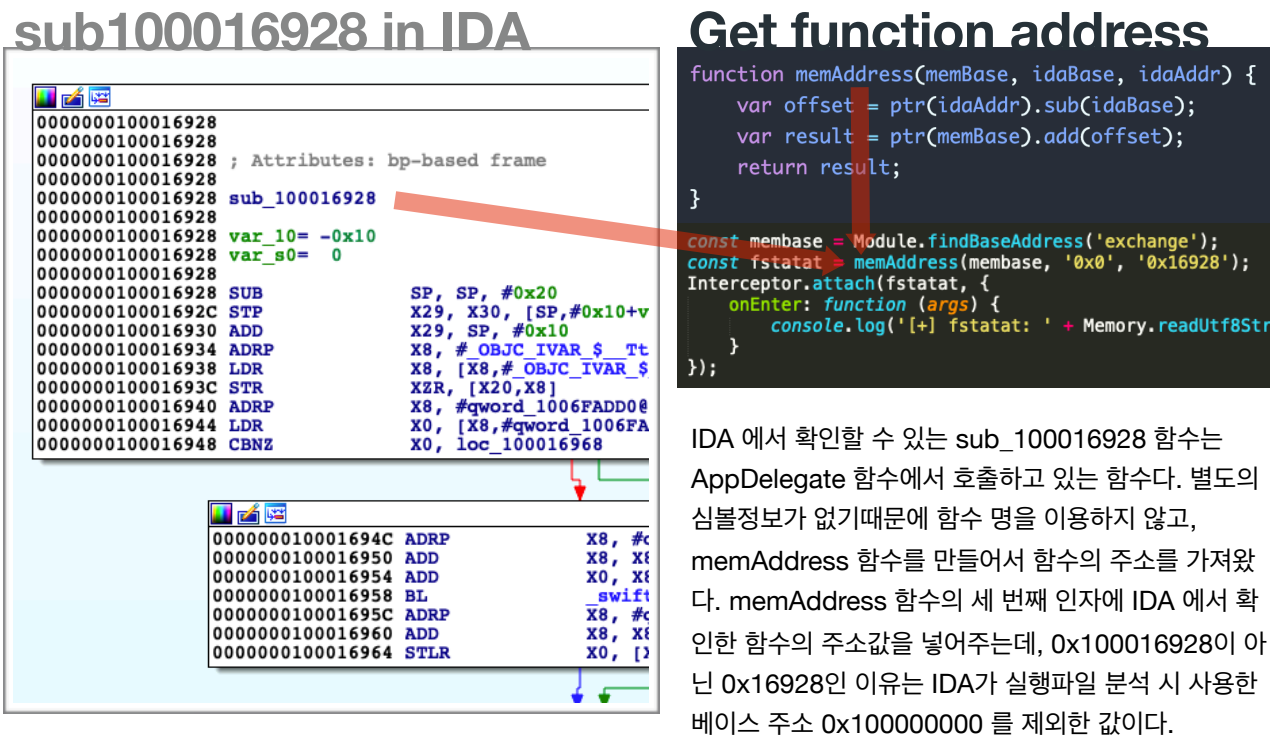
앱의 동작 과정 중 Google Firebase OAuth 인증의 일부분을 확인해보았다. FIRAuth 클래스에 있는 canHandleURL 함수의 인자(arg)값과 리턴(retval)값을 확인해보았는데, 인자값으로 Google에 사용자 정보를 요청하는 것으로 보이는 URL을 전달받고, 아무것도 리턴하지 않았다. Interceptor.attach(oldImpl.implementation, {}) 에 있는 onEnter: function(args) {} 과 onLeave: function(retval) {} 함수를 고쳐 함수의 인자값과 리턴값을 변경할 수 있다.

```
// change retval
var retnew = ObjC.classes.NSString.stringWithString_("false");
retval.replace(retnew);
console.log("new retval value:", obj.toString());
```

새로운 리턴값을 전달하기 위해 onLeave 함수에서 retval 값 변경(replace).

Convert IDA address to memory address

함수의 심볼정보가 없는 함수(예. sub_100016928)의 경우 앞서 사용한 스크립트를 이용해 공격할 수가 없다. IDA에서 확인한 함수의 주소값을 이용해 직접 함수를 후킹해야 한다. 아래 memAddress 함수를 이용해 IDA에서 확인한 주소로 함수가 앱에서 동작할 때의 주소를 가져올 수 있다.



IDA 에서 확인할 수 있는 sub_100016928 함수는 AppDelegate 함수에서 호출하고 있는 함수다. 별도의 심볼정보가 없기때문에 함수 명을 이용하지 않고, memAddress 함수를 만들어서 함수의 주소를 가져왔다. memAddress 함수의 세 번째 인자에 IDA 에서 확인한 함수의 주소값을 넣어주는데, 0x100016928이 아닌 0x16928인 이유는 IDA가 실행파일 분석 시 사용한 베이스 주소 0x100000000 를 제외한 값이다.

Frida CodeShare

보안을 위해 앱들이 가지고 있는 공통적인 기능(예를 들면, SSL Pinning, Anti Jailbreak 등)들이 있다. 이 기능들은 앱 점검을 위해 반드시 우회해야 하는 기능들이다. 이 기능들의 구현은 거의 비슷한 방법으로 이루어지기 때문에 Frida 스크립트를 이용해 우회 한다면 거의 동일한 코드가 작성될 것이다. 이 번거로움을 피하기 위해 Frida CodeShare Project가 만들어졌다. Frida CodeShare Project는 세계 각국의 개발자들로 구성되어 있으며, 다양한 아이디어를 공유하기 위해 만들어졌다. 여기서 우리는 앱의 다양한 보안 기능을 우회하거나 조작할 수 있다.
대표적으로 ios-ssl-bypass 코드가 있다. SSL Pinning 기술을 통해 서버와 통신하는 내용을 안전하게 보호할 수 있지만, 이 코드를 사용하면 손쉽게 우회가 가능하다. 별도의 코드 작성 없이 아래 명령

$ frida —codeshare dki/ios10-ssl-bypass -f <app identifier>

을 실행하면 보안 기능이 바로 우회된다. 다양한 코드가 공유되어 있으니, 검색해보자.

To bring this session to an end

Frida는 사용하는 방법에 따라 무한한 잠재력을 가지고 있고, 계속해서 기능이 추가되고 있다. 최신 버전의 Frida는 커널도 다룰 수 있다. 아이폰뿐만 아니라 안드로이드, 일반 PC, Mac OS 에서 동작하는 앱도 후킹이 가능하다. 갈고 닦아 내 것으로 만들자!!