GGSec Cortex · GG Advanced IT Security

Security Assessment Report

GGSec Cortex v1.0 · AI-Guided DAST · Context-Aware SAST · Target: ggsec-cortex-debugger-lab-full.scanplan.json · Generated: 2026-08-03 17:27

CONFIDENTIAL

Executive Summary

GGSec Cortex identified 14 confirmed and 2 likely vulnerabilities. Combined exploitation enables an unauthenticated attacker to achieve full administrative takeover. Every confirmed finding is evidence-backed (live proof-of-concept or a baseline-difference control), so false positives are minimised. **Immediate remediation is recommended.**

Intentionally vulnerable PHP lab application with SQL injection, reflected & stored XSS, IDOR, open redirect, path traversal, CSRF/mass assignment, session fixation, and information disclosure vulnerabilities across multiple endpoints.

Security Rating	**F · HIGH (4 High)**
Max CVSS (v3.1)	**8,9 (High)**
Severity distribution	Critical: 0 High: 4 Medium: 14 Low: 2
Compromise probability	Very High (~95-100%)
Confirmed findings	**14**
Likely (needs review)	**2**
Business impact	**Account / administrator takeover**
Exploitation complexity	Low — reachable by a remote attacker
Likelihood	High — confirmed with a live proof-of-concept
Scan	SAST 0s · DAST 2s · 64 requests

Assessment Scope

Target	http://192.168.0.25/ggsec-cortex-debugger-lab/
Base URL	http://192.168.0.25/ggsec-cortex-debugger-lab/
Analysis source	Cached SAST scan plan (replayed)
Fresh SAST this run	No (cached scan plan replayed)
DAST executed	Yes
Scan plan	ggsec-cortex-debugger-lab-full.scanplan.json
Endpoints tested	12
Active DAST probes	64
Total HTTP exchanges	170 (crawl + baseline + control + probes)
HTTP methods	GET, GET->POST->GET, POST, POST+GET
Vulnerability classes	14
Authentication	Form login
Scan duration	SAST 0s · DAST 2s
Completed	2026-08-03 17:27

Scan Timeline

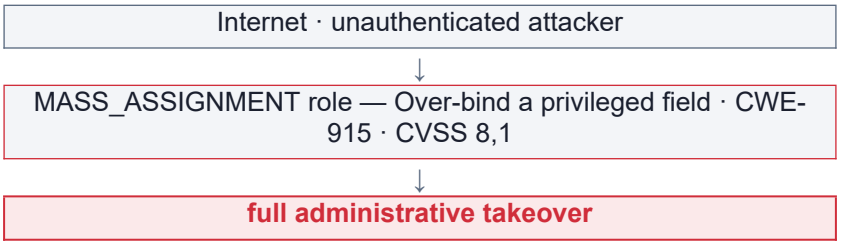
17:27:34 ● **Scan started**

17:27:34 ○ SAST replay (cached plan)

17:27:34 ○ DAST probing started
17:27:34 ○ GGC-SESSFIX-85F confirmed — SESSION_FIXATION PHPSESSID
17:27:34 ○ GGC-XSS-CE8 confirmed — XSS q
17:27:35 ○ GGC-SQLI-8D5 confirmed — SQL_QUERY q
17:27:35 ○ GGC-MASSASGN-993 confirmed — MASS_ASSIGNMENT role
17:27:35 ○ GGC-XSS-1B2 confirmed — XSS bio
17:27:35 ○ GGC-IDOR-AC7 confirmed — IDOR id
17:27:35 ○ GGC-IDOR-D48 confirmed — IDOR id
17:27:35 ○ GGC-IDOR-7BD confirmed — IDOR id
17:27:35 ● **Attack chain verified end-to-end → full administrative takeover**
17:27:35 ○ GGC-REDIR-FD7 confirmed — OPEN_REDIRECT next
17:27:35 ○ GGC-PATH-32F confirmed — PATH_TRAVERSAL file
17:27:35 ○ GGC-PROXYHDR-280 confirmed — UNTRUSTED_PROXY_HEADER X-Forwarded-For
17:27:36 ○ GGC-SQLI-A15 confirmed — SQL_QUERY username
17:27:36 ○ GGC-BIZLOGIC-0A0 confirmed — BUSINESS_LOGIC qty
17:27:36 ○ GGC-BIZLOGIC-B64 confirmed — BUSINESS_LOGIC discount
17:27:36 ○ DAST completed (2s, 64 requests)
17:27:36 ● **Report generated**

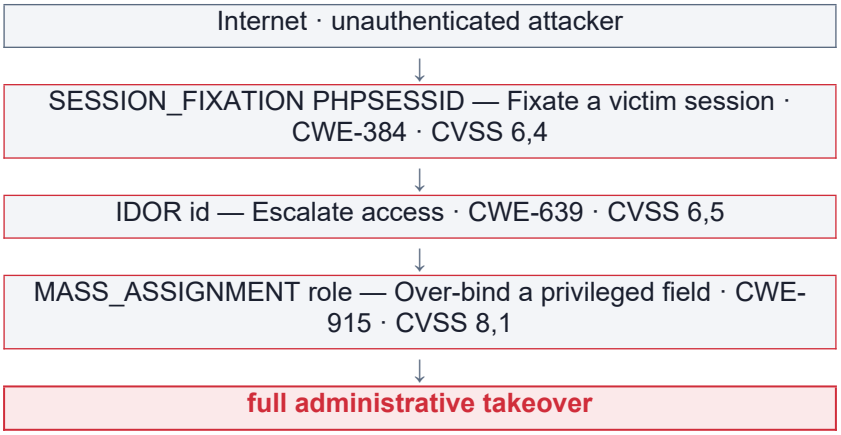
Exploitation Paths

Goal: full administrative takeover · Chain severity 9,4 (worst single 8,1) · Likelihood: High



Authenticated low-privilege user → over-bind a privileged field via MASS_ASSIGNMENT on 'role' (CWE-915) □ full administrative takeover — executed in one session and the final state (the account's role became admin) was read back.

Goal: full administrative takeover · Chain severity 10,0 (worst single 8,1) · Likelihood: High



Unauthenticated attacker → 1. fixate a victim session via SESSION_FIXATION on 'PHPSESSID' (CWE-384) → 2. escalate access via IDOR on 'id' (CWE-639) → 3. over-bind a privileged field via MASS_ASSIGNMENT on 'role' (CWE-915) □ full administrative takeover.

Risk Matrix

Likelihood \ Impact	Negligible	Minor	Moderate	Major	Severe
Almost certain			3	1	
Likely			4	4	
Possible			2		
Unlikely					
Rare					

Endpoint Summary

Endpoint	Requests	Findings	Risk
search.php	12	SQL_QUERY, XSS	Critical
login.php	11	SESSION_FIXATION, SQL_QUERY	Critical
http://192.168.0.25/ggsec-cortex-debugger-lab/login.php	1	SQL_QUERY	Critical
update_profile.php	1	MASS_ASSIGNMENT	High
http://192.168.0.25/ggsec-cortex-debugger-lab/update_profile.php	1	CSRF	High
download.php	22	PATH_TRAVERSAL	High
debug.php	1	UNTRUSTED_PROXY_HEADER	High
profile.php	5	IDOR, XSS	Medium
http://192.168.0.25/ggsec-cortex-debugger-lab/coupon.php	2	BUSINESS_LOGIC	Medium
api.php	2	HTTP_PARAMETER_POLLUTION, IDOR	Medium
notes.php	1	IDOR	Medium
redirect.php	5	OPEN_REDIRECT	Medium

Remediation Plan

ID	Finding	CVSS	Priority	SLA
GGC-SQLI-A15	SQL_QUERY — password, username	8,2	P1	Within 7 days
GGC-SQLI-8D5	SQL_QUERY — q	8,2	P1	Within 7 days
GGC-MASSASGN-993	MASS_ASSIGNMENT — role	8,1	P1	Within 7 days
GGC-PATH-32F	PATH_TRAVERSAL — file	7,5	P1	Within 7 days
GGC-PROXYHDR-280	UNTRUSTED_PROXY_HEADER — X-Forwarded-For	7,5	P1	Within 7 days
GGC-BIZLOGIC-B64	BUSINESS_LOGIC — discount	6,5	P2	Within 30 days
GGC-BIZLOGIC-0A0	BUSINESS_LOGIC — qty	6,5	P2	Within 30 days
GGC-IDOR-7BD	IDOR — id	6,5	P2	Within 30 days
GGC-IDOR-D48	IDOR — id	6,5	P2	Within 30 days
GGC-IDOR-AC7	IDOR — id	6,5	P2	Within 30 days
GGC-SESSFIX-85F	SESSION_FIXATION — PHPSESSID	6,4	P2	Within 30 days
GGC-REDIR-FD7	OPEN_REDIRECT — next	6,1	P2	Within 30 days
GGC-XSS-1B2	XSS — bio	5,4	P2	Within 30 days
GGC-XSS-CE8	XSS — q	5,4	P2	Within 30 days

Findings (18)

[1] GGC-SQLI-A15 · SQL_QUERY — password, username

CONFIRMED CVSS 8,2 High · CWE-89 SQL Injection (SQLite backend) · Priority P1 (Within 7 days)

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

SQL is built from untrusted input, letting an attacker alter the query to read/modify data or bypass authentication.

Compliance: A03:2021 – Injection · PCI-DSS v4.0 Req 6.2.4 (injection/XSS) · ISO/IEC 27001:2022 A.8.28 · GDPR Art. 32 (security of processing)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: crawl-form: http://192.168.0.25/ggsec-cortex-debugger-lab/login.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: A crawler-discovered POST form (not in the SAST plan) returns a SQL error when a quote is injected into a field.

Evidence strength: Medium (79/100)

+50 Confirmed (strong signal)
+20 Database error echoed in response (query parser reached)
+4 Reproduced by 4 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'http://192.168.0.25/ggsec-cortex-debugger-lab/login.php' --data
'username=%27&password=ggsecfill'
```

Evidence (live response):

```
[CRAWL] POST form field 'username' surfaced a SQL error for a single-quote payload. <!doctype
html><html><head><meta charset="utf-8"><meta name="viewport" content="width=device-width,initial-
scale=1"><title>Login</title><style> body{font-family:system-ui,Arial;margin:2rem…
```

Remediation

Use parameterized queries / prepared statements — never build queries by string concatenation.

- Replace interpolated SQL with bound parameters (PDO/mysqli prepared statements, '?'/named placeholders).
- For NoSQL, cast user input to the expected scalar type and reject array/operator inputs ((string)$x, type checks).
- Apply least-privilege DB credentials; the web user should not own DDL or admin rights.
- Add allowlist validation for structural elements that cannot be parameterized (column/table names, ORDER BY).

References: CWE-89 · OWASP: SQL Injection Prevention Cheat Sheet · CWE-943 (NoSQL)

---

[2] GGC-SQLI-8D5 · SQL_QUERY — q

CONFIRMED CVSS 8,2 High · CWE-89 SQL Injection (SQLite backend) · Priority P1 (Within 7 days)

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

SQL is built from untrusted input, letting an attacker alter the query to read/modify data or bypass authentication.

Compliance: A03:2021 – Injection · PCI-DSS v4.0 Req 6.2.4 (injection/XSS) · ISO/IEC 27001:2022 A.8.28 · GDPR Art. 32 (security of processing)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: search.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Evidence: @ offset 1207 — via sql-error-body: …'</p> SQL error: SQLSTATE[HY000]: General error: 1 unrecognized token: "'&q…

Flow: $_GET['q'] is directly interpolated into a SQL query via string concatenation: "WHERE username LIKE '%$q%' OR display_name LIKE '%$q%' OR bio LIKE '%$q%'". No parameterization or escaping.

Evidence strength: High (94/100)

+50 Confirmed (strong signal)
+20 Database error echoed in response (query parser reached)
+15 Damning token for this class in response ("SQLSTATE")
+4 Reproduced by 4 payload(s)

+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'search.php?q=%27'
```

Evidence (live response):

```
…ed XSS in raw mode --> <p>Results for: &#039;</p> <div class="box"><strong>SQL error:</strong>
SQLSTATE[HY000]: General error: 1 unrecognized token: &quot;&#039;&quot;</div> <hr><small>GGSEC
Cortex Debugger Lab — localhost only</small></body></html>
```

Remediation

Use parameterized queries / prepared statements — never build queries by string concatenation.

- Replace interpolated SQL with bound parameters (PDO/mysqli prepared statements, '?'/named placeholders).
- For NoSQL, cast user input to the expected scalar type and reject array/operator inputs ((string)$x, type checks).
- Apply least-privilege DB credentials; the web user should not own DDL or admin rights.
- Add allowlist validation for structural elements that cannot be parameterized (column/table names, ORDER BY).

References: CWE-89 · OWASP: SQL Injection Prevention Cheat Sheet · CWE-943 (NoSQL)

---

[3] GGC-MASSASGN-993 · MASS_ASSIGNMENT — role

CONFIRMED CVSS 8,1 High · CWE-915 Mass Assignment (Improperly Controlled Object-Attribute Modification) · Priority P1 (Within 7 days)

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

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: update_profile.php

Impact: PRIV_ESC (Direct)

Preconditions: Attacker privileges: User · Victim state: None · User interaction: None (exercised at: Customer)

Evidence: @ offset 122 — via mass-assignment-readback: …n ordinary authenticated request. A read-back after POSTing role=admin (form-urlencoded) shows the persisted value changed from 'u…

Flow: update_profile.php accepts POST without any CSRF token and allows mass assignment of the 'role' field. An attacker can craft a cross-site form that elevates a victim's role to 'admin'. The form also lacks CSRF protection for display_name and bio changes.

Evidence strength: Medium (75/100)

+50 Confirmed (strong signal)
+20 Direct detection (mass-assignment-readback)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'update_profile.php' --data 'role=admin'
```

Evidence (live response):

```
[MASS ASSIGNMENT] The privileged field 'role' is writable by an ordinary authenticated request. A
read-back after POSTing role=admin (form-urlencoded) shows the persisted value changed from 'user'
to 'admin' — the app binds request keys onto the record with no allow-list, so a cl…
```

Remediation

Bind only an explicit allow-list of request fields to the record; never map the whole request body onto the model.

- Define a server-side allow-list of client-updatable fields and assign only those (e.g. $fillable / DTO whitelist).
- Never iterate arbitrary request keys onto an UPDATE / model — reject or ignore unknown/privileged fields (role, is_admin, balance, permissions).
- Authorize privileged changes separately (an admin-only endpoint), and validate/cast each value to its expected type.

References: CWE-915 · OWASP API3:2023 Broken Object Property Level Authorization · OWASP: Mass Assignment Cheat Sheet

[4] GGC-PATH-32F · PATH_TRAVERSAL — file

CONFIRMED CVSS 7,5 High · CWE-22 Path Traversal / Arbitrary File Read · Priority P1 (Within 7 days)

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

User-controlled path components escape the intended directory, exposing arbitrary files (/etc/passwd, config, source).

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1083 File and Directory Discovery · T1005 Data from Local System

Location: download.php

Impact: FILE_READ (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: $_GET['file'] is appended to __DIR__.'/files/' and passed to realpath() then readfile(). The code checks that the resolved path starts with __DIR__ (the lab root), but any file within the lab directory tree is accessible — including data/lab.sqlite (the database with plaintext passwords and api_keys). A traversal like '../data/lab.sqlite' resolves inside __DIR__ and passes the check.

Evidence strength: Medium (77/100)

+50 Confirmed (strong signal)
+20 Direct detection (indicator)
+2 Reproduced by 2 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'download.php?file=..%2Fbootstrap.php'
```

Evidence (live response):

```
<?php declare(strict_types=1); session_start(); $dbPath = __DIR__ . '/data/lab.sqlite'; $firstRun = !file_exists($dbPath); $db = new PDO('sqlite:' . $dbPath); $db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION); if ($firstRun) { $db->exec(" CREATE TABLE u…
```

Remediation

Never build filesystem paths from user input — map requests to a fixed allowlist and canonicalize before use.

- Map the user-supplied identifier to a server-side allowlist of permitted files; never use it as a path directly.
- Canonicalize the resolved path (realpath) and verify it is still inside the intended base directory before reading.
- Reject traversal sequences AFTER decoding (../, ..\, encoded %2e%2e, and stripped variants like ..../).
- Run with least privilege so sensitive files (/etc/passwd, app secrets) are unreadable by the web user.

References: CWE-22 · CWE-73 · OWASP: Path Traversal · OWASP: File Path Traversal Prevention Cheat Sheet

[5] GGC-PROXYHDR-280 · UNTRUSTED_PROXY_HEADER — X-Forwarded-For

CONFIRMED CVSS 7,5 High · CWE-807 Reliance on Untrusted Inputs in a Security Decision (spoofable proxy header) · Priority P1 (Within 7 days)

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

A security decision (access/trust) is based on a client-controllable proxy header (X-Forwarded-For / X-Real-IP), which any attacker can spoof to bypass the check.

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: debug.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Evidence: @ offset 9 — via untrusted-proxy-header: Sending 'X-Forwarded-For: 127.0.0.1' — a client-controlled forwarding header — unlocked content…

Flow: debug.php trusts the X-Forwarded-For header to determine if the client is 'internal'. By spoofing this header to '127.0.0.1', any external attacker can access sensitive diagnostics including session data, cookies, all request headers, the database file path, and server configuration. This is accessible via GET with no authentication.

Evidence strength: Medium (75/100)

+50 Confirmed (strong signal)
+20 Direct detection (untrusted-proxy-header)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'debug.php'
```

Evidence (live response):

```
Sending 'X-Forwarded-For: 127.0.0.1' — a client-controlled forwarding header — unlocked content the SAME request without the header does not receive. The app trusts a spoofable header for an access/trust decision (CWE-807): any external attacker can set it to bypass the internal-…
```

Remediation

Never base an access/trust decision on a client-supplied proxy header (X-Forwarded-For, X-Real-IP, Client-IP) — any client can set it.

- For access decisions, use the real transport peer address (e.g. $_SERVER['REMOTE_ADDR']), not X-Forwarded-For / X-Real-IP / Client-IP.
- If you sit behind a reverse proxy, accept a forwarded client IP ONLY from a hardcoded allowlist of your proxy's own addresses, and read the LAST hop the proxy appended — never the whole client-supplied chain.
- Gate sensitive/diagnostic endpoints with real authentication + authorization, not an IP/localhost check.
- Strip inbound X-Forwarded-*/Forwarded headers at the edge so backends can't be reached with attacker-supplied values.

References: CWE-807 · CWE-346 · CWE-284 · OWASP: Authentication / Access Control Cheat Sheet

---

[6] GGC-CSRF-38A · CSRF — update_profile.php

PROBABLE CVSS 7,4 High (Base 8,1 High) · CWE-352 Cross-Site Request Forgery · Priority P1 (Within 7 days)

`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:U/C:H/I:H/A:N/E:P/RC:R`

A state-changing action lacks an anti-forgery token, so a malicious site can perform it as a logged-in victim.

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1539 Steal Web Session Cookie

Location: csrf-form: http://192.168.0.25/ggsec-cortex-debugger-lab/update_profile.php

Impact: PRIV_ESC (Direct)

Preconditions: Attacker privileges: None · Victim state: Authenticated · User interaction: Required (exercised at: Unauthenticated)

Flow: A state-changing POST form has no per-session/per-request anti-CSRF token, so a cross-site page can forge the request as the victim (CWE-352). This is a distinct root cause from any mass-assignment on the same endpoint.

Evidence strength: Low (43/100)

+18 Likely (weak signal only)
+20 Direct detection (csrf-missing-token)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'http://192.168.0.25/ggsec-cortex-debugger-lab/update_profile.php' --data 'display_name=Administrator&bio=Internal%20admin%20account&role=admin'
```

Evidence (live response):

```
[CSRF] The state-changing POST form at http://192.168.0.25/ggsec-cortex-debugger-lab/update_profile.php carries NO anti-CSRF token (no csrf/_token/nonce field) and accepts the request without one. Fields: display_name, bio, role. A malicious page can auto-submit this form on a lo…
```

Remediation

Require an anti-CSRF token on state-changing requests and use SameSite cookies.

- Add per-session/per-request CSRF tokens to all state-changing forms and verify them server-side.
- Set SameSite=Lax/Strict on session cookies; verify Origin/Referer for sensitive actions.
- Never perform state changes via GET.

References: CWE-352 · OWASP: CSRF Prevention Cheat Sheet

---

[7] GGC-BIZLOGIC-B64 · BUSINESS_LOGIC — discount

CONFIRMED CVSS 6,5 Medium · CWE-840 Business Logic Error (missing invariant validation — negative quantity / out-of-range discount → invalid total) · Priority P2 (Within 30 days)

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

The app enforces no business invariant on a client-supplied value (negative quantity, discount > 100%), so an attacker manipulates the price/total — financial fraud without any injection.

Compliance: A04:2021 – Insecure Design · PCI-DSS v4.0 Req 6.2.4 (business-logic) · ISO/IEC 27001:2022 A.8.28 / A.8.27 · GDPR Art. 25 + Art. 32 (data protection by design)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: business-logic: http://192.168.0.25/ggsec-cortex-debugger-lab/coupon.php

Impact: FINANCIAL_INTEGRITY (Direct)

Preconditions: Attacker privileges: User · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: A quantity/discount field reaches a price calculation with no server-side range check: a negative quantity or an over-100% discount yields an invalid (e.g. negative) total — business-logic / financial manipulation (CWE-840).

Evidence strength: Medium (75/100)

+50 Confirmed (strong signal)
+20 Direct detection (business-logic-invariant)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'http://192.168.0.25/ggsec-cortex-debugger-lab/coupon.php' --data
'qty=1&discount=-1'
```

Evidence (live response):

```
[BUSINESS LOGIC] Field 'discount' accepted the out-of-range value '-1' — accepted out-of-range
value '-1' into a price calculation with no validation error. A benign baseline request stays
valid; this one breaks a price/total invariant. <!doctype html><html><head><meta charset="u…
```

Remediation

Enforce business invariants on the SERVER for every value that affects price, quantity or totals — never trust the client's numbers.

- Validate ranges server-side before use: quantity >= 1, 0 <= discount <= 100, computed total >= 0 — reject (or clamp) anything outside and never proceed with an invalid value.
- Recompute prices/totals from server-side catalogue data; treat client-submitted price/total/discount as untrusted input, not as the source of truth.
- Use unsigned/whole-number types and explicit bounds for quantities; reject negatives and absurd magnitudes (integer-overflow / free-money bugs).
- Add regression tests asserting the invariants (negative qty, discount > 100, total < 0 must all be rejected).

References: CWE-840 · CWE-20 · OWASP: Business Logic Vulnerability / Input Validation Cheat Sheet

---

[8] GGC-BIZLOGIC-0A0 · BUSINESS_LOGIC — qty

CONFIRMED CVSS 6,5 Medium · CWE-840 Business Logic Error (missing invariant validation — negative quantity / out-of-range discount → invalid total) · Priority P2 (Within 30 days)

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

The app enforces no business invariant on a client-supplied value (negative quantity, discount > 100%), so an attacker manipulates the price/total — financial fraud without any injection.

Compliance: A04:2021 – Insecure Design · PCI-DSS v4.0 Req 6.2.4 (business-logic) · ISO/IEC 27001:2022 A.8.28 / A.8.27 · GDPR Art. 25 + Art. 32 (data protection by design)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: business-logic: http://192.168.0.25/ggsec-cortex-debugger-lab/coupon.php

Impact: FINANCIAL_INTEGRITY (Direct)

Preconditions: Attacker privileges: User · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: A quantity/discount field reaches a price calculation with no server-side range check: a negative quantity or an over-100% discount yields an invalid (e.g. negative) total — business-logic / financial manipulation (CWE-840).

Evidence strength: Medium (75/100)

+50 Confirmed (strong signal)
+20 Direct detection (business-logic-invariant)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'http://192.168.0.25/ggsec-cortex-debugger-lab/coupon.php' --data 'qty=-1&discount=10'
```

Evidence (live response):

```
[BUSINESS LOGIC] Field 'qty' accepted the out-of-range value '-1' — accepted out-of-range value '-1' into a price calculation with no validation error. A benign baseline request stays valid; this one breaks a price/total invariant. <!doctype html><html><head><meta charset="utf-8"…
```

Remediation

Enforce business invariants on the SERVER for every value that affects price, quantity or totals — never trust the client's numbers.

- Validate ranges server-side before use: quantity >= 1, 0 <= discount <= 100, computed total >= 0 — reject (or clamp) anything outside and never proceed with an invalid value.
- Recompute prices/totals from server-side catalogue data; treat client-submitted price/total/discount as untrusted input, not as the source of truth.
- Use unsigned/whole-number types and explicit bounds for quantities; reject negatives and absurd magnitudes (integer-overflow / free-money bugs).
- Add regression tests asserting the invariants (negative qty, discount > 100, total < 0 must all be rejected).

References: CWE-840 · CWE-20 · OWASP: Business Logic Vulnerability / Input Validation Cheat Sheet

---

[9] GGC-IDOR-7BD · IDOR — id

CONFIRMED CVSS 6,5 Medium · CWE-639 Authorization Bypass Through User-Controlled Key (IDOR) · Priority P2 (Within 30 days)

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

Object references aren't authorization-checked, so changing an id exposes or modifies other users' data (IDOR).

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1078 Valid Accounts · T1548 Abuse Elevation Control Mechanism

Location: api.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: User · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Evidence: @ offset 315 — via idor-enumeration: …y_name": "Administrator", "role": "admin", "api_key": "GGSE***REDACTED(20)***" } }

Flow: api.php?action=user&id=X returns user data including role and api_key fields for any user ID without any authentication. This is unauthenticated user enumeration with sensitive field exposure.

Evidence strength: High (99/100)

+50 Confirmed (strong signal)
+20 Direct detection (idor-enumeration)
+6 Expected indicator reflected
+5 Stable 2xx response
+18 Baseline-diff (matched control rejected)

Proof of Concept

Reproduce:

```
curl -i 'api.php?action=user&id=2'
```

Evidence (live response):

```
[IDOR param=id] id=1:200(obj); id=2:200(obj); id=3:200(obj); control=9999:200(rej) distinct records returned to one session: 3 sample: { "ok": true, "request_id_raw": "1", "user": { "id": 1, "username": "admin", "display_name": "Administrator",…
```

Remediation

Enforce per-object authorization on every request — verify the current user owns (or may access) the referenced object.

- On every object access, check ownership/permission server-side: the session identity must be authorized for the requested id, never the id alone.
- Scope queries to the authenticated user (WHERE owner_id = :sessionUser) instead of trusting a client-supplied id.
- Prefer unpredictable, non-sequential identifiers (UUIDs) so ids cannot be trivially enumerated — defense in depth, not a substitute for authz.
- Add access-control tests and centralize authorization in middleware/policies rather than ad-hoc per-endpoint checks.

References: CWE-639 · CWE-284 · OWASP: Insecure Direct Object Reference Prevention Cheat Sheet · OWASP API1:2023 Broken Object Level Authorization

---

[10] GGC-IDOR-D48 · IDOR — id

CONFIRMED CVSS 6,5 Medium · CWE-639 Authorization Bypass Through User-Controlled Key (IDOR) · Priority P2 (Within 30 days)

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

Object references aren't authorization-checked, so changing an id exposes or modifies other users' data (IDOR).

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1078 Valid Accounts · T1548 Abuse Elevation Control Mechanism

Location: profile.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: User · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: profile.php fetches any user profile by id without authorization check beyond requiring login. Any authenticated user can view any other user's profile including sensitive fields (api_key is in an HTML comment). Seeded user IDs: 1=admin, 2=alice, 3=bob.

Evidence strength: High (93/100)

+50 Confirmed (strong signal)
+20 Direct detection (idor-enumeration)
+5 Stable 2xx response
+18 Baseline-diff (matched control rejected)

Proof of Concept

Reproduce:

```
curl -i 'profile.php?id=2'
```

Evidence (live response):

```
[IDOR param=id] id=1:200(obj); id=2:200(obj); id=3:200(obj); control=9999:200(rej) distinct records returned to one session: 3 sample: <!doctype html><html><head><meta charset="utf-8"><meta name="viewport" content="width=device-width,initial-scale=1"><title>Profile</title><style>…
```

Remediation

Enforce per-object authorization on every request — verify the current user owns (or may access) the referenced object.

- On every object access, check ownership/permission server-side: the session identity must be authorized for the requested id, never the id alone.
- Scope queries to the authenticated user (WHERE owner_id = :sessionUser) instead of trusting a client-supplied id.

- Prefer unpredictable, non-sequential identifiers (UUIDs) so ids cannot be trivially enumerated — defense in depth, not a substitute for authz.
- Add access-control tests and centralize authorization in middleware/policies rather than ad-hoc per-endpoint checks.

References: CWE-639 · CWE-284 · OWASP: Insecure Direct Object Reference Prevention Cheat Sheet · OWASP API1:2023 Broken Object Level Authorization

---

[11] GGC-IDOR-AC7 · IDOR — id

CONFIRMED CVSS 6,5 Medium · CWE-639 Authorization Bypass Through User-Controlled Key (IDOR) · Priority P2 (Within 30 days)

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

Object references aren't authorization-checked, so changing an id exposes or modifies other users' data (IDOR).

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1078 Valid Accounts · T1548 Abuse Elevation Control Mechanism

Location: notes.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: User · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: notes.php fetches any note by id (via parameterized query) without checking that the note belongs to the logged-in user. A logged-in user can read private notes of other users by iterating note IDs. The seeded data includes private notes with tokens (e.g. note id=3 belongs to alice, note id=4 belongs to bob).

Evidence strength: High (93/100)

+50 Confirmed (strong signal)
+20 Direct detection (idor-enumeration)
+5 Stable 2xx response
+18 Baseline-diff (matched control rejected)

Proof of Concept

Reproduce:

```
curl -i 'notes.php?id=3'
```

Evidence (live response):

```
[IDOR param=id] id=1:200(rej); id=3:200(obj); id=4:200(obj); control=9999:200(rej) distinct records returned to one session: 2 sample: <!doctype html><html><head><meta charset="utf-8"><meta name="viewport" content="width=device-width,initial-scale=1"><title>Notes</title><style>…
```

Remediation

Enforce per-object authorization on every request — verify the current user owns (or may access) the referenced object.

- On every object access, check ownership/permission server-side: the session identity must be authorized for the requested id, never the id alone.
- Scope queries to the authenticated user (WHERE owner_id = :sessionUser) instead of trusting a client-supplied id.
- Prefer unpredictable, non-sequential identifiers (UUIDs) so ids cannot be trivially enumerated — defense in depth, not a substitute for authz.
- Add access-control tests and centralize authorization in middleware/policies rather than ad-hoc per-endpoint checks.

References: CWE-639 · CWE-284 · OWASP: Insecure Direct Object Reference Prevention Cheat Sheet · OWASP API1:2023 Broken Object Level Authorization

---

[12] GGC-SESSFIX-85F · SESSION_FIXATION — PHPSESSID

CONFIRMED CVSS 6,4 Medium (Base 6,8 Medium) · CWE-384 Session Fixation · Priority P2 (Within 30 days)

`CVSS:3.1/AV:N/AC:H/PR:L/UI:R/S:U/C:H/I:H/A:N/E:H/RC:C`

The session id is not regenerated on login, so an attacker who fixes a known id can hijack the authenticated session.

Compliance: A07:2021 – Identification and Authentication Failures · PCI-DSS v4.0 Req 8.3.1 / 8.6 / 3.6 (key mgmt) · ISO/IEC 27001:2022 A.8.5 / A.5.17 / A.8.24 · GDPR Art. 32 (security of processing)

MITRE ATT&CK: T1539 Steal Web Session Cookie

Location: login.php

Impact: SESSION_THEFT (Conditional)

Preconditions: Attacker privileges: None · Victim state: Authenticated · User interaction: Required (exercised at: Customer)

Flow: login.php sets $_SESSION['uid'] on successful authentication but never calls session_regenerate_id(). An attacker who pre-sets the session ID (e.g. via a link with ?PHPSESSID= or a Set-Cookie) can hijack the session after the victim authenticates.

Evidence strength: Medium (70/100)

+50 Confirmed (strong signal)
+20 Direct detection (session-fixation)

Proof of Concept

Reproduce:

```
curl -i -X POST 'login.php' --data 'SESSIONFIX:PHPSESSID=ggsecfix8d409650e357'
```

Evidence (live response):

```
[SESSION FIXATION] planted PHPSESSID=ggse***REDACTED(21)*** authenticated as 'alice', and the SAME id is still an authenticated session (server did not regenerate it; Set-Cookie: none). An attacker who fixes a victim's session id keeps access after the victim logs in.
```

Remediation

Regenerate the session identifier on every privilege change (login), and harden the session cookie.

- Call session_regenerate_id(true) / equivalent immediately after successful authentication.
- Reject session IDs the server didn't issue; never accept a session id from the URL.
- Set HttpOnly, Secure and SameSite on the session cookie; expire idle sessions.

References: CWE-384 · OWASP: Session Management Cheat Sheet

---

[13] GGC-REDIR-FD7 · OPEN_REDIRECT — next

CONFIRMED CVSS 6,1 Medium · CWE-601 Open Redirect · Priority P2 (Within 30 days)

`CVSS:3.1/AV:N/AC:L/PR:N/UI:R/S:C/C:L/I:L/A:N/E:H/RC:C`

An unvalidated redirect target lets an attacker send victims to a malicious site under the app's trust.

Compliance: A01:2021 – Broken Access Control · PCI-DSS v4.0 Req 6.2.4 / 7.2.1 · ISO/IEC 27001:2022 A.8.3 / A.5.15 · GDPR Art. 5(1)(f) + Art. 32 (confidentiality)

MITRE ATT&CK: T1204 User Execution

Location: redirect.php

Impact: SESSION_THEFT (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: Required (exercised at: Unauthenticated)

Flow: $_GET['next'] is passed directly to header('Location: ' . $next) without any validation or whitelist check. An attacker can redirect users to arbitrary external URLs.

Evidence strength: Medium (74/100)

+50 Confirmed (strong signal)
+20 Direct detection (redirect-header)
+4 Reproduced by 4 payload(s)

Proof of Concept

Reproduce:

```
curl -i 'redirect.php?next=https%3A%2F%2Fevil.com'
```

Evidence (live response):

```
(no response body captured)
```

Remediation

Validate redirect targets against an allowlist; never redirect to a raw user-supplied URL.

- Allowlist permitted destinations or use server-side keys mapped to URLs.
- Accept only relative paths; reject absolute URLs and protocol-relative (//) values.

References: CWE-601 · OWASP: Unvalidated Redirects and Forwards Cheat Sheet

---

[14] GGC-XSS-1B2 · XSS — bio

CONFIRMED CVSS 5,4 Medium (Base 6,1 Medium) · CWE-79 Cross-site Scripting · Priority P2 (Within 30 days)

CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N/E:H/RC:C

Untrusted input is reflected into a page without encoding, so attacker script runs in the victim's browser (session theft).

Compliance: A03:2021 – Injection · PCI-DSS v4.0 Req 6.2.4 (injection/XSS) · ISO/IEC 27001:2022 A.8.28 · GDPR Art. 32 (security of processing)

MITRE ATT&CK: T1059.007 Command and Scripting Interpreter: JavaScript · T1185 Browser Session Hijacking

Location: profile.php

Impact: SESSION_THEFT (Chained)

Preconditions: Attacker privileges: User · Victim state: Authenticated · User interaction: Required (exercised at: Customer)

Evidence: @ offset 954 — reflected UNENCODED (executes): … <!-- INTENTIONALLY VULNERABLE: stored XSS --> Bio: ><script>alert(1)</script> Edit my profile</p…

Flow: In profile.php, the 'bio' field from the database is output without escaping: ' Bio: <?= $user["bio"] ?> '. An attacker stores malicious HTML/JS via update_profile.php (which uses a parameterized INSERT so the store is safe), and it renders unescaped on any profile view. This is stored XSS.

Evidence strength: Medium (83/100)

+50 Confirmed (strong signal)
+20 Direct detection (stored-xss-post)
+6 Expected indicator reflected
+2 Reproduced by 2 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
[store] POST update_profile.php --data
'display_name=Alice&bio=><script>alert(1)</script>&role=user' → [render] GET profile.php
```

Evidence (live response):

```
…Display name: Alice</p> <!-- INTENTIONALLY VULNERABLE: stored XSS --> <p>Bio:
><script>alert(1)</script></p> <p><a href="update_profile.php">Edit my profile</a></p> <!--
INTENTIONALLY VULNERABLE: sensitive field exposed in HTML comment --> <!--
api_key=GGSE***REDACTED(20)*** -…
```

Remediation

Apply context-aware output encoding on every place user data reaches HTML/JS, and add a CSP.

- Encode on output by context: HTML body (htmlspecialchars/escapeHTML), attributes, JS, URL.
- Prefer auto-escaping template engines; avoid raw echo/print of tainted data.
- Deploy a strict Content-Security-Policy and set HttpOnly on session cookies.

References: CWE-79 · OWASP: XSS Prevention Cheat Sheet

---

[15] GGC-XSS-CE8 · XSS — q

CONFIRMED CVSS 5,4 Medium (Base 6,1 Medium) · CWE-79 Cross-site Scripting · Priority P2 (Within 30 days)

CVSS:3.1/AV:N/AC:L/PR:L/UI:R/S:C/C:L/I:L/A:N/E:H/RC:C

Untrusted input is reflected into a page without encoding, so attacker script runs in the victim's browser (session theft).

Compliance: A03:2021 – Injection · PCI-DSS v4.0 Req 6.2.4 (injection/XSS) · ISO/IEC 27001:2022 A.8.28 · GDPR Art. 32 (security of processing)

MITRE ATT&CK: T1059.007 Command and Scripting Interpreter: JavaScript · T1185 Browser Session Hijacking

Location: search.php

Impact: SESSION_THEFT (Direct)

Preconditions: Attacker privileges: User · Victim state: Authenticated · User interaction: Required (exercised at: Customer)

Evidence: @ offset 1203 — reflected UNENCODED (executes): …LNERABLE: reflected XSS in raw mode --> Results for: <script>alert('XSS')</script> SQL error: SQLSTATE[…

Flow: When mode=raw, $_GET['q'] is echoed directly without htmlspecialchars: '<?= $q ?>'. This is a reflected XSS vulnerability.

Evidence strength: High (85/100)

+50 Confirmed (strong signal)
+20 Direct detection (indicator)
+6 Expected indicator reflected
+4 Reproduced by 4 payload(s)

+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'search.php?mode=raw&q=%3Cscript%3Ealert%28%27XSS%27%29%3C%2Fscript%3E'
```

Evidence (live response):

```
…<p>Results for: <script>alert('XSS')</script></p> <div class="box"><strong>SQL error:</strong>
SQLSTATE[HY000]: General error: 1 near &quot;XSS&quot;: syntax error</div> <hr><small>GGSEC Cortex
Debugger Lab — localhost only</small></body></html>
```

Remediation

Apply context-aware output encoding on every place user data reaches HTML/JS, and add a CSP.

- Encode on output by context: HTML body (htmlspecialchars/escapeHTML), attributes, JS, URL.
- Prefer auto-escaping template engines; avoid raw echo/print of tainted data.
- Deploy a strict Content-Security-Policy and set HttpOnly on session cookies.

References: CWE-79 · OWASP: XSS Prevention Cheat Sheet

---

[16] GGC-HPP-C34 · HTTP_PARAMETER_POLLUTION — id

PROBABLE CVSS 3,4 Low (Base 3,7 Low) · CWE-235 Improper Handling of Extra / Duplicate Parameters (HPP / parameter type confusion — id[]=… or id=1&id=2; parser behaviour, no proven control bypass) · Priority P3 (Next release)

`CVSS:3.1/AV:N/AC:H/PR:N/UI:N/S:U/C:L/I:N/A:N/E:P/RC:R`

The app mishandles duplicate or array-typed parameters (id=1&id=2, id[]=1): PHP coerces or last-wins the value, so an attacker confuses which value/type is used to bypass filters or reach unintended records.

Compliance: A04:2021 – Insecure Design · PCI-DSS v4.0 Req 6.2.4 (business-logic) · ISO/IEC 27001:2022 A.8.28 / A.8.27 · GDPR Art. 25 + Art. 32 (data protection by design)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: hpp: api.php?action=user&id=PAYLOAD

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: The endpoint mishandles duplicate or array-typed parameters (id=1&id=2 last-wins, or id[]=1 coerced by PHP) — an attacker can confuse which value/type is used to bypass filters or reach unintended records (CWE-235).

Evidence strength: Low (43/100)

+18 Likely (weak signal only)
+20 Direct detection (hpp-typeconfusion)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'api.php?action=user&id=1&id=2'
```

Evidence (live response):

```
[HPP] Parameter 'id' @ api.php: array/type confusion (id[]=1 coerced by PHP) AND duplicate-
parameter last-wins (id=1&id=2 → second record). Clean id=1 and id=2 return different records; the
malformed request changed which value/type the app used. { "ok": true, "request_id…
```

Remediation

Accept exactly one value of the expected TYPE for each parameter — reject duplicates and array/object forms.

- Read parameters through a strict accessor that returns a single scalar and rejects (or 400s) duplicate keys and array syntax (id[]=…, id[x]=…).
- Validate the TYPE before use: cast/parse to the expected type and reject anything that isn't a plain integer/string (never let PHP coerce an array to a scalar).
- Normalise at the edge (web server / framework middleware) so the backend sees one canonical value per parameter.
- Add tests for id=1&id=2 and id[]=1 asserting a consistent, safe result (single value, or explicit rejection).

References: CWE-235 · CWE-843 · OWASP: Testing for HTTP Parameter Pollution

[17] GGC-SECRET-DE9 · HARDCODED_SECRET — db_seed_password

POTENTIAL CVSS 4,5 Medium (Base 5,3 Medium) · CWE-798 Use of Hard-coded Credentials · Priority P2 (Within 30 days)

`CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N/E:U/RC:U`

A credential (API key, password, token, private key) is hard-coded in source — anyone with repo access gets it; it must be moved to a secret store and rotated.

Compliance: A07:2021 – Identification and Authentication Failures · PCI-DSS v4.0 Req 8.3.1 / 8.6 / 3.6 (key mgmt) · ISO/IEC 27001:2022 A.8.5 / A.5.17 / A.8.24 · GDPR Art. 32 (security of processing)

MITRE ATT&CK: T1552.001 Unsecured Credentials: Credentials In Files

Location: bootstrap.php

Impact: INFORMATION_DISCLOSURE (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: Hard-coded credentials in source: bootstrap.php seeds plaintext user passwords (admin/Admin123!, alice/alice123) directly in code. Anyone with read access to the repository obtains the credentials. Move secrets to environment variables or a secret store and rotate.

Evidence strength: Low (38/100)

+18 Likely (weak signal only)
+20 Direct detection (sast-static)

Proof of Concept

Reproduce:

```
curl -i -X POST 'bootstrap.php' --data '(static analysis — no runtime probe)'
```

Evidence (live response):

```
Hard-coded credentials in source: bootstrap.php seeds plaintext user passwords (admin/Admin123!, alice/alice123) directly in code. Anyone with read access to the repository obtains the credentials. Move secrets to environment variables or a secret store and rotate.
```

Remediation

Remove the secret from source, load it from an environment variable / secret manager, and ROTATE it — it must be treated as compromised.

- Rotate/revoke the exposed credential NOW — it is in version control history and must be assumed leaked.
- Read it at runtime from an env var or a secret manager (Vault/AWS Secrets Manager/Azure Key Vault); never commit it.
- Purge it from git history (git filter-repo / BFG) and add a pre-commit secret scanner (e.g. this scan) to CI.

References: CWE-798 · OWASP: Secrets Management Cheat Sheet

---

[18] GGC-PLAINPW-B67 · PLAINTEXT_PASSWORD_STORAGE — password

POTENTIAL CVSS 3,7 Low (Base 4,4 Medium) · CWE-256 Plaintext Storage of a Password (credentials stored/compared unhashed) · Priority P3 (Next release)

`CVSS:3.1/AV:N/AC:H/PR:H/UI:N/S:U/C:H/I:N/A:N/E:U/RC:U`

User passwords are stored (or compared) in plaintext rather than a salted one-way hash, so anyone who reads the DB (e.g. via SQLi) gets every usable credential.

Compliance: A02:2021 – Cryptographic Failures · PCI-DSS v4.0 Req 4.2.1 / 6.2.4 / 8.3.1 · ISO/IEC 27001:2022 A.8.24 · GDPR Art. 32(1)(a) (encryption)

MITRE ATT&CK: T1190 Exploit Public-Facing Application

Location: login.php:12

Impact: CREDENTIAL_COMPROMISE (Direct)

Preconditions: Attacker privileges: None · Victim state: None · User interaction: None (exercised at: Unauthenticated)

Flow: The application stores or compares user passwords in PLAINTEXT at login.php:12: the password value is persisted or matched directly (e.g. WHERE password = '$password') without a one-way password hash such as password_hash()/password_verify(). Anyone who reads the users table obtains every usable credential. Static/source finding (no safe runtime proof) — verify.

Evidence strength: Low (38/100)

+18 Likely (weak signal only)
+20 Direct detection (sast-static)

Proof of Concept

Reproduce:

```
curl -i -X POST 'login.php:12' --data '(static analysis — no runtime probe)'
```

Evidence (live response):

```
The application stores or compares user passwords in PLAINTEXT at login.php:12: the password value is persisted or matched directly (e.g. WHERE password = '$password') without a one-way password hash such as password_hash()/password_verify(). Anyone who reads the users table obta…
```

Remediation

Never store or compare passwords in plaintext — persist only a salted one-way hash and verify against it.

- At registration/change, store password_hash($pw, PASSWORD_DEFAULT) (bcrypt/argon2id) — never the raw password.
- At login, fetch the user row by username only, then verify with password_verify($pw, $row['password_hash']); never put the password in the SQL WHERE clause.
- Migrate existing plaintext rows: re-hash on next successful login and drop the plaintext column.
- Add a lint/CI rule forbidding `password = '$…'` in SQL and any comparison of a raw password to a stored value.

References: CWE-256 · CWE-257 · CWE-312 · OWASP: Password Storage Cheat Sheet