□ 단국대학교 > 사이버보안 학과 > 인터넷 보안 > 보고서 템플릿

25년 1학기 인터넷 보안 수업

# 인터넷 뱅킹 모의해킹 (2)

**2025년 6월 10일**

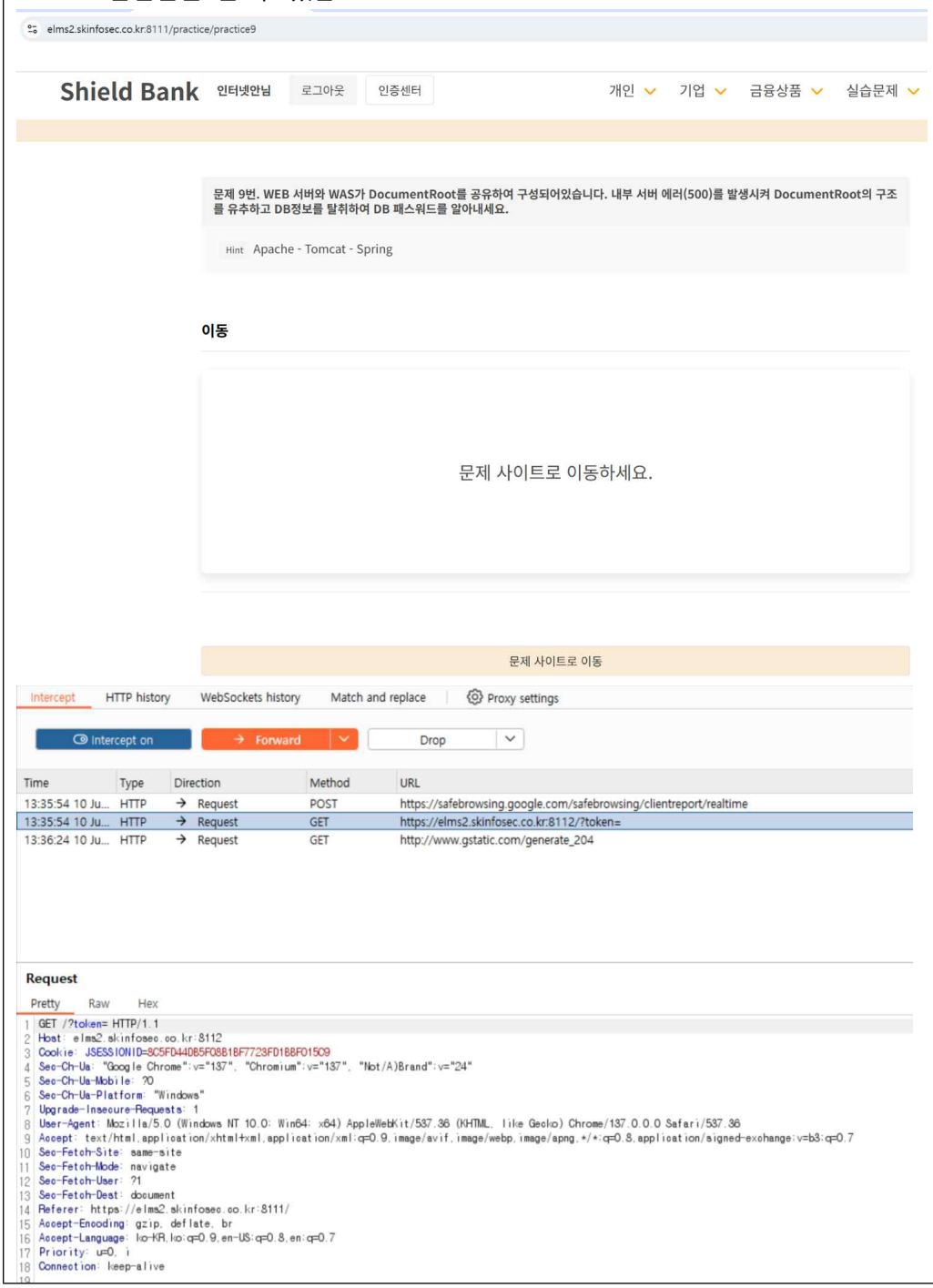
학번 : 32230324
이름 : 김가람

과정 설명

**<Q9 - 내부 서버 에러(500)를 발생시켜 DocumentRoot의 구조를 유추하고 DB 패스워드 알아내기>**

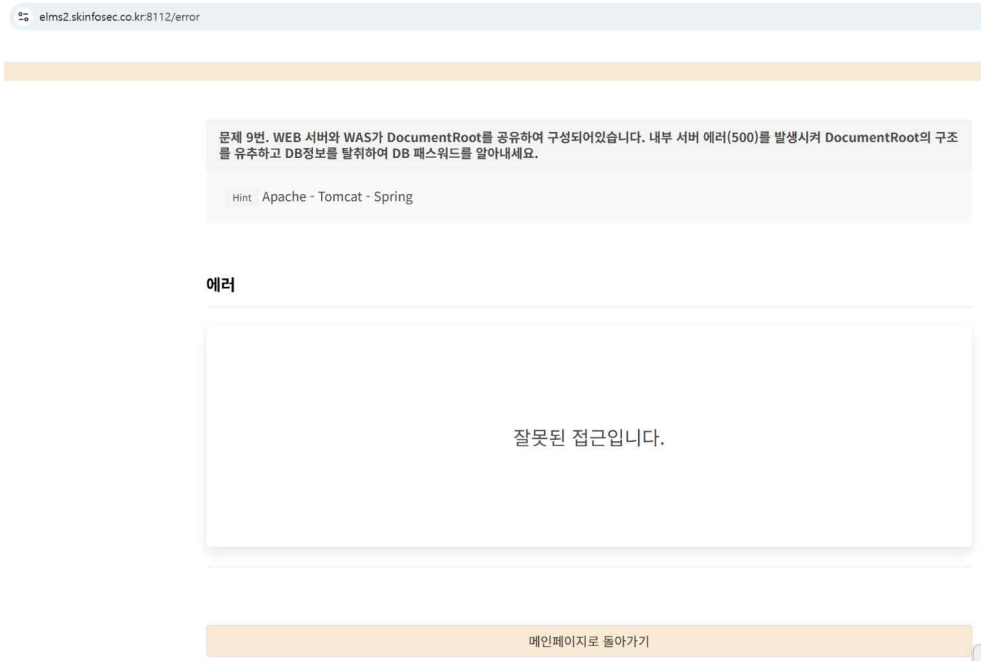
**Step 1. intercept**

: BurpSuite의 Intercept 기능을 켠 상태로 '문제 사이트로 이동' 버튼을 누르면 'https://elms2.skinfosec.co.kr:8112/?token='라는 Request가 나타나는 것으로 보아 기존 포트 번호인 8111(https://elms2.skinfosec.co.kr:8111/practice/practice9)이 아니라 새로운 포트 번호 8112로 연결됨을 알 수 있음.



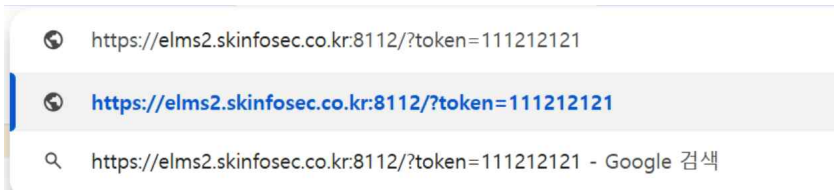
**Step 2. 잘못된 접근**

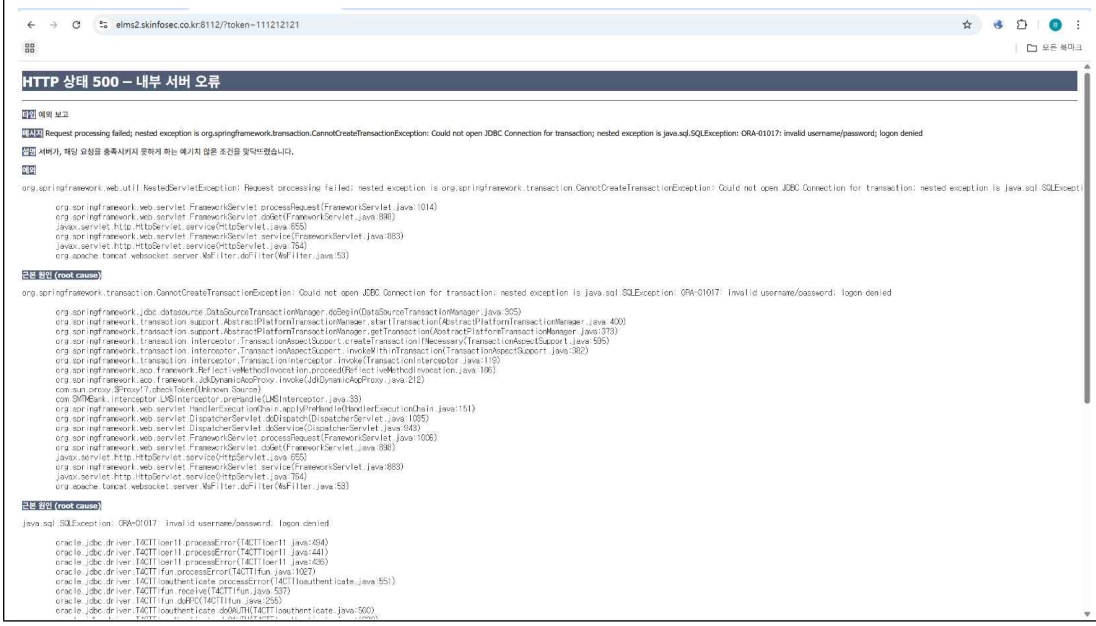
: Intercept 기능을 끄면 '잘못된 접근입니다.'라는 에러 메시지가 나오는 페이지로 연결됨. 이때, 포트 번호는 Step 1에서 알아낸 것과 같이 8112임을 확인할 수 있음.




**Step 3. 내부 서버 에러(500) 발생**

: 서버가 제대로 예외 처리를 해두지 않은 경우, 비정상적인 입력으로 인해 내부 서버 에러(500)가 발생할 수 있음. 이를 이용해 임의의 값을 토큰으로 전달함으로써 서버의 취약한 예외 처리를 유도하였고, 그 결과, 내부 서버 오류가 발생시키는 데 성공함.



**Step 3. 내부 서버 에러 페이지 분석**

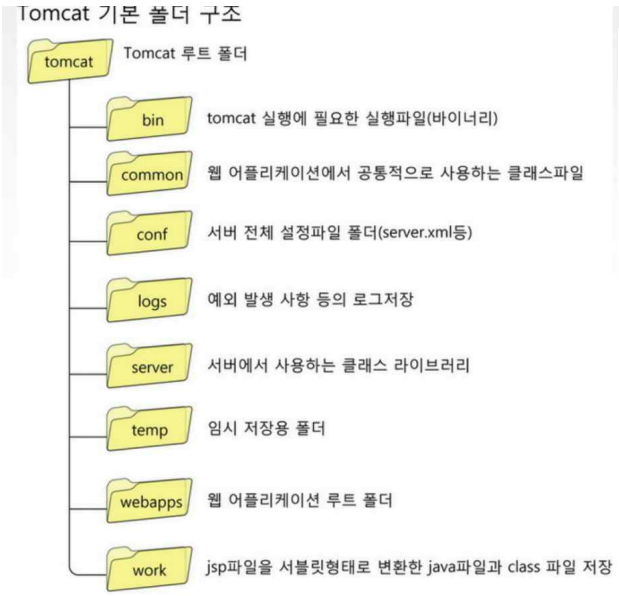
**- Tomcat**

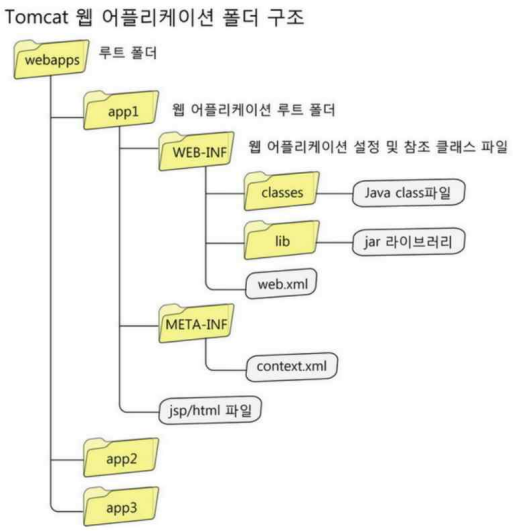
: org.apache.tomcat이 있는 것으로 보아 WAS(웹 애플리케이션 서버)로 Tomcat이 사용되고 있음을 알 수 있음. Tomcat이란 아파치 소프트웨어 재단에서 개발한 웹 서버로 Java로 작성된 웹 애플리케이션을 실행할 수 있도록 지원하며 HTTP 요청 처리, 서블릿 실행, JSP 컴파일 및 실행 기능을 제공함. 이러한 Tomcat의 구조를 알아야 내부적으로 어떠한 설정 파일 등이 있는지 파악할 수 있음. 예를 들어, webapps 디렉터리 아래의 WEB-INF/web.xml 파일은 환경 설정 파일들의 위치를 알려주는 역할을 함. 따라서 Tomcat 분석 시 해당 파일을 가장 먼저 확인하는 것이 좋음.

예외

```
org.springframework.web.util.NestedServletException: Request processing failed; nested exception is org.s

        org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1014)
        org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:898)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:655)
        org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:883)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
```




**- Spring Framework**

: org.springframework.* 패키지가 계속 보이는 것으로 보아 해당 서버는 Spring Framework 기반의 서버임을 알 수 있음. Spring Framework란 Java 플랫폼을 위한 오픈소스 애플리케이션 프레임 워크로 효율적인 구조화, 의존성 관리, 보안, 트랜잭션 처리 등 다양한 기능을 제공함.

( 다음 페이지에 이어서 )

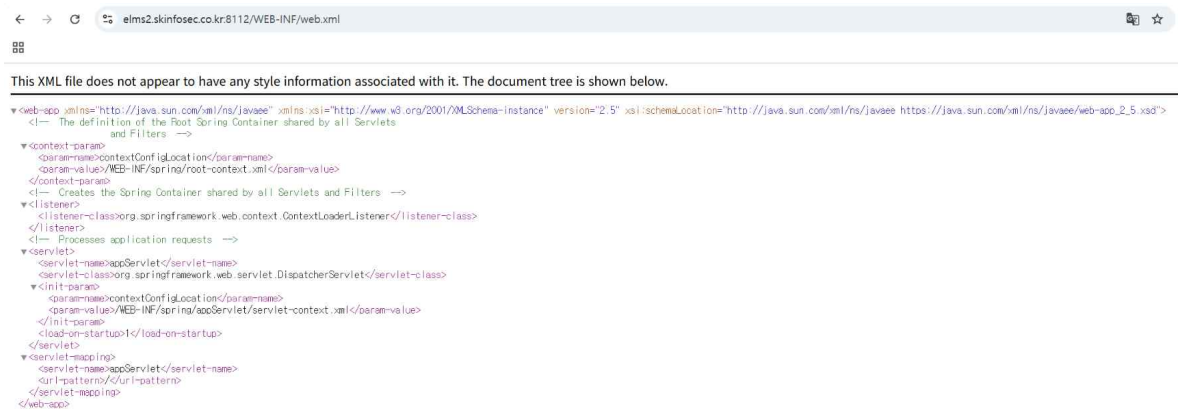
**근본 원인 (root cause)**

```
org.springframework.transaction.CannotCreateTransactionException: Could not open JDBC Connection for transac

        org.springframework.jdbc.datasource.DataSourceTransactionManager.doBegin(DataSourceTransactionManage
        org.springframework.transaction.support.AbstractPlatformTransactionManager.startTransaction(Abstract
        org.springframework.transaction.support.AbstractPlatformTransactionManager.getTransaction(AbstractPl
        org.springframework.transaction.interceptor.TransactionAspectSupport.createTransactionIfNecessary(Tr
        org.springframework.transaction.interceptor.TransactionAspectSupport.invokeWithinTransaction(Transac
        org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.jav
        org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java
        org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:212)
        com.sun.proxy.$Proxy17.checkToken(Unknown Source)
        com.SMTMBank.interceptor.LMSInterceptor.preHandle(LMSInterceptor.java:33)
        org.springframework.web.servlet.HandlerExecutionChain.applyPreHandle(HandlerExecutionChain.java:151)
        org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1035)
        org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:943)
        org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1006)
        org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:898)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:655)
        org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:883)
        javax.servlet.http.HttpServlet.service(HttpServlet.java:764)
        org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)
```

**Step 4. web.xml**

: Step 3에서 설명한 것처럼 가장 확인하는 것이 좋은 파일은 web.xml임. 이 파일은 WEB-INF 디렉터리 아래에 위치하므로 WEB-INF/web.xml라는 경로로 접근하면 내용을 확인할 수 있음. 해당 파일을 살펴보면 '/WEB-INF/spring/root-contex.xml'이라는 또 다른 설정 파일 경로가 명시되어 있는 것을 확인할 수 있음.

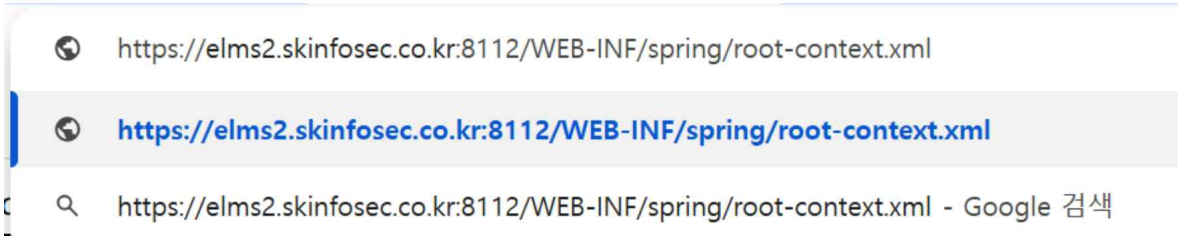

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" version="2.5" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee https://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <!-- The definition of the Root Spring Container shared by all Servlets
       and Filters -->
  <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>/WEB-INF/spring/root-context.xml</param-value>
  </context-param>
  <!-- Creates the Spring Container shared by all Servlets and Filters -->
  <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
  </listener>
  <!-- Processes application requests -->
  <servlet>
    <servlet-name>appServlet</servlet-name>
    <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
    <init-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/appServlet/servlet-context.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>appServlet</servlet-name>
    <url-pattern>/</url-pattern>
  </servlet-mapping>
</web-app>
```

```
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-inst
    <!-- The definition of the Root Spring Container shared by all Servlets
         and Filters -->
    <context-param>
      <param-name>contextConfigLocation</param-name>
      <param-value>/WEB-INF/spring/root-context.xml</param-value>
    </context-param>
    <!-- Creates the Spring Container shared by all Servlets and Filters -->
    <listener>
      <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <!-- Processes application requests -->
    <servlet>
```

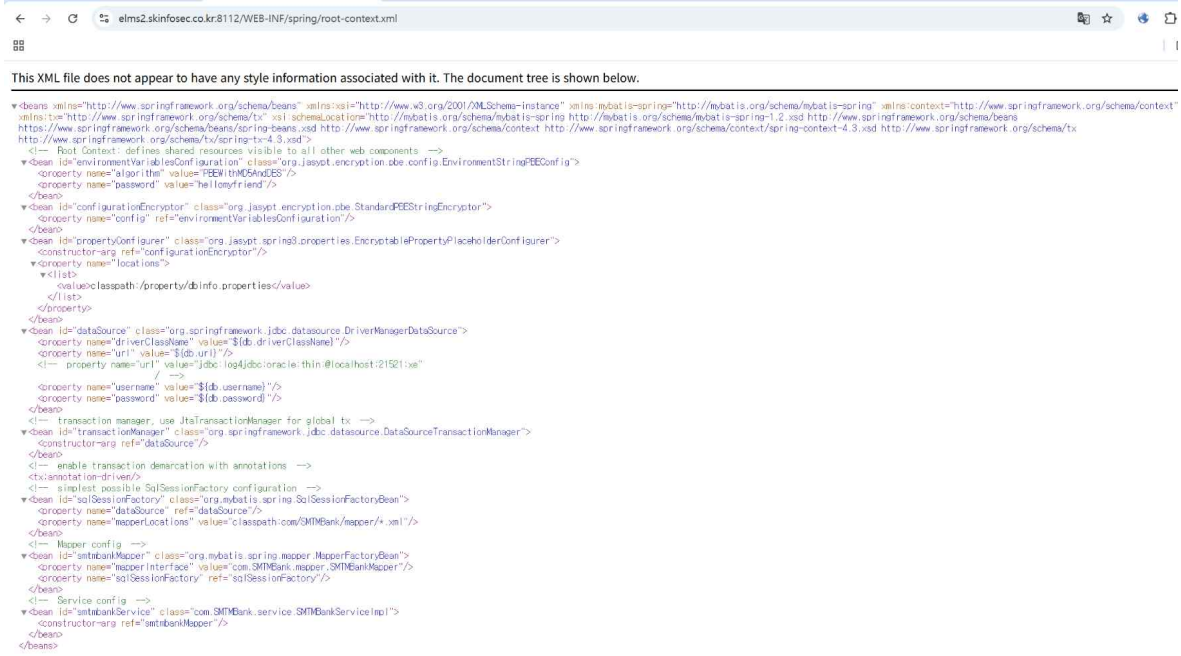
**Step 5. /WEB/INF/spring/root-context.xml**

**5-1. 경로로 연결**

: Step 4에서 찾은 경로로 연결해보면 새로운 설정 파일의 내용을 확인할 수 있음.



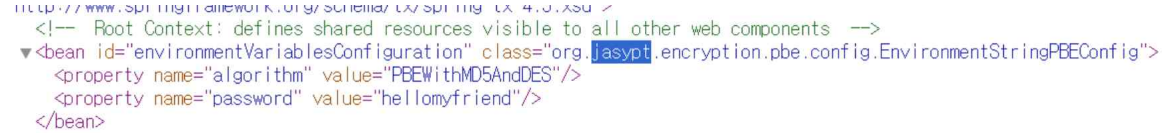

( 다음 페이지에 이어서 )




**5-2. 파일 분석**

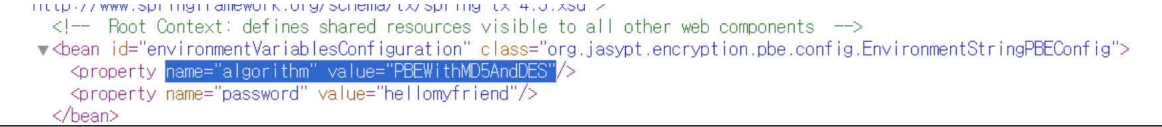
**- jasypt**

: Java 애플리케이션에서 암호화와 복호화를 쉽게 할 수 있도록 도와주는 jasypt 라이브러리가 사용되고 있음을 알 수 있음.




**- Algorithm**

: Jasypt가 사용할 암호화 알고리즘을 지정하는 부분으로, PBEWithMD5AndDES를 사용함. 이는 MD5 해시와 DES 대칭키 암호화를 조합한 알고리즘임.



**- Password**

: hellowfriend라는 비밀번호가 있는 것을 확인할 수 있는데, 이는 암호화 및 복호화에 사용할 기본키(비밀키)로 매우 민감한 정보임.

```
<!-- Root Context: defines shared resources visible to all other web components -->
<bean id="environmentVariablesConfiguration" class="org.jasypt.encryption.pbe.config.EnvironmentStringPBEConfig">
  <property name="algorithm" value="PBEWithMD5AndDES"/>
  <property name="password" value="hellomyfriend"/>
</bean>
```

**- Classpath**

: Java 클래스 파일들이 저장되어 있는 경로인 classpath를 발견할 수 있는데, 이는 WEB_INF/classes디렉터리 아래에 위치함.

```
<constructor-arg ref="configurationEncryptor"/>
<property name="locations">
  <list>
    <value>classpath:/property/dbinfo.properties</value>
  </list>
</property>
</bean>
```

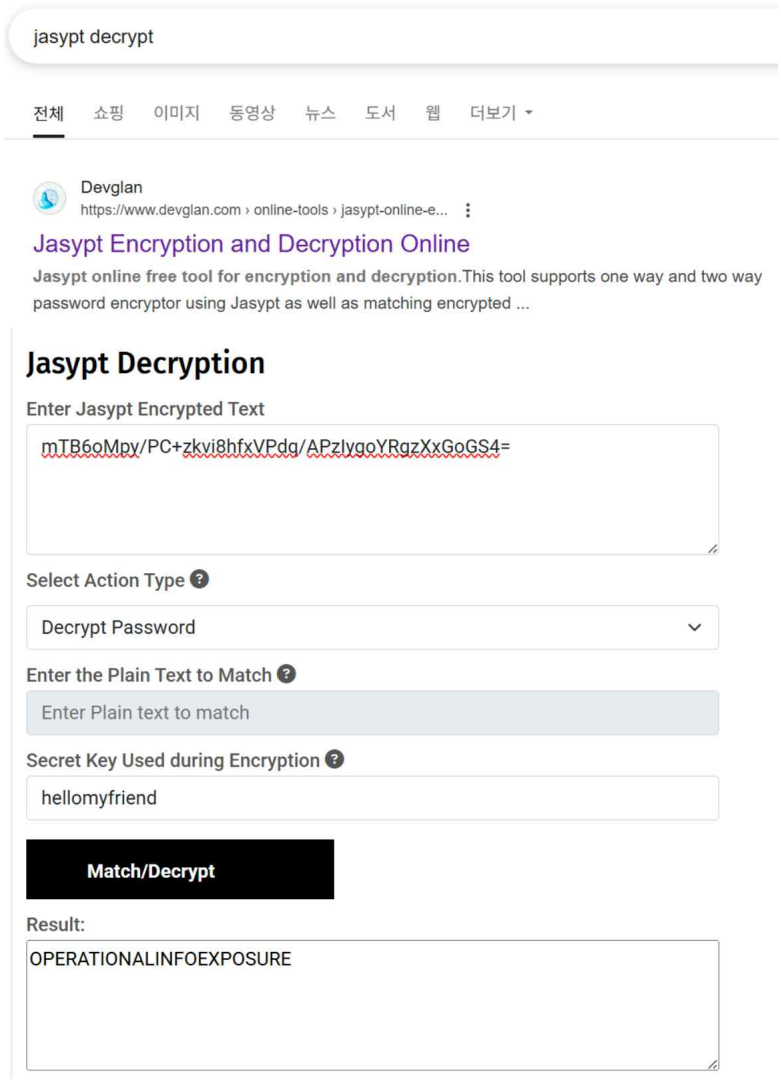
**Step 6. /property/dbinfo.properties**

: Step 5-2에서 찾은 /WEB-INF/classes/property/dbinfo.properties로 연결해보면 DB 정보를 확인할 수 있지만 DB 패스워드는 암호화되어 있음.



**Step 7. 복호화**

: Step 5-2에서, 암복호화에 jasypt 라이브러리가 사용되고 있음을 확인했음. 이에 따라 jasypt decrypt를 검색하면 복호화를 지원하는 사이트를 찾을 수 있음. 해당 사이트의 Decryption 부분으로 이동한 뒤, Step 6에서 발견한 암호화된 패스워드를 입력하고, Action Type은 Decrypt Password로 설정하고 Secret key에는 Step 5-2에서 발견했던 password인 hellomyfriend를 입력하여 복호화하면 'OPERATIONALINFOEXPOSURE'라는 문자열이 출력됨.




**Step 8. 정답 입력**

: Step 7에서 복호화를 통해 알아낸 DB 패스워드(OPERATIONALINFOEXPOSURE)를 정답란에 입력하면 정답으로 인정됨.

또 이러한 과정들을 통해 알 수 있었던 해당 페이지의 문제점이 있는데, 에러 처리가 잘 되어 있지 않고 주요 정보들이 그대로 드러나는 것임. 이러한 점을 방지하기 위해 web.xml에 접근하지 못하게 막는 것이 중요하며, 직접 접근할 경우, 403이 뜨도록 처리할 필요가 있음.

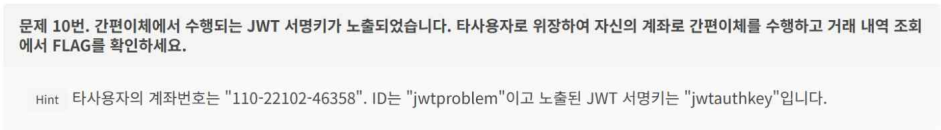
**<Q10 - 타사용자로 위장하여 간편이체를 수행하고 거래 내역에서 FLAG 확인>**

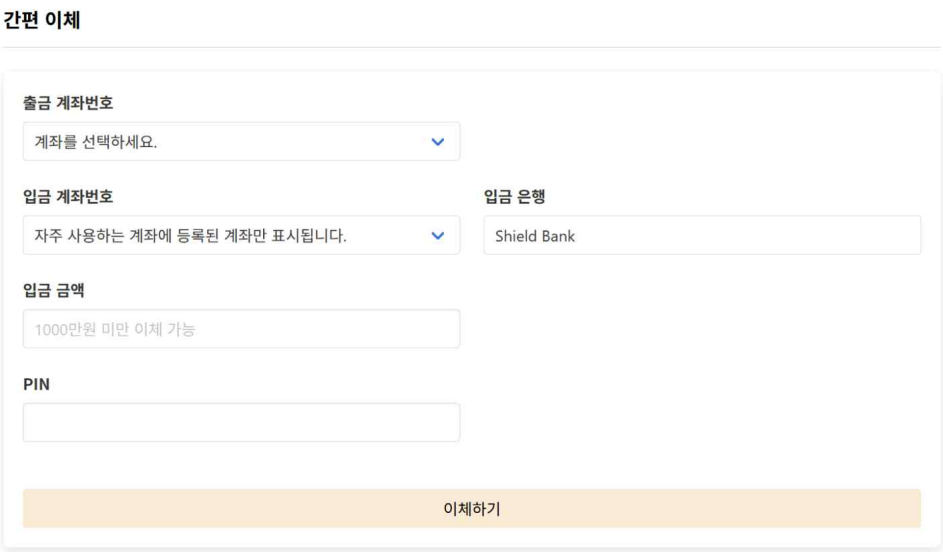
**Hint) 타사용자 계좌 : 1102210246358, ID : jwtproblem, 노출된 JWT 서명키 : jwtauthkey**

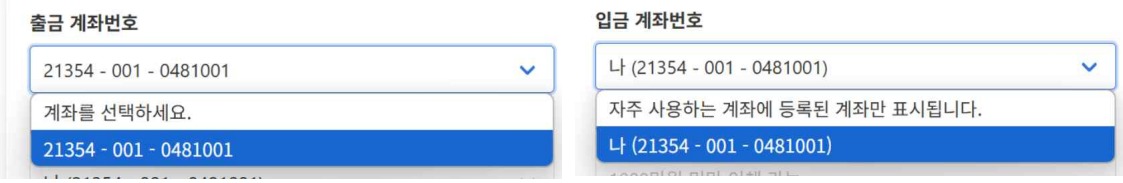
JWT란 JSON 기반의 토큰으로 인증과 정보 전달에 사용됨. 주로 사용자의 인증 정보를 안전하게 클라이언트에 저장하고 서버와 주고받을 때 사용됨. JWT는 마침표(.)를 기준으로 Header, Payload, Signature 총 세 부분으로 나뉨. Header는 토큰의 타입과 암호화 알고리즘 정보를 담고 있고, Payload는 토큰에 담길 실제 정보들을 포함하며 Signature는 Header와 Payload를 연결한 후 secret key와 함께 지정된 알고리즘으로 서명한 값으로 위조 방지를 위해 사용됨.

**Step 1. 기본 로직 분석**

: 간편 이체 페이지에서 출금 계좌번호와 입금 계좌번호를 선택하고 입금 금액을 입력과 PIN번호 인증 후 이체하는 방식으로 동작함. 출금 계좌번호는 현재 사용자의 계좌번호를 선택할 수 있게 되어 있고, 입금 계좌번호는 자주 사용하는 계좌에 등록된 계좌번호만 선택할 수 있게 되어 있음. 문제에서 타사용자로 위장하여 자신의 계좌로 간편이체를 수행하라고 했으므로 자주 사용하는 계좌 등록 페이지에 접속해 자신의 계좌를 등록하고 확인해보면 입금 계좌번호 목록에 나타남.



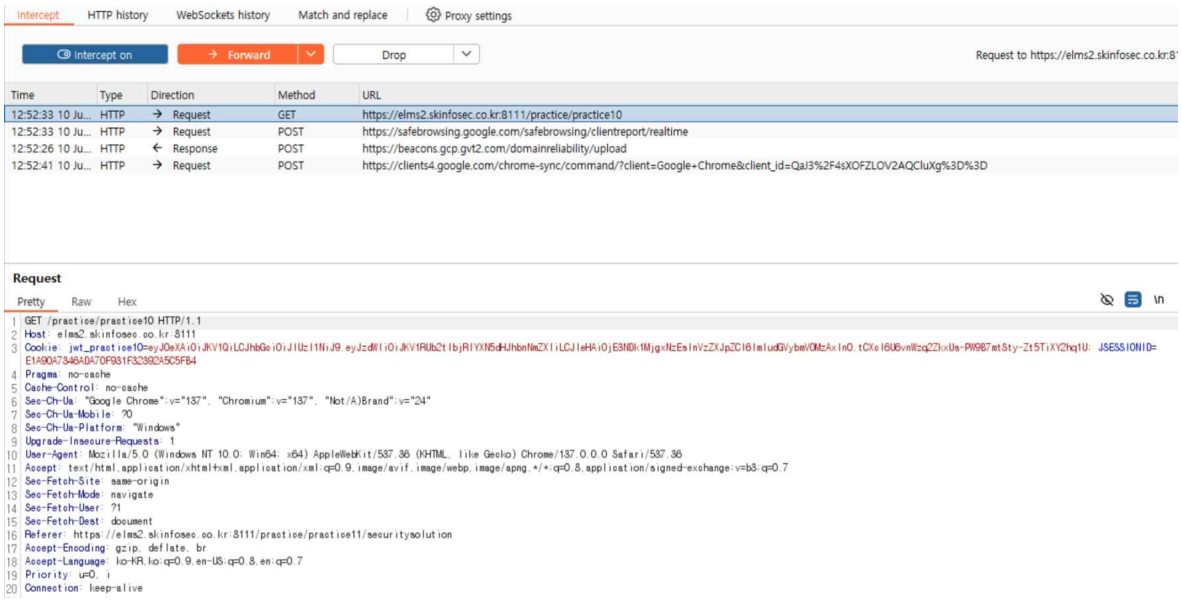



**Step 2. JWT 수정**

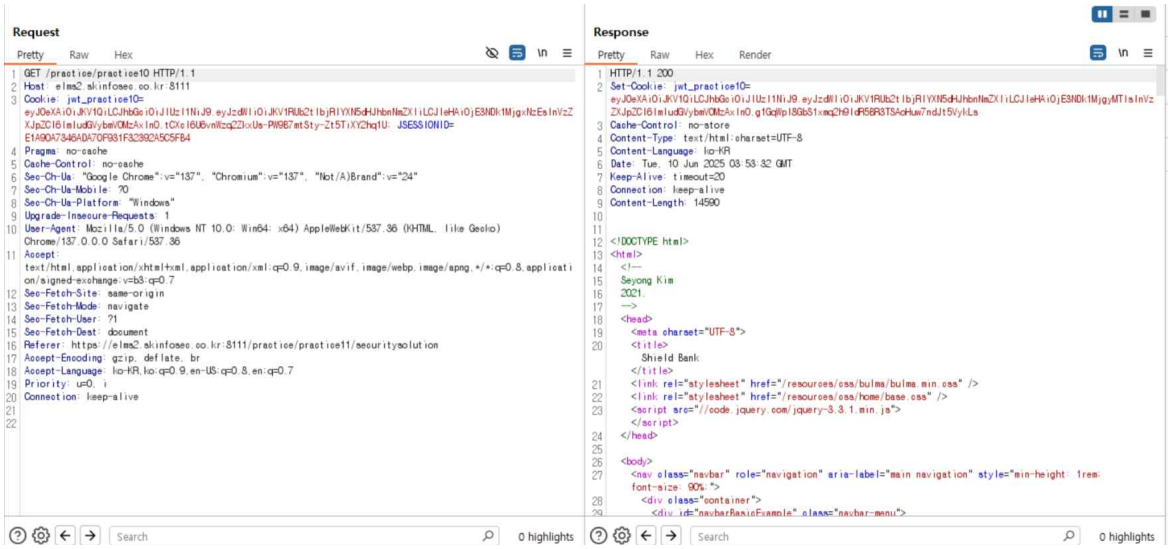
**2-1. Intercept**

: BurpSuite에서 Intercept 기능을 켠 후 간편 이체 페이지를 새로고침(Ctrl + F5)하면 Intercept 목록에 뜸(https://elms2.skinfosec.co.kr:8111/practice/practice10). Request를 살펴보면 jwt_pratice10이라는 암호화된 JWT 토큰이 저장되어 있는 부분을 확인할 수 있음.




**2-2. Forward**

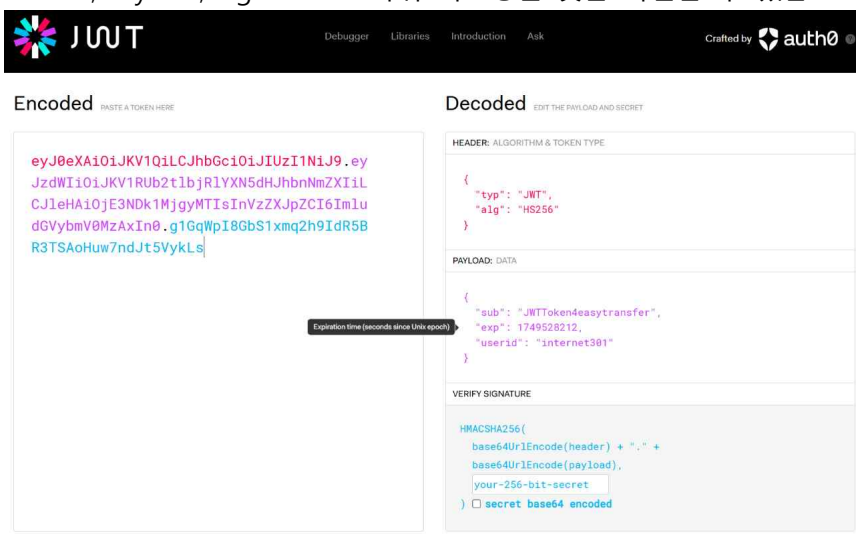
: 하지만 JWT는 세션 방식과 달리 서버가 아닌 클라이언트에 토큰을 저장하기 때문에 서버(Request)에서 사용자 정보를 변경해도 반영이 되지 않음. 따라서 해당 페이지를 Forward해 Response를 받아오면 보이는 jwt_practice10를 수정해줘야 함.




## 2-3. JWT 복호화, 암호화

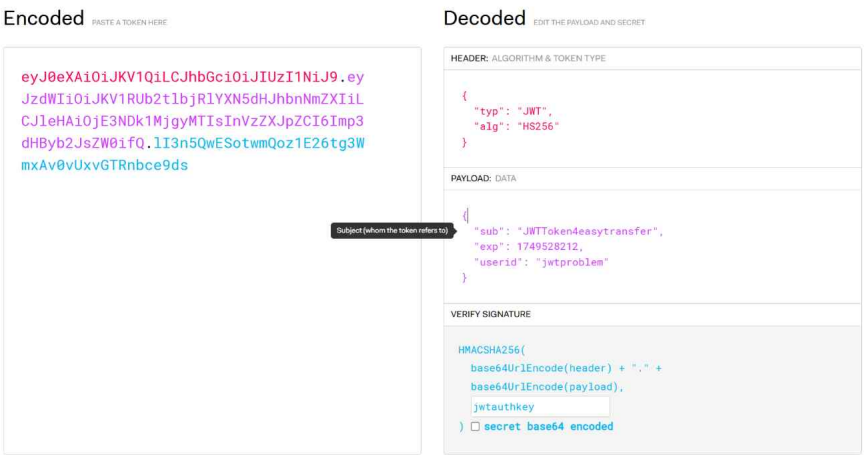
**- 붙여넣기**

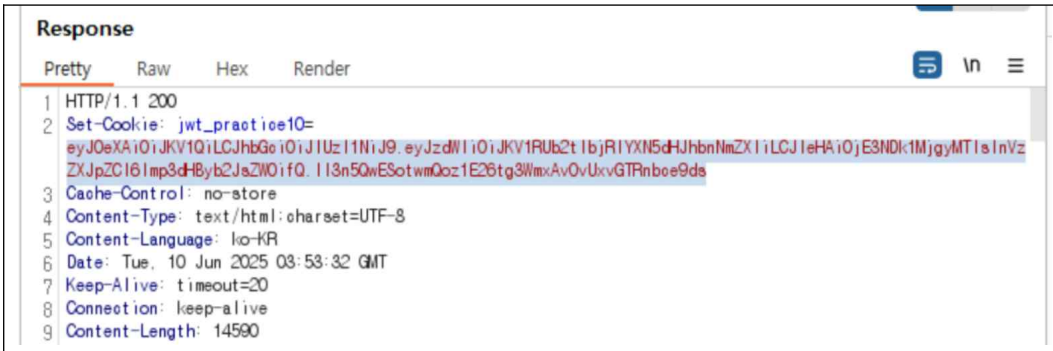
: Step 2-2의 Response에 있는 jwt_practice10을 복사한 후 JWT을 디코딩, 검증, 생성해주는 JWT 공식 사이트(jwt.io)에 들어가 Encoded 부분에 붙여넣음. Decoded 부분을 확인해보면 Header, Payload, Signature로 나눠 디코딩된 것을 확인할 수 있음.




**- 타사용자 정보로 수정**

: 타사용자로 위장하기 위해 userid를 jwtproblem, signature의 key를 jwtauthkey로 설정해 JWT를 재생성함. 이후 Response의 jwt_practice10 값을 새로 생성한 토큰으로 교체 후 Intercpet를 끔.



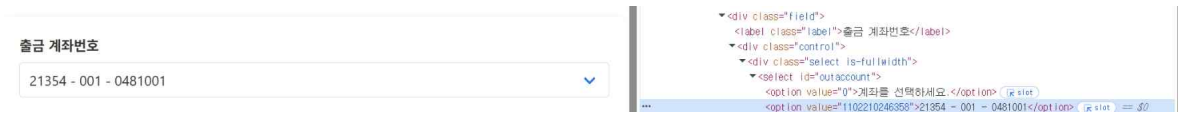



**Step 3. 출금 계좌번호 수정**

: 간편 이체 페이지로 돌아가 개발자 도구(F12)를 사용해 출금 계좌번호의 value 값을 본인 계좌번호에서 타사용자 계좌번호(1102210246358)로 수정함.

**- 수정 전 출금 계좌번호 : 본인**


**- 수정 후 출금 계좌번호 : 타사용자**




**Step 4. 간편 이체**

: 출금 계좌번호를 선택하고, (Step 3에서 value값만 수정하고 text는 그대로 두었기에 본인의 계좌처럼 보이지만 실제론 타사용자의 계좌가 선택된 상태임.) 입금 계좌번호 선택, 입금 금액 입력, PIN번호 인증하면 이체 완료 페이지로 이동됨.

문제 10번. 간편이체에서 수행되는 JWT 서명키가 노출되었습니다. 타사용자로 위장하여 자신의 계좌로 간편이체를 수행하고 거래 내역 조회에서 FLAG를 확인하세요.

Hint 타사용자의 계좌번호는 "110-22102-46358". ID는 "jwtproblem"이고 노출된 JWT 서명키는 "jwtauthkey"입니다.

완료

간편 이체가 완료되었습니다!

문제 페이지로 돌아가기

**Step 5. 거래 내역 조회**

: 거래 내역을 조회해보면 FLAG: JWTVULNERABILITY라는 이름의 사용자로부터 1000원을 입금받은 내역을 확인할 수 있음.

**거래 내역 조회**

계좌번호

21354 - 001 - 0481001

시작 날짜 연도-월-일

종료 날짜 연도-월-일

거래 유형 전체

검색

| 거래일시 | 보낸분/받는분 | 금액 | 잔액 | 거래 유형 |
|---|---|---|---|---|
| 2025-06-10 | FLAG: JWTVULNERABILITY | 1,000원 | 2,299,003,000원 | 입금 |
| 2025-06-10 | FLAG: JWTVULNERABILITY | 1,000원 | 2,299,002,000원 | 입금 |
| 2025-06-10 | FLAG: JWTVULNERABILITY | 1,000원 | 2,299,001,000원 | 입금 |
| 2025-05-27 | 이큐스트 | 1,000,000원 | 2,299,000,000원 | 출금 |
| 2025-05-27 | 대출금 입금 | 1,000,000,000원 | 2,300,000,000원 | 입금 |
| 2025-05-27 | 대출금 입금 | 100,000,000원 | 1,300,000,000원 | 입금 |
| 2025-05-27 | 대출금 입금 | 100,000,000원 | 1,200,000,000원 | 입금 |
| 2025-05-27 | 대출금 입금 | 100,000,000원 | 1,100,000,000원 | 입금 |

1

| 성명 | 프로젝트 후 소감 |
| --- | --- |
| 김가람 | 이번 실습을 통해 JWT라는 것에 대해 새롭게 알게 되었고, 이 토큰이 서명키 없이도 쉽게 디코딩될 수 있다는 점에서 보안의 중요성을 다시 느낌. JWT를 직접 디코딩하여 Header, Payload, Signature 각각을 확인하고 노출된 서명키와 ID를 통해 타사용자로 위장하는 실습을 통해 인증 우회가 실제로 어떻게 이루어지는지 체감할 수 있었음. 또한 Tomcat 구조에 대해 다시 한 번 공부함으로써 web.xml의 역할과 위치 등을 새롭게 알 수 있었고 Tomcat에 대한 이해도를 더욱 높일 수 있는 시간이었음. jasypt 복호화 과정에서는 DB 패스워드 같은 중요한 정보가 암호화되어 있더라도 설정이나 키가 노출되면 쉽게 복호화될 수 있음을 직접 경험할 수 있었음. 이를 통해 단순히 암호화만 하는 것이 아니라 환경 설정이나 키 등 여러 요소들에 대한 보안도 함께 이루어져야 함을 깨달음. 마지막으로 실제 생활 속에서 일어날 수 있는 시나리오를 직접 경햄해볼 수 있는 좋은 기회가 되었으며, 내가 생각했던 것보다 더 많은 보안 취약점이 존재한다는 것을 느낄 수 있었음. |