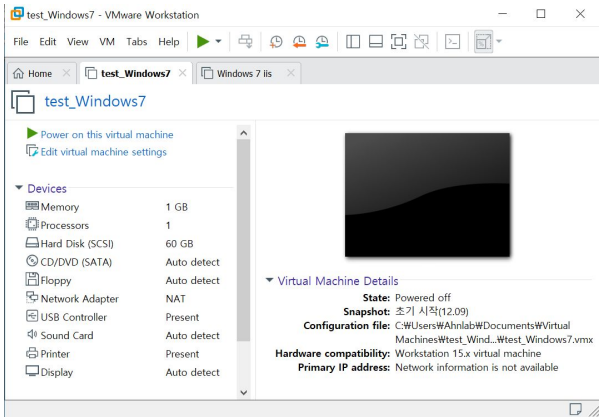
로그 분석 / 스노트 룰 스터디

권문정

Contents

01. 웹 공격 로그 분석을 위한 환경 구성

■ 실습 환경

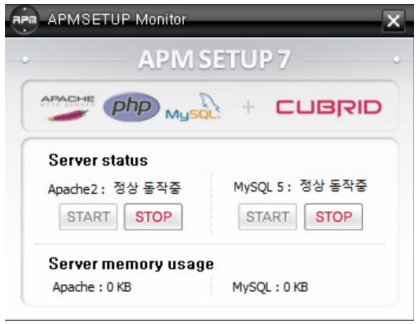


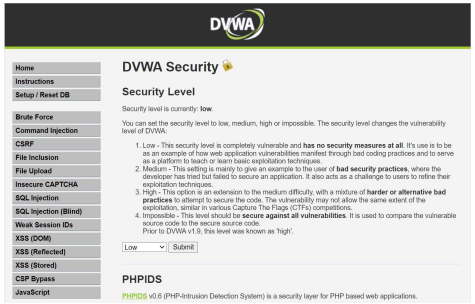
1. 가상 환경 구축
- VMware Workstation Pro(free trial 30일)
- OS : Window10 64 bit

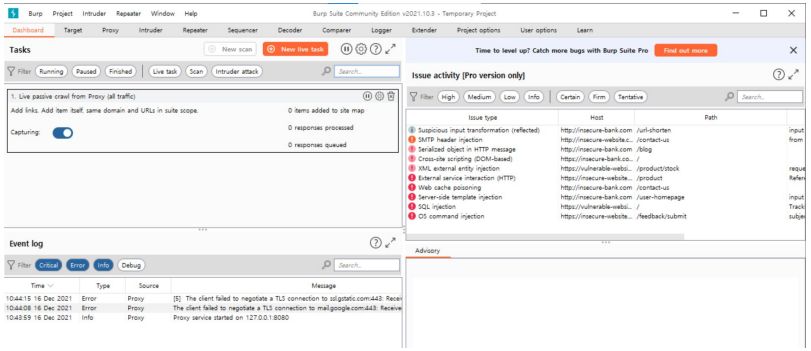
2. 실습 도구
- APMSETUP7 (window 32 bit)
  - Apache(2.2.14)
  - PHP(5.1.41-community)
  - MySQL5.1.41-community MySQL Community Server (GPL)

- Damn Vulnerable Web Application (DVWA)
  : 취약점 진단/모의해킹을 할 수 있는 웹 어플리케이션. PHP/MySQL 환경. Security Level : Low로 설정

- Burp Suite
  : 프록시(Proxy)를 사용하여 네트워크에서 통신하는 패킷을 가로채 분석 및 조작, 취약점을 확인하는 도구







02. Apache access.log 분석 – GET

■ Brute Force Attack

- 입력한 비밀번호는 GET 방식으로 password 파라미터에 담겨 전달됨
- password 파라미터에 여러 값들을 입력하여 수십 번의 로그인 시도를 하는 것으로 보아, Username이 'admin'인 계정의 비밀번호를 알아내기 위해 무차별 대입 공격을 하는 것으로 추정

127.0.0.1 - - [21/Dec/2021:09:36:51 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=1&Login=Login HTTP/1.1" 200 4241

127.0.0.1 - - [21/Dec/2021:09:36:51 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=qewr&Login=Login HTTP/1.1" 200 4241

127.0.0.1 - - [21/Dec/2021:09:36:51 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=1234&Login=Login HTTP/1.1" 200 4241

127.0.0.1 - - [21/Dec/2021:09:36:52 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=asdf&Login=Login HTTP/1.1" 200 4241

127.0.0.1 - - [21/Dec/2021:09:36:52 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=abcd&Login=Login HTTP/1.1" 200 4241

127.0.0.1 - - [21/Dec/2021:09:36:52 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=password&Login=Login HTTP/1.1" 200 4284

127.0.0.1 - - [21/Dec/2021:09:36:53 +0900] "GET /dvwa/vulnerabilities/brute/?username=admin&password=qwer1234&Login=Login HTTP/1.1" 200 4241

02. Apache access.log 분석 – GET

■ CSRF

- 비밀번호가 GET 방식으로 변경됨
- 변경되는 비밀번호의 값이 password_new 와 password_conf 파라미터에 담겨 전달됨
- 서버의 입장에서는 세션/쿠키에 담긴 로그인 정보가 검증 없이 전달되므로, 비밀번호 변경 요청이 사용자의 요청인지, 공격자가 로그인 정보를 가로채서 시도한 요청인지 알 수 없음
- 본 로그는 웹 공격을 위한 환경에서 생성되었으므로 경로명이 표시되어 명시적으로 알 수 있으나, 실제 상황에서 로그가 표시되었을 때 어떻게 판단해야 할지 어려움이 있을 것으로 추정
  → 검증되지 않은 요청을 서버가 받아들이는 것과 마찬가지로 분석자 또한 눈치채지 못할 수 있을 가능성 존재
  → 의심스러울 경우, 사용자의 중요한 상태를 바꾸는 요청 (ex. 비밀번호, 전화번호 변경) 시에 CAPTCHA, 현재 비밀번호 입력 등의 검증 절차가 구비되어 있는지 확인 필요
  → 관리자 계정의 정보 변경 시, 접근 권한과 기록을 주의 깊게 확인할 필요가 있음

127.0.0.1 - - [21/Dec/2021:09:40:59 +0900] "GET /dvwa/vulnerabilities/csrf/?password_new=abcd&password_conf=abcd&Change=Change HTTP/1.1" 200 4494

127.0.0.1 - - [21/Dec/2021:09:47:15 +0900] "GET /dvwa/vulnerabilities/csrf/?password_new=csrf&password_conf=csrf&Change=Change HTTP/1.1" 200 4494

127.0.0.1 - - [21/Dec/2021:10:48:05 +0900] "GET /dvwa/vulnerabilities/csrf/?password_new=csrf&password_conf=csrf&Change=Change HTTP/1.1" 200 4494

127.0.0.1 - - [21/Dec/2021:10:48:22 +0900] "GET /dvwa/vulnerabilities/csrf/?password_new=csrf&password_conf=csrf&Change=Change HTTP/1.1" 200 4494

127.0.0.1 - - [21/Dec/2021:13:08:46 +0900] "GET /dvwa/vulnerabilities/csrf/?password_new=csrfz&password_conf=csrfz&Change=Change HTTP/1.1" 200 4494

02. Apache access.log 분석 – GET

■ SQL Injection

- WHERE 절에서 1=1 라는 참일 수밖에 없는 문장을 OR와 함께 사용하여 모두 참인 문장을 삽입
- 뒷 부분은 #를 사용하여 주석처리 해줌으로써 어떤 비밀번호를 입력하든 결과를 반환 받을 수 있음
- SQL Injection의 가장 기본적인 형태
- GET 방식으로 공격이 진행되므로 값이 URL의 파라미터로 전달 됨. 즉, URL 인코딩이 된 문자열을 확인 가능
  → URL Decoder를 사용하여 문자열 해독 후 해석 필요

127.0.0.1 - - [21/Dec/2021:09:47:28 +0900]
"GET/dvwa/vulnerabilities/sqli/?id=%27+OR+1%3D1+%23&Submit=Submit HTTP/1.1" 200 438
→ **id=' OR 1=1 #**

127.0.0.1 - - [21/Dec/2021:09:50:12 +0900]
"GET/dvwa/vulnerabilities/sqli/?id=%27+OR+1%3D1+%23&Submit=Submit HTTP/1.1" 200 438
→ **id=' OR 1=1 #**

127.0.0.1 - - [21/Dec/2021:10:52:44 +0900]
"GET/dvwa/vulnerabilities/sqli/?id=%27+OR+1%3D1+%23&Submit=Submit HTTP/1.1" 200 4388
→ **id=' OR 1=1 #**

02. Apache access.log 분석 – GET

■ Blind SQL Injection

- id가 1인 사용자가 있는지, 없는지 알아내고 있음
- AND 연산이 실행되고 있는지 확인하여 SQL 쿼리문이 뒤에 있는지 확인하고 있음.
  WHY? SQL 쿼리문이 뒤에 있다면 SQL Injection으로 DB의 정보를 캐낼 수 있으므로.

127.0.0.1 - - [21/Dec/2021:09:51:29 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=1&Submit=Submit HTTP/1.1" 200 4155
→ **id=1**

127.0.0.1 - - [21/Dec/2021:09:51:41 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=1%27+AND+1%3D2&Submit=Submit HTTP/1.1" 200 5425
→ **id=1' AND 1=2**

127.0.0.1 - - [21/Dec/2021:09:51:56 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=1%27+AND+1%3D2+%23&Submit=Submit HTTP/1.1" 404 4161
→ **id=1' AND 1=2 #**

02. Apache access.log 분석 – GET

■ Blind SQL Injection

- 앞 부분의 문장을 모두 참으로 만들고, 뒷부분의 문장에 database(), length()함수를 통해 데이터베이스 이름의 길이를 확인하고 있음.
- 숫자가 4에서 멈춘 것으로 보아 데이터베이스의 이름은 4글자인 듯 함.

* database() : 데이터베이스 이름을 반환해주는 함수
* length() : 문자열의 길이를 반환해주는 함수

127.0.0.1 - - [21/Dec/2021:10:44:14 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=+1%27+and+length%28database%28%29%29+%3D+5%23&Submit=Submit HTTP/1.1" 404 4161
→ **id= 1' and length(database()) = 5#**

127.0.0.1 - - [21/Dec/2021:10:44:17 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=+1%27+and+length%28database%28%29%29+%3D+4+%23&Submit=Submit HTTP/1.1" 200 4155
→ **id= 1' and length(database()) = 4 #**

02. Apache access.log 분석 – GET

■ Blind SQL Injection

- 앞 부분의 문장을 모두 참으로 만들고, 뒷부분의 문장에 database(), substr() 함수를 통해 데이터베이스 이름의 철자를 확인하고 있음.

* substr() : 반환하는 문자를 알려주는 함수
* substr(database(),1,1)는 database()가 반환하는 문자열의 첫 번째 글자를 반환한다.
    - 즉, 데이터베이스 이름의 첫 번째 글자를 알아낸다.
    - 이런 식으로 4번째 글자까지 알아낸다.
    - 데이터베이스 이름을 확인하는 시도가 어디에서 멈췄는지 확인하여 1~4번째 글자가 d, v, w, a임을 알아냈다.

127.0.0.1 - - [21/Dec/2021:10:44:45 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=+1%27+and+substr%28database%28%29%2C1%2C1%29%3D+%27a%27+%23&Submit=Submit HTTP/1.1" 404 4161
→ **id= 1' and substr(database(),1,1)= 'a' #**

127.0.0.1 - - [21/Dec/2021:10:44:49 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=+1%27+and+substr%28database%28%29%2C1%2C1%29%3D+%27b%27+%23&Submit=Submit HTTP/1.1" 404 4161
→ **sqli_blind/?id= 1' and substr(database(),1,1)= 'b' #**

127.0.0.1 - - [21/Dec/2021:10:44:53 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=+1%27+and+substr%28database%28%29%2C1%2C1%29%3D+%27c%27+%23&Submit=Submit HTTP/1.1" 404 4161
→ **id= 1' and substr(database(),1,1)= 'c' #**

02. Apache access.log 분석 – GET

■ Blind SQL Injection

- users 테이블의 user가 'admin'인 첫 번째 사용자의 password 첫 글자를 확인
- 이런 식으로 password의 문자열 자리 수마다 문자를 대입하여 확인해 봄

127.0.0.1 - - [21/Dec/2021:10:53:17 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=1%27+AND+substr%28+%28SELECT+password+FROM+users+WHERE+user%3D%27admin%27+LIMIT+0%2C1%29%2C1%2C1%29+%3D+%27a%27+%23&Submit=Submit HTTP/1.1" 404 4161

→ id=1' AND substr((SELECT password FROM users WHERE user='admin' LIMIT 0,1),1,1) = 'a' #

127.0.0.1 - - [21/Dec/2021:10:53:20 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=1%27+AND+substr%28+%28SELECT+password+FROM+users+WHERE+user%3D%27admin%27+LIMIT+0%2C1%29%2C1%2C1%29+%3D+%27b%27+%23&Submit=Submit HTTP/1.1" 404 4161

→ id=1' AND substr((SELECT password FROM users WHERE user='admin' LIMIT 0,1),1,1) = 'b' #

127.0.0.1 - - [21/Dec/2021:10:53:23 +0900] "GET /dvwa/vulnerabilities/sqli_blind/?id=1%27+AND+substr%28+%28SELECT+password+FROM+users+WHERE+user%3D%27admin%27+LIMIT+0%2C1%29%2C1%2C1%29+%3D+%27c%27+%23&Submit=Submit HTTP/1.1" 200 4155

→ id=1' AND substr((SELECT password FROM users WHERE user='admin' LIMIT 0,1),1,1) = 'c' #

02. Apache access.log 분석 – GET

■ XSS

- URL에 HTML태그가 포함된 것으로 보아 XSS임을 유추할 수 있음
- ‘xss’, ‘script’라는 내용을 표시하는 alert 창 띄우기, 쿠키 내용을 알려주는 alert창 띄우기 등의 로그가 표시됨

127.0.0.1 - - [21/Dec/2021:10:47:37 +0900] "GET /dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27xss%27%29%3C%2Fscript%3E HTTP/1.1" 200 4244
→ **name=<script>alert('xss')</script>**

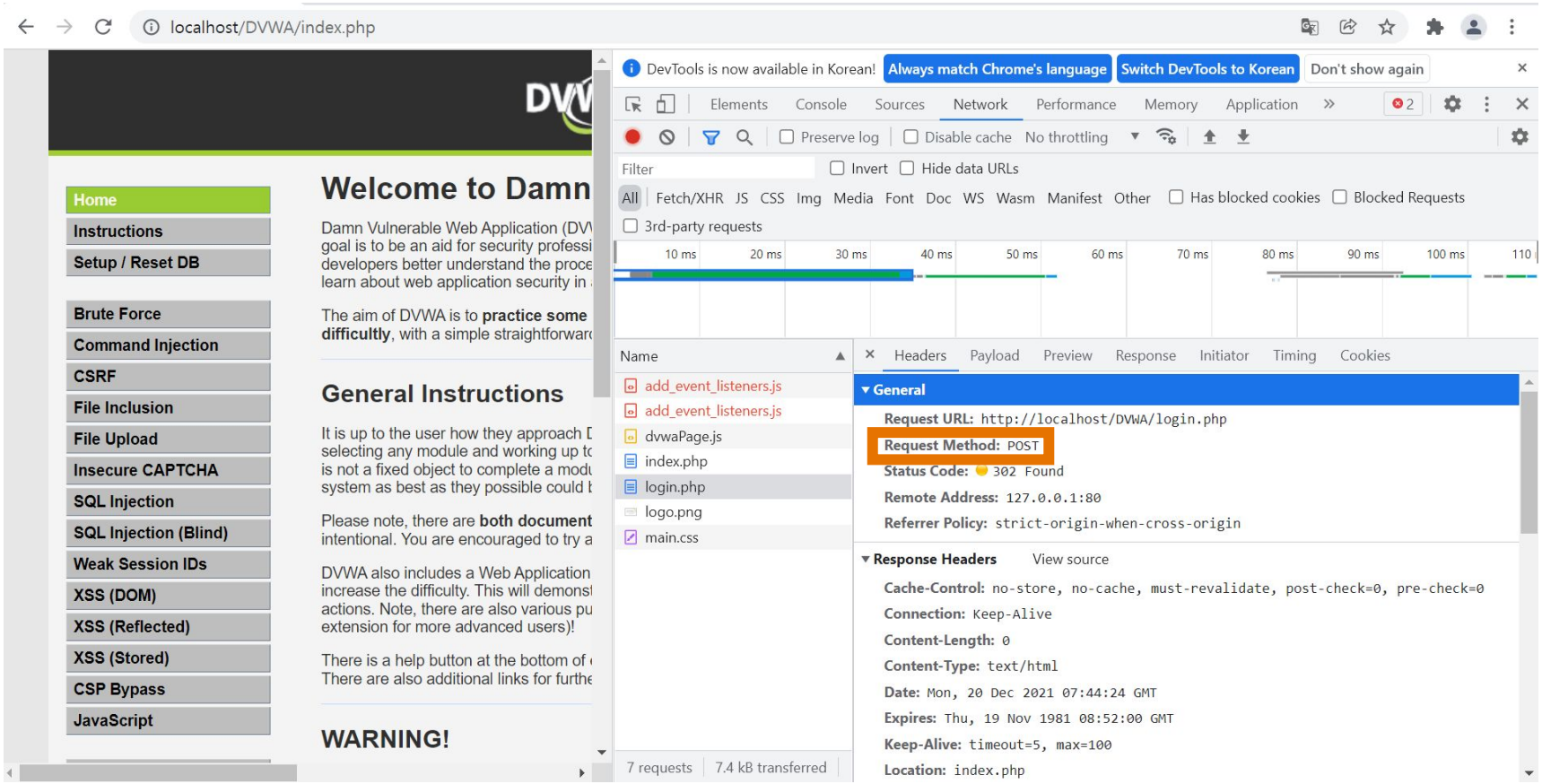
127.0.0.1 - - [21/Dec/2021:10:48:02 +0900] "GET /dvwa/vulnerabilities/xss_r/?name=%3Cscript%3Ealert%28%27script%27%29%3C%2Fscript%3E HTTP/1.1" 200 4247
→ **name=<script>alert('script')</script>**

127.0.0.1 - - [21/Dec/2021:09:48:00 +0900] "GET /dvwa/vulnerabilities/xss_r/?name=%3CSCRiPt%3Ealert%28document.cookie%29%3C%2Fscript%3E HTTP/1.1" 200 4254
→ **name=<SCRiPt>alert(document.cookie)</script>**

02. Live HTTP Headers를 활용한 로그 분석 – POST

■ POST 방식으로 전송되는 데이터 확인

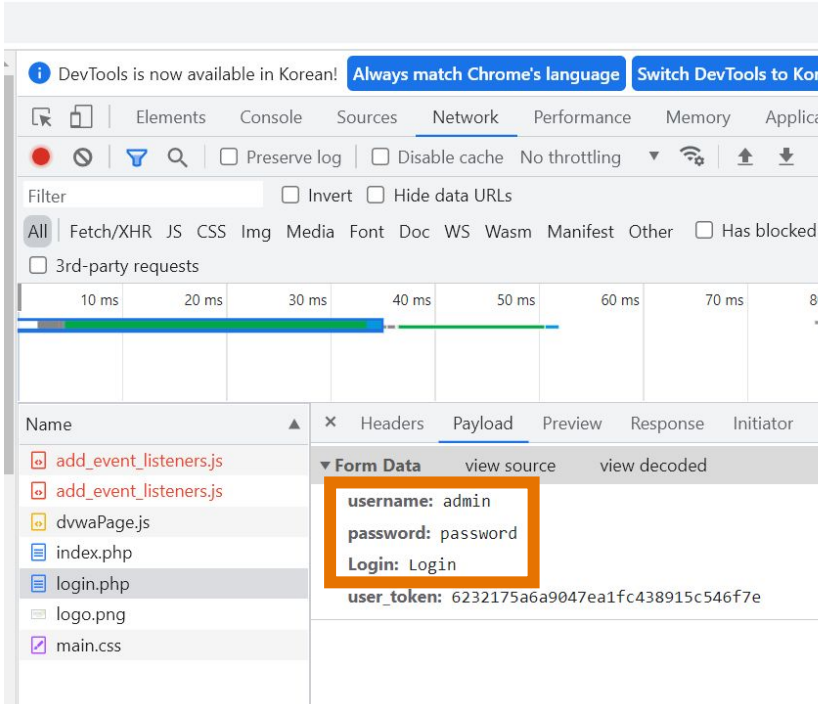
- DVWA 로그인 시에 데이터를 HTTP POST 방식으로 주고받는 것을 알 수 있음
- POST는 GET과 다르게 access.log에 기록이 상세히 남지 않아 분석이 어려움
  → Chrome의 Live HTTP Headers를 설치하여 바디의 폼 데이터 확인 필요

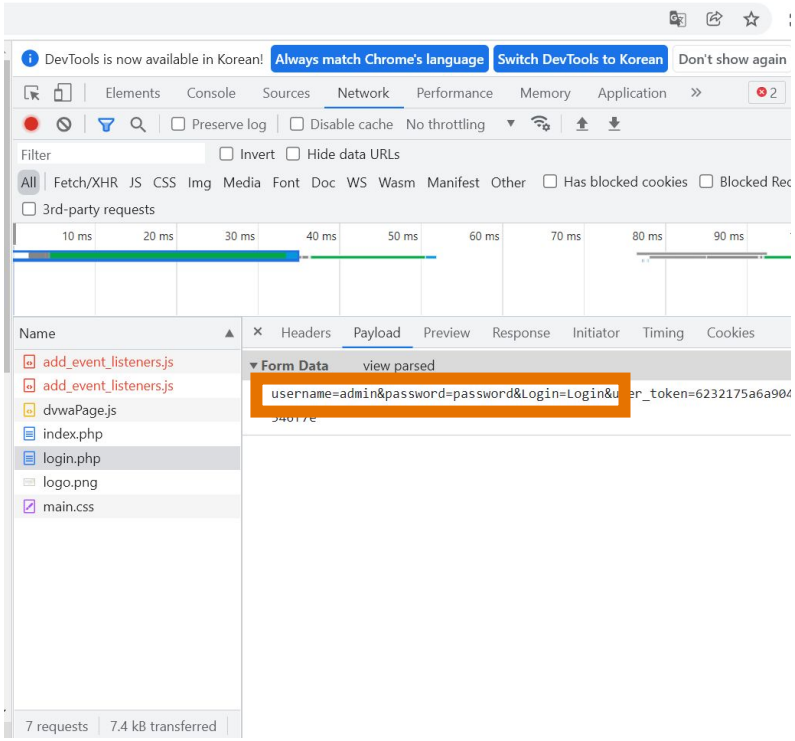


02. Live HTTP Headers를 활용한 로그 분석 – POST

■ POST 방식으로 전송되는 데이터 확인

- 폼 데이터를 확인하여 로그인할 때 사용되는 username과 password를 알아낼 수 있음
- POST 방식은 데이터를 헤더가 아닌, 폼에 담아서 보내므로 URL으로 조작 불가





02. Live HTTP Headers를 활용한 로그 분석 – POST

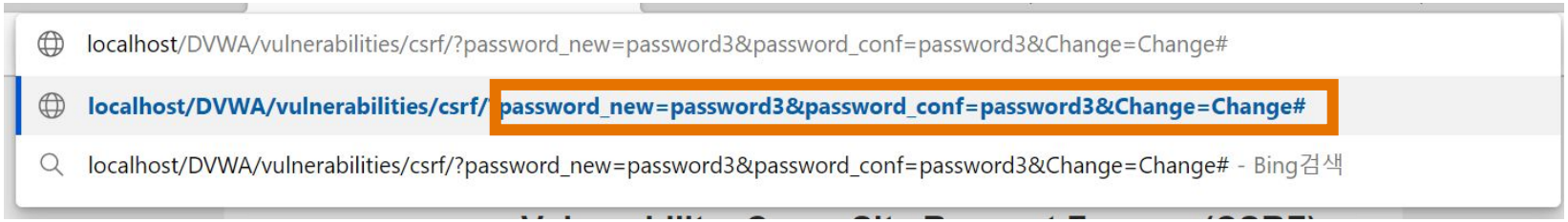
■ CSRF

- 사용자가 로그인한 상태에서 비밀번호를 바꾸는 요청을 수행
  세션/쿠키가 남아있는 상태에서 비밀번호 검증 없이 비밀번호를 수정
- 공격자의 입장에서 비밀번호를 수정하자, 비밀번호의 값이 GET 방식으로 전달되어 URL에 그대로 노출됨

02. Live HTTP Headers를 활용한 로그 분석 – POST

■ CSRF

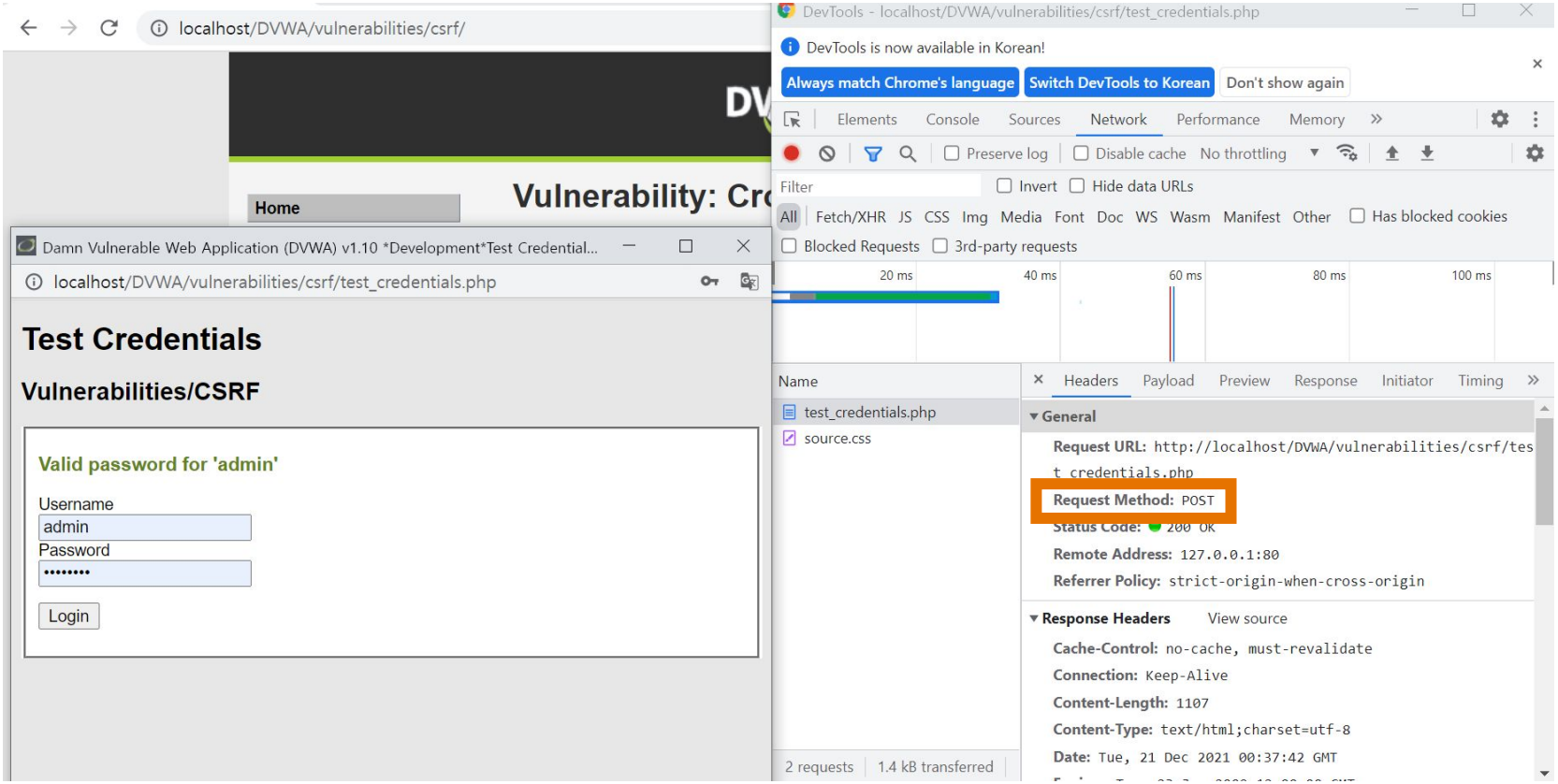
- URL 창에서 비밀번호 파라미터에 맞게 값을 바꿔주고 Enter키를 누르자, 비밀번호를 바꾸는 페이지에서 비밀번호가 변경되었다는 안내 메시지가 표시됨



02. Live HTTP Headers를 활용한 로그 분석 – POST

■ CSRF

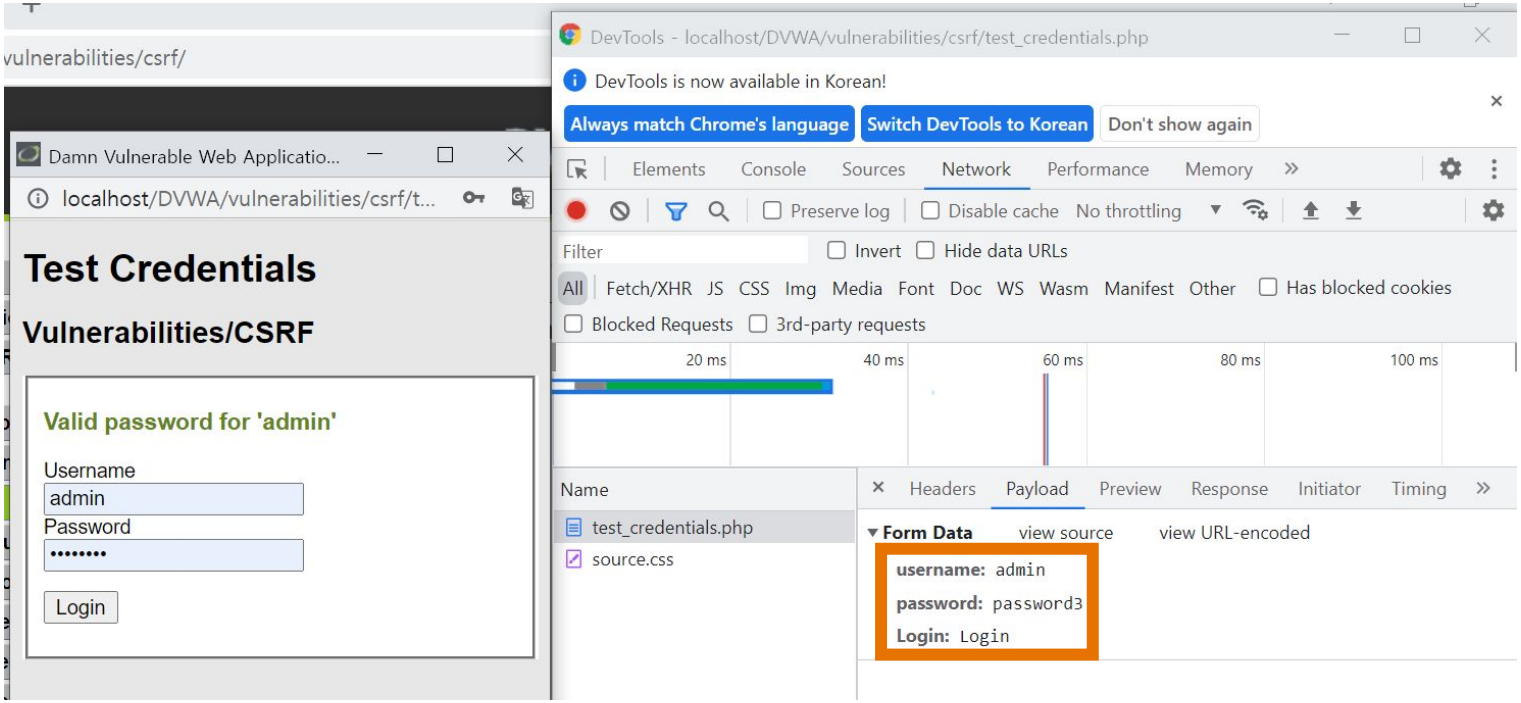
- 로그인 테스트 페이지는 POST 방식으로 데이터를 주고 받음



02. Live HTTP Headers를 활용한 로그 분석 – POST

■ CSRF

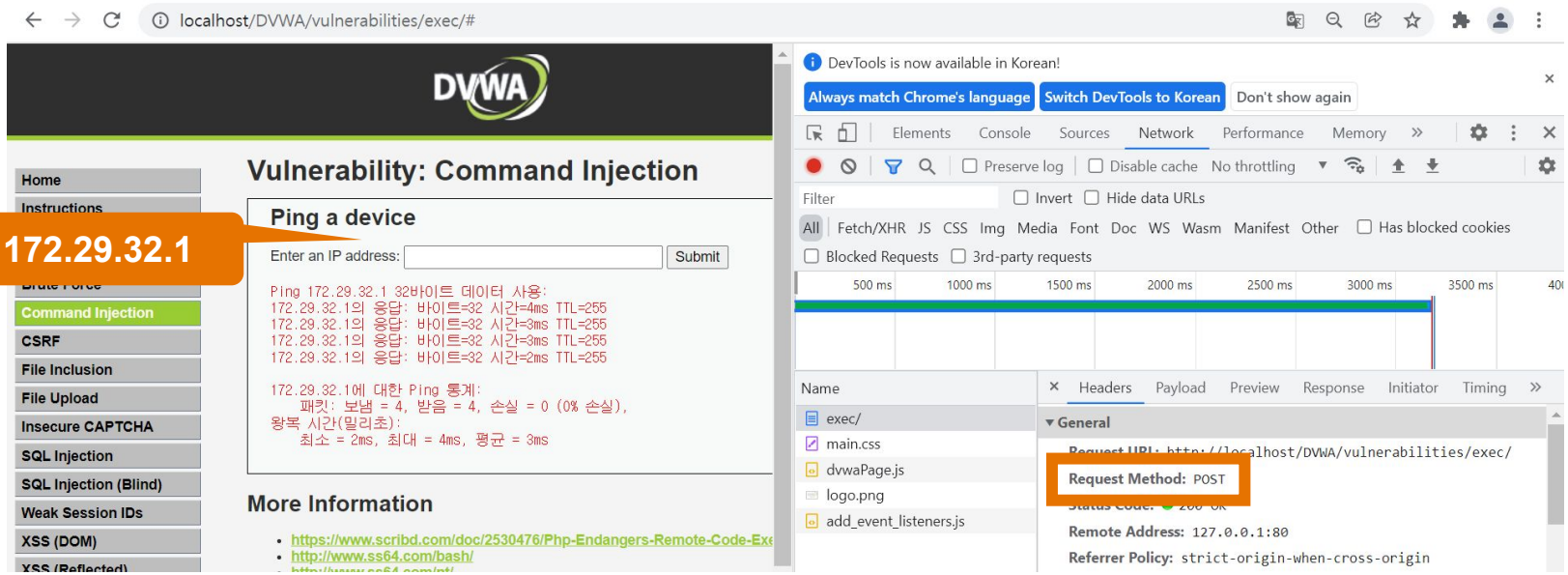
- 로그인 테스트 페이지에서 방금 URL에서 수정한 비밀번호를 입력하고 로그인 시도를 하자 비밀번호를 올바르게 입력했다고 안내함
- 폼 데이터에서 비밀번호를 방금 공격자가 위조한 값으로 입력한 것을 확인할 수 있음.

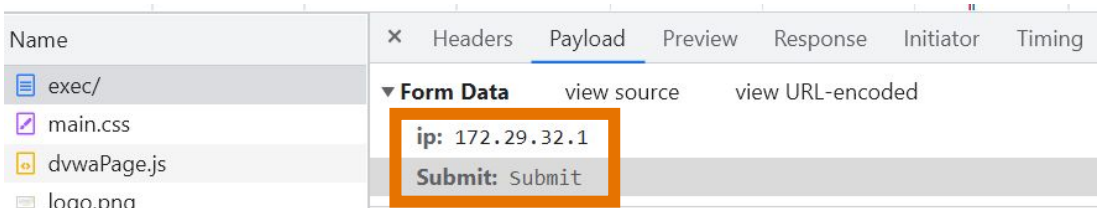


02. Live HTTP Headers를 활용한 로그 분석 – POST

■ Command Injection

- IP를 입력하는 창에 자신의 IP를 입력하고 Submit 버튼을 누르면, 그 값이 POST 방식으로 전달되어 수행됨
- 웹페이지에 CMD창에서 ping + IP 명령어를 입력하는 것과 동일한 결과가 표시됨
  → ping 명령어를 실행하는 웹 페이지라고 추측할 수 있음

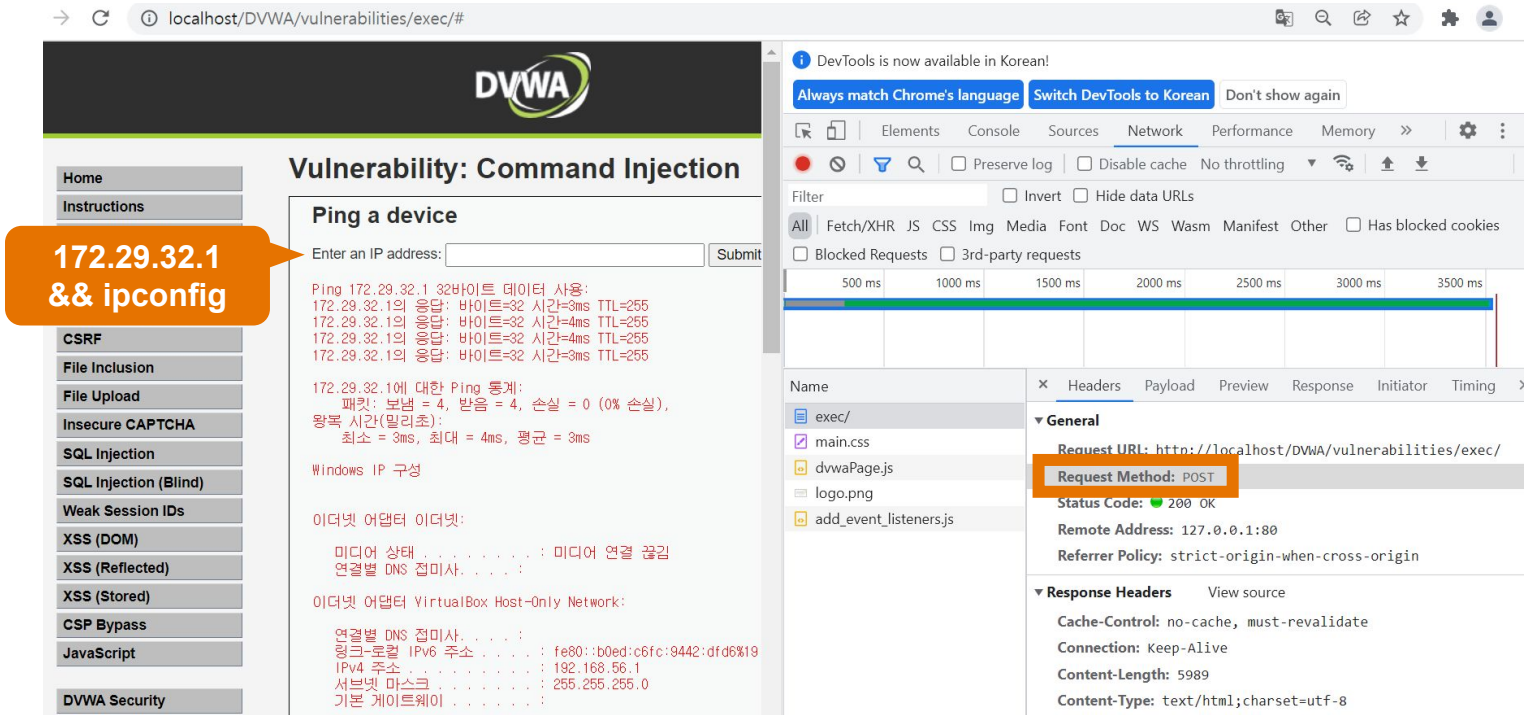


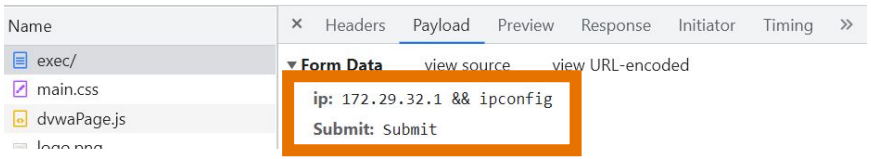


02. Live HTTP Headers를 활용한 로그 분석 – POST

■ Command Injection

- ping 명령어를 입력한 뒤 윈도우 운영체제에서 명령어 여러 개를 실행할 수 있게 해주는 **&&**를 사용하여 ipconfig 명령어를 입력하자, ping 명령어가 실행된 후 ipconfig 명령어가 함께 실행 됨
  → Command Injection 취약점이 존재함을 알 수 있음. POST 방식으로 전달되므로 폼 데이터를 통해 확인 가능

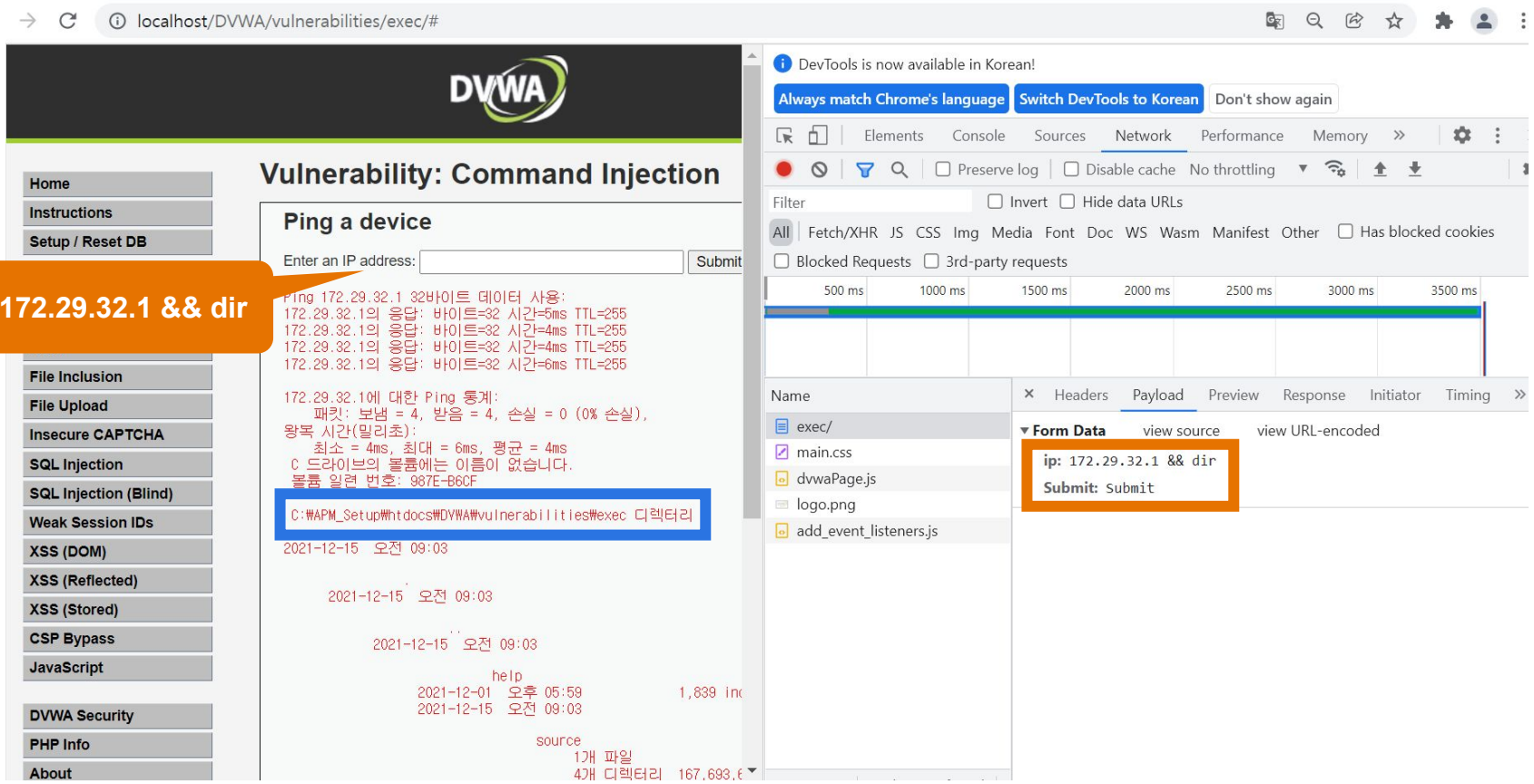




02. Live HTTP Headers를 활용한 로그 분석 – POST

■ Command Injection

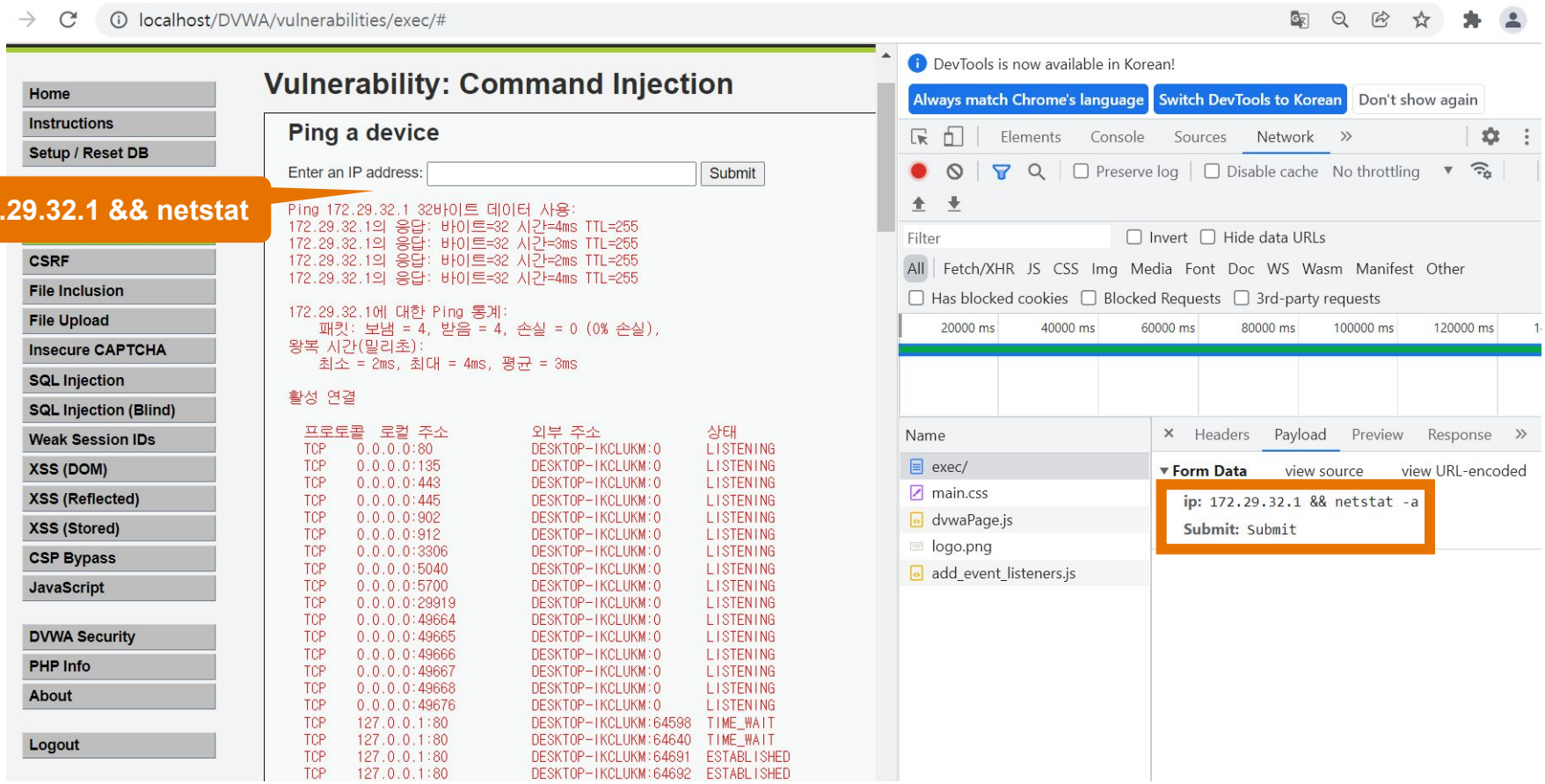
- ping 명령어를 입력한 뒤 &&를 사용하여 dir 명령어를 입력하자, ping 명령어가 실행된 후 dir 명령어가 함께 실행되어 현재 위치와 현재 위치에 있는 디렉토리 및 파일 목록들을 보여줌
  → Command Injection 공격으로 서버의 구조와 파일 내용을 알아내려고 시도하였음. 본 공격에 대한 데이터가 POST 형식으로 전달되므로, 공격이 시도 되었음을 폼 데이터를 통해 알 수 있음



02. Live HTTP Headers를 활용한 로그 분석 – POST

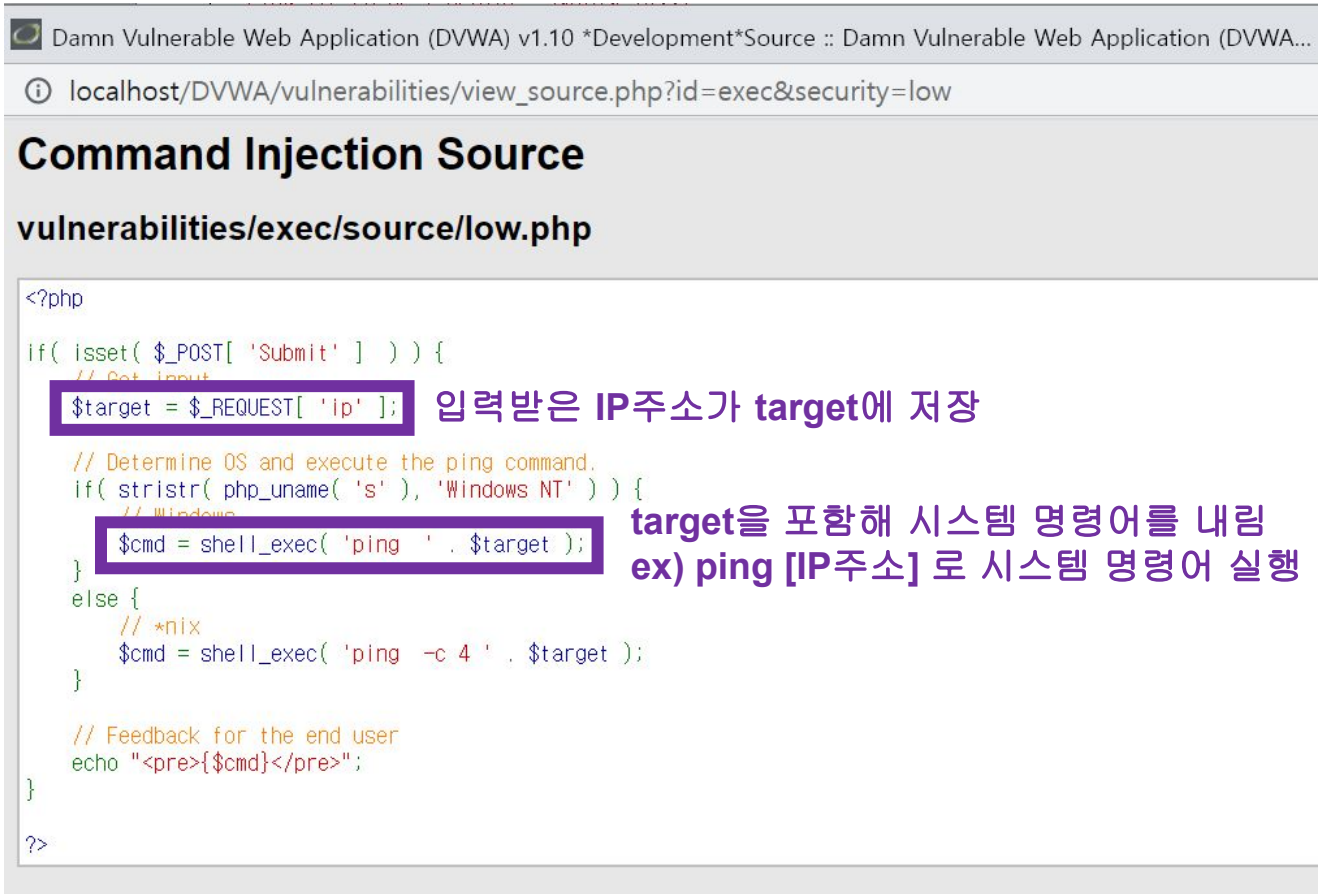
■ Command Injection

- ping 명령어와 함께 현재 열린 포트를 확인하는 netstat 명령어를 사용한 Command Injection 공격이 시도되었음을 폼 데이터에서 확인 가능. 이렇게 포트 스캔 후에 취약점을 찾으려고 했음을 알 수 있음
  (ex. 22, 3389 포트가 열려있는 경우, 원격 접속과 관련한 공격을 시도해볼 수 있음)



02. Live HTTP Headers를 활용한 로그 분석 – POST

■ **Command Injection**



References

- SQL Injection 공격 구문/패턴
  - https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=is_king&logNo=221402635339%20cap
  - https://woojoong2.tistory.com/39

- Blind SQL Injection 공격 구문/패턴
  - https://m.blog.naver.com/PostView.naver?isHttpsRedirect=true&blogId=okopok5019&logNo=221543684263%20%20%27%20UNION%20SELECT%201%20#
  - https://jdh5202.tistory.com/640

- ASCII Code 일람표
  https://m.blog.naver.com/ansdbtls4067/220624120433

- URL 인코더/디코더
  https://www.convertstring.com/ko/EncodeDecode/UrlDecode