

# AppSuit Cloud iOS 적용 가이드

v 25.0.0

(주)스틸리언

# Table of Contents

<b>1. 개요</b>	1
a. 적용 환경	1
b. 적용 과정	1
c. 적용 시 필요한 파일	2
<b>2. AppSuit 라이브러리 적용 방법</b>	3
a. AppSuit 라이브러리 파일 추가	3
b. AppSuit Cloud 관련 빌드 설정	5
c. Resource Encryption 관련 적용 방법	8
<b>3. 빌드 및 결과 확인</b>	11
a. Xcode에서 IPA 추출	11
b. AppSuit cloud-iwcm(2차) 적용	12
c. 빌드 결과 확인	14
<b>4. AppSuit Cloud 검증 방법</b>	15
a. AppSuit 솔루션 적용 여부 검증	15
b. Objective-C 문자열 암호화 검증	16
c. Swift 문자열 암호화 검증	18
d. Objective-C 클래스 난독화 검증	20
e. Swift 클래스 난독화 검증	23
f. Dynamic API Hiding 기능 검증	25
g. 심볼 제거 검증	30
h. 로그 데이터 삭제 검증	33
i. 탈옥 탐지 검증	35
j. 위변조 탐지 검증	39
k. Frida, 스위즐링 탐지 검증	52
l. Flex-3 탐지 검증	55
m. 디버깅 탐지 검증	57
<b>5. 배포 방법</b>	59
<b>부록</b>	60
A. [Optional] Bridging Header 설정	61
B. [Optional] Bridging Header 수동 설정	62
C. AppSuit Cloud 2차 적용 옵션	63
D. Class Name Obfuscation 적용 방법	67
E. Class Name Obfuscation 예외처리 방법	68
F. AppSuit Open Page	71

# 1. 개요

- 본 문서는 AppSuit Cloud 솔루션을 iOS Application에 적용하기 위한 절차와 트러블슈팅 등의 내용을 포함하고 있습니다.
- AppSuit Cloud 솔루션은 iOS Application에 대한 탈옥 탐지, 위변조 탐지, 소스코드 난독화 및 암호화 등의 기능을 제공합니다.

## a. 적용 환경

- 개발 환경
  - Xcode : 14 이상
  - iOS : iOS 12.0 이상
- AppSuit module 환경
  - AppSuit 라이브러리 (IXP) : 23.2.0 버전 이상
  - AppSuit cloud-iwcm : 23.2.0 버전 이상

## b. 적용 과정

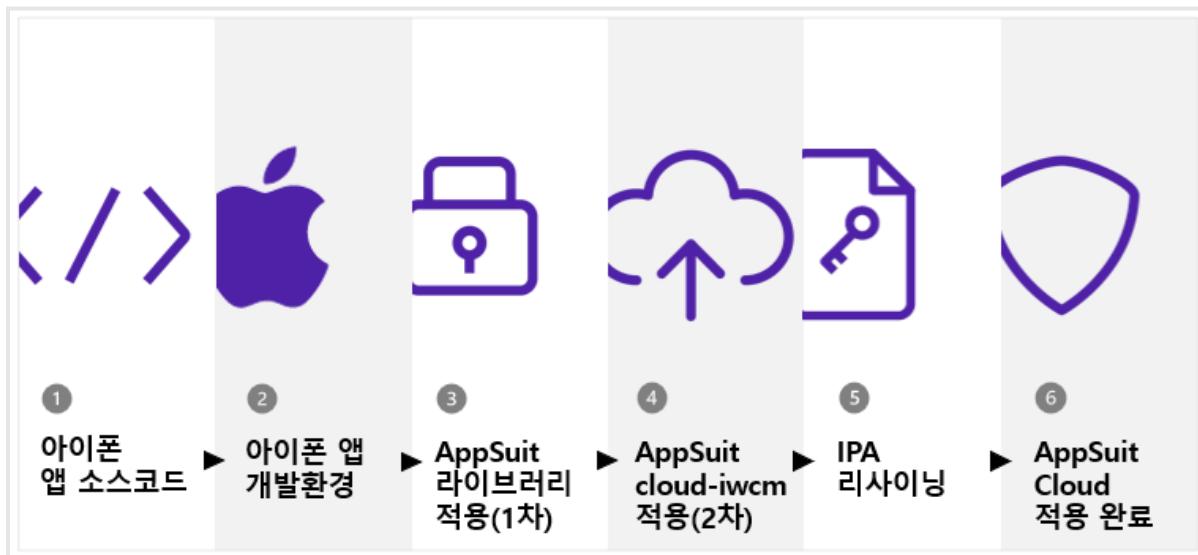


그림 1. AppSuit Cloud 적용 과정

- AppSuit 솔루션의 iOS 적용 과정은 아래와 같습니다.
  - AppSuit 라이브러리(1차)를 프로젝트에 추가 후 아카이브합니다.
  - 아카이브 한 IPA를 AppSuit Cloud 빌드 서버에서 AppSuit cloud-iwcm(2차) 적용을 진행합니다.
  - 최종적으로 IPA 리사이닝을 수행하여 보안이 적용된 앱을 생성합니다.
    - 위 과정은 CLI 기반 XcodePlugin을 통해 자동으로 진행할 수 있습니다.

## c. 적용 시 필요한 파일

### 1. AppSuit 라이브러리

- AppSuit Cloud 1차 적용을 위한 라이브러리 파일입니다.

### 2. AppSuit cloud-iwcm

- AppSuit Cloud 2차 적용을 위한 모듈입니다.  
스틸리언 내부 서버를 사용하는 경우 해당 파일은 제공되지 않습니다.

### 3. AppSuitSign

- AppSuit cloud-iwcm (2차) 빌드 후 재서명을 위한 도구입니다.

### 4. AppSuit Cloud iOS 적용 가이드

## 2. AppSuit 라이브러리 적용 방법

### a. AppSuit 라이브러리 파일 추가

1. Xcode Projects Navigator에서 "AppSuit"라는 이름으로 그룹을 생성합니다.

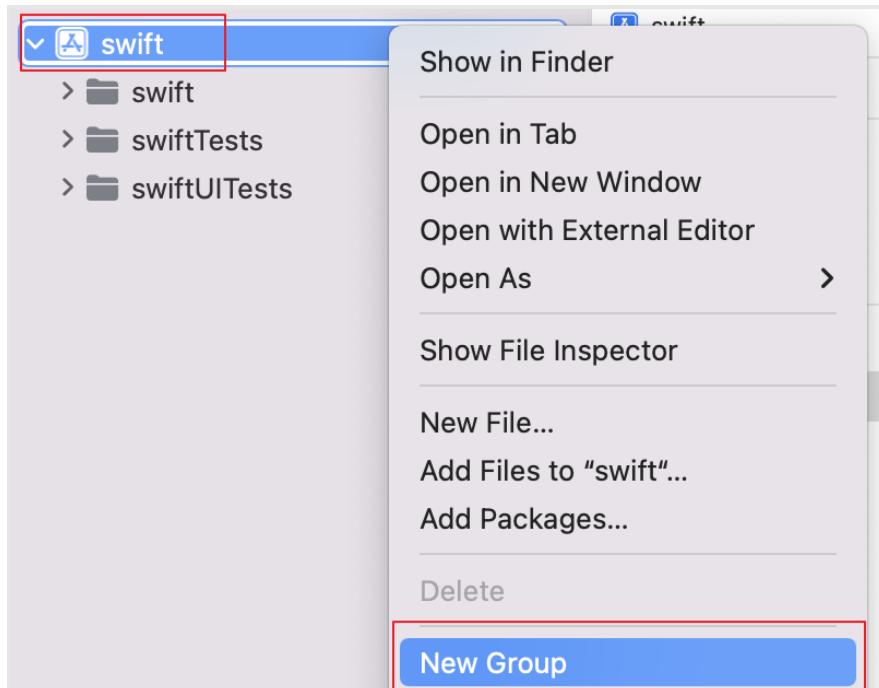


그림 2. Navigator에서 AppSuit 그룹 추가

2. 생성된 AppSuit 그룹에 스틸리언 엔지니어가 전달드린 AppSuit 라이브러리 파일 모두 복사합니다.

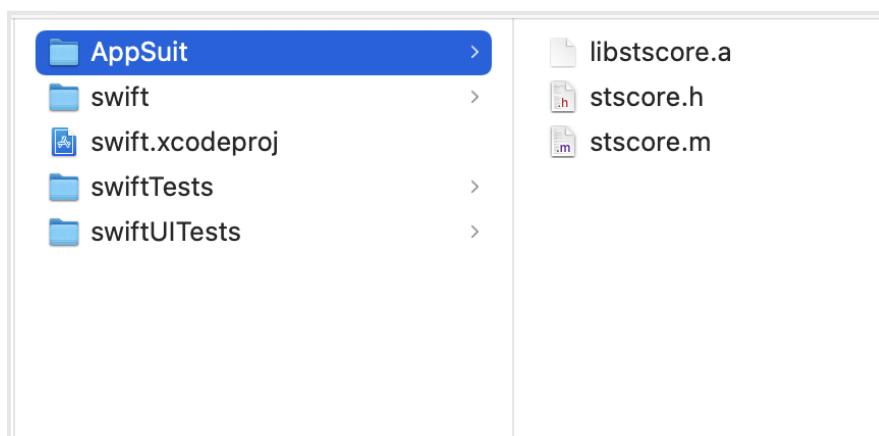


그림 3. 라이브러리 파일 복사

3. Xcode에서 AppSuit 그룹에 "stscore.h , stscode.m" 파일을 추가합니다.

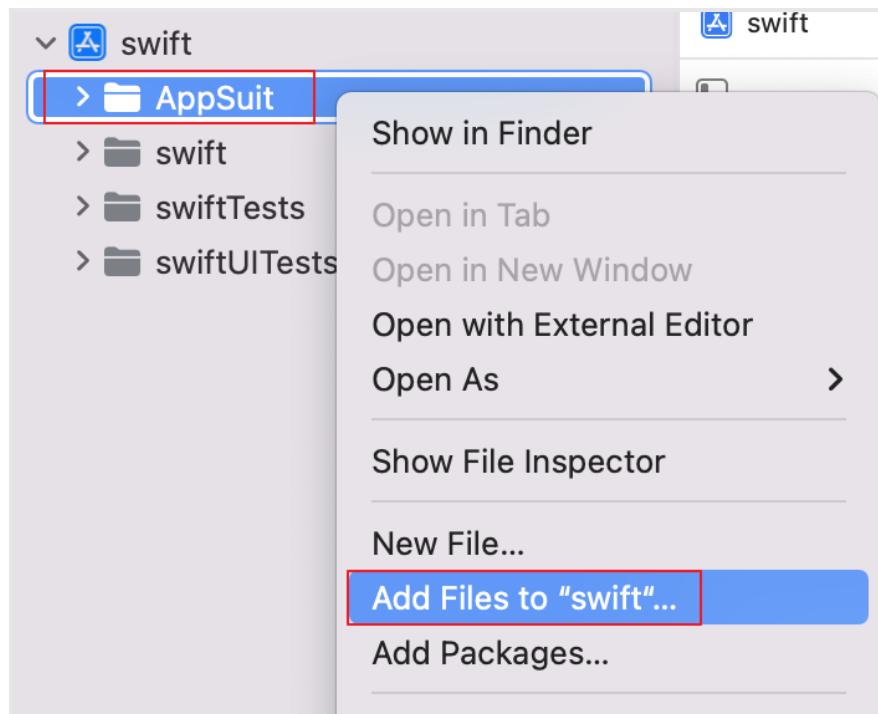


그림 4. 라이브러리 파일 추가

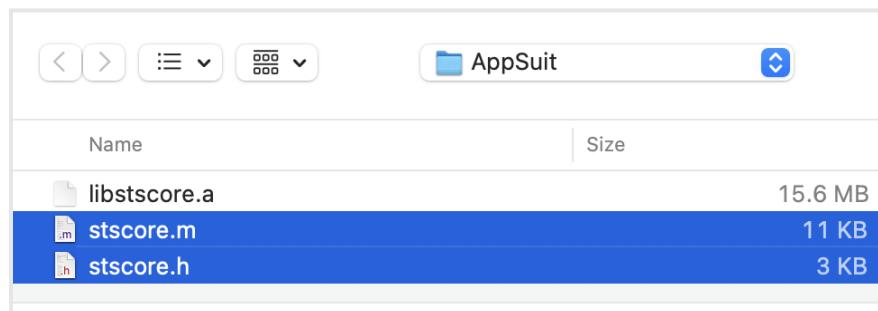


그림 5. AppSuit 그룹에 stscode.h, stscode.m 파일 추가

## b. AppSuit Cloud 관련 빌드 설정

- "Build Settings" - "Search Paths" - "Library Search Paths" 에 libstscore.a 파일이 존재하는 경로를 추가합니다.

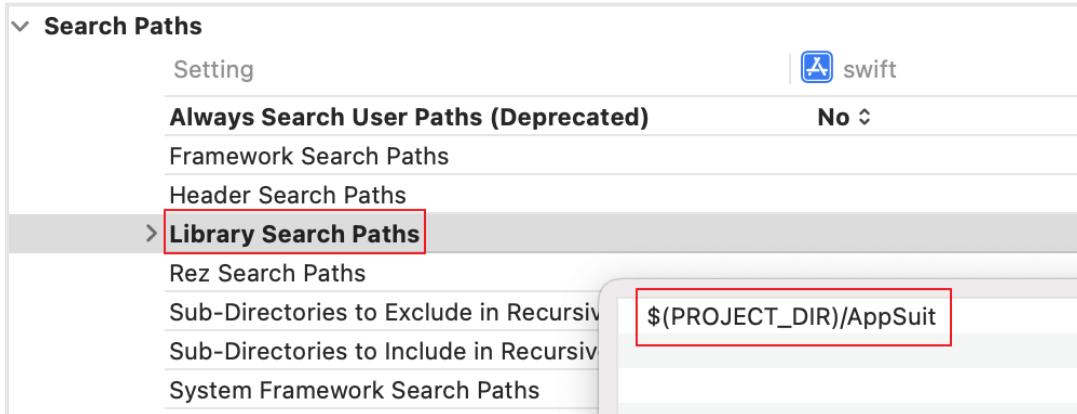


그림 6. 라이브러리 경로 추가

- "Build Settings" - "Linking" - "Other Linker Flags" 에 "-lstscore"를 추가합니다.  
(Debug 모드에서는 라이브러리가 Linking 되지 않도록, Release , Archive 등의 모드에만 적용합니다.)

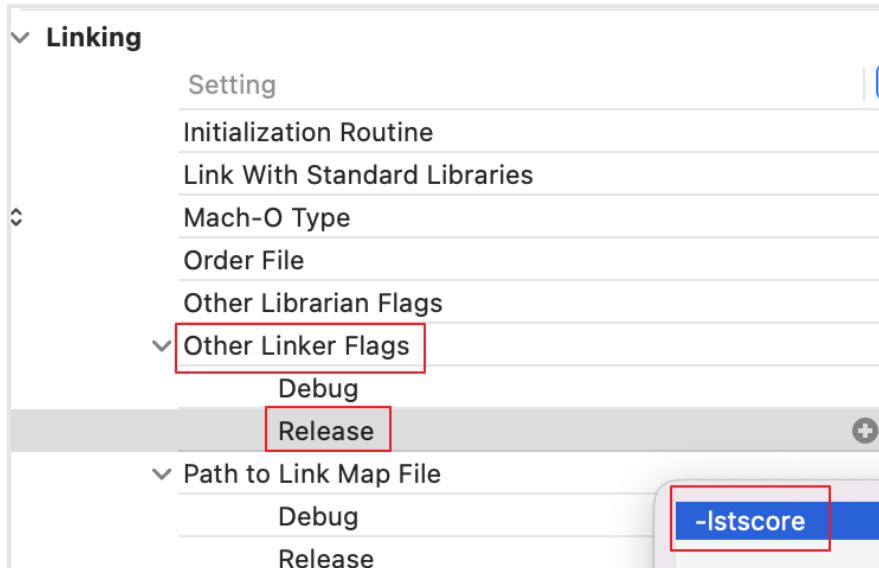


그림 7. Linker Flags 추가

### 3. "Build Settings" = "Build Options" - "Excluded Source File Names"에 "stscore.m" 파일을

추가합니다.

(아래 그림의 **Debug** Configuration 과 같이 AppSuit가 적용되지 않을 Configuration은 라이브러리의 API가 호출되지 않게 하기 위해 "stscore.m" 파일을 exclude 해줍니다.)



그림 8. Excluded Files 추가

### 4. "Build Settings" - "Build Options" - "Enable Bitcode"를 No로 설정합니다.

- a. Xcode 15 이상 버전부터 해당 기능은 Deprecate되어 Xcode 15 이상 버전을 사용하는 경우 별도로 설정하지 않아도 됩니다.

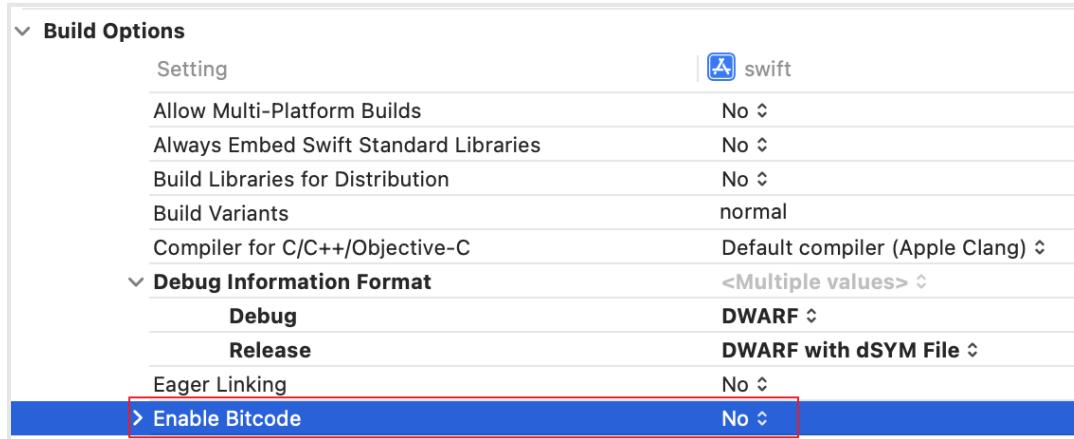


그림 9. 비트코드 비활성화

5. AppSuit 가 적용될 Build Configuration 에게는 "DEBUG" compile flag 가 전달되지 않도록 설정합니다.  
(아래 그림의 **QA**, **Release** Configuration 과 같이 AppSuit 가 적용될 Configuration 에게는 "DEBUG" flag 가 남아 있지 않도록 설정합니다.)

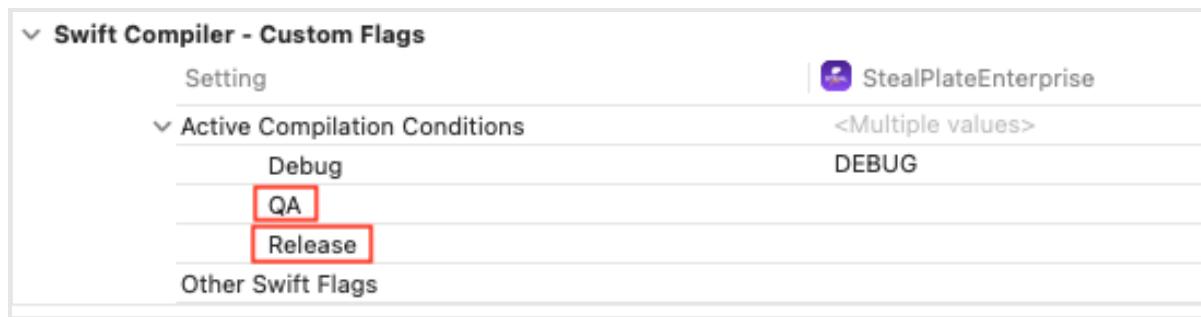


그림 10. Compile Flags 설정

## c. Resource Encryption 관련 적용 방법

- 개요

- iOS Native App 의 경우 **Encrypt Specified Files** 옵션을 사용하여 임의로 지정한 파일들을 암호화할 수 있습니다. 해당 옵션을 사용할 경우 AppSuit 에서 제공하는 API 를 사용하여 암호화한 파일들을 복호화할 수 있습니다.
- React-Native 프로젝트의 경우 **Encrypt main.jsbundle** 옵션을 사용하여 main.jsbundle 파일을 암호화할 수 있습니다. 해당 옵션을 사용할 경우 AppDelegate 에서 BundleURL 을 임의로 지정해 주어야 합니다.

- Encrypt Specified Files 옵션 적용 방법

1. "Build Phases" - "Copy Bundle Resources" 에 암호화 대상 파일을 추가합니다.

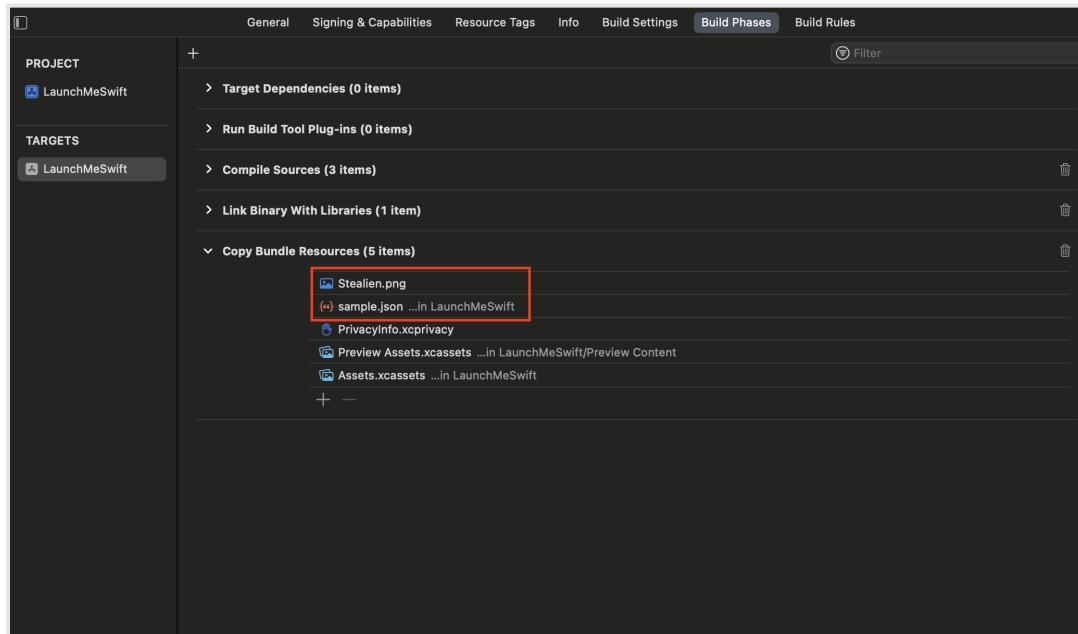


그림 11. 번들에 암호화 대상 파일 추가

2. 빌드 페이지의 **Resource Encryption** 텍스트 필드에 암호화 대상 파일을 지정하여 빌드합니다.

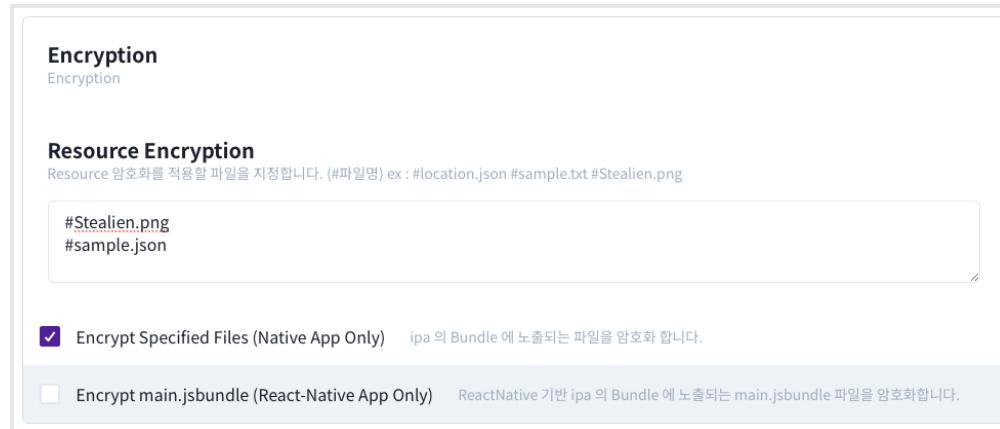


그림 12. 암호화 대상 파일 지정

3. 암호화한 파일은 stscore.h 파일의 AS\_Decrypt\_Resource 함수를 사용하여 불러옵니다.

```
NSData* AS_Decrypt_Resource(NSString *fileName, NSString *ofType);

// Swift
if let stealienImageData = AS_Decrypt_Resource("Stealien", "png") {
    if let image = UIImage(data: stealienImageData) {
        self.stealienImage = image
    }
}

// ObjectiveC
NSData *stealienImageData = AS_Decrypt_Resource(@"Stealien", @"png");
UIImage *stealienImage = [[UIImage alloc]
initWithData:stealienImageData];
```

- Encrypt main.jsbundle 옵션 적용 방법

1. ios/Project Directory/AppDelegate.mm 으로 이동합니다.

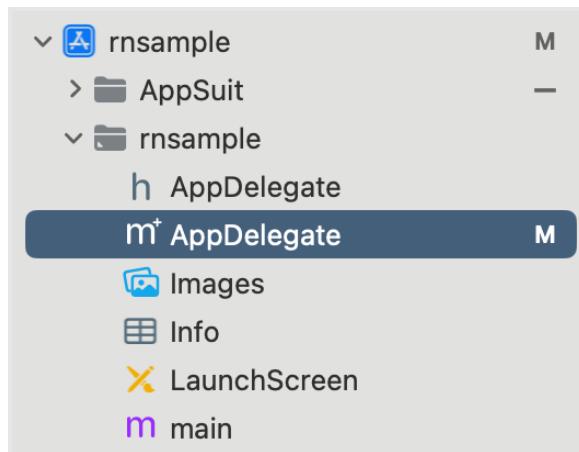


그림 13. AppDelegate.mm 파일

2. - (NSURL \*)getBundleURL 델리게이트 메서드를 찾아 아래 코드와 같이 구현 해줍니다.

```
- (NSURL *)getBundleURL
{
#if DEBUG
    return [[RCTBundleURLProvider sharedSettings]
    jsBundleURLForBundleRoot:@"index"];
#else
    NSString *sAppSuitMain = @"AppSuitMain";
    NSURL *documentsDirectoryURL = [[[NSFileManager defaultManager]
    URLsForDirectory:NSDocumentDirectory inDomains:NSUserDomainMask]
    lastObject];

    NSURL *mainJSFileURL = [documentsDirectoryURL
    URLByAppendingPathComponent:sAppSuitMain];

    return mainJSFileURL;
#endif
}
```

### 3. 빌드 및 결과 확인

#### a. Xcode에서 IPA 추출

1. 2차 기능 적용을 위해, Archive하여 IPA 파일을 추출합니다. (Release 모드로 빌드)

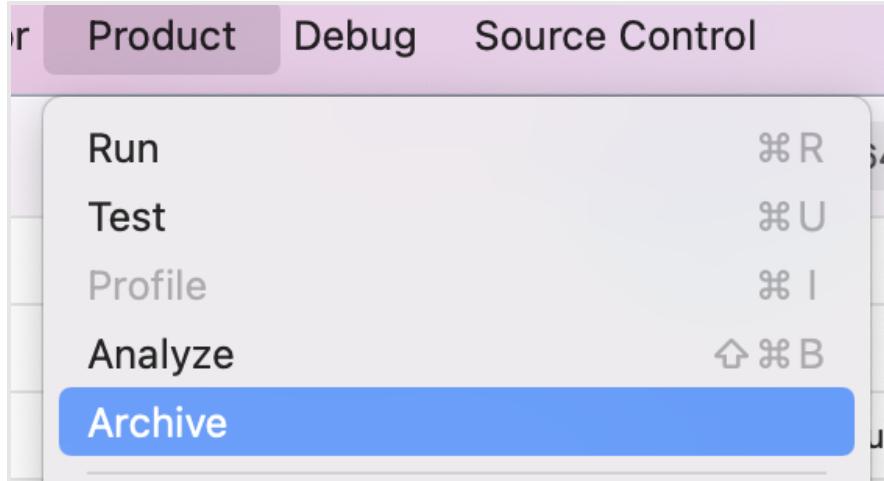


그림 14. "Archive" 버튼 클릭

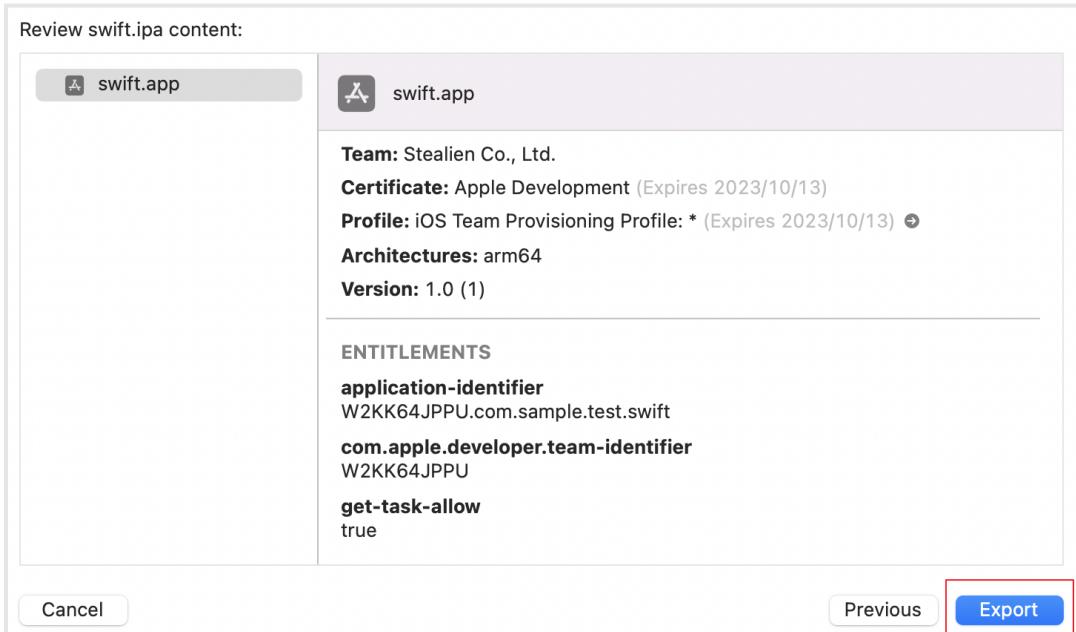


그림 15. "Export" 버튼 클릭

## b. AppSuit cloud-iwcm(2차) 적용

1. AppSuit Cloud 빌드 서버에 접속하여 인증된 계정으로 로그인합니다.
  - 해당 계정은 스틸리언 엔지니어가 발급하여 전달드립니다.
2. 화면 가운데에서 "Premium iOS"를 선택합니다.

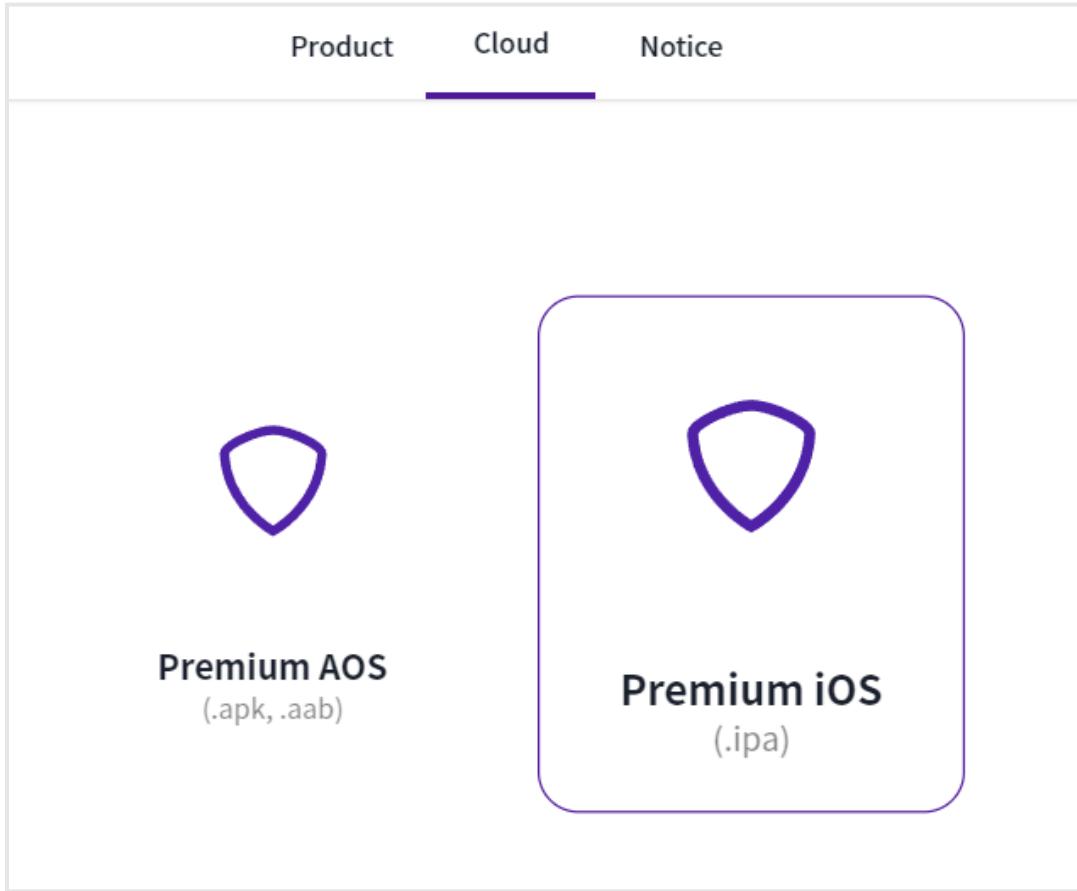


그림 16. "Premium iOS" 버튼 클릭

3. **Upload** 버튼을 클릭합니다.
4. AppSuit cloud-iwcm(2차)을 적용하려는 IPA 파일을 선택합니다.
5. [부록-C. AppSuit Cloud 2차 적용 옵션] 을 참고하여 옵션을 선택합니다.
6. "**Next**" 버튼을 클릭합니다.
  - AppSuit cloud-iwcm 적용을 진행하며, 잠시 후 다운로드 버튼이 활성화 됩니다.

7. "Download" 버튼을 눌러 AppSuit cloud-iwcm(2차) 적용한 파일을 다운로드 합니다.

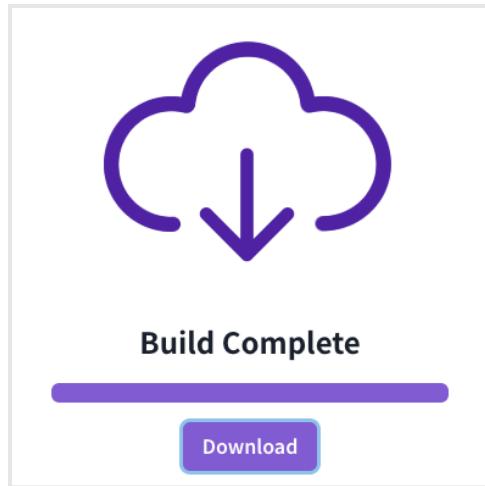


그림 17. "Download" 버튼 클릭

8. AppSuit에서 제공한 **AppSuitSign** 도구를 이용하여 다운로드 한 파일을 리사이닝합니다.

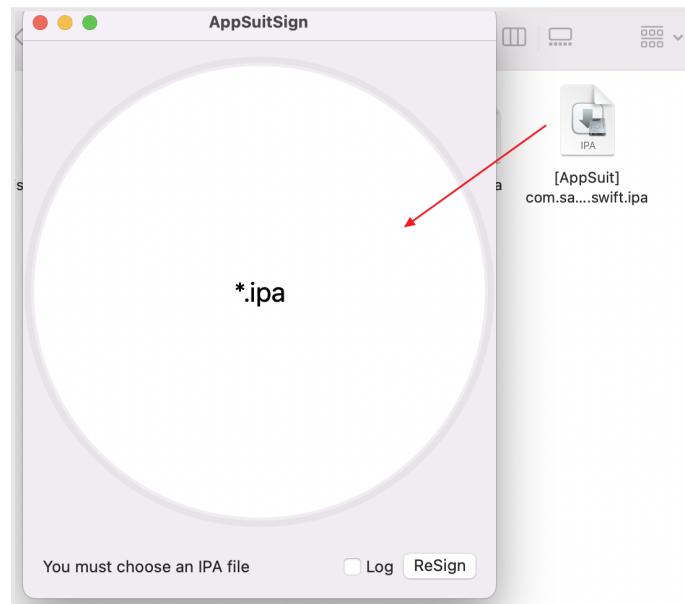


그림 18. AppSuitSign을 이용하여 리사이닝

## c. 빌드 결과 확인

- AppSuit Cloud 빌드 서버에서 확인하는 방법

- 빌드서버 페이지 우측 상단의 아이콘을 클릭합니다.
- 빌드 로그를 선택합니다.
- 적용되었던 앱의 목록이 표시됩니다.
  - Download 버튼이 활성화되어야 적용이 완료된 상태입니다.

선택	APP TYPE	PACKAGE NAME	BUILD VER.	REQUEST USER ID	TIME	STATUS	APP DOWNLOAD
<input type="checkbox"/>		com.stealien.LaunchMeSwift	v23.2.0	stealien	2023/07/14 17:25	SUCCESS	Download
<input type="checkbox"/>		com.stealien.LaunchMeSwift	v23.2.0	stealien	2023/07/14 17:25	SUCCESS	Download

그림 19. 빌드 결과 조회

- 디바이스에서 확인하는 방법

- 해당 방법의 경우 Xcode에서 Development로 Archive 시 가능합니다.

- 디바이스에 리사이닝까지 완료된 IPA를 설치합니다.
- 설치한 IPA를 실행하여 정상적으로 실행되는지 확인합니다.

# 4. AppSuit Cloud 검증 방법

## a. AppSuit 솔루션 적용 여부 검증

- 검증 환경

- AppSuit Cloud 적용 후 리사이닝까지 완료된 IPA
- XMachOViewer
  - XMachOViewer는 <https://github.com/horsicq/XMachOViewer/releases>에서 다운받을 수 있습니다

- AppSuit 솔루션 적용 여부 검증 시나리오

- 주석 > 세그먼트 > 섹션에서 \_\_data 섹션에서 AppSuit 적용 버전을 확인합니다.
- 주석 > 세그먼트 > 섹션에서 stscore1, stscore2 섹션이 존재하는 경우 AppSuit 라이브러리 적용(1차) IPA입니다.
- 해당 섹션이 각각 cstring, ustring이면 AppSuit cloud-iwcm 적용(2차) IPA입니다.
  - Enterprise Build 시 stscore1, stscore2 섹션이 생성되지 않습니다.

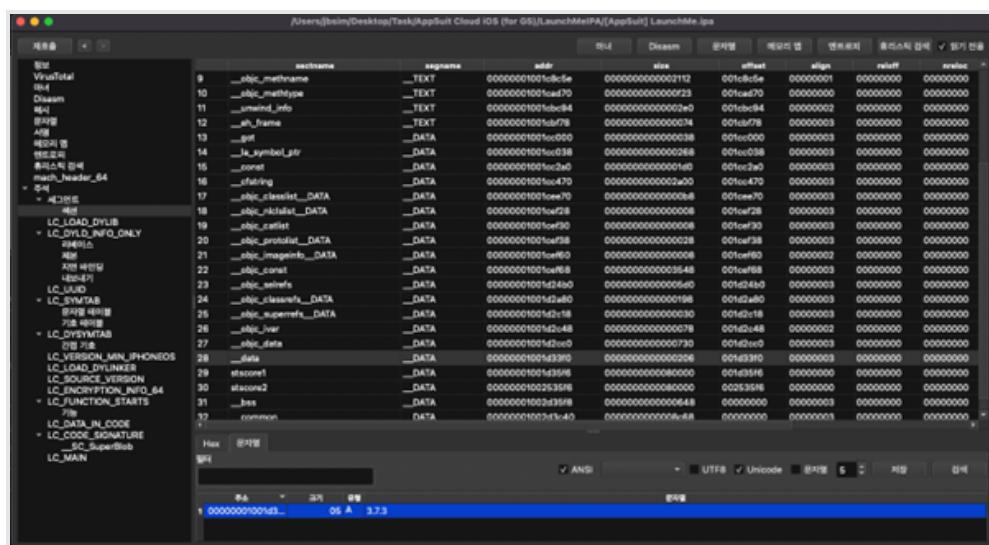


그림 20. 섹션 확인

## b. Objective-C 문자열 암호화 검증

- 검증 환경

- IDA 또는 Hopper Disassembler
  - Hopper Disassembler는 <https://www.hopperapp.com>에서 다운받을 수 있습니다
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
- AppSuit Cloud 미적용 IPA

- Objective-C 문자열 암호화 검증 시나리오(Using. IDA)

- AppSuit Cloud 미적용 IPA의 메인 바이너리 파일을 IDA 프로그램에서 실행합니다.
- Ctrl+1 단축키를 실행하여 Segments - \_\_cfstring를 클릭합니다.
- AppSuit Cloud 적용 IPA에 대해 동일하게 진행합니다.
- AppSuit Cloud 미적용 앱과 적용 앱을 비교하여 문자열 암호화된 것을 확인합니다.

- 문자열 암호화가 적용된 경우 기존에 보이던 문자열이 의미없는 글자들로 보이게 됩니다.

```
cfstring:00B48A50 stru_B48A50  _CFString <__CFConstantStringClassReference, 0x7C8, \
cfstring:00B48A50 ; DATA XREF: -[FIDO dereg]+24↑o
cfstring:00B48A50 ; -[FIDO dereg]+2A↑o ...
cfstring:00B48A50 ; [FIDO dereg]+15C↑o
cfstring:00B48A60 stru_B48A60  _CFString <__CFConstantStringClassReference, 0x7D0, asc_CF917A, 0xD>
cfstring:00B48A60 ; DATA XREF: -[FIDO run]!+156↑o
cfstring:00B48A60 ; [FIDO run]+15C↑o
cfstring:00B48A60 ; "防火墙的端口必须修改" [red box]
cfstring:00B48A70 stru_B48A70  _CFString <__CFConstantStringClassReference, 0x7D0, asc_CF9196, 0xB>
cfstring:00B48A70 ; DATA XREF: -[FIDO run]!+236↑o
cfstring:00B48A70 ; -[FIDO run]!+23E↑o ...
cfstring:00B48A70 ; "端口修改成功" [red box]
cfstring:00B48A70 ; "修改成功"
cfstring:00B48A80 stru_B48A80  _CFString <__CFConstantStringClassReference, 0x7D0, asc_CF91AE, 8>
cfstring:00B48A80 ; DATA XREF: -[CertIDPasswordViewController viewDidLoad]!+46↑o
cfstring:00B48A80 ; -[CertIDPasswordViewController viewDidLoad]!+4C↑o ...
cfstring:00B48A80 ; "长期存储将被重置" [red box]
cfstring:00B48A90 cfstr_UAS   _CFString <__CFConstantStringClassReference, 0x7D0, auAS, 0xD>
cfstring:00B48A90 ; DATA XREF: -[CertIDPasswordViewController getMessage]!+loc_25332↑o
cfstring:00B48A90 ; -[CertIDPasswordViewController getMessage]!+loc_25333↑o
cfstring:00B48A90 ; "端口修改成功" [red box]
cfstring:00B48A90 ; "端口修改成功" [red box]
cfstring:00B48AA0 stru_B48AA0  _CFString <__CFConstantStringClassReference, 0x7D0, asc_CF91DC, 0xE>
cfstring:00B48AA0 ; DATA XREF: -[CertIDPasswordViewController getMessage]!+134↑o
cfstring:00B48AA0 ; -[CertIDPasswordViewController getMessage]!+13C↑o
cfstring:00B48AA0 ; "端口修改成功" [red box]
cfstring:00B48AB0 stru_B48AB0  _CFString <__CFConstantStringClassReference, 0x7C8, \
cfstring:00B48AB0 ; DATA XREF: -[CertIDPasswordViewController appendCell]!+234↑o
cfstring:00B48AB0 ; -[CertIDPasswordViewController appendCell]!+23C↑o
cfstring:00B48AB0 ; "端口修改成功" [red box]
cfstring:00B48AC0 stru_B48AC0  _CFString <__CFConstantStringClassReference, 0x7D0, asc_CF91FA, 3>
cfstring:00B48AC0 ; DATA XREF: -[CertIDPasswordViewController touchedViewObject:cell:textField:]!+1D6↑o
cfstring:00B48AC0 ; -[CertIDPasswordViewController touchedViewObject:cell:textField:]!+1E0↑o ...
cfstring:00B48AC0 ; "端口修改成功" [red box]
```

그림 21. IDA를 이용한 문자열 암호화 확인

- Objective-C 문자열 암호화 검증 시나리오(Using. Hopper Disassembler)

1. AppSuit Cloud 미적용 IPA의 메인 바이너리 파일을 Hopper Disassembler 프로그램에서 실행합니다.
2. 왼쪽 Labels에서 cfstring을 검색합니다.

The screenshot shows the Hopper Disassembler interface with a search bar at the top containing 'Q v cfstring'. Below it is a 'Tag Scope' dropdown. A table lists memory addresses, types, and names. The names column contains Objective-C class and method names such as 'aCfstringcreate', 'aCfstringgettyp', 'aCfstringgetcst', 'cfstring\_Main', etc. The table has columns for Address, Type, and Name.

Address	Type	Name
0x1001c874a	A	aCfstringcreate
0x1001c878b	A	aCfstringgettyp
0x1001c879d	A	aCfstringgetcst
0x1001cc470	D	cfstring_Main
0x1001cc490	D	cfstring_ResultsController
0x1001cc4b0	D	cfstring_color
0x1001cc4d0	D	cfstring_text
0x1001cc4f0	D	cfstring_U_X_
0x1001cc510	D	cfstring_____
0x1001cc530	D	cfstring_AS_SCRAMClientFirstMsg_____NULL
0x1001cc550	D	cfstring_STEALIEN_clientFirstMsg_s
0x1001cc570	D	cfstring_CHALLENGE1
0x1001cc590	D	cfstring_dataResponse1_____NULL
0x1001cc5b0	D	cfstring_STEALIEN_dataResponse1_____
0x1001cc5d0	D	cfstring_DATA
0x1001cc5f0	D	cfstring_strResponse1_____NULL

그림 22. AppSuit Cloud 적용 전 문자열 암호화

3. AppSuit Cloud 적용 IPA에 대해 동일하게 진행합니다.
4. AppSuit Cloud 미적용 앱과 적용 앱을 비교하여 문자열 암호화된 것을 확인합니다.

The screenshot shows the Hopper Disassembler interface with a search bar at the top containing 'Q v cfstring'. Below it is a 'Tag Scope' dropdown. A table lists memory addresses, types, and names. The names column contains heavily obfuscated Objective-C class and method names such as 'aCfstringcreate', 'aCfstringgettyp', 'aCfstringgetcst', 'cfstring\_h\_p\_\_\_\_\_L\_F\_a\_Z\_u\_y\_U', etc. The table has columns for Address, Type, and Name.

Address	Type	Name
0x1001c874a	A	aCfstringcreate
0x1001c878b	A	aCfstringgettyp
0x1001c879d	A	aCfstringgetcst
0x1001cc470	D	cfstring_h_p_____L_F_a_Z_u_y_U
0x1001cc490	D	cfstring_____L_F_a_Z_u_y_U_I_
0x1001cc4b0	D	cfstring_u_y_U_I_C_D_dj_____
0x1001cc4d0	D	cfstring_y_U_I_C_D_dj_____0_
0x1001cc4f0	D	cfstring_J_L_0_____0_U
0x1001cc510	D	cfstring_L_0_____0_U_WsI_
0x1001cc530	D	cfstring_0_2N_Y_g_P_X_a
0x1001cc550	D	cfstring_x_a_a_Y_c_i_____
0x1001cc570	D	cfstring_c_i_____Fx_1_0
0x1001cc590	D	cfstring_____Fx_1_0_L_r_
0x1001cc5b0	D	cfstring_1_D_L_r_i_q_j_
0x1001cc5d0	D	cfstring_i_q_j_o_eq_Bc_Du_J_T_
0x1001cc5f0	D	cfstring_q_j_o_eq_Bc_Du_J_T_v_

그림 23. AppSuit Cloud 적용 후 문자열 암호화

## c. Swift 문자열 암호화 검증

- 검증 환경
  - Hopper Disassembler
  - 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
  - AppSuit Cloud 미적용 IPA
- Swift 문자열 암호화 검증 시나리오(Using. Hopper Disassembler)
  1. AppSuit Cloud 빌드 페이지의 빌드 로그에서 "Download Log"를 클릭하여 "dump.log" 파일을 준비합니다.
  2. "dump.log" 파일의 Swift String Encryption Self 항목에서 암호화된 주소 값 중 검증하고 싶은 주소 값을 복사합니다.

```
[DUMP] ---- Swift String Encryption Self ---- [ ***** ]  
[DUMP] ---- Swift String Encryption Self Depth ---- [ 3 ]  
  
[DUMP] ---- Manipulate target assembly code ---- [ Start ]  
  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100005b08 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100006f80 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x1000071ec ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x1000072cc ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x1000074e0 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000766c ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100007748 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100007c08 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100008748 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100008bcc ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100008ec0 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000a73c ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000a8a4 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000b3bc ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000b5fc ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000b7c4 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000c820 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000cac8 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000cd28 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000d0c8 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000e5a4 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000e6cc ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000ed14 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x10000eeb0 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100013fa8 ]  
[DUMP] ---- Encrypt Instrunction Address ---- [ 0x100014044 ]
```

그림 24. dump.log 파일의 암호화된 Swift 문자열 주소

3. AppSuit Cloud 미적용 IPA의 메인 바이너리 파일을 Hopper Disassembler 프로그램에서 실행합니다.

4. Hopper Disassembler에서 IPA 파일을 불러온 후, 키보드의 영어 ‘g’ 키를 눌러 복사한 address 값을 입력합니다.

```
0000000100024b00    mov      x21, #0x10
0000000100024b04    movk     x21, #0xd000, lsl #48
0000000100024b08    adrp     x8, #0x10051d000          ; 0x10051d260@PAGE
0000000100024b0c    add      x8, x8, #0x260         ; 0x10051d260@PAGEOFF, "To make your feelings better"
0000000100024b10    orr      x9, x21, #0xc
0000000100024b14    sub      x8, x8, #0x20          ; 0x10051d240
0000000100024b18    orr      x8, x8, [x19, #0x20]
0000000100024b1c    stp      x9, x8, [x19, #0x20]
0000000100024b20    str      x0, [x19, #0x30]
0000000100024b24    bl       sub_100025388           ; sub_100025388
0000000100024b28    cmp      x0, #0x1
0000000100024b2c    b.eq    loc_100024bb4
```

그림 25. AppSuit Cloud 적용 전 문자열 암호화

5. AppSuit Cloud 적용 IPA에 대해 동일하게 진행합니다.

6. AppSuit Cloud 미적용 앱과 적용 앱을 비교하여 참조 된 문자열을 식별하지 못하는 것을 확인합니다.

```
0000000100024b00    mov      x21, #0x10
0000000100024b04    movk     x21, #0xd000, lsl #48
0000000100024b08    adrp     x8, #0x100648000          ; 0x100648ac8@PAGE
0000000100024b0c    add      x8, x8, #0xac8         ; 0x100648ac8@PAGEOFF, 0x100648ac8
0000000100024b10    orr      x9, x21, #0xc
0000000100024b14    sub      x8, x8, #0x20          ; 0x100648aa8
0000000100024b18    orr      x8, x8, #0x8000000000000000
0000000100024b1c    stp      x9, x8, [x19, #0x20]
0000000100024b20    str      x0, [x19, #0x30]
0000000100024b24    bl       sub_100025388           ; sub_100025388
0000000100024b28    cmp      x0, #0x1
0000000100024b2c    b.eq    loc_100024bb4
```

그림 26. AppSuit Cloud 적용 후 문자열 암호화

## d. Objective-C 클래스 난독화 검증

- 검증 환경

- XMachOViewer
- Beyond Compare 등 파일 비교 툴
  - Beyond Compare는 <https://www.scootersoftware.com>에서 다운받을 수 있습니다
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
- AppSuit 라이브러리(1차)만 적용한 IPA

- Objective-C 클래스 난독화 검증 시나리오

- AppSuit 라이브러리(1차) 적용만 진행한 IPA 파일을 불러온 후,

주석 > 세그먼트 > 섹션 > `__objc_classname__TEXT` 섹션에서 클래스 이름 리스트를 확인합니다.

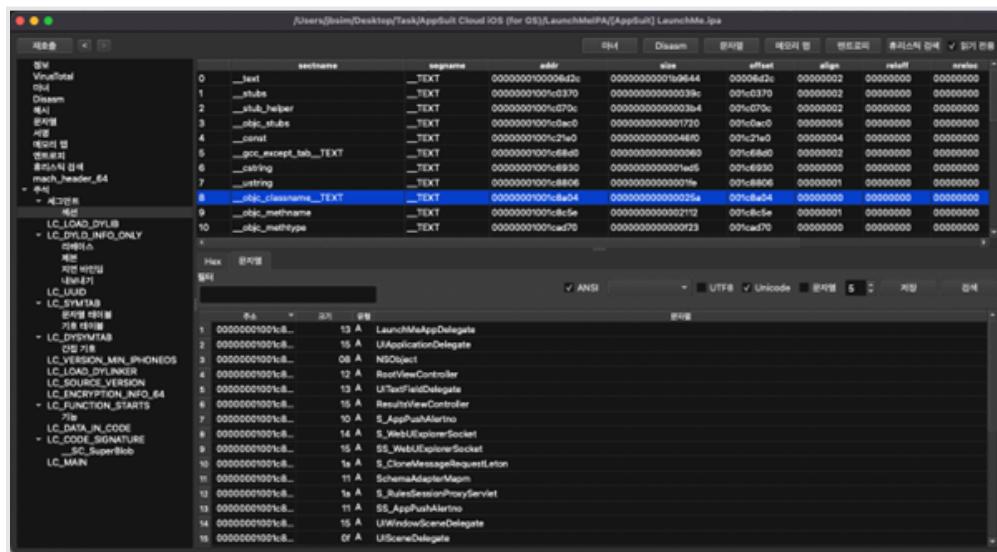


그림 27. `__objc_classname__TEXT` 섹션에서 클래스 이름 리스트 확인

2. 저장 버튼 클릭하여 클래스 난독화 적용 전 리스트 파일을 저장합니다.

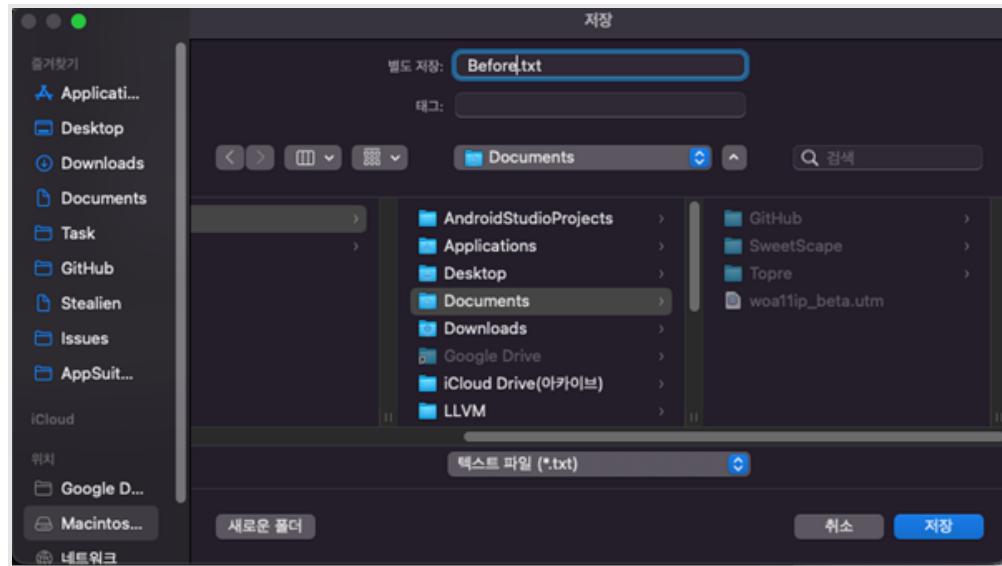


그림 28. AppSuit 라이브러리 적용(1차)한 IPA의 클래스 리스트 저장

3. 동일한 과정으로 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA 리스트 파일을 저장합니다.

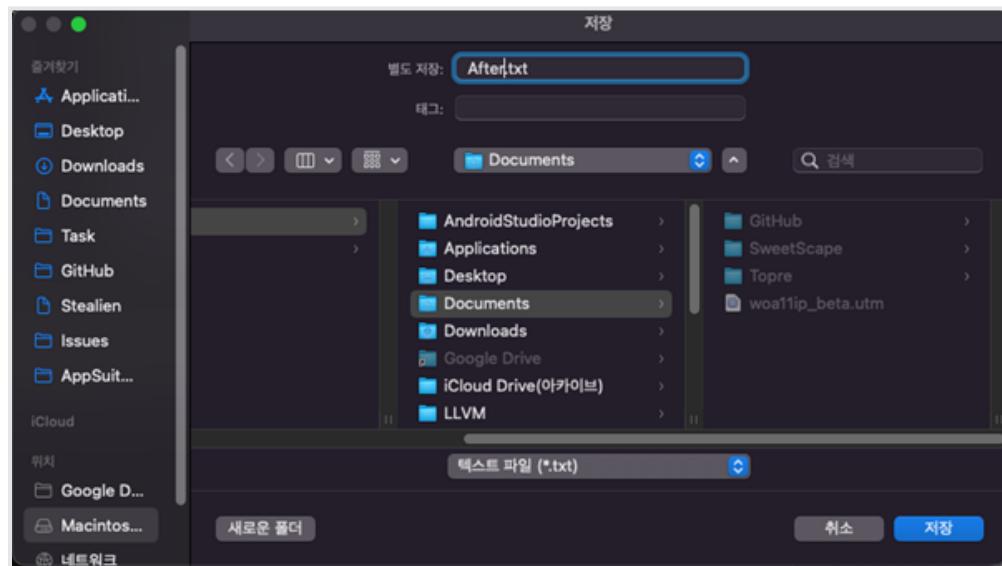


그림 29. AppSuit cloud-iwcm 적용(2차)한 IPA의 클래스 리스트 저장

#### 4. Beyond Compare 등 파일 비교 툴 사용하여 클래스 난독화를 확인합니다.

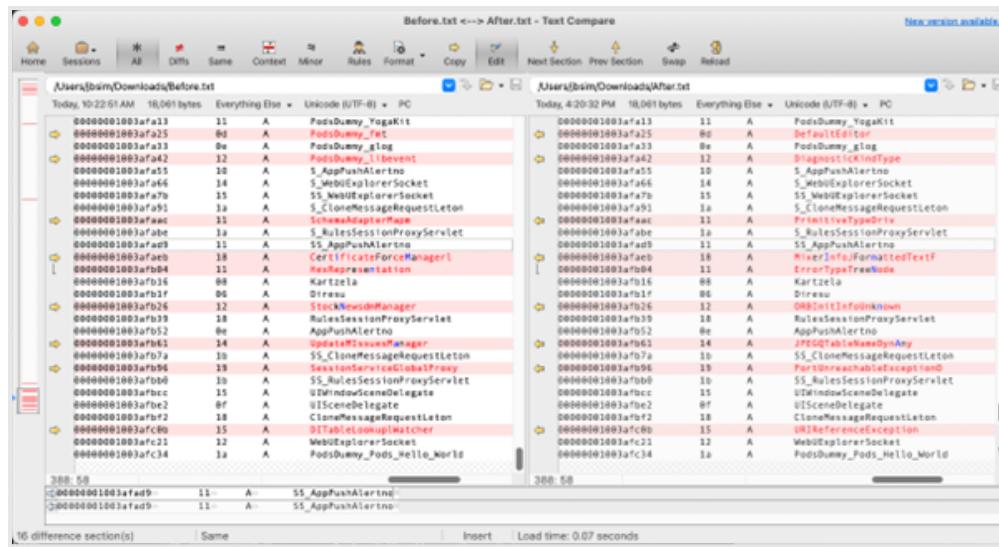


그림 30. 클래스 리스트 비교

- 일부 클래스에 대해 난독화가 진행되지 않는 이유

- iOS 앱 내에서 기본적으로 사용되는 시스템 클래스들(NS···, objc.. 등)과 delegate 관련 클래스 및 UI 관련 클래스들의 경우 난독화 시 앱 런타임에 문제가 생겨 AppSuit 내부 로직에 의해 자동으로 예외처리 됩니다.

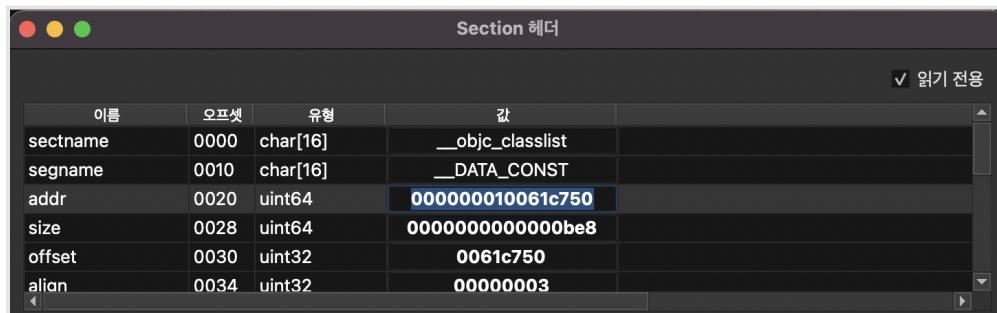
## e. Swift 클래스 난독화 검증

- 검증 환경

- XMachOViewer
- Hopper Disassembler
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
- AppSuit 라이브러리(1차)만 적용한 IPA

- Swift 클래스 난독화 검증 시나리오

1. XMachOViewer를 통해 AppSuit 라이브러리(1차) 적용만 진행한 IPA 파일을 불러온 후, \_\_objc\_classlist\_\_DATA\_CONST 섹션의 address 값을 복사합니다.



이름	오프셋	유형	값
sectname	0000	char[16]	__objc_classlist
segname	0010	char[16]	__DATA_CONST
addr	0020	uint64	000000010061c750
size	0028	uint64	0000000000000be8
offset	0030	uint32	0061c750
alion	0034	uint32	00000003

그림 31. \_\_objc\_classlist\_\_DATA\_CONST 섹션에서 address 값 복사

2. Hopper Disassembler에서 동일한 IPA 파일을 불러온 후, 키보드의 영어 'g' 키를 눌러 복사한 address 값을 입력합니다.

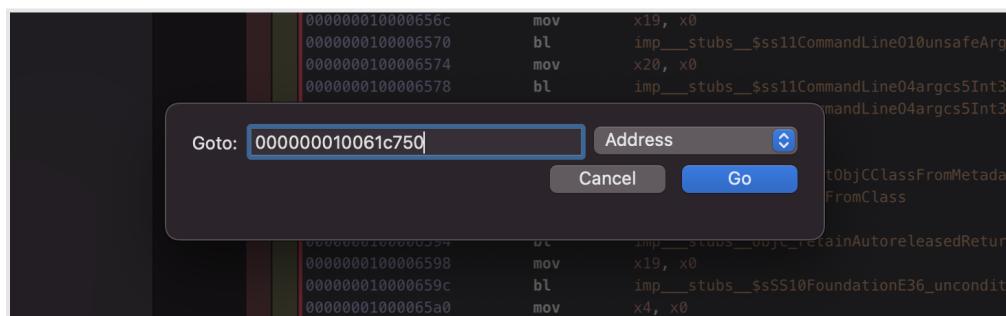


그림 32. AppSuit 라이브러리 적용(1차)한 IPA의 클래스 리스트 저장

3. \_\_objc\_class\_TtC..{앱 scheme 이름}..{Swift 클래스 이름}을 확인합니다.
4. 동일한 과정으로 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA의 클래스 이름을 확인합니다.

## 5. AppSuit 라이브러리(1차) 적용 IPA와 재서명까지 진행한 IPA의 클래스 이름을 비교합니다.

```
__objc_class__TtC15StealPlateSwift20ResultViewController_class  
__objc_class__TtC15StealPlateSwift4ISelectedFeelingsCollectionViewContentViewController_class  
__objc_class__TtC15StealPlateSwift28RestaurantCollectionViewController_cell_class  
__objc_class__TtC15StealPlateSwift15ChooseViewModel_class  
__objc_class__TtC15StealPlateSwift20chooseViewController_class  
__objc_class__TtC15StealPlateSwift18DependencyInjector_class  
__objc_class__TtC15StealPlateSwift19PopupViewController_class  
__objc_class__TtC15StealPlateSwift24RestaurantRepositoryImpl_class  
__objc_class__TtC15StealPlateSwift18BaseViewController_class  
__objc_class__TtC15StealPlateSwift15DetailViewModel_class  
__objc_class__TtC15StealPlateSwift21FeelingRepositoryImpl_class  
__objc_class__TtC15StealPlateSwift29LoadSelectedRestaurantUseCase_class  
__objc_class__TtC15StealPlateSwift20DetailViewController_class  
__objc_class__TtC15StealPlateSwift25RestaurantsJsonParser_class  
__objc_class__TtC15StealPlateSwift25MainSettingViewController_class  
__objc_class__TtC15StealPlateSwift21LoadRestaurantUseCase_class  
__objc_class__TtC15StealPlateSwift24ChooseCollectionViewCell_class  
__objc_class__TtC15StealPlateSwift24MainNavigationController_class  
__objc_class__TtC15StealPlateSwift26RestaurantRemoteDataSource_class  
__objc_class__TtC15StealPlateSwift11AppDelegate_class  
__objc_class__TtC15StealPlateSwift27LoadSelectedFeelingsUseCase_class
```

그림 33. AppSuit 라이브러리(1차) 적용 IPA 클래스 이름

```
__objc_class__TtC15StealPlateSwift20ResultViewController_class  
__objc_class__TtC15StealPlateSwift4ICertificateFactorySpiFileLockDesktopIcon_U_class  
__objc_class__TtC15StealPlateSwift28EventExceptionLocateRegistry_class  
__objc_class__TtC15StealPlateSwift15NameValuePairSe_class  
__objc_class__TtC15StealPlateSwift20ChooseViewController_class  
__objc_class__TtC15StealPlateSwift18StateEditSSLContext_class  
__objc_class__TtC15StealPlateSwift19PopupViewController_class  
__objc_class__TtC15StealPlateSwift24ClosedWatchServiceExcept_class  
__objc_class__TtC15StealPlateSwift18BaseViewController_class  
__objc_class__TtC15StealPlateSwift15ImageOutputStre_class  
__objc_class__TtC15StealPlateSwift21PermissionUnsupported_class  
__objc_class__TtC15StealPlateSwift29CharsetProviderPoint2DDynStru_class  
__objc_class__TtC15StealPlateSwift20DetailViewController_class  
__objc_class__TtC15StealPlateSwift25DataOutputStreamThreadPoo_class  
__objc_class__TtC15StealPlateSwift25MainSettingViewController_class  
__objc_class__TtC15StealPlateSwift21AbstractTypeVisitor7S_class  
__objc_class__TtC15StealPlateSwift24HTMLEditorKitParserStart_class  
__objc_class__TtC15StealPlateSwift24AttributedCharacterItera_class  
__objc_class__TtC15StealPlateSwift26InvalidKeyExceptionJIntern_class  
__objc_class__TtC15StealPlateSwift11AppDelegate_class  
__objc_class__TtC15StealPlateSwift27RuleBasedCollatorSerialSwif_class
```

그림 34. 재서명단계까지 진행 완료된 IPA 클래스 이름

- 일부 클래스에 대해 난독화가 진행되지 않는 이유

- iOS 앱 내에서 기본적으로 사용되는 시스템 클래스들(NS···, objc.. 등)과 delegate 관련 클래스 및 UI 관련 클래스들의 경우 난독화 시 앱 런타임에 문제가 생겨 AppSuit 내부 로직에 의해 자동으로 예외처리 됩니다.

## f. Dynamic API Hiding 기능 검증

- 검증 환경

- Hopper Disassembler
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
  - AppSuit cloud-iwcm (2차) 빌드 시 Dynamic API Hiding 옵션이 활성화 되어 있어야합니다.
- Dynamic API Hiding 옵션 미적용 IPA

- Dynamic API Hiding 기능 검증 시나리오

1. AppSuit Cloud 빌드 페이지의 빌드 로그에서 "Download Log"를 클릭하여 "dump.log" 파일을 준비합니다.

```
...
[DUMP] ---- Target Sub Address ---- [ 0x10052b314 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052ac48 | 0x10052b314 ]
[DUMP] ---- SL Symbol Ptr Address Offset ---- [ 0x7a4ec ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052adc0 | 0x10052b314 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052af14 | 0x10052b314 ]

[DUMP] ---- Target Sub Address ---- [ 0x10052b318 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052ab9c | 0x10052b318 ]
[DUMP] ---- SL Symbol Ptr Address Offset ---- [ 0x1150d8 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052ac28 | 0x10052b318 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052ad14 | 0x10052b318 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052ada0 | 0x10052b318 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052aeb8 | 0x10052b318 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052aef0 | 0x10052b318 ]
[DUMP] ---- Manipulate BL Instruction ---- [ 0x10052af00 | 0x10052b318 ]

...
```

그림 35. Dynamic API Hiding dump 로그 일부

- Target Sub Address : Dynamic API Hiding 이 적용될 함수의 원본 주소값
  - Manipulate BL Instruction : 해당 함수로 분기하는 BL instruction
2. "dump.log" 파일의 "Dynamic API Hiding" 항목에서 검증하고 싶은 "Target Sub Address" 주소를 복사합니다.
  3. Dynamic API Hiding 이 적용 된 IPA의 메인 바이너리 파일을 Hopper Disassembler 프로그램에서 실행합니다.
  4. Hopper Disassembler에서 IPA 파일을 불러온 후, 키보드의 영어 'g' 키를 눌러 복사한 address 값을 입력합니다.
  5. Dynamic API Hiding 미적용 IPA에 대해 동일하게 진행합니다.

6. Dynamic API Hiding 미적용 앱과 적용 앱을 비교하여 Dynamic API Hiding이 적용된 함수를 호출하는 명령어의 주소값을 식별하지 못하는 것을 확인합니다.

```
; ===== BEGINNING OF PROCEDURE =====
; Variables:
; var_10: -16
; var_20: -32
; var_30: -48
; var_38: int64_t, -56
; var_40: -64

sub_100529070:
    sub    $sp, $sp, #0x40          ; Begin of unwind block (FDE at 0x1006ebf1c), CODE XREF=sub_10000e514+28, sub_10000e578+28, sub_10004dc0+28, sub_10004dd94+28, sub_10004efb0+32
    stp    x22, x21, [$sp, #0x10]
    stp    x20, x19, [$sp, #0x20]
    stp    fp, lr, [$sp, #0x30]
    mov    x19, x3

0000000100529074
0000000100529078
000000010052907C
0000000100529080
```

그림 36. Dynamic API Hiding 미적용

```
; ===== BEGINNING OF PROCEDURE =====
; Variables:
; var_10: -16
; var_20: -32
; var_30: -48
; var_38: int64_t, -56
; var_40: -64

sub_100529070:
    sub    $sp, $sp, #0x40          ; Begin of unwind block (FDE at 0x1006ebf1c)
    stp    x22, x21, [$sp, #0x10]
    stp    x20, x19, [$sp, #0x20]
    stp    fp, lr, [$sp, #0x30]
    mov    x19, x3
```

그림 37. Dynamic API Hiding 적용

7. Hopper Disassembler에서 해당 함수의 주소값에서 마우스 우클릭하여 'References to [함수 주소]'를 눌러 해당 함수를 호출하는 분기문 리스트를 확인합니다.

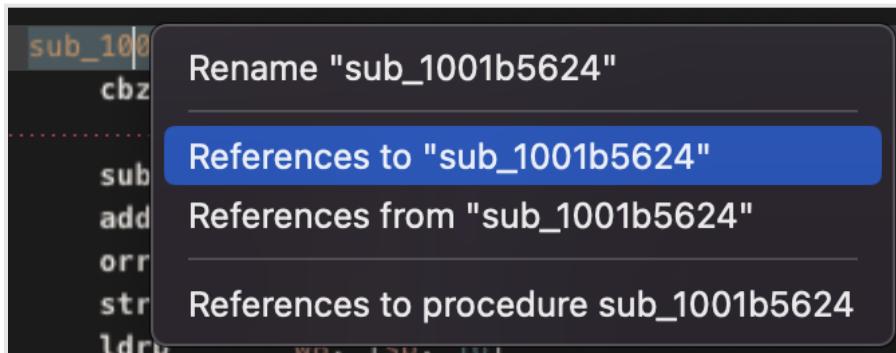


그림 38. Hopper Disassembler에서 함수 참조 확인

References to 0x100529070	
Address	Value
0x10000e530 (sub_10000e514 + 0x1c)	bl sub_100529070
0x10000e594 (sub_10000e578 + 0x1c)	bl sub_100529070
0x10004dc0 (sub_10004dc0 + 0x1c)	bl sub_100529070
0x10004dd0 (sub_10004dd0 + 0x1c)	bl sub_100529070
0x10004efd0 (sub_10004efd0 + 0x20)	bl sub_100529070
0x10007bb08 (sub_10007ba28 + 0xe0)	bl sub_100529070
0x10007c660 (sub_10007c5a8 + 0xb8)	bl sub_100529070
0x1000f9758 (sub_1000f95e0 + 0x178)	bl sub_100529070
0x10010192c (sub_100101910 + 0x1c)	bl sub_100529070

그림 39. Dynamic API Hiding 미적용

References to 0x100529070	
Address	Value

그림 40. Dynamic API Hiding 적용

8. "dump.log" 파일의 "Dynamic API Hiding" 항목에서 검증하고 싶은 "Manipulate BL Instruction" 주소를 복사합니다.
9. AppSuit Cloud 미적용 IPA의 메인 바이너리 파일을 Hopper Disassembler 프로그램에서 실행합니다.
10. Hopper Disassembler에서 IPA 파일을 불러온 후, 키보드의 영어 'g' 키를 눌러 복사한 address 값을 입력합니다.
11. AppSuit Cloud 적용 IPA에 대해 동일하게 진행합니다.
12. AppSuit Cloud 미적용 앱과 적용 앱을 비교하여 해당 명령어가 호출하고 있었던 함수를 식별하지 못하지 못하는 것을 확인합니다.

```

0000000100521b90    mov      w0, #0x2          ; argument #1 for method sub_100529070
0000000100521b94    mov      w1, #0xd          ; argument #2 for method sub_100529070
0000000100521b98    mov      w2, #0x0          ; argument #3 for method sub_100529070
0000000100521b9c    mov      w3, #0x0          ; argument #4 for method sub_100529070
0000000100521ba0    bl       sub_100529070      ; sub_100529070

```

그림 41. Dynamic API Hiding 미적용 BL Instruction

```

0000000100521b90    mov      w0, #0x2          ; argument #1 for method sub_100573968
0000000100521b94    mov      w1, #0xd          ; argument #2 for method sub_100573968
0000000100521b98    mov      w2, #0x0          ; argument #3 for method sub_100573968
0000000100521b9c    mov      w3, #0x0          ; argument #4 for method sub_100573968
0000000100521ba0    bl       sub_100573968      ; sub_100573968

```

그림 42. Dynamic API Hiding 적용 BL Instruction

13. 해당 명령어를 더블 클릭하여 해당 명령어가 호출하고 있는 함수를 확인합니다.
  - Dynamic API Hiding 미적용 시 BL Instruction은 원본 함수 주소로 분기하고 있는 것을 확인합니다.
  - Dynamic API Hiding 적용 시 BL Instruction이 분기하는 원본 함수를 식별할 수 없는 것을 확인합니다.

<pre> ; ===== BEGINNING OF PROCEDURE ===== ; Variables: ; var_18: -16 ; var_28: -32 ; var_38: -48 ; var_38: int64_t, -48 ; var_48: -64  sub_100529070:     sub    sp, sp, #0x40          ; Begin of unwind block (FDE at 0x1006e0)     stp    x22, x21, [sp, #0x10]     stp    x20, x19, [sp, #0x20]     stp    fp, lr, [sp, #0x30]     mov    x19, x3     mov    x20, x2     mov    x21, x1     mov    x22, x0     adrp   x8, #0x1006ed000     ldr    x8, [x8, #0x7c8]        ; 0x1006ed7c@PAGE0     ldr    x8, [x8]                ; 0x1006ed7c@PAGE0FF, __stack_chk_guard     str    x8, [sp, #0x0 + var_38]     adrp   x8, #0x100902000     ldr    x8, [x8, #0x3f8]        ; 0x1009023f@PAGE0     cmn    x8, #0x1     b.ne   loc_1005290f4 </pre>	<pre> ; ===== BEGINNING OF PROCEDURE ===== sub_100573968:     adr    x9, #0x100573968      ; CODE XREF=sub_10088c514+28, sub_10088     adrp   x11, #0x100882800     ldr    x12, [x11, #0x718]      ; qword_100882718     b     0x100553b40     endp </pre>
---	--

그림 43. Dynamic API Hiding 미적용 시

그림 44. Dynamic API Hiding 적용 시

14. Hopper Disassembler에서 우측 상단 메뉴를 통해 Pseudo-code로 전환합니다.



그림 45. Pseudo-code로 전환

15. AppSuit Cloud 미적용 앱과 적용 앱에 대한 Pseudo-code를 비교합니다.

```
int sub_1002005e0(int arg0, int arg1, int arg2, int arg3) {
    r2 = arg2;
    r0 = arg0;
    r31 = r31 - 0x40;
    var_30 = r22;
    stack[-40] = r21;
    var_10 = r20;
    stack[-24] = r19;
    var_10 = r20;
    stack[-8] = r30;
    r19 = arg3;
    r20 = r2;
    r21 = arg1;
    r22 = r0;
    if (*qword_100499b18 == -0x1) {
        if (*qword_100499b20 != 0x0) {
            var_38 = __stack_chk_guard;
            asm { bfi w9, w20, #0x8, #0x8 };
            asm { bfxil w9, w19, #0x0, #0x8 };
            r0 = qword_100499b20[0x1, &var_-40, r2];
            if (**__stack_chk_guard != var_38) {
                r0 = __stack_chk_fail();
            }
        }
    }
}
```

그림 46. Dynamic API Hiding 미적용 시 Pesudo-Code

```
void sub_10023edb4(int arg0, int arg1, int arg2, int arg3, int arg4, int arg5, int arg6, int arg7) {
    loc_100223dc0(arg0, arg1, arg2, arg3, arg4, arg5, arg6, arg7);
    return;
}
```

그림 47. Dynamic API Hiding 적용 시 Pesudo-Code

16. Hopper Disassembler에서 우측 상단 메뉴를 통해 Control Flow Graph로 전환합니다.

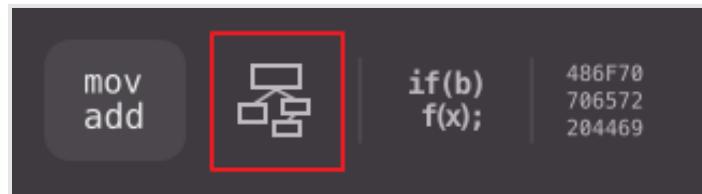


그림 48. Control Flow Graph로 전환

17. AppSuit Cloud 미적용 앱과 적용 앱에 대한 Control Flow Graph 를 비교합니다.

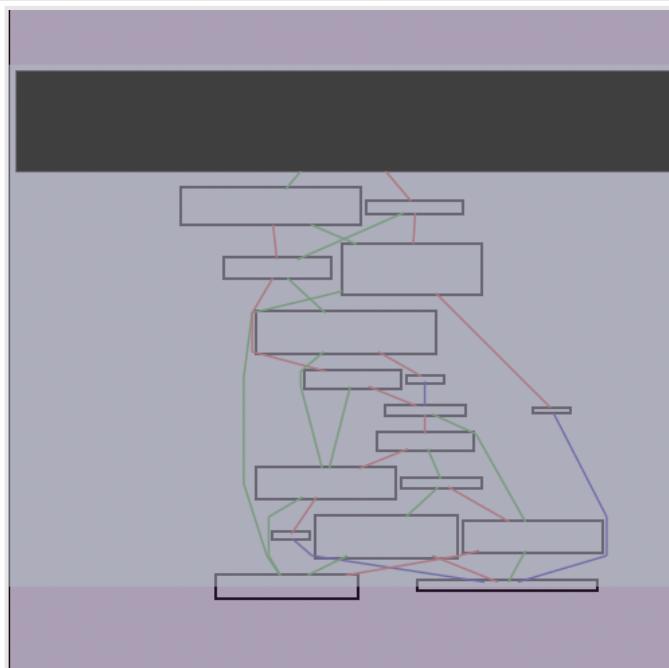


그림 49. Dynamic API Hiding 미적용 시 Control Flow Graph

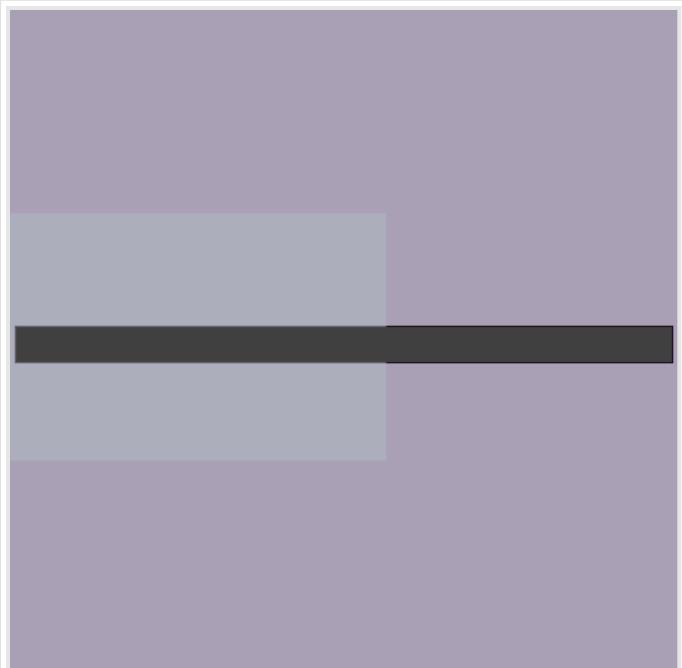


그림 50. Dynamic API Hiding 적용 시 Control Flow Graph

- 일부 함수에 대해 Dynamic API Hiding 이 적용되지 않는 이유
  - 앱을 실행하기 위한 Initializing 관련 함수에 Dynamic API Hiding 적용 시 앱 실행이 정상적으로 되지 않기 때문에, Dynamic API Hiding 적용 가능한 함수에 한해서 적용됩니다.

## g. 심볼 제거 검증

- 검증 환경

- XMachOViewer
- nm 명령어 사용 가능한 PC
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
- AppSuit Cloud 미적용 IPA

- 심볼 제거 시나리오(Using. XMachOViewer)

- AppSuit Cloud 미적용 IPA 메인 바이너리 파일을 XMachOViewer 프로그램에서 실행합니다.
- 주석 > LC\_DYSYMTAB > 간접기호에서 심볼 정보를 확인합니다.

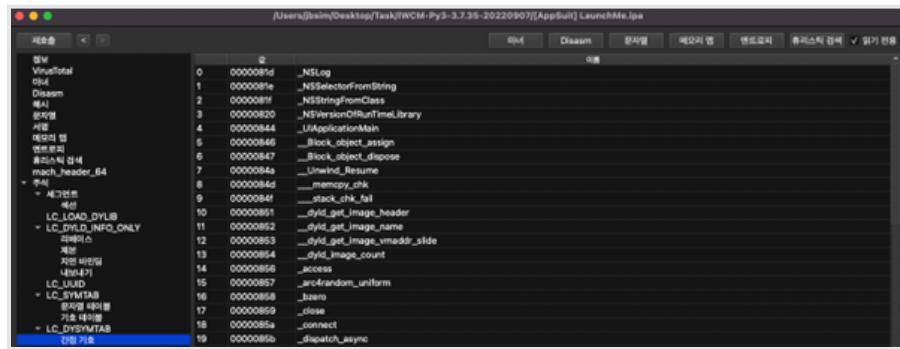


그림 51. AppSuit 미적용 앱 심볼 제거 검증

- AppSuit Cloud 적용 IPA 메인 바이너리 파일을 XMachOViewer 프로그램에서 실행합니다.
- 주석 > LC\_DYSYMTAB > 간접기호에서 심볼 정보를 확인합니다.

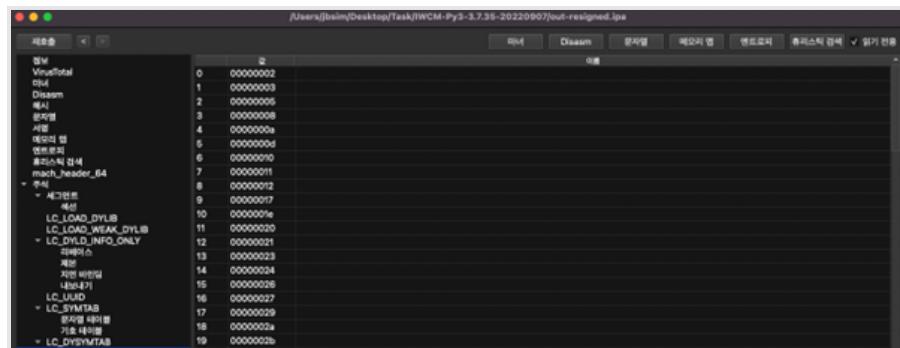


그림 52. AppSuit 적용 앱 심볼 제거 검증

- AppSuit Cloud 미적용 앱과 적용 앱을 비교하여 심볼 정보가 제거된 것을 확인합니다.

- 심볼 제거 검증 시나리오(Using. nm)

1. 아래 명령어를 입력하여 AppSuit Cloud 미적용 IPA 메인 바이너리 파일에 대한 심볼 정보를 확인합니다.

```
nm -a {바이너리 파일 경로}
```

```
jbsim@simjaebinuiMac ~/Documents> nm 1차 \ 적용 _LaunchMe
000000001000101ac t +[Diresu f]
000000001000101bc t +[Diresu free]
000000001000101b4 t +[Diresu g]
000000001000101a4 t +[Kartzelza callipygian]
0000000010019019t t +[StockKNewsdmManager bottomMostViewController]
00000000100161b0t t +[StockKNewsdmManager defRandomString:]
0000000010015e3e4t t +[StockKNewsdmManager defRandomString]
000000001001a2374t t +[StockKNewsdmManager errorForFailedLoginWithCode2]
0000000010019a46ct t +[StockKNewsdmManager errorForFailedLoginWithCode]
000000001001688b8t t +[StockKNewsdmManager errorMessageDomain:]
000000001001aa260t t +[StockKNewsdmManager fetchCachedProfile:]
0000000010016e394t t +[StockKNewsdmManager generateSerialNumber:ErrorDescriptionOutput:]
0000000010015e294t t +[StockKNewsdmManager loadAllMMOCache]
0000000010015e390t t +[StockKNewsdmManager loadCustomYield]
0000000010015cef8t t +[StockKNewsdmManager loadProfileWithCache]
000000001001b54cc t +[StockKNewsdmManager loadProfileWithCompletion:]
0000000010015cef4t t +[StockKNewsdmManager load]
00000000100178e60t t +[StockKNewsdmManager postServerMeta:]
000000001001721cc t +[StockKNewsdmManager postServerMeta]
0000000010018692ct t +[StockKNewsdmManager singleShotLogRequest]
00000000100189f00t t +[StockKNewsdmManager topMostViewController]
000000001001adbdc t +[StockKNewsdmManager unloadProfileWithCompletion:]
00000000100165220t t +[StockKNewsdmManager viewControllerForUserView]
00000000100196774t t +[StockKNewsdmManager viewControllerForView:]
0000000010017fb0t t +[StockKNewsdmManager willCircleProfile:]
0000000010018327ct t +[StockKNewsdmManager willCircleProfile:indexNumber:]
000000001001b791c t -[AppPushAlertno .cxx_destruct]
000000001001b6ef8t t -[AppPushAlertno application:didFinishLaunchingWithOptions:]
000000001001b7988t t -[AppPushAlertno rootViewController]
000000001001b7910t -[AppPushAlertno setRootViewController:]
000000001001b78fc t -[AppPushAlertno setWindow:]
000000001001b78f4t -[AppPushAlertno window]
0000000010000e1f4t -[CertificateForceManagerl alertView:clickedButtonAtIndex:]
0000000010000e1c0t -[CertificateForceManagerl didReceiveMemoryWarning]
```

그림 53. AppSuit 미적용 앱 심볼 정보 확인

2. 동일하게 명령어를 입력하여 AppSuit Cloud 적용 IPA 메인 바이너리 파일에 대한 심볼 정보를 확인합니다.

그림 54. AppSuit 적용 앱 심볼 정보 확인

3. AppSuit Cloud 미적용 앱과 적용 앱을 비교하여 심볼 정보가 제거된 것을 확인합니다.

※ 아래와 같이 AppSuit Cloud 적용 IPA에 대해 LC\_SYMTAB 이 깨져 읽어오지 못하는 경우도 동일하게 심볼 제거된 상태입니다.

```
jbsim@simjaebin-ui-iMac ~ ➔ nm StealPlateSwift_AppSuit  
/Applications/Xcode.app/Contents/Developer/Toolchains/XcodeDefault.xctoolchain/usr/bin/nm: error: S  
tealPlateSwift_AppSuit: truncated or malformed object (stroff field plus strsize field of LC_SYMTAB  
command 6 extends past the end of the file)
```

그림 55. AppSuit 적용 앱 심볼 정보 확인

## h. 로그 데이터 삭제 검증

- 검증 환경
  - Xcode
  - 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
    - AppSuit cloud-iwcm (2차) 빌드 시 Log Hiding 옵션이 활성화 되어 있어야합니다.
  - AppSuit Cloud 미적용 IPA

### • 로그 데이터 삭제 검증 시나리오

1. Xcode Device Console 을 통해 AppSuit Cloud 미적용 IPA에 대한 NSLog 출력 여부를 확인 합니다.

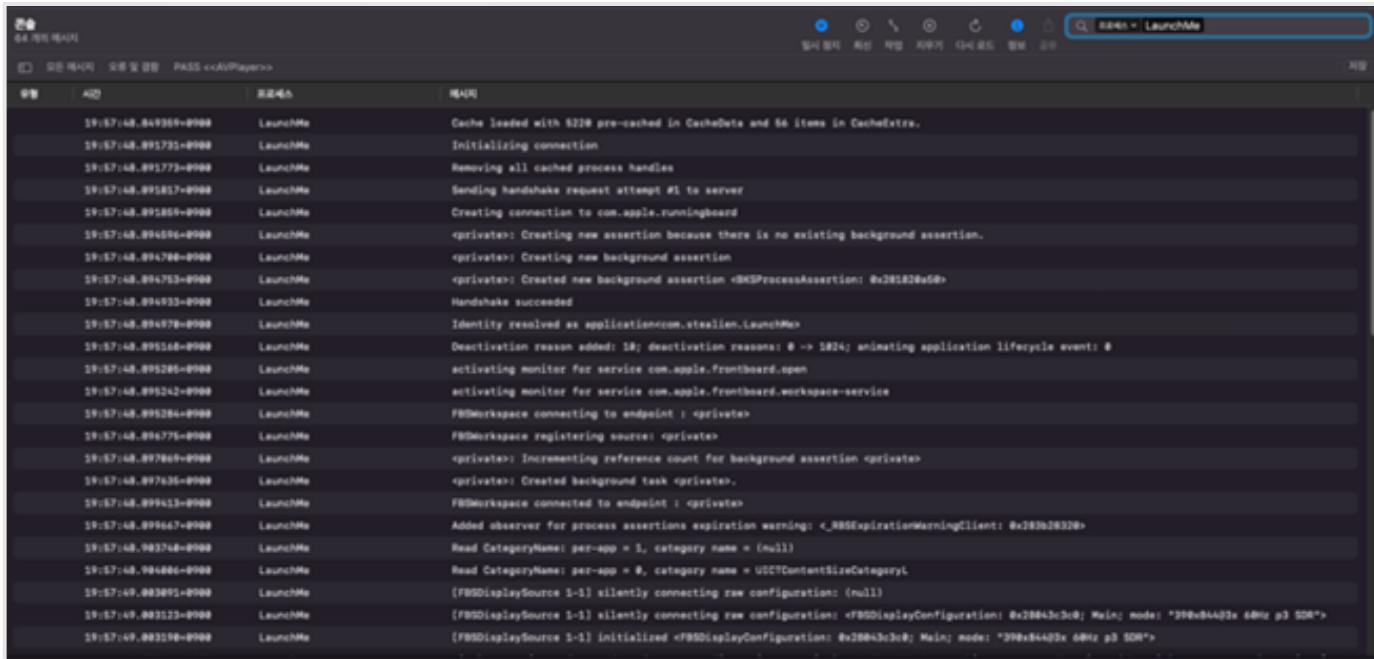


The screenshot shows the Xcode Device Console window. At the top, there are tabs for '콘솔' (Console), '모든 메시지' (All Messages), '모듈 및 결합' (Modules & Bundles), and 'PASS <<AVPlayer>>'. Below the tabs, there are buttons for filtering logs by '유형' (Type), '시간' (Time), and '프로세스' (Process). The main area displays a list of log entries. The log entries are timestamped and show repeated 'NSLog Test' messages from the 'LaunchMe' process. The timestamps range from 16:11:54.532294+0900 to 16:11:54.535847+0900. The log entries are as follows:

날짜	프로세스	메시지
16:11:54.532294+0900	LaunchMe	NSLog Test
16:11:54.532578+0900	LaunchMe	NSLog Test
16:11:54.532636+0900	LaunchMe	NSLog Test
16:11:54.532769+0900	LaunchMe	NSLog Test
16:11:54.532926+0900	LaunchMe	NSLog Test
16:11:54.533029+0900	LaunchMe	NSLog Test
16:11:54.533164+0900	LaunchMe	NSLog Test
16:11:54.533288+0900	LaunchMe	NSLog Test
16:11:54.533389+0900	LaunchMe	NSLog Test
16:11:54.533409+0900	LaunchMe	NSLog Test
16:11:54.533509+0900	LaunchMe	NSLog Test
16:11:54.534161+0900	LaunchMe	NSLog Test
16:11:54.534269+0900	LaunchMe	NSLog Test
16:11:54.534257+0900	LaunchMe	NSLog Test
16:11:54.534319+0900	LaunchMe	NSLog Test
16:11:54.534693+0900	LaunchMe	NSLog Test
16:11:54.534945+0900	LaunchMe	NSLog Test
16:11:54.534998+0900	LaunchMe	NSLog Test
16:11:54.535847+0900	LaunchMe	NSLog Test

그림 56. AppSuit Cloud 미적용 앱 NSLog 출력 확인

## 2. Xcode Device Console을 통해 AppSuit Cloud 적용 IPA에 대한 NSLog 출력 여부를 확인 합니다.



The screenshot shows the Xcode Device Console window titled 'LaunchMe'. The console displays a log of messages from the 'LaunchMe' application. The log includes various initialization steps, connection attempts, assertion creation, and workspace connectivity details. The log entries are timestamped and show the process name 'LaunchMe' throughout.

항목	시간	프로세스	메시지
19:57:48.8949369~0900	LaunchMe	Cache loaded with 5220 pre-cashed in CacheData and 66 items in CacheExtra.	
19:57:48.891731~0900	LaunchMe	Initializing connection	
19:57:48.891773~0900	LaunchMe	Removing all cached process handles	
19:57:48.891857~0900	LaunchMe	Sending handshake request attempt #1 to server	
19:57:48.891859~0900	LaunchMe	Creating connection to com.apple.runningboard	
19:57:48.894896~0900	LaunchMe	<private>: Creating new assertion because there is no existing background assertion.	
19:57:48.891780~0900	LaunchMe	<private>: Creating new background assertion	
19:57:48.891783~0900	LaunchMe	<private>: Created new background assertion <NSProcessAssertion: 0x281820a50>	
19:57:48.894923~0900	LaunchMe	Handshake succeeded	
19:57:48.894978~0900	LaunchMe	Identity resolved as application<com.stealien.LaunchMe>	
19:57:48.895168~0900	LaunchMe	Deactivation reason added: 10; deactivation reason: 8 -> 1024; animating application lifecycle event: #	
19:57:48.895265~0900	LaunchMe	activating monitor for service com.apple.frontboard.open	
19:57:48.895242~0900	LaunchMe	activating monitor for service com.apple.frontboard.workspace-service	
19:57:48.895284~0900	LaunchMe	FBDWorkspace connecting to endpoint : <private>	
19:57:48.894975~0900	LaunchMe	FBDWorkspace registering source: <private>	
19:57:48.897869~0900	LaunchMe	<private>: Incrementing reference count for background assertion <private>	
19:57:48.897636~0900	LaunchMe	<private>: Created background task <private>.	
19:57:48.899413~0900	LaunchMe	FBDWorkspace connected to endpoint : <private>	
19:57:48.899667~0900	LaunchMe	Added observer for process assertions expiration warning: <_NSSEExpirationWarningClient: 0x2830e28320>	
19:57:48.981748~0900	LaunchMe	Read CategoryName per-app = 1, category name = (null)	
19:57:48.984886~0900	LaunchMe	Read CategoryName: per-app = 0, category name = UDCTContentSizeCategoryL	
19:57:49.002091~0900	LaunchMe	{FBSDisplaySource 1-1} silently connecting raw configuration: (null)	
19:57:49.003123~0900	LaunchMe	{FBSDisplaySource 1-1} silently connecting raw configuration: <FBSDisplayConfiguration: 0x28043c3c0; Main; mode: "390x840@3x 60Hz p3 SDR">	
19:57:49.003190~0900	LaunchMe	{FBSDisplaySource 1-1} initialized <FBSDisplayConfiguration: 0x28043c3c0; Main; mode: "390x840@3x 60Hz p3 SDR">	

그림 57. AppSuit Cloud 적용 앱 NSLog 출력 확인

## 3. AppSuit Cloud 적용 IPA에서 NSLog가 출력되지 않는 것을 확인 합니다.

### • 로그 데이터 삭제 옵션 적용 시 다른 Log가 출력되는 이유

- 로그 데이터 삭제 옵션의 경우 프로젝트 내 NSLog를 사용하여 출력하는 로그에 대해서만 삭제 기능을 제공하고 있습니다.

따라서 앱 자체 시스템 로그 등은 제거되지 않는 점 참고 부탁드립니다.

## i. 탈옥 탐지 검증

- 검증 환경

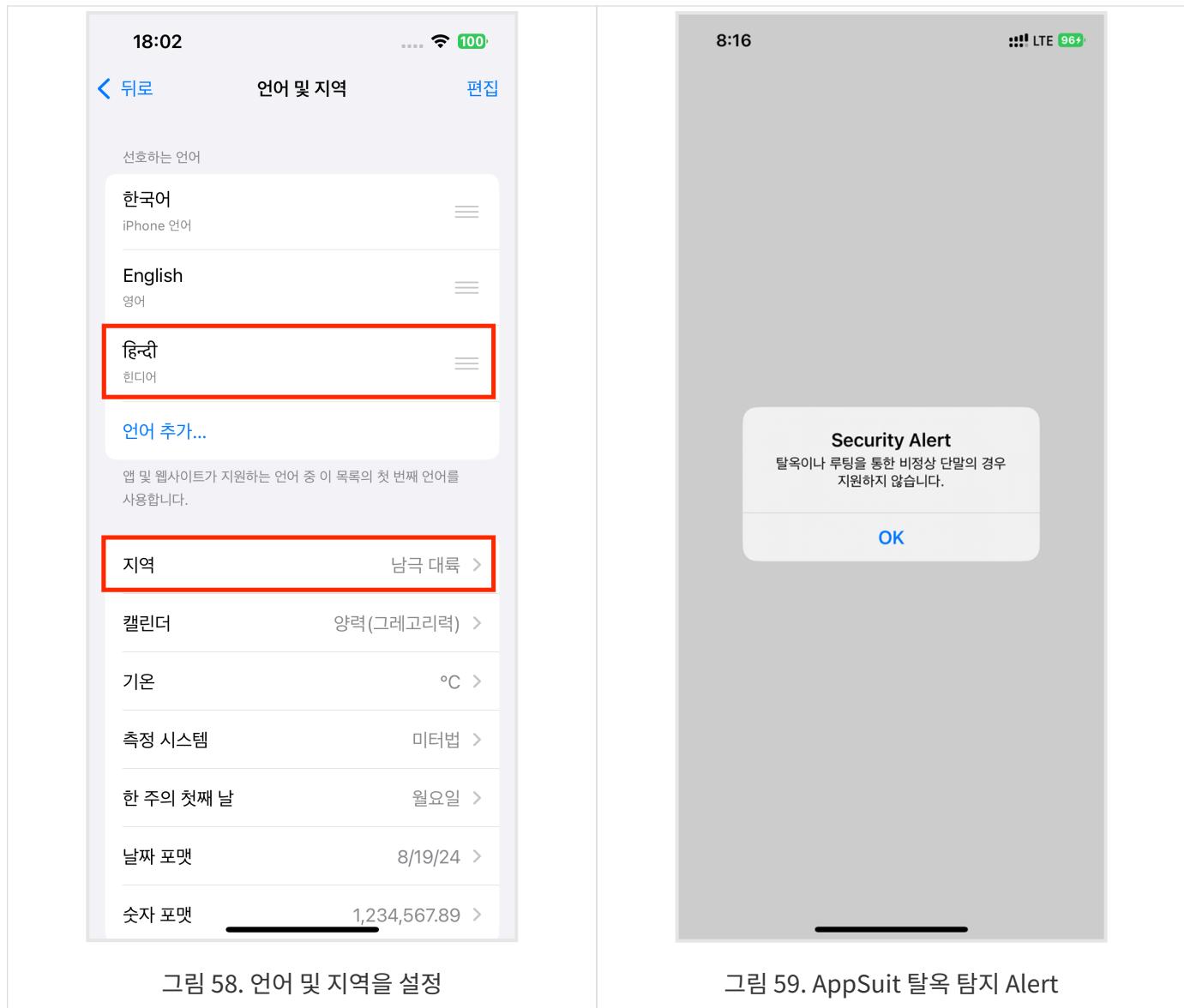
- 정상 디바이스, 탈옥 디바이스
- Xcode 가 설치되어 IPA 설치가 가능한 PC
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
- AppSuit Cloud 미적용 IPA

- 탈옥 탐지 검증 시나리오(For. 탈옥 디바이스)

1. 탈옥 단말기에 AppSuit Cloud 미적용 IPA를 설치합니다.
2. 앱을 실행 하여 정상적으로 실행되는 것을 확인합니다.
3. 설치한 AppSuit Cloud 미적용 IPA를 삭제합니다.
4. 탈옥 단말기에 AppSuit Cloud 적용 IPA를 설치합니다.
5. 앱을 실행 하여 탈옥 탐지 알럿가 뜨는 것을 확인합니다.

• 탈옥 탐지 검증 시나리오(For. 라이브러리 v24.3.0 이상)

1. [단말기 설정 / 일반 / 언어 및 지역] 메뉴에서 선호하는 언어에 **힌디어** 추가, 지역을 **남극 대륙**으로 설정합니다.
2. 단말기에 AppSuit Cloud 미적용 IPA를 설치합니다.
3. 앱을 실행 하여 정상적으로 실행되는 것을 확인합니다.
4. 설치한 AppSuit Cloud 미적용 IPA를 삭제합니다.
5. 단말기에 AppSuit Cloud 적용 IPA를 설치합니다.
6. 앱을 실행 하여 탈옥 탐지 알러트가 뜨는 것을 확인합니다.



- 탈옥 탐지 검증 시나리오(For. 라이브러리 v24.3.0 미만, iOS 17 미만 정상 디바이스)

1. 단말기 설정에서 기기이름을 **JBBB123456789** 또는 **bjbj123456789**로 변경합니다.
2. 단말기에 AppSuit Cloud 미적용 IPA를 설치합니다.
3. 앱을 실행 하여 정상적으로 실행되는 것을 확인합니다.
4. 설치한 AppSuit Cloud 미적용 IPA를 삭제합니다.
5. 단말기에 AppSuit Cloud 적용 IPA를 설치합니다.
6. 앱을 실행 하여 탈옥 탐지 알럿가 뜨는 것을 확인합니다.

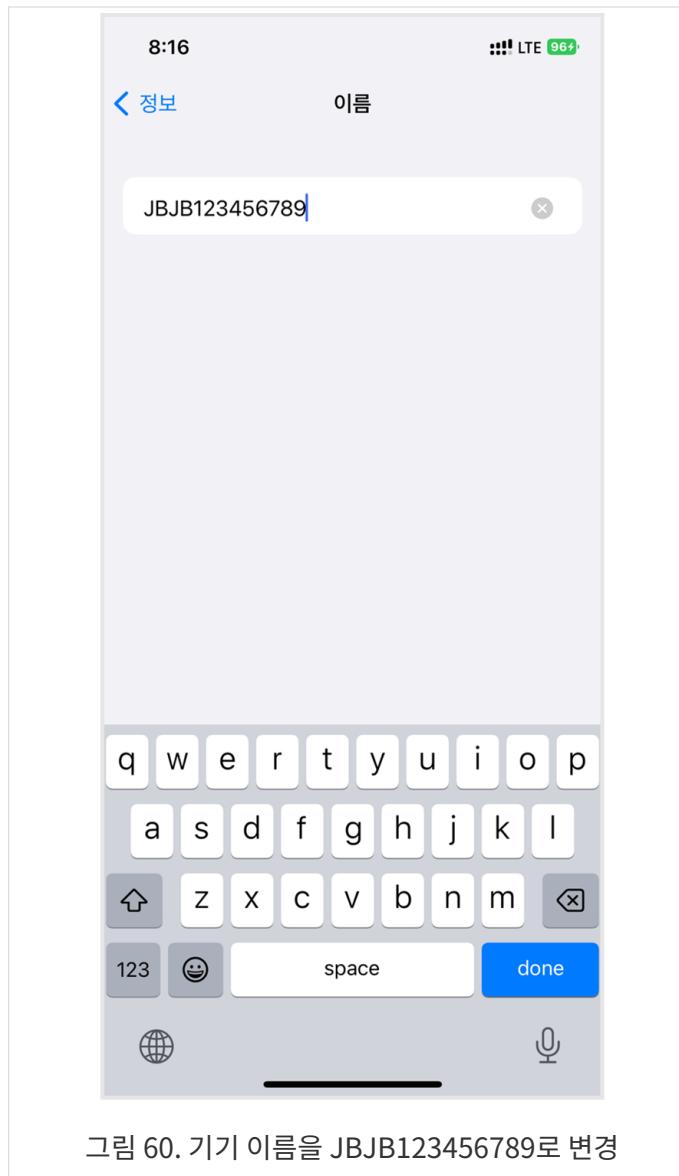


그림 60. 기기 이름을 JBBB123456789로 변경

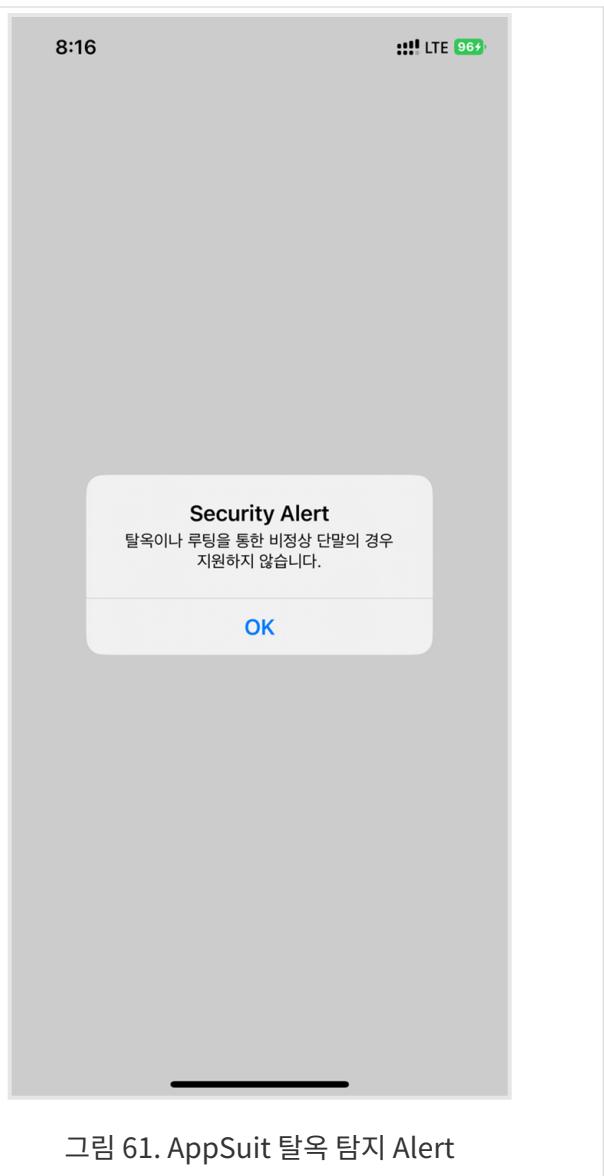
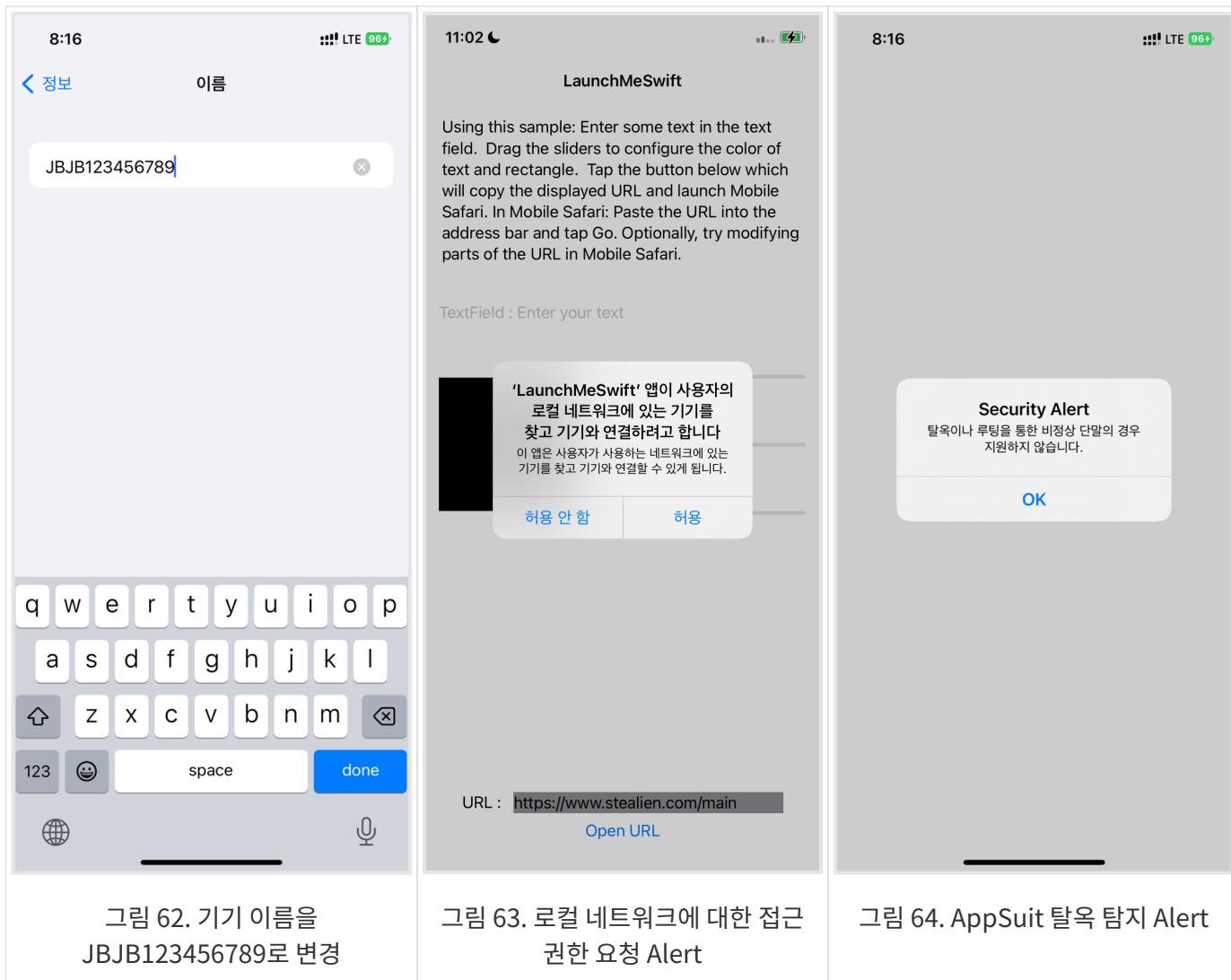


그림 61. AppSuit 탈옥 탐지 Alert

• 탈옥 탐지 검증 시나리오(For. 라이브러리 v24.3.0 미만, iOS 17 정상 디바이스)

1. 단말기 설정에서 기기이름을 **JBJB123456789** 또는 **bjjb123456789**로 변경합니다.
2. 단말기에 AppSuit Cloud 미적용 IPA를 설치합니다.
3. 앱을 실행 하여 정상적으로 실행되는 것을 확인합니다.
4. 설치한 AppSuit Cloud 미적용 IPA를 삭제합니다.
5. AppSuit cloud-iwcm (2차) 빌드 시 "Jailbreak Check Test for iOS 17 and above" 옵션을 활성화 합니다.
6. 단말기에 AppSuit Cloud 적용 IPA를 설치합니다.
7. 앱 최초 실행 시, "로컬 네트워크에 대한 접근 권한"을 요청하는 Alert가 발생하며 허용 버튼을 눌러줍니다.
8. 앱을 종료하고 앱을 다시 실행 하여 탈옥 탐지 알러트가 뜨는 것을 확인합니다.
  - ※ 탈옥 탐지 알러트가 출력되지 않을 경우, 단말기 설정에서 기기이름을 **JBJB123456789** 또는 **bjjb123456789** 외에 다른 이름으로 변경 한 뒤, 다시 **JBJB123456789** 또는 **bjjb123456789** 으로 변경 후 앱을 실행합니다.



## j. 위변조 탐지 검증

- 검증 환경

- (Using. Sideloadly)
  - 정상 디바이스
  - IPA 설치를 위한 Xcode, iOS App Signer, Sideloadly 가 설치 된 PC
    - iOS App Signer 는 다음 <https://www.iosappsigner.com> 에서 다운받을 수 있습니다
    - Sideloadly 는 다음 <https://sideloadly.io> 에서 다운받을 수 있습니다
- (Using. iOS App Signer)
  - 정상 디바이스
  - IPA 설치를 위한 Xcode, iOS App Signer 가 설치 된 PC
- (Using. Cydia & Filza)
  - 탈옥 디바이스
    - Cydia와 Filza 설치가 가능해야합니다.

- 위변조 탐지 검증 시나리오(Using. Sideloadly)

1. AppSuit 라이브러리 적용(1차)하여 Xcode Archive Export 를 진행합니다. 이때 인증서 A 를 사용하여 서명을 진행합니다.

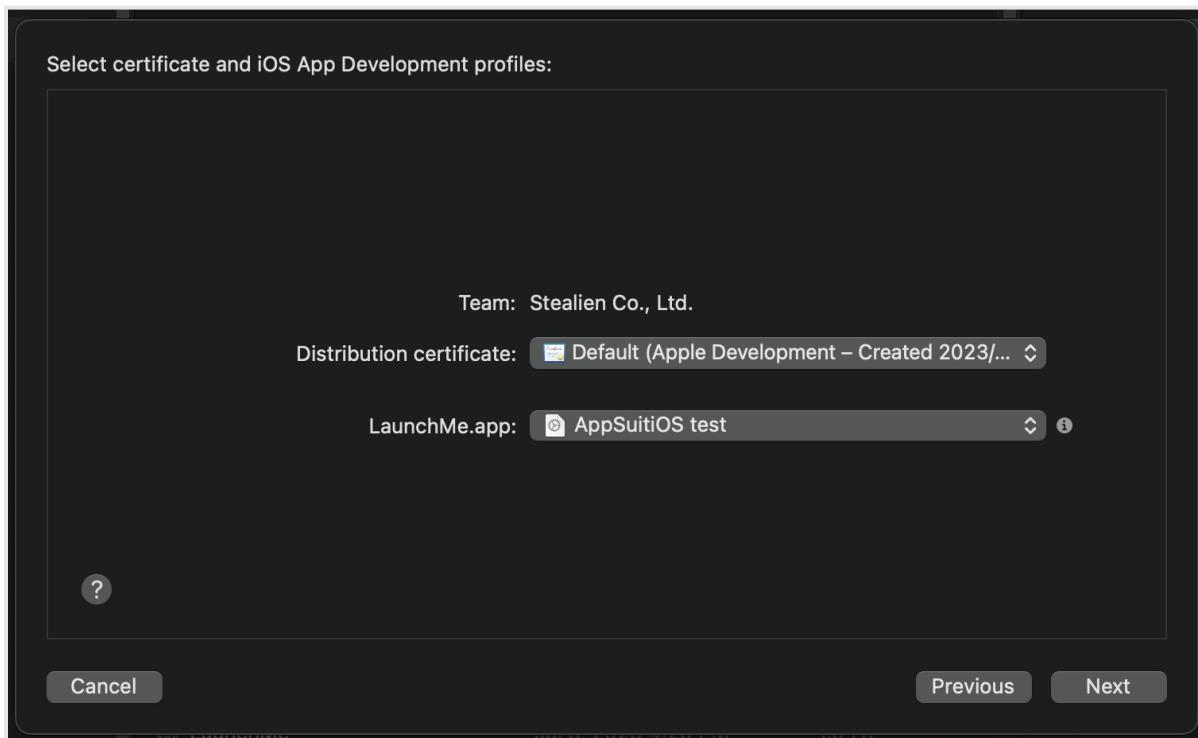


그림 65. 인증서 선택

2. 추출한 IPA로 AppSuit cloud-iwcm 적용(2차)를 진행합니다.
3. iOS App Signer를 이용하여 AppSuit cloud-iwcm 적용(2차)한 IPA를 인증서 B로 서명을 진행합니다.
  - Provisioning Profile 설정은 필요하지 않습니다.

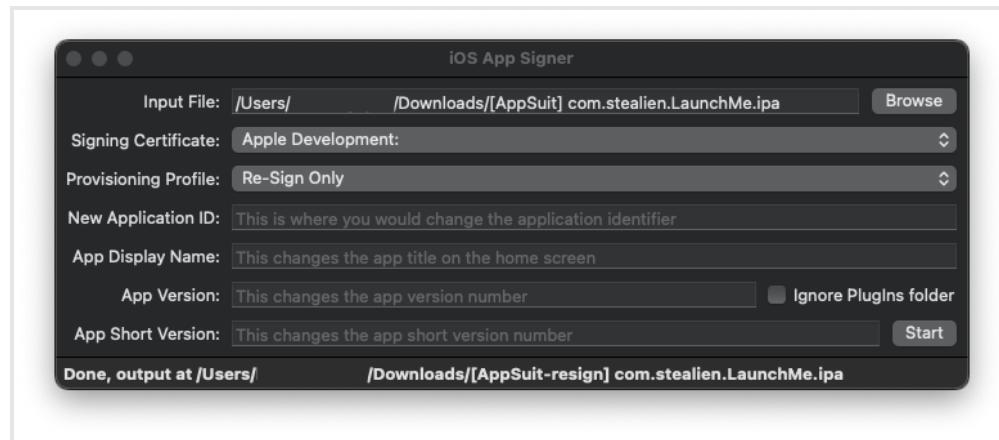


그림 66. iOS App Signer로 재서명

4. 서명 완료한 IPA를 Sideloadly를 통해 디바이스에 iOS App Signer로 서명 진행한 IPA를 설치 후 탐지 여부를 확인합니다.
  - i. Sideloadly에서 설치할 기기와 Apple ID를 설정한 뒤, Start 버튼을 클릭합니다.

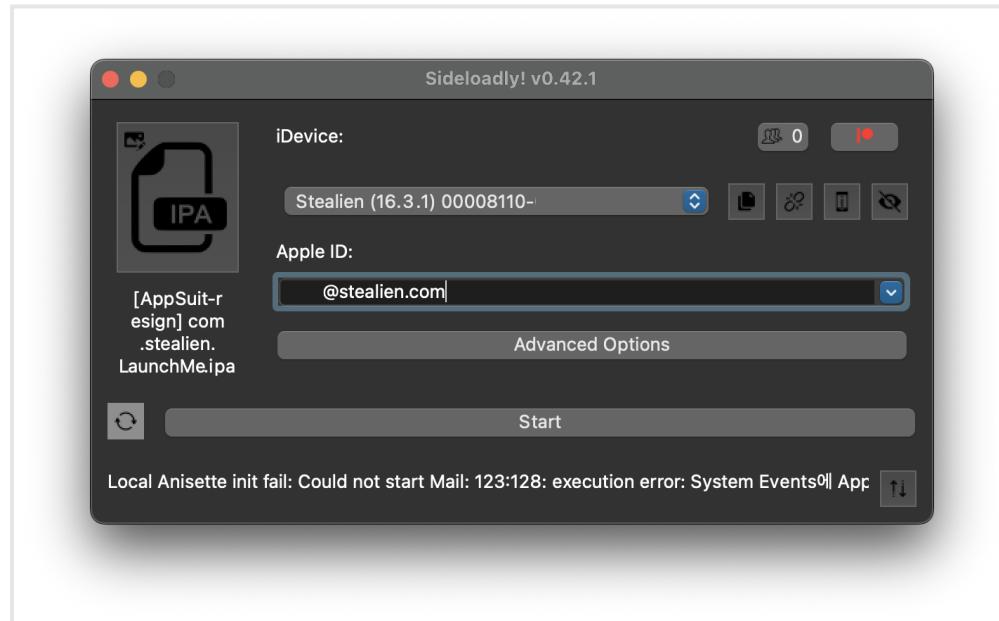


그림 67. Sideloadly로 IPA 설치

ii. 팀 선택 창이 나오면 개인 인증서로 진행합니다.

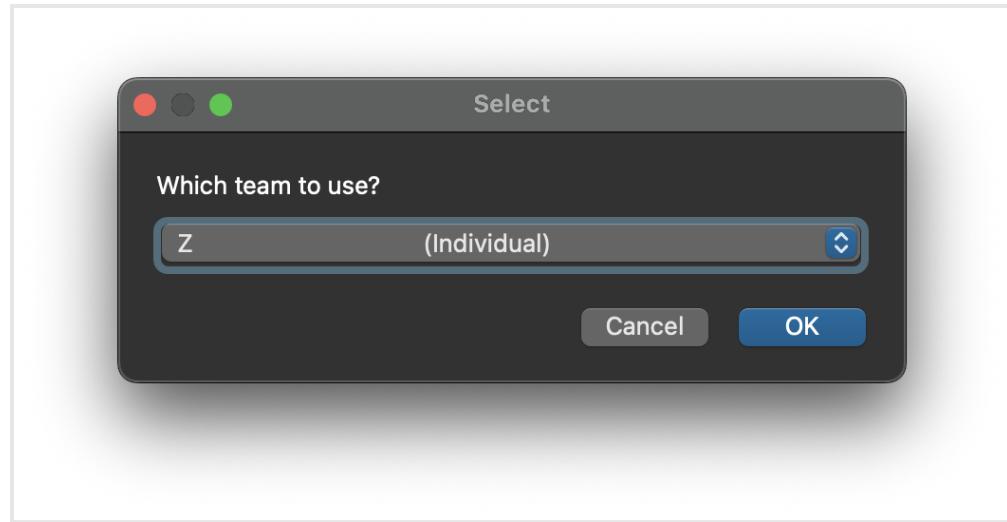


그림 68. 개인 인증서 선택

5. 설치한 기기의 설정 > 일반 > VPN 및 기기관리 > 개발자 앱에서 개발자 신뢰를 클릭합니다.

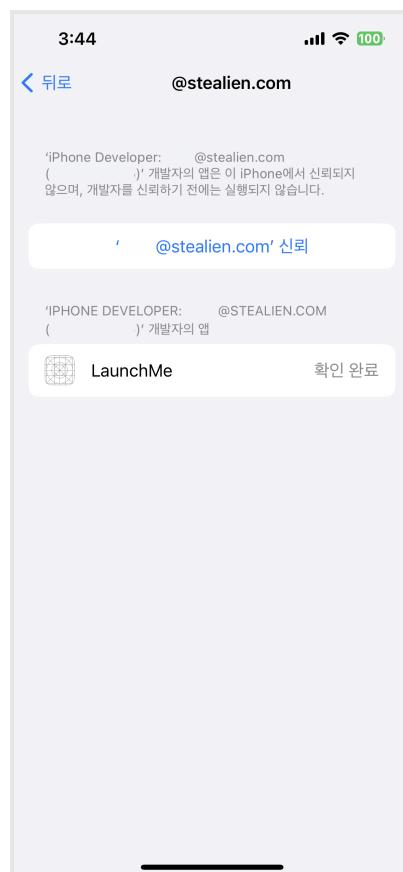


그림 69. 앱 개발자 신뢰

6. 설치한 앱을 실행하여 탐지 여부를 확인합니다.

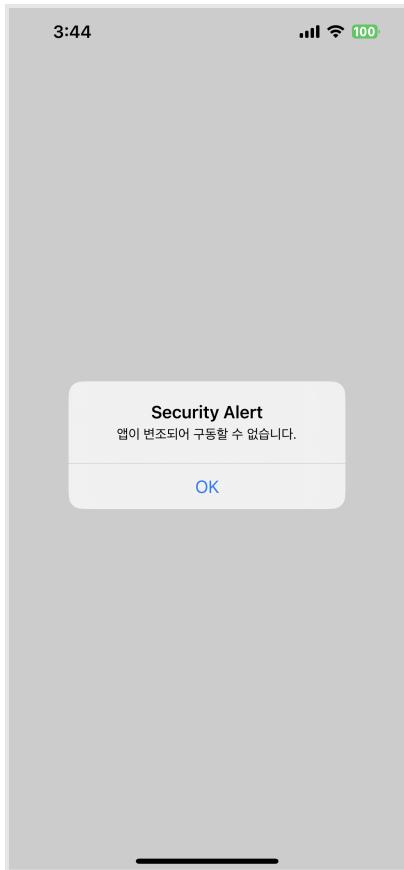


그림 70. AppSuit 위변조 탐지 알러트 확인

- 위변조 탐지 검증 시나리오(Using. iOS App Signer)

- 해당 검증 방법은 Xcode 자체 무결성 검증 통과를 위해 같은 Provisioning Profile에 포함되어 있는 서로 인증서 두 가지를 사용하여 진행됩니다.

1. 사내 Apple 계정으로 Apple Developer 사이트에서 Certificates, Identifiers & Profiles 접속합니다.

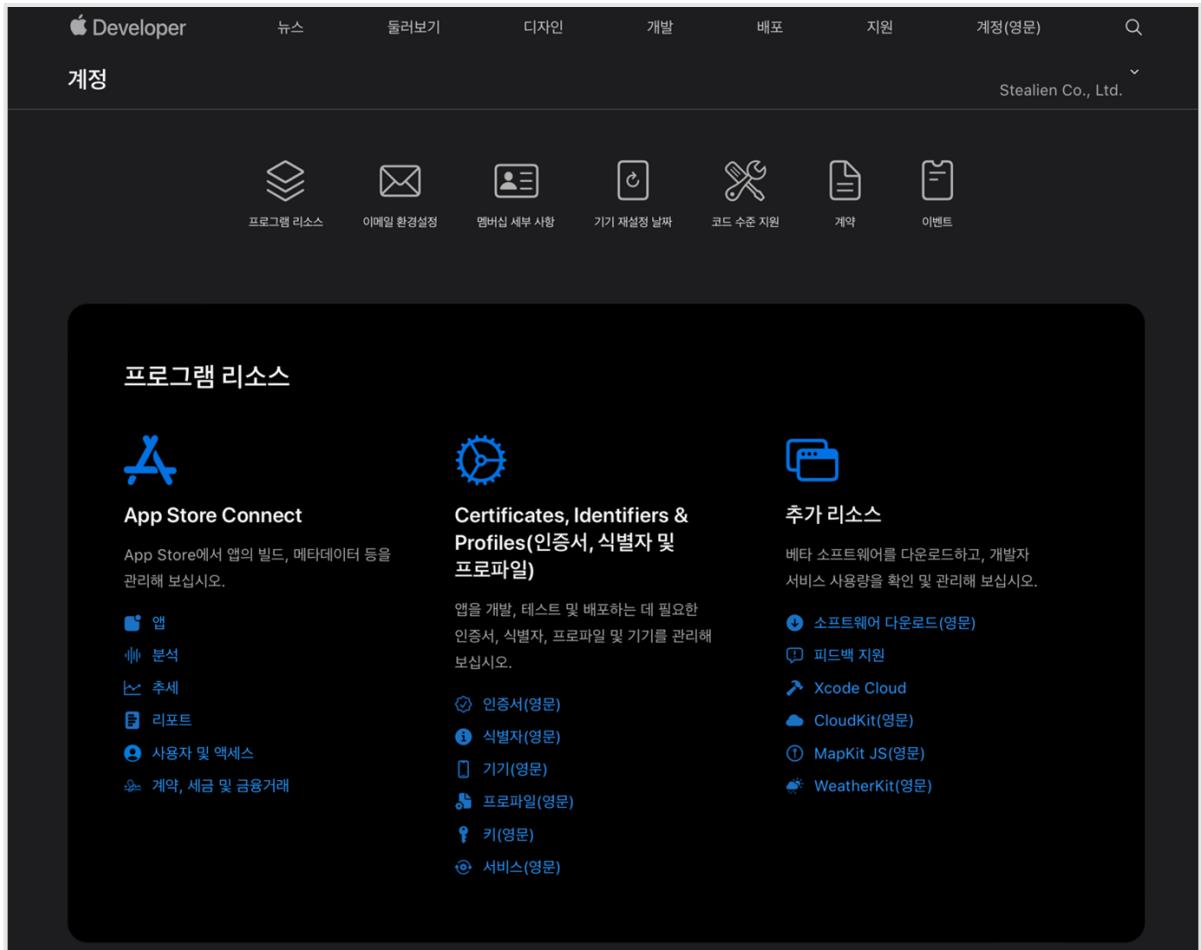


그림 71. Apple Developer 사이트 접속

2. 동일한 Provisioning Profile에 서로 다른 인증서 두 가지(이하 A인증서, B인증서)를 생성합니다.
3. Profiles에서 검증에 사용할 Profile을 클릭합니다.
4. 선택한 Profile에서 Edit를 클릭합니다.

5. 해당 Profile에 생성한 A인증서와 B인증서를 추가합니다.

< All Profiles

## Generate a Provisioning Profile

Save

Name	Status
AppSuitiOS test	Active
Type	Expires
Development	2023/10/04
Created By	Enabled Capabilities
	None

App ID	424 App IDs
XC Wildcard (W2KK64JPPU.*)	X   v

Certificates

Select All      20 of 32 item(s) selected

Alex Jin(Development) For use in Xcode 11 or later  
 (Development) For use in Xcode 11 or later  
 Alex Jin(Development) For use in Xcode 11 or later  
 Alex Jin(Development) For use in Xcode 11 or later  
 Alex Jin(Development) For use in Xcode 11 or later  
 (Development) For use in Xcode 11 or later  
 Alex Jin(Development) For use in Xcode 11 or later

그림 72. 인증서 추가

6. 검증할 PC에서 키체인 접근을 열어 추가한 인증서 두 가지가 키체인에 있는지 확인합니다.

이름	종류	만료	키체인
> iPhone Developer: Alex Jin (63X5UTL66C)	인증서	2023. 2. 16. 오후 3:18:35	로그인
> Developer ID Certification Authority	인증서	2031. 9. 17. 오전 9:00:00	로그인
> Developer ID Application: Stealien Co., Ltd. (W2KK64JPPU)	인증서	2027. 9. 16. 오후 1:07:41	로그인
blog.gokoro.me	인증서	2021. 12. 4. 오전 8:59:59	로그인
Apple Worldwide Developer Relations Certification Authority	인증서	2030. 2. 20. 오전 9:00:00	로그인
Apple Root CA	인증서	2035. 2. 10. 오전 6:40:36	로그인
Apple iPhone OS Provisioning Profile Signing	인증서	2027. 3. 17. 오전 6:29:17	로그인
Apple iPhone Certification Authority	인증서	2030. 12. 31. 오전 9:00:00	로그인
> Apple Distribution: Stealien Co., Ltd. (W2KK64JPPU)	인증서	2023. 7. 26. 오후 5:04:38	로그인
> Apple Development: (LVSKYNNISLU)	인증서	2023. 7. 15. 오후 5:38:08	로그인
> Apple Development: teila@korea.ac.kr (SQKUFP786V)	인증서	2023. 8. 30. 오후 6:22:38	로그인
> Apple Development: jbsim@stealien.com (6N3QZD2FAQ)	인증서	2023. 8. 30. 오후 2:47:48	로그인
> Apple Development: alexjin33@icloud.com (6346M23NXE)	인증서	2023. 8. 8. 오후 5:08:39	로그인

그림 73. 키체인 접근에서 인증서 확인

7. AppSuit 라이브러리 적용(1차)하여 Xcode Archive Export 시 A인증서를 선택합니다.

8. AppSuit cloud-iwcm 적용(2차)을 진행합니다.

9. AppSuit cloud-iwcm 적용(2차)한 IPA를 iOS App Signer를 이용하여 B인증서로 재서명합니다.

- iOS App Signer 사용 시 Provisioning Profile 은 두 가지 인증서 추가한 Profile로 선택합니다.

10. 단말기에 재서명한 IPA를 설치하여 위변조 탐지 여부를 확인합니다.

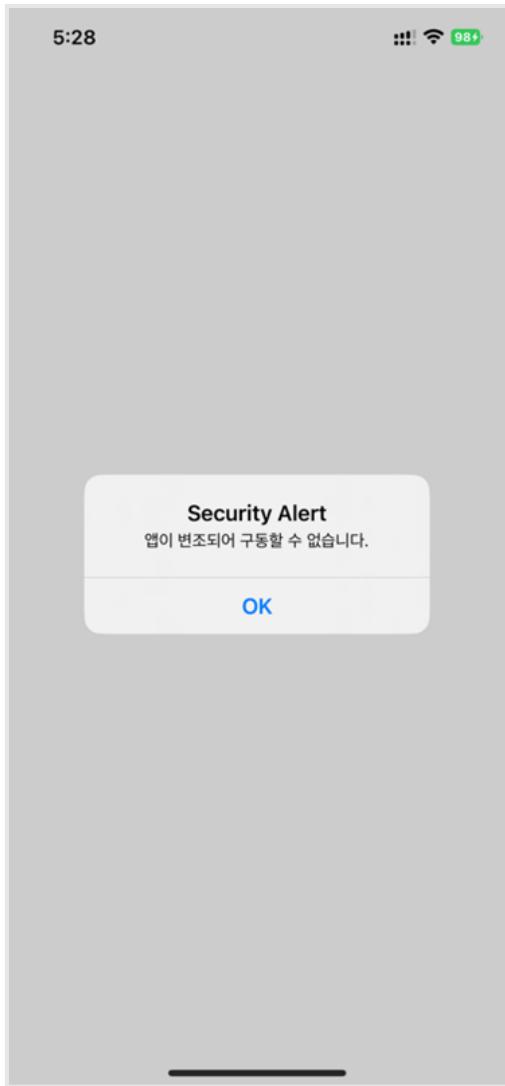


그림 74. AppSuit 위변조 탐지 알러트 확인

- 위변조 탐지 검증 시나리오(Using. Cydia & Filza)

- 탈옥 단말기에 설치되어 있는 Frida-Server를 제거해야 진행 가능합니다.

1. 탈옥 단말기에서 AppSuit Cloud 적용 앱을 실행하기 위해 AppSuit cloud-iwcm (2차) 빌드 시 탈옥 탐지와 위협 탐지 옵션을 제외합니다.  
("Jailbreak Check", "Threat Check", "Flex 3 Check" 옵션을 해제합니다.)

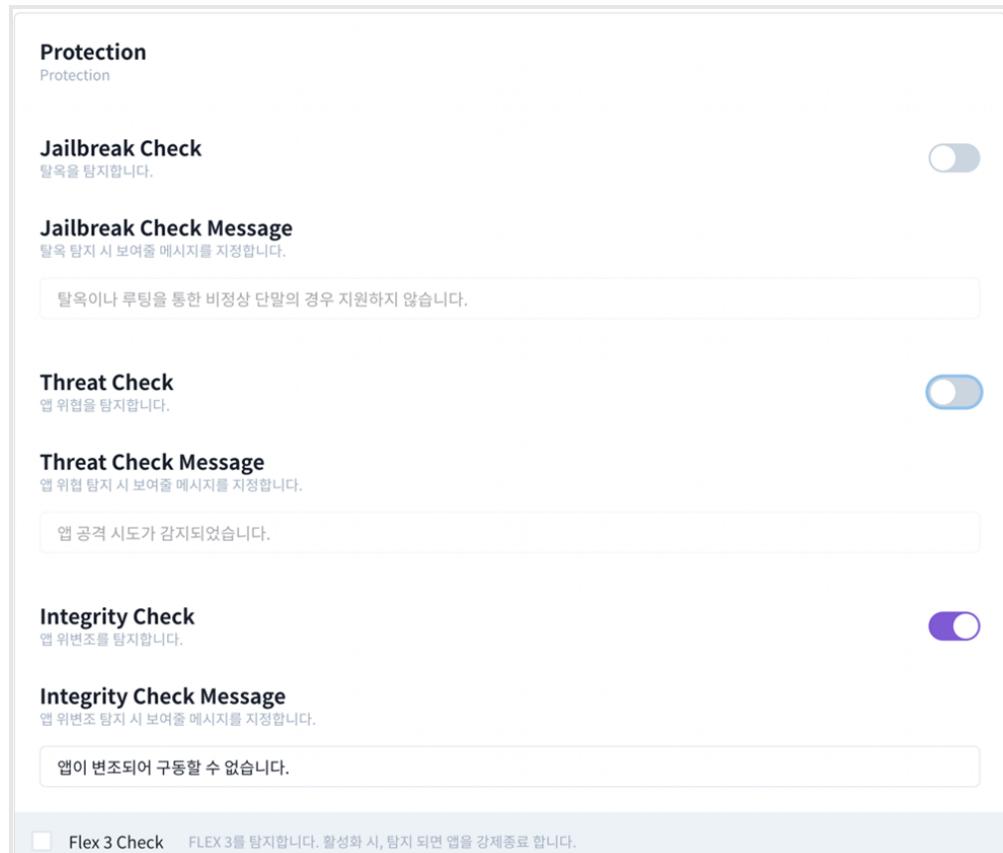


그림 75. AppSuit cloud-iwcm (2차) 옵션 해제

2. 탈옥단말기에서 Filza를 실행 후, "/var/containers/Bundle/Application/" 경로 내 검증할 앱의 Scheme 이름으로 되어있는 경로로 이동합니다.

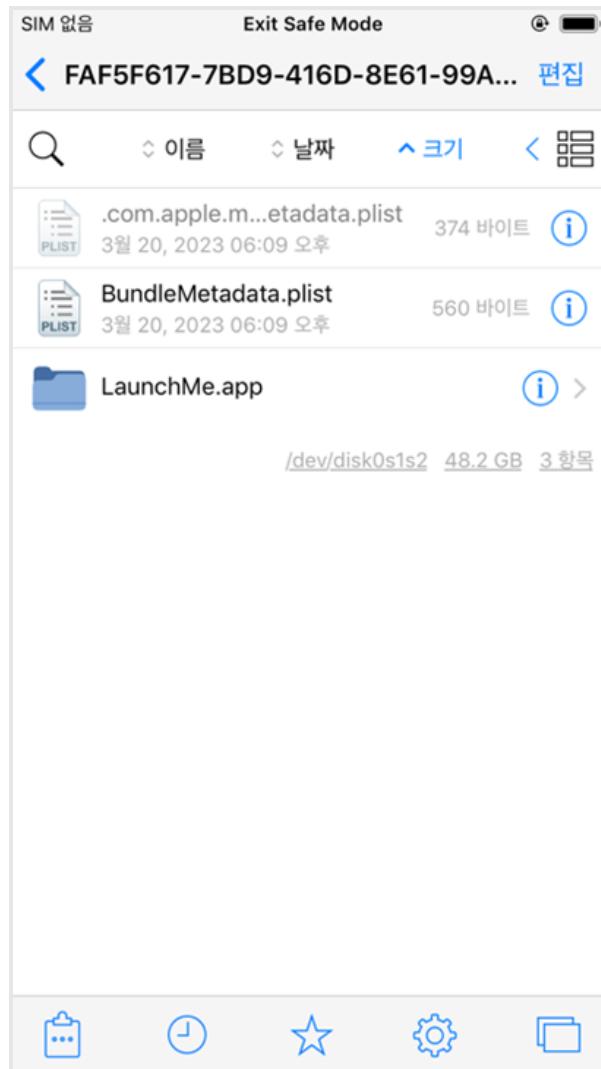


그림 76. 앱의 Scheme 이름의 경로로 이동

3. [앱 Scheme 이름].app 경로 내 앱 Scheme 이름과 동일한 앱 메인 바이너리 파일을 선택합니다.

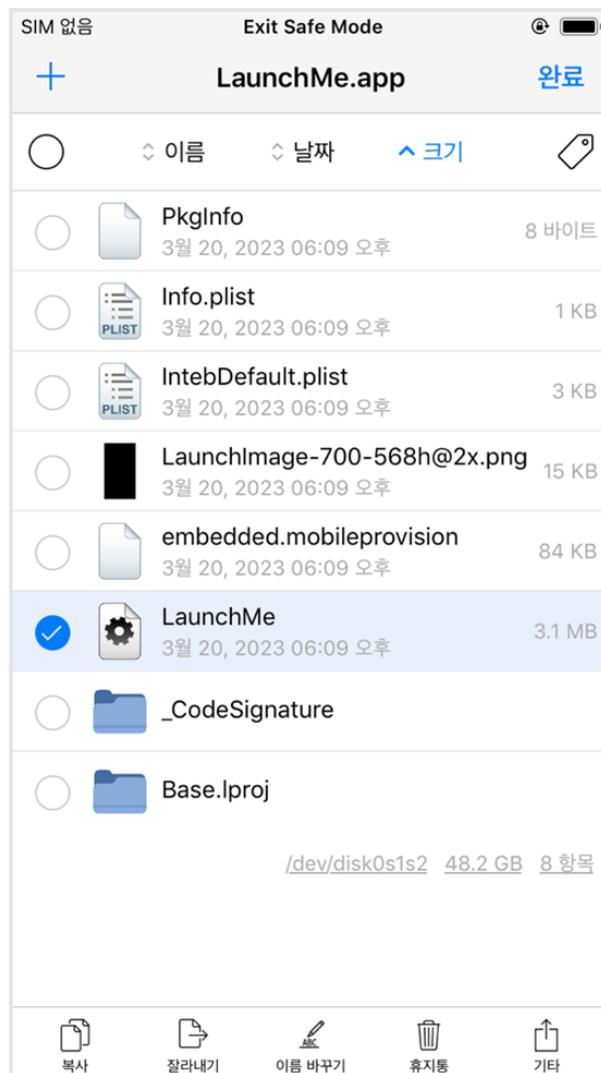


그림 77. 바이너리 파일 선택

4. 하단 메뉴에서 "기타 - 내장 앱으로 열기 - Hex 편집기"를 클릭합니다.

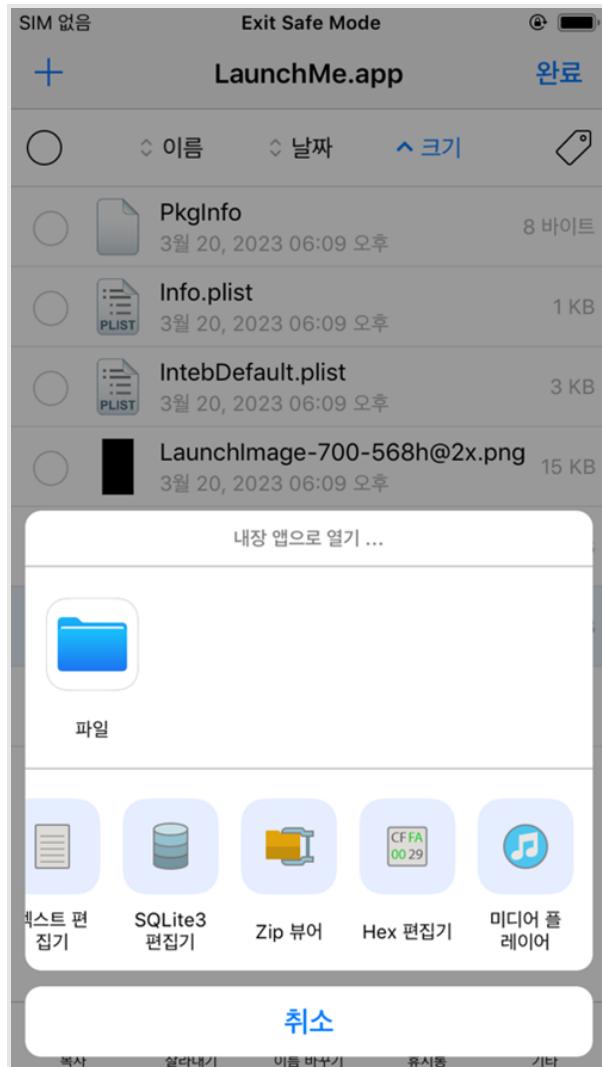


그림 78. Hex 편집기 클릭

5. 편집기에서 최상단의 \_\_Text와 같은 헤더 값들을 제외한 데이터 부분의 Hex 값을 수정합니다.  
수정 중 파일 백업 경고창이 뜨면 "아니요"를 클릭합니다.

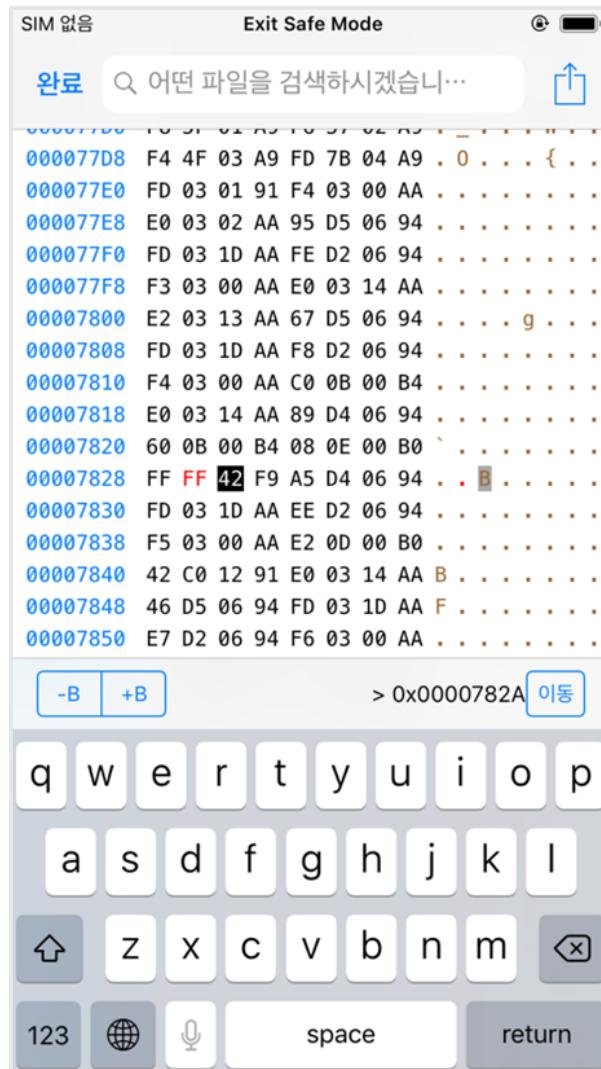


그림 79. Hex 값 수정

6. 수정 완료 후, 앱을 실행하여 위변조 탐지 여부를 확인합니다.

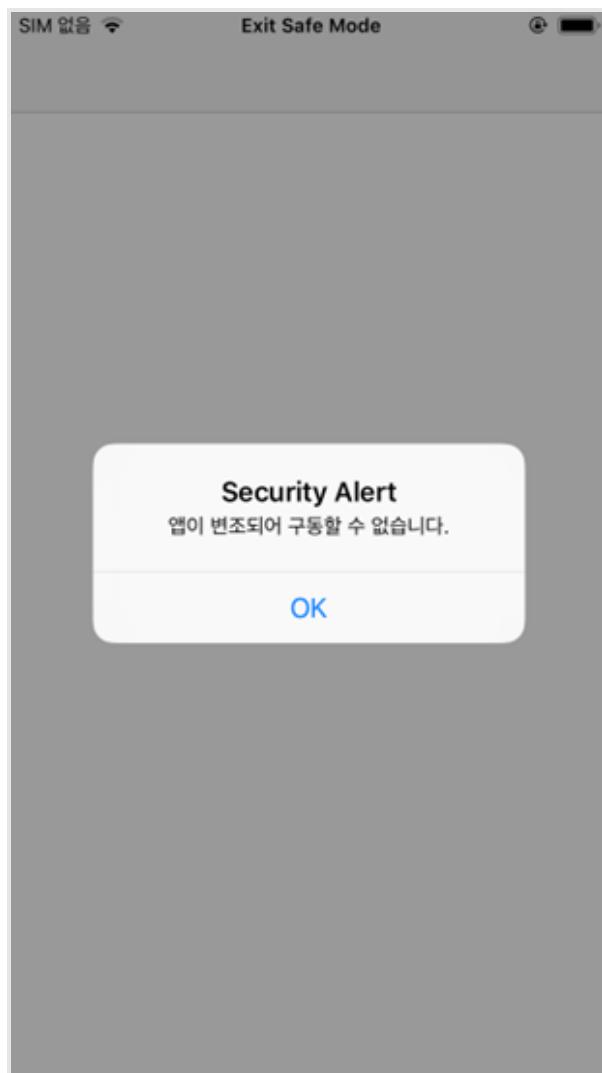


그림 80. AppSuit 위변조 탐지 알러트 확인

## k. Frida, 스위즐링 탐지 검증

- 검증 환경
  - 탈옥 디바이스
    - Cydia와 Frida 설치가 가능해야합니다.
  - Xcode, Frida 설치된 PC
  - 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
  - AppSuit Cloud 미적용 IPA

## • 탈옥 탐지 검증 시나리오

1. 탈옥 단말기에서 Cydia를 통해 Frida를 설치합니다.

- 탈옥에 사용한 탈옥 툴에 따라 최신 Frida 버전 사용이 불가할 수 있습니다.
- Frida 사용이 불가하다면 Frida 16.1 이하 버전 사용을 권장드립니다.

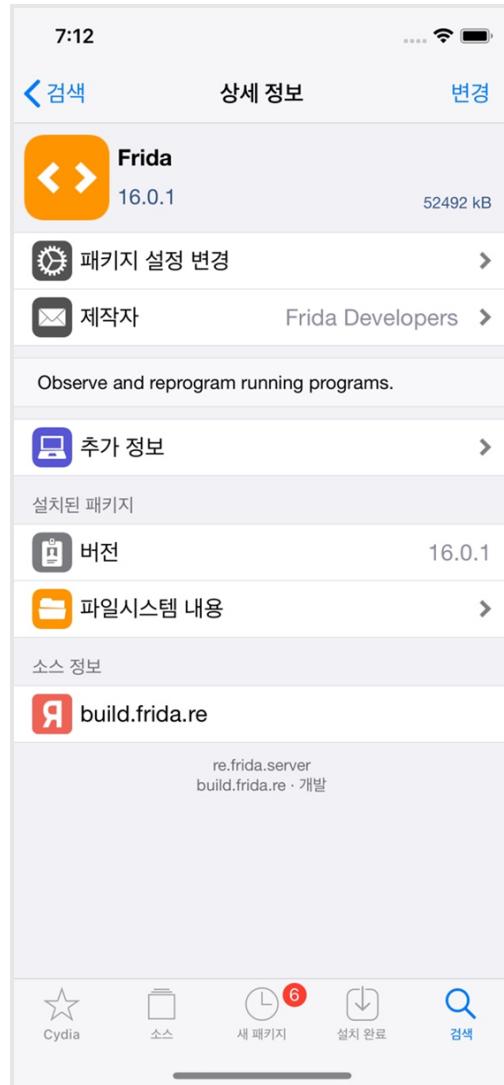


그림 81. Frida 설치

2. 아래 명령어를 입력하여 Frida Hooking 시 앱 강제 종료 여부를 확인합니다.

```
frida -U -f [IPA Bundle Identifier]
```

- Frida 탐지 시 알럿가 뜨지 않고 앱 실행 시 바로 종료됩니다.

```
frida -U -f com.stealien.LaunchMe
_____
| _ |   Frida 16.0.1 - 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/
| . . . |
| . . . |     Connected to iPhone (id=00008020-001334691A46002E)
| . . . | Spawning `com.stealien.LaunchMe'. Resuming main thread!
| iPhone::com.stealien.LaunchMe ]-> Process terminated
| iPhone::com.stealien.LaunchMe ]->
```

그림 82. Frida 탐지 확인

## I. Flex-3 탐지 검증

- 검증 환경

- 탈옥 디바이스
- Xcode 가 설치되어 IPA 설치가 가능한 PC
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
  - AppSuit cloud-iwcm (2차) 빌드 시 Flex 3 Check 옵션이 활성화 되어 있어야합니다.
- AppSuit Cloud 미적용 IPA

- Flex-3 탐지 검증 시나리오

1. 탈옥 단말기에서 Cydia 를 통해 Flex-3 를 설치합니다.

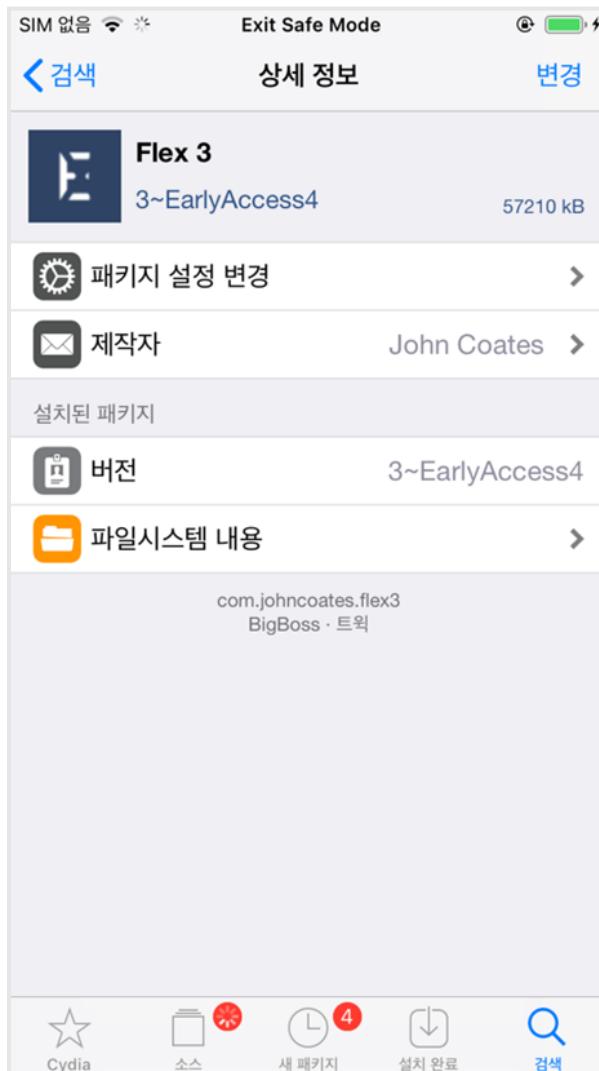


그림 83. Flex-3 설치

2. AppSuit Cloud 미적용 앱과 적용 앱을 각각 설치 후, 실행하여 Flex-3 탐지 여부를 확인합니다.

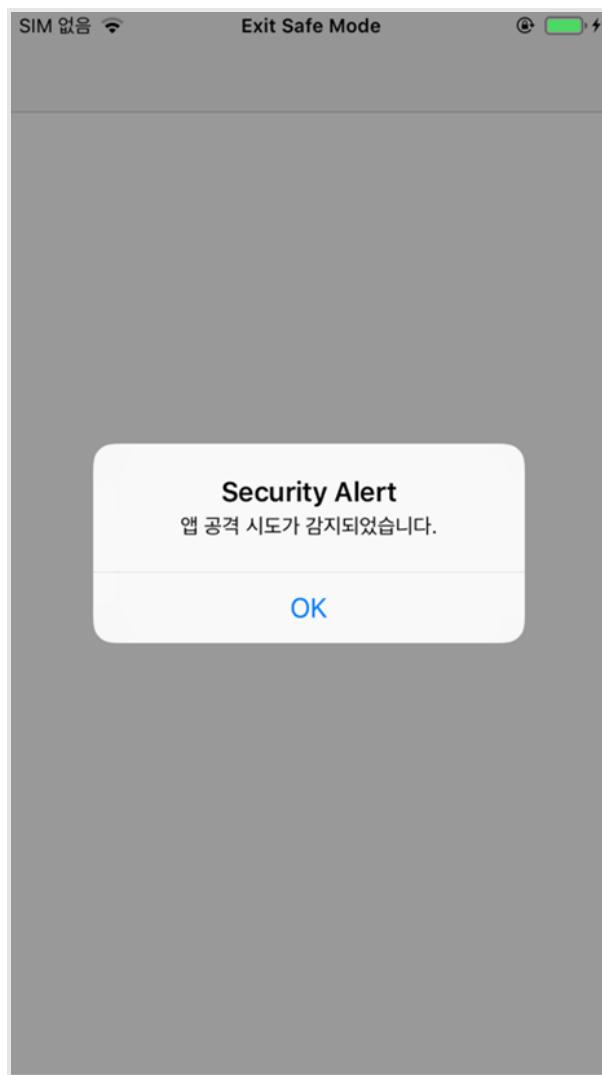


그림 84. Flex-3 탐지 확인

## m. 디버깅 탐지 검증

- 검증 환경

- 탈옥 디바이스
- Xcode 가 설치되어 IPA 설치가 가능한 PC
  - Provisioning Profile, iOS App Signer가 필요합니다.
- 재서명단계까지 진행 완료된 AppSuit Cloud 적용 IPA
- AppSuit Cloud 미적용 IPA

- 디버깅 탐지 검증 시나리오

- 탈옥 단말기에서 Cydia 를 통해 OpenSSH와 lldb-10를 설치합니다.

- 탈옥 단말기의 버전에 따라 설치 가능한 lldb 버전을 설치합니다.

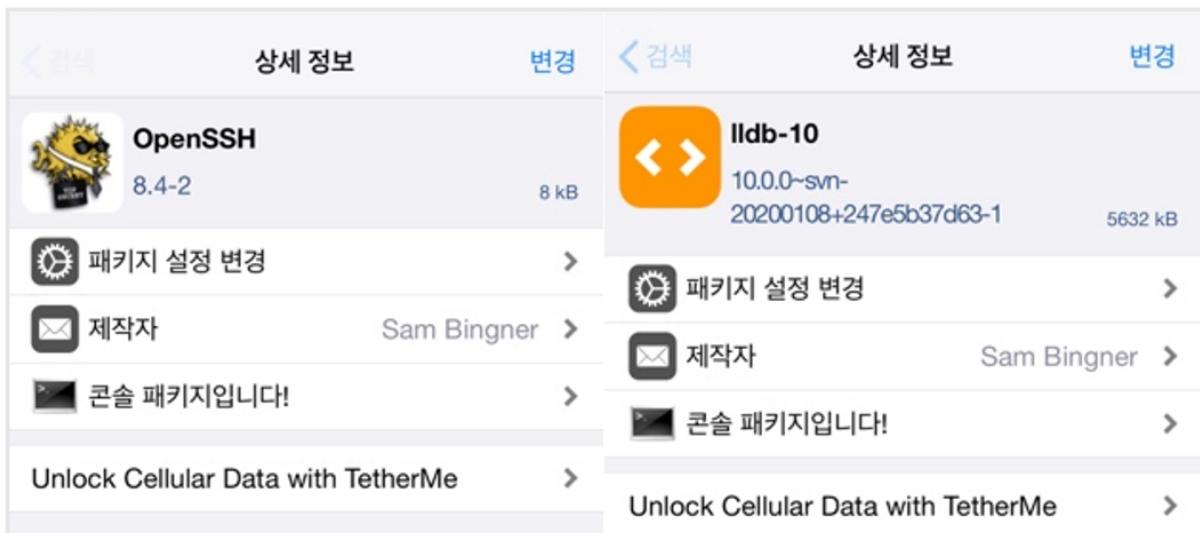


그림 85. OpenSSH와 lldb-10 설치

- 탈옥 단말기 설정 – Wi-Fi 속성에서 디바이스 IP 주소를 확인합니다.

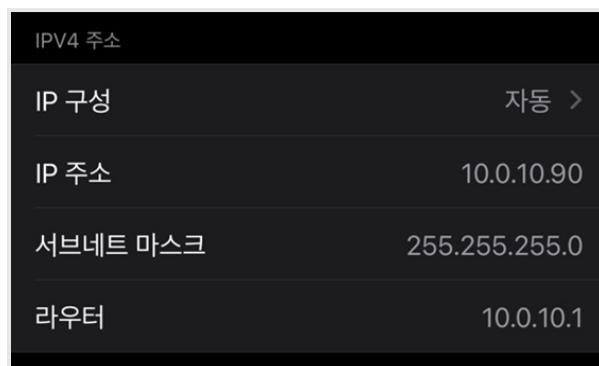


그림 86. 디바이스 IP 주소 확인

3. 디바이스 IP로 ssh 접속합니다.

- 기본 패스워드는 alpine입니다.

```
~ ssh root@10.0.10.90
root@10.0.10.90's password:
KM-TMs-jailbreak-iPhone:~ root#
```

그림 87. 디바이스 접속

4. AppSuit Cloud 적용 앱을 실행하여 아래 명령어를 통해 해당 앱의 pid를 확인합니다.

```
ps -ef
```

5. 아래 명령어를 입력하여 디버깅 시도 시 "lldb-10 error: attach failed: lost connection"으로 디버깅 실패하는 것을 확인합니다.

```
lldb-10 -p {PID}
```

6. 아래와 같이 **lost connection**이 뜨는 것을 확인합니다.

```
KM-TMs-jailbreak-iPhone:~ root# ps -ef | grep Launch
 501  3240      1   0  7:16PM ??          0:00.14 /var/containers/Bundle/Application/3079C7BD-DD39-46F4-BB98-664CDF8AA884/LaunchMe.app/LaunchMe
     0  3242  2415      0  7:16PM ttys001    0:00.03 grep Launch
KM-TMs-jailbreak-iPhone:~ root# lldb-10 -p 3240
(lldb) process attach --pid 3240
error: attach failed: lost connection
```

그림 88. lldb-10으로 디버깅 탐지 검증

## 5. 배포 방법

1. Transporter 프로그램을 설치 합니다.
2. Apple ID로 로그인합니다.
3. Transporter 프로그램을 실행하여 배포하고자하는 AppSuit Cloud 적용 IPA를 추가합니다.
  - 앱 관련 정보가 App Store Connect에 등록되어 있어야합니다.

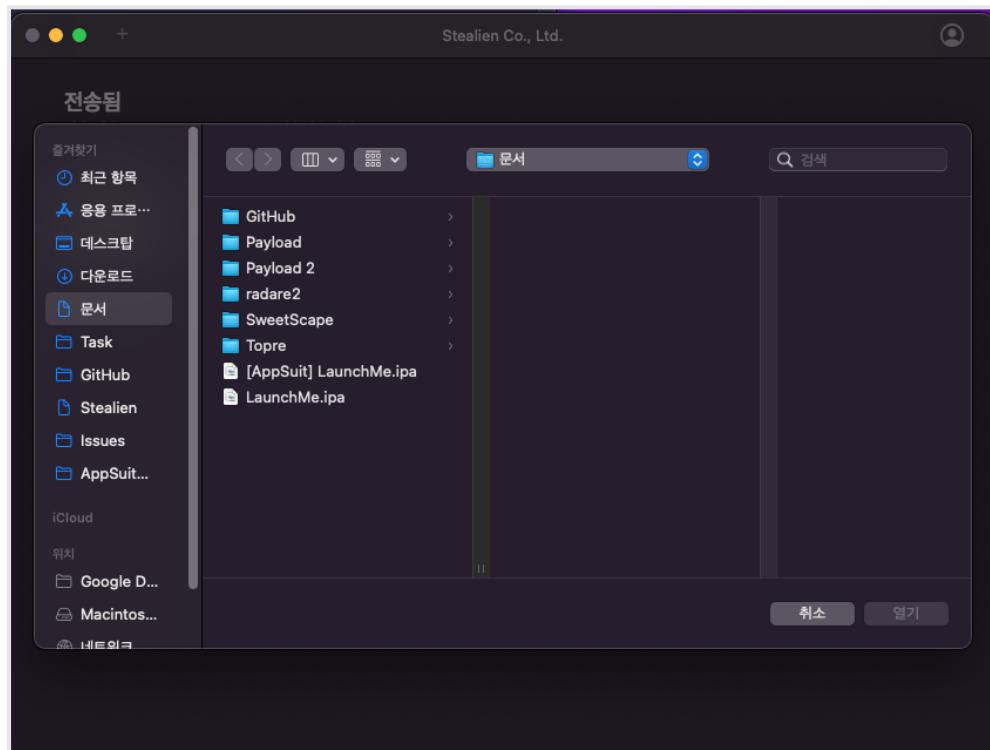


그림 89. Transporter 앱 추가

4. 지침에 따라 심사를 진행합니다.
5. 심사 완료 후 앱을 배포합니다.

부록

## A. [Optional] Bridging Header 설정

1. Swift 프로젝트의 경우, 추가적으로 **bridging header** 설정이 필요합니다.
2. AppSuit 라이브러리 파일 추가 과정에서 아래와 같은 창이 뜨는 경우, **Create Bridging Header**를 선택합니다.
  - 아래와 같이 메시지가 뜨지 않는다면 이미 bridging header가 설정되어 있는 것이므로, 설정된 bridging header(일반적인 경우 "프로젝트명-Bridging-Header.h" 파일)를 찾습니다.

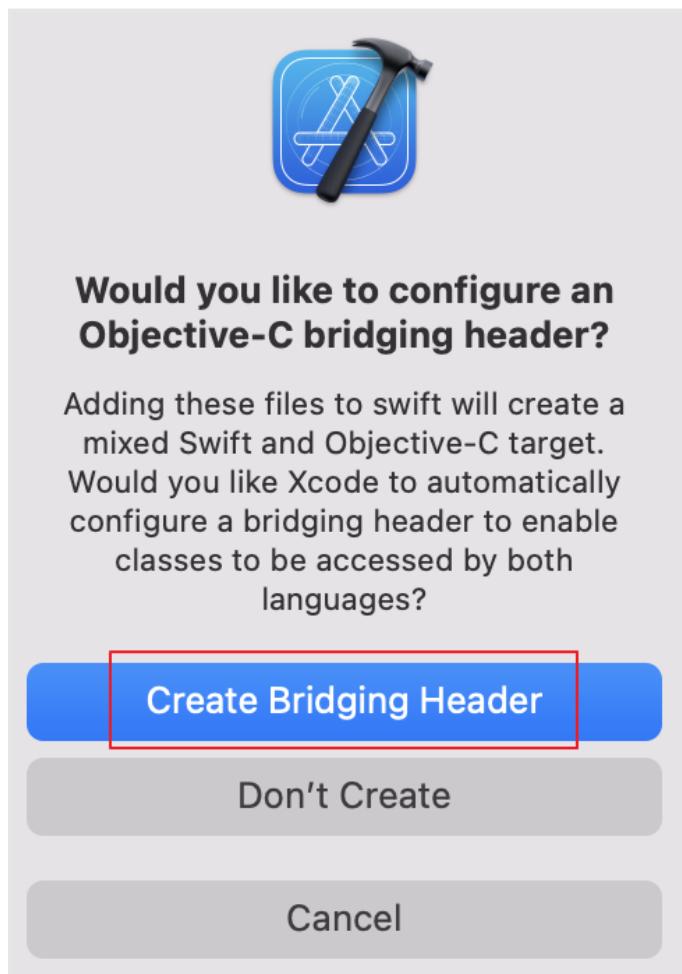


그림 90. Create Bridging Header 추가

3. Bridging Header 파일에 AppSuit 헤더파일 **stscore.h**를 **include** 합니다.

```
Ex) #include "stscore.h"
```

## B. [Optional] Bridging Header 수동 설정

- 파일 생성창이 뜨지 않고 Bridging Header 파일이 존재하지 않는 경우 다음과 같이 진행합니다.

- AppSuit 그룹에서 파일 생성을 합니다.

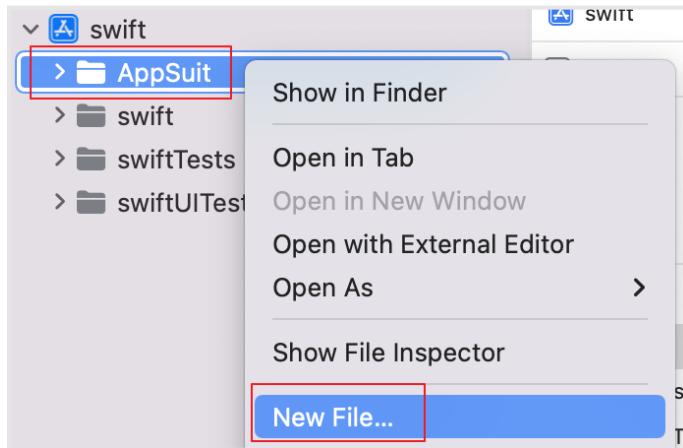


그림 91. 파일 생성

- 파일 생성에서 Header File을 선택하여 "프로젝트명-Bridging-Header.h" 파일을 생성합니다.
- Xcode Project Setting - Build Settings - Objective-C Bridging-Header에 Bridging Header 파일 경로를 지정합니다.

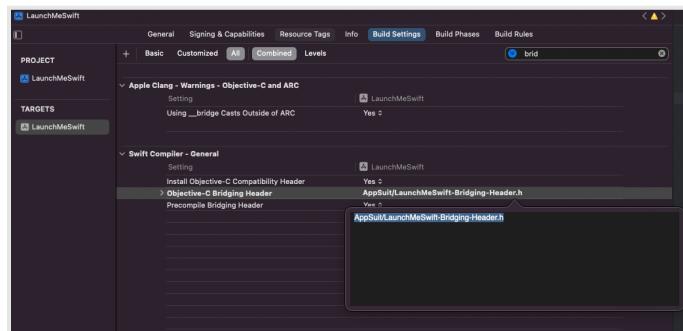


그림 92. Bridging Header 파일 생성

- 생성한 Bridging Header 파일에 AppSuit 헤더파일 `stscore.h`를 include 합니다.

```
Ex) #include "stscore.h"
```

## C. AppSuit Cloud 2차 적용 옵션

- **String Encryption**

- **Objective-C String Encryption**

Objective-C 문자열 암호화 여부를 지정합니다.

- \_\_cfstring 에 대해서만 문자열 암호화를 지원합니다.
    - Enterprise Distribution 의 경우 지원하지 않습니다.

- **Swift String Encryption**

Swift 문자열 암호화 여부를 지정합니다.

- \_\_cstring 섹션에 노출되어있는 문자열에 대해서만 암호화를 지원합니다.
    - Enterprise Distribution 의 경우 지원하지 않습니다.

- **Protection**

- **Jailbreak Check**

탈옥을 탐지합니다.

- **Jailbreak Check Message**

탈옥 탐지 시 보여줄 메시지를 지정합니다. 기본값은 "탈옥이나 루팅을 통한 비정상 단말의 경우 지원하지 않습니다." 입니다.

- **Threat Check**

앱 위협을 탐지합니다.

- **Threat Check Message**

앱 위협 탐지 시 보여줄 메시지를 지정합니다. 기본값은 "앱 공격 시도가 감지되었습니다" 입니다.

- **Integrity Check**

앱 위변조를 탐지합니다.

- **Integrity Check Message**

앱 위변조 탐지 시 보여줄 메시지를 지정합니다. 기본값은 "앱이 변조되어 구동할 수 없습니다." 입니다.

- **Flex 3 Check**

FLEX 3를 탐지합니다.

- **Server Authentication**

※ 본 기능은 서버인증 제품(AppSuit ServerAuth) 및 handler 적용을 위한 옵션입니다.

- **Server Auth Manually**

서버 인증을 수동으로 구현하여 사용할 수 있습니다.

- **Terminate On Detect**

탐지 후 자동으로 앱을 종료하기 위한 옵션입니다.

Server Auth Manually 옵션 활성화 시 사용 가능합니다.

- **Terminate On Detect Timer**

탐지 후 자동으로 종료할 시간을 설정합니다.

Terminate On Detect 옵션 활성화 시 사용 가능합니다.

- **Obfuscation**

- **Objective-C Class Name Obfuscation**

Objective-C 클래스명을 난독화합니다.

- **Swift Class Name Obfuscation**

Swift 클래스명을 난독화합니다.

- **Symbol Delete**

심볼 및 디버깅 정보를 제거합니다.

- **Log Hiding**

Xcode Device Console에서 NSLog 출력을 안하도록 설정합니다.

- **Dynamic API Hiding**

Decompiler를 통한 정적 분석 시 함수를 식별하지 못하도록 숨깁니다.

- **Class Name Obfuscation**

클래스명 난독화 시, 예외 처리할 클래스명을 지정합니다 (#클래스명)(\*\*전체 예외 처리할 단어)

- Utility

- **Enable Local Alert Message**

- 앱 보안 위협 탐지 시 알림 창을 띄울지 정합니다.

- 활성화 시, 기존과 동일하게 탐지 Alert 를 띄우고 8초 뒤 앱이 자동으로 종료됩니다.

- 비활성화 시, 탐지 Alert 를 띄우지 않고 앱을 즉시 종료합니다.

- **Jailbreak Check Test for iOS 17**

- iOS 17 에서 탈옥 탐지 테스트 활성화 여부를 지정합니다.

- 앱 최초 실행 시, "로컬 네트워크에 대한 접근 권한"을 요청하는 Alert 가 발생하며, 해당 접근 허용 후 앱 재시작 시 기능 사용이 가능합니다.

- "로컬 네트워크에 대한 접근 권한"의 경우 정상 단말기에서 진행하는 탈옥 탐지 테스트를 하기 위한 권한이며, 탈옥 단말기에서는 권한 허용을 하지 않아도 정상적으로 탐지됩니다.

- 기능 동작을 위해 앱의 NSUserDefaults 에 "AS\_CHECK" 키 값이 추가됩니다.

- 기능 동작을 위해 앱의 Info.plist에 "Bonjour services" 관련 값이 자동으로 추가됩니다.

- (Key) Bonjour services

- (Value) \_bonjour.\_tcp

- (Value) \_lnp.\_tcp

- 비활성화 시 iOS 17 미만 버전에서만 "JBBB123456789" 또는 "bjbjb123456789" 으로 탈옥 탐지 테스트 가능합니다.

- **※ 참고 사항**

- 현재 가이드 내용으로 적용된 앱으로 최초 탈옥 탐지 테스트 진행 시 검은화면이 나오고 8초뒤 앱이 종료되는 경우, 앱이 완전히 종료되지 않고 메모리에 남아있는 경우입니다. 해당 경우 네트워크 권한 허용 후 단말기를 재부팅하여 탐지 Alert를 확인할 수 있습니다.

- 디바이스 이름 변경 후에도 탈옥 탐지 테스트가 되지 않는 경우, 디바이스 이름이 바르게 변경되지 않은 경우입니다. 해당 경우 디바이스 이름을 다른 이름으로 설정 후 다시 JBBB123456789 또는 jbjb123456789로 변경하여 탐지 여부를 확인할 수 있습니다.

- **Use CFBundleVersion for Server Auth**

- 서버인증 시 CFBundleShortVersionString 대신 CFBundleVersion 을 사용합니다.

- 비활성화 시, 기존 서버인증 시 사용되는 버전과 동일한 "CFBundleShortVersionString" 값으로 인증을 수행합니다.

- 활성화 시, "CFBundleVersion" 값으로 인증을 수행합니다.

- 서버인증 관리 페이지 앱 등록 시 CFBundleVersion으로 앱을 등록해야 합니다.

- **Use Radar Server**

- AppSuit Radar 서버를 사용여부를 설정합니다.

- 활성화 시, 입력된 URL 내 AppSuit Radar 서버로 탐지 로그를 전송하여 대시보드를 통해 가시화된 데이터를 확인할 수 있습니다.

- (자세한 설정 방법은 Radar 사용 가이드를 참고 부탁드립니다.)

- **Set Radar Server URL**

탐지 데이터를 송신하기 위해 AppSuit Radar 서버의 URL 을 설정합니다.

- 반드시 Radar 서버의 URL을 작성해야 하며, **https://[Radar 서버 주소]/radar/** 형식으로 작성되어야 합니다.  
(자세한 설정 방법은 Radar 사용 가이드를 참고 부탁드립니다.)

## D. Class Name Obfuscation 적용 방법

- 전체 클래스에 대해 난독화를 적용합니다.

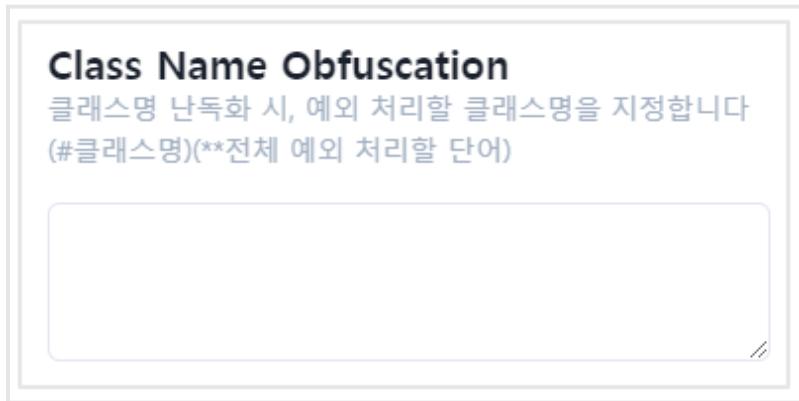


그림 93. 전체 클래스 난독화 적용

- 특정 클래스에 대해 난독화를 제외합니다.

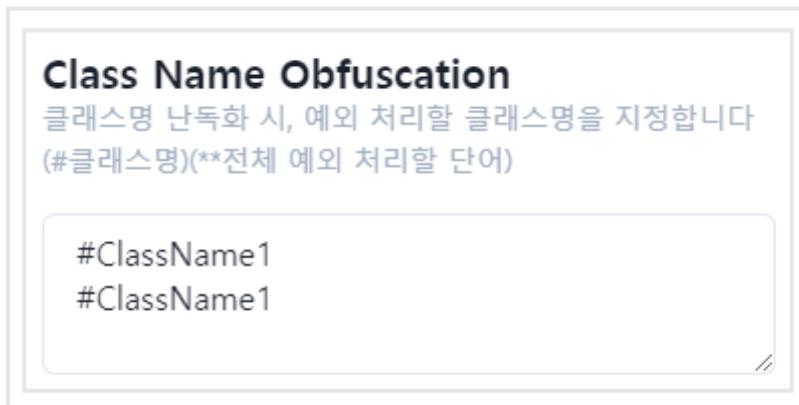


그림 94. 특정 클래스 난독화 제외

## E. Class Name Obfuscation 예외처리 방법

※ 클래스 난독화로 인한 앱 Crash 발생 시 진행합니다. 무분별하게 적용 시 보안성이 매우 떨어질 수 있습니다.

1. Xcode에서 Command + Shift + 2 + 를 눌러 디바이스 설정 창을 엽니다.
2. Open Console 버튼을 클릭하여 Xcode Device Console을 엽니다.

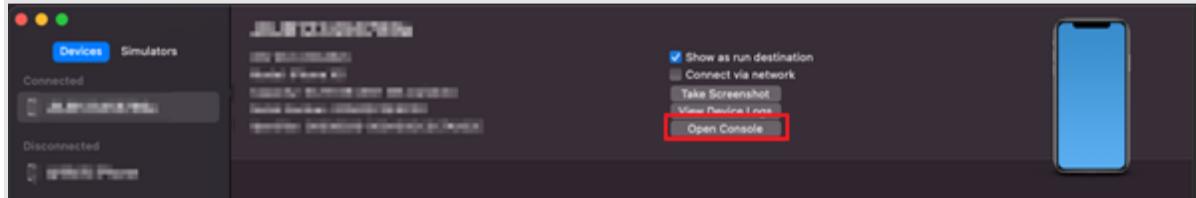
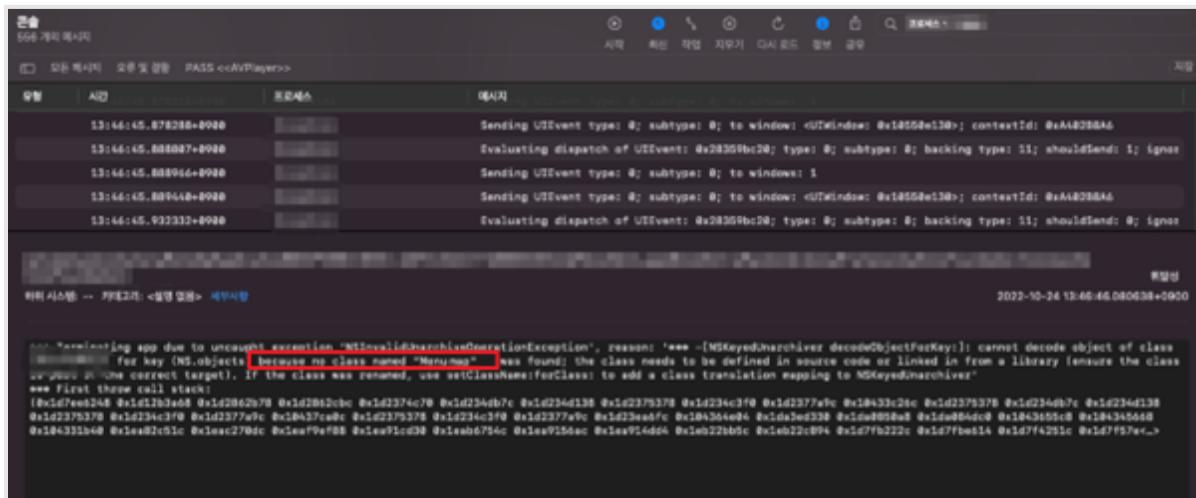


그림 95. Xcode Device Console 열기

3. 이슈가 발생하는 디바이스를 연결합니다.
4. Console에서 프로세스 앱 스키마명으로 검색 후 앱을 실행합니다.
5. 실행한 앱에 대한 Crash 로그를 확인합니다.



```
*** Terminating app due to uncaught exception 'NSInvalidArgumentException', reason: '*** -[NSKeyedUnarchiver decodeObjectForKey:]: cannot decode object of class (MenuMap) for key (NS.objects) because no class named "MenuMap" was found; the class needs to be defined in source code or linked in from a library (ensure the class is part of the correct target). If the class was renamed, use setClassName:forClass: to add a class translation mapping to NSKeyedUnarchiver'
```

그림 96. Crash 로그 확인

6. AppSuit Cloud 서버에서 해당 IPA 빌드로그 - Download Log 를 통해 dump.log 파일을 다운로드합니다.
7. Crash 로그에서 확인한 클래스를 dump.log 에서 검색합니다.
8. 검색하여 해당 클래스의 난독화 전 클래스 이름을 확인합니다.
  - [ [난독화 전 클래스 이름] ⇒ [난독화 후 클래스 이름] ] 형식으로 검색됩니다.
9. 확인한 난독화 전 클래스를 AppSuit Cloud Admin 페이지 - Preset - Class Name Obfuscation에 입력하여 예외처리합니다.



그림 97. 이슈 발생 클래스에 대한 난독화 예외처리

- NSClassFromString과 같이 **클래스의 이름을 String으로 호출하는 메소드**를 사용하는 경우

클래스 이름은 난독화가 되어 있지만, 인자 값으로 들어가는 난독화 적용 전 String 클래스 이름을 찾지 못하여 Crash가 발생합니다.

따라서 **let aClass = NSClassFromString("stealien")**과 같은 경우 다음과 같이 예외처리합니다.

- 올바른 예외처리



그림 98. 올바른 예외처리

- 잘못된 예외처리



그림 99. 잘못된 예외처리

## F. AppSuit Open Page

- (주)스틸리언에서는 솔루션의 최신화 및 안정적인 지원을 위해 Open Page 를 제공하고 있습니다.  
AppSuit Cloud 적용 시 필요한 파일 및 FAQ 등은 아래 스틸리언 Open Page 참고 부탁드립니다.

<https://appsuit.notion.site/AppSuit-Premium-5eafc8bde42246a7ae8542e45fb1ccd7>

- 지원 환경
  - AppSuit Cloud 를 적용하기 위해 필요한 환경에 대한 정보와 가이드가 작성되어 있습니다.
- ReleaseNote
  - AppSuit Cloud 적용 시 필요한 AppSuit 라이브러리 (1차) 과 cloud-iwcm (2차) 에 대한 정보가 작성되어 있습니다.
- Code Snippets & Guides
  - AppSuit Cloud 사용 시 함께 사용 가능한 제품에 대한 정보가 작성되어 있습니다.
- Trouble Shooting
  - FAQ에 작성되어 있지 않는 이슈 발생 시 스틸리언 엔지니어에게 전달해주실 정보가 작성되어 있습니다.
- FAQ
  - AppSuit Cloud 적용 시 발생하는 이슈에 대한 해결방안이 작성되어 있습니다.
- Utilities
  - AppSuit Cloud 적용 시 사용 가능한 유틸리티에 대한 정보가 작성되어 있습니다.

(주) 스텔리언

TEL : 02-2088-4835

E-mail : [se@stealien.com](mailto:se@stealien.com)

주소 : 서울특별시 용산구 원효로90길 11, 더프라임타워 업무동 12층

홈페이지 : <https://www.stealien.com/main>