

수업 내용 정리 보고서

작성일: 2026년 7월 16일

챕터 구성

- Snort 룰셋 설계 및 엔진별 문법 가이드
- 클라이언트 사이드 보안 로직 우회 및 취약점 분석
- 세션 관리 취약점과 세션 하이재킹 원리
- 웹 및 네트워크 보안 위협 대응 점검 가이드

Snort 룰셋 설계 및 엔진별 문법 가이드

침입 탐지 시스템(IDS)인 Snort의 핵심은 네트워크 트래픽을 분석하여 공격 패턴을 찾아내는 **룰셋(Rule Set)** 설계에 있습니다. 룰셋은 단순히 문자열을 찾는 것을 넘어, 네트워크의 방향, 세션의 상태, 특정 프로토콜의 버퍼 영역 등을 정교하게 지정하여 오탐(False Positive)을 줄이고 탐지 정밀도를 높이는 방식으로 설계됩니다.

1. Snort 룰의 기본 구조

Snort 룰은 크게 **룰 헤더(Rule Header)**와 **룰 옵션(Rule Options)**으로 구성됩니다. 헤더는 "어떤 트래픽을 검사할 것인가"를 정의하고, 옵션은 "어떤 조건일 때 탐지할 것인가"를 정의합니다.

```
alert http EXTERNAL_NET any -> HOME_NET any (
  msg:"HTTP POST Login";
  flow:established,to_server;
  http_method;
  content:"POST";
  sid:2000001;
  rev:1;
)
```

구분	역할	상세 설명 및 예시
Action	매칭 시 동작	alert(경고 발생), drop(패킷 차단), pass(허용)
Protocol	검사 대상 프로토콜	tcp, udp, icmp 및 전용 프로토콜인 http 등
Source / Destination	출발지 및 목적지	EXTERNAL_NET any(외부망 모든 포트) → HOME_NET 80(내부망 80번 포트)
Options	세부 탐지 조건	괄호 () 안에 정의하며 content, flow, sid 등이 포함됨

2. 네트워크 변수 정의 및 설정

룰셋의 효율적인 관리를 위해 IP 대역을 변수로 지정하여 사용합니다. 특히 보호 대상인 내부망(HOME_NET)과 외부망(EXTERNAL_NET)의 구분이 중요합니다.

- **HOME_NET**: 보호하려는 내부 네트워크, 서버망 또는 DMZ 대역을 의미합니다.
- **EXTERNAL_NET**: 외부 인터넷 영역을 의미하며, 일반적으로 !\$HOME_NET(내부망이 아닌 모든 곳)으로 설정하여 명확히 구분합니다.

엔진 버전별로 설정 파일의 경로와 문법이 다르므로 주의가 필요합니다.

- **Snort 3**: /usr/local/snort/etc/snort/snort.lua 파일에서 Lua 문법을 사용합니다.
(예: HOME_NET = '6.10.64.0/24')
- **Snort 2 / Security Onion**: /etc/nsm/[센서명-인터페이스]/snort.conf 파일에서 ipvar 명령어를 사용합니다.
(예: ipvar HOME_NET 6.10.64.0/24)

3. 탐지 정밀도를 높이는 옵션 설계

3.1 Content 및 Sticky Buffer

content 옵션은 패킷 내에서 특정 문자열이나 16진수(Hex) 패턴을 검색합니다. 하지만 전체 패킷을 모두 검사하면 성능이 저하되고 오탐이 발생할 수 있으므로, **Sticky Buffer**를 사용하여 검사 영역을 제한합니다.

- **문자열 표현**: content:"POST"; 또는 16진수 content:"|50 4F 53 54|"; (두 표현은 동일한 바이트를 탐지함)
- **주요 Sticky Buffer**:
 - http_method: GET, POST 등 HTTP 메서드 영역만 검사
 - http_uri: 요청 URI 영역만 검사
 - http_header: HTTP 헤더 영역만 검사
 - http_client_body: POST 요청의 본문 영역만 검사
 - http_stat_code: 서버 응답 코드(200, 404 등) 영역만 검사

3.2 Flow 및 Flowbits (세션 및 상태 추적)

단일 패킷이 아닌 TCP 세션 전체의 흐름을 분석하여 공격을 탐지합니다.

- **flow**: 세션의 성립 상태와 방향을 지정합니다.
 - established: TCP 3-Way Handshake가 완료된 세션만 검사
 - to_server: 클라이언트 → 서버 방향 트래픽

- `to_client`: 서버 → 클라이언트 방향 트래픽
- **flowbits**: 동일 세션 내에서 이전 패킷의 상태를 기억하여 연쇄적인 탐지를 수행합니다.
 - `set`: 특정 상태(비트)를 저장 (예: 로그인 시도 확인)
 - `isset`: 저장된 상태가 있는지 확인 (예: 이전 단계에서 로그인 시도가 있었는가?)
 - `unset`: 저장된 상태를 제거
 - `noalert`: 상태만 저장하고 실제 경고(Alert)는 발생시키지 않음

3.3 세부 제어 옵션 (Modifier)

문자열 검색의 범위를 더 세밀하게 조정하여 탐지 정확도를 높입니다.

옵션	기준점	설명
<code>offset / depth</code>	버퍼 시작점	시작점에서 몇 바이트 뒤부터(<code>offset</code>), 최대 몇 바이트까지(<code>depth</code>) 검사할지 지정
<code>distance / within</code>	이전 매칭 지점	이전 content가 끝난 지점에서 몇 바이트 뒤부터(<code>distance</code>), 몇 바이트 이내에(<code>within</code>) 다음 패턴이 있는지 검사
<code>nocase</code>	-	영문 대소문자를 구분하지 않음
<code>pcre</code>	-	정규표현식을 사용하여 복잡한 패턴 검사

4. 관리 옵션 및 식별자

운영 단계에서 룰을 효율적으로 관리하기 위해 다음과 같은 관리 옵션을 필수적으로 포함합니다.

- **msg**: Alert 발생 시 표시될 설명 (공격 명칭 및 내용 작성)
- **sid (Snort ID)**: 룰의 고유 식별 번호. 중복되지 않게 관리하며, 보통 사용자 정의 룰은 2000000번대 이상을 사용합니다.
- **rev (Revision)**: 룰 수정 버전. 내용 변경 시 1씩 증가시켜 이력을 관리합니다.
- **classtype**: 공격 유형 분류 (예: `web-application-activity`)

5. 엔진별 문법 비교 (Snort 2 vs Snort 3 vs Suricata)

가장 흔한 실수는 버전별로 HTTP 버퍼 지정 순서가 다른 점을 간과하는 것입니다. 특히 Snort 3와 Suricata는 **Sticky Buffer**를 먼저 선언하는 구조를 가집니다.

구분	Snort 2.9	Snort 3	Suricata
문법 구조	content 먼저 → modifier 나중	Sticky Buffer → content	Sticky Buffer → content
버퍼 표기법	<code>http_uri</code> (언더바)	<code>http_uri</code> (언더바)	<code>http.uri</code> (점 표기법)
예시 (URI 탐지)	<code>content: "/login"; http_uri;</code>	<code>http_uri; content: "/login";</code>	<code>http.uri; content: "/login";</code>
예시 (Body 탐지)	<code>content: "user"; http_client_body;</code>	<code>http_client_body; content: "user";</code>	<code>http.request_body; content: "user";</code>

[학습 요약 및 점검 체크리스트]

- **설계 순서**: 대상 네트워크 지정(\$HOME_NET) → 세션 상태 정의(flow) → 검사 영역 지정(Sticky Buffer) → 핵심 패턴 정의(content) → 관리 정보 부여(sid, rev) 순으로 설계했는가?
- **오탐 방지**: 전체 패킷 검색 대신 `http_uri`나 `http_method` 같은 전용 버퍼를 사용하여 검색 범위를 최소화했는가?
- **상태 추적**: 단순 일회성 패킷이 아닌, 특정 순서(로그인 → 실패 응답)로 일어나는 공격을 탐지하기 위해 **flowbits**를 적절히 활용했는가?
- **버전 확인**: 현재 사용 중인 엔진이 Snort 2인지 3인지, 혹은 Suricata인지 확인하고 그에 맞는 버퍼 선언 순서(content 우선 vs 버퍼 우선)를 적용했는가?

클라이언트 사이드 보안 로직 우회 및 취약점 분석

웹 보안의 핵심 원칙 중 하나는 **"클라이언트는 신뢰할 수 없다"**는 것입니다. 사용자의 브라우저(클라이언트 사이드)에서 수행되는 모든 검증, 인증, 데이터 가공 로직은 사용자가 개발자 도구 등을 통해 얼마든지 조작할 수 있기 때문입니다. 본 절에서는 브라우저 내부에서 수행되는 보안 로직을 우회하는 방법과, 예측 가능한 세션 ID를 이용한 세션 하이재킹의 원리를 실습을 통해 분석합니다.

1. JavaScript Attack: 클라이언트 사이드 로직 우회

JavaScript Attack은 웹 브라우저 단에서 수행되는 난독화(Obfuscation), 인증(Authentication), 혹은 입력값 검증(Validation) 로직이 얼마나 취약한지를 이용하는 공격입니다. 특히 보안에 중요한 데이터(예: 인증 토큰)를 서버가 아닌 클라이언트가 직접 생성하여 서버로 전송하도록 설계된 경우, 공격자는 이 과정을 역추적하여 위조된 데이터를 보낼 수 있습니다.

1.1 취약점 분석 및 작동 원리

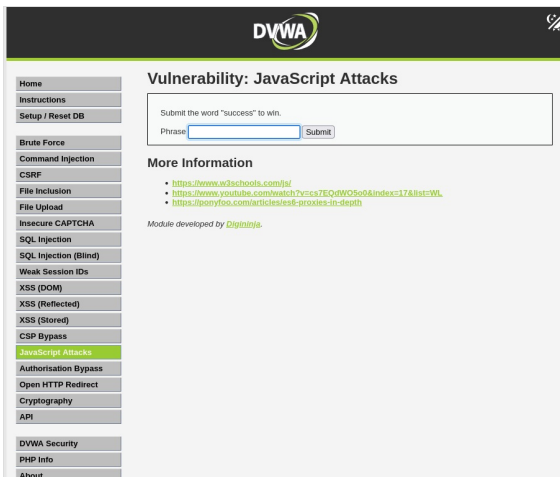
실습 대상인 DVWA의 JavaScript Attacks 항목은 사용자가 입력한 문자열을 특정 방식으로 변조한 뒤, 그 결과값(Hash)을 토큰으로 사용하여 서버에 제출하는 구조를 가지고 있습니다. 브라우저 내부의 dvwaPage.js 파일을 분석하면 다음과 같은 핵심 로직을 확인할 수 있습니다.

- **입력값 획득:** document.getElementById("phrase").value 명령을 통해 사용자가 입력창에 타이핑한 텍스트를 가져옵니다.
- **데이터 변조(Obfuscation):** 가져온 문자열을 rot13() 함수를 통해 알파벳을 13칸씩 밀어내는 방식으로 꼬아버립니다.
- **해시 생성:** 변조된 문자열을 다시 md5() 함수를 사용하여 32자리의 고유한 해시값으로 변환합니다.
- **토큰 주입:** 최종 생성된 해시값을 화면에 숨겨진 token 입력창에 강제로 삽입합니다.

이 과정은 페이지가 처음 로딩될 때 generate_token() 함수에 의해 1회 자동으로 실행됩니다. 하지만 여기서 치명적인 **설계 결함**이 발견됩니다. 사용자가 입력창의 내용을 변경하더라도, 새로운 값에 대해 토큰을 다시 계산해주는 **이벤트 리스너(Event Listener)**가 구현되어 있지 않다는 점입니다. 즉, 사용자가 글자를 바꾸어도 토큰은 페이지 로딩 시점의 옛날 값을 유지하게 됩니다.

먼저 실습을 위해 DVWA의 JavaScript Attacks 페이지에 접속합니다. 화면에는 정답인 "success"를 입력하라는 안내와 함께 입력창(Phrase)이 나타납니다.

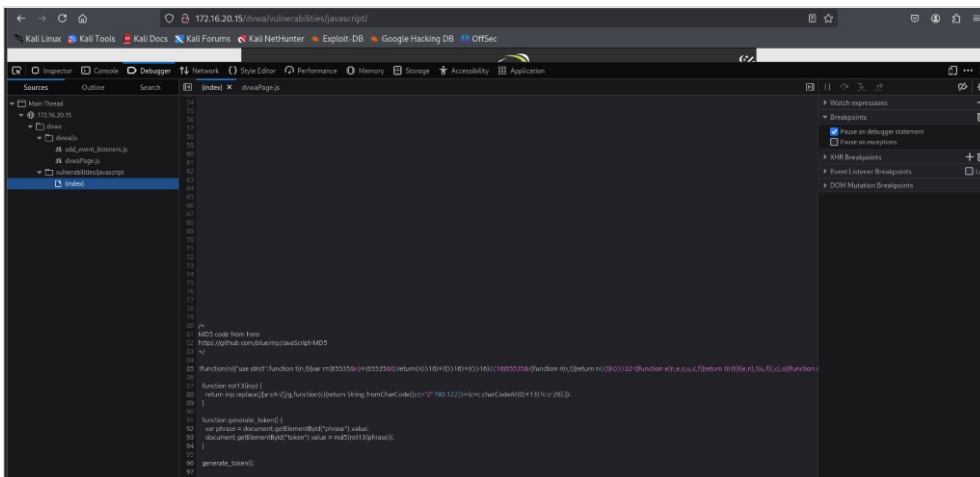
"Vulnerability: JavaScript Attacks" 제목과 Phrase 입력 필드



DVWA의 JavaScript Attacks 실습을 위한 입력창과 제출 버튼이 있는 웹 페이지 화면

공격자는 개발자 도구(F12)의 소스(Sources) 탭을 통해 자바스크립트 내부 로직을 파헤칩니다. dvwaPage.js 파일의 91~96번 라인에서 토큰이 생성되는 구체적인 수식을 확인할 수 있습니다.

`generate\_token()` 함수와 `md5`, `rot13` 호출 코드



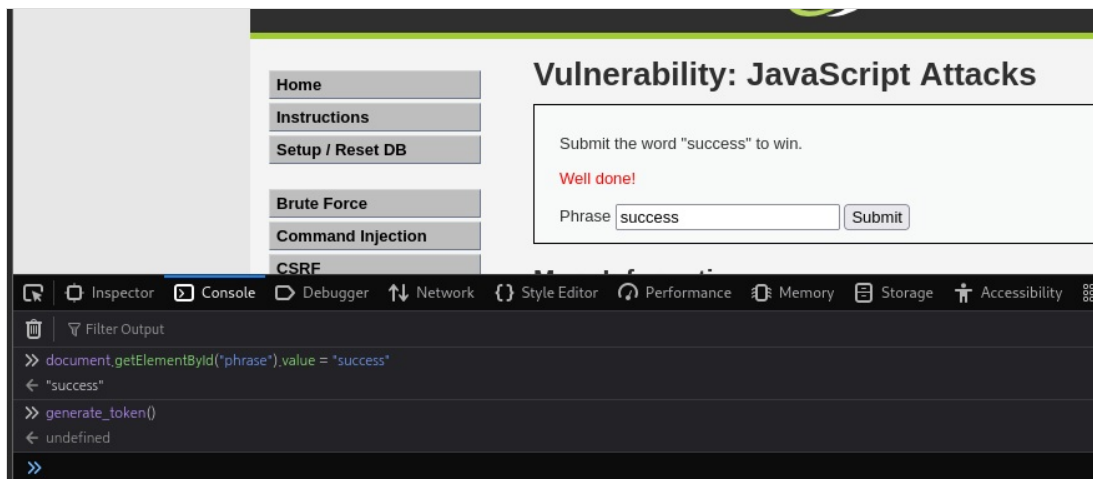
개발자 도구 소스 탭에서 토큰 생성 로직이 포함된 dvwaPage.js 소스 코드

이제 이 결함을 이용하여 우회 공격을 수행합니다. 개발자 도구의 콘솔(Console) 탭을 열어, 입력창의 값을 "success"로 강제 대입하고, 누락된 토큰 갱신 함수를 수동으로 실행시킵니다.

```
document.getElementById("phrase").value = "success";
generate_token();
```

콘솔에서 위 명령어를 입력하면, 입력창의 값은 "success"로 바뀌고, 동시에 숨겨진 토큰 칸에는 "success"의 rot13 및 MD5 값이 올바르게 갱신됩니다. 이후 제출(Submmit) 버튼을 누르면 서버는 클라이언트가 보낸 토큰이 유효하다고 판단하여 성공 메시지를 출력합니다.

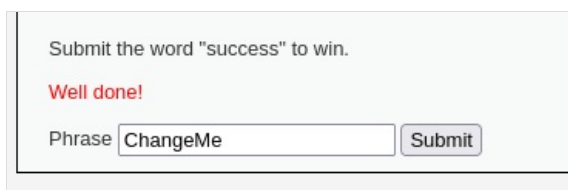
`document.getElementById("phrase").value = "success"` 명령어 입력



개발자 도구 콘솔에서 입력값을 수정하고 토큰 생성 함수를 수동으로 실행하는 과정

콘솔 조작 후 페이지에 "Well done!"이라는 메시지가 출력되면 클라이언트 사이드 로직 우회에 성공한 것입니다.

빨간색 "Well done!" 문구 출력



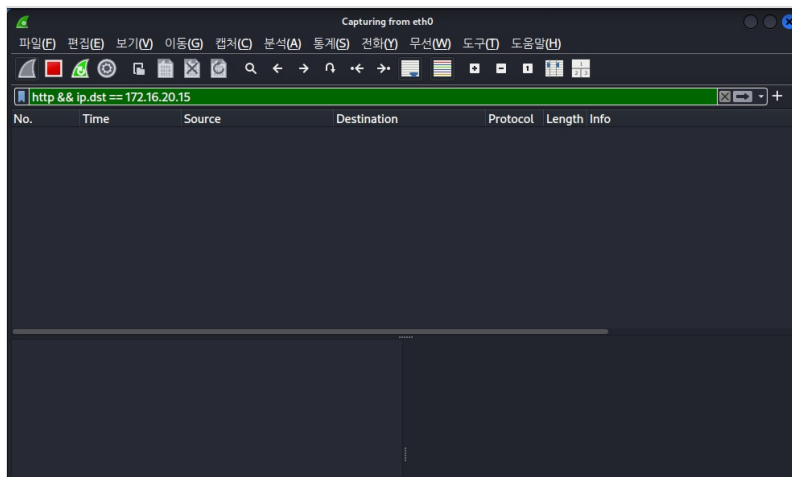
콘솔 조작 후 페이지에 Well done! 메시지가 출력되며 공격 성공을 확인

1.2 공격 자동화 및 패킷 분석

해커는 이러한 과정을 수동으로 하지 않고 파이썬(Python) 스크립트를 통해 자동화할 수 있습니다. 자동화 코드는 서버의 CSRF 토큰을 먼저 수집한 뒤, 로직을 역추적하여 위조된 토큰을 생성하고 서버로 전송합니다. 이 과정에서 와이어샤크(Wireshark)와 같은 패킷 분석 도구를 사용하여 공격이 성공하는지 확인합니다.

먼저 와이어샤크에서 공격 대상 서버의 IP와 HTTP 프로토콜만을 필터링하여 캡처를 준비합니다.

`http && ip.dst == 172.16.20.15` 필터링 조건



와이어샤크에서 특정 목적지 IP와 HTTP 프로토콜을 대상으로 하는 패킷 캡처 필터 설정

파이썬 스크립트를 실행하면, 로그에 단계별 진행 상황이 출력됩니다. 스크립트는 1단계에서 토큰을 수집하고, 5단계에서 "success"에 대응하는 위조된 토큰을 계산하며, 최종적으로 서버에 패킷을 전송합니다.

터미널의 "공격 성공: 서버로부터 'Well done!' 응답확인" 메시지

```

choi@choi: ~
세션 동작 편집하기 보기 도움말
zsh: locking failed for /home/choi/.zsh_history: 읽기 전용 파일 시스템: readin
g anyway
(choi@choi)-[~]
$ sudo python3 /home/choi/java.py
[sudo] choi 암호:
[*] 1단계: 토끼 리눅스 DVWA 서버 접속 및 CSRF 토큰 수집 중 ...
[*] CSRF 토큰 획득 완료: 294223eb5c58967bfbfa22eb4e630fe6
[*] 2단계: 자동화 공격 인증 시도 중 ...
[*] 로그인 인증 성공
[*] 타겟 페이지 보안 단계를 'Low'로 가감 완료.
[*] 5단계: 우회 페이로드(Payload) 연산 성공:
- 전송할 문(pharse) : success
- 위조된 토큰(token) : 38581812b435834ebf84ebcc2c6424d6
[*] 6단계: 브라우저를 우회하여 조작된 최종 패킷 전송 중 ...

공격 성공: 서버로부터 'Well done!' 응답 확인.

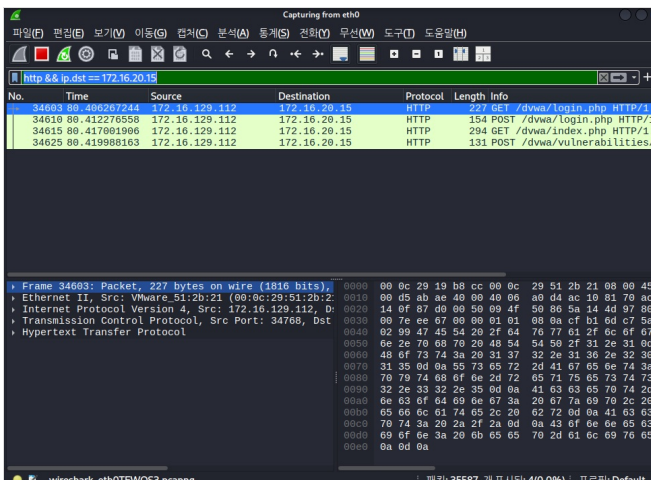
(choi@choi)-[~]
$

```

파이썬 자동화 스크립트를 실행하여 위조된 토큰을 전송하고 성공 응답을 받는 과정

전송된 패킷을 와이어샤크로 확인하면, 공격자가 의도한 대로 HTTP 트래픽이 발생하는 것을 볼 수 있습니다.

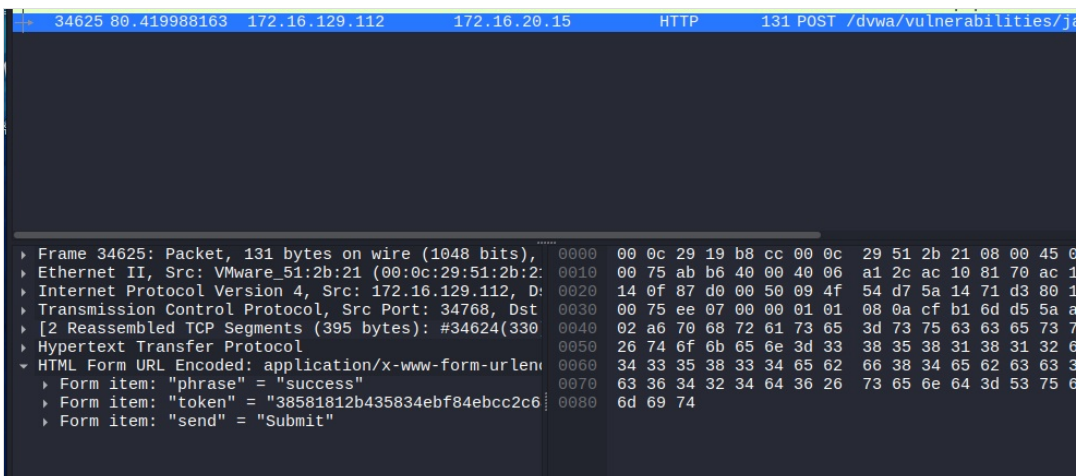
POST /dvwa/vulnerabilities/를 포함한 패킷 리스트



자동화 공격 수행 시 발생한 HTTP POST 요청을 포함한 네트워크 패킷 목록

가장 중요한 검증 단계는 캡처된 패킷의 상세 내용을 분석하는 것입니다. HTTP POST 요청의 바디(Body) 부분을 확인하면, 사용자가 입력한 phrase="success"와 스크립트가 계산해낸 token="38581812..." 값이 정확히 포함되어 있음을 알 수 있습니다.

Form item: "phrase" = "success" 및 토큰 값 상세 정보



캡처된 패킷 상세 분석을 통해 조작된 phrase와 위조된 token 값이 전송됨을 확인

이 실습을 통해 데이터 검증 로직이 클라이언트에 있을 경우, 공격자가 로직을 투명하게 파악하고 위조된 값을 전송함으로써 보안을 완전히 무력화할 수 있음을 확인할 수 있습니다.

2. Weak Session IDs: 예측 가능한 세션 관리

세션(Session)은 사용자가 로그인을 유지할 수 있도록 서버가 브라우저에 부여하는 일종의 "임시 통행증"입니다. 이 통행증의 번호인 세션 ID는 타인이 추측할 수

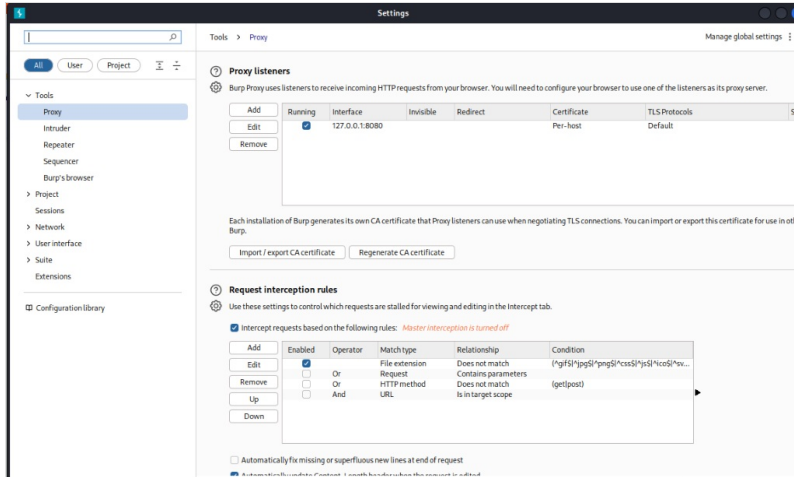
없도록 매우 복잡하고 무작위(Random)해야 합니다. 만약 세션 ID가 일정한 규칙을 가지거나 너무 짧다면, 공격자는 이를 예측하여 다른 사용자의 계정을 탈취하는 세션 하이재킹(Session Hijacking) 공격을 수행할 수 있습니다.

2.1 취약점 확인 및 실습 절차

DVWA의 Weak Session IDs 페이지는 버튼을 클릭할 때마다 새로운 세션 ID를 발급합니다. 이 과정에서 발급되는 ID의 규칙성을 확인하기 위해 Burp Suite를 이용한 프록시(Proxy) 설정이 필요합니다.

먼저 Burp Suite를 실행하고, 프록시 리스너가 127.0.0.1:8080 포트에서 정상적으로 동작하고 있는지 확인합니다.

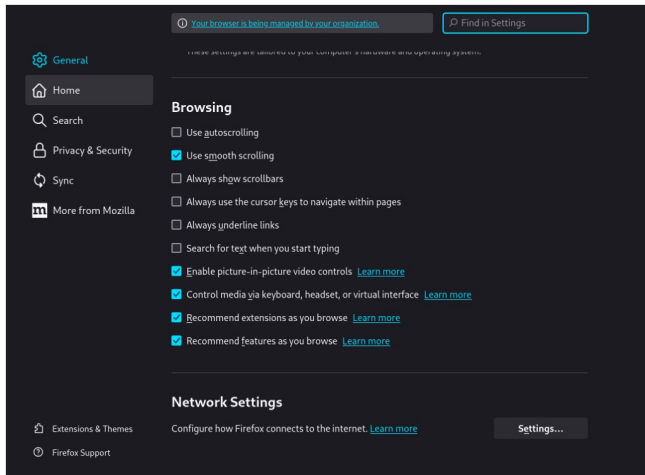
"Proxy listeners" 항목의 "127.0.0.1:8080" 및 "Running" 표시



Burp Suite 프록시 리스너가 127.0.0.1:8080 포트에서 동작 중인 상태

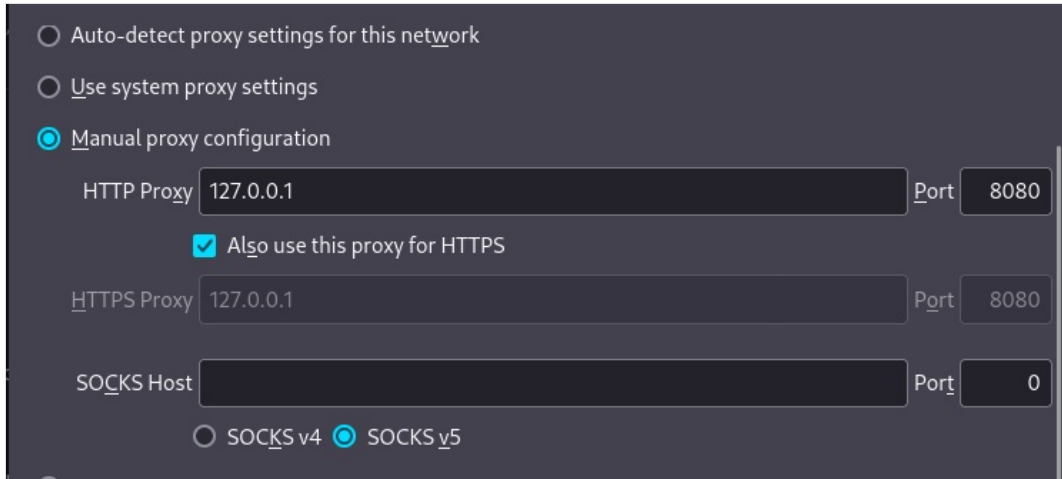
브라우저(Firefox)에서 Burp Suite의 패킷을 받아올 수 있도록 수동 프록시 설정을 진행합니다. 브라우저의 메뉴를 통해 네트워크 설정으로 진입합니다.

Firefox 메뉴 구성 항목



Firefox 브라우저의 주요 메뉴 및 관리 옵션 목록

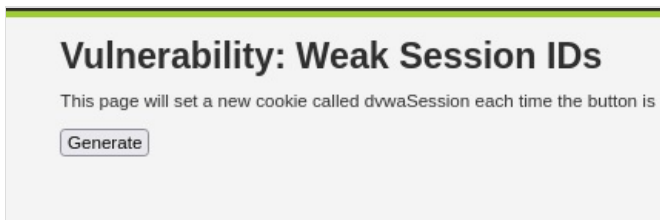
"Network Settings" 헤더 및 설정 버튼



Firefox 설정 내 네트워크 설정(Network Settings) 섹션

설정에서 **Manual proxy configuration**을 선택하고, HTTP 프록시 주소를 **127.0.0.1**, 포트를 **8080**으로 지정한 뒤, HTTPS 통신도 동일한 프록시를 사용하도록 체크합니다.

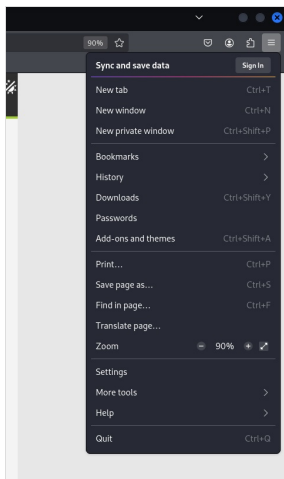
"Manual proxy configuration" 및 "127.0.0.1:8080" 입력값



HTTP 프록시를 127.0.0.1:8080으로 지정한 수동 프록시 설정 값

설정이 완료되면 Burp Suite의 **Intercept** 기능을 활성화하여 브라우저에서 나가는 패킷을 가로챌 준비를 합니다.

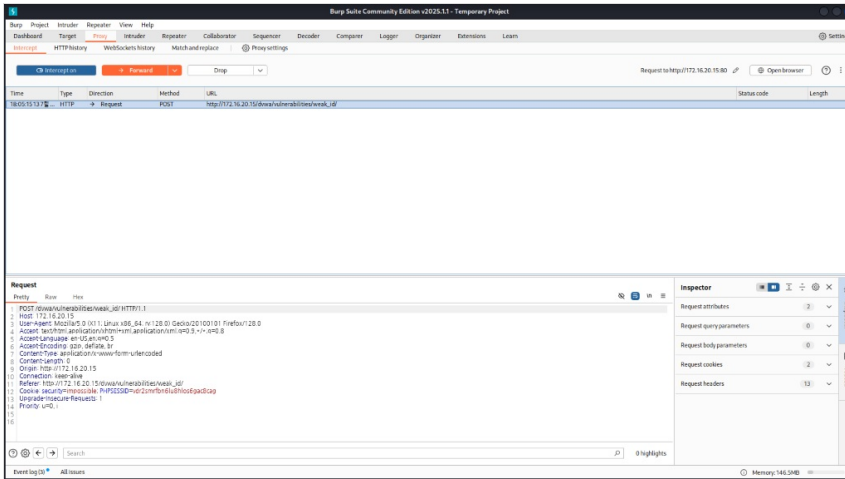
"Intercept on" 버튼 및 Intercept 탭 UI



Burp Suite Intercept 탭의 패킷 가로채기 활성화 버튼

이제 DVWA 페이지로 돌아가 세션 생성(Generate) 버튼을 클릭합니다. Burp Suite에 패킷이 잡히면, 해당 요청의 상세 내용을 분석합니다.

URL이 `/dwva/vulnerabilities/weak_id/`이며 PHPSESSID 쿠키가 포함된 요청이 대기 중임



Burp Suite Intercept 기능을 통해 캡처된 weak_id 관련 POST 요청 상세

여러 번 버튼을 클릭하여 생성된 연속적인 요청들을 HTTP history 탭에서 확인합니다. 각 요청이 서버에 성공적으로 전달되었는지 목록을 검토합니다.

HTTP history 탭 내에 172.16.20.15 서버를 대상으로 한 GET 및 POST 요청이 나열됨

Burp Project Intruder Repeater View Help						
Dashboard Target Proxy Intruder Repeater Collaborator Sequencer Decoder C						
Intercept HTTP history WebSockets history Match and replace Proxy settings						
Filter settings: Hiding CSS, image and general binary content						
#	Host	Method	URL	Params	Edited	Status code
1	http://detectportal.firefox.com	GET	/success.txt?ipv6	✓		200
2	http://detectportal.firefox.com	GET	/success.txt?ipv4	✓		200
3	http://detectportal.firefox.com	GET	/success.txt?ipv6	✓		200
4	http://detectportal.firefox.com	GET	/success.txt?ipv4	✓		200
5	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200
6	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200
7	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200
8	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200
9	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200
10	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200
11	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200
12	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200
13	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200

Burp Suite HTTP history 탭에 기록된 대상 서버로의 연속적인 요청 목록

마지막으로 각 요청에 포함된 쿠키(Cookie) 값을 정밀 분석합니다. 확인 결과, dwvaSession 값이 1, 2, 3과 같이 매우 단순한 숫자로 순차적으로 증가하는 것을 발견할 수 있습니다.

Cookies 열에 dwvaSession=1, dwvaSession=2, dwvaSession=3이 연속적으로 확인됨

#	Host	Method	URL	Params	Edited	Status code	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response t
26	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:10.137...	8080	2	
27	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:24.137...	8080	2	
28	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:38.137...	8080	4	
29	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:52.137...	8080	13	
30	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:46.137...	8080	2	
31	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:40.137...	8080	2	
32	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4290	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:40.137...	8080	2	
33	http://172.16.20.15	GET	/dwa/vulnerabilities/weak_id/			200	4291	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:40.137...	8080	4	
34	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200	4318	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:08:50.137...	8080	3	
35	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200	4318	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:10:52.137...	8080	2	
36	http://172.16.20.15	POST	/dwa/vulnerabilities/weak_id/			200	4318	HTML		Vulnerability: Weak Sess...			172.16.20.15	18:11:01.137...	8080	3	
1	http://detectportal.firefox.com	GET	/success.txt?ipv6	✓		200	371	text	txt				193.107.193.91	18:09:50.137...	8080	3	
2	http://detectportal.firefox.com	GET	/success.txt?ipv4	✓		200	371	text	txt				193.107.193.91	18:09:50.137...	8080	4	

Request	Raw	Hex
1 POST /dwa/vulnerabilities/weak_id/ HTTP/1.1		
2 Host: 172.16.20.15		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:128.0) Gecko/20100101 Firefox/128.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate, br		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 0		
9 Origin: http://172.16.20.15		
10 Connection: keep-alive		
11 Referer: http://172.16.20.15/dwa/vulnerabilities/weak_id/		
12 Cookie: secure=1; PHPSESSID=2cmrbndubln0spadcap		
13 Upgrade-Insecure-Requests: 1		
14 Priority: u=0, i		
15		
16		

Response	Raw	Hex	Render
1 HTTP/1.1 200 OK			
2 Date: Mon, 13 Jul 2025 09:10:38 GMT			
3 Server: Apache/2.4.61 (Ubuntu)			
4 X-Powered-By: PHP/8.0.30			
5 Pragma: no-cache			
6 Cache-Control: no-cache, must-revalidate			
7 Expires: Tue, 23 Jun 2029 12:00:00 GMT			
8 Set-Cookie: dwvasession=1			
9 Keep-Alive: timeout=5, max=100			
10 Connection: keep-alive			
11 Content-Type: text/html; charset=utf-8			
12 Content-Length: 3553			
13			
14 (DOCTYPE html)			
15			
16 <html lang="en-GB">			
17			
18 <head>			
19 <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">			
20			
21 </head>			
22 Vulnerability: Weak Session (Ds : Damn Vulnerable Web Application (DVWA))			
23 <div>			
24 <div rel="noopener" base="text/css" href="/_dwa/css/main.css">			
25			
26 <div rel="noopener" base="img/ico" href="/_dwa/ico">			
27 <script base="text/javascript" src="/_dwa/js/dwvaf.js">			

HTTP history 내 Cookies 항목에서 dwvaSession 값이 1, 2, 3으로 순차 증가하는 양상

이러한 순차적 세션 발급은 공격자가 브루트포스(Brute-force)나 단순 예측만으로도 유효한 세션 ID를 알아낼 수 있게 만들며, 이는 곧 서비스 전체의 권한 탈취로 이어지는 매우 심각한 취약점입니다.

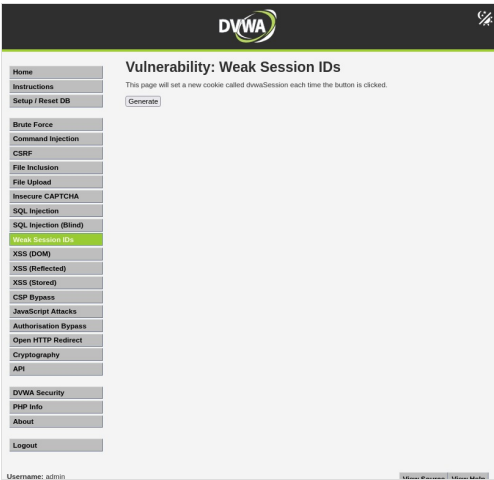
실무 및 시험 핵심 포인트

구분	점검 항목 및 판단 기준	대응 방안 (Best Practice)
클라이언트 로직	중요 데이터(토큰, 인증값)의 생성/검증이 브라우저에서 이루어지는가?	모든 보안 검증 및 데이터 가공은 반드시 서버 사이드 에서 수행해야 함.
이벤트 처리	사용자 입력 변경 시 보안 관련 변수가 실시간으로 갱신되는가?	입력값 변경 시 토큰을 재계산하는 EventListener 를 반드시 구현함.
세션 관리	세션 ID가 예측 가능한 패턴(순차적 숫자, 날짜 등)을 가지는가?	암호학적으로 안전한 CSPRNG 를 사용하여 충분히 긴 무작위 값을 생성함.
세션 탈취 방지	세션 ID가 노출되거나 추측하기 쉬운가?	세션 ID의 길이를 늘리고, HttpOnly 및 Secure 플래그를 설정함.

주의사항: 클라이언트 사이드 보안 로직을 우회하는 실습은 반드시 격리된 실습 환경(DVWA 등)에서 수행해야 하며, 실제 운영 중인 서비스에 시도하는 것은 불법적인 행위입니다.

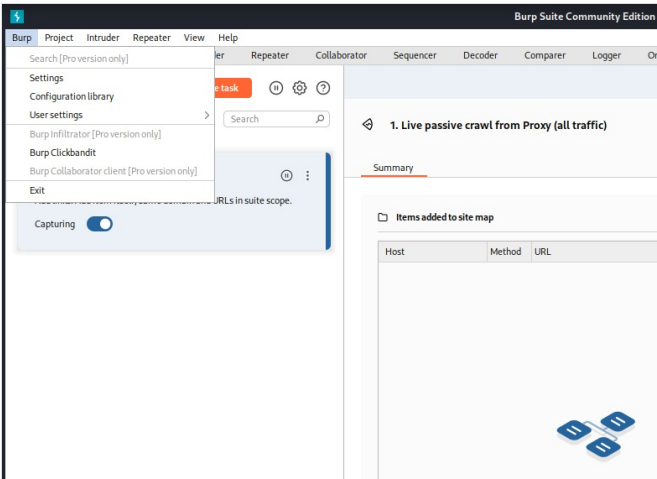
추가 실습 화면

"Vulnerability: Weak Session IDs" 문구와 "Generate" 버튼



DVWA의 Weak Session IDs 취약점 테스트를 위한 세션 생성 버튼 화면

"Live passive crawl from Proxy" 문구 및 사이트 맵 영역



Burp Suite Dashboard 화면

세션 관리 취약점과 세션 하이재킹 원리

웹 애플리케이션 보안에서 가장 중요한 요소 중 하나는 **세션 관리(Session Management)**입니다. 사용자가 웹사이트에 로그인하면 서버는 해당 사용자를 식별하기 위한 고유한 번호표를 발급하는데, 이것이 바로 **세션 ID(Session ID)**입니다. 서버는 이 ID를 브라우저의 **쿠키(Cookie)**에 저장하도록 지시하며, 이후 브라우저는 매 요청마다 이 쿠키를 서버에 전송하여 자신이 로그인된 상태임을 증명합니다.

만약 이 세션 ID가 보안 요구사항을 충족하지 못할 경우, 공격자는 타인의 세션을 가로채는 **세션 하이재킹(Session Hijacking)** 공격을 수행할 수 있습니다. 본 장에서는 취약한 세션 ID가 무엇인지 정의하고, 실제 실습을 통해 어떻게 그 규칙성을 찾아내는지 학습합니다.

1. 취약한 세션 ID (Weak Session IDs)의 정의

세션 ID는 암호로서 충분한 복잡성을 가져야 하며, 공격자가 이를 추측할 수 없도록 설계되어야 합니다. 다음과 같은 특성을 가질 경우 **Weak Session IDs**라고 판단합니다.

- **규칙성(Pattern):** 발급되는 ID에 일정한 패턴이 존재하는 경우
- **예측 가능성(Predictability):** 시간, 사용자 ID 등 예측 가능한 값을 기반으로 생성되는 경우
- **낮은 복잡성 및 짧은 길이:** 무작위 대입 공격(Brute-force)에 취약할 정도로 길이가 짧거나 무작위성이 부족한 경우

공격자가 이러한 규칙을 파악하면, 아이디와 비밀번호를 몰라도 유효한 세션 ID를 직접 생성하거나 예측하여 다른 사용자의 권한을 탈취할 수 있습니다.

2. 실습: 세션 ID의 규칙성 분석 및 취약점 확인

DVWA(Damn Vulnerable Web Application)의 **Weak Session IDs** 모듈을 사용하여, 서버가 발급하는 세션 ID가 얼마나 취약한지 직접 확인해 보겠습니다. 이 실습에서는 Burp Suite를 사용하여 브라우저와 서버 사이의 트래픽을 가로채고 분석합니다.

단계 1: 실습 환경 준비 및 대상 확인

먼저 DVWA에 접속하여 Weak Session IDs 페이지로 이동합니다. 이 페이지는 'Generate' 버튼을 클릭할 때마다 새로운 세션 ID를 생성하여 쿠키에 저장하는 기능을 제공합니다.

세션 ID의 규칙성을 확인하기 위해서는 브라우저의 모든 HTTP 요청을 가로채서 살펴볼 수 있는 프록시(Proxy) 환경을 구축해야 합니다.

단계 2: 브라우저 프록시(Proxy) 설정

Firefox 브라우저의 트래픽이 Burp Suite로 전달되도록 수동 프록시 설정을 진행합니다.

1. Firefox 브라우저의 메뉴를 통해 설정 창으로 진입합니다.
2. 설정 내 **Network Settings** 섹션을 찾습니다.
3. **Manual proxy configuration**을 선택하고, 주소에 **127.0.0.1**, 포트에 **8080**을 입력합니다. 또한 **"Also use this proxy for HTTPS"** 체크박스를 반드시 선택해야 합니다.

단계 3: Burp Suite 리스너 및 인터셉트 활성화

이제 Burp Suite가 브라우저로부터 오는 신호를 받을 준비가 되었는지 확인합니다. 설정한 포트(8080)에서 리스너가 정상적으로 동작하고 있어야 합니다.

패킷을 실시간으로 가로채기 위해 **Intercept** 기능을 활성화합니다.

단계 4: 세션 생성 및 패킷 분석

설정이 완료되었다면, DVWA 페이지로 돌아가 'Generate' 버튼을 클릭합니다. Burp Suite에 패킷이 잡히면 내용을 확인하고 **Forward**를 눌러 통과시킵니다.

이제 연속적으로 버튼을 클릭하여 생성되는 세션 ID들의 패턴을 분석하기 위해 **HTTP history** 탭으로 이동합니다. 여러 번의 요청 내역이 기록되어 있는 것을 확인할 수 있습니다.

가장 중요한 단계입니다. 기록된 각 요청의 **Cookies** 항목을 상세히 분석하여 **dwvaSession** 값이 어떻게 변하는지 확인합니다. 실습 결과, 세션 ID가 **1, 2, 3**과 같이 단순한 숫자로 순차적으로 증가하는 것을 발견했습니다.

이 결과는 해당 웹 애플리케이션이 매우 취약한 세션 관리 메커니즘을 가지고 있음을 입증합니다. 공격자는 다음 세션 ID가 무엇일지 완벽하게 예측할 수 있으므로, 매우 쉽게 세션 하이재킹 공격을 성공시킬 수 있습니다.

3. 결론 및 보안 점검 포인트

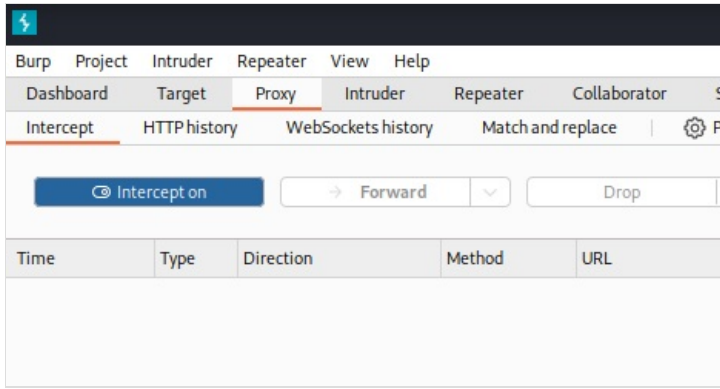
세션 하이재킹은 사용자 인증 과정을 우회하여 권한을 탈취하는 치명적인 공격입니다. 이를 방지하기 위해 개발 및 운영 단계에서 다음과 같은 점검이 필요합니다.

점검 항목	상세 내용	보안 권고 사항
ID의 무작위성	세션 ID 생성 시 예측 가능한 값(순차적 숫자, 시간 등) 사용 여부	Cryptographically Secure Pseudo-Random Number Generator (CSPRNG)를 사용하여 충분한 엔트로피를 가진 무작위 값을 생성해야 함
ID의 복잡성 및 길이	세션 ID가 브루트포스 공격에 견딜 만큼 충분히 긴지 여부	충분한 길이(최소 128비트 이상의 엔트로피)를 확보하여 무작위 대입 공격을 무력화해야 함
전송 보안	세션 ID가 평문으로 노출되는지 여부	반드시 HTTPS(TLS)를 사용하여 세션 쿠키가 네트워크 상에서 탈취되지 않도록 보호해야 함
쿠키 속성 설정	쿠키의 보안 속성 적용 여부	HttpOnly(자바스크립트 접근 차단) 및 Secure(HTTPS 전송 강제) 속성을 반드시 설정해야 함

실무 포인트: 세션 관리 취약점은 코드 한 줄의 실수로 발생할 수 있습니다. 세션 ID 생성 로직이 서버 측에서 안전하게 제어되고 있는지, 그리고 클라이언트(브라우저) 측에서 세션 정보를 조작하거나 예측할 수 있는 단서가 노출되지 않는지 반드시 확인해야 합니다.

추가 실습 화면

"Request interception rules" 표 및 활성화 체크박스



Burp Suite의 Request interception 규칙 설정 화면

웹 및 네트워크 보안 위협 대응 점검 가이드

본 가이드는 네트워크 침입 탐지 시스템(IDS)의 핵심인 Snort 룰셋 설계 방법부터, 클라이언트 사이드(Client-side)의 자바스크립트 로직 우회, 그리고 세션 관리 취약점을 이용한 세션 하이재킹까지의 전 과정을 다룹니다. 공격자의 관점에서 취약점을 분석하고, 방어자의 관점에서 이를 어떻게 탐지하고 차단할 것인지에 대한 통합적인 점검 체계를 제공합니다.

1. 네트워크 위협 탐지: Snort 룰셋 설계 및 운영

네트워크 보안의 첫 단계는 유입되는 트래픽 중 악성 패턴을 식별하는 것입니다. Snort는 룰(Rule)을 통해 특정 조건의 패킷을 탐지하며, 효율적인 룰 작성을 위해서는 헤더와 옵션의 구조를 정확히 이해해야 합니다.

1.1 Snort 룰의 기본 구조

Snort 룰은 트래픽의 대상을 정의하는 **헤더(Header)**와 탐지 조건을 상세히 설정하는 **옵션(Options)**으로 구성됩니다.

- **Action:** 매칭 시 수행할 동작 (예: `alert` - 알림, `drop` - 패킷 차단, `pass` - 통과)
- **Protocol:** 검사할 프로토콜 (예: `tcp`, `udp`, `http`)
- **Source/Destination:** 출발지 및 목적지의 IP 주소와 포트 번호
- **Rule Options:** 괄호 () 내에 정의되며, 문자열 패턴, 세션 상태, 식별 번호 등을 지정

1.2 핵심 탐지 옵션 및 작동 원리

정교한 탐지를 위해 다음과 같은 고급 옵션을 조합하여 사용합니다.

- **네트워크 변수:** `HOME_NET`(보호 대상 내부망)과 `EXTERNAL_NET`(외부망)을 정의하여 검사 범위를 제한합니다. 외부망을 정의할 때 `!$HOME_NET`(내부망이 아닌 모든 곳) 형식을 사용하면 내부 통신을 제외한 순수 외부 트래픽만 정밀하게 타격할 수 있습니다.
- **Content 및 Sticky Buffer:** `content` 옵션은 패킷 내 특정 문자열이나 16진수 패턴을 검색합니다. 이때 **Sticky Buffer**(예: `http_uri`, `http_client_body`)를 먼저 선언하면, 전체 패킷이 아닌 HTTP의 특정 영역(URI, 본문 등)에서만 검색하므로 오탐을 줄이고 성능을 높일 수 있습니다.
- **Flow 제어:** `flow:established,to_server`와 같이 설정하여 TCP 3-Way Handshake가 완료된 세션 중 클라이언트에서 서버로 향하는 요청 패킷만 검사하도록 제한합니다.
- **Flowbits:** 동일한 TCP 세션 내에서 이전 룰의 매칭 상태를 기억하는 기능입니다. 예를 들어 '로그인 시도(set)' → '로그인 실패 응답 확인(isset)' 순으로 룰을 구성하여 논리적인 공격 흐름을 추적할 수 있습니다.

1.3 탐지 정밀도 향상을 위한 수정자(Modifier)

단순 문자열 매칭의 한계를 극복하기 위해 위치와 범위 제한 옵션을 사용합니다.

옵션	기준점	설명
<code>offset / depth</code>	버퍼 시작점	시작점으로부터 특정 바이트 뒤부터, 혹은 특정 범위 내에서 검색
<code>distance / within</code>	이전 content 매칭 지점	이전 패턴이 끝난 지점으로부터 거리와 범위를 지정하여 검색

1.4 엔진별 HTTP 문법 차이 및 관리 포인트

사용하는 엔진(Snort 2, Snort 3, Suricata)에 따라 HTTP 버퍼를 지정하는 문법이 다르므로 주의가 필요합니다.

- **Snort 2.9:** `content: "POST"; http_method;` (콘텐츠 선언 후 수정자 배치)
- **Snort 3:** `http_method; content: "POST";` (언더바 형태의 Sticky Buffer 선언 후 콘텐츠 배치)
- **Suricata:** `http.method; content: "POST";` (점 표기법 형태의 Sticky Buffer 선언 후 콘텐츠 배치)

운영 시에는 sid(고유 식별 번호)와 rev(수정 버전)를 체계적으로 관리하여 롤의 중복을 방지하고 이력을 추적해야 합니다.

2. 클라이언트 사이드 보안 취약점: 자바스크립트 어택

웹 애플리케이션의 검증 로직이 서버가 아닌 사용자의 브라우저(클라이언트 사이드)에서 수행될 때 발생하는 취약점입니다. 공격자는 개발자 도구를 통해 실행 중인 코드를 분석하고 메모리 값을 조작하여 보안 로직을 우회할 수 있습니다.

2.1 취약한 로직 분석

특정 웹 페이지에서 "success"라는 값을 입력해야 성공하는 시나리오에서, 내부적으로 다음과 같은 토큰 생성 과정이 자바스크립트로 구현되어 있다고 가정합니다.

1. 사용자 입력값(phrase) 가져오기
2. `rot13()` 함수를 통해 알파벳을 13칸 밀어 문자열 변조
3. 변조된 값을 `md5()` 해시 함수로 32자리 고유값으로 변환
4. 생성된 해시값을 숨겨진 입력창(token)에 주입하여 서버로 전송

이 과정의 치명적인 약점은 **이벤트 리스너(Event Listener)**의 부재입니다. 페이지 로드 시 1회만 실행되므로, 사용자가 입력창의 값을 바꾸더라도 토큰 값은 갱신되지 않아 서버에서 "Invalid token" 에러가 발생합니다.

실습 화면 해석: 자바스크립트 우회 공격

먼저 분석 대상이 되는 웹 페이지의 구조를 확인합니다.

이후 브라우저 개발자 도구의 디버거(Debugger) 탭을 통해 토큰 생성 로직이 포함된 JS 파일을 분석합니다.

분석 결과, `generate_token()` 함수가 입력값의 ROT13 변조 후 MD5 해싱을 수행함을 확인했습니다. 이를 우회하기 위해 콘솔(Console) 탭에서 직접 DOM 객체의 값을 수정하고 함수를 강제로 재실행합니다.

- `document.getElementById("phrase").value = "success"` : 입력창 값을 성공 조건으로 변경
- `generate_token()` : 변경된 값에 맞는 올바른 토큰을 강제로 생성하여 주입

조작된 값을 가지고 제출 버튼을 누르면, 서버는 클라이언트가 계산해서 보낸 위조된 토큰을 신뢰하게 되어 공격이 성공합니다.

2.2 공격 자동화 및 네트워크 패킷 분석

수동 우회 과정을 파이썬 스크립트로 자동화하여 서버에 직접 위조된 페이로드를 전송할 수 있습니다. 이 과정에서 발생하는 트래픽을 와이어샤크(Wireshark)로 캡처하여 분석합니다.

먼저 목적지 IP(172.16.20.15)와 HTTP 프로토콜만 필터링하여 캡처를 준비합니다.

파이썬 스크립트를 실행하면 서버 접속, CSRF 토큰 수집, 인증, 페이로드 연산(ROT13 → MD5), 최종 패킷 전송 순으로 진행되며 'Well done!' 응답을 확인합니다.

캡처된 패킷 목록에서 `POST /dvwa/vulnerabilities/javascript` 요청을 확인할 수 있습니다.

해당 패킷의 상세 내용을 분석하면 `phrase=success`와 함께 정교하게 위조된 `token` 값이 전송되었음을 증명할 수 있습니다.

2.3 보안 점검 포인트

- **검증 위치의 적절성**: 중요 데이터의 가공 및 검증 로직은 반드시 서버 사이드(Server-side)에서 수행되어야 합니다.
- **토큰 생성 주체**: 브라우저가 스스로 토큰을 계산하게 하지 말고, 서버가 생성하여 발급하는 방식을 사용해야 합니다.
- **동적 검증 메커니즘**: 입력값 변경 시 실시간으로 상호 검증하는 장치가 없다면 단순한 값 조작만으로 우회가 가능합니다.

3. 세션 관리 취약점: Weak Session IDs

세션 ID는 사용자를 식별하는 임시 암호와 같으므로 예측 불가능한 복잡성을 가져야 합니다. 만약 세션 ID 발급 규칙에 일정한 패턴이 있다면, 공격자는 이를 통해 타인의 세션을 탈취하는 세션 하이재킹(Session Hijacking)을 수행할 수 있습니다.

3.1 취약한 세션 ID의 정의

다음과 같은 경우 **취약한 세션 ID(Weak Session ID)**로 판단합니다.

- 발급 과정에서 일정한 숫자/문자 패턴이 존재하는 경우 (예: 1, 2, 3 순차 증가)
- 사용자 ID, IP 주소 등 예측 가능한 값을 기반으로 생성된 경우
- 세션 ID의 길이가 너무 짧아 브루트포스(Brute-force) 공격에 취약한 경우

3.2 세션 분석 실습 절차

Burp Suite와 Firefox 브라우저를 연동하여 서버가 발급하는 세션 ID의 규칙성을 분석합니다.

1. **Burp Suite 설정**: Proxy 리스너를 127.0.0.1:8080으로 설정하고 인터셉트(Intercept) 기능을 활성화합니다.

2. **브라우저 프록시 설정:** Firefox 네트워크 설정에서 수동 프록시 구성(Manual proxy configuration)을 선택하고 127.0.0.1:8080을 입력합니다.

3. **세션 ID 수집:** DVWA의 Weak Session IDs 페이지에서 'Generate' 버튼을 반복적으로 클릭하며 발급되는 쿠키 값을 확인합니다.

인터셉트된 패킷의 Cookie 헤더를 통해 서버가 전달하는 세션 값을 확인합니다.

HTTP history 탭에서 연속적으로 발급된 요청 내역을 분석합니다.

분석 결과, dvwaSession 값이 1, 2, 3과 같이 단순히 증가하는 패턴이 확인되었습니다. 이는 공격자가 다음 세션 ID를 쉽게 예측하여 타인의 계정 권한을 탈취할 수 있음을 의미합니다.

3.3 세션 보안 강화 방안

- **암호학적 난수 생성:** 예측 불가능한 강력한 난수 생성기(CSPRNG)를 사용하여 세션 ID를 생성해야 합니다.
- **충분한 길이 확보:** 브루트포스 공격을 방지하기 위해 충분한 길이의 문자열을 사용합니다.
- **세션 만료 정책:** 일정 시간 활동이 없으면 세션을 만료시키고, 로그아웃 시 서버에서 세션을 즉시 폐기해야 합니다.

종합 점검 체크리스트

점검 영역	핵심 확인 항목	판단 기준 (Pass/Fail)
네트워크 탐지	HTTP 영역별 Sticky Buffer 사용 여부	전체 패킷이 아닌 특정 버퍼(URL, Body 등)를 지정했는가?
클라이언트 보안	중요 검증 로직의 수행 위치	모든 최종 검증이 서버 사이드에서 수행되는가?
세션 관리	세션 ID의 무작위성 및 복잡성	ID 발급 시 일정한 패턴이나 순차적 증가가 없는가?
인증 메커니즘	토큰 생성 방식의 투명성	클라이언트에서 계산 가능한 단순 알고리즘(MD5 등)을 사용하지 않는가?