VULNERABILITY ASSESSMENT REPORT

testphp.vulnweb.com

Author: Weddy Gacheri

Date: 30/01/2026

Assignment: Future Interns Task 1

Executive Summary

Overview

This report details the findings of a vulnerability assessment conducted on testphp.vulnweb.com. The assessment aimed to identify potential security weaknesses that could be exploited by attackers.This report outlines the identified vulnerabilities, their potential impact, and recommended remediation steps.

Summary of Key Findings

Risk Level	Number of Findings	Examples
High	2	SQL Injection, Default Admin Credentials
Medium	2	Missing CSRF Tokens, Exposed CVS Directory

Critical Risks Identified

1. **SQL Injection (High Risk)**
   An attacker can manipulate database queries through the cat parameter in listproducts.php, potentially leading to data theft, authentication bypass, or complete database compromise.
2. **Default Admin Credentials (High Risk)**
   The admin panel is accessible using default credentials (test:test), allowing unauthorized access to sensitive user data including names, email addresses, and credit card information.
3. **Missing Anti-CSRF Tokens (Medium Risk)**
   Forms lack CSRF protection, making users vulnerable to having unintended actions performed

on their behalf.

4. **Exposed CVS Directory (Medium Risk)**
   Version control metadata is publicly accessible, revealing internal file structure and potentially sensitive information about the application's development history.

Business Impact

If exploited, these vulnerabilities could lead to:

- **Data Breach:** Exposure of customer personal and financial information
- **Account Takeover:** Attackers gaining administrative access
- **Reputation Damage:** Loss of customer trust and potential regulatory penalties
- **Financial Loss:** Fraudulent transactions or remediation costs

Recommendations

Immediate actions should include:

- Fixing the SQL injection vulnerability by implementing parameterized queries
- Changing default admin credentials and restricting access to the admin panel
- Implementing anti-CSRF tokens across all forms
- Restricting public access to version control directories

Conclusion

While this application is intentionally vulnerable for testing purposes, the findings demonstrate common real-world security gaps that must be addressed in production environments. Addressing these issues will significantly improve the application's security posture.

Scope & Methodology

- **Target URL:** testphp.vulnweb.com
- Assessment type: External Vulnerability Assessment
- **Date of Assessment:** 30/01/2026
- **Tools Used:** OWASP ZAP 2.14.0, Brave Browser, Manual Testing Techniques
- **Testing Approach:** The assessment involved a combination of automated scanning and manual verification

Methodology Overview

The assessment followed a structured approach to identify security vulnerabilities in the target application:

Phase 1: Reconnaissance

- Initial exploration of the application to understand its structure and functionality
- Manual browsing of all accessible pages including the homepage, product listings, guestbook, and admin areas
- Identification of input points (forms, URL parameters) that could be tested for vulnerabilities

Phase 2: Automated Scanning

- Configuration of OWASP ZAP as a local proxy to intercept and analyze traffic
- Running automated spidering to discover all accessible pages and endpoints
- Performing active scanning to test for common web vulnerabilities including:
  - SQL Injection
  - Cross-Site Scripting (XSS)
  - Cross-Site Request Forgery (CSRF)
  - Security misconfigurations
  - Information disclosure issues

Phase 3: Manual Verification

- Confirming automated findings through manual testing
- Attempting to exploit identified vulnerabilities to assess real-world impact
- Testing for logic flaws and business logic vulnerabilities
- Verifying default credentials and access controls

Risk Classification

Risk Rating	Description	Impact	Likelihood
High	Critical vulnerability that can be easily exploited.	Severe data breach, system compromise.	Highly likely to be exploited.
Medium	Vulnerability that can be exploited under certain conditions.	Moderate data breach, partial system compromise.	Likely to be exploited.
Low	Minor vulnerability with limited impact.	Minimal data breach, no system compromise.	Unlikely to be exploited.

Limitations

- This assessment was conducted against a deliberately vulnerable test application

- No authentication bypass or privilege escalation testing was performed beyond default credentials
- Testing was limited to vulnerabilities detectable through external scanning and manual testing
- Denial of Service (DoS) attacks were not attempted

Testing Standards Referenced

- OWASP Top 10 (2021) - Web Application Security Risks
- OWASP Testing Guide v4.0
- CWE (Common Weakness Enumeration) classifications

Finding 1: SQL Injection

Risk Level	HIGH
Location	http://testphp.vulnweb.com/listproducts.php?cat=1
Parameter	cat
CWE	CWE-89: Improper Neutralization of Special Elements used in an SQL Command

Description

The application fails to properly validate user input passed through the cat parameter in the listproducts.php page. This allows an attacker to inject arbitrary SQL commands into the database query.

When a normal request is made:

http://testphp.vulnweb.com/listproducts.php?cat=1

The application displays products from category 1.

However, when a single quote is injected:

http://testphp.vulnweb.com/listproducts.php?cat=1'

The application returns a detailed SQL error message, confirming that user input is being directly inserted into SQL queries without proper sanitization or parameterization.

This vulnerability can be further exploited using UNION-based attacks to extract data from other database tables.

Business Impact

A successful SQL injection attack can have severe consequences:

- **Data Breach:** Attackers can extract sensitive information from the database, including user credentials, personal information, and financial data
- **Authentication Bypass:** Attackers may be able to log in as any user without knowing their password
- **Data Manipulation:** Attackers can modify or delete database content, corrupting the application
- **Privilege Escalation:** In some configurations, attackers may gain operating system access to the underlying server
- **Regulatory Violations:** Data breaches can lead to violations of GDPR, CCPA, or other data protection regulations, resulting in significant fines

For an e-commerce application like this one, customer data including names, addresses, and payment information could be at risk.

Remediation

To fix SQL injection vulnerabilities:

1. **Use Parameterized Queries (Prepared Statements):**
   Replace dynamic query construction with parameterized queries. This ensures user input is treated as data, not executable code.
2. **Input Validation:**
   Validate that the cat parameter contains only expected characters (e.g., only digits). Reject any input that doesn't match.
3. **Least Privilege:**
   Ensure the database account used by the application has minimal necessary privileges. It should not have permission to modify database structure or access system tables.
4. **Error Handling:**
   Disable detailed error messages in production. Display a generic error page while logging details securely for developers.
5. **Web Application Firewall (WAF):**
   Deploy a WAF as an additional layer of defense, though this should not replace secure coding practices.

Evidence:

Figure 1: SQL error message displayed after injecting a single quote

Finding 2: Default Admin Credentials

Risk Level	HIGH
Location	http://testphp.vulnweb.com/admin/
Credentials	Username: test / Password: test
CWE	CWE-798: Use of Hard-coded Credentials

Description

The administrative panel of the application is protected only by a login form that accepts **default, well-known credentials**. Using the username test and password test provides immediate access to the admin area.

Once logged in, the administrator dashboard reveals sensitive information including:

- User details (names, email addresses)
- Credit card information
- Order history
- Database contents

This is not a "hidden" or "secret" admin panel it's publicly accessible at /admin/, and the credentials are trivial to guess or widely known from application documentation.

Business Impact

This vulnerability represents a **critical failure in access control** with severe consequences:

- **Complete System Compromise:** An attacker gains full administrative control over the application
- **Mass Data Theft:** All customer data, including financial information, can be exported
- **Identity Fraud:** Stolen personal information can be used for identity theft and fraud
- **Regulatory Fines:** Exposure of credit card data violates PCI DSS requirements, potentially resulting in fines of $5,000 to $100,000 per month
- **Reputation Damage:** Customers lose trust when their data is compromised
- **Legal Liability:** Affected customers may pursue legal action

In a real e-commerce application, this would be a **catastrophic security failure** requiring immediate incident response and customer notification.

Remediation

Immediate actions required:

1. **Change Default Credentials IMMEDIATELY:**
   Replace the default test/test credentials with strong, unique credentials:
   - Minimum 12 characters
   - Mix of uppercase, lowercase, numbers, and special characters
   - Not based on dictionary words or common patterns
2. **Implement Proper Access Controls:**
   - Restrict admin panel access by IP address where possible
   - Implement multi-factor authentication (MFA) for all admin accounts
   - Create individual accounts for each administrator (no shared accounts)
3. **Secure the Admin Panel:**
   - Move the admin panel to a non-standard location (e.g., /secure-admin-xyz/ instead of /admin/)
   - Implement rate limiting to prevent brute-force attacks
   - Log all admin access attempts and actions for audit purposes
4. **Development Process Improvements:**
   - Remove all default credentials from production code
   - Implement automated security testing to detect hard-coded credentials
   - Conduct security training for developers on secure authentication practices

Evidence:

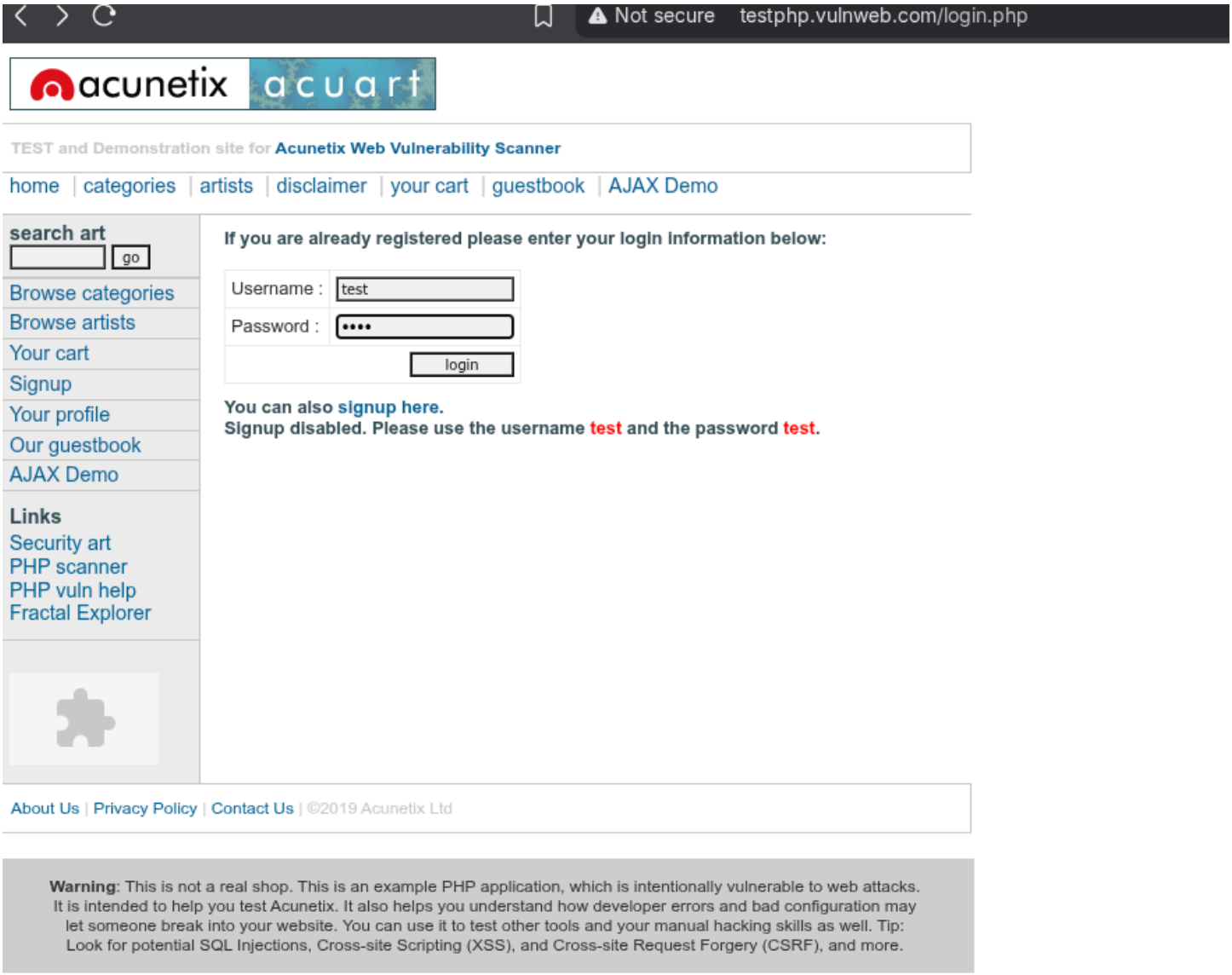

Figure 2: Admin login page with default credentials

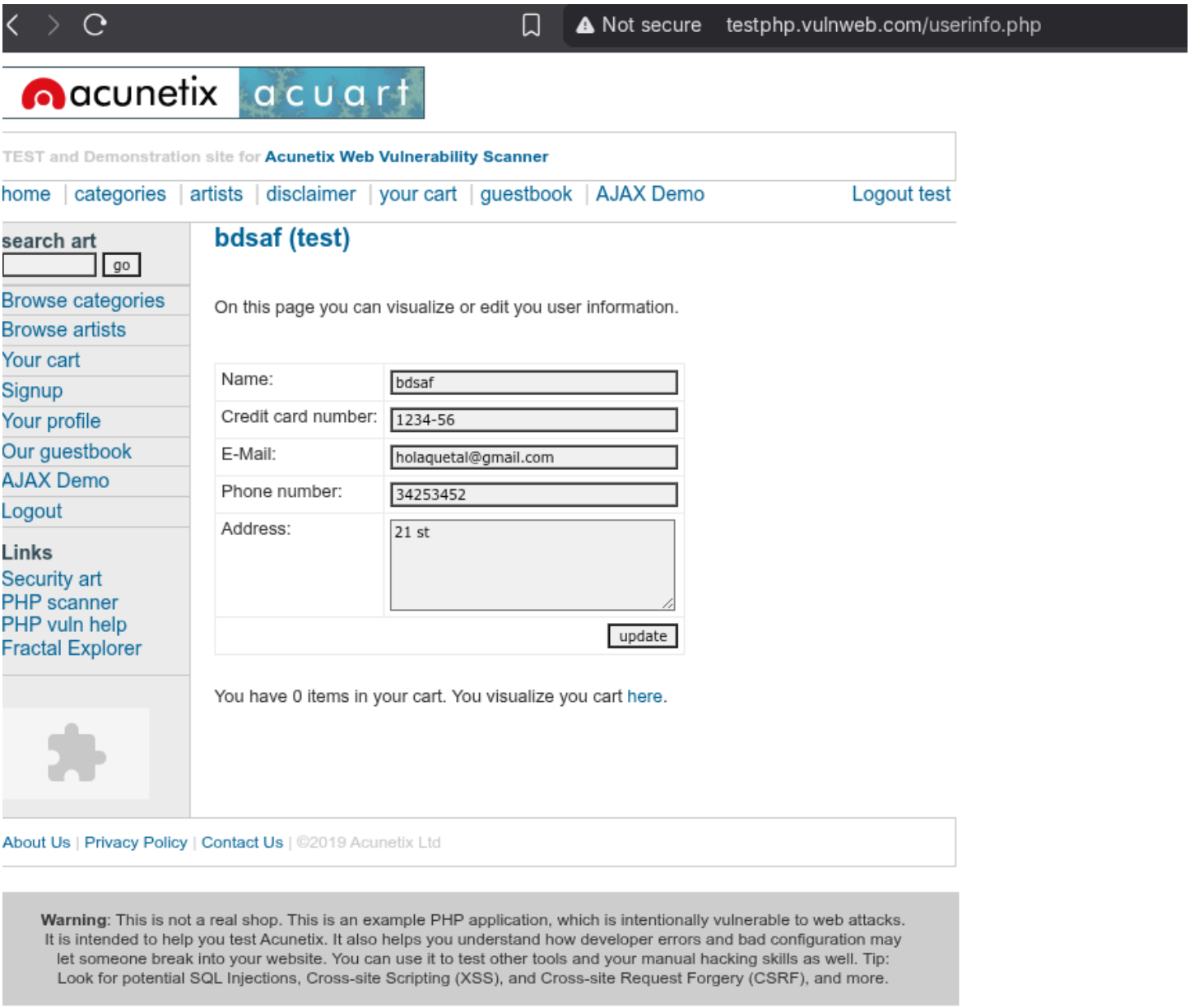

Figure 3: Admin dashboard displaying sensitive customer information after login

Finding 3: Exposed CVS Directory

Risk Level	MEDIUM
Location	http://testphp.vulnweb.com/CVS/
Exposed Files	Entries, Entries.Log, Repository, Root
CWE	CWE-548: Information Exposure Through Directory Listing

Description

The application exposes its **CVS (Concurrent Versions System) version control directory** to the public internet. Directory listing is enabled, allowing anyone to browse the contents of /CVS/ and access version control metadata files.

The exposed files include:

- **Entries** - Lists all files under version control in this directory
- **Repository** - Contains the path to the CVS repository on the server
- **Root** - Specifies the location of the CVS root directory
- **Entries.Log** - Tracks changes to the Entries file

Version control directories like CVS and .svn should never be deployed to production servers. They contain internal project metadata that helps attackers understand the application's structure and development history.

Business Impact

While this vulnerability doesn't directly allow system compromise, it provides valuable intelligence to attackers:

- **Attack Surface Mapping:** Attackers can discover file names and directory structures they wouldn't otherwise know about
- **Version Information:** Old file versions may contain vulnerabilities that were fixed in later versions
- **Social Engineering:** Knowledge of internal file paths and developer comments can be used in targeted phishing attacks
- **Source Code Exposure:** In some configurations, actual source code files may be exposed through version control metadata
- **Credential Leakage:** Developers sometimes accidentally commit credentials or API keys to version control

This information makes other attacks more effective. An attacker who knows your file structure can craft more precise SQL injection or path traversal attacks.

Remediation

1. **Immediate Action - Block Access:**
   Configure the web server to deny access to the /CVS/ directory:
2. **Remove Version Control Directories from Production:**
   Never deploy version control directories to production servers. Use build scripts that copy only necessary files:
3. **Disable Directory Listings:**
   Ensure directory listings are disabled globally on the web server:
4. **Security in Development Process:**
   - Add version control directories to .gitignore (if using Git)
   - Implement CI/CD pipelines that clean deployment artifacts
   - Conduct regular security scans to detect exposed metadata

Evidence:

Figure 4: Exposed CVS directory with version control metadata files

Finding 4:Missing Anti-CSRF Tokens

Risk Level	MEDIUM
Location	Throughout the application (multiple forms)
CWE	CWE-352: Cross-Site Request Forgery (CSRF)

Description

The application processes form submissions without implementing **anti-CSRF tokens**. CSRF tokens are unique, secret values embedded in forms that verify the submission came from the legitimate application interface, not from an external malicious site.

When a user submits a form, the server checks that the submitted token matches the one it generated. Without this check, there is no way to distinguish between:

- A legitimate form submission from the actual website
- A forged request triggered by an attacker from another site

Forms throughout the application—including login forms, data entry forms, and administrative functions lack this protection mechanism.

Business Impact

CSRF vulnerabilities allow attackers to **perform actions on behalf of authenticated users without their knowledge or consent**:

- **Unauthorized Actions:** An attacker could trick an administrator into creating new admin accounts, changing settings, or deleting content
- **State-Changing Operations:** Password changes, email updates, fund transfers, or purchases could be initiated without user awareness
- **Chained Attacks:** CSRF is often combined with other vulnerabilities (like XSS) to create more powerful exploits
- **Widespread Impact:** A single malicious webpage could affect multiple authenticated users who visit it

Real-world scenario: An attacker creates a harmless-looking webpage. When an authenticated administrator visits it, hidden code submits a form to create a new admin account. The server processes this as if the administrator intentionally created it. The attacker now has persistent admin access.

While CSRF typically doesn't result in data theft directly, it enables attackers to **manipulate application state and data** in ways that can lead to privilege escalation, data corruption, or further compromise.

Remediation

1. **Implement Anti-CSRF Tokens:**
   Add unique, unpredictable tokens to all forms that process state-changing requests:
   - Generate a cryptographically random token per session or per form
   - Include the token as a hidden field in the form
   - Validate the token on the server side for every submission
   - Reject requests with missing, invalid, or expired tokens
2. **Use SameSite Cookie Attribute:**
   Set cookies with SameSite=Strict or SameSite=Lax to prevent browsers from sending them with cross-site requests
3. **Framework Built-in Protection:**
   Modern frameworks (Laravel, Django, Spring, ASP.NET Core) have built-in CSRF protection. Enable and use these features instead of building custom solutions.
4. **Additional Defenses:**
   - Require re-authentication for sensitive actions (password changes, money transfers)
   - Implement CAPTCHA for critical functions
   - Use custom request headers for AJAX requests (since cross-origin JavaScript cannot set custom headers)
5. **Verify Origin Headers:**
   Check the Origin and Referer headers to ensure requests originate from your own domain. Note that this is a defense-in-depth measure, not a replacement for tokens.

Evidence:

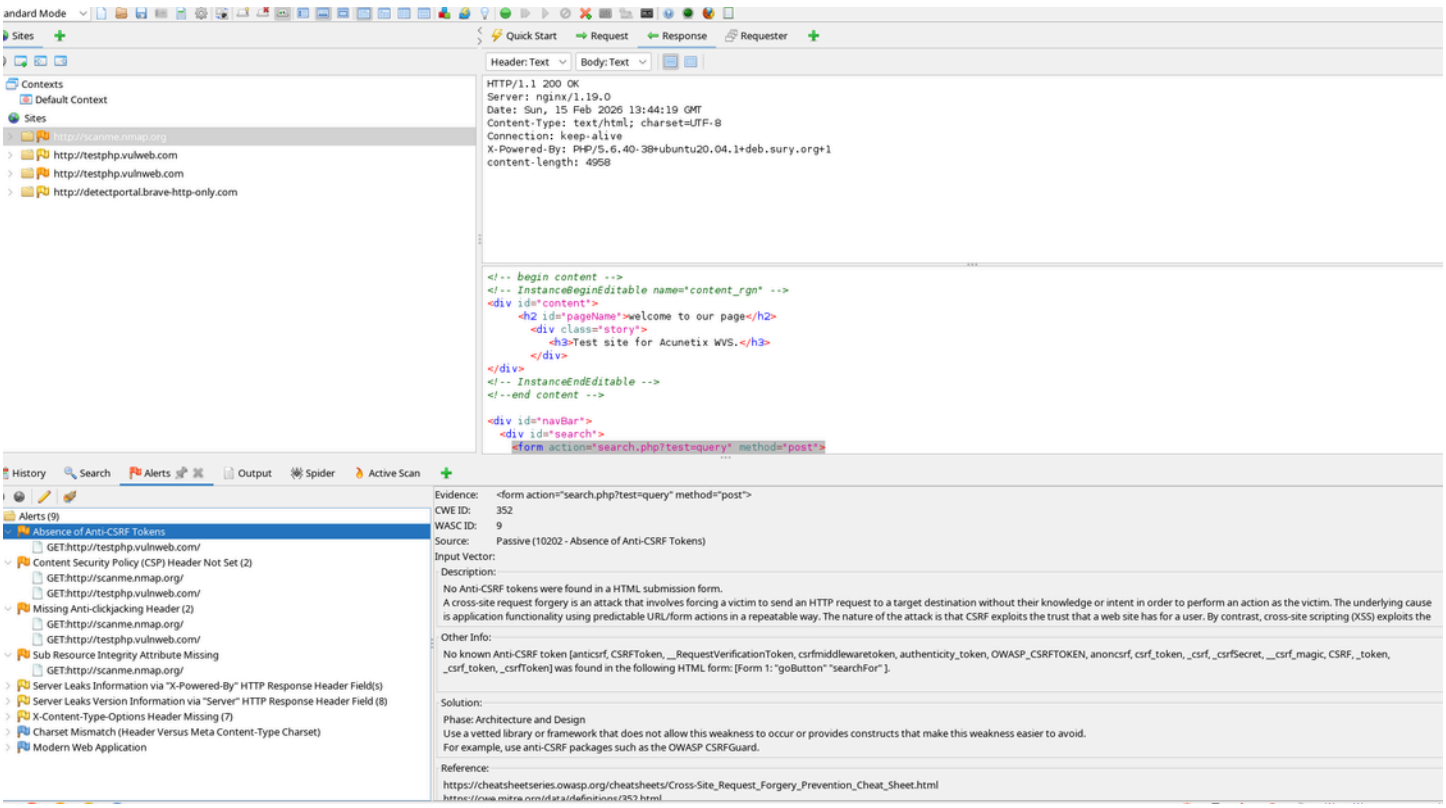

Figure 5: OWASP ZAP alert identifying missing anti-CSRF tokens

Conclusion & Recommendations

Summary of Findings

The vulnerability assessment conducted on **testphp.vulnweb.com** identified **four security vulnerabilities** across the application:

Risk Level	Count	Vulnerabilities
High	2	SQL Injection, Default Admin Credentials
Medium	2	Exposed CVS Directory, Missing CSRF Tokens

These findings represent significant security gaps that could be exploited by attackers to compromise the application, access sensitive data, and perform unauthorized actions.

Overall Risk Assessment

Overall Risk Level	**HIGH**
Confidence	High - Findings were manually verified
Business Impact	Critical - Customer data at risk, potential for complete compromise

The combination of **SQL Injection** and **Default Admin Credentials** creates a particularly dangerous scenario. An attacker could:

1. Discover the admin panel through directory enumeration
2. Log in using default credentials
3. Extract all customer data including credit card information
4. Use SQL injection to further compromise the database

Prioritized Remediation Roadmap

IMMEDIATE (Next 24-48 Hours)

Priority	Finding	Action Required
1	Default Admin Credentials	Change default test/test credentials immediately. Restrict admin panel access by IP.
2	SQL Injection	Implement parameterized queries for the cat parameter. Disable detailed error messages.

SHORT-TERM (Next 1-2 Weeks)

Priority	Finding	Action Required
3	Exposed CVS Directory	Block access to /CVS/ directory. Remove version control metadata from production.
4	Missing CSRF Tokens	Implement anti-CSRF tokens across all forms. Enable SameSite cookie attributes.

LONG-TERM (Ongoing)

- **Security Training:** Educate developers on secure coding practices (OWASP Top 10)
- **Automated Scanning:** Integrate security tools into CI/CD pipeline
- **Regular Assessments:** Conduct quarterly vulnerability assessments
- **Penetration Testing:** Schedule annual manual penetration tests
- **Incident Response:** Develop and test incident response procedures

Business Benefits of Remediation

Addressing these vulnerabilities will:

Benefit	Impact
Customer Trust	Protect user data and maintain brand reputation
Regulatory Compliance	Avoid GDPR, PCI DSS, and CCPA violations and fines
Operational Stability	Prevent data loss, corruption, and downtime
Competitive Advantage	Demonstrate security commitment to customers
Reduced Liability	Minimize risk of lawsuits and regulatory action

Final Remarks

While testphp.vulnweb.com is deliberately vulnerable for educational purposes, the issues identified here **mirror real-world vulnerabilities** found in production applications every day.

Organizations that fail to address these common security gaps risk:

- **Data breaches** exposing customer information
- **Financial losses** from fraud and remediation costs
- **Reputation damage** that takes years to rebuild
- **Legal penalties** from regulatory non-compliance

Security is not a one-time effort but an **ongoing process**. Regular testing, secure development practices, and a culture of security awareness are essential to protecting modern web applications.