|  |  |
| --- | --- |
|  |  |
| **웹 취약점점검 통합 결과보고서**  **< NO-UH 온라인 강의 플랫폼>** | |
|  |  |

2024. 12. 19.

SK쉴더스

경기도 성남시 분당구 판교로 227번길23

2020. 01. 20

**목 차**

[1. 사업 개요 7](#_Toc185603178)

[1.1. 수행 목적 7](#_Toc185603179)

[1.2. 수행 대상 7](#_Toc185603180)

[1.3. 수행 일정 7](#_Toc185603181)

[1.4. 수행 인원 8](#_Toc185603182)

[2. 수행 방법 9](#_Toc185603183)

[2.1. 점검 항목 9](#_Toc185603184)

[2.1.1. 주요정보통신기반시설 기술적 취약점 분석평가 상세가이드 9](#_Toc185603185)

[2.1.2. 전자금융 기반시설 취약점 분석·평가(종합 점검) 11](#_Toc185603186)

[2.2. 사용 도구 11](#_Toc185603187)

[3. 결과 요약 12](#_Toc185603188)

[3.1. 총평 12](#_Toc185603189)

[3.2. 취약점 요약 13](#_Toc185603190)

[4. 상세 결과 14](#_Toc185603191)

[4.1. NO-UH.com 14](#_Toc185603192)

[4.1.1. SQL 인젝션 14](#_Toc185603193)

[4.1.2. 악성 콘텐츠 18](#_Toc185603194)

[4.1.3. 크로스사이트 스크립팅 21](#_Toc185603195)

[4.1.4. 약한 문자열 강도 24](#_Toc185603196)

[4.1.5. 크로스사이트 리퀘스트 변조(CSRF) 26](#_Toc185603197)

[4.1.6. 불충분한 세션 만료 29](#_Toc185603198)

[4.1.7. 자동화 공격 31](#_Toc185603199)

[4.1.8. 파일 업로드 34](#_Toc185603200)

[4.1.9. 파일 다운로드 35](#_Toc185603201)

[4.1.10. 위치 공개 37](#_Toc185603202)

[4.1.11. 서버 사이드 템플릿 인젝션(SSTI) 38](#_Toc185603203)

[5. 보안 대책 42](#_Toc185603204)

[5.1. SQL 인젝션 42](#_Toc185603205)

[5.2. 악성 콘텐츠 45](#_Toc185603206)

[5.3. 크로스사이트 스크립팅 46](#_Toc185603207)

[5.4. 약한 문자열 강도 48](#_Toc185603208)

[5.5. 크로스사이트 리퀘스트 변조(CSRF) 49](#_Toc185603209)

[5.6. 불충분한 세션 만료 50](#_Toc185603210)

[5.7. 자동화 공격 51](#_Toc185603211)

[5.8. 파일 업로드 52](#_Toc185603212)

[5.9. 파일 다운로드 55](#_Toc185603213)

[5.10. 위치 공개 57](#_Toc185603214)

[5.11. 서버 사이드 템플릿 인젝션(SSTI) 58](#_Toc185603215)

표 목차

[[표 1] 수행 대상 7](#_Toc185603216)

[[표 2] 수행 일정 7](#_Toc185603217)

[[표 3] 수행 인원 8](#_Toc185603218)

[[표 4] “주요정보통신기반시설 기술적 취약점 분석평가 상세가이드” 취약점 점검 항목 11](#_Toc185603219)

[[표 5] “전자금융 기반시설 취약점 분석·평가(종합 점검)” 취약점 점검 항목 11](#_Toc185603220)

[[표 6] 웹 취약점 점검 사용 도구 11](#_Toc185603221)

[[표 7] 취약점 요약 13](#_Toc185603222)

[[표 8] SQL 인젝션 - 취약점 요약 14](#_Toc185603223)

[[표 9] 데이터베이스명 탈취 명령어 17](#_Toc185603224)

[[표 10] 악성 콘텐츠 - 취약점 요약 18](#_Toc185603225)

[[표 11] 크로스사이트 스크립팅 - 취약점 요약 21](#_Toc185603226)

[[표 12] 약한 문자열 강도 - 취약점 요약 24](#_Toc185603227)

[[표 13] 크로스사이트 리퀘스트 변조(CSRF) - 취약점 요약 26](#_Toc185603228)

[[표 14] 불충분한 세션 만료 - 취약점 요약 29](#_Toc185603229)

[[표 15] 자동화 공격 - 취약점 요약 31](#_Toc185603230)

[[표 16] 불충분한 인가(1) - 취약점 요약 34](#_Toc185603231)

[[표 17] 파일 다운로드 - 취약점 요약 35](#_Toc185603232)

[[표 18] 위치 공개 - 취약점 요약 37](#_Toc185603233)

[[표 19] SSTI - 취약점 요약 38](#_Toc185603234)

그림 목차

[[그림 1] 악의적 구문 삽입 지점 확인 15](#_Toc185603235)

[[그림 2] SQL 인젝션 참 값 삽입 15](#_Toc185603236)

[[그림 3] SQL 인젝션 거짓 값 삽입 16](#_Toc185603237)

[[그림 4] SQL 인젝션을 통한 데이터 베이스명 추출 16](#_Toc185603238)

[[그림 5] 악성 컨텐츠 업로드 18](#_Toc185603239)

[[그림 6] 악성 컨텐츠 삽입 완료 19](#_Toc185603240)

[[그림 7] 악성 콘텐츠 게시글 확인 19](#_Toc185603241)

[[그림 8] 악성 콘텐츠 파일 삭제 여부 확인 20](#_Toc185603242)

[[그림 9] 게시글 작성 페이지 악의적 구문 삽입 22](#_Toc185603243)

[[그림 10] XSS – 필터링된 내용 확인 22](#_Toc185603244)

[[그림 11] XSS - 필터링된 내용 수정 23](#_Toc185603245)

[[그림 12] XSS 작동 23](#_Toc185603246)

[[그림 13] 약한 문자열로 회원가입 진행 24](#_Toc185603247)

[[그림 14] 회원가입 데이터 25](#_Toc185603248)

[[그림 15] 회원가입 완료 25](#_Toc185603249)

[[그림 16] CSRF 악의적인 구문 삽입 27](#_Toc185603250)

[[그림 17] CSRF - 필터링 된 내용 확인 27](#_Toc185603251)

[[그림 18] CSRF - 필터링 된 내용 수정 28](#_Toc185603252)

[[그림 19] 회원탈퇴 악의적 구문 삽입 게시글 확인 28](#_Toc185603253)

[[그림 20] 로그인 후 로그인 여부 확인 29](#_Toc185603254)

[[그림 21] 로그인 후 2시간 뒤 로그아웃 여부 확인 30](#_Toc185603255)

[[그림 22] 반복적으로 작성될 데이터 작성 31](#_Toc185603256)

[[그림 23] 퀴즈 테이블의 PK값 확인 32](#_Toc185603257)

[[그림 24] 자동화 공격 - 10개의 게시글 작성 32](#_Toc185603258)

[[그림 25] 작성된 10개의 게시글 확인 33](#_Toc185603259)

[[그림 26] php 파일 업로드 완료 34](#_Toc185603260)

[[그림 27] URL에 공격 구문 삽입 35](#_Toc185603261)

[[그림 28] /etc/passwd 파일 다운로드 36](#_Toc185603262)

[[그림 29] /admin 페이지 누출 37](#_Toc185603263)

[[그림 30] SQL 인젝션 삽입 지점 확인 38](#_Toc185603264)

[[그림 31] SSTI - 악의적 구문을 통한 데이터 타입 추출 39](#_Toc185603265)

[[그림 32] SSTI - 악의적 구문을 통한 클래스 추출 40](#_Toc185603266)

[[그림 33] SSTI - cat /etc/passwd 실행 41](#_Toc185603267)

1. 사업 개요
   1. 수행 목적

본 점검의 목적은 NO-UH 웹 서비스를 대상으로 모의해킹을 실시하여 침해 사고를 초래할 가능성이 있는 보안 약점을 도출하고, 발견된 보안 약점의 원인을 분석하여 대응방안을 제안하며, 이를 통해서 NO-UH 웹 서비스의 침해사고 발생 위험을 사전에 보완함으로써 서비스의 신뢰성과 안전한 운영에 기여하고자 합니다.

* 1. 수행 대상

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 번호 | 환경 | 서비스 이름 | 플랫폼 | URL |
| 1 | 인터넷망 | NO-UH | WEB | https://no-uh.com |

[표 1] 수행 대상

* 1. 수행 일정

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 구분 | 세부 업무 | 2024. 11. 12 ~ 2024. 11. 28 | | |
| 11/12~11/25 | 11/25 ~ 11/26 | 11/26~11/28 |
| 웹  취약점점검 | 환경 구축 |  |  |  |
| 웹 서비스 점검 |  |  |  |
| 보고서 작성 및 제출 |  |  |  |

[표 2] 수행 일정

* 1. 수행 인원

|  |  |  |  |
| --- | --- | --- | --- |
| 담당 업무 | 이름 | 이메일 | 비고 |
| 웹 취약점점검 | 김현우 | unun4972@naver.com | 팀장 |
| 웹 취약점점검 | 이재성 | fl53hn@naver.com | 보고서 작성 |
| 웹 취약점점검 | 임가은 | bedehator@gmail.com | - |
| 웹 취약점점검 | 전시현 | jn.snoir@gmail.com | - |
| 웹 취약점점검 | 정우석 | jungws0530@gmail.com | - |
| 웹 취약점점검 | 정웅진 | mini10001100101@gmail.com | - |
| 웹 취약점점검 | 정윤서 | ellajung0620@gmail.com | - |
| 웹 취약점점검 | 조수호 | ymzkxm905@gmail.com | - |
| 웹 취약점점검 | 진혜윤 | ymcacom665@gmail.com | - |

[표 3] 수행 인원

1. 수행 방법
   1. 점검 항목

취약점 점검 항목은 “주요정보통신기반시설 기술적 취약점 분석평가 상세가이드” 내용을 바탕으로 구성한 28개의 항목과 “전자금융 기반시설 취약점 분석·평가(종합 점검)”에서 추가한 1개 항목 총 29개 항목을 기준으로 진단하였습니다.

* + 1. 주요정보통신기반시설 기술적 취약점 분석평가 상세가이드

| 순번 | 점검 항목 | 점검 내용 | 위험도 |
| --- | --- | --- | --- |
| 1 | 버퍼 오버플로우 | 개발 시에 할당된 저장 공간보다 더 큰 값의 입력이 가능하고 이로 인한 오류 발생 시 의도되지 않은 정보 노출, 프로그램에 대한 비인가  접근 및 사용 등이 발생하는 취약점 | 상 |
| 2 | 포맷스트링 | C언어로 만드는 프로그램 중 변수의 값을 출력하거나 입력 받을 때 입력 받은 값을 조작하여 프로그램의 메모리 위치를 반환 받고 메모리 주소를 변조하여 시스템의 관리자 권한을 획득할 수 있는 취약점 | 상 |
| 3 | LDAP 인젝션 | 응용 프로그램이 사용자 입력 값에 대한 적절한 필터링 및 유효성 검증을 하지 않아 공격자는 로컬 프록시를 사용함으로써 LDAP 문의 변조가 가능한 취약점 | 상 |
| 4 | 운영체제 명령 실행 | 적절한 검증절차를 거치지 않은 사용자 입력 값에 의해 의도하지 않은 시스템 명령어가 실행되는 취약점 | 상 |
| 5 | SQL 인젝션 | 비정상적인 SQL 쿼리로 DBMS 및 데이터(Data)를 열람하거나 조작 가능한 취약점 | 상 |
| 6 | SSI 인젝션 | 웹 서버 상에 있는 파일을 include 시켜 명령문 실행이 가능하게 하여 불법적으로 데이터에 접근할 수 있는 취약점 | 상 |
| 7 | XPath 인젝션 | 프로그래머가 의도하지 않았던 문자열을 전달하여 쿼리문의 의미를 왜곡시키거나 그 구조를 변경하고 임의의 쿼리를 실행하여 인가되지 않은 데이터를 열람할 수 있는 취약점 | 상 |
| 8 | 디렉토리 인덱싱 | 요청 파일이 존재하지 않거나 URI에서 디렉토리까지 입력했을 때 자동으로 디렉토리 리스트를 출력하는 취약점 | 상 |
| 9 | 정보 누출 | 웹사이트 데이터가 노출되는 것으로 개발과정의 주석이나 오류 메시지 등에서 중요한 정보가 노출되어 공격자에게 2차 공격을 하기 위한 중요한 정보를 제공할 수 있는 취약점 | 상 |
| 10 | 악성 콘텐츠 | 웹 사이트 게시판, 댓글 등에 정상적인 콘텐츠 대신 악성 콘텐츠를 주입하여 실행될 경우 사용자가 해당 콘텐츠 열람 시 악성코드 감염 및 웹 페이지 변조 등의 위험에 노출될 수 있는 취약점 | 상 |
| 11 | 크로스사이트 스크립팅 | 웹 애플리케이션의 입력 값을 통해 다른 최종 사용자의 클라이언트에서 임의의 스크립트가 실행되는 취약점 | 상 |
| 12 | 약한 문자열 강도 | 유추가 용이한 계정 및 패스워드의 사용으로 인한 사용자 권한 탈취 위험이 존재하는 취약점 | 상 |
| 13 | 불충분한 인증 | 민감한 데이터에 접근할 수 있는 곳에 취약한 인증 메커니즘으로 구현되어 인증 과정을 우회할 수 있는 취약점 | 상 |
| 14 | 취약한 패스워드 복구 | 취약한 패스워드 복구 로직(패스워드 찾기 등)으로 인하여 공격자가 불법적으로 다른 사용자의 패스워드를 획득, 변경할 수 있는 취약점 | 상 |
| 15 | 크로스사이트 리퀘스트 변조(CSRF) | 사용자의 신뢰(인증) 정보 내에서 사용자의 요청(Request)을 변조함으로써 해당 사용자의 권한으로 악의적인 공격을 수행할 수 있는 취약점 | 상 |
| 16 | 세션 예측 | 사용자에게 전달하는 세션 ID가 일정한 패턴을 가지고 있는 경우 공격자가 세션 ID를 추측하여 불법적인 접근을 시도할 수 있는 취약점 | 상 |
| 17 | 불충분한 인가 | 민감한 데이터 또는 기능에 대한 접근 권한 제한을 두지 않은 취약점 | 상 |
| 18 | 불충분한 세션 만료 | 세션의 만료 기간을 정하지 않거나, 만료 기한을 너무 길게 설정한 경우 악의적인 사용자가 만료되지 않은 세션을 활용하여 불법적인 접근이 가능한 취약점 | 상 |
| 19 | 세션 고정 | 사용자 로그인 시 항상 일정하게 고정된 세션 ID가 발행되는 경우 세션 ID를 도용한 비인가자의 접근 및 권한 우회가 가능한 취약점 | 상 |
| 20 | 자동화 공격 | 웹 애플리케이션의 특정 프로세스에 대한 반복적인 요청을 통제하지 않을 경우 무차별 대입 공격으로 인해 사용자 계정을 탈취할 수 있고, 자동화공격으로 게시글 등록 또는 SMS 발송 요청을 반복하여 웹 애플리케이션 자원을 고갈시킬 수 있는 취약점 | 상 |
| 21 | 프로세스 검증 누락 | 인증이 필요한 웹 사이트의 중요(관리자 페이지, 회원 변경 페이지 등) 페이지에 대한 접근 제어가 미흡할 경우 하위 URL 직접 접근, 스크립트 조작 등의 방법으로 중요한 페이지에 대한 접근이 가능한 취약점 | 상 |
| 22 | 파일 업로드 | 파일 업로드 기능을 이용하여 시스템 명령어를 실행시킬 수 있는 웹쉘을 업로드할 수 있는 취약점 | 상 |
| 23 | 파일 다운로드 | 파일 다운로드 기능을 이용하여 첨부된 파일 외에 서버 소스코드 및 시스템 파일 등 주요 파일을 다운로드 할 수 있는 취약점 | 상 |
| 24 | 관리자 페이지 노출 | 관리자 페이지 이름(admin, manager 등)이나 설정, 프로그램 설계상의 오류로 인해 관리자 메뉴에 직접 접근할 수 있는 취약점 | 상 |
| 25 | 경로 추적 | 웹 서버와 웹 애플리케이션의 파일 또는 디렉터리 접근이 통제되지 않아 웹 서버 또는 웹 애플리케이션의 중요한 파일과 데이터에 접근을 허용하는 취약점 | 상 |
| 26 | 위치 공개 | 폴더나 파일명의 위치가 예측 가능하여 쉽게 노출될 경우 공격자는 이를 악용하여 대상에 대한 정보를 획득하고 민감한 데이터에 접근이 가능한 취약점 | 상 |
| 27 | 데이터 평문 전송 | 서버와 클라이언트 간 통신 시 중요 정보가 암호화 과정 없이 평문으로 전송되는 취약점 | 상 |
| 28 | 쿠키 변조 | 클라이언트에 전달되는 쿠키의 사용자 식별 값이 평문으로 노출될 경우 쿠키 변조를 통해 다른 사용자의 유효한 세션을 취득할 수 있으며, 기타 중요 정보의 유출 및 변조가 가능한 취약점 | 상 |

[표 4] “주요정보통신기반시설 기술적 취약점 분석평가 상세가이드” 취약점 점검 항목

* + 1. 전자금융 기반시설 취약점 분석·평가(종합 점검)

| 순번 | 점검 항목 | 점검 내용 | 위험도 |
| --- | --- | --- | --- |
| 1 | 서버 사이드 템플릿  인젝션(SSTI) | 템플릿 엔진에 임의 코드 주입을 통한 원격 코드 실행 등의 위협이 발생될 수 있는 취약점 | 상 |

[표 5] “전자금융 기반시설 취약점 분석·평가(종합 점검)” 취약점 점검 항목

* 1. 사용 도구

|  |  |  |
| --- | --- | --- |
| 구분 | 도구명 | 설명 |
| 프록시 | BurpSuite | 서버로 전송되는 파라미터 값을 확인하고 수정하기 위해 사용 |
| 인젝션 | Sqlmap | SQL 인젝션 공격 시 자동화된 방식을 사용하기 위해 사용 |
| 웹 퍼징 | dirbuster | 웹/응용 프로그램 서버에서 디렉터리 및 파일 이름을 무차별 대입하기 위해 사용 |

[표 6] 웹 취약점 점검 사용 도구

1. 결과 요약
   1. 총평

NO-UH 웹 서비스에서 취약점 점검 결과 SQL 인젝션, 악성 콘텐츠, 크로스사이트 스크립팅, 약한 문자열 강도, 크로스사이트 리퀘스트 변조(CSRF), 불충분한 세션 만료, 자동화 공격, 파일 업로드, 파일 다운로드, 위치 공개, 서버 사이드 템플릿 인젝션(SSTI)으로 전체 29개의 항목에서 총 11건의 취약점이 발견되었습니다.

SQL injection의 경우는 입력 값 검증 미흡으로 참/거짓 질의문이 삽입 가능하여 DB 내 정보 추출이 가능하고, 악성 컨텐츠의 경우는 웹 사이트 게시판, 댓글 등의 정상적인 콘텐츠 대신 악성 콘텐츠를 주입하여 실행될 경우 사용자가 해당 콘텐츠 열람 시 악성코드 감염 및 웹 페이지 변조 등의 위험에 노출될 수 있습니다. 크로스 사이트 스크립팅의 경우 웹 애플리케이션의 입력 값을 통해 다른 최종 사용자의 클라이언트에서 임의의 스크립트가 실행되며, 약한 문자열 강도의 경우 유추가 용이한 계정 및 패스워드의 사용으로 인한 사용자 권한 탈취 위험이 존재합니다. 크로스사이트 리퀘스트 변조(CSRF)의 경우 사용자의 신뢰(인증) 정보 내에서 사용자의 요청(Request)을 변조함으로써 해당 사용자의 권한으로 악의적인 공격을 수행할 수 있으며, 불충분한 세션 만료의 경우 세션의 만료 기간을 정하지 않거나, 만료 기한이 너무 길게 설정된 경우 악의적인 사용자가 만료되지 않은 세션을 활용하여 불법적인 접근이 가능합니다. 자동화공격의 경우 웹 애플리케이션의 특정 프로세스에 대한 반복적인 요청을 통제하지 않을 시 무차별 대입 공격으로 인해 사용자 계정을 탈취할 수 있고, 자동화공격으로 게시글 등록 또는 SMS 발송 요청을 반복하여 웹 애플리케이션 자원을 고갈시킬 수 있습니다. 파일 업로드의 경우 파일 업로드 기능을 이용하여 시스템 명령어를 실행시킬 수 있는 웹쉘을 업로드할 수 있으며, 파일 다운로드의 경우 파일 다운로드 기능을 이용하여 첨부된 파일 외에 서버 소스코드 및 시스템 파일 등 주요 파일을 다운로드 할 수 있습니다. 위치 공개의 경우 폴더나 파일명의 위치가 예측 가능하며 쉽게 노출될 경우 공격자는 이를 악용하여 대상에 대한 정보를 획득하고 민감한 데이터에 접근이 가능한 취약점입니다. 서버 사이드 템플릿 인젝션(SSTI)의 경우 템플릿 엔진에 임의 코드 주입을 통한 원격 코드 실행 등의 위협이 발생될 수 있는 취약점이므로 보안 조치가 필요합니다.

* 1. 취약점 요약

도출된 취약점은 다음과 같습니다.

|  |  |  |
| --- | --- | --- |
| 대상 | 취약점 항목 | 취약 여부 |
| NO-UH  (no-uh.com) | 버퍼 오버플로우 | **양호** |
| 포맷스트링 | **양호** |
| LDAP 인젝션 | **양호** |
| 운영체제 명령 실행 | **양호** |
| SQL 인젝션 | **취약** |
| SSI 인젝션 | **양호** |
| XPath 인젝션 | **양호** |
| 디렉토리 인덱싱 | **양호** |
| 정보 누출 | **양호** |
| 악성 콘텐츠 | **취약** |
| 크로스사이트 스크립팅 | **취약** |
| 약한 문자열 강도 | **취약** |
| 불충분한 인증 | **양호** |
| 취약한 패스워드 복구 | **양호** |
| 크로스사이트 리퀘스트 변조(CSRF) | **취약** |
| 세션 예측 | **양호** |
| 불충분한 인가 | **양호** |
| 불충분한 세션 만료 | **취약** |
| 세션 고정 | **양호** |
| 자동화 공격 | **취약** |
| 프로세스 검증 누락 | **양호** |
| 파일 업로드 | **취약** |
| 파일 다운로드 | **취약** |
| 관리자 페이지 노출 | **양호** |
| 경로 추적 | **양호** |
| 위치 공개 | **취약** |
| 데이터 평문 전송 | **양호** |
| 쿠키 변조 | **양호** |
| 서버 사이드 템플릿 인젝션(SSTI) | **취약** |

[표 7] 취약점 요약

**※ 취약점이 발견된 모듈의 경우 동일 모듈이 존재할 시 확산 조치해야 함.**

1. 상세 결과
   1. NO-UH.com
      1. SQL 인젝션

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | SQL 인젝션 | 중요도 | | 상 |
| 취약점 설명 | | 악의적인 명령을 실행하는 임의의 쿼리를 삽입해 공격이 가능한 취약점 | | | |
| 영향 효과 | | 인증 우회, 데이터베이스 접근과 같은 공격이 가능하기 때문에 데이터 유출, 변조, 삭제 등의 피해 가능성 존재 | | | |
| 권고 사항 | | 1. Prepared Statement 사용  2. 입력 값 필터링 및 정제  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.1 SQL 인젝션 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/board/lists | | | title, content | |
| 2 | https://no-uh.com/quiz/lists | | | question, answer, opt1, opt2, opt3, comment | |
| 3 | https://no-uh.com/lecture/lists | | | subject, title, file | |

[표 8] SQL 인젝션 - 취약점 요약

**[취약점 검증 절차]**

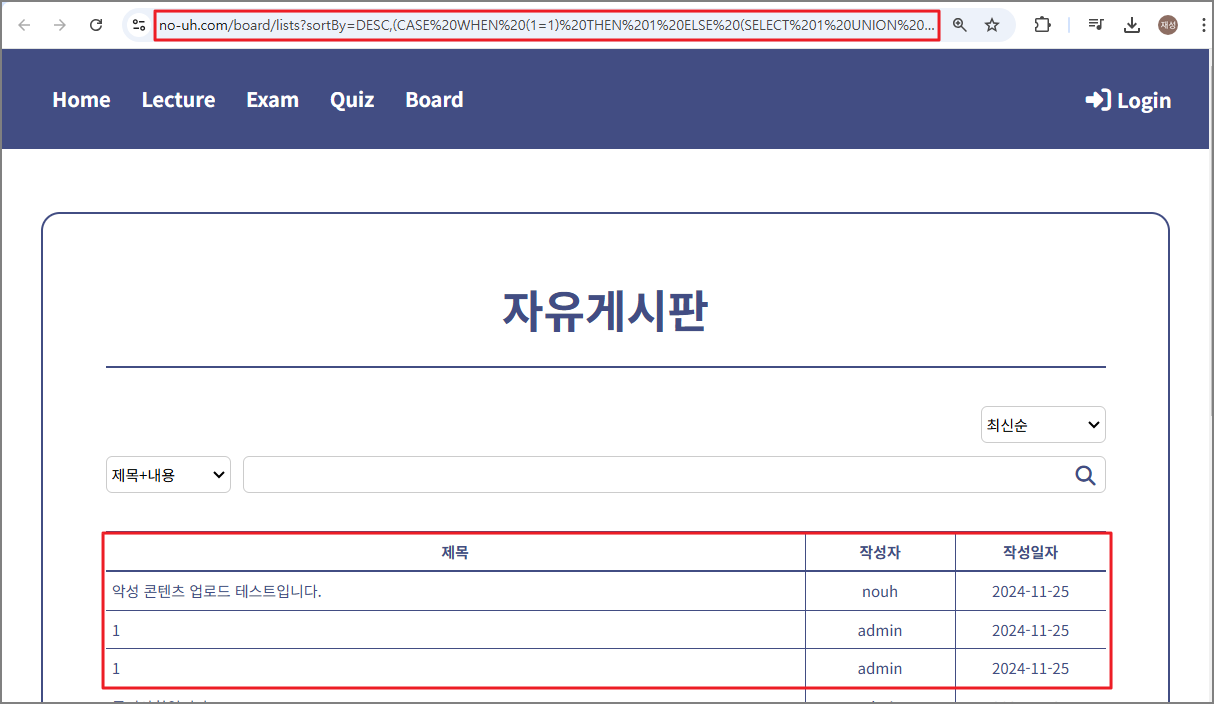
Step 1. “Board“ 의 자유게시판 목록 확인



[그림 1] 악의적 구문 삽입 지점 확인

Step 2. 악의적 구문 참 값 삽입 시 게시글 목록 확인

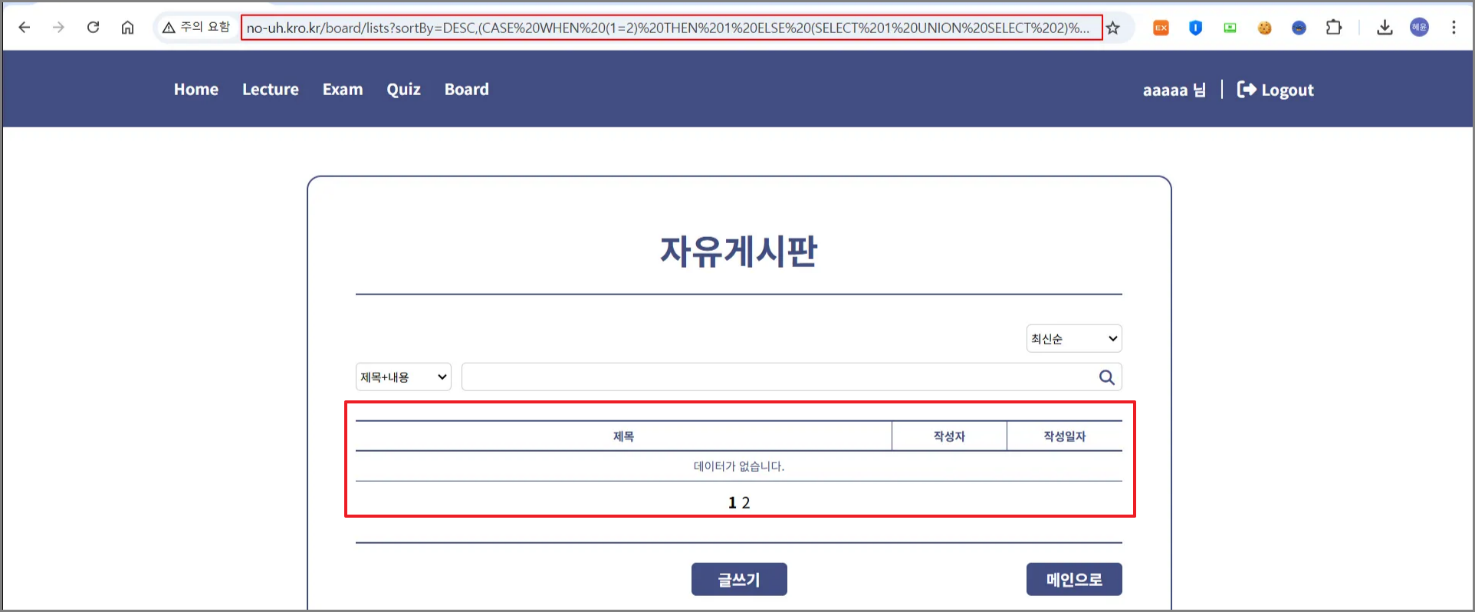
악의적 구문: {sortBy=DESC,(CASE WHEN (1=1) THEN 1 ELSE (SELECT 1 UNION SELECT 2) END)&searchBy=0&keyword=}



[그림 2] SQL 인젝션 참 값 삽입

Step 3. 악의적 구문 거짓 값 삽입 시 게시글 목록 확인

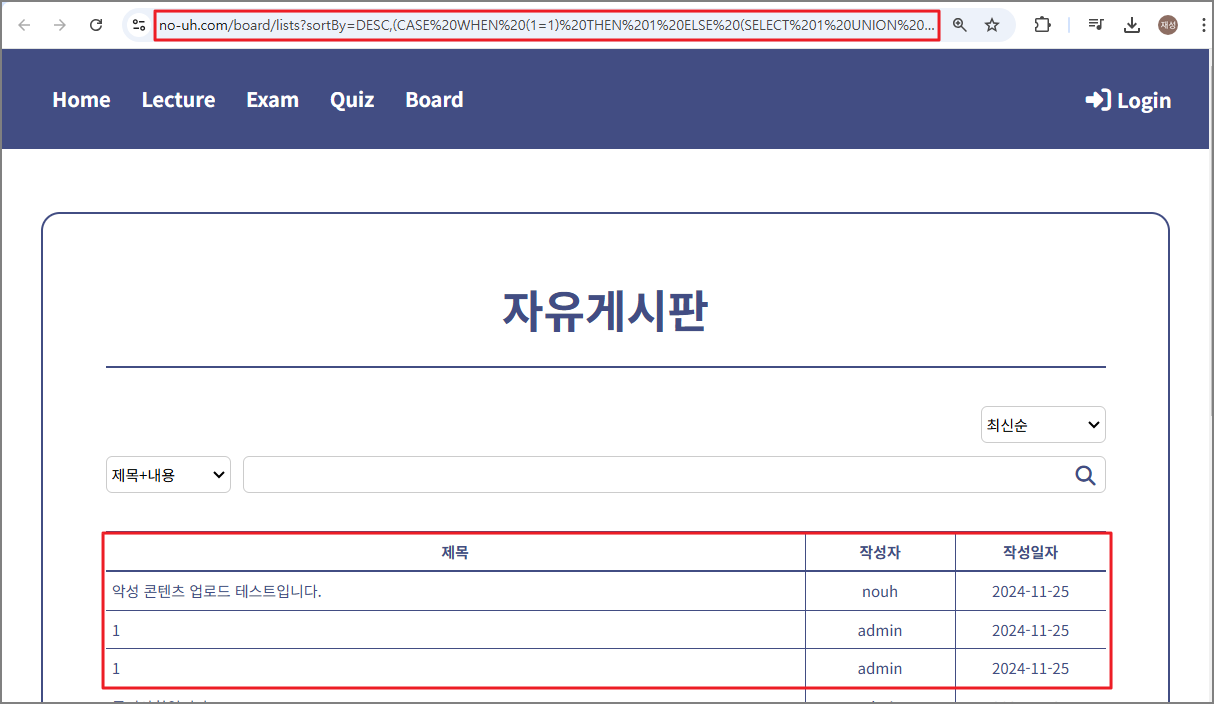
악의적 구문: {sortBy=DESC,(CASE WHEN (1=2) THEN 1 ELSE (SELECT 1 UNION SELECT 2) END)&searchBy=0&keyword=}



[그림 3] SQL 인젝션 거짓 값 삽입

Step 4. SQL 인젝션을 통한 데이터베이스 값 추출

악의적 구문: {sortBy=DESC,(CASE WHEN (SUBSTRING(DATABASE(), 1, 1) = 'N') THEN 1 ELSE (SELECT 1 UNION SELECT 2) END)&searchBy=0&keyword=}



[그림 4] SQL 인젝션을 통한 데이터 베이스명 추출

자유게시판 페이지의 URL에서 데이터베이스명을 알아내기 위해 sortBy 파라미터를 다음과 같이 수정하여 요청함.

|  |
| --- |
| # 데이터베이스 길이 추출 (6자리)  http://no-uh.kro.kr/board/lists?sortBy=DESC,(CASEWHEN(LENGTH(DATABASE()) = 6) THEN 1 ELSE (SELECT 1 UNION SELECT 2) END)&searchBy=0&keyword=  # (a=1, N=110), (a=2, N=111), (a=3, N=117, (a=4, N=104), (a=5, N=100), (a=6, N=98)  # 데이터베이스명: nouhdb  https://no-uh.com/board/lists?sortBy=DESC,(CASE WHEN (SUBSTRING(DATABASE(), a, 1) = 'N') THEN 1 ELSE (SELECT 1 UNION SELECT 2) END)&searchBy=0&keyword= |

[표 9] 데이터베이스명 탈취 명령어

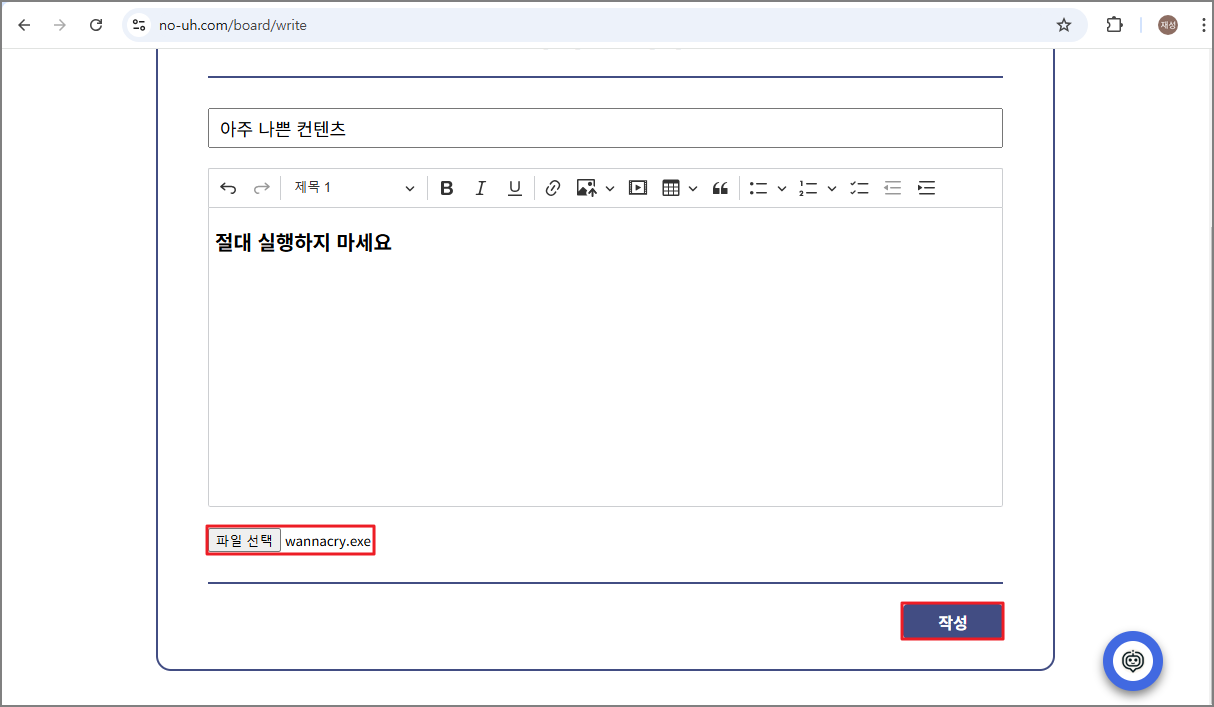
* + 1. 악성 콘텐츠

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 악성 콘텐츠 | 위험도 | | 상 |
| 취약점 설명 | | 파일 업로드 사용 시 악의적인 파일 업로드가 가능 | | | |
| 영향 효과 | | 해당 콘텐츠 열람 시 악성코드 감염 및 웹 페이지 변조 등의 가능성 존재 | | | |
| 권고 사항 | | 1. 기능에 Flash 파일이나 avi 동영상 파일, exe 실행 파일 등 악성코드가 포함될 수 있는 콘텐츠를 삽입 또는 업로드 하지 못하게 필터링을 적용  2. 주기적으로 업로드 된 파일을 대상으로 바이러스 검사를 실시  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.2 악성 콘텐츠 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/board/write | | | title, content | |
| 2 | https://no-uh.com/lecture/create | | | subject, title, file | |

[표 10] 악성 콘텐츠 - 취약점 요약

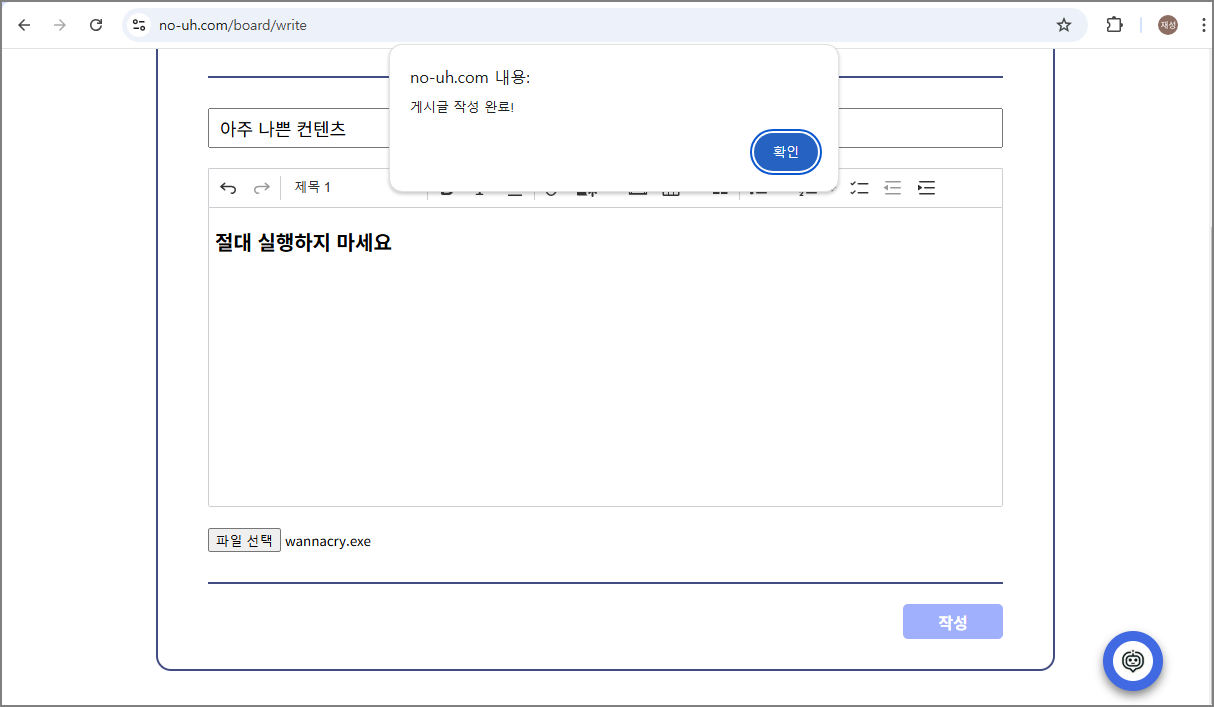
**[취약점 검증 절차]**

Step 1. “Board“ 에서 글쓰기를 통해 악성 콘텐츠를 업로드



[그림 5] 악성 컨텐츠 업로드

Step 2. 악성 컨텐츠 삽입된 게시글 작성 완료



[그림 6] 악성 컨텐츠 삽입 완료

Step 3. 악성 콘텐츠가 업로드 된 게시물을 확인



[그림 7] 악성 콘텐츠 게시글 확인

Step 4. 서버 측에서 악성 콘텐츠 파일의 삭제 여부를 확인



[그림 8] 악성 콘텐츠 파일 삭제 여부 확인

* + 1. 크로스사이트 스크립팅

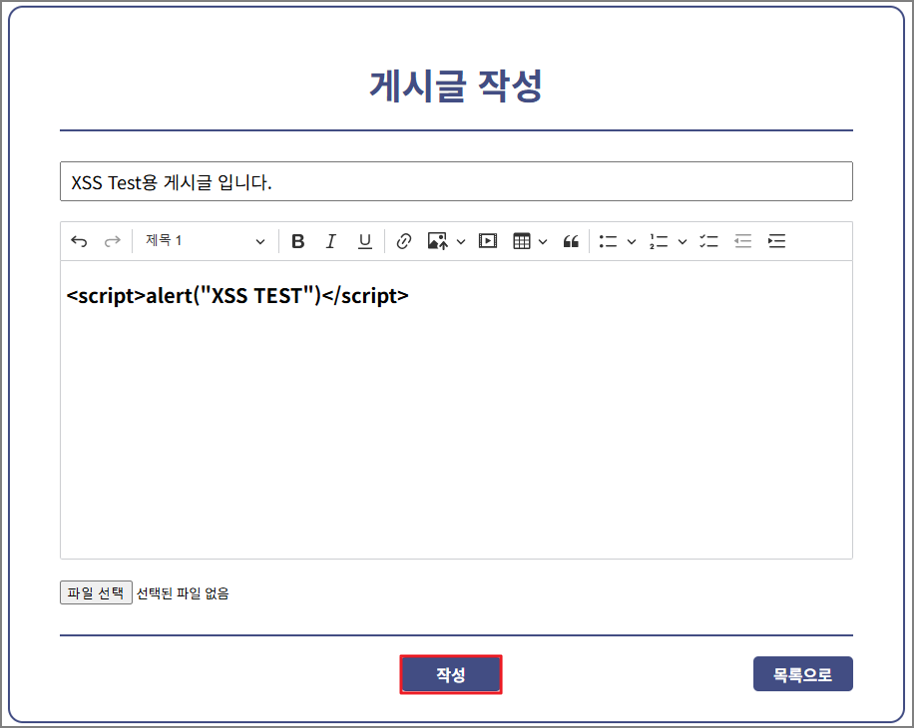
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 크로스사이트 스크립팅 | 위험도 | | 상 |
| 취약점 설명 | | 글쓰기 시 악의적인 구문 삽입이 가능한 취약점 | | | |
| 영향 효과 | | 쿠키 값 또는 세션 등 사용자의 정보를 탈취하거나 피싱 사이트로의 접근 유도 등 사용자에게 직접적인 피해 존재 | | | |
| 권고 사항 | | 1. 특수문자 치환  2. XSS 외부 라이브러리 활용  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.3 크로스사이트 스크립팅 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/board/write | | | title, content | |

[표 11] 크로스사이트 스크립팅 - 취약점 요약

**[취약점 검증 절차]**

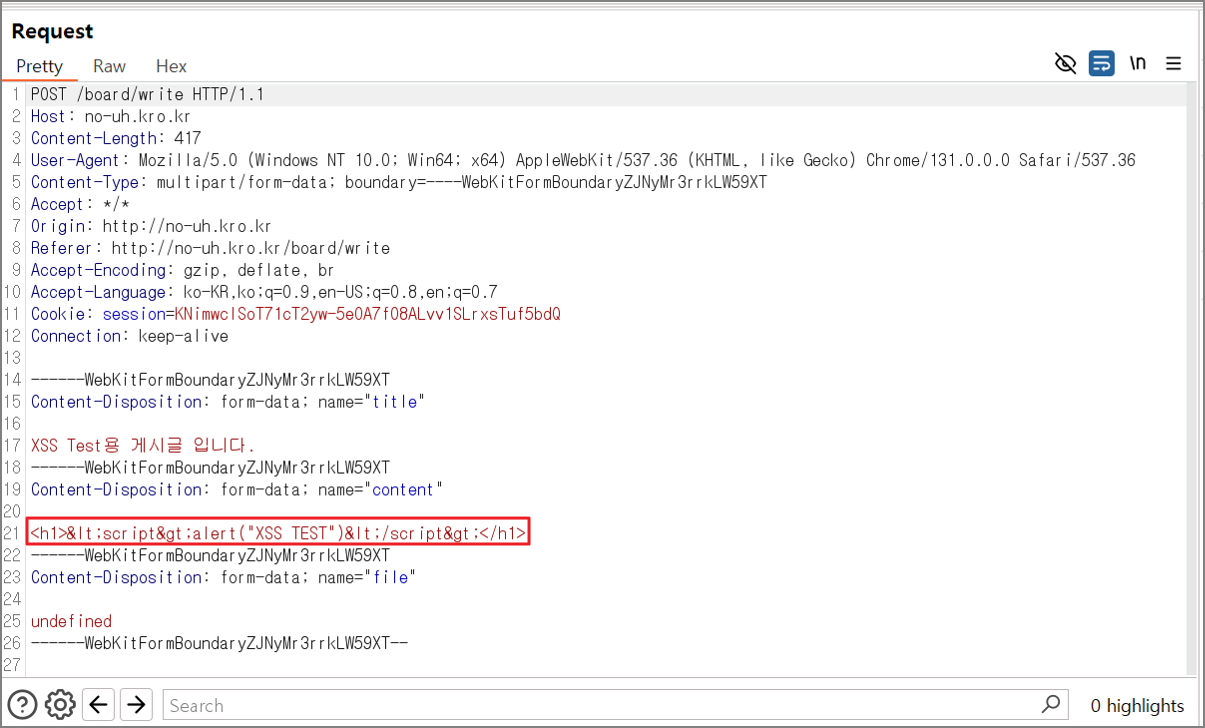
Step 1. “Board > 글쓰기“에서 악성 구문 삽입

악의적 구문: {<script>alert(“XSS TEST”)</script>}



[그림 9] 게시글 작성 페이지 악의적 구문 삽입

Step 2. Burp Suite를 통해 특수문자 필터링된 내용 확인



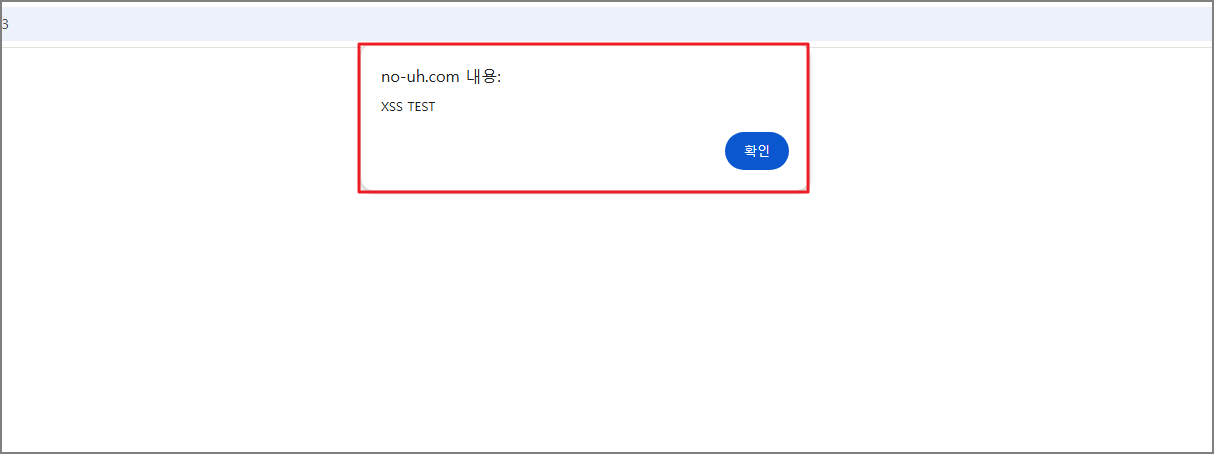
[그림 10] XSS – 필터링된 내용 확인

Step 3. Burp Suite를 통해 필터링된 내용을 재수정



[그림 11] XSS - 필터링된 내용 수정

Step 4. 악의적인 구문이 삽입된 게시글 확인



[그림 12] XSS 작동

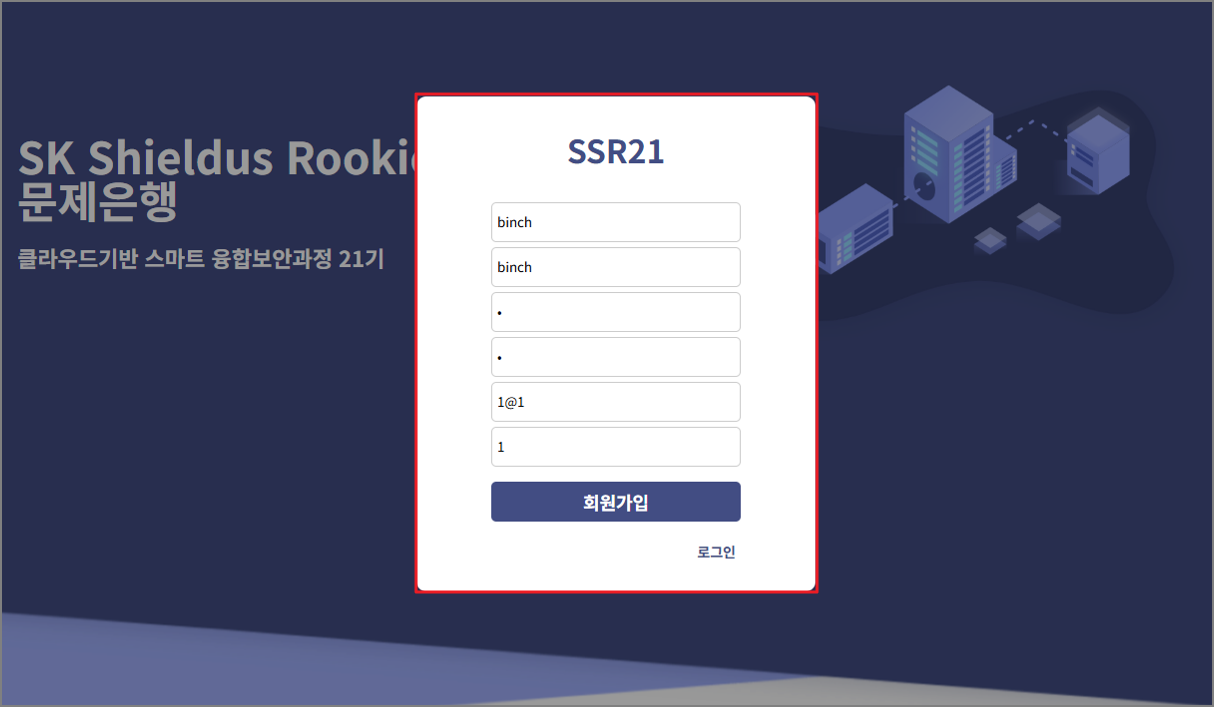
* + 1. 약한 문자열 강도

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 약한 문자열 강도 | 위험도 | | 상 |
| 취약점 설명 | | 유추 가능한 취약한 문자열 사용을 통해 회원가입이 가능 | | | |
| 영향 효과 | | 유추가 용이한 계정 및 패스워드의 사용으로 인한 사용자 권한 탈취 위험이 존재 | | | |
| 권고 사항 | | 1. Prepared Statement 사용  2. 입력 값 필터링 및 정제  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.4 약한 문자열 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/member/register | | | userName, userId, userPw, checkPw, userEmail, userPhone | |
| 2 | https://no-uh.com/member/login | | | userId, userPw | |

[표 12] 약한 문자열 강도 - 취약점 요약

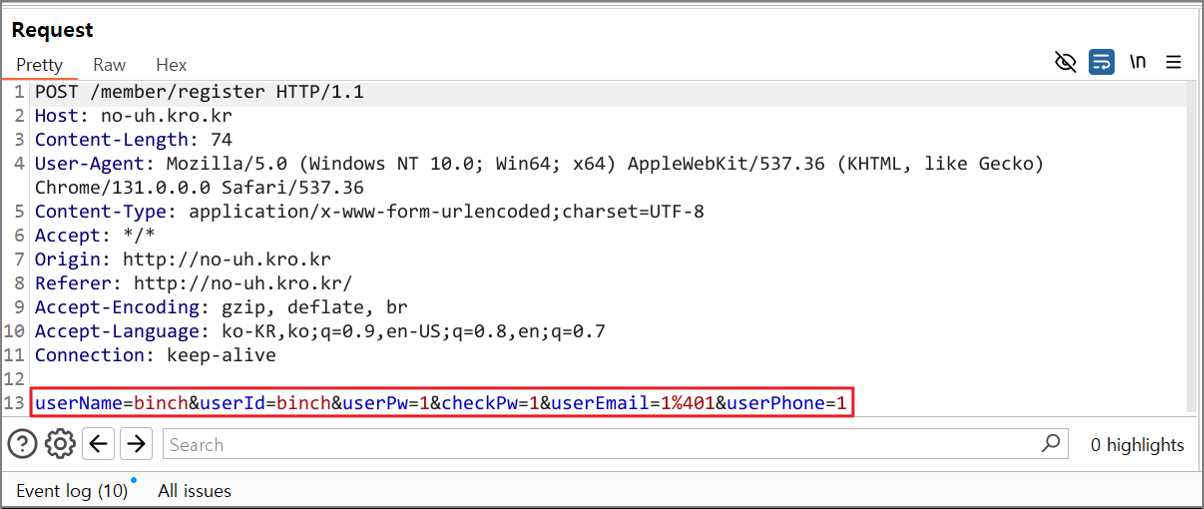
**[취약점 검증 절차]**

Step 1. “Login > 회원가입“ 에서 짧은 패스워드 입력의 회원가입 가능 여부 확인



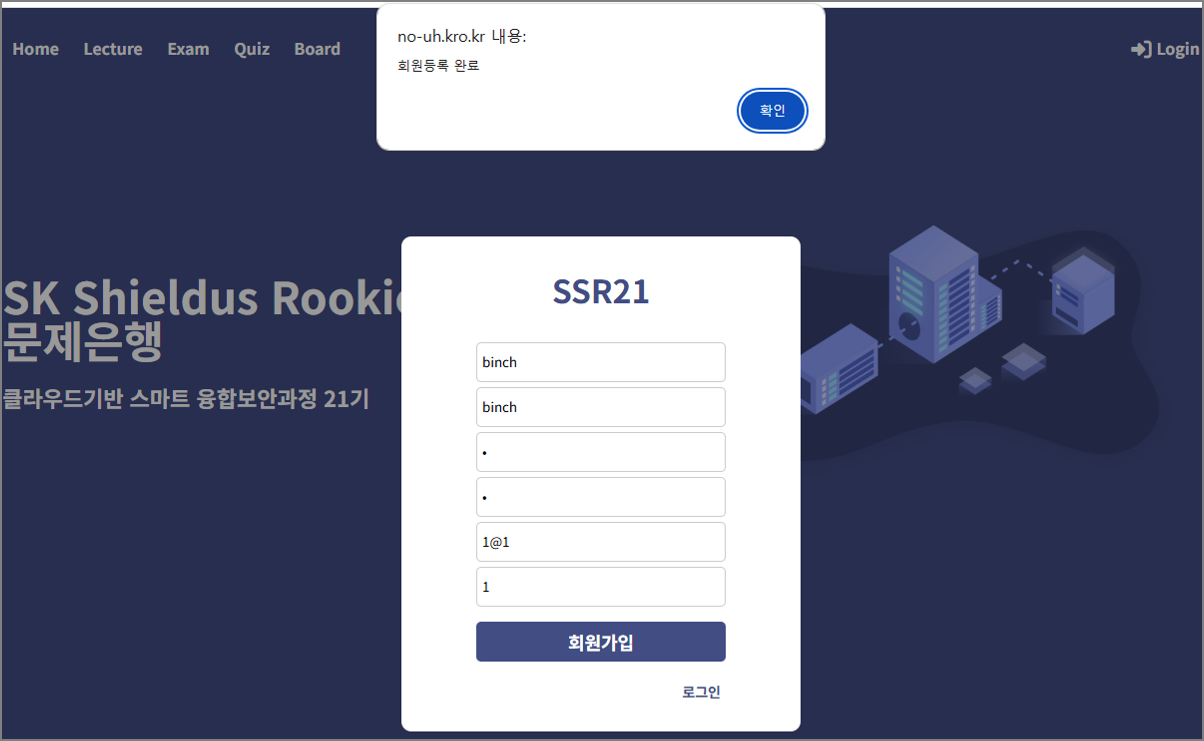
[그림 13] 약한 문자열로 회원가입 진행

Step 2. Burp Suite에서 userPw값 “1” 삽입 확인



[그림 14] 회원가입 데이터

Step 3. 회원가입이 완료된 모습



[그림 15] 회원가입 완료

* + 1. 크로스사이트 리퀘스트 변조(CSRF)

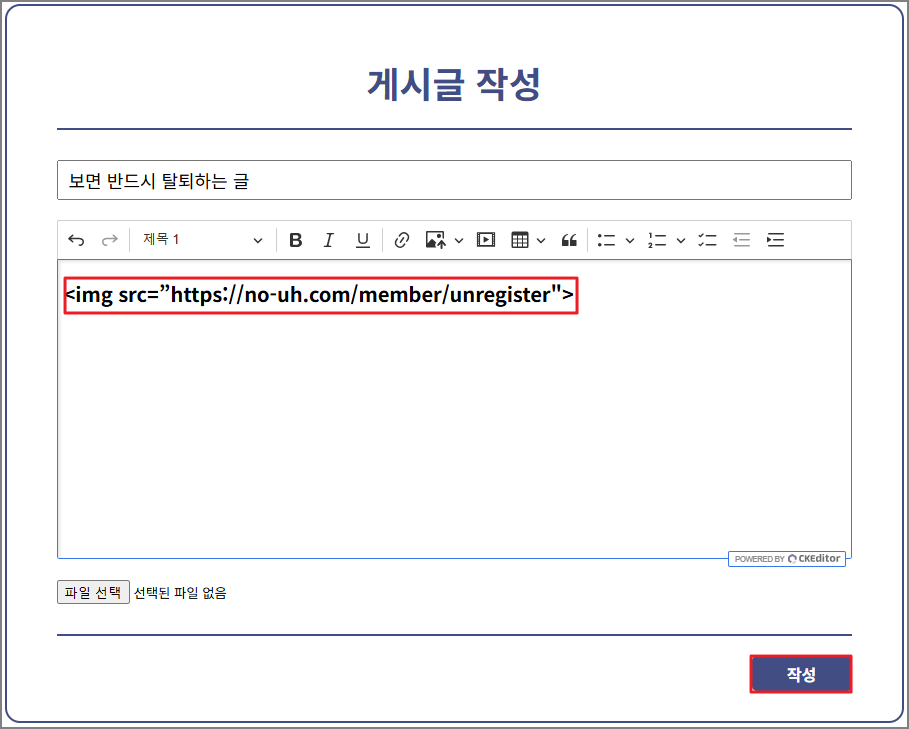
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 크로스사이트 리퀘스트 변조(CSRF) | 위험도 | | 상 |
| 취약점 설명 | | 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)를 특정 웹 사이트에 요청하게 함 | | | |
| 영향 효과 | | 해당 사용자의 권한으로 악의적인 공격을 수행할 수 있음 | | | |
| 권고 사항 | | SMS 또는 OTP 2차인증을 사용하거나 번거로울 경우 csrf 토큰을 사용 권고  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.5 크로스사이트 리퀘스트 변조(CSRF) 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/board/write | | | title, content | |

[표 13] 크로스사이트 리퀘스트 변조(CSRF) - 취약점 요약

**[취약점 검증 절차]**

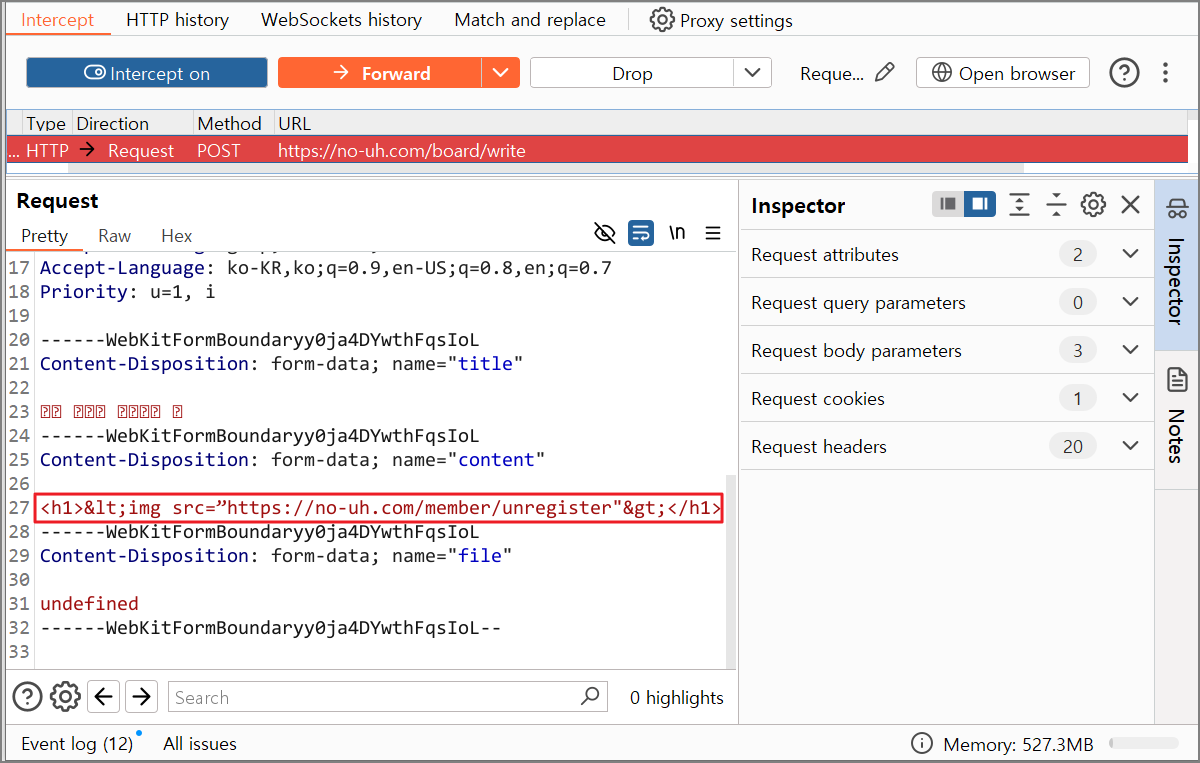
Step 1. “게시글 > 글쓰기” 내에서 악의적인 구문 작성 후 게시

악의적 구문: {<img src=”https://no-uh.com/member/unregister”>}



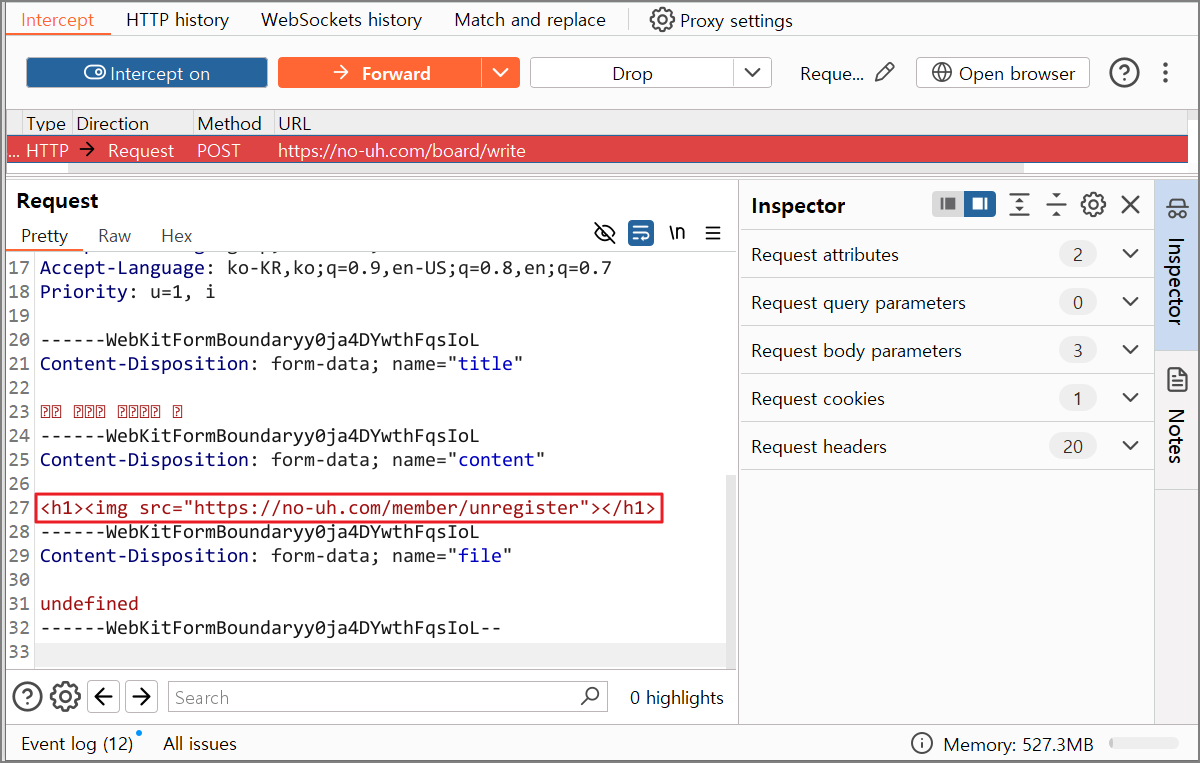
[그림 16] CSRF 악의적인 구문 삽입

Step 2. Burp Suite를 통해 특수문자 필터링된 내용 확인



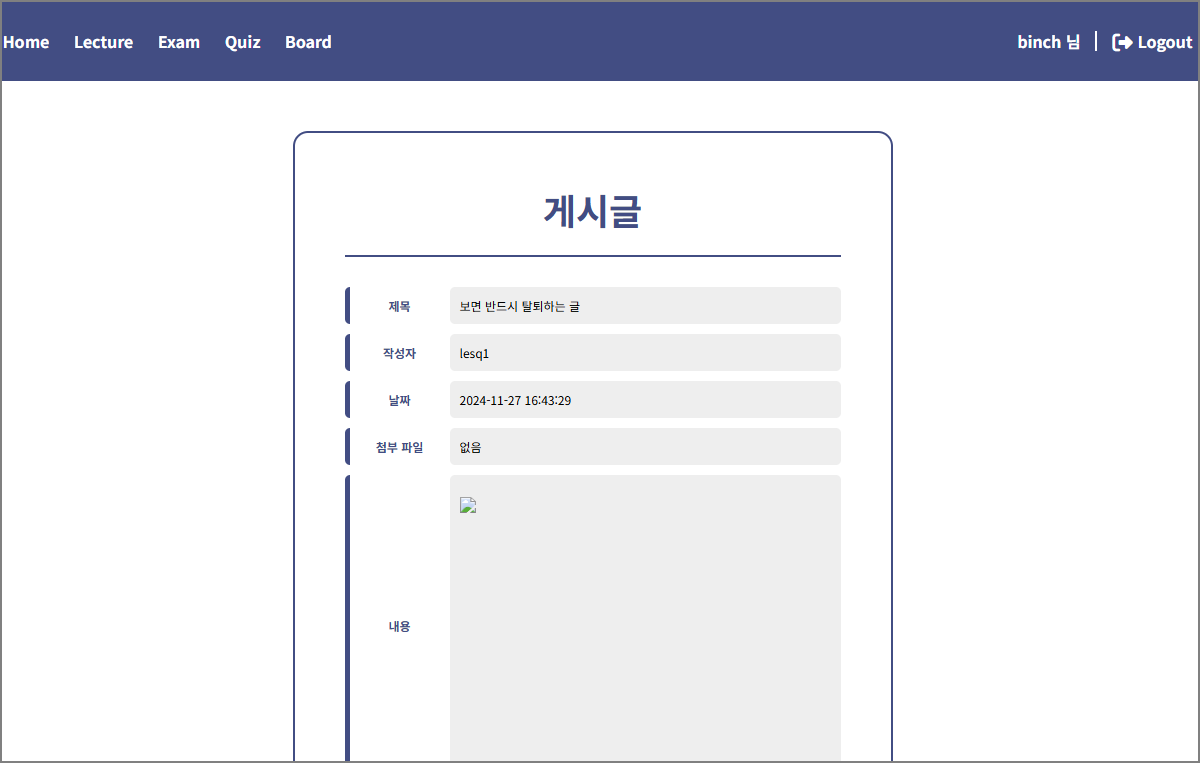
[그림 17] CSRF - 필터링 된 내용 확인

Step 3. Burp Suite를 통해 필터링된 내용 수정



[그림 18] CSRF - 필터링 된 내용 수정

Step 4. 작성된 게시글을 확인 시 회원탈퇴가 이루어짐



[그림 19] 회원탈퇴 악의적 구문 삽입 게시글 확인

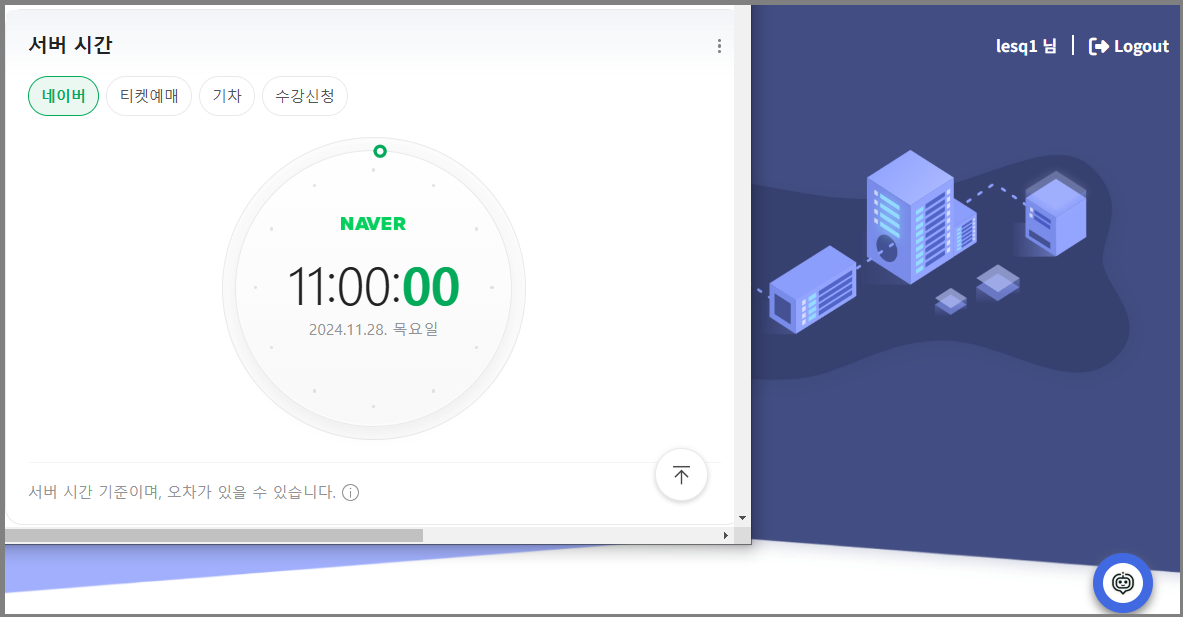
* + 1. 불충분한 세션 만료

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 불충분한 세션 만료 | 위험도 | | 상 |
| 취약점 설명 | | 세션의 만료 기간을 정하지 않거나, 만료 기한을 너무 길게 설정하여 발생하는 취약점 | | | |
| 영향 효과 | | 악의적인 사용자가 만료되지 않은 세션을 활용하여 불법적인 접근이 가능 | | | |
| 권고 사항 | | 1. 세션 타임아웃 구현 시 타임아웃 시간은 10분으로 설정  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.6 불충분한 세션 만료 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com | | |  | |

[표 14] 불충분한 세션 만료 - 취약점 요약

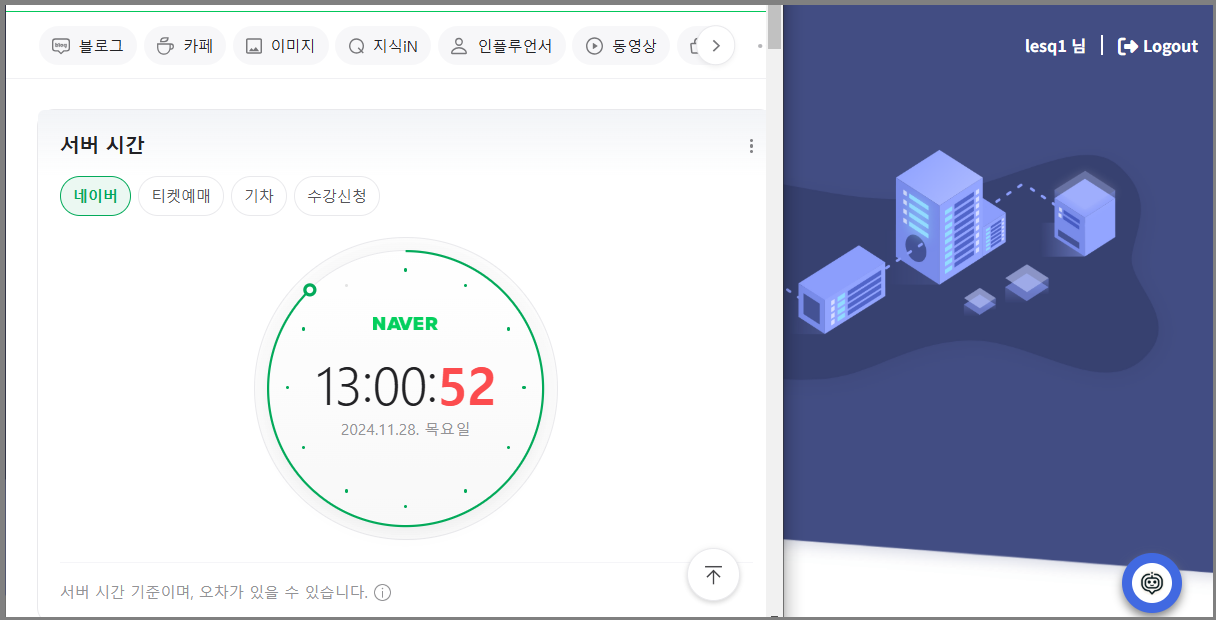
**[취약점 검증 절차]**

Step 1. 로그인한 페이지 확인 (시간: 11:00am)



[그림 20] 로그인 후 로그인 여부 확인

Step 2. 2시간 뒤에도 계정 로그아웃이 이뤄지지 않음 (시간: 13:00pm)



[그림 21] 로그인 후 2시간 뒤 로그아웃 여부 확인

* + 1. 자동화 공격

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 자동화 공격 | 위험도 | | 상 |
| 취약점 설명 | | 웹 애플리케이션의 특정 프로세스(로그인 시도, 게시글 등록, SMS 발송 등)에 대한 반복적 요청을 보내는 취약점 | | | |
| 영향 효과 | | 1. 무차별 대입 공격으로 인해 사용자 계정을 탈취 가능  2. 게시글 등록 또는 SMS 발송 요청을 반복하여 웹 애플리케이션 자원 고갈 | | | |
| 권고 사항 | | 1. 웹 애플리케이션의 특정 프로세스에 대한 대량 사용 통제 로직 구현  2. 웹 방화벽 룰셋을 통해 대량의 불특정 프로세스 요청 차단  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.7 자동화 공격 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/quiz/create | | | question, answer, opt1, opt2, opt3, comment | |
| 2 | https://no-uh.com/lecture/create | | | subject, title, file | |
| 3 | https://no-uh.com/board/write | | | title, content | |
| 4 | https://no-uh.com/member/login | | | userId, userPw | |

[표 15] 자동화 공격 - 취약점 요약

**[취약점 검증 절차]**

Step 1. “Quiz > 추가“ 에서 반복적으로 작성될 데이터 작성



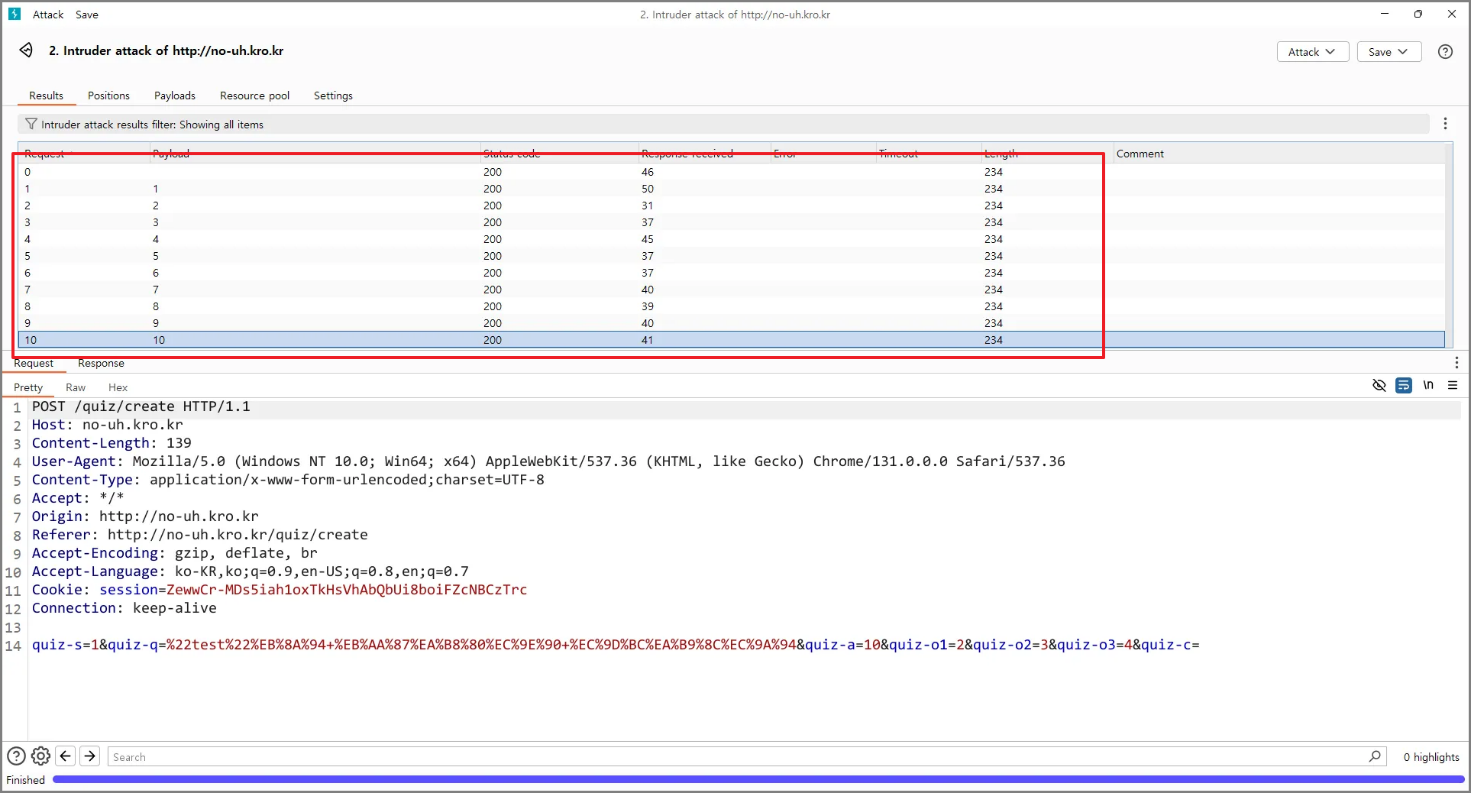
[그림 22] 반복적으로 작성될 데이터 작성

Step 2. Burp Suite를 통해 작성될 데이터 및 퀴즈 테이블의 PK값 확인



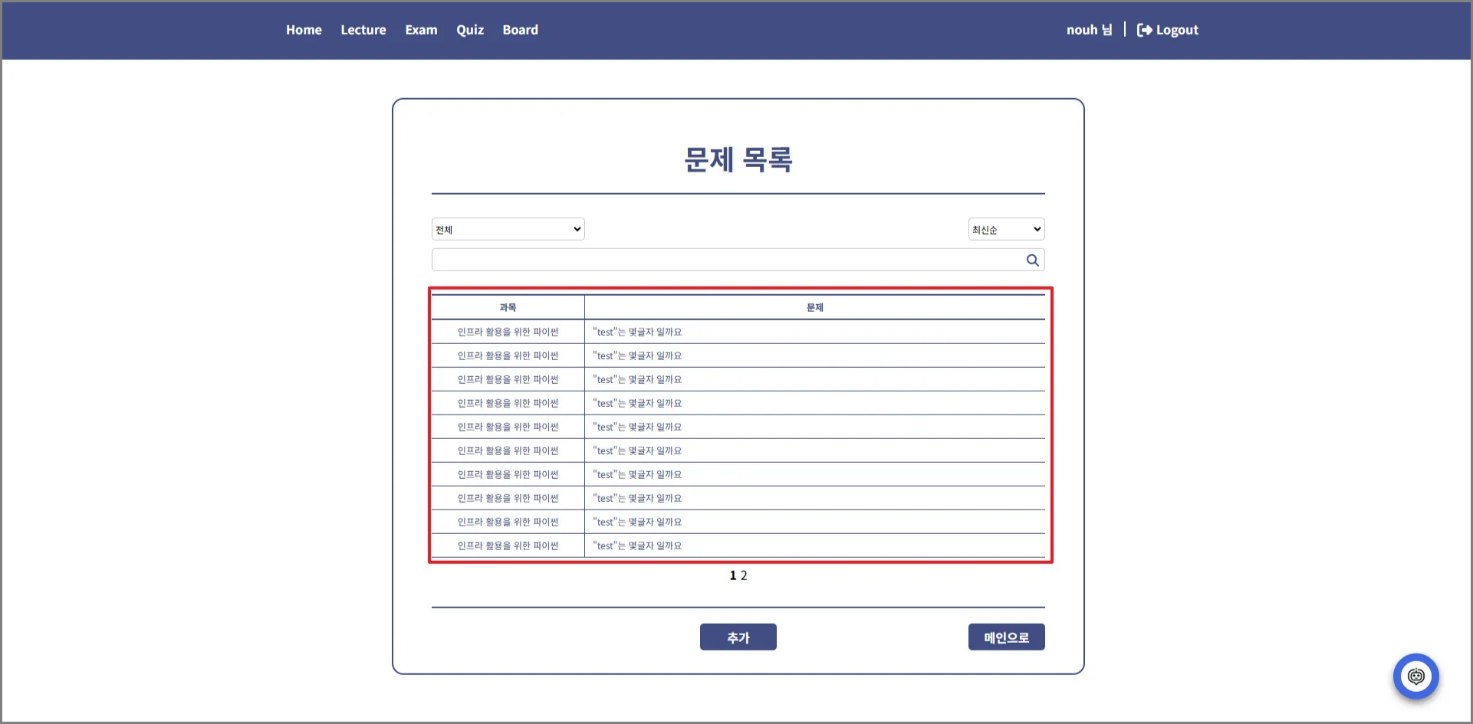
[그림 23] 퀴즈 테이블의 PK값 확인

Step 3. Burp Suite의 intruder 기능을 활용하여 10개의 게시글 작성 자동화



[그림 24] 자동화 공격 - 10개의 게시글 작성

Step 4. 자동화 공격을 통해 작성된 10개의 게시글 확인(반복적인 요청 방어 메커니즘 취약)



[그림 25] 작성된 10개의 게시글 확인

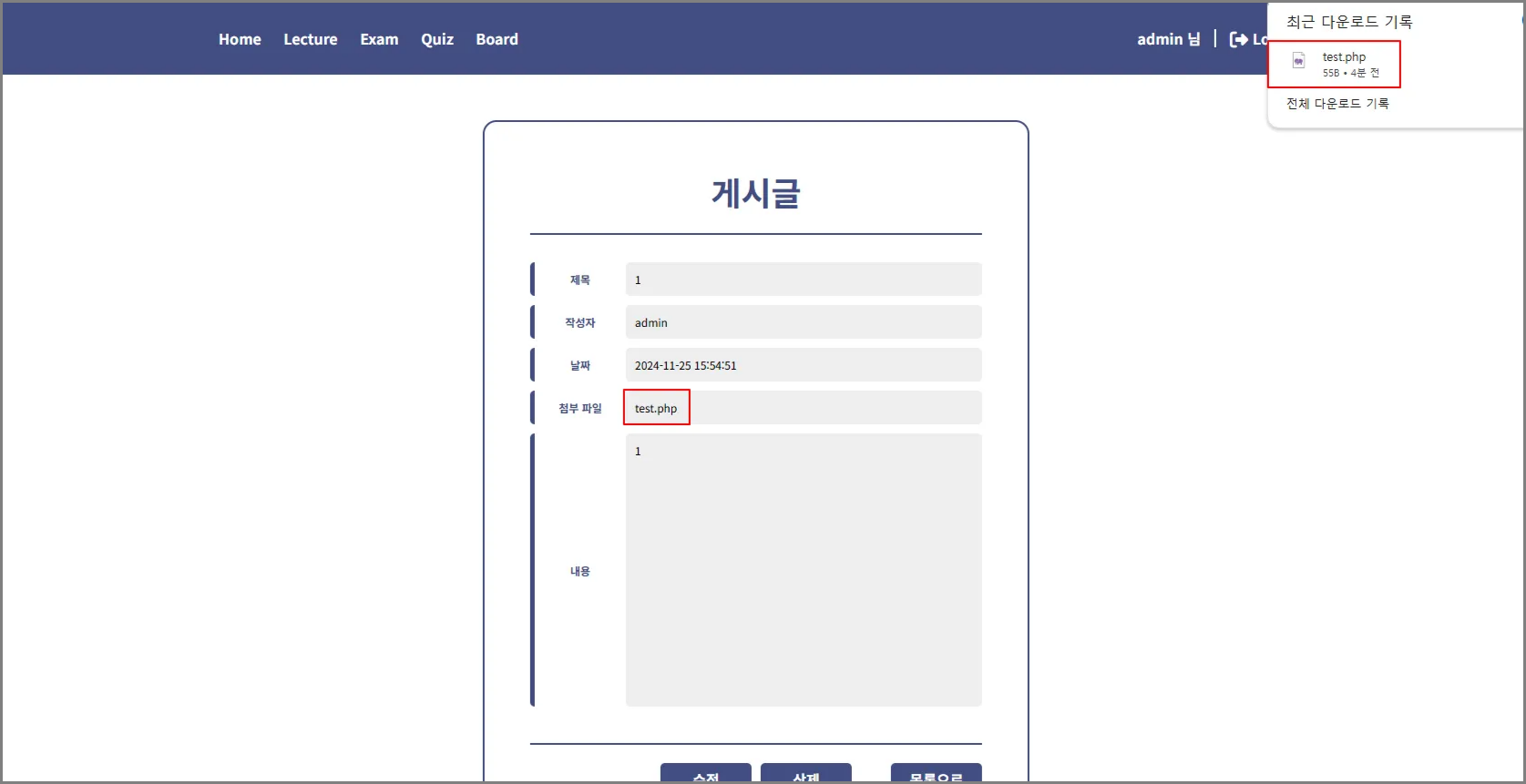
* + 1. 파일 업로드

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 파일 업로드 | 위험도 | | 상 |
| 취약점 설명 | | 조작된 Server Side Script 파일을 서버에 업로드 및 실행하는 취약점 | | | |
| 영향 효과 | | 시스템 관리자 권한 획득 또는 인접 서버에 대한 침입을 시도할 수 있음 | | | |
| 권고 사항 | | 1. 업로드 되는 파일에 대한 확장자 검증 및 실행 권한 제거  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.8 파일 업로드 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/board/write | | | title, content | |
| 3 | https://no-uh.com/lecture/create | | | subject, title, file | |

[표 16] 불충분한 인가(1) - 취약점 요약

**[취약점 검증 절차]**

Step 1. “Board > 추가“ 에서 php파일이 업로드 되고, 다운로드가 가능함



[그림 26] php 파일 업로드 완료

* + 1. 파일 다운로드

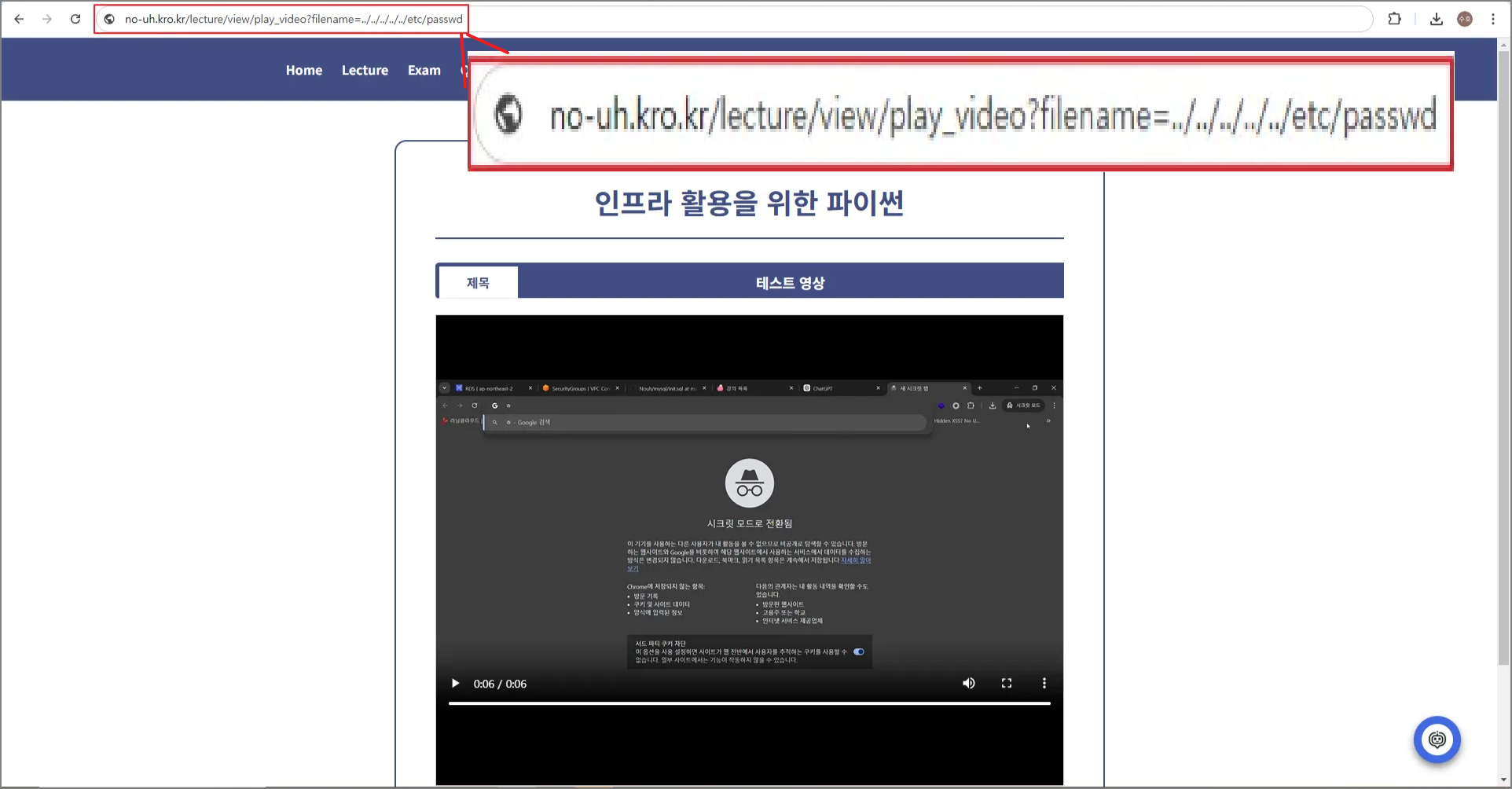
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 파일 다운로드 | 위험도 | | 상 |
| 취약점 설명 | | 애플리케이션의 파라미터 값을 조작하여 의도에서 벗어난 파일 다운로드 | | | |
| 영향 효과 | | 1. 중요한 파일(DB 커넥션 파일, 애플리케이션 파일 등) 다운로드 가능  2. 웹 서버 루트에 있는 중요한 설정 파일(passwd, shadow 등)을 다운로드 가능 | | | |
| 권고 사항 | | 1. 다운로드 시 허용된 경로 이외의 디렉터리와 파일에 접근할 수 없도록 구현  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.9 파일 다운로드 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 3 | https://no-uh.com/lecture/create | | | subject, title, file | |

[표 17] 파일 다운로드 - 취약점 요약

**[취약점 검증 절차]**

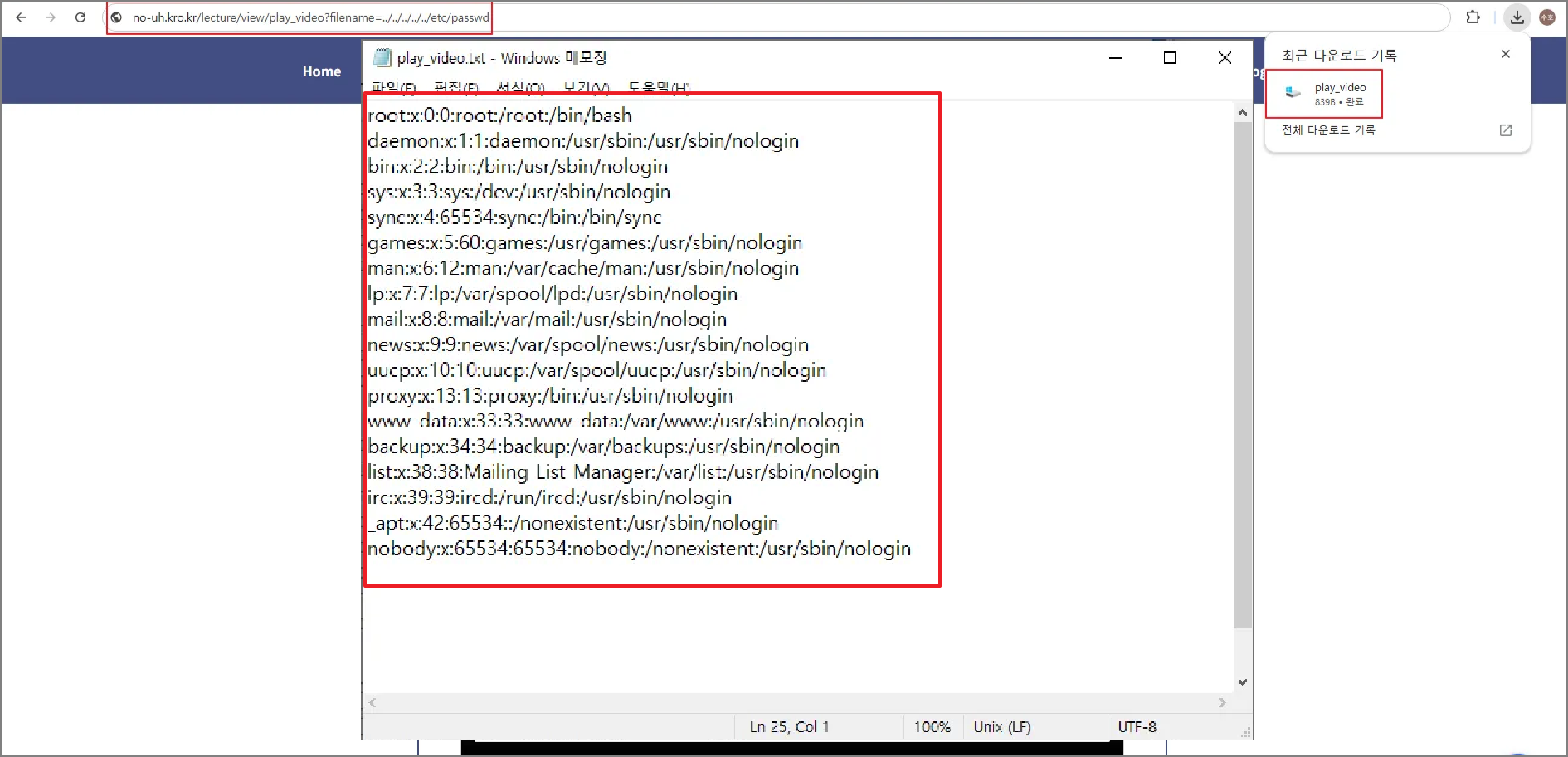
Step 1. “Lecture > view“에서 URL에 악성 구문 삽입

악의적 구문: {../../../../../etc/passwd}



[그림 27] URL에 공격 구문 삽입

Step 2. /etc/passwd 파일 다운로드 확인



[그림 28] /etc/passwd 파일 다운로드

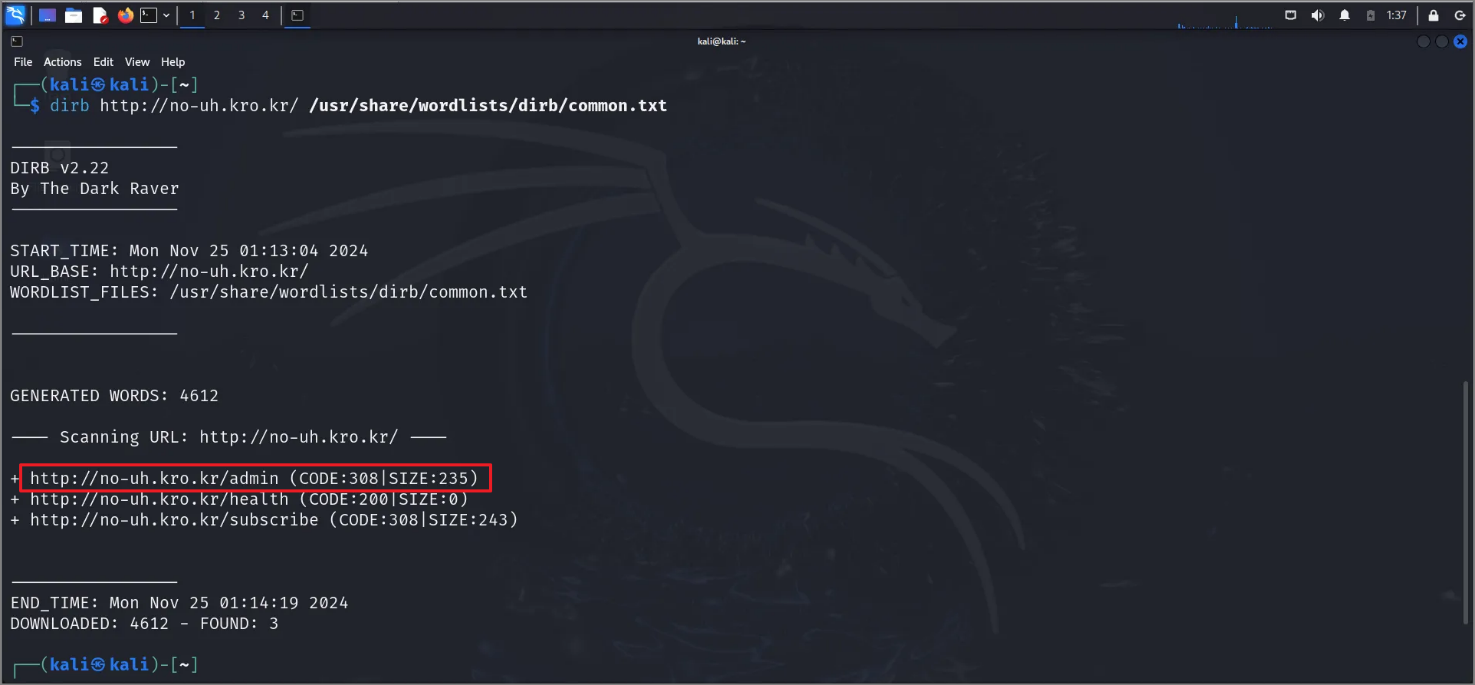
* + 1. 위치 공개

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 취약점 명 | | 위치 공개 | 위험도 | | 상 |
| 취약점 설명 | | 폴더나 파일명의 위치가 예측 가능하여 쉽게 노출되는 취약점 | | | |
| 영향 효과 | | 대상에 대한 정보를 획득하고 민감한 데이터에 접근 가능 | | | |
| 권고 사항 | | 1. 웹 루트 디렉터리 이하 모든 불필요한 파일 및 샘플 페이지 삭제  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.10 위치 공개 참고** | | | |
| No | 발견 URL 및 경로 | | | 파라미터 | |
| 1 | https://no-uh.com/admin | | | admin | |

[표 18] 위치 공개 - 취약점 요약

**[취약점 검증 절차]**

Step 1. DirBuster를 통해 /admin 페이지가 누출됨



[그림 29] /admin 페이지 누출

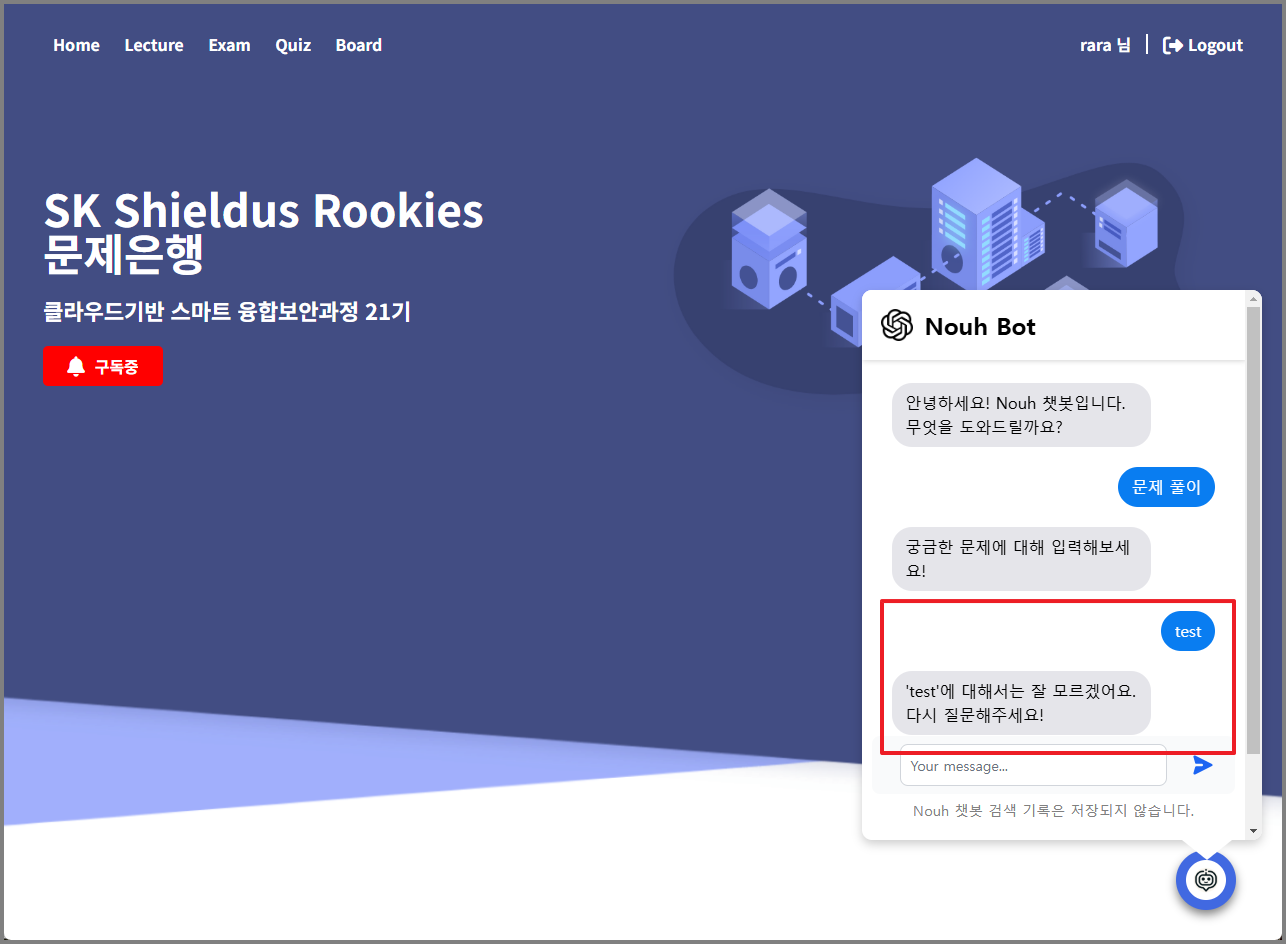
* + 1. 서버 사이드 템플릿 인젝션(SSTI)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 취약점 명 | | SSTI(Sever-Side Template Injection) | 위험도 | 상 |
| 취약점 설명 | | 공격자가 서버 측의 템플릿을 조작할 수 있는 공격 기법 | | |
| 영향 효과 | | 템플릿 엔진에 임의 코드 주입을 통한 원격 코드 실행 등의 위협이 발생할 수 있음 | | |
| 권고 사항 | | 1. 안전한 함수 사용 권고(render\_templete(string) 사용 제한)  **\* 동일한 로직 전수 조치 \***  **상세조치방안: 5. 대응방안 > 5.11 SSTI(Sever-Side Template Injection)** **참고** | | |
| No | 발견 URL 및 경로 | | | |
| 1 | https://no-uh.com/chatbot | | | |

[표 19] SSTI - 취약점 요약

**[취약점 검증 절차]**

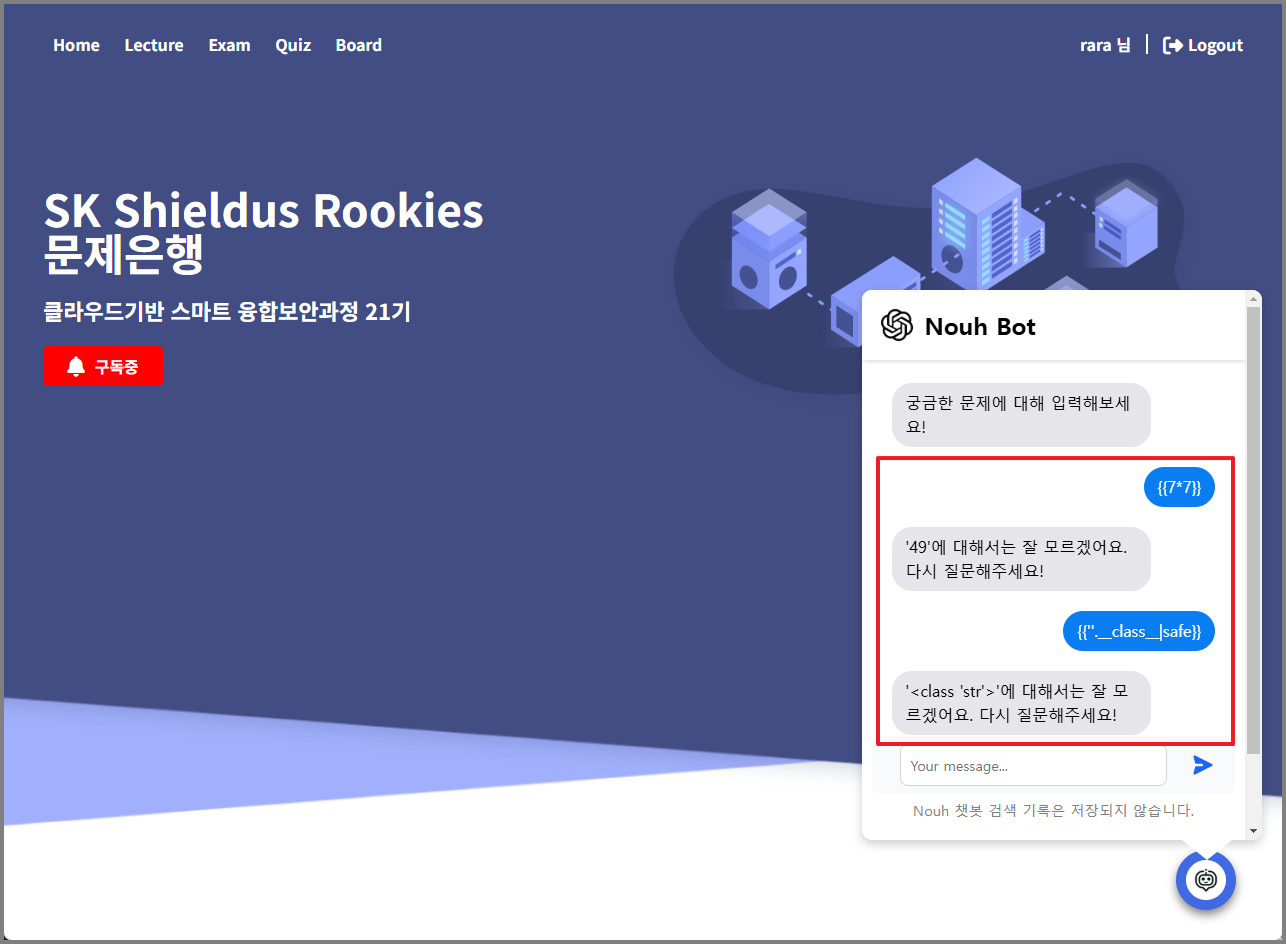
Step 1. 메인 페이지에서 Nouh Bot 확인



[그림 30] SQL 인젝션 삽입 지점 확인

Step 2. NouhBot에게 악의적 구문 삽입을 통해 데이터 타입 알아내기

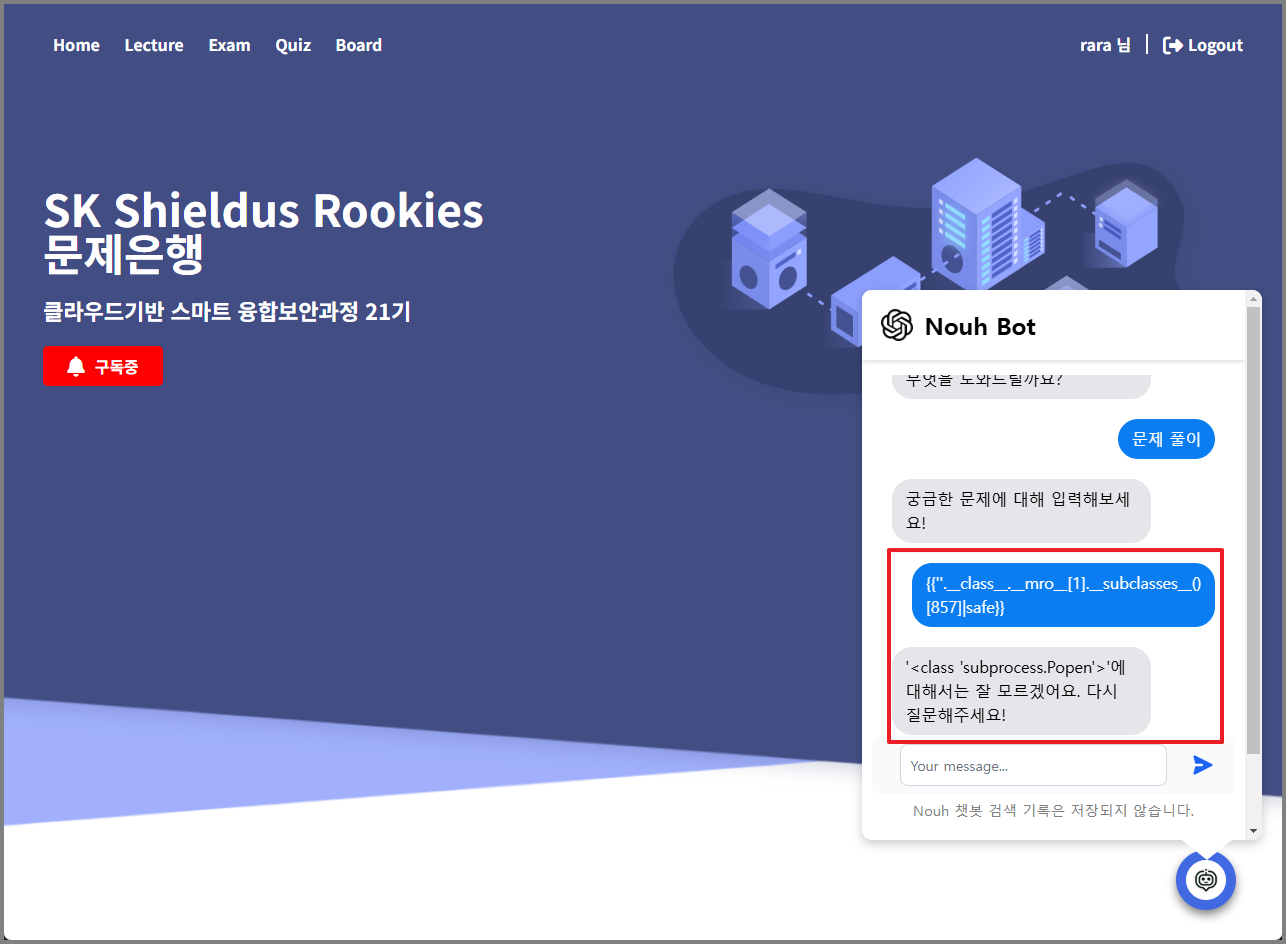
악의적 구문 : { {{7\*7}} , {{“.\_\_class\_\_|safe}} }



[그림 31] SSTI - 악의적 구문을 통한 데이터 타입 추출

Step 3. NouhBot에게 악의적 구문 삽입을 통해 클래스 출력

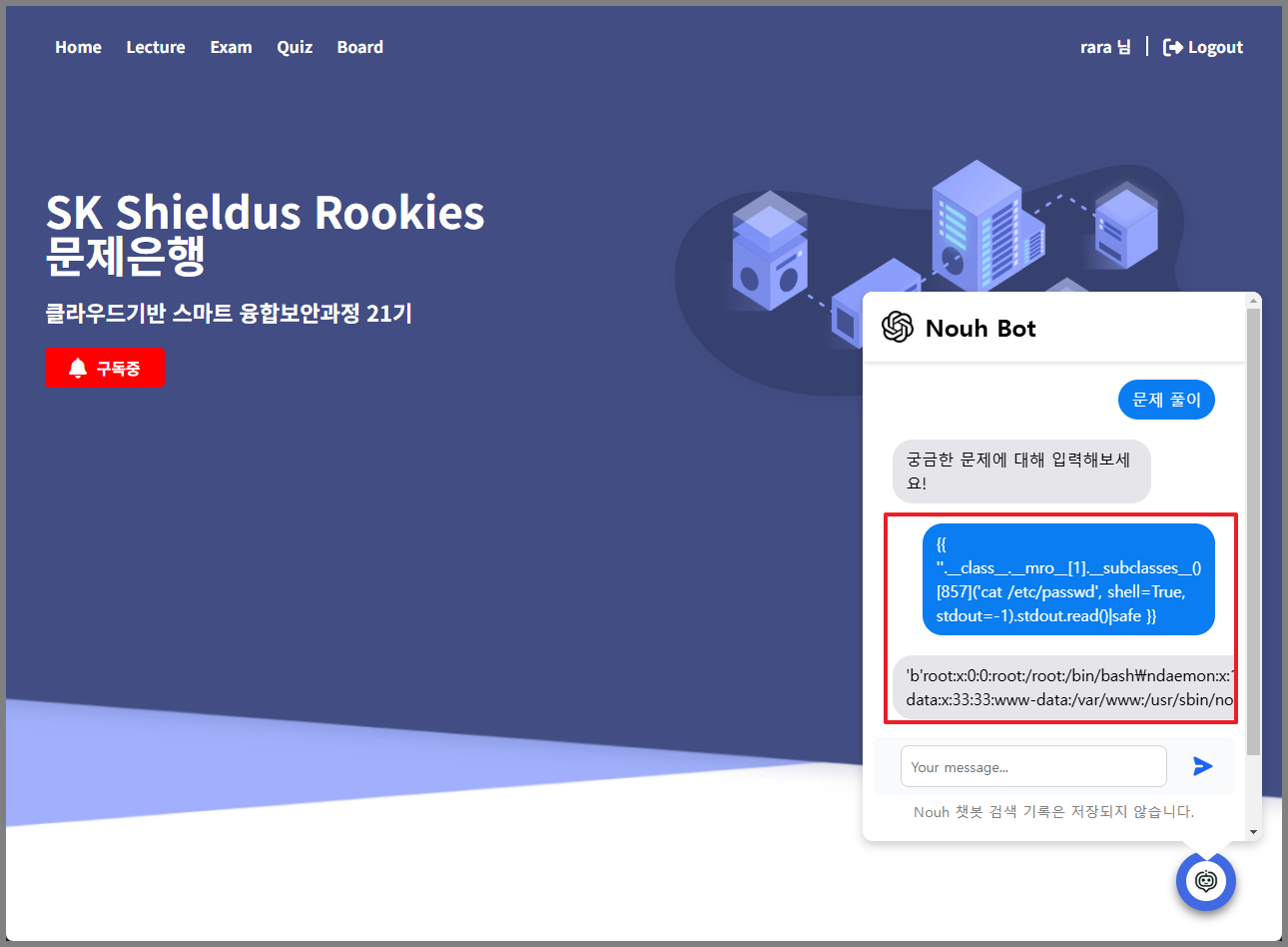
악의적 구문 : { {{''.\_\_class\_\_.\_\_mro\_\_[1].\_\_subclasses\_\_()[857]|safe}} }



[그림 32] SSTI - 악의적 구문을 통한 클래스 추출

Step 4. NouhBot에게 악의적 구문 삽입을 통해 원격 코드(RCE) cat /etc/passwd 를 실행

악의적 구문 : { {{ ''.\_\_class\_\_.\_\_mro\_\_[1].\_\_subclasses\_\_()[857]('cat /etc/passwd', shell=True, stdout=-1).stdout.read()|safe }} }



[그림 33] SSTI - cat /etc/passwd 실행

1. 보안 대책
   1. SQL 인젝션

|  |  |  |  |
| --- | --- | --- | --- |
| **취약점** | 입력값 검증 미흡 | **중요도** | 상 |
| 사용자의 입력값에 의해 SQL 질의문을 만들어 사용하는 곳에서 공격자가 이를 악용해 비정상 질의를 하여 특정 명령을 실행하거나 데이터베이스의 임의 데이터를 조회할 수 있는 취약점  **\* 점검 기준**  - URL 파라미터 또는 XML 등 입력하는 부분에 SQL 구문 입력 후 서버에서 응답한 값에 대한 위험성 점검  - SQL문으로 해석될 수 있는 값(글 번호, 검색 내용 등)을 입력하여 데이터베이스내에 저장된 정보 열람 및 시스템 명령 실행가능 여부 점검  - 조작된 XPath 쿼리를 보내어 비정상적인 질의 가능 여부 점검 등 | | | |
| **보안대책** | | | |
| **1. 웹 서버 보안**   * + - * 1. 웹 서버의 오류 정보가 사용자에게 노출되지 않도록 조치   - Internal Error(500) 오류 노출 금지   * + - * 1. 웹 애플리케이션과 연동되는 데이터베이스의 접근 권한을 최소화         2. 사용자 입력 폼(로그인 폼, 검색 폼, URL 등)을 대상으로 특수문자, 특수 구분 필터링 규칙 적용  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | 필터링 대상 | | | | | | | ‘ | “ | - | # | ( | ) | | = | \*/ | /\* | + | < | > | | user\_tables | user\_table\_columns | | table\_name | column\_name | syscolumns | | union | select | insert | drop | update | and | | or | if | join | substring | from | where | | declare | substr | openrowset | xp\_ | ‘sysobjects | % |   **2. 어플리케이션 보안**   * + - * 1. Parameterized Queries 사용   - name 값을 %s 바인딩 변수 사용  - name = request.POST.get('name', '')에서 name 값을 받아오는데, 기본값을 빈 문자열로 설정   * + - * 1. DB API 사용 예제   - 정적인 쿼리문을 사전에 생성하고 사용자 입력을 바인딩하는 방법을 적용  - “?”를 Placeholder로 사용하거나 “name”처럼 Named Placeholder를 사용   |  | | --- | | 안전하지 않은 소스코드 (DB API) | | with dbconn.cursor() as curs:  # 외부로부터 입력 받은 값을 검증 없이 사용할 경우 안전하지 않다  name = request.POST.get('name', '')  content\_id = request.POST.get('content\_id', '')  # 사용자의 검증되지 않은 입력값을 사용해 동적 쿼리문 생성  sql\_query = "update board set name='" + name + "' where content\_id='“ + content\_id + "'"  # 외부 입력값이 검증 없이 쿼리로 포함되어 안전하지 않다  curs.execute(sql\_query) | | 안전한 소스코드 (DB API) | | with dbconn.cursor() as curs:  name = request.POST.get('name', '')  content\_id = request.POST.get('content\_id', '')  # 외부 입력값 조작으로부터 안전한 인자화된 쿼리를 생성한다.  sql\_query = 'update board set name=%s where content\_id=%s'  # 사용자의 입력값이 인자화된 쿼리에 바인딩 후 실행되므로 안전하다.  curs.execute(sql\_query, (name, content\_id)) |   3) ORM 사용 예제  - list 형은 %s, 딕셔너리 형은 %(key)를 사용  - querysets인 쿼리 인자화를 사용해 외부 입력값을 바인딩 변수로 사용   |  | | --- | | 안전하지 않은 소스코드 ORM | | name = request.POST.get('name', ')  # 입력값을 검증 없이 쿼리문 생성에 사용해 안전하지 않다 query=“select \* from member where name=‘” + name + “’”  # 외부 입력값을 검증 없이 사용한 쿼리문을 raw()함수로 실행하면 안전하지 않다 data = Member.objects.raw(query) | | 안전한 소스코드 ORM | | name = request.POST.get('name', ')  # 외부 입력값을 raw() 함수 실행 시 바인딩 변수로 사용하여 쿼리 구조가  # 변경되지 않도록 한다.(list 형은 %s, dictionary 형은 %(key)s를 사용) query='select \* from member where name=%s'  # 인자화된 쿼리문을 사용하여 raw() 함수를 호출해 안전하다 data = Member.objects.raw(query, [name]) | | | | |

* 1. 악성 콘텐츠

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 악성 exe 파일 업로드 | **중요도** | 상 |
| 웹 사이트 게시판, 댓글, 자료실 등에 정상적인 콘텐츠 대신 악성 콘텐츠를 주입하여 실행될 경우 사용자가 해당 콘텐츠 열람 시 악성코드 감염 및 웹 페이지 변조 등 보안상 심각한 위험에 노출될 수 있음  **\* 점검 기준**  - 악의적 콘텐츠가 입력되며, 실행되는 경우 | | | |
| **보안대책** | | | |
| 1. **보안 정책**   악성 콘텐츠가 삽입 되어있는 페이지에 대하여 증거자료(화면, 소스 등)를 남기고, 삽입된 악성 콘텐츠를 삭제하거나 페이지의 삭제 등을 실시함  취득한 증거자료를 가지고 악성 콘텐츠의 삽입 원인에 대해 분석하여 원인을 제거할 것을 권고   1. **확장자 필터링**   게시판의 글 등록 및 파일 업로드 기능에 Flash 파일이나 avi 동영상 파일, exe 실행 파일 등 악성코드가 포함될 수 있는 콘텐츠를 삽입 또는 업로드 하지 못하게 필터링 적용   1. **주기적 바이러스 검사**   주기적으로 업로드 된 파일을 대상으로 바이러스 검사 실시 | | | |

* 1. 크로스사이트 스크립팅

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 입력 값 검증 미흡 | **중요도** | 상 |
| 공격자가 업로드한 스크립트의 타 이용자 사용환경(브라우저, 웹 뷰 등)에서의 실행 가능 여부를 점검  **\* 점검 기준**  - 크로스사이트 스크립팅 문자열을 데이터베이스에 저장된 값 확인 가능한 페이지에 업로드 시, 해당 스크립트의 실행 여부 점검 등 | | | |
| **보안대책** | | | |
| **1. HTML 특수문자 -> HTML Entity 치환**  사용자의 입력 값에 포함된 HTML 특수문자를 HTML Entity로 치환하여 검증   |  |  |  |  | | --- | --- | --- | --- | | HTML특수문자 | HTML Entity | HTML 특수문자 | HTML Entity | | < | &lt; | > | &gt; | | ' | &#x27; | " | &quot; | | ( | &#40; | ) | &#41; | | / | &#x2F; |  | |   **\*** 저장할 때 치환  저장할 때 치환하여 DB안에 저장하는 것이 가장 안전함. 하지만 기존에 운영 중인 DB에서 치환하여 저장하고 있지 않을 경우에 치환하여 저장하기 시작하면 기존에 존재하던 데이터들과 신규로 저장되는 데이터들이 사용하는 HTML 특수문자가 달라 검색 기능 등 운영에 문제가 발생할 수 있음  **\*** 출력할 때 치환  치환하여 출력하는 경우 기존 운영 중인 DB에 적용해도 문제가 발생하지 않음. 저장은 기존 방식대로 치환하지 않고 저장하는 대신 클라이언트로 데이터를 전달 할 때 치환하는 방식임. 하지만 이러한 경우 문제점은 데이터가 출력되는 모든 곳에서 치환해주는 작업을 해야 하기 때문에 취약한 페이지가 존재할 가능성이 존재함  **\*** Python Jinja2 모듈 사용할 때  HTML, XML 등의 마크업 언어를 처리할 때, 템플릿 내부에서 특수 문자가 자동으로 이스케이프(escape)되도록 설정 함. Autoescape 기능을 활성화 해 사용자에게 반환되는 렌더링 값을 출력할 때 치환해주는 역할을 함   |  | | --- | | 안전하지 않은 소스코드 (Python) | | env = Environment(loader=PackageLoader("foobar"),  autoescape=False)  template = env.get\_template("mytemplate.html") | | 안전한 소스코드 (Python) | | env = Environment(loader=PackageLoader("foobar"),  autoescape=True)  template = env.get\_template("mytemplate.html") |   **2. HTML Entity로 치환 및 특정 HTML 태그 허용**  위의 방식대로 HTML에서 사용되는 특수문자를 치환하게 될 경우 사용하고 싶은 HTML 특수문자를 사용할 수 없어 근본적으로 크로스 사이트 스크립팅 공격을 막을 수 있으나 게시판과 같이 특정 HTML 태그를 사용하게 하고 싶을 경우 사용자 입력 값을 HTML Entity로 전체 치환한 뒤, 특정 허용할 HTML 태그들만 White List 방식으로 복구 해야함  **\* 주의할** **점**  특정 HTML 태그만 허용하는 White List 방식으로 복구 시켜주게 될 경우 허용된 HTML 태그를 이용해 공격할 위험이 존재하므로 허용할 HTML 태그에 스크립트를 삽입할 위험이 있는 이벤트 핸들러 등을 제외하여야 함  **HTML Entity 치환 및 특정 HTML 태그 허용 예시 코드 (Python)**   |  | | --- | | String searchWord = request.getParameter("searchWord");  if (searchWord != null) {  searchWord = searchWord.replaceAll("<","&lt;");  searchWord = searchWord.replaceAll(">","&gt;");  searchWord = searchWord.replaceAll("(","&#40;");  searchWord = searchWord.replaceAll(")","&#41;");  } |   **3. XSS 보안 Library 사용**  OWASP ESAPI 라이브러리, lucy-xss-filter, AntiXSS 라이브러리와 같이 XSS 취약점 예방 라이브러리를 사용할 수 있음. 위와 같은 보안 라이브러리를 사용하여 XSS의 HTML 특수 문자를 일괄적으로 HTML Entity로 치환 가능함 | | | |

* 1. 약한 문자열 강도

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 문자열 보안 검증 미흡 | **중요도** | 상 |
| 취약한 계정 및 패스워드를 삭제하고, 사용자가 취약한 계정이나 패스워드를 등록하지 못하도록 패스워드 규정이 반영된 체크 로직을 회원가입, 정보변경, 패스워드 변경 등 적용 필요한 페이지에 모두 구현되어야 함.  **\* 점검 기준**  - 패스워드 길이, 특수문자 및 대,소문자, 숫자가 포함된 패스워드가 로그인이 되는지 확인 | | | |
| **보안대책** | | | |
| o 로그인 시 패스워드 입력 실패가 일정 횟수(3~5회) 를 초과할 경우 관리자에게 통보 및 계정 잠금. 인증 실패 횟수를 Client Side Script(Javascript, VBScript 등)를 사용하면 사용자가 임의로 수정할 수 있으므로 Server Side Script(PHP, ASP, JSP 등)를 통하여 구현  **Step 1)** 다음 각 항목의 문자 종류 중 2종류 이상을 조합하여 최소 10자리 이상 또는 3종류 이상을 조합하여 최소 8자리 이상의 길이로 구성  (1) 영문 대문자(26개)  (2) 영문 소문자(26개)  (3) 숫자(10개)  (4) 특수문자(32개)  **Step 2)** 연속적인 숫자나 생일, 전화번호 등 추측하기 쉬운 개인정보 및 아이디와 비슷한 비밀번호는 사용하지 않는 것을 권고  **Step 3)** 비밀번호에 유효기간을 설정하여 반기별 1회 이상 변경  **Step 4)** 최근 사용되었던 패스워드 재사용 금지 | | | |

* 1. 크로스사이트 리퀘스트 변조(CSRF)

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 기능 요청 검증 미흡 | **중요도** | 상 |
| 공격자가 취약한 로직으로 구현된 정상적인 기능을 악용해 사용자의 요청을 위조해 서버를 속이는 취약점  **\* 점검 기준**  - 페이지 내 기능을 통해 서버에 위조된 요청을 보내고 서버의 응답이 이뤄지는지 확인 | | | |
| **보안대책** | | | |
| **1. CSRF 보안 토큰 사용**  보안 토큰은 ‘발급’ 페이지에서 공격자가 유추할 수 없는 랜덤값을 매번 생성하고, 요청 시 마다 변경되며, 토큰 ‘검증’ 페이지에서 유효성 검사를 진행함. 토큰은 한 번 사용 후 만료 처리를 하여 재사용을 방지  **CSRF 보안 토큰을 사용한 보안 로직**   |  | | --- | | from flask import Flask  from flask\_wtf.csrf import CSRFProtect  # CSRF 설정 추가  csrf = CSRFProtect(app)  app = Flask(\_\_name\_\_)  app.config[‘SECRET\_KEY’] = os.environ.get('SECRET\_KEY')  csrf.init\_app(app)  … |   **2. 사용자 요청 검증**  서버로 전송되는 요청이 정상적인 사용자에 의한 요청인지 확인하는 검증 로직을 추가하는 방법. 검증 로직은 Submit Button, Alert 창 등을 이용해, 피해자 몰래 실행되는 것이 아닌 알림창을 확인하고 실행하도록 로직을 변경 | | | |

* 1. 불충분한 세션 만료

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 세션 만료 검증 미흡 | **위험도** | 상 |
| o 웹사이트의 사용자들의 세션이 오랜 시간 동안 남아 세션을 재사용하여 비 인가자가 인증없이 로그인이 가능한 취약점  **\* 점검 기준**  - 로그인 이후 1시간 이상 지났을 때, 만료되지 않고 같은 세션인지 확인 | | | |
| **보안대책** | | | |
| **1. 세션 타임아웃 설정**  o 세션 타임아웃 구현 시 타임아웃 시간은 10분으로 설정할 것을 권고  **세션 타임아웃 설정 (Python)**   |  | | --- | | …  app = Flask(\_\_name\_\_)  app.secret\_key = 'secretkey' # secret\_key는 서버상에 동작하는 어플리케이션 구분하기 위해 사용하고 복잡하게 만들어야 함  app.config["PERMANENT\_SESSION\_LIFETIME"] = timedelta(minutes=1) # 로그인 지속시간 지정 현재 1분  … | | | | |

* 1. 자동화 공격

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 반복적인 요청 | **중요도** | 상 |
| o 웹 애플리케이션의 특정 프로세스(로그인 시도, 게시글 등록, SMS 발송 등)에 대한 반복적인 요청을 자동으로 보내는 취약점  **\* 점검 기준**  - 웹 어플리케이션의 특정 프로세스에 대한 반복적인 요청 시 통제가 미흡한 경우 | | | |
| **보안대책** | | | |
| o 로그인 시도, 게시글 등록, SMS 발송 등에 대한 정상적인 요청 정보를 식별하여 반복적으로 요청 시 통제가 이루어지는지 확인  **1. 일회성 인증 구현**  o 로그인 시도, 게시글 등록, SMS 발송 등에 대한 사용자 요청이 일회성이 될 수 있도록, 캡차(이미지를 이용하여 확인 값을 표시하고 사용자가 값을 등록하여 인증함) 등 일회성 확인 로직을 구현  **※ 캡차(CAPTCHA):** 자동화된 컴퓨터와 사람을 판별하기 위한 기술의 일종    **2. 요청 임계값 설정**  o 게시판에 자동으로 생성하는 요청을 막기위해 임계값을 설정하여 일정 시간 내에 동일한 IP나 사용자 계정으로 너무 많은 요청을 보내는 경우 이를 차단하거나 제한하는 메커니즘을 설정한다.  **요청 임계값 설정 (Python)**   |  | | --- | | # 설정값: 요청 제한을 위한 임계값  MAX\_REQUESTS = 5 # 한 번에 최대 5번 요청 허용  if int(request\_count) >= MAX\_REQUESTS:  return | | | | |

* 1. 파일 업로드

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 파일 확장자 검증 미흡 | **위험도** | 상 |
| 파일 업로드 기능이 존재하는 웹 어플리케이션에서 확장자 필터링이 제대로 이루어 지지 않았을 경우 공격자가 악성 스크립트 파일(웹쉘)을 업로드 하여 웹을 통해 해당 시스템을 제어할 수 있어 명령어 실행 및 디렉터리 열람이 가능하고 웹 페이지 또한 변조가 가능한 취약점  **\* 점검 기준**  - 이미지, 한글(hwp) 등의 파일을 업로드 할 수 있는 부분에 스크립트 파일 업로드 가능 여부 점검  - 업로드 된 파일의 위치 및 실행 가능 여부 점검 등  \* 웹쉘(web shell): 업로드 취약점을 통하여 해커가 원격에서 웹 서버를 조종할 수 있도록 작성한 웹 스크립트 (단체표준 TTAK.KO-12.0002/R3 정보 보호 기술 용어) | | | |
| 보안대책 | | | |
| **1. 시큐어 코딩 (Python)**   |  | | --- | | 악성파일 업로드 시큐어 코딩 보안 대책 | | 파일의 크기와 개수 제한 , 종류 제한(확장자 white list) | | 업로드한 파일을 외부에서 접근할 수 없는 경로에 저장 | | 업로드한 파일의 저장 경로와 파일명을 외부에서 알 수 없도록 함(변경하여 저장)  - 경로는 Web document root 밖에 위치, 웹을 통한 직접 접근 차단 | | 업로드한 파일에 실행 권한을 제거 | | 안전하지 않은 코드 예 (Python) | | from django.shortcuts import render  from django.core.files.storage import FileSystemStorage  def file\_upload(request):  if request.FILES['upload\_file']:  # 사용자가 업로드하는 파일을 검증 없이 저장하고 있어  # 안전하지 않다  upload\_file = request.FILES['upload\_file'] fs = FileSystemStorage(location='media/screenshot', base\_url='media/screenshot')  # 업로드 하는 파일에 대한 크기, 개수, 확장자 등을 검증하지 않음  filename = fs.save(upload\_file.name, upload\_file) return render(request, '/success.html', {'filename':filename})  return render(request, '/error.html', {'error':'파일 업로드 실패'}) | | 안전한 코드 예 (Python) | | import os  from django.shortcuts import render  from django.core.files.storage import FileSystemStorage  # 업로드 하는 파일 개수, 크기, 확장자 제한  FILE\_COUNT\_LIMIT = 5  # 업로드 하는 파일의 최대 사이즈 제한 예 ) 5MB - 5\*1024\*1024  FILE\_SIZE\_LIMIT = 5242880  # 허용하는 확장자는 화이트리스트로 관리한다.  WHITE\_LIST\_EXT = [  '.jpg',  '.jpeg'  ]  def file\_upload(request):  # 파일 개수 제한  if len(request.FILES) == 0 or len(request.FILES) > FILE\_COUNT\_LIMIT:  return render(request, '/error.html', {'error': '파일 개수 초과'})  for filename, upload\_file in request.FILES.items():  # 파일 타입 체크  if upload\_file.content\_type != 'image/jpeg':  return render(request, '/error.html', {'error': '파일 타입 오류'})  # 파일 크기 제한  if upload\_file.size > FILE\_SIZE\_LIMIT:  return render(request, '/error.html', {'error': '파일사이즈 오류'})  # 파일 확장자 검사  file\_name, file\_ext = os.path.splitext(upload\_file.name)  if file\_ext.lower() not in WHITE\_LIST\_EXT:  return render(request, '/error.html', {'error': '파일 타입 오류'}) |   **2. 파일이 업로드 되는 디렉토리(위치 및 파일명)가 사용자에게 노출되지 않도록 조치**  - 파일을 저장할 경우 파일명을 변경하여 저장하고(파일명과 확장자를 추측할 수 없도록 치환) 실제 파일명은 데이터베이스에 보관하는 등 정보를 이원화하여 운영  **3. 웹 방화벽 설정**  파일 업로드 필드를 대상으로 특수문자 필터링 규칙 적용 (웹 방화벽에서 설정 가능)   |  | | --- | | 필터링 문자 | | ../ | | ./ | | ..\ | | .\ | | % | | ; |   - 업로드 되는 파일 명에는 “../”, “;”, “%0” 등과 같은 특수문자를 사용할 수 없도록 Server Side Language에서 제한해야 하며 업로드 파일 크기를 제한(파일 사이즈가 너무 크거나 작은 파일에 대한 제한조치) 설정 할 것을 권고  **4. 오픈소스, 게시판 사용 시**  - 최신 보안 패치 적용 (CWE, CVE 등 확인 필요) | | | |

* 1. 파일 다운로드

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 경로 이동 | **위험도** | 상 |
| 다운로드 파일이 저장된 허용 디렉터리 외에 외부 입력 값을 변조하여 다른 디렉토리에 접근이 가능한 경우, 임의조작을 통해 허용되지 않은 파일을 열람하거나 다운로드하여 시스템 중요정보 및 파일 등에 접근할 수 있는 취약점  **\* 점검 기준**  - 파일 다운로드 시 파일이 저장된 디렉터리 이외의 경로에 접근하여 주요 파일의 다운로드 가능 여부 점검 등  주요 파일: 소스코드, 서버설정파일 등. | | | |
| **보안대책** | | | |
| **1. 파일 다운로드 취약점 웹 방화벽(WAF) 조치**  필터링 적용 (웹어플리케이션, 방화벽)  - 요청 파일명에 경로 조작 문자 (../\ )는 제거하여 사용할 수 있도록 필터링  - 웹 방화벽, 어플리케이션에서 위험문자열을 제거하는 필터링 적용   |  |  | | --- | --- | | 문자 | 설명 | | . | Path Traversal 가능 | | / | 특정 Path의 접근 가능성 확인 | | \ | 운영 환경에 따른 Path 접근 확인 | | % | UTF 인코딩 파라미터 |   서버 기반 접근 통제 수행  - 허가된 사용자 : 접근통제  - 안전한 파일 검증 : 다운로드 파일 이름을 데이터베이스에 저장, 다운로드 수행 시 요청 파일 이름과 비교하여 적절한지 확인하여 사용자가 조작할 수 있는 변수 제거함  - 파일 경로와 이름을 생성할 때 외부 값을 사용하는 경우, 정해진 경로 이외의 디렉토리와 파일에 접근할 수 없도록 처리(특정 디렉터리로 한정)  - 파일 다운로드 시 우선적으로 웹 루트 상위 경로로 접근하지 않도록 권한 설정  - 다운로드를 제공하는 페이지의 유효 세션 체크 로직 필수 적용  **2. 웹 애플리케이션 소스코드 시큐어 코딩 적용**   |  | | --- | | 안전한 소스코드 (Python) | | # 1. 파일 경로 설정 (특정 디렉토리 내에서만 파일을 다운로드하도록 제한)  download\_dir = settings.FILE\_DOWNLOAD\_DIR  # 설정 파일에서 다운로드 디렉토리 경로 읽기  safe\_filename = os.path.basename(filename)  # 경로 조작을 막기 위해 파일 이름만 가져오기  # 2. 파일 이름 검증 (위험한 파일명 차단: '..' 등의 경로 조작 문자를 방지)  if '..' in safe\_filename or safe\_filename.startswith('/'):  # 3. 다운로드할 파일 경로 설정  file\_path = os.path.join(download\_dir, safe\_filename)  # 4. 파일이 실제로 존재하는지 확인  if not os.path.isfile(file\_path):  # 5. 파일 확장자 검증 (예: 이미지 파일만 허용)  allowed\_extensions = ['.jpg', '.jpeg', '.png', '.txt']  if not any(safe\_filename.endswith(ext) for ext in allowed\_extensions):  # 6. 파일 다운로드 처리 (HTTP 응답으로 파일 보내기)  with open(file\_path, 'rb') as file:  response = HttpResponse(file.read(), content\_type='application/octet-stream')  response['Content-Disposition'] = f'attachment; filename="{safe\_filename}"' | | 파일 다운로드 시 우선적으로 웹 루트 상위 경로로 접근되지 않도록 권한 설정  파일 경로 및 파일 이름을 처리하는 파라미터 변수에 대해 null 여부를 체크 및 상태 경로를 설정할 수 있는 문자(/, \\, &, , 등)가 존재할 경우 파일 다운로드가 불가능하도록 구현하길 권장 | | | | |

* 1. 위치 공개

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 스캐너를 통한 경로 공개 | **중요도** | 상 |
| 페이지 내 존재하는 디렉터리 경로가 공개되어 있는 취약점으로 다른 취약점으로 연계하기 위한 정보수집으로 이어질 수 있음.  **\* 점검 기준**  - 스캐너 및 수동 점검 시 공개된 경로의 유무를 확인 | | | |
| **보안대책** | | | |
| **1. HTML 특수문자 -> HTML Entity 치환**  온라인 강의 플랫폼은 Nginx 서버를 사용하고 있으며 Nginx에서는 autoindex 설정을 “off”로 설정하여 디렉터리 리스팅을 막을 수 있습니다. Nginx.conf 또는 해당 사이트의 설정 파일에서 설정  **Autoindex 설정 (Nginx)**   |  | | --- | | server {  location / {  autoindex off;  }  } | | | | |

* 1. 서버 사이드 템플릿 인젝션(SSTI)

|  |  |  |  |
| --- | --- | --- | --- |
| 취약점 | 입력 값 검증 미흡 | **중요도** | 상 |
| 공격자의 입력 값이 검증 과정을 거치지 않고 서버 측 템플릿에 포함되어 실행되는 취약점  **\* 점검 기준**  - 템플릿 구문 테스트 문자열을 사용자 입력 값에 넣어, 서버 측에서 해당 구문의 실행 여부 점검 | | | |
| **보안대책** | | | |
| **1. render\_template() 함수 사용**  Jinja Template의 경우 render\_template() 함수를 사용하면 사용자 입력 값을 명령어가 아닌 문자열로 처리하기 때문에 SSTI에 대한 근본적인 대응 방안임.  **render\_template() 함수 사용 예시 코드(Python)**   |  | | --- | | @bp.route(“/”)  def index():  return render\_template(“main/index.html”) |   **2. 템플릿에 사용되는 문자열 필터링**  위의 방식대로 render\_template() 함수를 사용할 경우 사용자의 입력 값이 명령어가 아닌 문자열로 해석되기 때문에 근본적으로 SSTI 공격을 막을 수 있음. 그러나 부득이하게 render\_template\_string() 함수를 사용하게 될 경우 템플릿 구문에 사용되는 특수문자를 사용할 수 없도록 필터링 해야함  **템플릿 문자열 필터링 예시 코드(Python)**   |  | | --- | | @bp.route(“/”)  def index():  # 사용자가 입력한 문자열에서 위험한 특수 문자 제거  sanitized\_kw = re.sub(r'[-=+.#/?:^$.@\*"\“\”※~&%!()[]<>`{}|\\]', '', kw)  # 필요시 추가적인 필터링을 통해 다른 위험 문자를 제거할 수 있음  return sanitized\_kw | | | | |