# 웹 해킹 및 정보보안 실습

Chapter 3. WebGoat 모의 해킹 실습

# 학습 핵심 키워드 (Keyword)

본 챕터에서 다루게 될 주요 해킹 및 보안 개념들입니다.

# Keyword

## #WebGoat

자바 기반으로 개발된
취약한 웹 사이트 실습 환경

## #General

HTTP 개념, 프록시, 개발자 도구, 보안 3요소

## #Broken Access Control

인가 결함 (Session, IDOR, Function Access)

## #Cryptographic Failures

인코딩, 해싱, 대칭키/비대칭키 암호화 결함

# Chapter 3. WebGoat 모의 해킹 실습

OWASP에서 선정한 다양한 웹 취약점을 사이트에 그대로 옮겨 놓은
WebGoat 환경에서 본격적인 모의 해킹 실습을 진행해봅니다.

# Section 01. General (기본 개념)

웹 공격 기법에 앞서 공격자가 반드시 숙지해야 하는
HTTP 통신, 개발자 도구, 보안 3요소를 살펴봅니다.

# HTTP란 무엇인가?

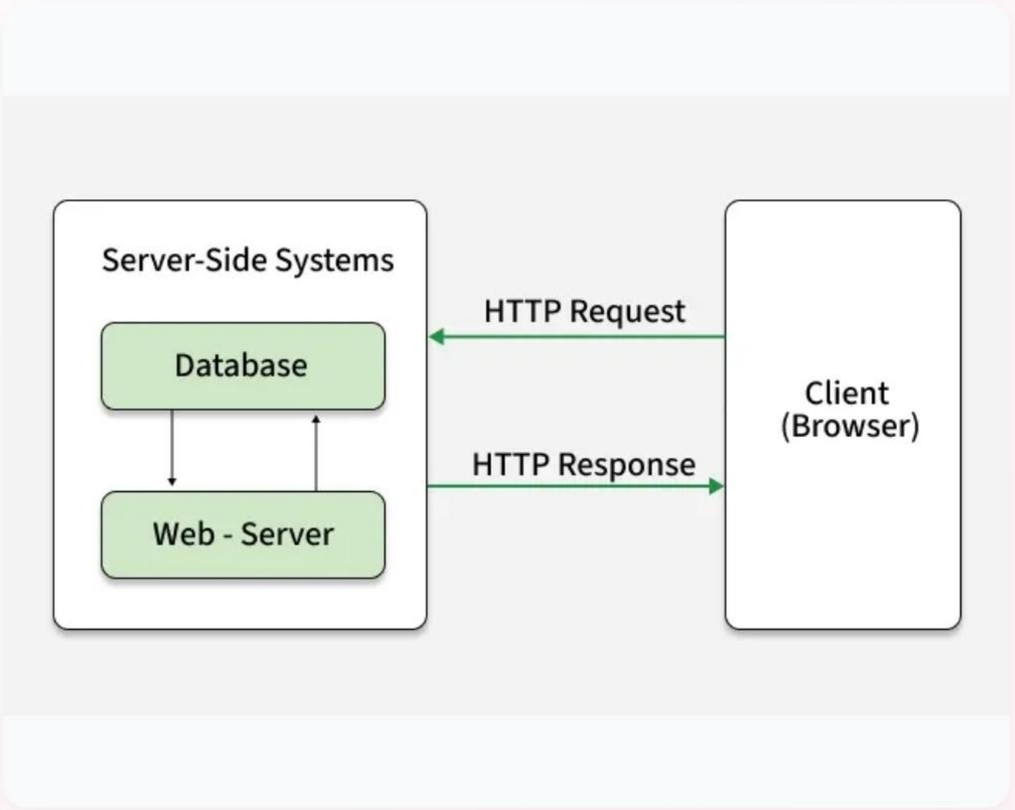
**HTTP (HyperText Transfer Protocol)**는 인터넷에서 데이터를 주고받기 위한 서버/클라이언트 모델 기반의 텍스트 프로토콜입니다.

- **Connectionless (비연결성):** 클라이언트가 요청을 보내고 서버가 응답을 마치면 연결을 즉시 끊습니다. (자원 낭비 최소화)
- **Stateless (무상태성):** 통신이 끝나면 상태를 유지하지 않습니다. 따라서 서버는 과거의 요청이나 사용자가 누구인지 기억하지 못합니다.

# HTTP 통신 구조

HTTP는 클라이언트가 서버로 무언가를 요구하는 **요청(Request)**과 서버가 그에 대해 대답하는 **응답(Response)**의 쌍으로 이루어집니다.

- **Request:** "내 정보 페이지를 보여줘!" (GET /profile)
- **Response:** "여기 네 정보가 담긴 HTML 데이터야!" (200 OK)



# HTTP Request (요청) 구조

클라이언트가 서버로 보내는 HTTP 요청은 3가지 파트로 나뉩니다.

- **Start Line:** HTTP 메서드, 요청 대상(URI), HTTP 버전이 위치합니다. (예: `GET /user/join HTTP/1.1`)
- **Headers:** 요청에 대한 부가적인 속성 메타데이터가 들어갑니다. (Host, User-Agent 등)
- **Body:** 서버로 실제로 전송해야 할 데이터가 들어갑니다. (회원가입 폼 데이터 등)

# HTTP Request (요청) 메시지 예시

앞서 배운 Start Line, Headers, Body가 실제 패킷에서는 어떻게 생겼는지 확인해 봅니다. (헤더와 바디 사이에는 반드시 **빈 줄(Blank Line)**이 존재합니다.)

```
POST /WebGoat/login HTTP/1.1 Host: 127.0.0.1:8080 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) Chrome/120.0.0.0 Content-Type: application/x-www-form-urlencoded Cookie: JSESSIONID=abc123xyz456 Connection: keep-alive username=tom&password=cat
```

# HTTP Request Methods (요청 메서드)

클라이언트가 서버에게 어떤 '행위'를 원하는지 Start Line 맨 앞에 명시합니다.

- **GET:** 서버로부터 데이터를 조회할 때 사용합니다. 파라미터가 URL에 노출되어 전달됩니다. (보안 취약점 발생 가능성)
- **POST:** 서버에 새로운 데이터를 생성/등록할 때 사용합니다. (회원가입, 글쓰기) 데이터가 Body에 숨겨져 전송됩니다.

# REST와 RESTful API의 개념

다양한 HTTP 메서드의 활용을 이해하기 위해서는 현대 웹 아키텍처의 표준인 **REST**를 알아야 합니다.

- **REST (Representational State Transfer):** 자원(Resource)의 이름으로 대상을 구분하여, 상태(정보)를 주고받는 소프트웨어 아키텍처 형식입니다.
- **RESTful:** 이러한 REST의 원칙과 제약 조건을 철저히 잘 지켜서 설계된 API 시스템을 가리켜 "RESTful하다"고 부릅니다.
- **핵심 원리:** URI(주소창)는 `/deleteUser?id=1`처럼 행위를 포함하지 않고 `/users/1`처럼 오직 '자원'만을 명시합니다. 행위는 오직 HTTP 메서드로만 구분합니다.

# RESTful API와 다양한 메서드

RESTful 구조에 따라 GET, POST 외에도 다음과 같은 세밀한 메서드를 사용하여 자원에 대한 행위(CRUD)를 명확히 정의합니다.

- **PUT:** 서버에 존재하는 리소스를 전체 수정(덮어쓰기)할 때 사용합니다.
- **PATCH:** 리소스의 일부만 수정할 때 사용합니다.
- **DELETE:** 리소스를 삭제할 때 사용합니다.
- **OPTIONS:** 서버가 해당 자원에 대해 어떤 메서드를 허용하는지 미리 물어볼 때 사용합니다.

# HTTP Request Headers (요청 헤더)

서버가 클라이언트의 상태를 파악할 수 있도록 돕는 속성 정보들입니다.

- **Host:** 요청이 전송되는 대상 서버의 도메인(필수 요소)
- **User-Agent:** 클라이언트의 브라우저 및 OS 정보 (크롬, 윈도우 등)
- **Accept:** 클라이언트가 처리할 수 있는 데이터 타입 (text/html, application/json 등)
- **Content-Type:** Body에 담긴 데이터의 형식 표시

# Stateless 극복: Cookie와 Session

HTTP는 무상태성(Stateless)이므로 내가 방금 로그인했다는 사실을 서버가 기억하지 못합니다. 이를 해결하기 위한 핵심 기술입니다.

- **Cookie (쿠키):** 클라이언트(브라우저) 측에 저장되는 키-값 형태의 데이터입니다. 매 요청 시 HTTP 헤더에 담겨 전송됩니다.
- **Session (세션):** 중요한 인증 정보는 서버 측 메모리나 DB에 저장하고,
  클라이언트에게는 그 정보를 찾을 수 있는 '세션 ID'만 발급하여 쿠키에 담아줍니다.

# HTTP Response (응답) 구조

서버가 클라이언트의 요청을 처리한 뒤 돌려주는 응답 메시지 구조입니다.

- **Status Line:** HTTP 버전, 상태 코드, 상태 메시지가 들어갑니다. (예: `HTTP/1.1 200 OK`)
- **Headers:** 응답에 대한 메타데이터 (Content-Type, 쿠키 세팅 명령 등)
- **Body:** 클라이언트가 요청한 실제 데이터 (HTML 소스코드, 이미지, JSON 덩어리 등)

# HTTP Response (응답) 메시지 예시

Status Line, Headers, 빈 줄(Blank Line), Body 데이터가 포함된 실제 서버의 응답 패킷 예시입니다. 클라이언트가 파싱할 수 있도록 다양한 지시가 헤더에 포함됩니다.

```
HTTP/1.1 200 OK Date: Sun, 15 Mar 2026 12:00:00 GMT Server: Apache/2.4.41 (Ubuntu) Set-Cookie: hijack_cookie=7581771400844129083-1680611361571; Path=/WebGoat; Secure Content-Type: application/json; charset=utf-8 Content-Length: 64 { "status": "success", "role": 3, "userId": "2342384" }
```

# HTTP Response Status Code (성공/이동)

서버가 클라이언트의 요청 결과를 3자리 숫자로 명확하게 알려주는 상태 코드입니다.

- **2xx (성공):** 요청이 성공적으로 처리되었음을 의미. 가장 대표적인 코드는 **200 OK** 입니다.
- **3xx (리다이렉션):** 요청을 완료하려면 클라이언트가 추가 조치를 취해야 함을 의미. 주로 다른 URL로 강제 이동시킬 때 **302 Found** 등을 씁니다.

# HTTP Response Status Code (에러)

문제가 발생했을 때 출력되는 상태 코드입니다.

- **4xx (클라이언트 에러):** 브라우저의 요청 자체가 잘못되었을 때.
  - **403 Forbidden:** 권한이 없어 접근 거부됨. (웹 해킹 실습에서 자주 봄)
  - **404 Not Found:** 요청한 URL을 서버에서 찾을 수 없음.
- **5xx (서버 에러):** 서버 내부 로직이나 DB 오류로 처리를 못했을 때. (예: **500 Internal Server Error**)

# HTTP Response Headers (응답 헤더)

서버가 돌려주는 주요 헤더 속성들입니다.

- **Content-Type:** Body에 담겨있는 데이터의 종류 (예: text/html, application/json) 클라이언트가 이 값을 보고 렌더링 방식을 결정합니다.
- **Set-Cookie:** 클라이언트(브라우저)에게 특정 쿠키를 생성하라고 지시할 때 사용합니다. (로그인 성공 시 세션 ID 발급 등)
- **Server:** 웹 서버의 소프트웨어 정보 (Nginx, Apache 등. 보안상 가리기도 합니다)

# 전체 통신 흐름 요약

클라이언트와 서버는 아래와 같이 핑퐁 형태로 구조화된 텍스트 메시지를 주고받습니다. 해커는 이 중간에서 메시지를 낚아채어 공격을 시도합니다.

```
[Request] POST /user/join HTTP/1.1 Host: test.com Cookie: JSESSIONID=abc123xyz id=admin&pw=password
```

```
[Response] HTTP/1.1 200 OK Content-Type: text/html Set-Cookie: new_token=777 Login Success
```

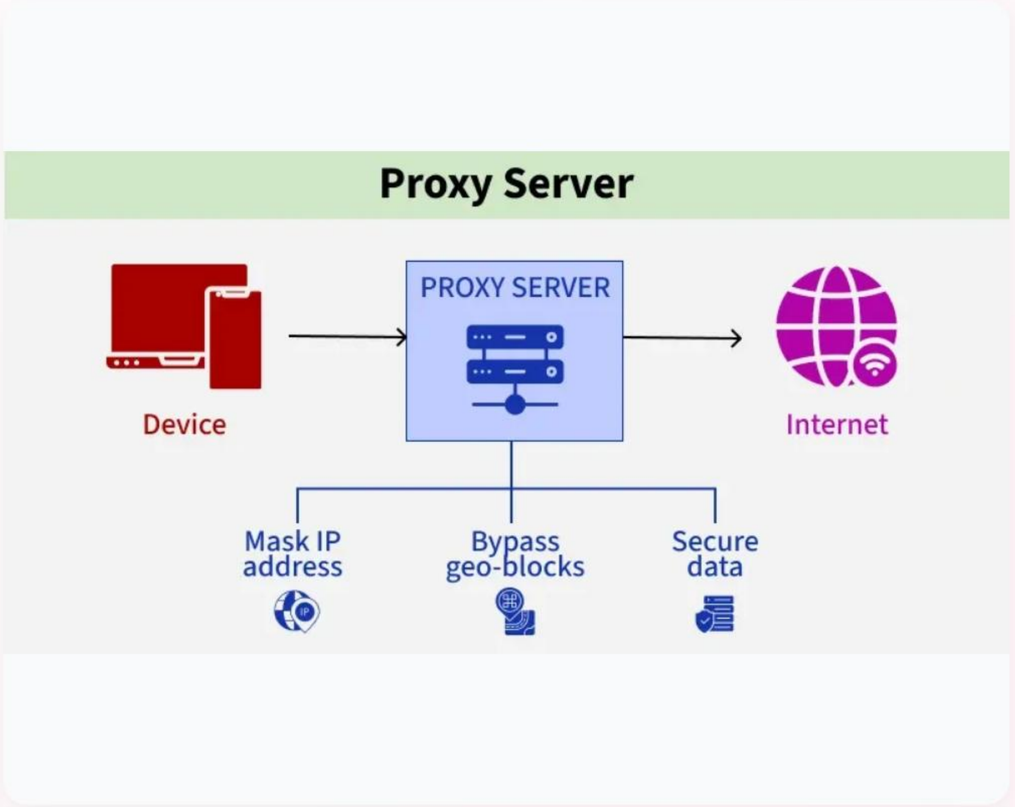
# Section 01. General (WebGoat 기초)

HTTP 기초를 다졌으니, 이제 이를 바탕으로 프록시와
개발자 도구를 활용하는 실습으로 들어갑니다.

# 01. HTTP Proxies (프록시)

프록시 서버란 클라이언트 PC와 서버 사이에 중계기 역할을 수행하여 데이터를 전달해 주는 서버입니다.

- 프록시 서버는 클라이언트로부터 받은 요청을 서버에 전달하고, 서버의 응답을 다시 클라이언트로 전송합니다.
- 중간에 위치하기 때문에 **요청과 응답의 확인 및 수정**이 가능합니다.



# 프록시 서버의 일반적인 사용 목적

실제 IT 환경에서 프록시는 다양하게 쓰입니다.

- **IP 우회:** 서버에서 내 IP를 차단했다면, 다른 프록시 서버를 경유하여 접속함으로써 차단을 우회할 수 있습니다.
- **보안 통제:** 회사 내부망에서 악성 파일 다운로드를 막기 위해 NGFW(차세대 방화벽) 장비가 프록시 역할을 수행하며 트래픽을 감시/차단합니다.

# 공격자에게 HTTP 프록시가 필요한 이유

방금 배운 HTTP 통신 구조 때문입니다.

공격자는 브라우저가 자동으로 만들어주는 정상적인 패킷을 **프록시 툴을 이용해 중간에서 낚아챈(Intercept) 뒤,** Headers, Body, Start Line 등을 악의적으로 변조하여 서버로 보냅니다.

이것이 웹 해킹의 시작이자 가장 강력 도구가 됩니다.

# Loopback 프록시 구성

공격을 위해 거창한 외부 서버가 필요한 것은 아닙니다.

내 PC(클라이언트)에 프록시 소프트웨어를 설치하고, 웹 브라우저의 트래픽이 **127.0.0.1 (localhost)** 자기 자신을 먼저 거쳐가도록 설정하면 Loopback 형태의 완벽한 프록시 환경이 완성됩니다.

# 대표적인 프록시 툴

공격자들이 즐겨 사용하는 HTTP 프록시 애플리케이션은 다음과 같습니다.

- **Burp Suite:** 현존하는 가장 범용적이고 강력한 상용/무료 프록시 툴 (본 과정 실습 툴)
- **OWASP ZAP:** OWASP 재단에서 만든 강력한 오픈소스 프록시 툴
- **Fiddler:** 패킷 캡처 및 디버깅에 유용한 툴
- **Paros:** 고전적인 자바 기반 프록시 툴

# Burp Suite 설치 및 실행 (1)

01

## 공식 홈페이지 접속

구글 크롬을 실행한 후 검색란에 'burp suite'를 입력하여 검색합니다.

검색 결과 최상단의 **Portswigger.net** 공식 홈페이지 링크를 클릭하여 접속합니다.

# Burp Suite 설치 및 실행 (2)

02

## 에디션 선택

홈 화면이 나타나면 상단 메뉴 바의 [Products]를 클릭합니다.

드롭다운 메뉴에서 무료 버전인 **[Burp Suite Community Edition]**을 선택합니다.

(Enterprise와 Professional은 기능이 더 많은 기업용 유료 버전입니다)

# Burp Suite 설치 및 실행 (3)

03

## 다운로드 페이지 진입

Community Edition 화면이 나타나면 좌측 하단의 **[Go straight to downloads]** 버튼을 클릭하여 다운로드 영역으로 이동합니다.

# Burp Suite 설치 및 실행 (4)

04

## OS 확인 및 파일 다운로드

버전 종류(Professional / Community)가 맞는지 확인하고,
드롭다운에서 본인의 PC 운영체제(Windows/Mac)가
일치하는지 봅니다.

이후 **[Download]** 버튼을 클릭해 설치 파일을 받습니다.

# Burp Suite 설치 및 실행 (5)

05

## 설치 진행 (Windows 기준)

다운로드된 Setup 실행 파일을 엽니다. 대화상자가 나타나면 [Next] 버튼을 클릭합니다.

설치 경로가 나오는 화면에서 나중에 프로그램을 쉽게 찾기 위해 **경로 텍스트를 복사(Ctrl+C)**해 둔 뒤 [Next]를 클릭합니다.

# Burp Suite 설치 및 실행 (6)

06

## 시작 메뉴 폴더 생성 해제

Select Start Menu Folder 페이지가 나타나면, 불필요한 폴더 생성을 막기 위해 **[Create a Start Menu folder]**의 체크를 해제한 후 [Next] 버튼을 눌러 설치를 진행합니다.

# Burp Suite 설치 및 실행 (7)

07

## 설치 완료

설치 프로그래스 바가 다 차고 완료 페이지(Completing the Setup Wizard)가 나타나면, 하단의 **[Finish]** 버튼을 클릭하여 창을 닫습니다.

# Burp Suite 설치 및 실행 (8)

08

## Burp Suite 실행

윈도우 파일 탐색기를 열고, 주소창에 아까 5단계에서 복사해 둔 경로를 붙여넣어 이동합니다.

폴더 안의 **BurpSuiteCommunity.exe** 파일을 더블클릭하여 프로그램을 실행합니다. (바탕화면 바로가기를 만들어두면 편합니다.)

# Mac OS에서 Burp Suite 설치 (1)

Mac OS 사용자의 경우 설치 과정이 더 단순합니다.

다운로드한 `.dmg` 파일을 더블클릭하여 마운트시킨 뒤, 나타나는 창에서 Burp Suite 앱 아이콘을 **Applications (응용 프로그램)** 폴더 쪽으로 드래그 앤 드롭합니다.

# Mac OS에서 Burp Suite 설치 (2)

설치가 완료되면 **SpotLight (Cmd+Space)** 또는 Launch pad를 실행하여 'Burp Suite'를 검색합니다.

검색된 앱을 클릭하여 실행하면 초기 준비가 끝납니다.

# Burp Suite 초기 구동 설정 (1)

Burp Suite를 처음 실행하면 프로젝트 유형을 묻는 창이 나옵니다.

무료 버전인 Community Edition은 디스크에 프로젝트를 저장하는 기능을 제공하지 않으므로, **[Temporary project in memory]** 라디오 버튼이 기본 선택된 상태 그대로 우측 하단의 **[Next]** 버튼을 클릭합니다.

# Burp Suite 초기 구동 설정 (2)

다음으로 설정 파일(Configuration)을 불러오는 화면이 나옵니다.

기본 설정을 사용할 것이므로 **[Use Burp defaults]**가 선택된 상태에서 하단의 **[Start Burp]** 버튼을 클릭합니다.

잠시 후 Burp Suite의 메인 대시보드 화면이 화려하게 열립니다.

# Burp Suite 메인 대시보드 구조

화면에 수많은 메뉴와 탭이 보입니다. 이 중 모의 해킹 실습의 90% 이상은 상단 탭 중 **[Proxy]** 메뉴에서 이루어집니다.

가장 먼저 **[Proxy]** 탭을 마우스로 클릭하여 이동해 주세요.

# Proxy 탭의 핵심 기능 요약

클라이언트의 요청을 가로채고 서버로 보내기 전 수정하는 핵심 공간입니다.

- **Intercept is on / off:** [on]일 때는 트래픽을 서버로 넘기기 전에 붙잡아(Hold) 대기시킵니다. [off]일 때는 붙잡지 않고 통과시킵니다.
- **Forward:** 붙잡아둔 요청/응답 내역을 확인하거나 수정한 뒤, 최종적으로 목적지에 전송합니다.
- **Drop:** 붙잡은 요청을 서버로 보내지 않고 통신을 즉시 차단(폐기)해 버립니다.

# Burp Suite 포트 충돌 문제

### 포트 번호 변경 필수

Burp Suite는 기본적으로 127.0.0.1의 **8080** 포트에서 프록시 대기를 합니다.
그런데 우리가 실습할 서버인 WebGoat 역시 8080 포트를 사용합니다. 포트가 겹치면 충돌이 나므로 Burp 쪽의 포트를 바꿔주어야 합니다.

# Burp Suite 포트 변경 방법

- 상단 최우측의 **[Settings]** (톱니바퀴) 버튼을 누릅니다.
- 검색창에서 Proxy를 치거나 **[Proxy listeners]** 항목을 찾습니다.
- 등록된 `127.0.0.1:8080`을 마우스로 선택 후 **[Edit]**를 누릅니다.
- Bind to port 값을 8080에서 **8888**로 수정한 뒤 [OK]를 눌러 저장합니다.

# Burp 전용 브라우저 실행

크롬 등 일반 브라우저에서 프록시 세팅을 수동으로 잡는 것은 매우 귀찮은 일입니다.

Burp Suite의 Proxy 탭에 있는 **[Open browser]** 버튼을 누르면, 모든 프록시 세팅(8888번 포트 연동)이 이미 끝난 구글 크로미움(Chromium) 기반의 독립된 브라우저 창이 하나 뜹니다.

이 브라우저 주소창에 `http://127.0.0.1:8080/WebGoat`를 치고 접속하면 즉시 패킷 가로채기가 작동합니다.

# [수동 방법] 일반 브라우저 프록시 설정 (Windows)

내장 브라우저를 쓰지 않고 평소 쓰는 크롬을 프록시로 연결하고 싶다면?

- 크롬 우측 상단 점 3개 -> [설정] -> 검색창에 '프록시' 입력 -> [컴퓨터 프록시 설정 열기] 클릭
- 윈도우 네트워크 설정 창이 열립니다. **[프록시 서버 사용]**을 '켬'으로 바꿉니다.
- 주소에 `127.0.0.1`, 포트에 방금 바꾼 8888을 적고 [저장]합니다.

# [수동 방법] 일반 브라우저 프록시 설정 (Mac)

- 시스템 환경설정 -> **[네트워크]** -> 우측 하단 **[고급]** 버튼 클릭.
- **[프록시]** 탭으로 이동.
- 프로토콜 선택에서 **'웹 프록시(HTTP)'**와 **'보안 웹 프록시(HTTPS)'** 두 개를 모두 체크합니다.
- 주소 칸에 `127.0.0.1`, 포트에 8888을 적고 [확인], [적용]을 누릅니다.

# [수동 방법] HTTPS 패킷 암호화 문제

WebGoat는 HTTP 통신이라 바로 패킷이 보입니다.

하지만 네이버, 구글 같은 일반 **HTTPS 통신** 사이트는 암호화되어 있어 Burp Suite가 가로채더라도 내용을 해독할 수 없고 브라우저는 보안 에러를 뿜습니다.

이를 해결하려면 Burp Suite가 생성한 **자체 암호화 루트 인증서(CA Certificate)**를 운영체제에 강제로 심어주어 신뢰하도록 만들어야 합니다.

# Burp Suite 인증서 설치 (Windows)

- 프록시가 잡힌 브라우저 주소창에 **http://burp/** 를 입력하고 접속.
- 우측 상단의 **[CA Certificate]** 클릭하여 인증서 파일 다운로드.
- 파일 더블클릭 -> [인증서 설치] -> 저장소 위치 [현재 사용자] 선택.
- **[모든 인증서를 다음 저장소에 저장]** 선택 후 [찾아보기] 클릭.
- 반드시 **[신뢰할 수 있는 루트 인증 기관]** 폴더를 골라 [다음] -> [마침].

# Burp Suite 인증서 설치 (Mac)

- 동일하게 **http://burp/** 접속 후 CA 인증서 다운로드.
- 다운받은 파일을 더블클릭하여 '키체인 접근' 창을 엽니다.
- 목록에서 **PortSwigger CA** 항목을 찾아 더블클릭합니다.
- 신뢰 탭을 열어 **[이 인증서 사용 시]** 옵션을 **[항상 신뢰]**로 변경 후 닫습니다.
- 브라우저를 껐다 켜면 완벽 적용됩니다.

# HTTP Proxies 실습 문제 풀이 시작

다시 WebGoat 화면으로 돌아와 General 섹션의 **[HTTP Proxies]** 항목 5번 페이지를 엽니다.

목표는 화면 아래쪽의 [Submit] 버튼을 눌렀을 때 날아가는 HTTP 요청을 Burp Suite로 붙잡아 다음 세 가지를 고치는 것입니다.

- 1. Method를 POST에서 GET으로 변경
- 2. x-request-intercepted:true 헤더 추가
- 3. changeMe 파라미터 값 변조

# 1단계: 패킷 가로채기 (Intercept)

Burp Suite의 Proxy 탭에서 **[Intercept is on]** 파란 버튼이 활성화된 상태인지 확인합니다.

WebGoat 페이지에서 [Submit] 버튼을 클릭하면 화면이 로딩 상태로 멈춥니다.

Burp 화면을 띄워보면 다음과 같이 원본 POST 패킷이 잡혀있습니다.

```
POST /WebGoat/HttpProxies/intercept-request HTTP/1.1 Host: 127.0.0.1:8080 ... changeMe=doesn't+matter+really
```

# 2단계: Method 강제 변환 기능

POST를 GET으로 바꾸기 위해 일일이 지우고 쓸 필요 없습니다.

Burp의 패킷 영역 빈 공간에서 마우스 우클릭을 하고 **[Change request method]**를 선택합니다.

마법처럼 첫 줄의 POST가 GET으로 바뀌며, 아래쪽 Body에 있던 `changeMe=...` 파라미터가 싹 복사되어 맨 위 URI 뒤쪽 `?changeMe=...` 형태로 자동 이동합니다.

# 3단계: 파라미터 및 헤더 변조 후 전송

URI 뒤에 달라붙은 값을 지우고 문제에서 요구한 정답 문자열로 바꿉니다.

```
GET /WebGoat/.../intercept-request?changeMe=Requests+are+tampered+easily
```

그리고 중간 헤더 영역에 엔터를 쳐서 빈 줄을 만들고 `x-request-intercepted:true`를 한 줄 수동으로 입력해 줍니다.

수정이 다 끝났다면 상단의 **[Forward]** 버튼을 눌러 서버로 보내고, **[Intercept is off]**를 눌러 대기를 풉니다.

# 프록시 변조 실습 성공!

OK

웹 브라우저의 WebGoat 화면으로 가보면 대기 중이던 페이지가 로딩을 마치고 아래와 같은 초록색 축하 문구를 띄웁니다.

**"Well done, you tampered the request as expected"**

클라이언트(브라우저)와 서버 모두 눈치채지 못하게 중간에서 완벽히 데이터를 조작해내는 프록시의 위력을 체험했습니다.

# [TIP] 공격 타겟만 필터링하기 (Scope 설정 1)

Burp를 켜두면 내가 가고 싶은 사이트 외에도, 윈도우 백그라운드 업데이트, 확장 프로그램 통신 등 쓰레기 트래픽이 엄청나게 잡혀 실습이 지저분해집니다.

이를 막기 위해 상단 **[Target] -> [Site map]** 탭으로 이동합니다.

목록에서 `http://127.0.0.1:8080` (WebGoat 서버)을 찾아 우클릭하고 **[Add to scope]**를 클릭해 타겟으로 지정합니다.

# [TIP] 공격 타겟만 필터링하기 (Scope 설정 2)

Add to scope를 누르면 "Out-of-scope 아이템들은 히스토리에서 로깅을 멈추겠습니까?" 하는 창이 뜹니다. **[No]**를 눌러줍니다.

그리고 Proxy 탭으로 돌아와, [HTTP history] 위쪽의 필터 바(Filter: Hiding CSS...)를 콕 클릭하여 엽니다.

팝업 설정 창에서 **[Show only in-scope items]** 박스에 체크하고 [Apply]를 누릅니다.
이제 WebGoat 트래픽만 아주 깔끔하게 필터링되어 보입니다.

# [TIP] 특정 URL 경로만 공격 제외하기

WebGoat 트래픽 중에서도 수시로 날아가는 `/service/...` 경로의 폴링 트래픽은 보기 싫을 수 있습니다.

Target -> Site map에서 `127.0.0.1:8080` 폴더를 펼쳐 `service` 폴더를 우클릭 한 뒤 **[Remove from scope]**를 누르면 해당 경로만 귀신같이 타겟 필터링에서 제거되어 보이지 않게 됩니다.

# 02. Developer Tools (개발자 도구)

프록시 툴만큼이나 해커들이 애용하는 도구가 바로 브라우저 자체에 내장된 '개발자 도구'입니다.

# 개발자 도구 실행 방법 3가지

HTML 구조를 뜯어고치거나(DOM 변조), 자바스크립트 코드를 실행하고,
프론트엔드단에서 발생하는 네트워크 통신을 빠르게 엿볼 수 있습니다.

- **방법 1:** 웹 페이지 빈 공간에 마우스 우클릭 -> **[검사]** 클릭
- **방법 2:** 크롬 우측 상단 점 3개 -> [도구 더보기] -> [개발자 도구]
- **실행 방법 3: 단축키 F12 또는 Ctrl + Shift + I (Mac: Cmd + Option + I)**

# Console 탭을 이용한 JS 변조 실습

WebGoat [Developer Tools] 메뉴 첫 번째 문제입니다.

F12를 눌러 개발자 도구를 켜고 **[Console]** 탭으로 이동한 뒤, 문제에서 요구한 자바스크립트 함수를 콘솔 입력창에 직접 타이핑 후 엔터를 칩니다.

```
> webgoat.customjs.phoneHome()
```

서버가 아닌 클라이언트 단에 은닉되어 있던 로직이 돌면서 "phoneHome Response is 1832506377" 같은 정답 난수를 로그로 뱉어냅니다. 이를 화면에 입력하면 통과입니다.

# Network 탭을 이용한 트래픽 엿보기

두 번째 문제는 네트워크 탭을 사용하는 실습입니다.

개발자 도구 상단 **[Network]** 탭을 클릭하여 켜둔 상태로, 화면의 [Go!] 버튼을 누릅니다.

하단 통신 리스트에 network라는 이름의 API 호출이 번쩍 뜹니다. 이를 클릭하고 우측 **[Payload]** (혹은 Headers 하단) 탭을 보면 `networkNum: 1.3230...` 이라는 난수 값이 서버로 쥐도 새도 모르게 전송되고 있음을 적발할 수 있습니다.

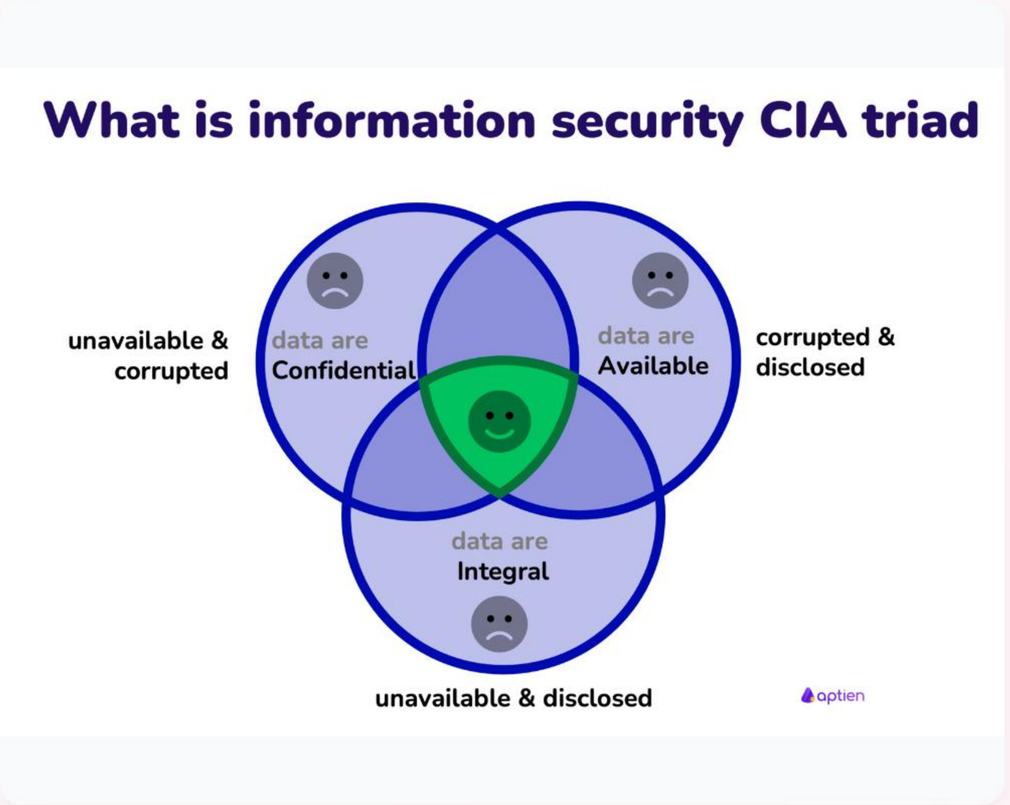
# 03. CIA Triad

모든 정보보안 정책과 프레임워크의 절대적인 기준이 되는
가장 기초적인 '보안의 3대 요소'에 대해 알아봅니다.

# 보안의 3대 요소 (CIA Triad)

CIA Triad는 정보보안의 절대적인 뼈대가 되는 세 가지 기둥(기밀성, 무결성, 가용성)을 의미합니다.

이 세 가지 요소 중 단 하나라도 훼손된다면 그 시스템은 보안상 완벽하게 타협(손상)된 것으로 간주할 만큼 매우 중요합니다.



# C - Confidentiality (기밀성)

C

오직 인가된(허락된) 사람들만 정보에 접근하고 읽을 수 있도록 통제하는 성질입니다.

내가 보낸 카카오톡 메시지를 해커가 중간에서 훔쳐보지 못하도록 '암호화' 처리를 하는 행위가 바로 이 기밀성을 지키기 위한 대표적 보안 조치입니다.

# I - Integrity (무결성)

적절한 권한을 가진 사람들만이 인가된 방법으로 정보를 추가, 삭제, 수정할 수 있게 보호하는 성질입니다.

누군가 은행 서버를 해킹해서 내 통장 잔고 10만원을 100억원으로 조작해 버린다면? 데이터의 위변조가 일어났으므로 무결성이 박살 난 것입니다.

# A - Availability (가용성)

A

권한을 가진 사용자가 시스템과 데이터가 필요할 때 언제,
어디서든 정상적으로 서비스를 이용할 수 있는 상태입니다.

해커들이 서버에 트래픽 폭탄을 날리는 디도스(DDoS)
공격으로 서버를 뻗게 만들면, 정상 유저들이 접속을 못 하므로
바로 이 가용성이 훼손된 것입니다.

# CIA Triad 실습문제 (Q1)

**문제: 공격자가 기밀성을 해치는 행위는 무엇인가?**

정답은 **"이름과 이메일이 저장된 데이터베이스를 훔쳐 다른 웹 사이트에 업로드하는 행위"** 입니다.

인가되지 않은 자가 개인정보를 무단 열람하고 유출하여 남들이 보게 만들었으므로 명백한 기밀성 침해입니다.

# CIA Triad 실습문제 (Q2)

**문제: 공격자가 무결성을 해치는 행위는 무엇인가?**

정답은 **"데이터베이스에 저장된 하나 이상의 사용자 이름과 이메일을 변경하는 행위"** 입니다.

네트워크 트래픽을 몰래 엿듣는 행위 자체는 기밀성 침해에 그치지만, 직접 데이터를 조작(수정)하는 순간 무결성이 침해된 것입니다.

# CIA Triad 실습문제 (Q3)

**문제: 공격자가 가용성을 해치는 행위는 무엇인가?**

정답은 **"서버에 서비스 거부(DoS) 공격을 가하는 행위"** 입니다.

소프트웨어 버그나 물리적 전원 차단뿐 아니라, 트래픽 폭탄으로 자원을 고갈시켜 시스템 운영을 불가하게 만드는 것이 가용성 파괴의 전형입니다.

# CIA Triad 실습문제 (Q4)

**문제: 세 가지 보안 목표 중 적어도 하나라도 침해된다면?**

정답은 **"단 하나의 요소만 침해당하더라도 시스템 보안은 완전히 타협(손상)된 것으로 간주한다"** 입니다.

세 가지 중 무엇이 더 중요하고 덜 중요하고는 없습니다. 완벽한 삼위일체가 이루어져야만 보안성이 확보됩니다.

# Section 02. Broken Access Control

OWASP 취약점 1위로 등극한 '부적절한 접근 제어'.
서비스 데이터와 기능에 대한 권한 검증이 뚫리는 사례들을 파헤칩니다.

# 01. Hijack a session (세션 탈취)

개발자가 **직접 개발한 커스텀 세션이나 토큰**을 사용할 때, 그 생성 알고리즘이 단순하여 무차별 대입 공격(Brute-force)으로 세션을 탈취할 수 있는 취약점을 다룹니다.

목표: 타인(인증된 다른 사용자)의 세션 쿠키값을 예측해 내어 로그인 우회하기.

# 1단계: 샘플 쿠키 수집

WebGoat 실습 2번 항목에서, 로그인 정보가 없으니 무작정 `test / test` 로 아무렇게나 입력하고 [Access] 버튼을 누릅니다.

Burp Suite의 **[HTTP history]** 탭을 켜서 방금 날아간 POST 요청에 대한 응답(Response)을 살펴봅니다.

```
Set-Cookie: hijack_cookie=75817714008441290 83-1680611361571; path=/WebGoat; ...
```

실패한 로그인에 대해서도 `hijack_cookie` 값이 발급되는 것을 확인할 수 있습니다.

# 2단계: Intruder로 다량 요청 쏘기

하나의 쿠키만으로는 규칙을 알 수 없습니다. 수십 개의 쿠키를 빠르게 수집하기 위해, 해당 요청을 우클릭하여 **[Send to Intruder]**로 보냅니다.

Intruder의 [Positions] 탭에서 우측 **[Clear §]**를 눌러 초기화 한 뒤, Body 파라미터 중 아무 곳(예: password=test)을 드래그해 **[Add §]**로 변조 타겟을 잡아줍니다.

# 3단계: Payload 설정 및 공격

[Payloads] 탭으로 이동해 설정합니다.

- Payload type: **Numbers** 선택
- Number range Type: **Sequential** (순차 대입)
- From 0, To 9, Step 1 입력

상단 **[Start attack]** 버튼을 클릭하여 총 10번의 요청을 1초 만에 서버에 다다닥 발사합니다.

# 4단계: 생성 규칙(패턴) 분석 1

Attack Results 창이 뜨면 여러 개의 200 OK 응답이 보입니다. 이 중 4번과 5번의 Response 탭을 번갈아 가며 `hijack_cookie` 값을 매의 눈으로 비교해봅니다.

예시:

`...129117-1680612448416` (4번)

`...129118-1680612448594` (5번)

하이픈 앞의 긴 숫자가 **단순히 1씩 규칙적으로 증가**하고 있다는 끔찍한 사실을 발견했습니다!

# 5단계: 생성 규칙(패턴) 분석 2

하이픈(-) 뒷자리의 숫자 168061... 은 무엇일까요?

이것은 리눅스나 프로그래밍에서 흔히 쓰이는 **Unix 타임스탬프(밀리초 시간값)**입니다.

결국 이 세션 쿠키는 [의미없는긴숫자 + 1씩증가] - [현재시간] 이라는 누구나 100% 때려 맞출 수 있는 초보적인 알고리즘으로 만들어진 것입니다.

# 6단계: 누락된 번호(정답) 추적

로그를 쭉 내리다 보면, 어느 순간 앞자리 시퀀스가 **1이 아니라 2가 점프하는 구간**을 발견합니다. (예: 119 다음 121 등장)

이것은 우리가 공격을 날리는 그 찰나에, **진짜 인증된 유저가 접속해서 중간 번호(120번) 쿠키를 발급받아 빼갔다**는 뜻입니다.

이제 우리가 탈취할 타겟 쿠키의 앞부분은 무조건 **120**으로 확정되었습니다.

# 7단계: 정밀 타격 (무차별 대입)

앞자리는 찾았고, 타임스탬프(뒷자리)만 맞추면 됩니다.

119번의 뒷자리가 785고, 121번의 뒷자리가 984라면 타겟은 그사이의 시간대입니다.

다시 Intruder Position으로 가서, 이번엔 요청 헤더에 `Cookie: hijack_cookie=...120-1680...§785§` 형태로 뒷 세 자리만 타겟을 잡아줍니다.

Payload 범위를 From 785, To 984로 잡고 200번의 공격을 쏩니다.

# 세션 탈취 성공 및 조치방안

!

수백 개의 요청 중 길이가 미세하게 다른 유일한 응답 하나(예: 끝자리 786)를 확인해보면 **"lessonCompleted": true**를 뱉어내며 하이재킹에 대성공합니다.

**조치 방안:** 프레임워크 내장 세션이 아닌 자체 세션/토큰을 발급할 때는, **반드시 길고 복잡한 영문/숫자 조합의 강력한 난수 기반 알고리즘**을 사용해 예측을 불허하게 만들어야 합니다.

# 02. IDOR 취약점 기초

**Insecure Direct Object References (안전하지 않은 직접 객체 참조)**

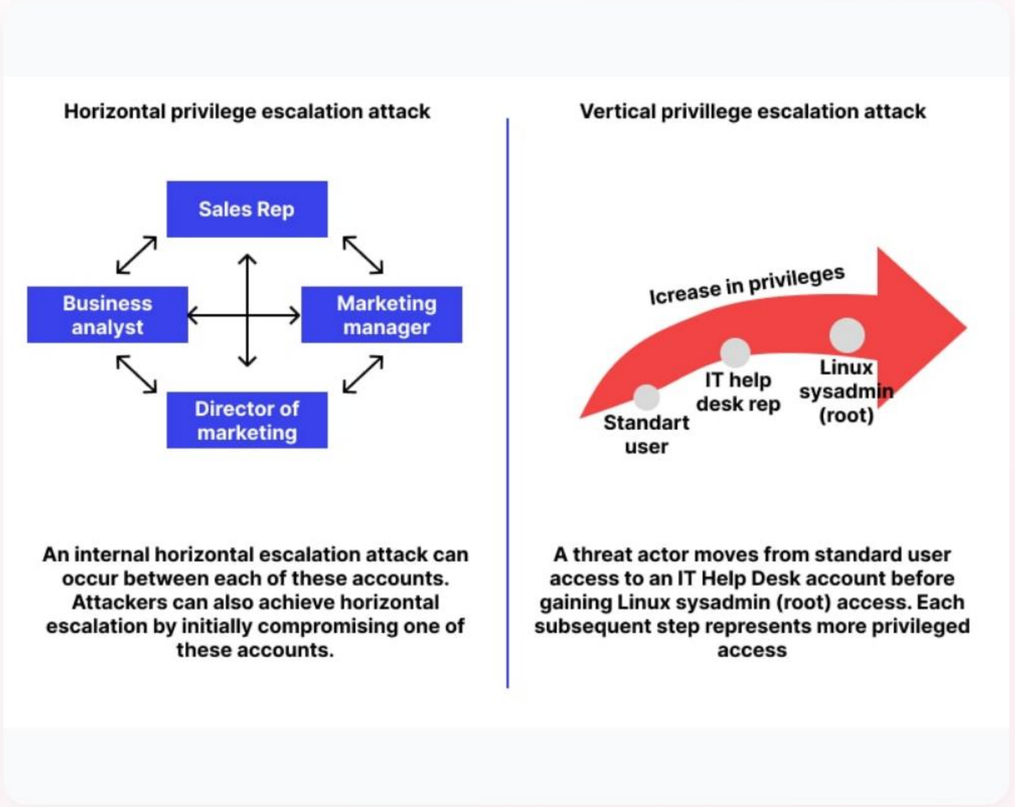
클라이언트가 브라우저나 패킷을 통해 특정 번호(예: 유저ID 100번)를 요청할 때, 서버가 그 번호의 주인이 현재 로그인한 사람이 맞는지 **권한 검증을 빼먹어 버리는** 취약점입니다.

단순히 남의 게시글/장바구니 번호를 주소창에 치는 것만으로 남의 데이터를 다 털어볼 수 있습니다.

# 수평적 권한 상승 vs 수직적 권한 상승

IDOR 취약점은 두 가지 방향의 끔찍한 권한 상승을 야기합니다.

- **수평적(Horizontal):** 나와 동일한 일반 사용자 등급을 가진 남의 계정에 몰래 접근하는 것
- **수직적(Vertical):** 일반 사용자인 내가, 나보다 더 높은 권한을 가진 '관리자'의 데이터에 접근하여 시스템을 농락하는 것



# IDOR 실습 1: 은닉된 파라미터 훔쳐보기

문제에 주어진 허술한 계정 `tom / cat`으로 먼저 정상 로그인합니다.

[View Profile] 버튼을 누르면 화면에는 이름, 색상, 사이즈 정보 3개만 보입니다.

하지만 Burp Suite HTTP History에서 이 응답(Response) Raw JSON 데이터를 까보면, 화면엔 출력되지 않는 **role: 3** 값과 **userId: 2342384** 값이 통째로 노출되어 있음을 발견합니다.

# RESTful API 아키텍처 특성

이 웹 앱은 최신 백엔드 구조인 RESTful 패턴을 따릅니다.

REST 구조에서는 데이터를 조회할 때 `/profile?id=123` 형태가 아니라, **URI 자체에 자원의 식별자를 포함**시키는 `/profile/123` 방식을 주로 사용합니다.

또한, 조회는 GET, 수정은 PUT, 생성은 POST 등 HTTP 메서드만 바꿔서 하나의 주소로 뼈대를 재활용하는 세련된 구조를 갖습니다.

# IDOR 실습 2: REST 패턴으로 경로 예측

현재 내 프로필을 부르는 API가 /WebGoat/IDOR/profile로 되어있는데, 객체 식별자가 생략되어 있습니다.

방금 응답 데이터에서 훔쳐본 내 **userId(2342384)**를 뒤에 덧붙여 WebGoat/IDOR/profile/2342384 형태로 주소창에 치면 직접 객체 참조 방식으로 내 프로필이 다시 성공적으로 렌더링됩니다.

# IDOR 실습 3: 타인 프로필 도둑질 (수평 상승)

내 userId가 순차적인 숫자라면, 다른 사람 번호도 비슷할 것입니다.

Burp Suite **[Repeater]** 탭을 켜고, 방금의 URI에서 끝자리를 살짝 올려 `.../profile/2342388` 로 변조하여 전송(Send)해 봅니다.

접근 제어 검증이 뚫려있어, 내 계정(Tom)으로 로그인했는데도 **Buffalo Bill**이라는 완전 다른 사용자의 프로필 JSON 데이터가 고스란히 응답으로 반환됩니다.

# IDOR 실습 4: 타인 프로필 무단 조작 (수직 상승)

단순 조회를 넘어, RESTful 규칙을 악용해 타인의 정보를 파괴해 봅니다.

Repeater에서 HTTP 메서드를 조회가 아닌 수정용인 **PUT**으로 바꿉니다. Content-Type은 `application/json`으로 명시합니다.

Body 란에 훔쳐낸 Buffalo Bill의 JSON 포맷을 붙여넣고, 권한을 상위 권한인 `"role":1`로 올리고, 색상은 `"color":"red"`로 조작하여 날려보냅니다.

# 무단 데이터 덮어쓰기 성공

전송 즉시 서버는 200 OK 응답과 함께 수정된 타겟 유저의 데이터를 반환합니다.

권한 검증이 아예 동작하지 않았기 때문에, 일반 유저 Tom이 타인 Buffalo Bill의 권한을 관리자급(1)으로 강제로 승격시켜 버리는 끔찍한 보안 사고(수직적 권한 상승)가 성공했습니다.

이것이 왜 발생하는지 실제 자바 코드를 까서 확인해 보겠습니다.

# 백엔드 원인 분석 (조회 Controller)

Github에서 코드를 열어봅니다.

`src/main/java/org/owasp/webgoat/lessons/idor/IDORViewOtherProfile.java`

```
    String authUserId = (String) userSessionData.getValue("idor-authenticated-user-id"); if (userId != nul
l && !userId.equals(authUserId)) { // on the right track return success(this); }
```

authUserId는 IDORLogin.java에서 로그인 시 세션에 할당되는 식별자이고, userId는 주소창에 친 타겟 번호입니다.

# 의도된 논리적 오류 (Logic Flaw)

코드를 보면 기이하게도 `!userId.equals(authUserId)`, 즉 **요청한 번호와 로그인한 내 번호가 다를 때(!) 오히려 조회를 허용해버리는** 함정 코드가 심어져 있습니다.

이 때문에 정상적인 자신의 프로필 접근은 막히고 남의 프로필에 접근할 때만 뚫리는 IDOR 실습 로직이 완성된 것입니다.

# 백엔드 원인 분석 (수정 Controller)

이번엔 PUT을 처리하는 수정용 컨트롤러 코드를 뜯어봅니다.

`src/main/java/org/owasp/webgoat/lessons/idor/IDOREditOtherProfile.java`

```
    if (userSubmittedProfile.getUserId() != null && !userSubmittedProfile.getUserId().equals(authUserId))
{ currentUserProfile.setRole(userSubmittedProfile.getRole()); }
```

수정 컨트롤러 역시 전송받은 Body 내의 userId와 세션의 userId가 다를 때만 `setRole()` 덮어쓰기 로직을 수행하도록 코딩되어 있습니다.

# IDOR 코드 완벽 조치 방안

이 코드를 안전하게 고치려면 아주 간단합니다.

조건문 `!userId.equals(authUserId)` 의 가장 앞에 있는 논리 부정 연산자 **느낌표(!) 하나만 지워주면** 됩니다.

이렇게 하면 타겟 ID와 로그인 세션 ID가 완전히 일치하는 '본인'의 객체일 때만 조회와 수정을 허용하도록 완벽한 접근 제어가 구현됩니다.

# 재컴파일(Build) 및 검증 (Maven)

느낌표를 지워서 자바 코드를 수정한 뒤, 터미널에서 컨테이너 내부로 들어가 Maven으로 서버를 다시 빌드합니다.

```
root@...:# ./mvnw spotless:apply && ./mvnw clean install -DskipTests
```

빌드 완료 후 `./mvnw spring-boot:run`으로 서버를 다시 켜고 아까의 Repeater 공격을 쏴보면, 이제는 얄짤없이 에러를 뱉으며 공격이 차단됨을 확인할 수 있습니다.

# 03. Missing Function Level Access Control

**기능 수준의 접근 제어 누락:** 사용 권한이 없는 특정 '기능(메뉴버튼, API Endpoint)' 자체를 임의로 호출하여 사용할 수 있는 취약점입니다.

**IDOR과의 차이점:** IDOR은 객체/데이터 단위의 검증 누락이라면, 이 취약점은 공지사항 작성, 관리자 콘솔 진입 등 **기능 단위의 권한 검증 누락**을 의미합니다.

# 프론트엔드 은닉의 함정 (실습 1)

개발자가 권한 없는 관리자 메뉴를 백엔드에서 검증하지 않고, **단순히 HTML/CSS에서 안 보이게 (display:none) 숨기는 심각한 실수**를 자주 범합니다.

브라우저에서 F12 개발자 도구를 켜서 DOM 구조를 뜯어보면, 우측 끝에 코드가 떡하니 노출되어 있습니다.

# 숨겨진 기능 강제 활성화

이 요소에 적용된 CSS를 보면 `visibility: hidden;` 처리가 되어 있습니다.

해커는 서버를 해킹할 필요 없이 그냥 내 브라우저에서 이 **hidden 클래스 텍스트를 더블클릭해서 백스페이스로 싹 지워버립니다.**

그 즉시 화면 상단에 감춰져 있던 **[Admin] -> [Users, Config]** 꿀메뉴 2개가 버젓이 나타납니다.

# 숨겨진 메뉴 클릭 및 에러 발생

방금 살려낸 [Admin] -> [Users] 메뉴를 클릭하면 권한 부족이라며 404 에러가 뜹니다.

포기하지 않고 Burp Suite HTTP History에서 패킷을 봅니다. `/access-control/users`로 날아갔습니다.

서버 경로 매핑 문제로 판단하고, 앞부분에 `/WebGoat`를 손수 붙여 Repeater로 다시 찔러 봅니다. 이번엔 500 Internal Server Error로 바뀌었습니다!

# 500 에러의 원인 분석

백엔드 API 소스를 열람할 수 있다고 가정하고 다음 경로의 파일을 봅니다.

`src/main/java/org/owasp/webgoat/lessons/missingac/MissingFunctionACUsers.java`

```
@GetMapping(path="access-control/users", consumes="application/json")
```

원인은 백엔드에서 이 API 호출 시 무조건 **Content-Type이 application/json이어야 한다**고 강제해 놨기 때문이었습니다. 브라우저의 기본 GET 요청엔 이 헤더가 없었습니다.

# 백엔드 접근 통제 부재 증명

Repeater에서 요청 헤더에 `Content-Type: application/json`을 한 줄 손수 타이핑해서 추가하고 다시 날립니다.

결과는 충격적입니다. **접근 통제 로직이 아예 없었기 때문에**, 관리자만 볼 수 있는 전체 사용자의 아이디와 민감한 User Hash 목록이 화면에 주르륵 쏟아져 나옵니다.

프론트엔드 화면에서만 숨기는 것이 얼마나 부질없는 짓인지 증명되었습니다.

# 긴급 패치 후의 우회 공격 (실습 2)

회사가 문제를 인지하고 '사용자 정보 조회(GET)' 기능에 긴급하게 권한 체크 로직을 넣어서 403 Forbidden 방어를 세웠습니다.

하지만 조회 API만 막았을 뿐, **'사용자 추가(POST)' API에 대한 방어는 까먹었을 수 있습니다.**

메서드를 **POST**로 바꾸고 `/access-control/users-admin-fix` 주소로 찔러 봅니다.

# 백도어 어드민 계정 생성 (POST)

요청 Body에 JSON 포맷으로 내가 원하는 백도어 관리자 계정 데이터를 밀어 넣습니다.

```
{"username":"webhacking", "admin":true, "password":null}
```

전송 결과, 서버는 200 OK를 떨어뜨리며 **시스템 내부에 진짜 최고 관리자 권한을 가진 내 계정이 무단으로 생성**되어 버렸습니다.

# 백도어 생성 후 해시 재탈취 성공

이제 저는 최고 관리자 계정 보유자입니다.

다시 아까 막혔던 사용자 정보 조회(GET) 요청을 쏘면, 새로 판 관리자 세션을 인식하고 403 에러 없이 **모든 유저 해시 데이터를 완벽하게 뱉어냅니다.**

이것이 기능 접근 통제를 일부만 방어했을 때 뚫리는 전형적인 우회 시나리오입니다.

# Missing ACL 소스코드 원인 분석

긴급 패치된 사용자 추가(POST) 자바 코드 파일의 내부를 봅니다.

`src/main/java/org/owasp/webgoat/lessons/missingac/MissingFunctionACUsers.java`

```
    if (currentUser != null && currentUser.isAdmin()) { return users... }
```

GET(조회) 부분엔 위처럼 방어가 되어 있습니다. 하지만 그 아래쪽 @PostMapping (사용자 추가) 컨트롤러를 보면 userRepository.save(newUser);만 달랑 있을 뿐,

**어드민인지 체크하는 if문이 쏙 빠져있습니다.**

# 코드 패치 및 보안성 강화

해당 파일(MissingFunctionACUsers.java)의 POST 컨트롤러 안에도 조회 시 썼던 로직과 똑같이 현재 세션을 가져와 검증하는 방어막을 세워줍니다.

```
    var currentUser = userRepository.findByUsername(webSession.getUserName());
if (currentUser != null && currentUser.isAdmin()) { userRepository.save(newUse
r); ...
```

이렇게 수정한 뒤 메이븐 빌드로 재구동하면, 더 이상 POST 꼼수 통과가 불가능해집니다.

# 04. Spoofing an Auth Cookie (세션 변조)

안전한 랜덤 난수가 아닌, **사용자 이름 등의 텍스트를 단순 암호화(인코딩) 조합하여 쿠키로 발급**하는 허술한 서비스의 맹점을 다루는 챕터입니다.

공격자가 내가 가진 쿠키값의 생성 알고리즘 규칙성만 파악해 낸다면, 이를 역산하여 관리자의 위조 쿠키(Spoofing)를 만들어내고 로그인 없이 권한을 완전히 탈취할 수 있습니다.

# 인증 쿠키 발급 규칙 알아내기 (1)

먼저 제공된 테스트 계정 `webgoat / webgoat` 로 로그인해 봅니다.

화면 아래쪽이나 패킷을 보면 `spoof_auth=N2E0Mz...` 형태의 굉장히 수상한 쿠키값이 떨어집니다.

끝에 == 가 붙거나 대소문자/숫자만 있는 폼을 보아하니 전형적인 **Base64 인코딩** 데이터입니다. 이를 뜯어보러 갑니다.

# 바이너리 통신 시 데이터 누락이 발생하는 이유

Base64 디코딩을 하기 전에, **왜 Base64 같은 인코딩이 필요한지** 알아야 합니다.

과거 네트워크 및 통신 장비(라우터 등)는 영문자와 기본 기호만 처리하는 **7비트 ASCII** 포맷을 기준으로 설계되었습니다. 사진, 파일 등 8비트 이진(Binary) 데이터를 그대로 전송하면 어떻게 될까요?

시스템이 이를 줄바꿈(CR/LF)이나 통신 제어 문자(Control Character)로 오인하여 **데이터 구조를 변형시키거나 중간에 잘라버려 누락이 발생**합니다. 이를 피하기 위해 안전한 텍스트로 변환하는 과정이 필요합니다.

# Base64 인코딩의 특징

- 앞서 설명한 8비트 이진 데이터 누락을 방지하기 위해 널리 쓰이는 표준입니다.
- 시스템이 절대 오해하지 않는 안전한 **64개의 문자(알파벳 대소문자 52개, 숫자 10개, 기호 '+', '/')**만을 이용하여 데이터를 인코딩합니다.
- 문자열 끝에 빈자리를 채워 넣는 패딩 기호인 '=' 가 자주 붙는 특징이 있어, 해커는 값을 보고 단숨에 Base64 여부를 추측할 수 있습니다.

# 쿠키값 디코딩 (Decoding)

Burp Suite 상단 **[Decoder]** 탭에 쿠키값 전체를 붙여넣습니다.

우측 **Decode as → Base64**를 선택하면 16진수 데이터가 도출됩니다.

한 번 더 **Decode as → ASCII hex**를 돌려 평문으로 변환합니다.

# 허술한 쿠키 알고리즘 정체 발각

16진수 디코딩 결과, 긴 더미 문자열 끝에 `taogbew` 라는 값이 매달려 있는 것을 발견했습니다.

가만히 보니 이는 우리가 방금 로그인한 계정명인 **'webgoat'를 단순히 알파벳 거꾸로 뒤집어 놓은 문자열**입니다!

즉, 이 서버는 "앞부분 고정 더미 문자열 + 계정명 거꾸로"를 합친 뒤, Hex와 Base64로 감싸서 쿠키를 굽는 초보적인 방식을 쓰고 있었습니다.

# 교차 검증 (어드민 계정 쿠키 분석)

확신을 위해 제공된 두 번째 계정인 admin / admin 으로 로그인해서 그 쿠키도 뽑아 디코딩을 돌려봅니다.

역시나 디코딩 결과 맨 끝에 nimda ('admin'의 역순)가 예쁘게 붙어 있습니다.

가설이 100% 입증되었습니다. 이제 타겟팅을 해 봅니다.

# Tom 계정 탈취를 위한 쿠키 위조

문제의 목표는 **Tom** 계정으로 로그인하는 것입니다.

방금 전 디코더 탭의 아랫쪽 빈칸에, 고정된 앞부분 더미 문자열을 그대로 복사하고 맨 끝에만 `mot` ('tom'의 역순)를 붙여 평문을 직접 완성합니다.

그리고 리버스 과정의 역방향으로, **Encode as -> ASCII hex** 변환 후, 다시 한번 **Encode as -> Base64** 인코딩을 먹여 최종 가짜 쿠키를 찍어냅니다.

# 위조 쿠키 전송 및 권한 탈취 성공

이제 브라우저로 돌아가, 아이디 비밀번호 칸에 아무 내용이나 치고 로그인을 시도하되 **Burp Suite로 패킷을 낚아챕니다(Intercept).**

패킷 본문의 Cookie 헤더를 아까 직접 수제작한 `spoof_auth=가짜Base64문자열...` 로 싹 바꿔치기한 뒤 [Forward] 전송 버튼을 누릅니다.

놀랍게도 서버는 나를 Tom으로 완벽히 착각하고 **Congratulations** 메시지와 함께 문을 열어줍니다.

# Section 03. Cryptographic Failures

암호화 모듈 설계 오류나 약한 알고리즘을 사용함으로써
초래되는 무서운 민감 정보 유출 사태를 심층적으로 다룹니다.

# 01. Crypto Basics (기초 암호화 개념)

많은 개발자들이 **인코딩과 암호화(Encryption)의 개념을 혼동**하여 대형 사고를 칩니다.

인코딩은 특정 데이터 전송 포맷에 맞게 문자 껍데기만 살짝 바꿔두는 것으로,
비밀번호나 보안키가 없어도 누구나 1초 만에 원본으로 되돌릴 수 있는 기술입니다.

보안을 위해서는 반드시 키를 가진 자만이 복원할 수 있는 '암호화'를 거쳐야 합니다.

# Basic Authentication과 Base64의 치명적 단점

웹 애플리케이션의 인증 중 **Basic Auth** 방식은 유저명과 패스워드를 admin:1234 형식으로 합친 뒤, 단순히 **Base64 인코딩**만 거쳐 헤더에 실어 보냅니다.

```
Authorization: Basic d2ViaGFja...yZA==
```

이 패킷이 중간에 캡처당하면, 해커는 Burp Suite Decoder 탭에서 클릭 한 번에 즉시 평문 아이디/비밀번호를 통째로 얻어냅니다. 따라서 HTTPS 환경 적용이 생명줄과 같습니다.

# 웹에서 쓰이는 기타 인코딩 방식들

- **URL Encoding:** 브라우저 주소창에 넣을 수 없는 한글, 특수문자, 띄어쓰기 등을 %20 같은 16진수 문자열로 포장하는 기술입니다.
- **HTML Entity Encoding:** 웹 페이지 HTML 코드 상에서 < > 같은 태그 명령어들을 단순 텍스트로 브라우저에 출력하기 위해 < 형태로 치환하는 기술입니다. (크로스사이트 스크립팅/XSS 해킹 방어의 핵심 원리입니다)
- **Base64:** 바이너리 데이터를 안전한 ASCII 문자로만 구성되도록 변환하는 방식

# UUEncode (Unix to Unix)

과거 이메일에 그림 파일 등 일반 텍스트가 아닌 바이너리 첨부 파일을 넣을 때, 네트워크 중간 기지국들을 거치며 데이터가 파손되는 것을 막기 위해 ASCII 텍스트 형태로 변환하던 고전적인 인코딩입니다.

현재는 MIME 방식과 Base64 인코딩 표준으로 완전히 대체되어 역사 속으로 사라진 기법입니다.

# XOR Encoding (XOR 암호화)

단순한 비트단위 배타적 논리합(XOR) 연산을 사용해 평문을 꼬아놓는 기법입니다.

과거 IBM WebSphere 서버 같은 곳에서 설정 파일 내부의 중요 패스워드를 가려놓기 위한 1차적인 난독화 수단으로 사용되었습니다.

하지만 알고리즘과 디폴트 키가 너무 뻔히 노출되어 있어, 절대 암호화 목적으로 사용해선 안 됩니다.

# XOR 디코딩 실습의 교훈

문제에 주어진 암호화된 스트링 `{xor}Oz4rPj0...` 을 복사합니다.

인터넷에 **'WebSphere xor decoder'** 라고 검색하여 나오는 유명 툴 사이트에 해당 문자열을 붙여넣고 디코드 버튼을 누르면, 평문 비밀번호가 허무하리만치 쉽게 뽑혀 나옵니다.

시스템 구축 시 제공되는 제조사의 기본(Default) 암호화 방식을 맹신하지 말고, 반드시 독자적이고 강력한 최신 암호화 세팅으로 튜닝해야 함을 보여줍니다.

# Plain Hashing (단방향 해싱)

해싱은 문서를 믹서기에 갈아 넣는 것과 같아서, 결과값인 '해시'만 보고는 원문이 무엇인지 절대 복원할 수 없는 **불가역적(단방향) 암호 기술**입니다.

문서나 비밀번호의 '변조(무결성)'를 탐지할 때 주로 쓰이며, 단 1바이트만 글자가 바뀌어도 전혀 다른 형태의 긴 해시 문자열을 뱉어냅니다.

# MD5와 SHA-1의 몰락 (충돌 저항성 붕괴)

과거 가장 많이 쓰이던 MD5나 SHA-1 알고리즘은 결과값이 짧고 컴퓨터의 연산 속도가 빨라짐에 따라 보안 업계에서 퇴출되었습니다.

왜냐하면 서로 다른 원본 데이터를 넣었는데 결과물 해시값이 우연히 똑같이 나와버리는 **'해시 충돌(Collision)'** 현상이 발견되었고, 심지어 이를 고의로 만들어낼 수 있음이 수학적으로 입증되었기 때문입니다.

# Rainbow Table 공격 (레인보우 테이블)

해싱이 복호화가 안 된다고 완벽한 것은 아닙니다. 공격자들은 "123456", "password" 같은 수억 개의 흔한 평문들을 미리 해싱 알고리즘에 돌려 **[원문-해시값 짝짓기 사전]**을 테라바이트급으로 구축해 두었습니다.

DB가 해킹당하면 유출된 해시 문자열을 이 사전에 검색(Lookup)만 하여 0.1초 만에 원문 비밀번호를 찾아냅니다.

# Salted Hashes (소금치기 기법)

## Salt

레인보우 테이블 공격을 방어하는 최선의 방법은 해싱을 하기 전, 사용자 비밀번호 원문 뒤에 **임의의 복잡한 난수 문자열 (Salt)**을 하나 더 붙이는 것입니다.

소금을 치면, 아무리 뻔한 비밀번호라도 완전히 세상에 존재하지 않는 복잡한 원문이 되어버리기 때문에 공격자의 레인보우 테이블 사전에 매칭되지 않아 공격을 원천 봉쇄할 수 있습니다.

# 단방향 암호화 크래킹 실습 (1)

실습문제에서 던져준 두 개의 긴 해시 데이터 원문을 찾는 미션입니다.

Salt가 뿌려지지 않은 순수 구형 해시이므로 레인보우 테이블 사전을 제공하는 사이트를 활용하면 됩니다.

구글에 **CrackStation**을 검색하여 온라인 패스워드 해시 크래커 툴에 접속합니다.

# 단방향 암호화 크래킹 실습 (2)

첫 번째 해시값을 CrackStation 텍스트 박스에 넣고 로봇 방지 캡차 통과 후 크랙 버튼을 누릅니다.

결과 화면에 MD5 타입으로 암호화되었으며 원문은 password 였음이 즉각 도출됩니다.

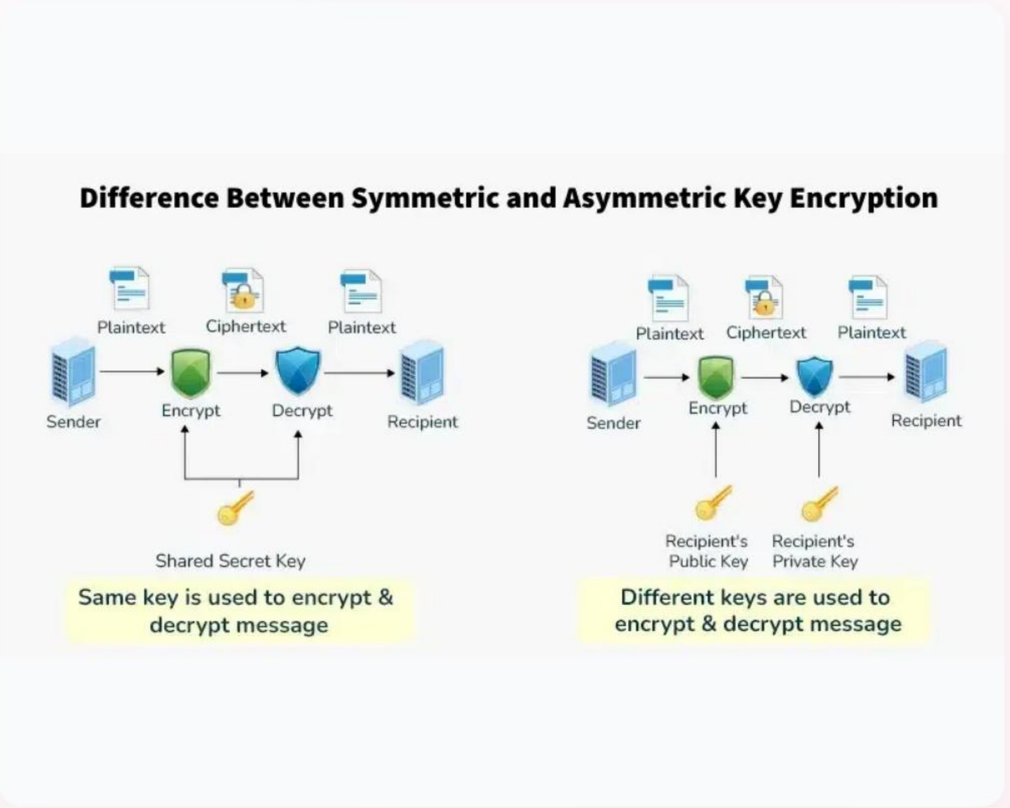
두 번째 해시값 역시 넣고 돌려보면, SHA-256으로 알고리즘만 달랐을 뿐 역시 평문은 password로 허무하게 뚫리는 것을 목격할 수 있습니다.

# Symmetric Encryption (대칭키 암호화)

단방향이 아니라, 다시 원문으로 되돌릴 수 있는 양방향 암호화 기술 중 하나입니다.

금고를 **잠글 때(암호화)와 열 때(복호화) 완전히 똑같은 열쇠 1개**를 서로 나눠 가지는 방식입니다.

연산 속도가 엄청나게 빠르고 효율적이지만, 열쇠를 상대방에게 택배로 넘겨주는 과정에서 해커에게 열쇠만 탈취당하면 보안이 완전히 붕괴되는 치명적 단점이 있습니다. (AES 알고리즘 등)



# Asymmetric Encryption (비대칭키 암호화)

열쇠 배달 사고를 원천 차단하기 위해 **잠글 때 쓰는 공개키(Public Key)**와 **열 때 쓰는 개인키(Private Key)** 두 개의 열쇠를 한 세트로 운용하는 혁신적인 방식입니다.

잠금용 키는 인터넷에 마음껏 뿌려도, 열어볼 수 있는 마스터키는 내 컴퓨터 안에만 안전히 보관되어 유출 우려가 극히 적습니다.

보안은 강력하지만 복잡한 수학(소인수분해 등)이 들어가 연산 속도가 엄청나게 느리다는 단점이 있습니다. (RSA 알고리즘 등)

# SSL/TLS 프로토콜의 등장

앞서 배운 **HTTP**는 암호화되지 않아 누구나 중간에서 내용을 들여다볼 수 있습니다 (패킷 스니핑).

이를 방지하기 위해 **전송 계층(Transport Layer)에서 데이터를 강력하게 암호화하여 보호하는 프로토콜**이 바로 SSL(Secure Sockets Layer)과 그 후속 버전인 TLS(Transport Layer Security)입니다.

이 프로토콜이 적용된 HTTP가 바로 우리가 안전하게 사용하는 **HTTPS**입니다.

# 봉투 암호화 (Envelope Encryption) 체계

대칭키의 '속도'와 비대칭키의 '안전성'을 결합한 하이브리드 암호화의 핵심 원리입니다.

- 실제 대용량 데이터는 빠른 **대칭키(데이터 암호화 키, DEK)**로 암호화하여 봉투에 넣습니다.
- 그리고 그 봉투를 여는 열쇠(대칭키) 자체를 **비대칭키(키 암호화 키, KEK)의 공개키**로 한 번 더 안전하게 암호화하여 함께 전송합니다.
- 수신자는 자신의 **개인키**로 열쇠를 먼저 풀고, 그 열쇠로 봉투 안의 데이터를 해독합니다.

# 현대 웹의 하이브리드 암호화 (HTTPS)

HTTPS 환경은 위에서 배운 **봉투 암호화** 방식을 통해 대칭키와 비대칭키의 단점을 상호 보완합니다.

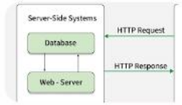
- **초기 접속 시 (Handshake):** 느리지만 안전한 **비대칭키** 기술을 사용해, 앞으로 진짜 쓸 '대칭키 열쇠'를 안전하게 포장해서 교환합니다.
- **본격 통신 시:** 안전하게 교환받은 **대칭키**를 활용하여 영화 다운로드, 채팅 등 무거운 데이터를 초고속으로 암호화 통신합니다.

# Chapter 3 완료

가장 강력한 웹 프록시 도구 Burp Suite의 활용법부터,
실제 인가 로직 파괴 실습, 그리고 암호학의 정수까지 체득하셨습니다.

**수고하셨습니다. 다음 과정에서 뵙겠습니다!**

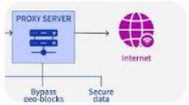
# Image Sources




https://media.geeksforgeeks.org/wp-content/uploads/20250707160103024541/http_request.webp

Source: www.geeksforgeeks.org

---




https://media.geeksforgeeks.org/wp-content/uploads/20250804163627434033/proxy_server.webp

Source: www.geeksforgeeks.org

---

https://aptien.com/uploads/kb_block_image/what-is-cia-triad/detail_information-security-cia-triad.png

Source: aptien.com

---

https://cdn.prod.website-files.com/5ff66329429d880392f6cba2/6348192f8e2b1916df69dc17_464.3-min.jpg

Source: www.wallarm.com