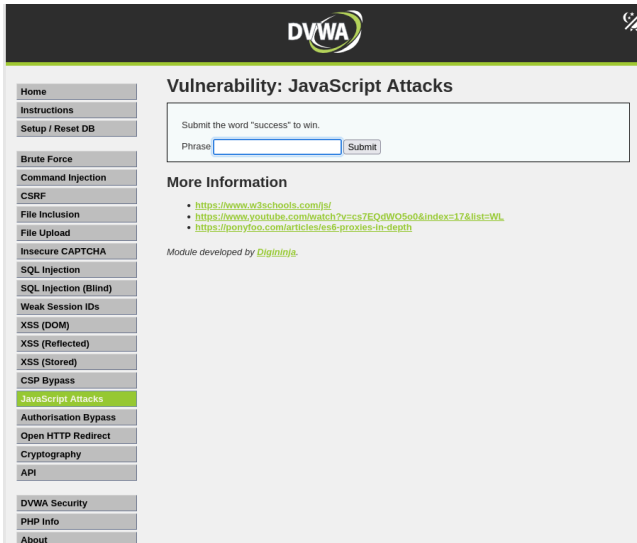


자바스크립트어택

2026년 7월 14일 화요일 오후 3:45

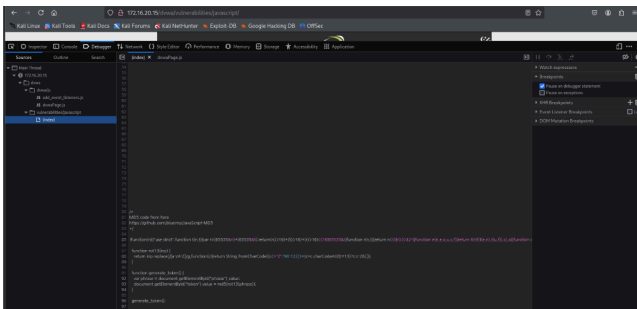
자바스크립트 어택은 웹사이트단에서의



이 문제는 웹 브라우저단(클라이언트 사이드)에서 자바스크립트가 수행하는 난독화, 인증, 혹은 입력값 검증이 얼마나 취약한지를 직접 체감하고 우회해보는 실습 항목

그냥 success를 입력하고 전송하면 "Invalid token" 같은 에러가 발생하며 거부당함 왜냐하면 사용자가 버튼을 누르는 순간, 백그라운드에서 자바스크립트 코드가 입력값을 특정 방식으로 변조하거나 토큰(MD5 해시 등)을 생성해서 서버로 함께 보내기 때문에

즉, 이 항목은 "서버로 데이터가 넘어가기 전, 브라우저에서 자바스크립트로 조작하는 보안 로직을 어떻게 파헤치고 우회할 것인가?"를 연습하는 문제



해당항목에서 f12를 눌러 개발자도구 들어간 뒤 디버거 항목에서 자바스크립트의 텍스트파일로 들어가서 맨 아래로 드래그

```
function generate_token() {
  var phrase = document.getElementById("phrase").value;
  document.getElementById("token").value = md5(rot13(phrase));
}

generate_token();
```

91~96번 라인: 토큰 생성 및 자동 실행 (generate_token)

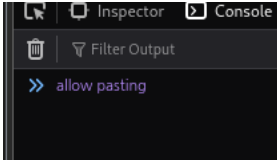
이 문제의 핵심 동작이 일어나는 구간입니다.

1. `var phrase = document.getElementById("phrase").value;`
 - 화면의 입력창(id가 phrase인 엘리먼트)에 사용자가 타이핑한 문자열(예: ChangeMe)을 가져옵니다. `document.getElementById("phrase")`: 브라우저 화면(HTML 문서) 전체를 통틀어서, 아이디(id) 속성값이 정확히 "phrase"인 태그(입력창 엘리먼트)를 살살이 찾아내라는 뜻. `.value`: 찾아낸 그 입력창 상자 안에 사용자가 실제로 키보드로 타이핑해서 입력해 둔 글자(텍스트)를 꺼내오라는 명령어.
2. `md5(rot13(phrase))`
 - 가져온 문자열을 먼저 `rot13` 함수로 변조합니다. 알파벳을 딱 반 바퀴(13칸) 뒤로 밀어서 글자를 꼬아버리는 자바스크립트 내부 함수
 -

- 변조된 문자열을 다시 md5 함수를 이용해 32글자 해시값으로 변환합니다.
- 위에서 꼬아놓은 문자열을 받아, 깨기 힘든 32자리의 고유한 암호화 해시값으로 압축해 버리는 MD5 해시 함수
-
- 3. `document.getElementById("token").value = ...`
- 최종적으로 만들어진 해시값을 화면에 숨겨진 입력창(id가 token인 엘리먼트)에 강제로 집어넣습니다. 이번에는 화면(HTML)에서 id가 "token"인 태그를 찾으라는 명령어
-
- 4. `generate_token();` (96번 라인)
- 페이지가 처음 로딩될 때 이 전체 과정을 **최초 1회 자동으로 실행**합니다.

이 코드는 사용자가 입력창에 글자를 바꿔도 페이지가 처음 켜질 때 딱 한번 실행된 이후로는 다시 실행되지 않는다는 맹점이 있음.

개발자도구에서 콘솔탭을 클릭한뒤



allow pasting

이렇게 입력해준다 이유는 브라우저가 기본적으로 막아둔 복사 붙여넣기 잠금을 해제하기 위함.

이제 아래 코드를 넣어보자.

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

해당 코드 분석해보자

Document : 현재 브라우저가 보여주고 있는 웹페이지 전체를 뜻함.

`.getElementById("phrase")`: 웹페이지 안에서 id이름이 phrase인 요소를 찾아라 ---> 입력창 뜻함.

`.value` : 선택한 요소의 안에 들어있는 값(텍스트) ---> 입력창에 적혀 있는 글자 그 자체를 의미함.

`= "success"` : 프로그래밍에서 `=`은 "같다"가 아니라 "오른쪽에 있는 값을 왼쪽에 집어넣어라(대입)"라는 뜻으로 해당 입력창의 값을 문자열 success로 덮어쓰워

success에 맞는 정상적인 토큰을 계산해서 변수에 주입

`generate_token()`

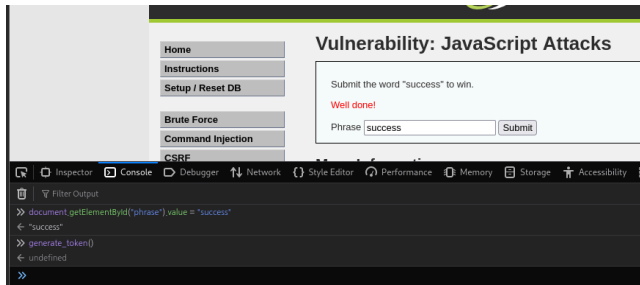
이 함수를 콘솔창에서 수동으로 강제 실행하여 처음 1회만 실행된걸 해쉬값을바꿔준뒤에 갱신

여기서 드는 의문점?

입력창에 값을 success로 넣으라는 코드인데 그럼 동작이 웹사이트의 입력창에서 success를 넣는것과 뭐가 다르길래 이렇게 하는건가 하는 의문점

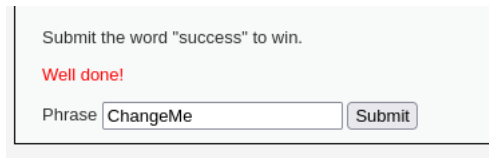
이유: 위의 자바스크립트 코드에서 `generate_token()` 이 부분이 최초 1회 자동으로 실행되는데 그때에 입력창에 있는 changeme의 해시값을 계산해서 토큰칸에 넣어둔채로 새로이 입력받는 값에 대해서 토큰을 새로 갱신해주는 이벤트 리스터 기능이 누락되어 있어서 success를 입력해도 changeme의 해쉬값과 매칭이안되기 때문에 개발자도구로 수정해주어야 함

- 이벤트 리스너(Event Listener)는 웹 페이지에서 일어나는 모든 움직임(이벤트)을 눈을 크게 뜨고 감시하다가, "어떤 사건이 터지는 순간 지정된 자바스크립트 함수를 실행시키는 일종의 '센서'



하고나면 이렇게 입력창에 success가 입력되어있고 해쉬값이 갱신되어있는 상태가 됨.

그대로 써밋누르기.



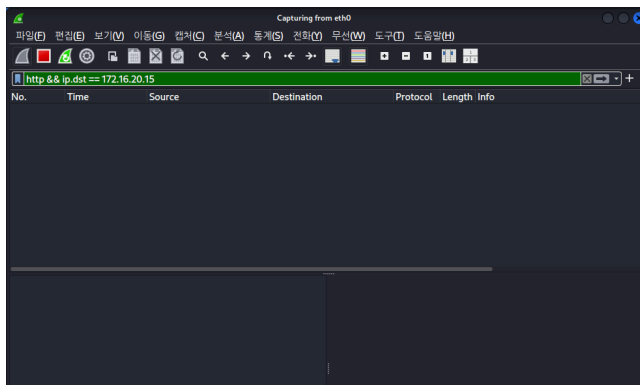
Well done! 이 출력되면 문제풀이성공.

이 실습으로 알 수 있는 취약점

1. 데이터의 검증 및 보안로직이 웹사이트단에서 수행된다는것
--> 중요 데이터를 가공하거나 검증하는 로직이 사용자 브라우저에서 개발자도구를 열어 수정이 가능한 점.
2. 서버가 토큰을 직접 생성해서 사용자에게 발급해주는것이 아닌 브라우저가 스스로 토큰을 계산해서 서버로 올리도록 만든 점
--> 브라우저 내부에서 토큰을 만드는 과정이 개발자도구를 통해 투명하게 보임 (rot13후 md5) 즉, 공격자가 이를 역추적하거나 직접 실행해서 서버가 신뢰하는 토큰을 위조 가능.
3. 입력한 텍스트에 1ㄷ1로 대응하는 고정된 형태의 토큰 구조를 가졌으며, 사용자가 입력값을 변경할때마다 실시간으로 발급해주거나 상호 검증해주는 동적장치가 없음.
--> 사용자가 입력창에 글자를 직접 바꿨음에도 자바스크립트가 동작하지 않아 과거의 토큰이 그대로 전송되는 허점 존재.

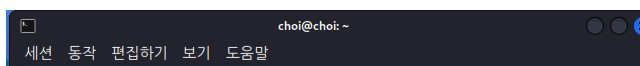
문제풀이는 끝났지만 이제 해커의 입장에서 자동화 코드를 통해 해당 사이트의 취약점을 이용해 우회하여 토큰값을 탈취해보자

와이어샹크로 패킷캡처준비



http && ip.dst == 172.16.20.15

칼리리눅스에서 파이썬 프로그램 실행



```

zsh: locking failed for /home/choi/.zsh_history: 읽기전용 파일 시스템: readin
g anyway
(choi@choi)-[~]
└─$ sudo python3 /home/choi/java.py
[sudo] choi 암호:
[*] 1단계: 트키 리눅스 DVWA 서버 접속 및 CSRF 토큰 수신 중 ...
[+] CSRF 토큰 획득 완료: 294223eb5c58967bfbfa22eb4e630fe6
[*] 2단계: 자동화 계정 인증 시도 중 ...
[+] 로그인 인증 성공
[+] 타겟 페이지 보안 단계를 'Low'로 가공 완료.
[*] 5단계: 우회 페이로드(Payload) 연산 성공:
- 전송할 평문 (phrase) : success
- 위조된 토큰 (token) : 38581812b435834ebf84ebcc2c6424d6
[*] 6단계: 브라우저를 우회하여 조작된 최종 패킷 전송 중 ...

공격 성공: 서버로부터 'Well done!' 응답 확인.

(choi@choi)-[~]

```

Capturing from eth0

파일(F) 편집(E) 보기(V) 이동(M) 검색(S) 통계(S) 전화(N) 무선(W) 도구(T) 도움말(H)

http://ip:port==172.16.20.15

No.	Time	Source	Destination	Protocol	Length	Info
34603	00.406267244	172.16.129.112	172.16.20.15	HTTP	227	GET /dwa/login.php HTTP/1.1
34610	00.412276558	172.16.129.112	172.16.20.15	HTTP	154	POST /dwa/login.php HTTP/1.1
34615	00.417901986	172.16.129.112	172.16.20.15	HTTP	294	GET /dwa/index.php HTTP/1.1
34625	00.419986163	172.16.129.112	172.16.20.15	HTTP	131	POST /dwa/vulnerabilities/

Frame 34603: Packet, 227 bytes on wire (1816 bits)

Ethernet II, Src: VMware 51:2b:21 (00:0c:29:51:2b:21), Dst: 08:00:0c:29:51:2b:21

Internet Protocol Version 4, Src: 172.16.129.112, Dst: 172.16.20.15

Transmission Control Protocol, Src Port: 34768, Dst Port: 80

Hypertext Transfer Protocol

0000 00 0c 29 19 b8 cc 00 0c 29 51 2b 21 08 00 45
0010 08 05 ab ae 40 00 40 06 a0 d4 ac 10 81 70 ac
0020 14 0f 87 d0 00 50 09 4f 50 86 5a 14 4d 97 88
0030 00 76 ee 67 00 00 01 01 08 0a cf b1 6d c7 5a
0040 02 99 47 45 54 20 2f 64 76 77 61 2f 6c 6f 67
0050 06 26 70 68 70 20 48 54 54 50 2f 51 2e 31 00
0060 40 6f 73 74 3a 20 31 37 32 2e 31 30 2e 32 30
0070 31 35 0d 0a 55 73 65 72 2d 41 67 65 6e 74 3a
0080 70 79 74 68 6f 6e 2d 72 65 71 75 65 73 74 73
0090 32 2e 33 32 2e 35 0d 0a 41 63 63 65 70 74 2d
00a0 06 63 6f 64 69 6e 67 3a 20 67 7a 69 70 2c 20
00b0 65 66 6c 61 74 65 2c 20 62 72 6d 0a 41 63 63
00c0 70 74 3a 20 2a 2f 2a 0d 0a 43 6f 6e 65 63
00d0 69 6f 6e 3a 20 6b 65 65 70 2d 61 6c 69 76 65
00e0 0a 0d 0a

패킷 확인

34625 00.419986163 172.16.129.112 172.16.20.15 HTTP 131 POST /dwa/vulnerabilities/

Frame 34625: Packet, 131 bytes on wire (1048 bits)

Ethernet II, Src: VMware 51:2b:21 (00:0c:29:51:2b:21), Dst: 08:00:0c:29:51:2b:21

Internet Protocol Version 4, Src: 172.16.129.112, Dst: 172.16.20.15

Transmission Control Protocol, Src Port: 34768, Dst Port: 80

[2 Reassembled TCP Segments (395 bytes): #34624(338)]

Hypertext Transfer Protocol

HTML Form URL Encoded: application/x-www-form-urlencoded

Form item: "phrase" = "success"

Form item: "token" = "38581812b435834ebf84ebcc2c6"

Form item: "send" = "Submit"

내용에 해커에 의해 조작된 문자열확인

