

PROOFSCAN 0.1.0

Web Application Security Assessment

http://127.0.0.1:8080

Scan run	22 October 2026, 02:38 UTC, took 2m 51s
Report generated	22 October 2026, 17:39
Coverage	48 pages, 20 injection points, 368 requests
Findings proved	3 High 3 Medium
Tests run as	authenticated

Every finding in this report was proved against the running application. Nothing was inferred from a version number or a banner. Candidates that could not be proved are listed too, with what was tried, in *Candidates thrown out*.

Summary



The scan reached 48 pages and found 20 places where input goes into the application. Stage one flagged 12 of them as suspicious. Stage two proved 6 and threw out 6.

Severity	Count	
High	3	
Medium	3	

#	Finding	Where	CVSS	CWE
1	SQL injection	GET http://127.0.0.1:8080/vulnerabilities/brute/ [username]	8.8	CWE-89
2	SQL injection	GET http://127.0.0.1:8080/vulnerabilities/sqli/ [id]	8.8	CWE-89
3	SQL injection	GET http://127.0.0.1:8080/vulnerabilities/sqli_blind/ [id]	8.8	CWE-89
4	Cross site scripting	GET http://127.0.0.1:8080/vulnerabilities/xss_r/ [name]	5.4	CWE-79
5	Cross site scripting	POST http://127.0.0.1:8080/vulnerabilities/xss_s/ [mtxMessage]	5.4	CWE-79
6	Cross site scripting	POST http://127.0.0.1:8080/vulnerabilities/xss_s/ [txtName]	5.4	CWE-79

How these were proved

The true and false test

A condition that is always true and one that is always false are sent in the same place, identical to each other in every way except the condition. If the two responses come back meaningfully different, the database evaluated the input rather than storing it as text. How different they have to be is measured against how much the page disagrees with itself when asked the same question three times, so a page that changes on its own sets its own bar. Anything that looks like a hit is asked a second time, because a real difference repeats.

The timing test

For injection that changes nothing on the page. Two payloads both carry a delay, one behind a true condition and one behind a false one, so they are the same length and take the same path through the parser. Confirmation needs both of two things: the medians apart by most of the delay asked for, and the slowest response to the false condition still quicker than the fastest response to the true one. Random slowness lands in both sets and makes those ranges overlap. A real delay never does.

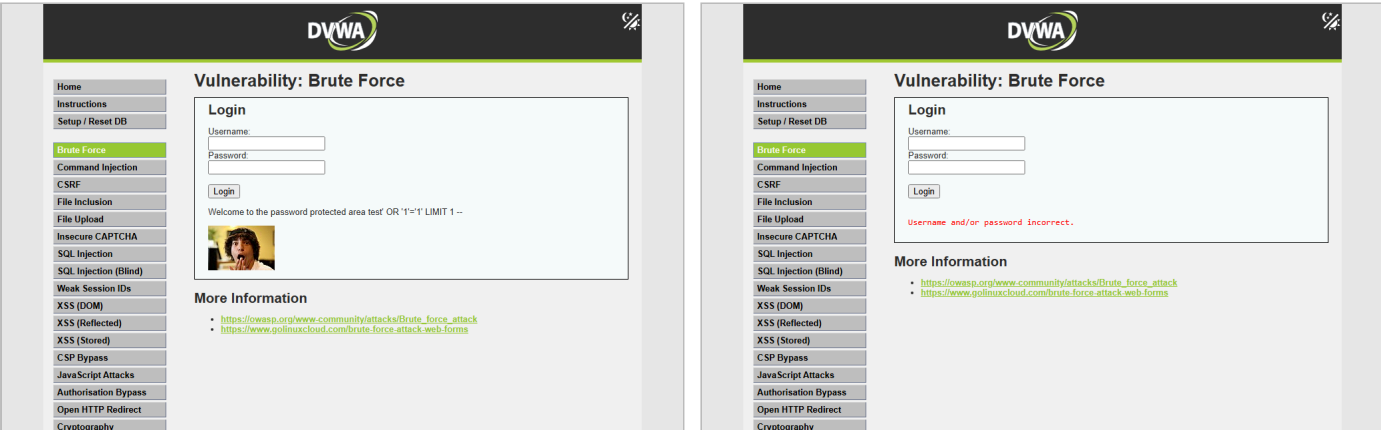
The browser test

Reflection is not execution. A page can hand input straight back and still be safe, because it escaped the brackets or because a Content-Security-Policy header stops the script running. So the response body is not read at all. A payload sets one variable to a value generated for that single attempt, the page is loaded in headless Chromium, and the browser is asked what that variable holds. Nothing else can set it.

Confirmed findings

1. SQL injection 8.8 High	
GET http://127.0.0.1:8080/vulnerabilities/brute/ parameter username (form)	
true and false conditions gave different responses, twice (similarity 0.9948 then 0.9948, page noise floor 1.0000).	
CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Weakness	CWE-89 – Improper Neutralization of Special Elements used in an SQL Command
OWASP Top 10 2021	A03:2021 Injection
Privileges required	L, because the scanner had to log in to reach this point
THE PROOF	
Payload, true condition	test' OR '1'='1' LIMIT 1 --
Payload, false condition	test' OR '1'='2' LIMIT 1 --
Response to true	HTTP 200, 4570 bytes
Response to false	HTTP 200, 4509 bytes
Similarity	0.9948, and 0.9948 when the same pair was sent again
The page's own noise	it scores 1.0000 against itself, so a pair had to come in under 0.9980 to count

WHAT THE BROWSER SHOWED



True condition: test' OR '1'='1' LIMIT 1 --

False condition: test' OR '1'='2' LIMIT 1 --

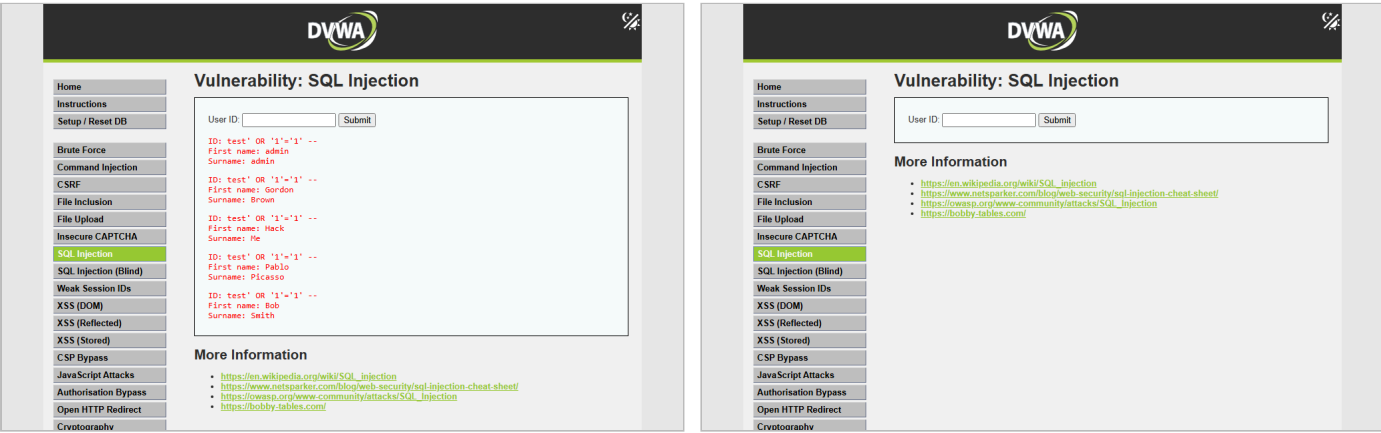
What this does not prove. Confidentiality, integrity and availability are the impacts normally assigned to SQL injection. ProofScan proves the database evaluates injected input. It does not read the data out, write to the database, or take it offline, because a scanner that did those things to prove a point would be causing the damage it reports.

HOW TO FIX IT

Use parameterised queries or prepared statements everywhere, so that input is passed to the database as a value and never assembled into the statement text. Escaping and blocklists are worked around routinely and are not a substitute. Where a query has to take an identifier such as a column or table name, check it against a fixed list of permitted values rather than escaping it. Give the application's database account only the rights it needs, so that an injection that is missed costs less.

2. SQL injection 8.8 High	
GET http://127.0.0.1:8080/vulnerabilities/sqli/ parameter id (form)	
true and false conditions gave different responses, twice (similarity 0.9687 then 0.9687, page noise floor 1.0000).	
CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Weakness	CWE-89 – Improper Neutralization of Special Elements used in an SQL Command
OWASP Top 10 2021	A03:2021 Injection
Privileges required	L, because the scanner had to log in to reach this point
THE PROOF	
Payload, true condition	test' OR '1'='1' --
Payload, false condition	test' OR '1'='2' --
Response to true	HTTP 200, 4979 bytes
Response to false	HTTP 200, 4592 bytes
Similarity	0.9687, and 0.9687 when the same pair was sent again
The page's own noise	it scores 1.0000 against itself, so a pair had to come in under 0.9980 to count

WHAT THE BROWSER SHOWED



True condition: test' OR '1'='1' --

False condition: test' OR '1'='2' --

What this does not prove. Confidentiality, integrity and availability are the impacts normally assigned to SQL injection. ProofScan proves the database evaluates injected input. It does not read the data out, write to the database, or take it offline, because a scanner that did those things to prove a point would be causing the damage it reports.

HOW TO FIX IT

Use parameterised queries or prepared statements everywhere, so that input is passed to the database as a value and never assembled into the statement text. Escaping and blocklists are worked around routinely and are not a substitute. Where a query has to take an identifier such as a column or table name, check it against a fixed list of

permitted values rather than escaping it. Give the application's database account only the rights it needs, so that an injection that is missed costs less.

3. SQL injection 8.8 High

GET http://127.0.0.1:8080/vulnerabilities/sqli_blind/ parameter **id** (form)

true condition took 10001 ms longer than false, every time, over 10 samples each.

CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:H
Weakness	CWE-89 – Improper Neutralization of Special Elements used in an SQL Command
OWASP Top 10 2021	A03:2021 Injection
Privileges required	L, because the scanner had to log in to reach this point

THE PROOF

Payload, true condition	test' OR IF(1=1,SLEEP(2),0) --
Payload, false condition	test' OR IF(1=2,SLEEP(2),0) --
Median response time	10007 ms against 6 ms, a gap of 10001 ms
Delay asked for	2000 ms
Ranges	fastest response to the true condition 10006 ms, slowest to the false one 6 ms, so the two never overlap
Samples	10 of each

What this does not prove. Confidentiality, integrity and availability are the impacts normally assigned to SQL injection. ProofScan proves the database evaluates injected input. It does not read the data out, write to the database, or take it offline, because a scanner that did those things to prove a point would be causing the damage it reports.

HOW TO FIX IT

Use parameterised queries or prepared statements everywhere, so that input is passed to the database as a value and never assembled into the statement text. Escaping and blocklists are worked around routinely and are not a substitute. Where a query has to take an identifier such as a column or table name, check it against a fixed list of permitted values rather than escaping it. Give the application's database account only the rights it needs, so that an injection that is missed costs less.

4. Cross site scripting 5.4 Medium

GET http://127.0.0.1:8080/vulnerabilities/xss_r/ parameter **name** (form)

the injected script ran in a real browser.

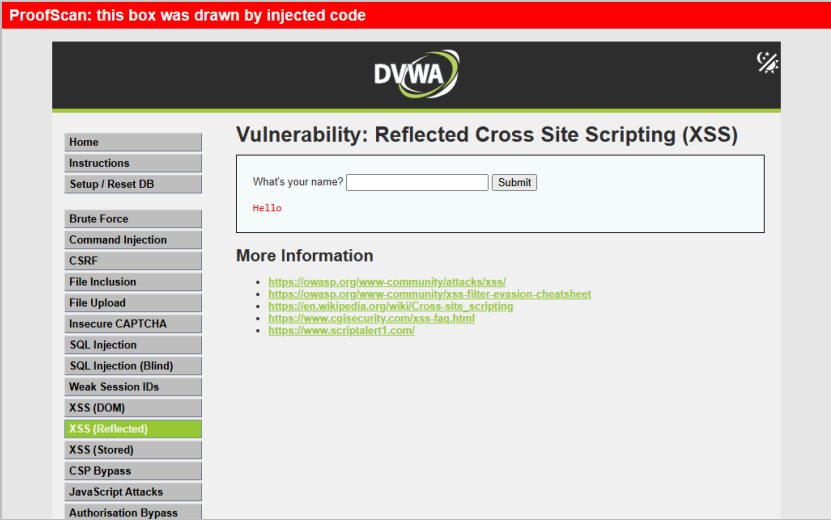
CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N
Weakness	CWE-79 – Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
OWASP Top 10 2021	A03:2021 Injection
Privileges required	L, because the scanner had to log in to reach this point

THE PROOF

Payload	<script>window.__proofscan_xss="pf409c26dc592aa798"</script>
---------	--

What it sets	window.__proofscan_xss, to a value generated for this one attempt and nothing else
Value sent	pf409c26dc592aa798
Value the browser gave back	pf409c26dc592aa798

WHAT THE BROWSER SHOWED



The injected script ran and drew the red box (pfb08b9016ab14509c)

What this does not prove. The usual metrics for cross-site scripting. Scope is Changed because the injected script runs in the browser rather than in the vulnerable application, and user interaction is Required because somebody has to load the page. ProofScan proves the script runs. It does not steal a session or act as the visitor.

HOW TO FIX IT

Encode output for the context it lands in: html body, attribute, javascript and url each need different treatment, and a template engine that escapes by default handles most of it. Validate input on the way in as well, but treat that as a second line rather than the fix. Add a Content-Security-Policy that refuses inline scripts, which limits the damage of anything that gets through, and set HttpOnly on session cookies so a script that does run cannot read them.

5. Cross site scripting 5.4 Medium

POST http://127.0.0.1:8080/vulnerabilities/xss_s/ parameter mtxMessage (form)
the injected script ran in a real browser.

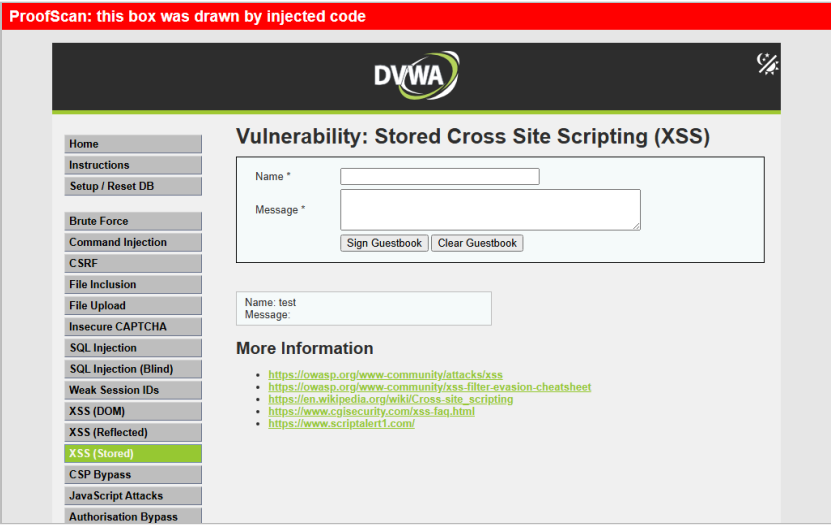
CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N
Weakness	CWE-79 – Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
OWASP Top 10 2021	A03:2021 Injection
Privileges required	L, because the scanner had to log in to reach this point

THE PROOF

Payload	<script>>window.__proofscan_xss="pfe55e708063df26dd"</script>
What it sets	window.__proofscan_xss, to a value generated for this one attempt and nothing else
Value sent	pfe55e708063df26dd

Value the browser gave back pfe55e708063df26dd

WHAT THE BROWSER SHOWED



The injected script ran and drew the red box (pf4be7e11c0842b86f)

What this does not prove. The usual metrics for cross-site scripting. Scope is Changed because the injected script runs in the browser rather than in the vulnerable application, and user interaction is Required because somebody has to load the page. ProofScan proves the script runs. It does not steal a session or act as the visitor.

HOW TO FIX IT

Encode output for the context it lands in: html body, attribute, javascript and url each need different treatment, and a template engine that escapes by default handles most of it. Validate input on the way in as well, but treat that as a second line rather than the fix. Add a Content-Security-Policy that refuses inline scripts, which limits the damage of anything that gets through, and set HttpOnly on session cookies so a script that does run cannot read them.

6. Cross site scripting 5.4 Medium

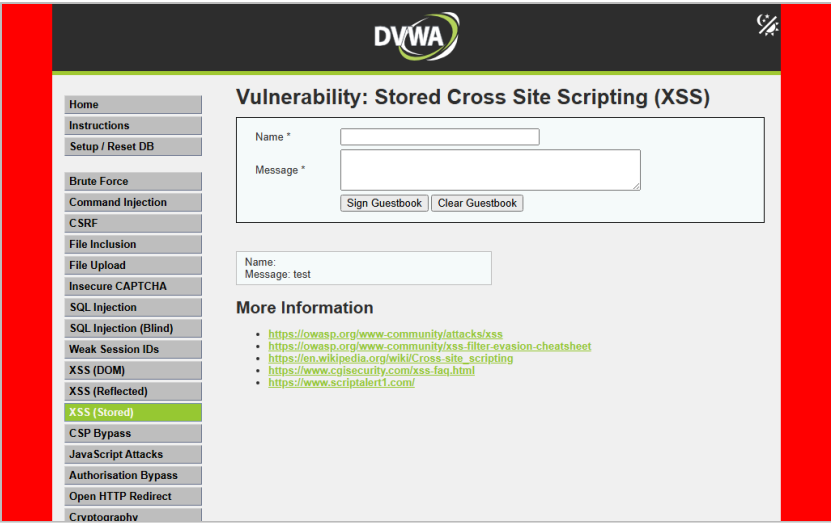
POST http://127.0.0.1:8080/vulnerabilities/xss_s/ parameter `txtName` (form)
the injected script ran in a real browser.

CVSS v3.1	CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N
Weakness	CWE-79 – Improper Neutralization of Input During Web Page Generation (Cross-site Scripting)
OWASP Top 10 2021	A03:2021 Injection
Privileges required	L, because the scanner had to log in to reach this point

THE PROOF

Payload	<code><script>window.__proofscan_xss="pf7acf01a0b12aea1a"</script></code>
What it sets	<code>window.__proofscan_xss</code> , to a value generated for this one attempt and nothing else
Value sent	pf7acf01a0b12aea1a
Value the browser gave back	pf7acf01a0b12aea1a

WHAT THE BROWSER SHOWED



The injected script ran and turned the page red (pfedec89217f228c31)

What this does not prove. The usual metrics for cross-site scripting. Scope is Changed because the injected script runs in the browser rather than in the vulnerable application, and user interaction is Required because somebody has to load the page. ProofScan proves the script runs. It does not steal a session or act as the visitor.

HOW TO FIX IT

Encode output for the context it lands in: html body, attribute, javascript and url each need different treatment, and a template engine that escapes by default handles most of it. Validate input on the way in as well, but treat that as a second line rather than the fix. Add a Content-Security-Policy that refuses inline scripts, which limits the damage of anything that gets through, and set HttpOnly on session cookies so a script that does run cannot read them.

Candidates thrown out

6 of the 12 suspected points were disproved and do not appear as findings. Each one is a false alarm that a scanner reporting on the response alone would have handed over for somebody to check by hand. What was tried against each is listed, because a false alarm count is only worth something with the work behind it shown.

1. SQL injection not proved

GET http://127.0.0.1:8080/vulnerabilities/fi/ parameter page

Stage one flagged it	response changed when a quote was added
Thrown out because	no true/false pair beat the page's own noise (floor 1.0000)

What was sent	What came back
file3.php AND 1=1 / file3.php AND 1=2	no more different than the page is from itself (similarity 1.0000)
file3.php' AND '1'='1 / file3.php' AND '1'='2	no more different than the page is from itself (similarity 1.0000)
file3.php' OR '1'='1' -- / file3.php' OR '1'='2' --	no more different than the page is from itself (similarity 1.0000)
file3.php OR 1=1 / file3.php OR 1=2	no more different than the page is from itself (similarity 1.0000)

file3.php' OR '1'='1' LIMIT 1 -- / file3.php' OR '1'='2' LIMIT 1 --	no more different than the page is from itself (similarity 1.0000)
file3.php OR 1=1 LIMIT 1 / file3.php OR 1=2 LIMIT 1	no more different than the page is from itself (similarity 1.0000)

2. SQL injection not proved

GET http://127.0.0.1:8080/instructions.php parameter **doc**

Stage one flagged it	response changed when a quote was added
Thrown out because	no true/false pair beat the page's own noise (floor 1.0000)

What was sent	What came back
copying AND 1=1 / copying AND 1=2	no more different than the page is from itself (similarity 1.0000)
copying' AND '1'='1' / copying' AND '1'='2	no more different than the page is from itself (similarity 1.0000)
copying' OR '1'='1' -- / copying' OR '1'='2' --	no more different than the page is from itself (similarity 1.0000)
copying OR 1=1 / copying OR 1=2	no more different than the page is from itself (similarity 1.0000)
copying' OR '1'='1' LIMIT 1 -- / copying' OR '1'='2' LIMIT 1 --	no more different than the page is from itself (similarity 1.0000)
copying OR 1=1 LIMIT 1 / copying OR 1=2 LIMIT 1	no more different than the page is from itself (similarity 1.0000)

3. SQL injection not proved

GET http://127.0.0.1:8080/vulnerabilities/open_redirect/source/info.php parameter **id**

Stage one flagged it	status changed 200 to 500
Thrown out because	no true/false pair beat the page's own noise (floor 1.0000)

What was sent	What came back
2 AND 1=1 / 2 AND 1=2	no more different than the page is from itself (similarity 1.0000)
2' AND '1'='1' / 2' AND '1'='2	no more different than the page is from itself (similarity 1.0000)
2' OR '1'='1' -- / 2' OR '1'='2' --	no more different than the page is from itself (similarity 1.0000)
2 OR 1=1 / 2 OR 1=2	no more different than the page is from itself (similarity 1.0000)

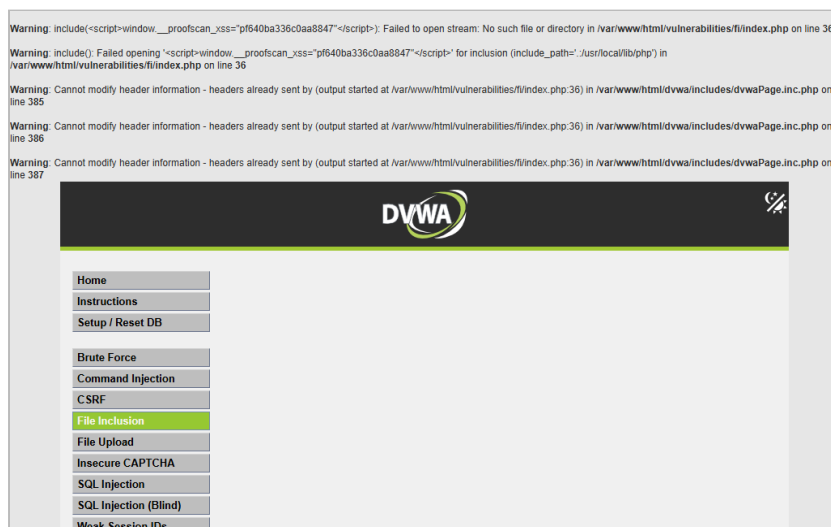
2' OR '1'='1' LIMIT 1 -- / 2' OR '1'='2' LIMIT 1 --	no more different than the page is from itself (similarity 1.0000)
2 OR 1=1 LIMIT 1 / 2 OR 1=2 LIMIT 1	no more different than the page is from itself (similarity 1.0000)

4. Cross site scripting not proved

GET http://127.0.0.1:8080/vulnerabilities/fi/ parameter **page**

Stage one flagged it	input is reflected and ' came back unescaped
Thrown out because	input comes back, but none of the 5 payloads executed in a browser

What was sent	What came back
<script>window.__proofscan_xss="pf640ba336c0aa8847"</script>	loaded, but nothing ran
"><script>window.__proofscan_xss="pf7a155d2cd06a5063"</script>	loaded, but nothing ran
'><script>window.__proofscan_xss="pffcc321a0213ee430"</script>	loaded, but nothing ran
	loaded, but nothing ran
" onfocus=window.__proofscan_xss='pf4322fe7d73d44c33' autofocus x="	loaded, but nothing ran



How the browser showed the page with the first payload in it. Nothing ran.

5. Cross site scripting not proved

POST http://127.0.0.1:8080/vulnerabilities/csp/ parameter **include**

Stage one flagged it	input is reflected and <>" came back unescaped
Thrown out because	input comes back, but none of the 5 payloads executed in a browser

What was sent	What came back
<script>window.__proofscan_xss="pf714312ac885832ac"</script>	loaded, but nothing ran
"><script>window.__proofscan_xss="pfb07e61b7c6aa03a"</script>	loaded, but nothing ran
'><script>window.__proofscan_xss="pf2bd3df48a85caea7"</script>	loaded, but nothing ran

	loaded, but nothing ran
" onfocus=window.__proofscan_xss='pff76840b304d1d510' autofocus x="	loaded, but nothing ran



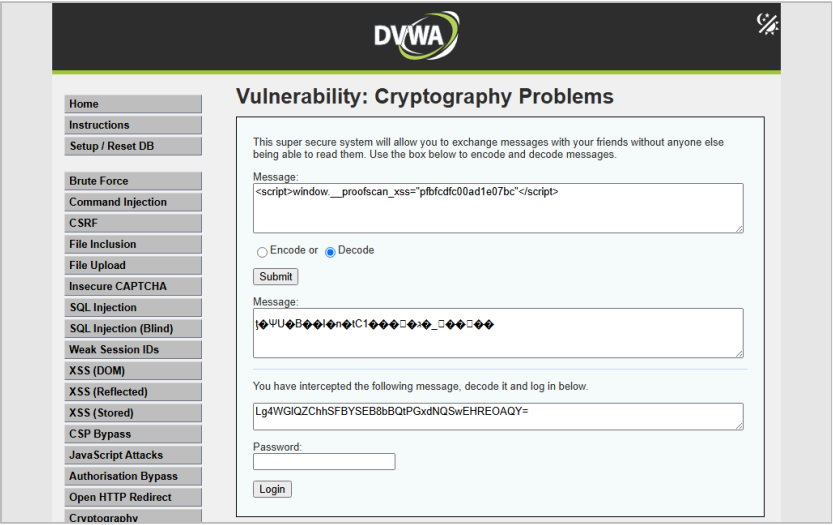
How the browser showed the page with the first payload in it. Nothing ran.

6. Cross site scripting not proved

POST http://127.0.0.1:8080/vulnerabilities/cryptography/index.php parameter message

Stage one flagged it	input is reflected, but the interesting characters came back escaped
Thrown out because	input comes back, but none of the 5 payloads executed in a browser

What was sent	What came back
<script>window.__proofscan_xss="pfbfcdfc00ad1e07bc"</script>	loaded, but nothing ran
"><script>window.__proofscan_xss="pf8ea53f8f9e44f998"</script>	loaded, but nothing ran
'><script>window.__proofscan_xss="pf51d432cc4b37088c"</script>	loaded, but nothing ran
	loaded, but nothing ran
" onfocus=window.__proofscan_xss='pf006d1c5608fac359' autofocus x="	loaded, but nothing ran



How the browser showed the page with the first payload in it. Nothing ran.

Scope and limitations

- Only SQL injection and cross site scripting were looked for. Command injection, file inclusion, insecure direct object references, cross site request forgery and open redirects are outside this version's scope, and a page carrying one of those would be reported clean here.
- DOM based cross site scripting is not detected. The check looks at what the server sends, and a payload handled entirely in javascript never appears there.
- Pages that change how the target behaves were left alone on purpose: logout, setup, and security or settings pages. They are listed in the appendix. A scan that alters what it is measuring proves nothing about what it measured afterwards.
- Nothing here was proved by reading a version number or a banner. Every confirmed finding was demonstrated against the running application.
- The scan was authenticated, so Privileges Required is scored as Low on every finding. That is a simplification: a reflected cross site scripting bug behind a login can still be reached by someone with no account, by sending the link to a person who has one.

Appendix: what was reached

PAGES CRAWLED (48)

http://127.0.0.1:8080	http://127.0.0.1:8080/instructions.php?doc=copying
http://127.0.0.1:8080/	http://127.0.0.1:8080/README.ar.md
http://127.0.0.1:8080/instructions.php	http://127.0.0.1:8080/README.zh.md
http://127.0.0.1:8080/vulnerabilities/brute/	http://127.0.0.1:8080/README.fr.md
http://127.0.0.1:8080/vulnerabilities/exec/	http://127.0.0.1:8080/README.ko.md
http://127.0.0.1:8080/vulnerabilities/csrf/	http://127.0.0.1:8080/README.fa.md
http://127.0.0.1:8080/vulnerabilities/fi/?page=include.php	http://127.0.0.1:8080/README.pl.md
http://127.0.0.1:8080/vulnerabilities/upload/	http://127.0.0.1:8080/README.pt.md
http://127.0.0.1:8080/vulnerabilities/captcha/	http://127.0.0.1:8080/README.es.md
http://127.0.0.1:8080/vulnerabilities/sqli/	http://127.0.0.1:8080/README.tr.md
http://127.0.0.1:8080/vulnerabilities/sqli_blind/	http://127.0.0.1:8080/README.id.md
http://127.0.0.1:8080/vulnerabilities/weak_id/	http://127.0.0.1:8080/README.vi.md
http://127.0.0.1:8080/vulnerabilities/xss_d/	http://127.0.0.1:8080/README.it.md
http://127.0.0.1:8080/vulnerabilities/xss_r/	http://127.0.0.1:8080/README.uk.md
http://127.0.0.1:8080/vulnerabilities/xss_s/	http://127.0.0.1:8080/README.ru.md
http://127.0.0.1:8080/vulnerabilities/csp/	http://127.0.0.1:8080/compose.yml
http://127.0.0.1:8080/vulnerabilities/javascript/	http://127.0.0.1:8080/vulnerabilities/fi/?page=file1.php
http://127.0.0.1:8080/vulnerabilities/authbypass/	http://127.0.0.1:8080/vulnerabilities/fi/?page=file2.php
http://127.0.0.1:8080/vulnerabilities/open_redirect/	http://127.0.0.1:8080/vulnerabilities/fi/?page=file3.php
http://127.0.0.1:8080/vulnerabilities/cryptography/	http://127.0.0.1:8080/vulnerabilities/open_redirect/source/info.php?id=1
http://127.0.0.1:8080/vulnerabilities/api/	http://127.0.0.1:8080/vulnerabilities/open_redirect/source/info.php?id=2
http://127.0.0.1:8080/phpinfo.php	http://127.0.0.1:8080/docs/DVWA_v1.3.pdf
http://127.0.0.1:8080/about.php	
http://127.0.0.1:8080/instructions.php?doc=readme	
http://127.0.0.1:8080/instructions.php?doc=PDF	
http://127.0.0.1:8080/instructions.php?doc=changelog	

LEFT ALONE ON PURPOSE (6)

In scope and reachable, but requesting them would have changed how the target behaves and made everything scanned afterwards meaningless.

http://127.0.0.1:8080/setup.php	http://127.0.0.1:8080/config/config.inc.php.dist
http://127.0.0.1:8080/security.php	http://127.0.0.1:8080/vulnerabilities/csrf/
http://127.0.0.1:8080/logout.php	http://127.0.0.1:8080/vulnerabilities/captcha/