

프리다를 이용한 안드로이드 앱 모의해킹

BCG 연구실 IT정보공학과 김아은

INDEX

프리다를 이용한
안드로이드 앱 모의해킹

- 프리다(Frida)란?

- 모의해킹 실습

- 루팅 탐지 우회

- 암호 복호화

- 로그인 우회

- 브루트포스

- SSL Pinning

프리다(Frida)란?

❖ 프리다란?

- 다양한 플랫폼(윈도우, 맥, GNU/Linux, iOS, Android, QNX)에서 프로세스에 대한 인젝션이 가능해 큰 확장성을 가진 DBI 프레임 워크
- 자바스크립트를 네이티브 앱에 삽입 가능
- 프로세스를 모니터/디버깅하는 데 사용할 수 있는 툴킷

❖ 주요 기능

- 함수 후킹 (특정 함수에 연결하여 반환 값 변경, 함수 재작성 등)
- 애플리케이션 디버깅
- 힙 메모리 내 객체 인스턴스 검색 및 사용
- 실시간 트래픽 스니핑 또는 암호 해독
- 탈옥 또는 루팅되지 않은 단말기에서도 사용 가능



프리다(Frida)란?

❖ 실습 환경



- Frida
- Nox
- Anaconda
- Frida-server
- Burp suite



프리다(Frida) 기본 명령어

❖ 명령어

- frida : 신속한 프로토타이핑과 손쉬운 디버깅이 목표인 명령어
- frida-ps : 연결된 프로세스 목록을 출력하기 위한 명령어
- frida-kill : 프로세스를 종료하는 명령어
- frida-trace : 함수 호출을 동적으로 추적하기 위한 명령어
- frida-ls-devices : 프리다에 연결된 프로세스 목록을 출력하기 위한 명령어



프리다(Frida) 기본 문법

❖ 기본 문법

- `Java.perform(fn)`
- `Java.use(className)`
- `Java.choose(className, callbacks)`
- `Java.enumerateLoadedClasses(callbacks)`
- `setImmediate(fn)`



프리다(Frida) 기본 문법

❖ Java.perform(fn)

- 현재 스레드가 가상머신에 연결되어 있는지 확인하고 fn을 호출
- frida를 활용하려면 이 메소드를 활용하여 가상머신 연결 여부를 확인함

```
Java.perform(function() {  
    ...  
})
```



프리다(Frida) 기본 문법

❖ Java.use(className)

- 변수와 메소드에 액세스할 수 있는 클래스 객체를 반환 (인스턴스 반환이 아님 !!)
- 메소드 구현을 변경하려면 새로운 메소드로 덮어 써야 함

```
Java.perform(function() {  
    var myClass = Java.use(com.[mypackage].name.class)  
    myClass.[myMethod].implementation = function(param) #(param1, param2)  
    {  
        ...  
    }  
})
```

use 함수를 이용하여 myClass에 앱에서 사용하는 클래스를 정의
.implementation → 클래스의 메소드 재작성



프리다(Frida) 기본 문법

❖ Java.choose(className, callbacks)

- 힙에서 인스턴스화 된 객체 찾기 가능
- onMatch : 실시간으로 인스턴스에 대해 호출, 요청에 일치하는 것을 찾으면 하나 이상의 인수로 호출됨
- onComplete : function()의 모든 인스턴스가 열거될 때 호출, 모든 호출에 대해 완료하면 종료

```
Java.perform(function() {  
  Java.choose(com.[mypackage].name.class,  
  {  
    "onMatch": function(instance) {  
      console.log(instance.toString())  
    },  
    "onComplete": function() {  
    }  
  })  
})
```

1. perform을 통해 가상머신과의 연결을 확인하고,
2. choose를 통해 패키지의 인스턴스를 찾으면 instance 변수에 저장되고
3. onMatch에서 인스턴스에 대한 내용이 콘솔로그로 출력되고
4. 모든 출력을 마치면 onComplete가 사용됨



프리다(Frida) 기본 문법

❖ setImmediate(fn)

- frida는 애플레이션이 느려지면 시간초과되어 연결을 자동으로 종료하는 경우가 있음
- 애플레이터가 느려져 timeout이 되더라도 이 함수가 frida를 자동으로 백그라운드로 재실행함

```
setImmediate(function() { // prevent timeout
    console.log("[*] Starting script");

    Java.perform(function() {
        var myClass = Java.use("com.package.name.class.name")
        myClass.implementation = function(v) {
            ...
        }
    })
})
```



프리다를 활용한 모의해킹

❖ 모의해킹

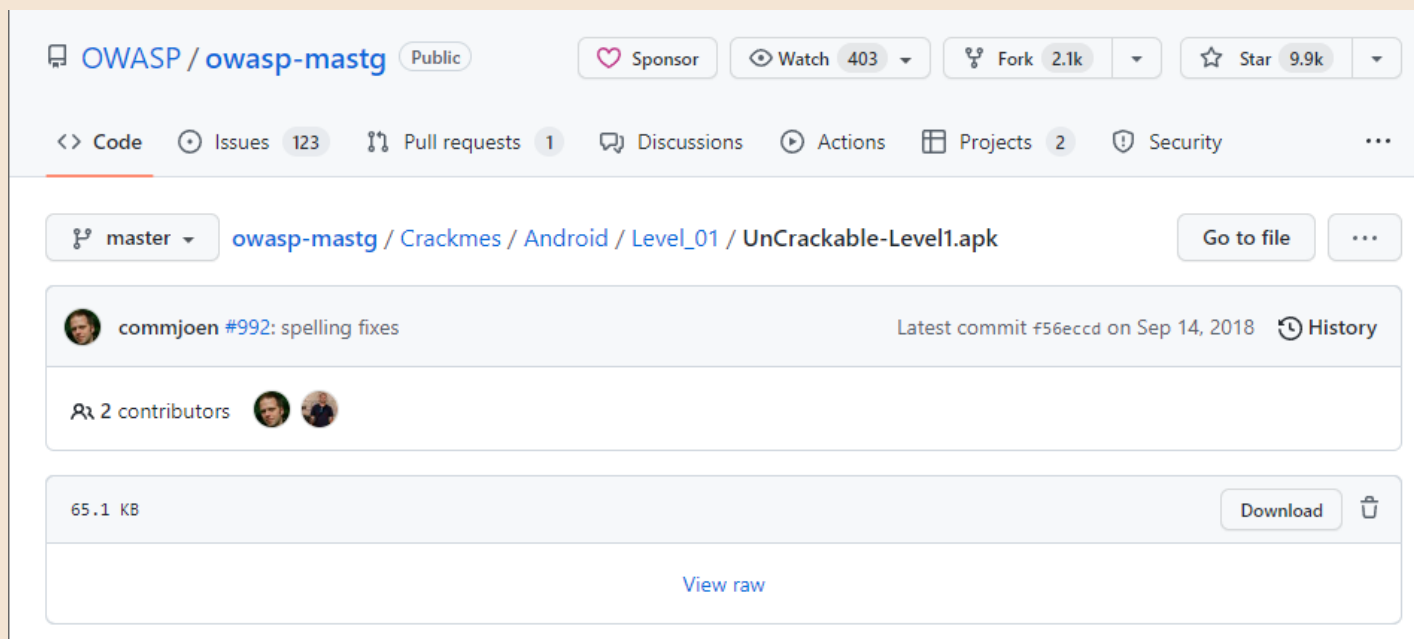
- 루팅 탐지 우회
 - 암호 복호화
 - 로그인 우회
 - 브루트포스
 - SSL Pinning
- UnCrackable-Level1.apk
- sieve.apk
- android-ssl-pinning.apk



프리다를 활용한 모의해킹

❖ UnCrackable-Level1.apk

- 링크 : https://github.com/OWASP/owasp-mastg/blob/master/Crackmes/Android/Level_01/UnCrackable-Level1.apk
- 앱 실행 시 root 탐지 및 복호화된 암호(secret string) 찾기



프리다를 활용한 모의해킹 -루팅 탐지 우회

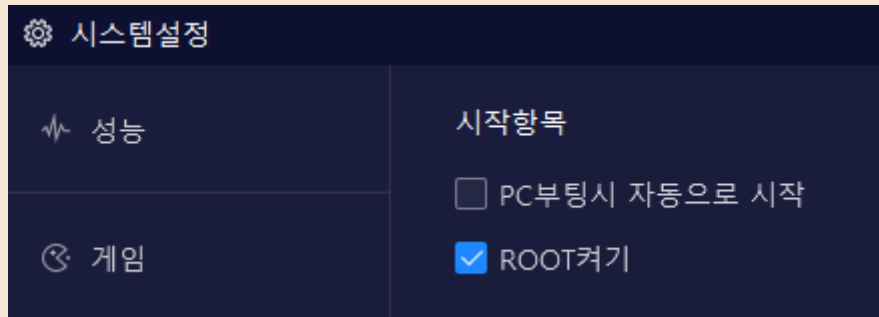
❖ 안드로이드 루팅(rooting)

- 시스템 파일에 제한 없이 접근하거나 변경, 삭제, 수정이 가능한 기술
- 장치의 소프트웨어 코드 수정 가능
- 안드로이드 장치에 이점을 제공할 수 있지만, 중요한 파일에 접근하는 것은 다양한 문제 발생 가능
- 애플리케이션을 만들 때, 개발자는 루트 탐지 메커니즘을 포함하여 사용자가 루트 안드로이드 장치에서 사용하지 못하도록 설정함
- 루트 탐지 → 애플리케이션 설치 거부 또는 종료
- 애플리케이션 코드를 변경하여 루트 안드로이드 장치에서 애플리케이션 실행 가능

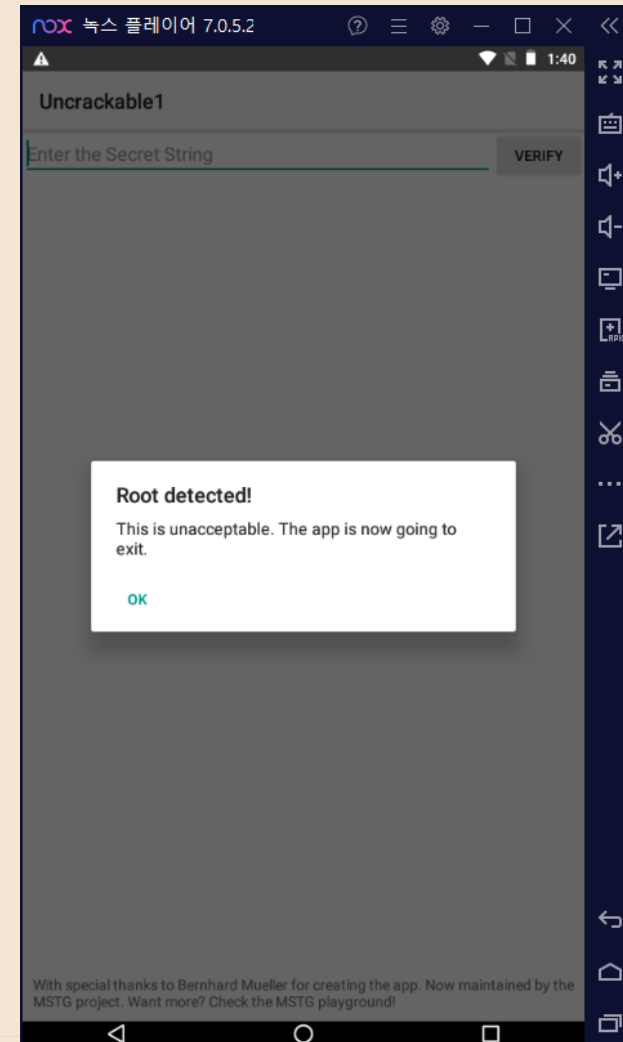


프리다를 활용한 모의해킹 -루팅 탐지 우회

❖ 앱 실행



- Nox 에뮬레이터 설정에서 ROOT를 켜 상태로 UnCrackable 앱 실행
- 'Root detected!'라며 루트를 탐지함
- OK 버튼을 누르면 실행 중인 앱이 강제로 종료됨



프리다를 활용한 모의해킹 -루팅 탐지 우회

❖ 소스코드 확인

```
p000sg.vantagepoint.uncrackable1.MainActivity ✕  
14  
15 /* renamed from: sg.vantagepoint.uncrackable1.MainActivity */  
16 public class MainActivity extends Activity {  
17     /* renamed from: a */  
18     private void m5a(String str) {  
19         AlertDialog create = new Builder(this).create();  
20         create.setTitle(str);  
21         create.setMessage("This is unacceptable. The app is now going to exit.");  
22         create.setButton(-3, "OK", new OnClickListener() {  
23             public void onClick(DialogInterface dialogInterface, int i) {  
24                 System.exit(0);  
25             }  
26         });  
27         create.setCancelable(false);  
28         create.show();  
29     }  
30  
31     /* access modifiers changed from: protected */  
32     public void onCreate(Bundle bundle) {  
33         if (C0002c.m2a() || C0002c.m3b() || C0002c.m4c()) {  
34             m5a("Root detected!");  
35         }  
36         if (C0001b.m1a(getApplicationContext())) {
```

Root detected!

This is unacceptable. The app is now going to exit.

OK

OK 버튼을 눌러도 앱이 종료되지 않게
메서드를 재작성한다면?



프리다를 활용한 모의해킹 -루팅 탐지 우회

❖ 루팅 탐지 우회 스크립트 작성

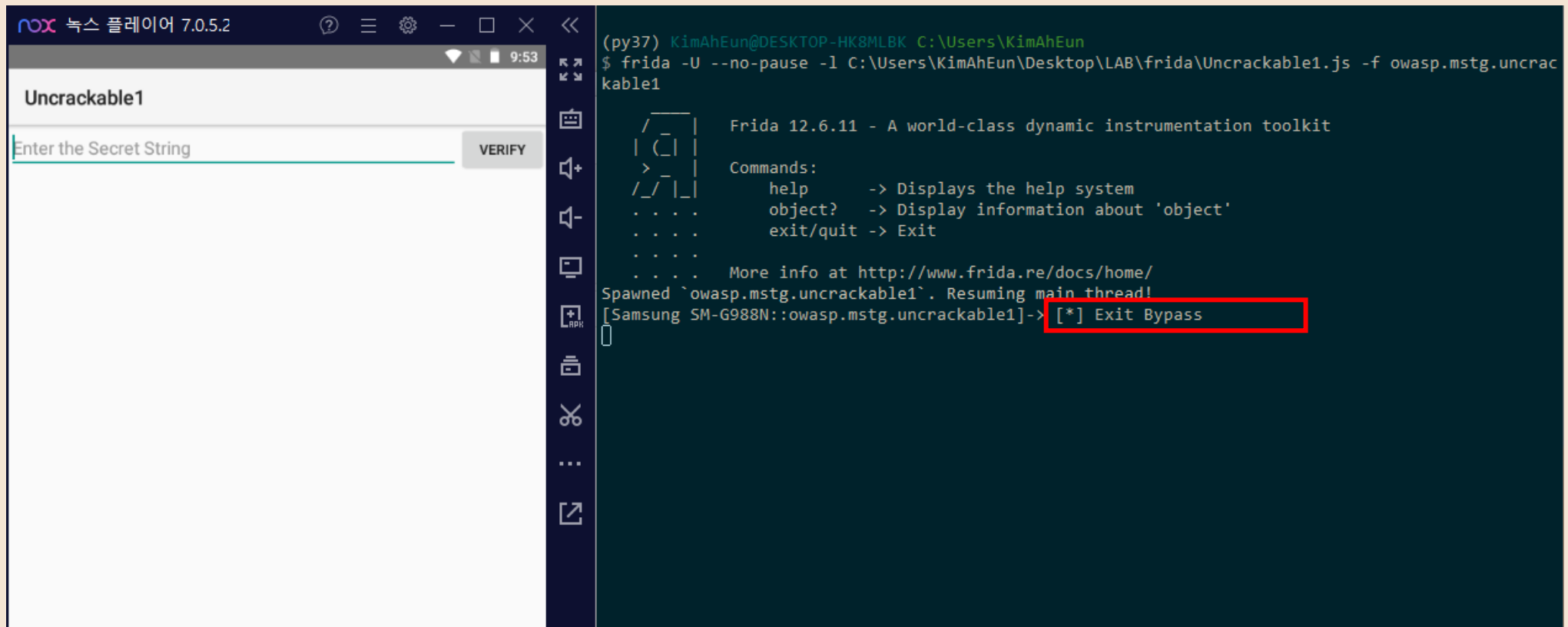
```
1  setImmediate(function() {  
2      Java.perform(function() {  
3          var System = Java.use("java.lang.System");  
4          System.exit.implementation = function() {  
5              console.log("[*] Exit Bypass");  
6          }  
7      })  
8  })
```

→ exit 함수가 단순 콘솔로그만 출력하도록 재작성



프리다를 활용한 모의해킹 -루팅 탐지 우회

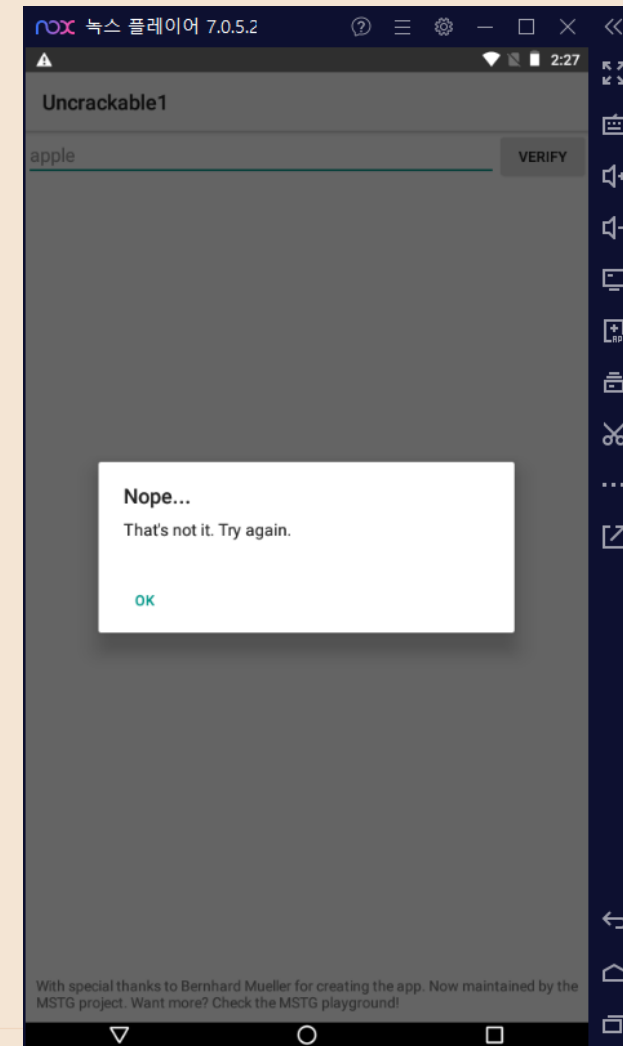
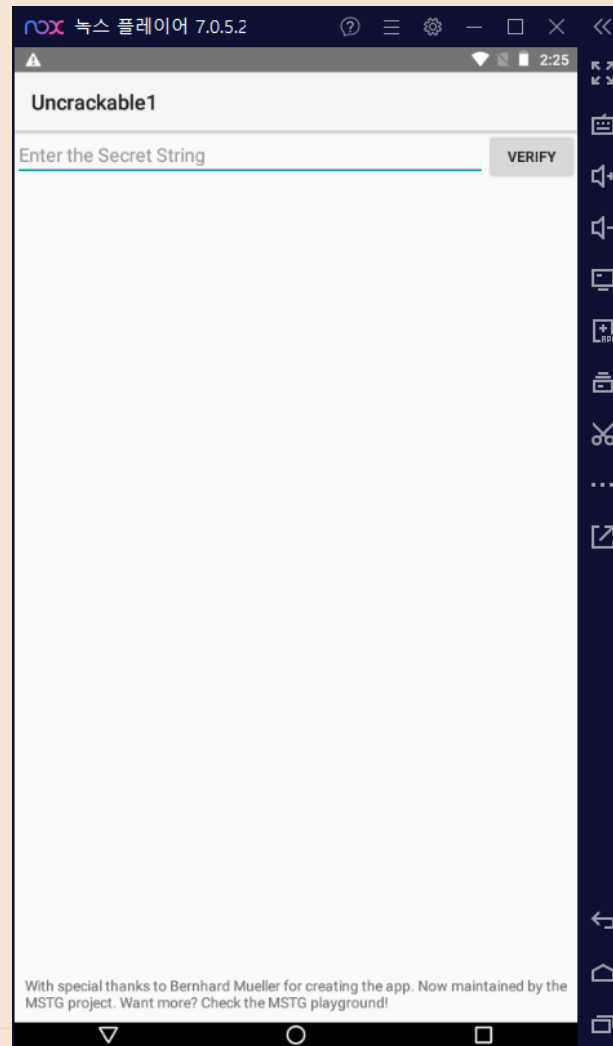
❖ 스크립트 인젝션



프리다를 활용한 모의해킹 - 암호 복호화

❖ Secret String 입력

→ 암호가 맞지 않다면
Nope... 팝업창 출력



프리다를 활용한 모의해킹 - 암호 복호화

❖ 소스코드 확인 (verify)

```
p000sg.vantagepoint.uncrackable1.MainActivity ✕  
42  
43 public void verify(View view) {  
44     String str;  
45     String obj = ((EditText) findViewById(R.id.edit_text)).getText().toString();  
46     AlertDialog create = new Builder(this).create();  
47     if (C0005a.m6a(obj)) {  
48         create.setTitle("Success!");  
49         str = "This is the correct secret.";  
50     } else {  
51         create.setTitle("Nope...");  
52         str = "That's not it. Try again.";  
53     }  
54     create.setMessage(str);  
55     create.setButton(-3, "OK", new OnClickListener() {  
56         public void onClick(DialogInterface dialogInterface, int i) {  
57             dialogInterface.dismiss();  
58         }  
59     });  
60     create.show();  
61 }  
62 }
```

Nope...

That's not it. Try again.

OK



프리다를 활용한 모의해킹 - 암호 복호화

❖ 소스코드 확인 (m6a)

```
p000sg.vantagepoint.uncrackable1.MainActivity ✕ p000sg.vantagepoint.uncrackable1.C0005a ✕
7  /* renamed from: sg.vantagepoint.uncrackable1.a */
8  public class C0005a {
9      /* renamed from: a */
10     public static boolean m6a(String str) {
11         byte[] bArr;
12         String str2 = "8d127684cbc37c17616d806cf50473cc";
13         byte[] bArr2 = new byte[0];
14         try {
15             bArr = C0000a.m0a(m7b(str2), Base64.decode("5UJiFctbmgbDoLXmpL12mkno8HT4Lv8dlat8FxR2G0c=", 0));
16         } catch (Exception e) {
17             StringBuilder sb = new StringBuilder();
18             sb.append("AES error:");
19             sb.append(e.getMessage());
20             Log.d("CodeCheck", sb.toString());
21             bArr = bArr2;
22         }
23         return str.equals(new String(bArr));
24     }
```



프리다를 활용한 모의해킹 -암호 복호화

❖ 스크립트 작성

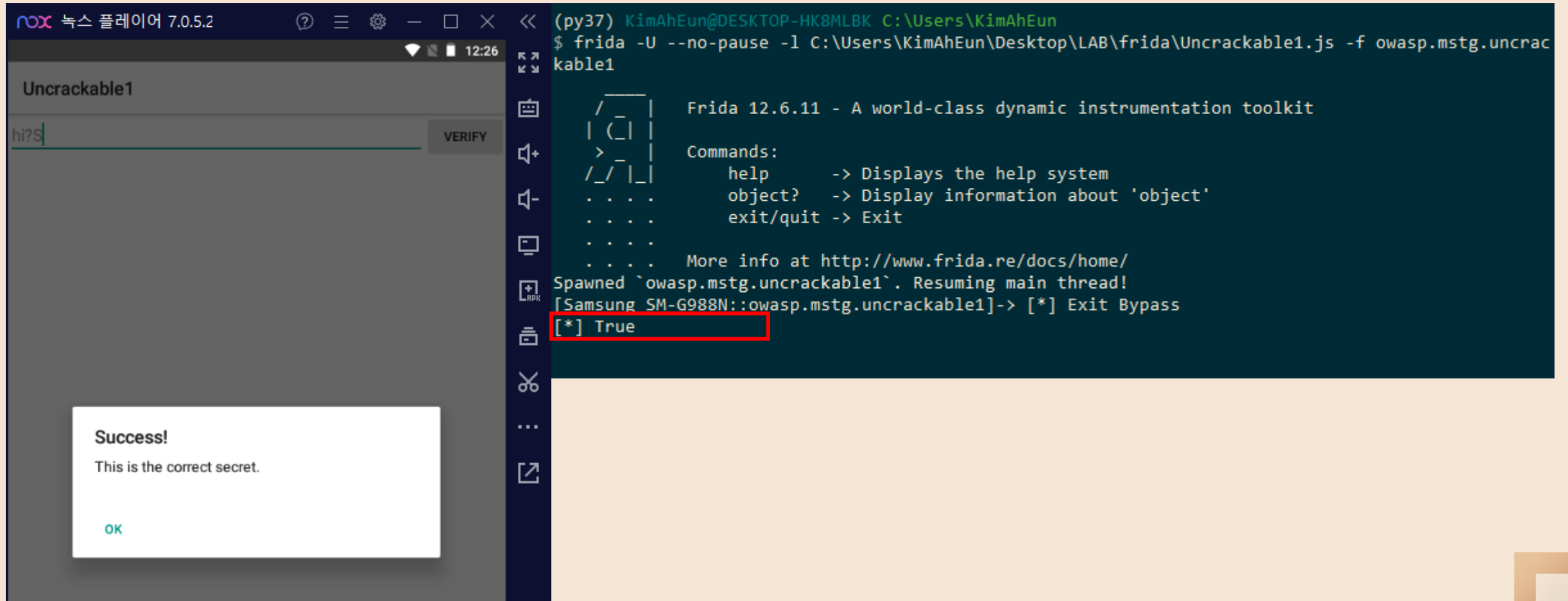
```
1  setImmediate(function() {  
2      Java.perform(function() {  
3          var System = Java.use("java.lang.Sysyme");  
4          System.exit.implementation = function() {  
5              console.log("[*] Exit Bypass");  
6          }  
7  
8          var trueClass = Java.use("p000sg.vantagepoint.uncrackable1.C0005a");  
9          trueClass.m6a.implementation = function(str) {  
10             console.log("[*] True");  
11             return true;  
12         }  
13     })  
14 })
```

→ m6a 함수가 콘솔로그 출력 및 항상 true 리턴하도록 재작성



프리다를 활용한 모의해킹 - 암호 복호화

❖ 스크립트 인젝션



nox 녹스 플레이어 7.0.5.2 12:26

Uncrackable1

hi?S VERIFY

Success!
This is the correct secret.
OK

```
(py37) KimAhEun@DESKTOP-HK8MLBK C:\Users\KimAhEun
$ frida -U --no-pause -l C:\Users\KimAhEun\Desktop\LAB\frida\Uncrackable1.js -f owasp.mstg.uncrackable1

/ _ |   Frida 12.6.11 - 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] Spawned `owasp.mstg.uncrackable1`. Resuming main thread!
[Samsung SM-G988N::owasp.mstg.uncrackable1]-> [*] Exit Bypass
[*] True
```

프리다를 활용한 모의해킹 - 암호 복호화

❖ 스크립트 작성 (실제 secret string 구하기)

```
1  setImmediate(function() {
2      Java.perform(function() {
3          var System = Java.use("java.lang.System");
4          System.exit.implementation = function() {
5              console.log("[*] Exit Bypass");
6          }
7
8          var decrypteClass = Java.use("sg.vantagepoint.a.a");
9          decrypteClass.a.implementation = function(arr1, arr2) {
10             var secret_string = this.a(arr1, arr2); // C0000a.m0a로 bArr 얻기
11             var str = "";
12             for(var i=0; i<secret_string.length; i++) { // 배열이라 for문으로 출력
13                 str += String.fromCharCode(secret_string[i]);
14             }
15             console.log("[*] Decrypted: " + str);
16             return secret_string;
17         }
18     })
19 })
```

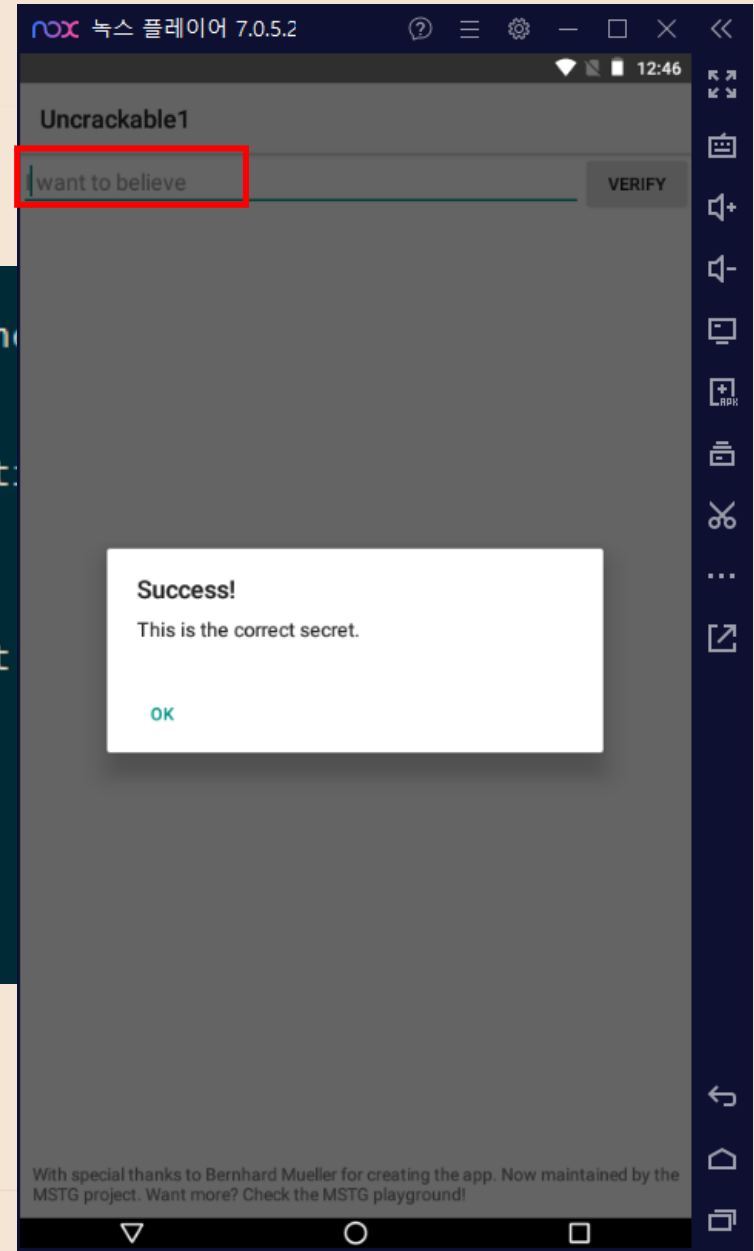


프리다를 활용한 모의해킹 - 암호 복호화

❖ 스크립트 작성 (실제 secret string 구하기)

```
(py37) KimAhEun@DESKTOP-HK8MLBK C:\Users\KimAhEun
$ frida -U --no-pause -l C:\Users\KimAhEun\Desktop\LAB\frida\Uncr
kable1

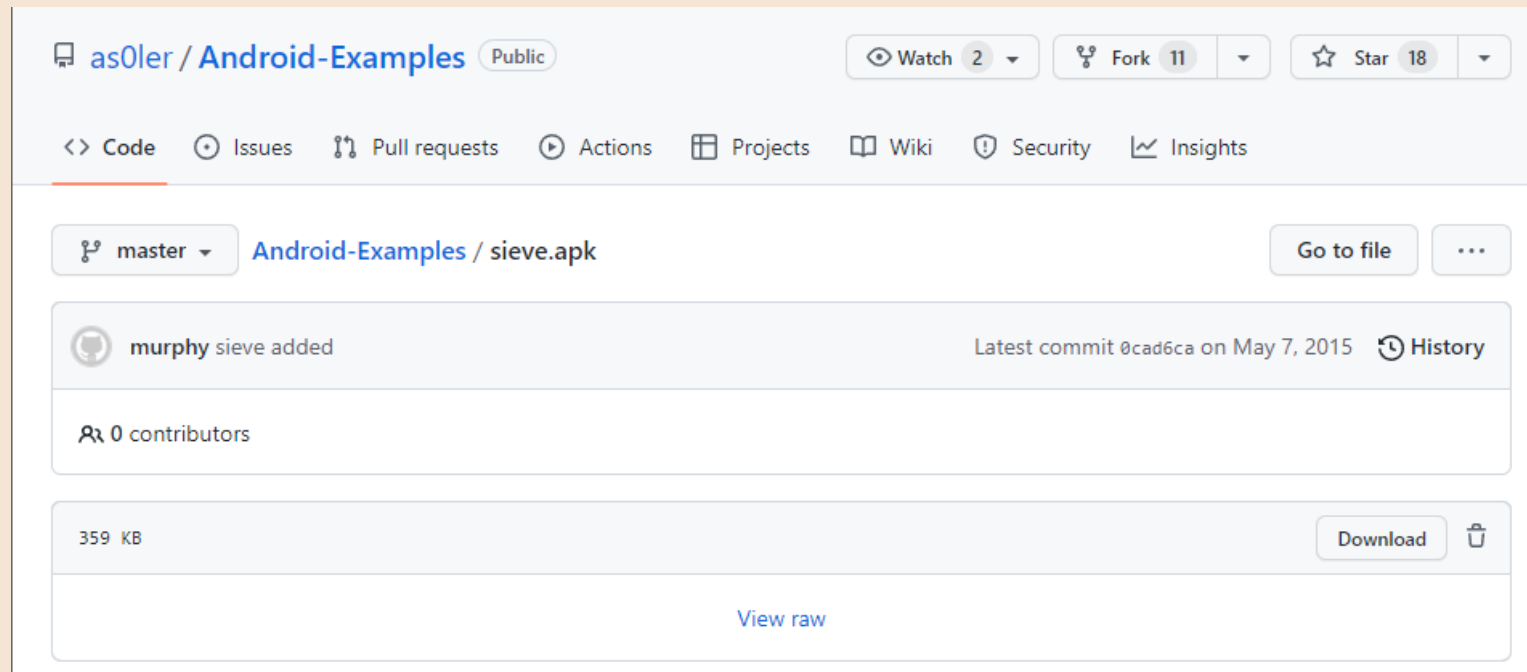
/ _ |   Frida 12.6.11 - A world-class dynamic instrumentat
| ( |   |
> _ |   |
/_/_|_ |   |
. . . .   |   |
. . . .   |   |
. . . .   |   |
. . . .   |   |
. . . .   |   |
More info at http://www.frida.re/docs/home/
Spawned `owasp.mstg.uncrackable1`. Resuming main thread!
[Samsung SM-G988N::owasp.mstg.uncrackable1]-> [*] Exit Bypass
[*] Decrypted: I want to believe
```



프리다를 활용한 모의해킹

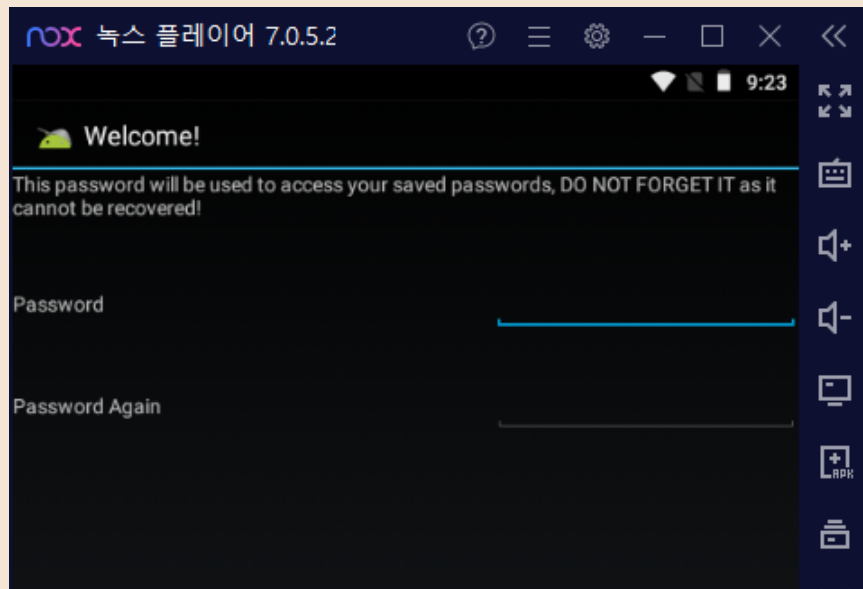
❖ sieve.apk

- 링크 : <https://github.com/as0ler/Android-Examples/blob/master/sieve.apk>
- 앱 첫 접속 시 master password 설정 → 4자리의 pin 설정
→ 특정 서비스의 계정 정보를 등록할 수 있음

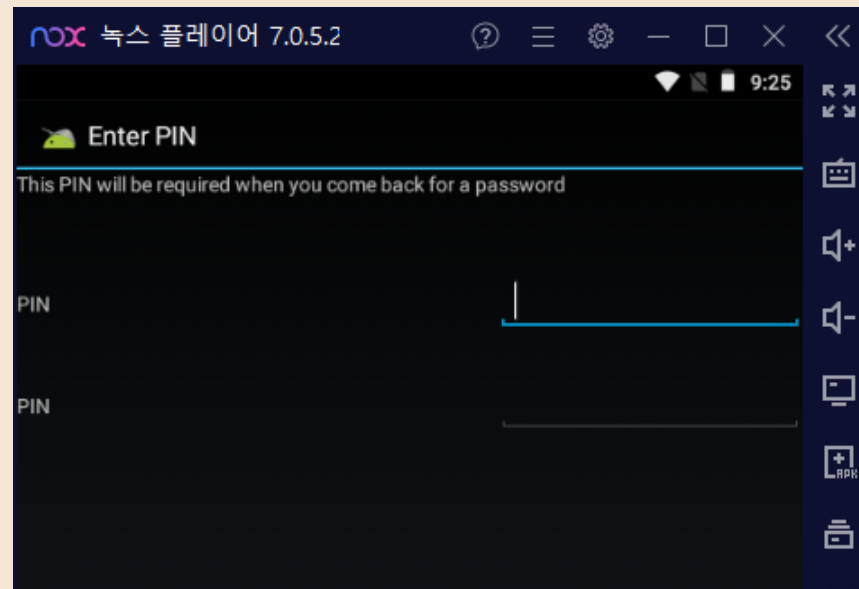


프리다를 활용한 모의해킹 -로그인 우회

❖ 앱 동작 과정



① password 설정

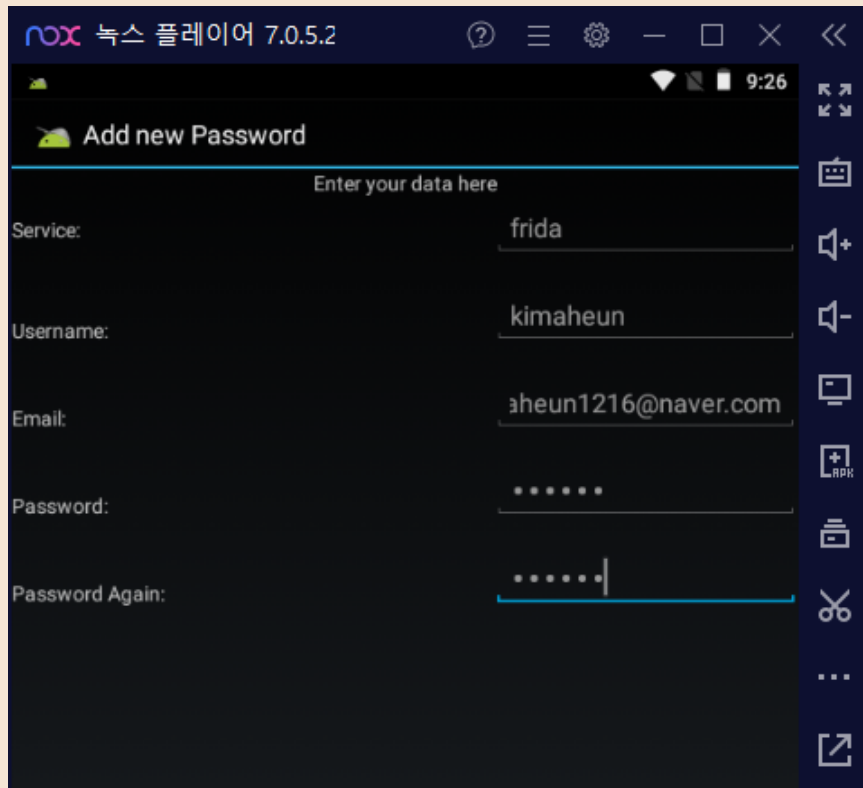


② pin 설정

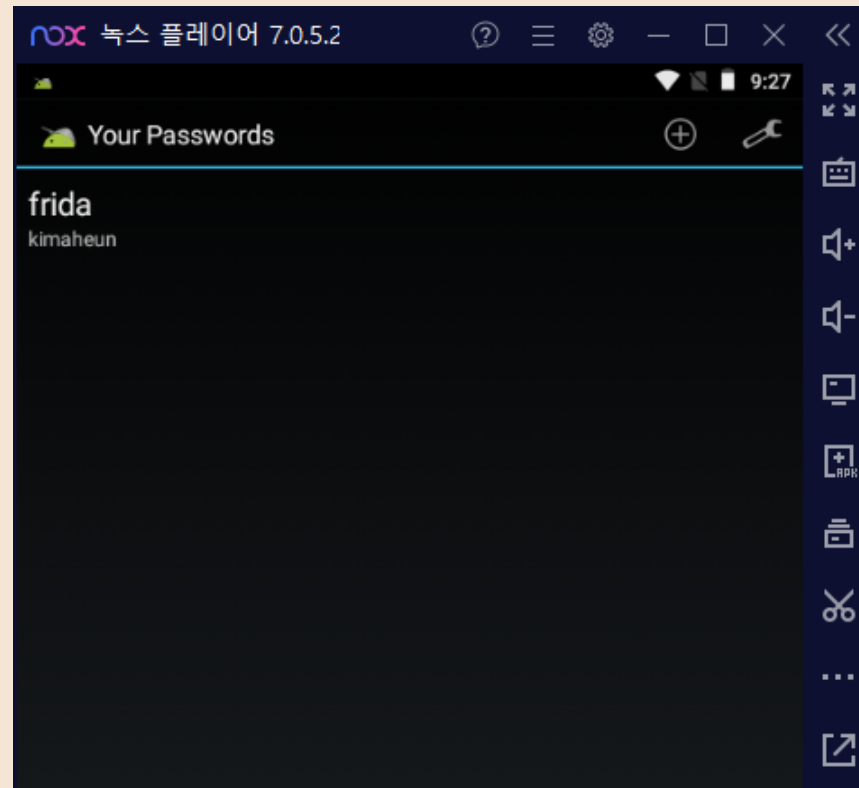


프리다를 활용한 모의해킹 -로그인 우회

❖ 앱 동작 과정



③ 서비스별 계정정보 등록



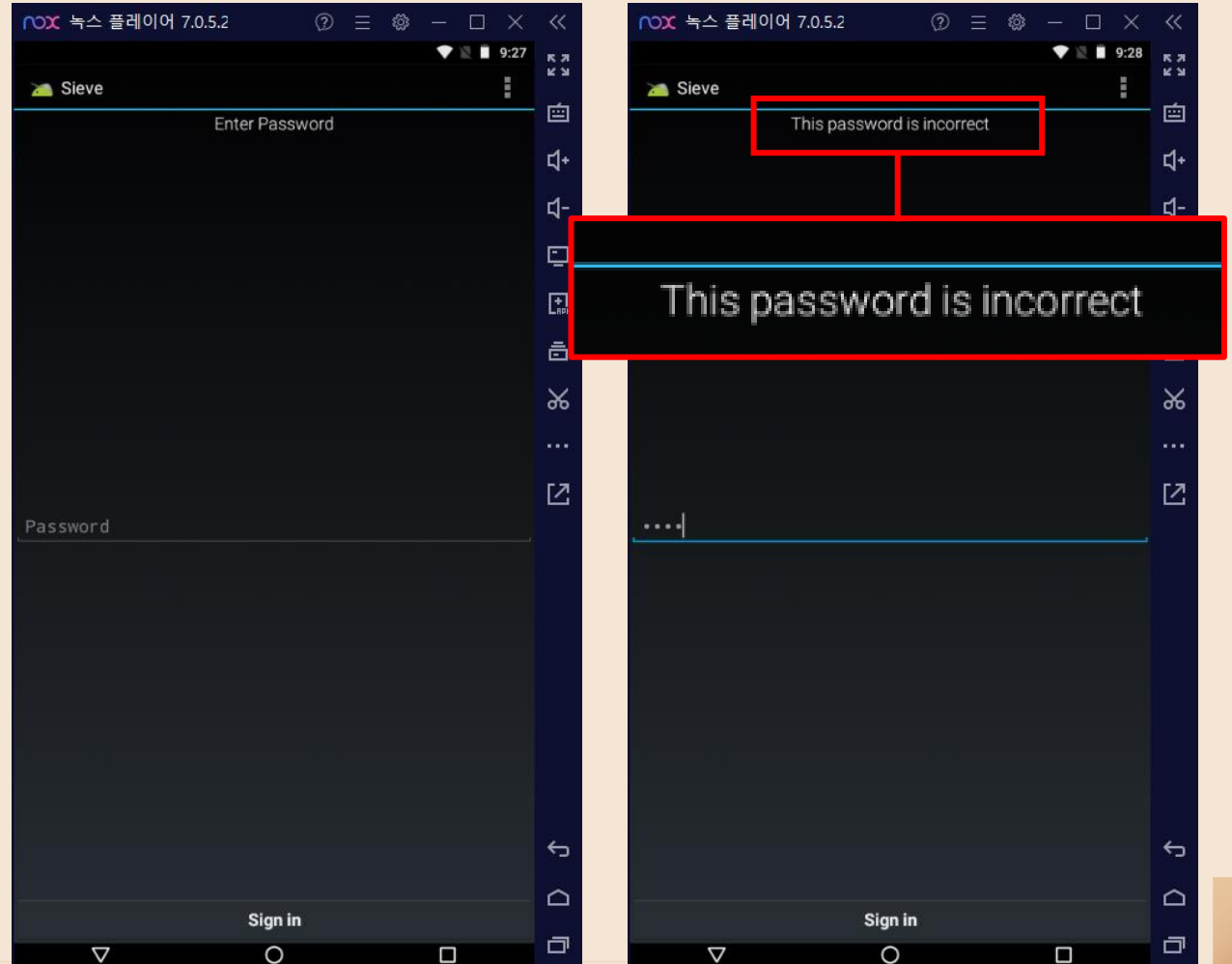
④ 계정정보 목록 출력



프리다를 활용한 모의해킹 -로그인 우회

❖ 로그인 시도

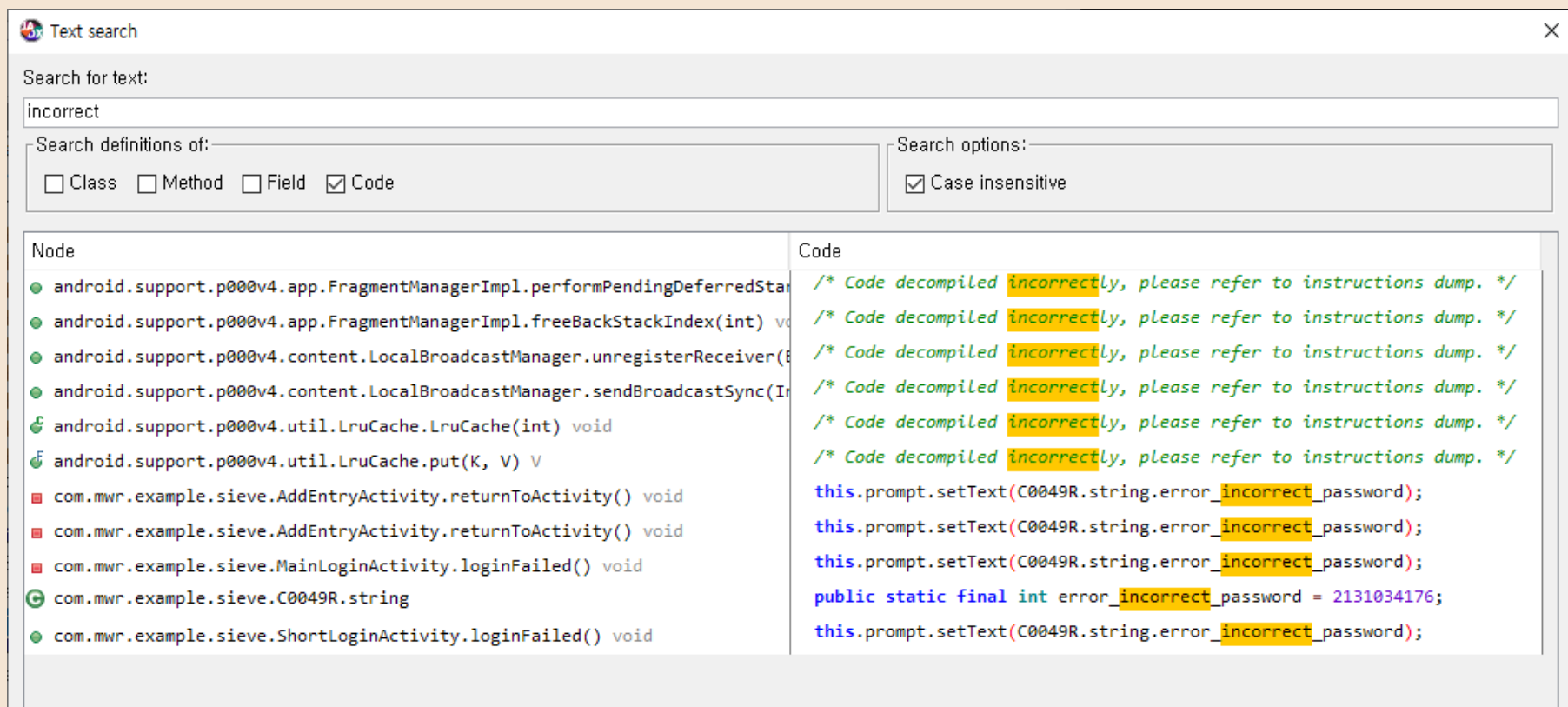
- 현재 설정된 비밀번호 : bcglabaheunkim123
- '1234' 입력 후 로그인을 시도하면,
상단에 'This password is incorrect' 출력됨



프리다를 활용한 모의해킹 -로그인 우회

❖ 소스코드 분석

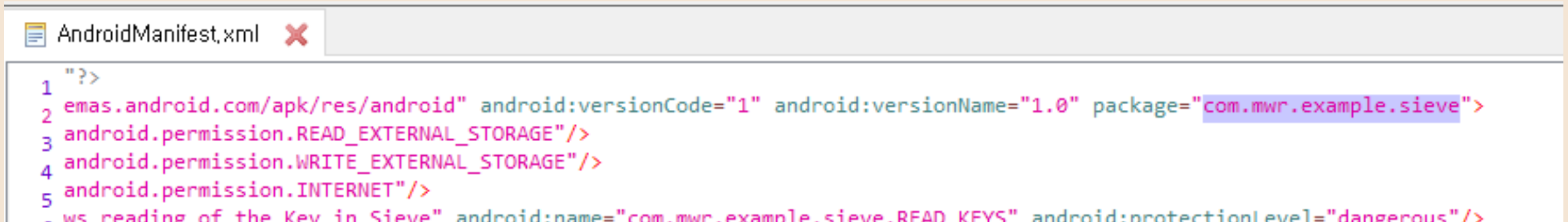
- 화면 상단에 출력되었던 문자열을 검색해보았으나 정확한 내용을 찾을 수 없었음



프리다를 활용한 모의해킹 -로그인 우회

❖ 소스코드 분석 (AndroidManifest.xml)

- AndroidManifest.xml 파일에서 패키지의 이름을 찾은 뒤, 해당 패키지 내에 있는 메인 액티비티 확인
- 패키지 이름 : com.mwr.example.sieve



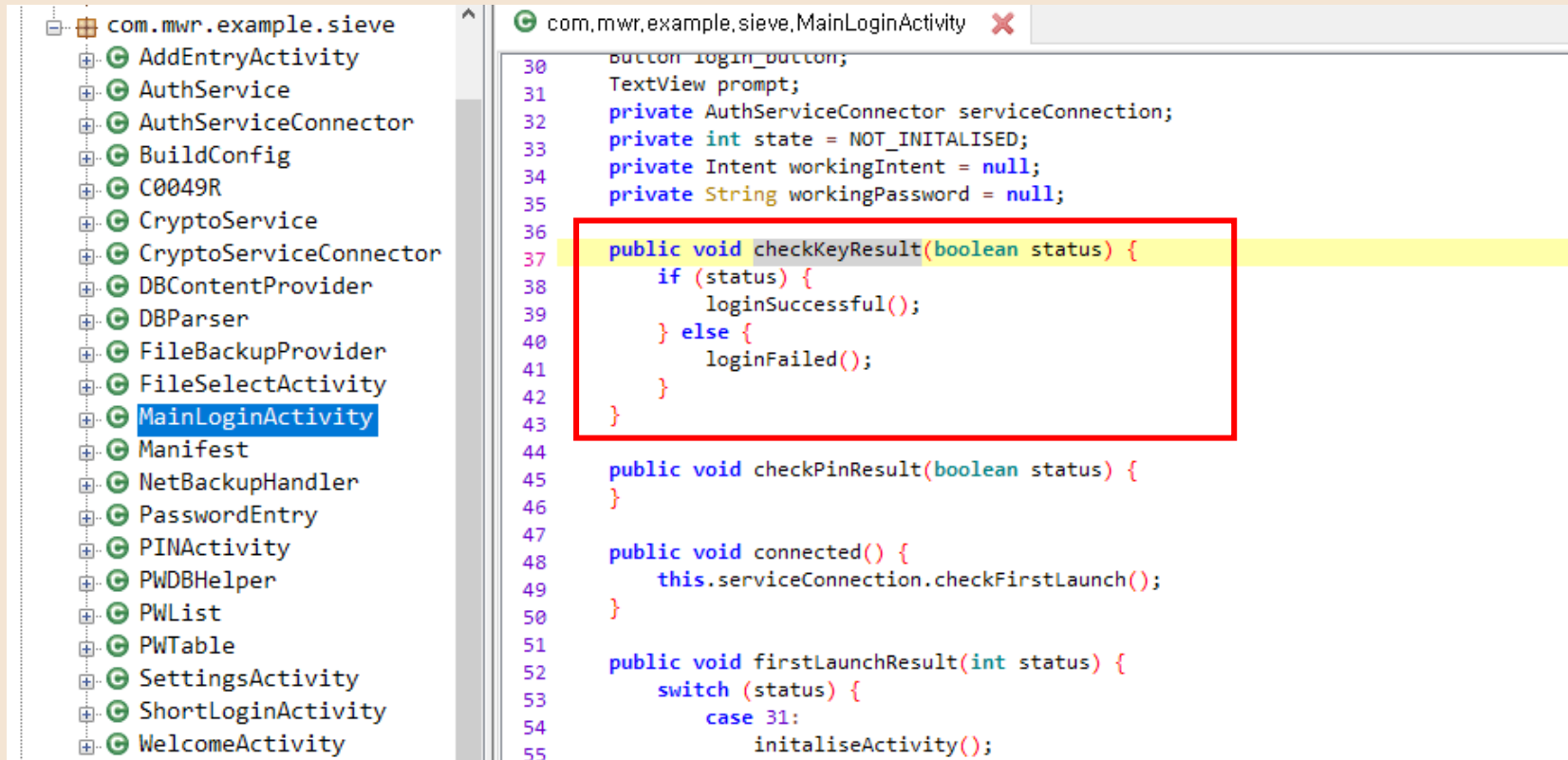
```
1  ">
2  mas.android.com/apk/res/android" android:versionCode="1" android:versionName="1.0" package="com.mwr.example.sieve">
3  android.permission.READ_EXTERNAL_STORAGE"/>
4  android.permission.WRITE_EXTERNAL_STORAGE"/>
5  android.permission.INTERNET"/>
6  ws reading of the Key in Sieve" android:name="com.mwr.example.sieve.READ_KEYS" android:protectionLevel="dangerous"/>
```



프리다를 활용한 모의해킹 -로그인 우회

❖ 소스코드 분석 (checkKeyResult)

- AndroidManifest.xml 파일에서 패키지의 이름을 찾은 뒤, 해당 패키지 내에 있는 메인 액티비티 확인
- 메인 액티비티로 예상되는 클래스 : MainLoginActivity



```
com.mwr.example.sieve
├── AddEntryActivity
├── AuthService
├── AuthServiceConnector
├── BuildConfig
├── C0049R
├── CryptoService
├── CryptoServiceConnector
├── DBContentProvider
├── DBParser
├── FileBackupProvider
├── FileSelectActivity
├── MainLoginActivity
├── Manifest
├── NetBackupHandler
├── PasswordEntry
├── PINActivity
├── PWDBHelper
├── PWList
├── PWTable
├── SettingsActivity
├── ShortLoginActivity
└── WelcomeActivity

com.mwr.example.sieve.MainLoginActivity
30 button login_button;
31 TextView prompt;
32 private AuthServiceConnector serviceConnection;
33 private int state = NOT_INITIALISED;
34 private Intent workingIntent = null;
35 private String workingPassword = null;
36
37 public void checkKeyResult(boolean status) {
38     if (status) {
39         loginSuccessful();
40     } else {
41         loginFailed();
42     }
43 }
44
45 public void checkPinResult(boolean status) {
46 }
47
48 public void connected() {
49     this.serviceConnection.checkFirstLaunch();
50 }
51
52 public void firstLaunchResult(int status) {
53     switch (status) {
54         case 31:
55             initialiseActivity();
```



프리다를 활용한 모의해킹 -로그인 우회

❖ 스크립트 작성

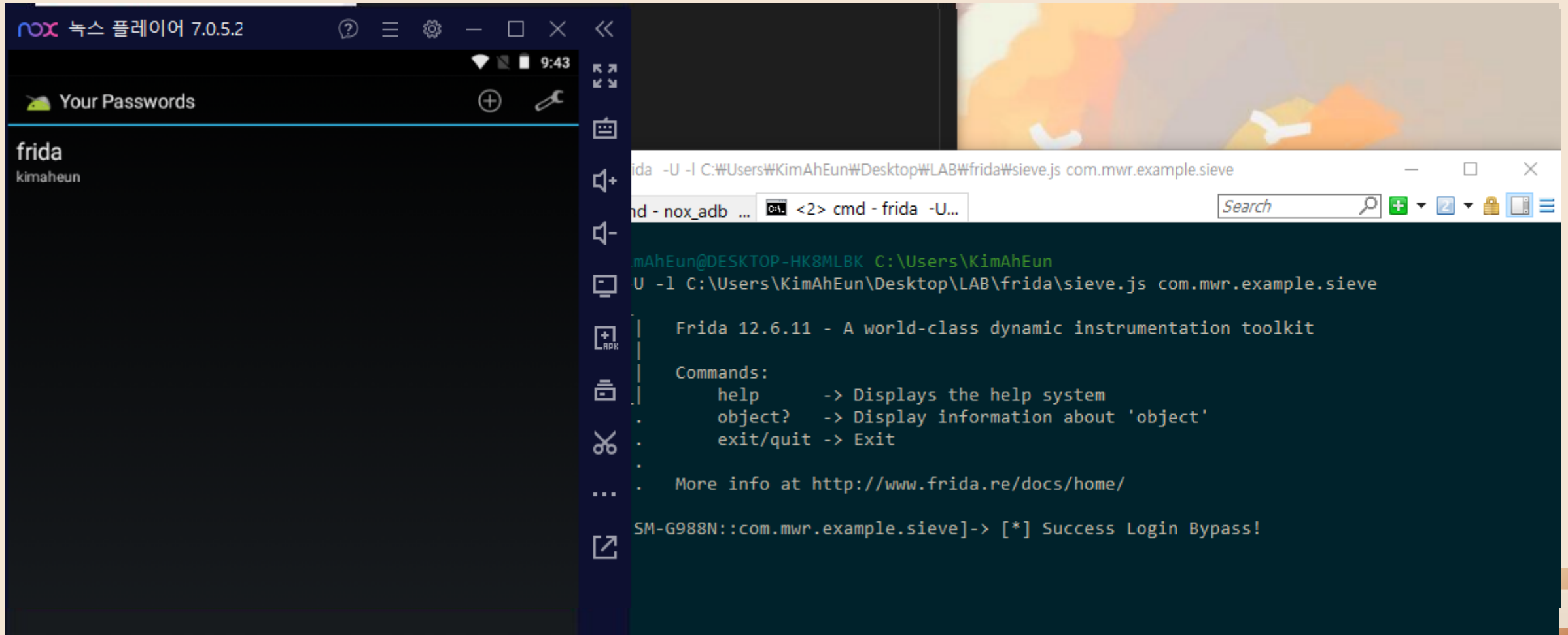
- AndroidManifest.xml 파일에서 패키지의 이름을 찾은 뒤, 해당 패키지 내에 있는 메인 액티비티 확인
- 메인 액티비티로 예상되는 클래스 : MainLoginActivity

```
1  setImmediate(function() {  
2      Java.perform(function() {  
3          var loginBypass = Java.use("com.mwr.example.sieve.MainLoginActivity");  
4          loginBypass.checkKeyResult.implementation = function(arg) {  
5              console.log("[*] Success Login Bypass!");  
6              this.checkKeyResult(true);  
7          }  
8      })  
9  })
```



프리다를 활용한 모의해킹 -로그인 우회

❖ 스크립트 인젝션

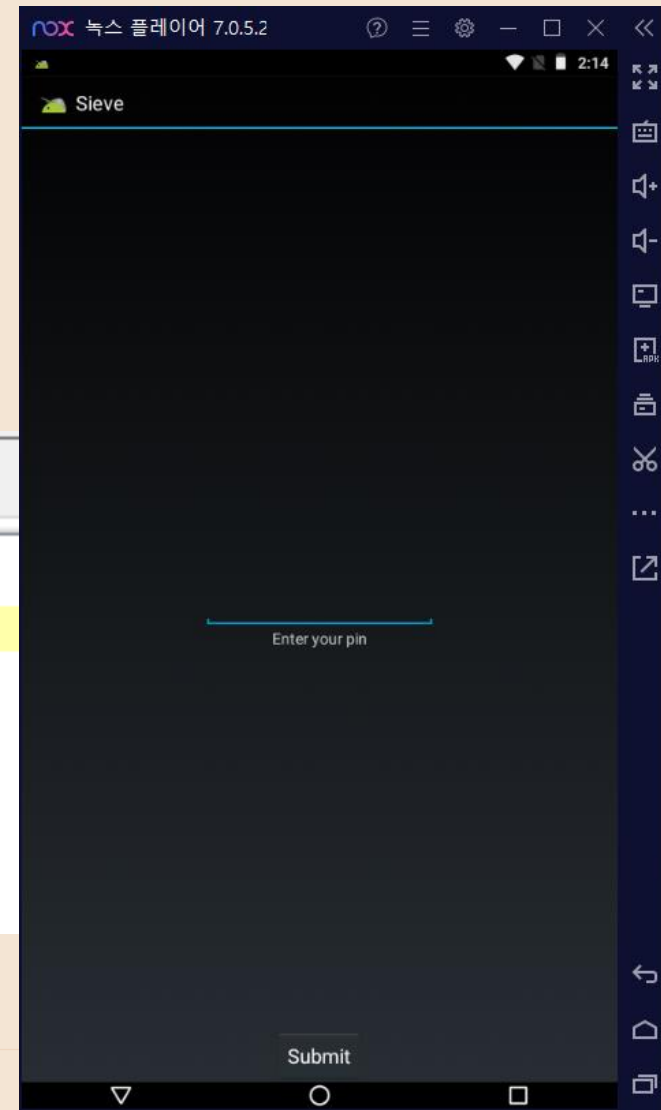


프리다를 활용한 모의해킹 -로그인 브루트포스

❖ 소스코드 분석 (checkPinResult)

- key와 비슷하게, checkPinResult 함수 검색
→ ShortLoginActivity 클래스에
pin에 관련된 메서드들 존재

```
com.mwr.example.sieve.ShortLoginActivity ✕  
105  
106  
107 public void checkPinResult(boolean status) {  
108     if (status) {  
109         loginSuccessful();  
110     } else {  
111         loginFailed();  
112     }  
113 }
```



프리다를 활용한 모의해킹 -로그인 브루트포스

❖ 소스코드 분석 (submit)

- 입력값 workingPIN에 저장

→ serviceConnection.checkPin에서 pin 검증을 수행하는 것으로 예상됨

com.mwr.example.sieve.ShortLoginActivity

```
65
66     public void submit(View view) {
67         this.workingPIN = this.pwEntry.getText().toString();
68         Log.d(TAG, "user has entered a pin: " + this.workingPIN);
69         this.serviceConnection.checkPin(this.workingPIN);
70         this.submitButton.setEnabled(false);
71     }
```



프리다를 활용한 모의해킹 -로그인 브루트포스

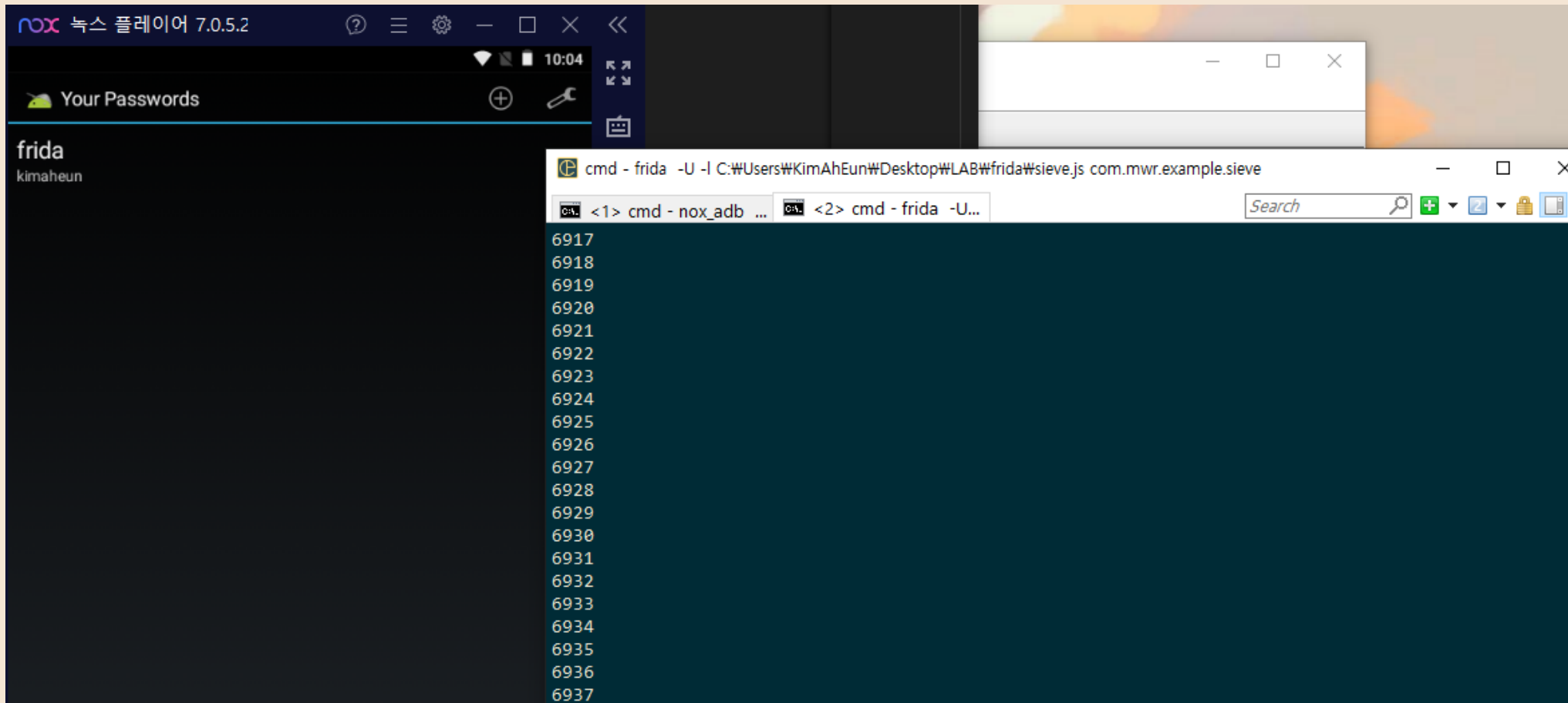
❖ 소스코드 인젝션

```
8
9      // PIN 네자리 만들기
10     function pinToFour(number) {
11         if(number<=9999) {
12             number = ("000" + number).slice(-4); //0001 -> 0001, 000333 -> 033
13             return String(number);
14         }
15     }
16
17     var pinBypass = Java.use("com.mwr.example.sieve.ShortLoginActivity");
18     pinBypass.submit.implementation = function(arg) {
19         var service = this.serviceConnection.value; //serviceConnection 메서드
20         for(var i=0; i<9999; i++) {
21             service.checkPin(pinToFour(i));
22             console.log(pinToFour(i));
23         }
24     }
25 })
26 })
```



프리다를 활용한 모의해킹 -로그인 브루트포스

❖ 소스코드 인젝션



프리다를 활용한 모의해킹 -SSL Pinning

❖ SSLPinning

- SSL 통신에서 취약하다고 알려져 있는 중간자 공격을 방지할 수 있는 기법
- 클라이언트 측에 신뢰할 수 있는 인증서를 저장하고, 이후 실제 통신 과정에서 서버가 제공하는 인증서와 비교
→ 두 인증서가 일치하지 않으면 연결은 중단되며 유저의 정보가 서버로 전송되지 않음
- 즉, 클라이언트 측에 설치되는 다른 인증서로는 통신이 되지 않고 서버에서 제공하는 인증서로만 통신할 수 있음
- SSL Pinning이 적용된 클라이언트에서 보내는 패킷은 중간에서 패킷을 가로챌 수 없음



프리다를 활용한 모의해킹 -SSL Pinning

❖ Burp suite로 패킷 확인

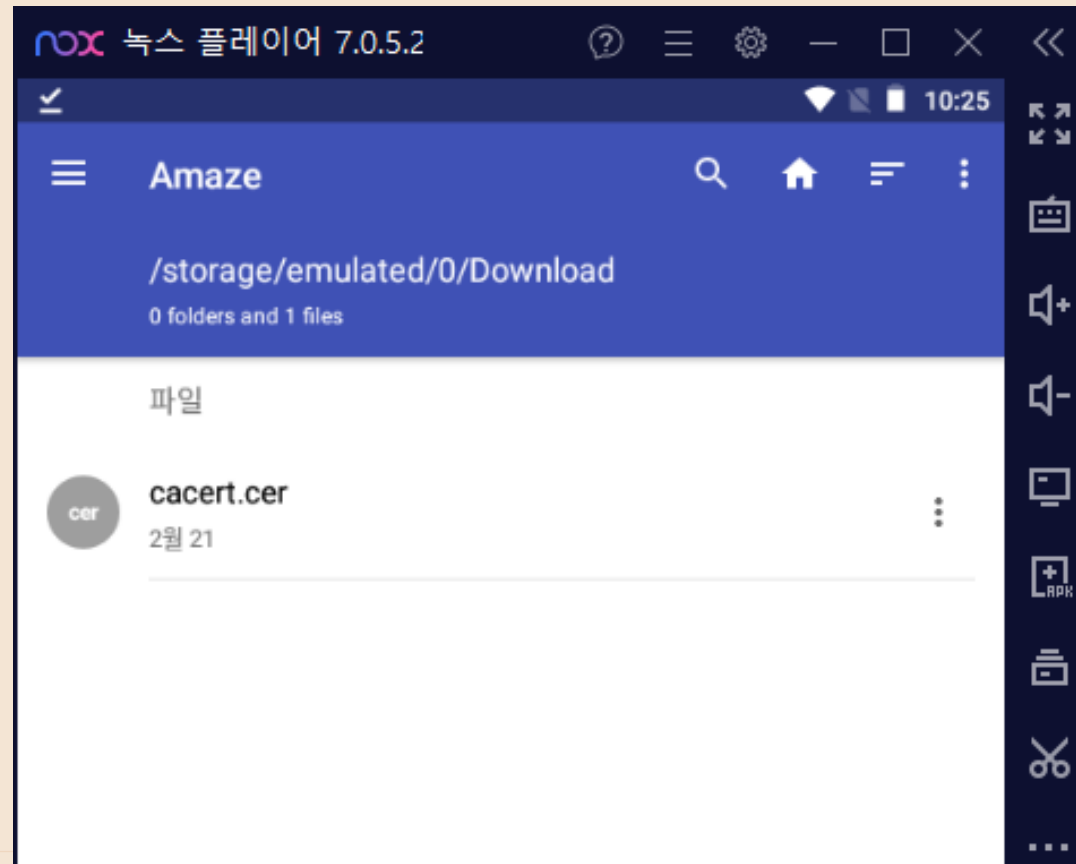
- http의 패킷들만 보임

Burp Suite Community Edition v2023.1.2 - Temporary Project											
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extensions Learn Deserialization Scanner											
Intercept HTTP history WebSockets history Proxy settings											
Filter: Hiding CSS, image and general binary content											
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title	
8	http://google.com	GET	/			301	547	HTML		301 Moved	
6	http://update.googleapis.com	POST	/service/update2/json	✓		200	850	JSON			
4	http://update.googleapis.com	POST	/service/update2/json	✓		200	850	JSON			
2	http://update.googleapis.com	POST	/service/update2/json?cup2key=12:u...	✓		200	7579	JSON			
1	http://heqoo.net	GET	/files/attach/xeicon/mobicon.png			301	506	HTML	png	301 Moved Permanently	



프리다를 활용한 모의해킹 -SSL Pinning

- ❖ 클라이언트(Nox) 측에 인증서 설치
 - burp suite 인증서 설치 → cacert.der



프리다를 활용한 모의해킹 -SSL Pinning

❖ 인증서 설치 후 Burp suite로 패킷 확인

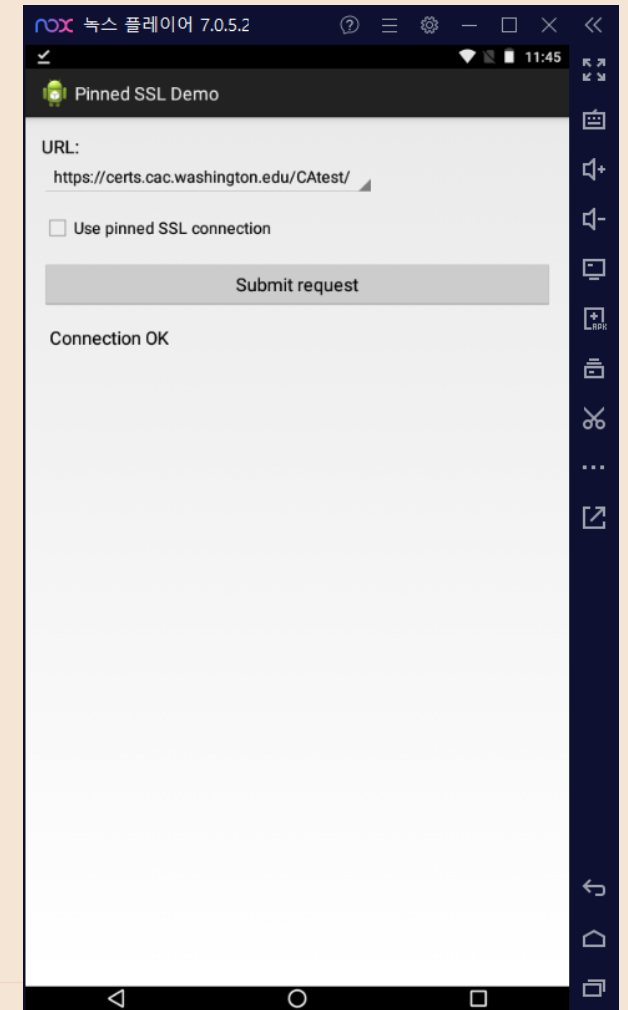
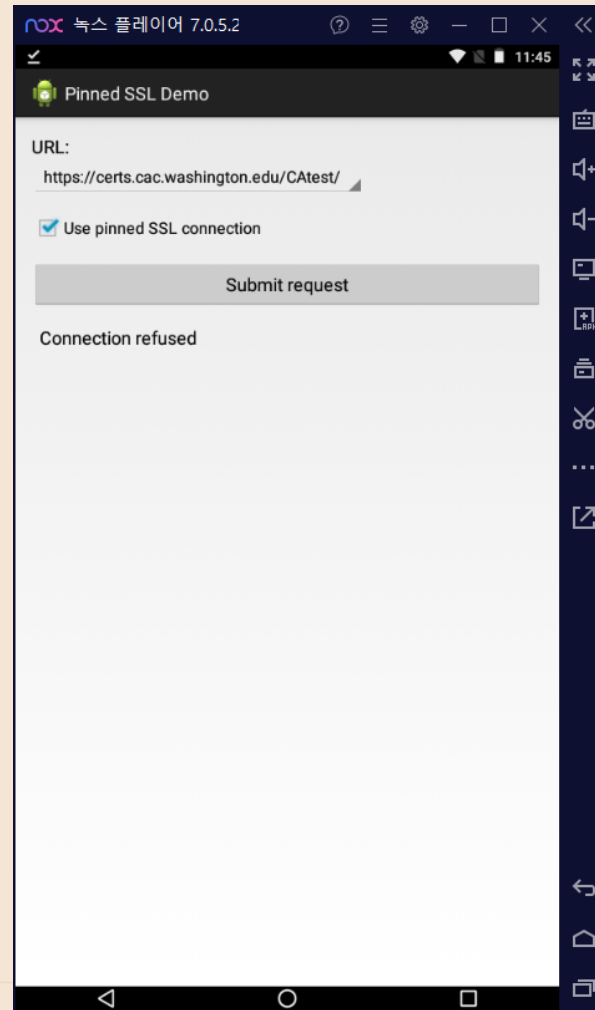
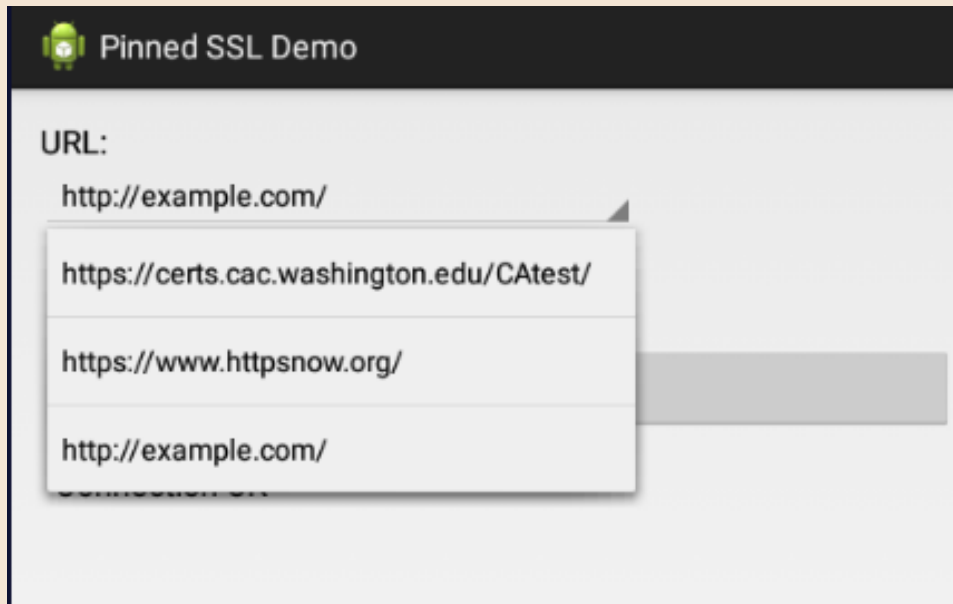
- https의 패킷도 확인됨

Burp Suite Community Edition v2023.1.2 - Temporary Project												
Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Logger Extensions Learn Deserialization Scanner												
Intercept HTTP history WebSockets history Proxy settings												
Filter: Hiding CSS, image and general binary content												
#	Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension	Title		
13	https://beacons.gcp.gvt2.com	POST	/domainreliability/upload	✓		200	1152	script				
12	https://beacons.gcp.gvt2.com	POST	/domainreliability/upload	✓		200	1195	script				
9	http://update.googleapis.com	POST	/service/update2/json	✓		200	850	JSON				
8	http://google.com	GET	/			301	547	HTML		301 Moved		
6	http://update.googleapis.com	POST	/service/update2/json	✓		200	850	JSON				
4	http://update.googleapis.com	POST	/service/update2/json	✓		200	850	JSON				
2	http://update.googleapis.com	POST	/service/update2/json?cup2key=12:u...	✓		200	7579	JSON				
1	http://theqoo.net	GET	/files/attach/xeicon/mobicon.png			301	506	HTML	png	301 Moved Permanently		



프리다를 활용한 모의해킹 -SSL Pinning

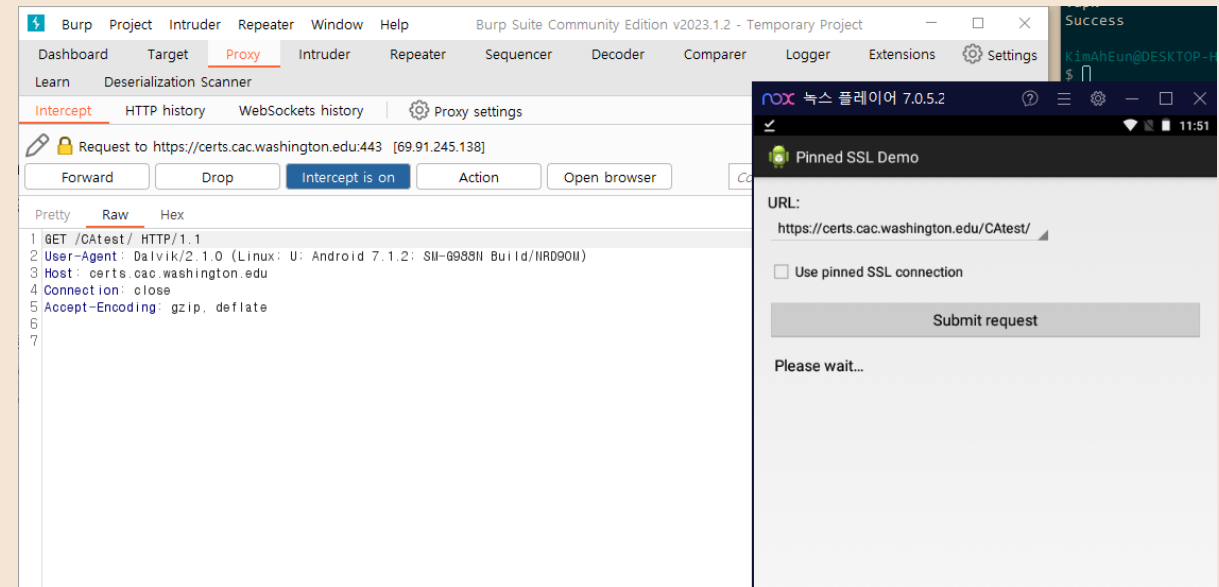
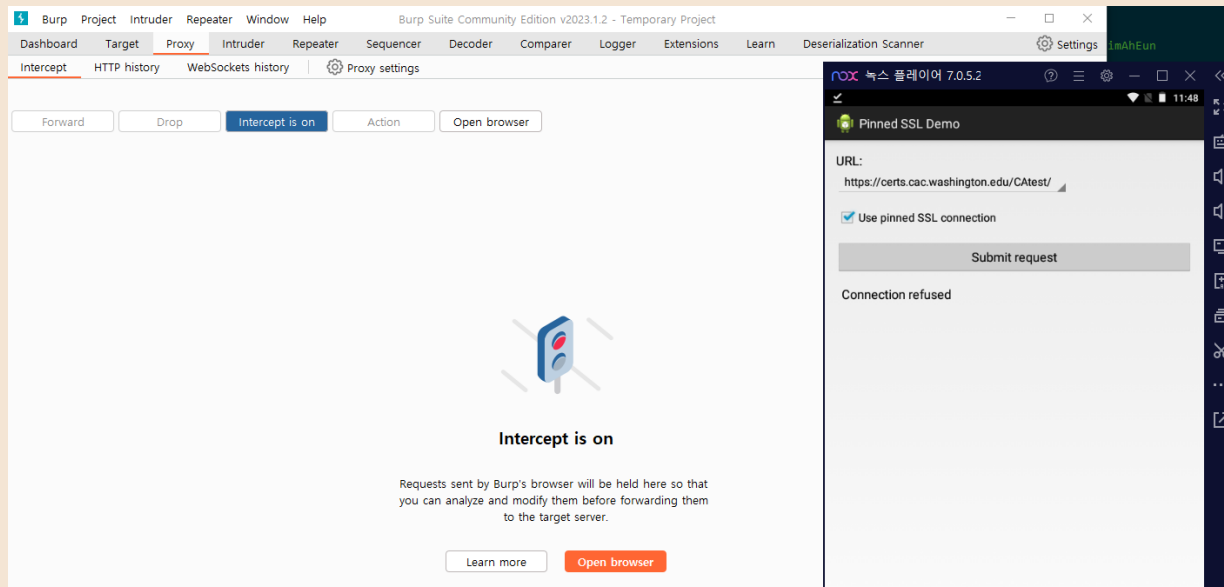
- ❖ android-ssl-pinning.apk 동작
 - SSL 핀 연결 사용 → 연결 X
 - SSL 핀 연결 안함 → 연결 O



프리다를 활용한 모의해킹 -SSL Pinning

❖ android-ssl-pinning.apk 동작

- SSL 핀 연결 사용 → 연결 X
- SSL 핀 연결 안함 → 연결 O



프리다를 활용한 모의해킹 -SSL Pinning

❖ frida codeshare

- frida에서 사용할 수 있는 여러 스크립트 코드 공유 사이트

Most Popular Projects

Universal Android SSL Pinning Bypass with Frida

👍 61 | 👁 221K

Uploaded by: [@pcipolloni](#)

Android SSL Re-Pinning, more information can be found here
<https://techblog.mediaservice.net/2017/07/universal-android-ssl-pinning-bypass-with-frida/>

[PROJECT PAGE](#)

frida-multiple-unpinning

👍 34 | 👁 60K

Uploaded by: [@akabe1](#)

Another Android ssl certificate pinning bypass script for various methods
(<https://gist.github.com/akabe1/5632cbc1cd49f0237cbd0a93bc8e4452>)

[PROJECT PAGE](#)

fridantiroot

👍 24 | 👁 72K

Uploaded by: [@dzonerzy](#)

aesinfo

👍 10 | 👁 23K

Uploaded by: [@dzonerzy](#)



프리다를 활용한 모의해킹 -SSL Pinning

❖ frida codeshare

- frida에서 사용할 수 있는 여러 스크립트 코드 공유 사이트

```
11
12 ~ setTimeout(function(){
13 ~   Java.perform(function (){
14     console.log("");
15     console.log("[.] Cert Pinning Bypass/Re-Pinning");
16
17     var CertificateFactory = Java.use("java.security.cert.CertificateFactory");
18     var FileInputStream = Java.use("java.io.FileInputStream");
19     var BufferedInputStream = Java.use("java.io.BufferedInputStream");
20     var X509Certificate = Java.use("java.security.cert.X509Certificate");
21     var KeyStore = Java.use("java.security.KeyStore");
22     var TrustManagerFactory = Java.use("javax.net.ssl.TrustManagerFactory");
23     var SSLContext = Java.use("javax.net.ssl.SSLContext");
24
25     // Load CAs from an InputStream
26     console.log("[+] Loading our CA...")
27     var cf = CertificateFactory.getInstance("X.509");
28
29 ~   try {
30     var fileInputStream = FileInputStream.$new("/data/local/tmp/cert-der.crt");
31   }
32 ~   catch(err) {
33     console.log("[o] " + err);
34   }
35
36 ~   var bufferedInputStream = BufferedInputStream.$new(fileInputStream);
37   var ca = cf.generateCertificate(bufferedInputStream);
38   bufferedInputStream.close();
39
40   var certInfo = Java.cast(ca, X509Certificate);
41   console.log("[o] Our CA Info: " + certInfo.getSubjectDN());
42
43   // Create a KeyStore containing our trusted CAs
44   console.log("[+] Creating a KeyStore for our CA...");
45   var keyStoreType = KeyStore.getDefaultType();
```



프리다를 활용한 모의해킹 -SSL Pinning

❖ frida codeshare 스크립트 실행

The image shows a mobile emulator interface on the left and a terminal window on the right. The emulator, titled 'nox 녹스 플레이어 7.0.5.2', displays a 'Pinned SSL Demo' app. The app's URL is 'https://certs.cac.washington.edu/CAtest/' and the checkbox 'Use pinned SSL connection' is checked. A 'Submit request' button is visible, and the status 'Connection OK' is shown at the bottom.

The terminal window on the right shows the execution of the following command:

```
cmd - frida -U --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida --no-pause -f smuldr.sslpin
```

The output of the command is as follows:

```
(py37) KimAhEun@DESKTOP-HK8MLBK C:\Users\KimAhEun
$ frida -U --codeshare pcipolloni/universal-android-ssl-pinning-bypass-with-frida --no-pause -f smuldr.sslpin

  / _ _ |   Frida 12.6.11 - 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/

Spawned `smuldr.sslpin`. Resuming main thread!
[Samsung SM-G988N::smuldr.sslpin]->
[.] Cert Pinning Bypass/Re-Pinning
[+] Loading our CA...
[o] Our CA Info: CN=PortSwigger CA, OU=PortSwigger CA, O=PortSwigger, L=PortSwigger, ST=PortSwigger, C=PortSwigger
[+] Creating a KeyStore for our CA...
[+] Creating a TrustManager that trusts the CA in our KeyStore...
[+] Our TrustManager is ready...
[+] Hijacking SSLContext methods now...
[-] Waiting for the app to invoke SSLContext.init()...
[o] App invoked javax.net.ssl.SSLContext.init...
[+] SSLContext initialized with our custom TrustManager!
```

프리다를 활용한 모의해킹 -SSL Pinning

❖ 소스코드 분석 (onSubmit)

- pinning에 체크 여부 확인 → getPinnedSSLContext로 sslContext 설정 → 연결 확인

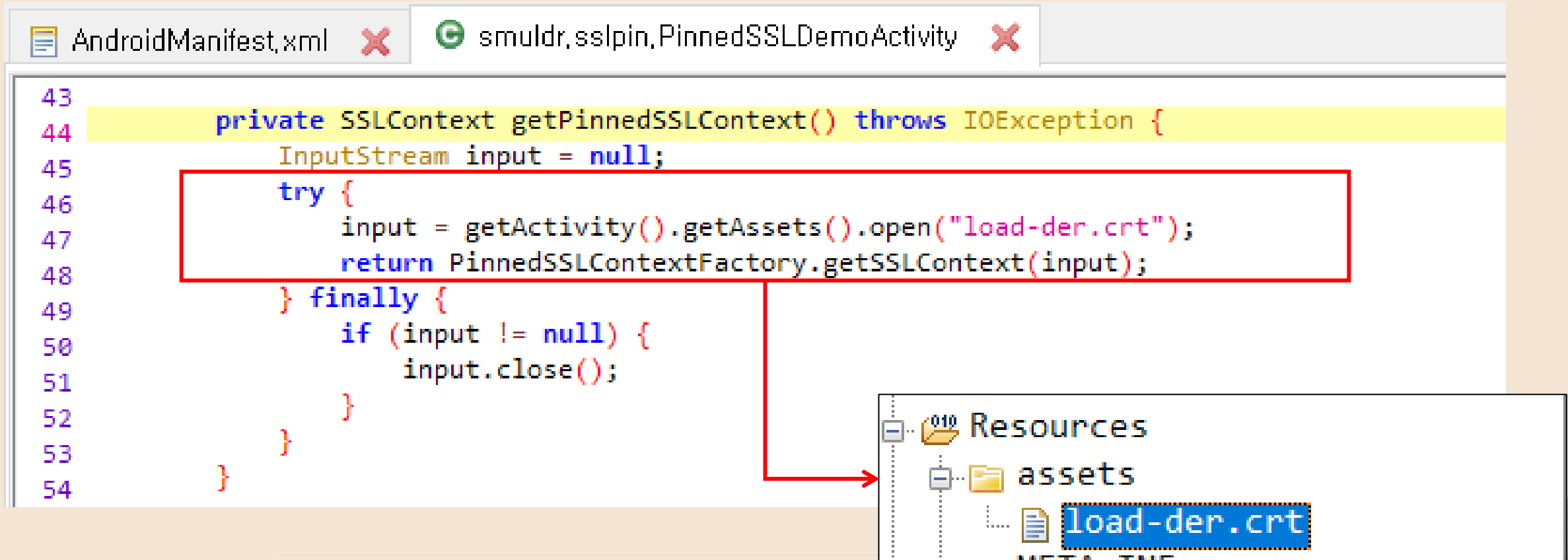
```
51         input.close();
52     }
53 }
54
55 /* access modifiers changed from: private */
56 public void onSubmit() {
57     this.mResultView.setText(getString(C0112R.string.loading));
58     SSLContext sslContext = null;
59     if (this.mEnablePinning.isChecked()) {
60         try {
61             sslContext = getPinnedSSLContext();
62         } catch (IOException e) {
63             Log.e(PinnedSSLDemoActivity.TAG, "Failed create pinned SSL context", e);
64         }
65     }
66     String url = (String) this.mUrlsSpinner.getSelectedItem();
67     new TestConnectionTask(sslContext, this).execute(new String[]{url});
68 }
69
```



프리다를 활용한 모의해킹 -SSL Pinning

❖ 소스코드 분석 (getPinnedSSLContext)

- load-der.crt 인증서 확인 → burp suite 인증서 경로로 변조하면 ssl pinning 우회가 되지 않을까?



The screenshot displays an IDE interface with two tabs at the top: 'AndroidManifest.xml' and 'smuldr, sslpin, PinnedSSLDemoActivity'. The main editor shows the following Java code for the `getPinnedSSLContext()` method:

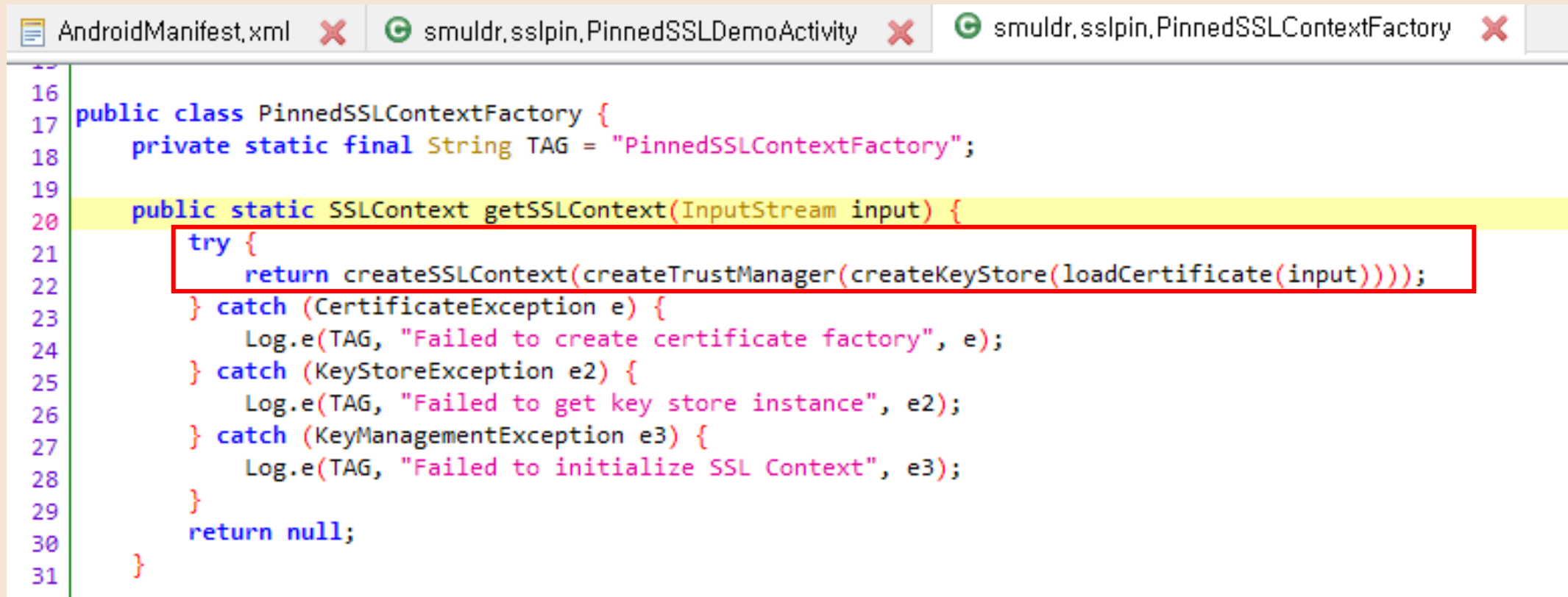
```
43 private SSLContext getPinnedSSLContext() throws IOException {  
44     InputStream input = null;  
45     try {  
46         input = getActivity().getAssets().open("load-der.crt");  
47         return PinnedSSLContextFactory.getSSLContext(input);  
48     } finally {  
49         if (input != null) {  
50             input.close();  
51         }  
52     }  
53 }  
54
```

A red rectangular box highlights the code block from line 46 to 48. A red arrow originates from this box and points to a file explorer view in the bottom right corner. The file explorer shows the 'Resources' directory containing an 'assets' subdirectory, which in turn contains the file 'load-der.crt'.

프리다를 활용한 모의해킹 -SSL Pinning

❖ 소스코드 분석 (getSSLContext)

- return `createSSLContext(createTrustManager(createKeyStore(loadCertificate(input))))`;



```
AndroidManifest.xml ✕ smuldr,sslpin,PinnedSSLDemoActivity ✕ smuldr,sslpin,PinnedSSLContextFactory ✕
16 public class PinnedSSLContextFactory {
17     private static final String TAG = "PinnedSSLContextFactory";
18
19     public static SSLContext getSSLContext(InputStream input) {
20         try {
21             return createSSLContext(createTrustManager(createKeyStore(loadCertificate(input))));
22         } catch (CertificateException e) {
23             Log.e(TAG, "Failed to create certificate factory", e);
24         } catch (KeyStoreException e2) {
25             Log.e(TAG, "Failed to get key store instance", e2);
26         } catch (KeyManagementException e3) {
27             Log.e(TAG, "Failed to initialize SSL Context", e3);
28         }
29         return null;
30     }
31 }
```



프리다를 활용한 모의해킹 -SSL Pinning

❖ 소스코드 분석 (getSSLContext)

- return `createSSLContext(createTrustManager(createKeyStore(loadCertificate(input))))`;
- 안드로이드에서 SSL 통신 시 클라이언트의 인증서를 사용하는 순서
 1. 클라이언트의 인증서 로드
 2. 클라이언트 인증서 이용하여 keyManager 생성
 3. 클라이언트 인증서 이용하여 TrustManager 생성
 4. keyManager와 TrustManager를 이용하여 SSLContext 생성



프리다를 활용한 모의해킹 -SSL Pinning

❖ 스크립트 작성

```
1  setTimeout(function() {
2      Java.perform(function() {
3          // loadCertificate Method
4          var CertificateFactory = Java.use("java.security.cert.CertificateFactory");
5          var cf = CertificateFactory.getInstance("X.509");
6
7          var FileInputStream = Java.use("java.io.FileInputStream");
8          var my_crt = FileInputStream.$new("/data/local/tmp/cert-der.crt"); // $new는 생성
9          var ca = cf.generateCertificate(my_crt);
10
11         // createKeyStore Method
12         var KeyStore = Java.use("java.security.KeyStore");
13         var ks_type = KeyStore.getDefaultType();
14         var ks = KeyStore.getInstance(ks_type);
15         ks.load(null, null);
16         ks.setCertificateEntry("ca", ca);
17     });
```

```
18         // createTrustManager Method
19         var TrustManagerFactory = Java.use("javax.net.ssl.TrustManagerFactory");
20         var tmf_alg = TrustManagerFactory.getDefaultAlgorithm();
21         var tmf = TrustManagerFactory.getInstance(tmf_alg);
22         tmf.init(ks);
23         var tmf_get = tmf.getTrustManagers();
24
25         // createSSLContext Method
26         var SSLContext = Java.use("javax.net.ssl.SSLContext");
27         SSLContext.getInstance("TLS");
28         SSLContext.init.implementation = function(a, b, c) { // init 메소드가 여러개인지
29             SSLContext.init.call(this, a, tmf_get, c); // call로 기존의 init코드를 불러와
```

프리다를 활용한 모의해킹 -SSL Pinning

❖ 스크립트 작성

The image shows a mobile emulator interface on the left and a terminal window in the center. The emulator displays a 'Pinned SSL Demo' screen with a URL field containing 'https://certs.cac.washington.edu/CAtest/'. Below the URL field is a checkbox labeled 'Use pinned SSL connection' which is checked. A 'Submit request' button is visible. The terminal window shows the command 'frida -U -l C:\Users\KimAhEun\Desktop\LAB\frida\ssl_pinning.js smuldr.sslpin' being executed. The output shows the Frida 12.6.11 toolkit interface with commands like 'help', 'object?', and 'exit/quit'. On the right, a Burp Suite window is open, showing the 'Proxy' tab. It displays a request to 'https://certs.cac.washington.edu:443 [69.91.245.138]'. The 'Intercept' button is highlighted, and the 'Intercept is on' status is shown. Below the request details, the raw HTTP request is displayed in a list format:

```
1 GET /CAtest/ HTTP/1.1
2 User-Agent: Dalvik/2.1.0 (Linux; U; Android 7.1.2; SM-G988N Build/NRD90M)
3 Host: certs.cac.washington.edu
4 Connection: close
5 Accept-Encoding: gzip, deflate
6
7
```



감사합니다

