ACME Corporation Web Application Security Assessment Report

https://www.acme.com

REPORT PUBLISH DATE **Tue May 23 2024 11:15:40 GMT+0000 (UTC)**

Contents

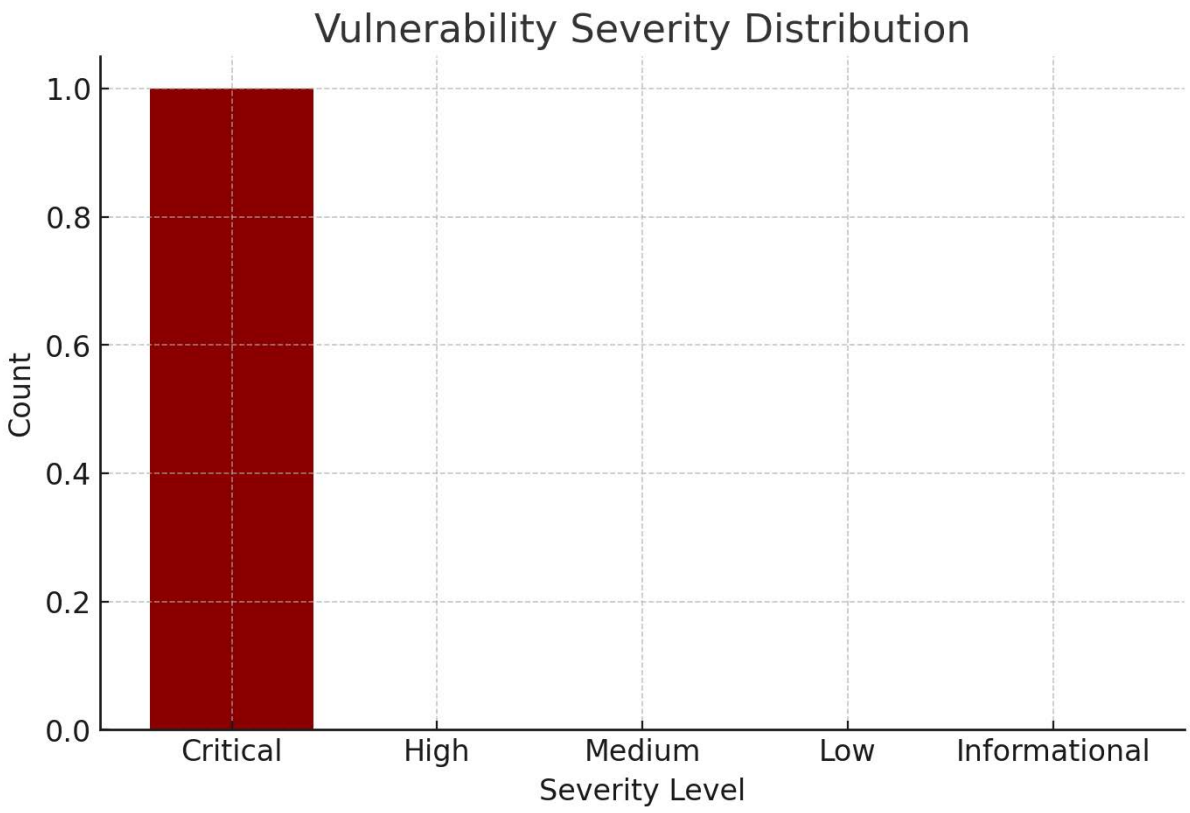
Executive Summary

AppSecure successfully conducted a comprehensive Vulnerability Assessment and Penetration Testing (VAPT) engagement for ACME Corporation’s web application. The objective was to identify security gaps that could expose the organization to potential threats and ensure compliance with industry best practices.

The assessment methodology included both manual and automated testing approaches, providing an in-depth evaluation of the application’s security posture. Testing was guided by the OWASP Top 10 Web Application Security Risks, covering critical areas such as authentication, access control, input validation, and session management.

During the assessment, AppSecure’s expert team identified one critical vulnerability, rated with a CVSS score of 7.0 or higher. While the overall number of findings is low, the criticality of this vulnerability represents a significant security risk to ACME Corporation if exploited by a malicious actor. Immediate remediation is strongly recommended.

Below is a graphical representation of the vulnerabilities discovered



Security Checklist

1. Identity Management Testing	
1.1 Test Role Definitions	1.2 Test User Registration Process
1.3 Test Account Provisioning Process	1.4 Testing for Account Enumeration and Guessable User Account
1.5 Testing for Weak or Unenforced Username Policy	

2. Authentication Testing	
2.1 Testing for Credentials Transported over an Encrypted Channel	2.2 Testing for Default Credentials
2.3 Testing for Weak Lock Out Mechanism	2.4 Testing for Bypassing Authentication Schema
2.5 Testing for Vulnerable Remember Password	2.6 Testing for Browser Cache Weaknesses
2.7 Testing for Weak Password Policy	2.8 Testing for Weak Security Question Answer
2.9 Testing for Weak Password Change or Reset Functionalities	2.10 Testing for Weaker Authentication in Alternative Channel

3. Authorization Testing	
3.1 Testing Directory Traversal File Include	3.2 Testing for Bypassing Authorization Schema
3.3 Testing for Privilege Escalation	3.4 Testing for Insecure Direct Object References

4. Session Management Testing	
4.1 Testing for Session Management Schema	4.2 Testing for Cookies Attributes
4.3 Testing for Session Fixation	4.4 Testing for Exposed Session Variables
4.5 Testing for Cross Site Request Forgery	4.6 Testing for Logout Functionality
4.7 Testing Session Timeout	4.8 Testing for Session Puzzling

5. Input Validation Testing	
5.1 Testing for Reflected Cross Site Scripting	5.2 Testing for Stored Cross Site Scripting
5.3 Testing for HTTP Verb Tampering	5.4 Testing for HTTP Parameter Pollution
5.5 Testing for SQL Injection	5.6 Testing for LDAP Injection
5.7 Testing for XML Injection	5.8 Testing for SSI Injection
5.9 Testing for XPath Injection	5.10 Testing for IMAP SMTP Injection
5.11 Testing for Code Injection	5.12 Testing for Command Injection
5.13 Testing for Buffer Overflow	5.14 Testing for Prompt Injection
5.15 Testing for HTTP Splitting Smuggling	5.16 Testing for HTTP Incoming Requests
5.17 Testing for Host Header Injection	5.18 Testing for Server Side Template Injection

6. Testing for Error Handling	
6.1 Testing for Error Code	6.2 Testing for Stack Traces

7. Testing for Weak Cryptography	
7.1 Testing for Weak SSL TLS Ciphers Insufficient Transport Layer Protection	7.2 Testing for Padding Oracle
7.3 Testing for Sensitive Information Sent via Unencrypted Channels	7.4 Testing for Weak Encryption

8. Business Logic Testing	
8.1 Introduction to Business Logic	8.2 Test Business Logic Data Validation
8.3 Test Ability to Forge Requests	8.4 Test Integrity Checks
8.5 Test for Process Timing	8.6 Test Number of Times a Function Can Be Used Limits
8.7 Testing for the Circumvention of Work Flows	8.8 Test Defenses Against Application Misuse
8.9 Test Upload of Unexpected File Types	8.10 Test Upload of Malicious Files

9. Client Side Testing	
9.1 Testing for DOM Based Cross Site Scripting	9.2 Testing for JavaScript Execution
9.3 Testing for HTML Injection	9.4 Testing for Client Side URL Redirect
9.5 Testing for CSS Injection	9.6 Testing for Client Side Resource Manipulation
9.7 Testing Cross Origin Resource Sharing	9.8 Testing for Cross Site Flashing
9.9 Testing for Clickjacking	9.10 Testing WebSockets
9.11 Testing Web Messaging	9.12 Testing Browser Storage
9.13 Testing for Cross Site Script Inclusion	

Scope of Work

Coverage

This penetration test was a manual assessment of the security of the app's functionality, business logic, and vulnerabilities such as those cataloged in the OWASP Top 10. The assessment also included a review of security controls and requirements listed in the OWASP Application Security Verification Standard (ASVS). The researchers rely on tools to facilitate their work, but the majority of the assessment involves manual analysis.

The following is a quick summary of the main tests performed on the web application:

- Authenticated user testing for session and authentication issues
- Authorization testing for privilege escalation and access control issues
- Input injection tests (SQL injection, XSS, and others)
- Platform configuration and infrastructure tests
- OWASP Top 10 Assessment

The team had access to authenticated users, enabling them to test security controls across roles and permissions.

Target description

The following URLs/apps were in scope for this assessment:

- https://www.acme.com

Assumptions/Constraints

1. The issues identified and proposed action plans in this report are based on our testing performed within the limited timespan and limited access to the servers. We made specific efforts to verify the accuracy and authenticity of the information gathered only in those cases where it was deemed necessary.

2. While precautions have been taken in the preparation of this document, Appsecure the publisher, and the author(s) assume no responsibility for errors, omissions, or for damages resulting from the use of the information contained herein. Use of Appsecure’s services does not guarantee the security of a system, or that computer intrusions will not occur.

3. Any configuration changes or software/hardware updates made on hosts/machines or on the application covered in this test after the date mentioned herein may impact the security posture either positively or negatively and hence invalidates the claims & observations in this report. Whenever there is a change in the architecture, we recommend that you conduct a vulnerability assessment and penetration test to ensure that your security posture is compliant with your security policies.

Methodology

The test was done according to penetration testing best practices. The flow from start to finish is listed below.

Pre Engagement

- Scoping
- Discovery

Penetration Testing

- Tool assisted assessment
- Manual assessment of OWASP top 10/SANS top 25 & business logic flaws
- Exploitation
- Risk analysis
- Reporting

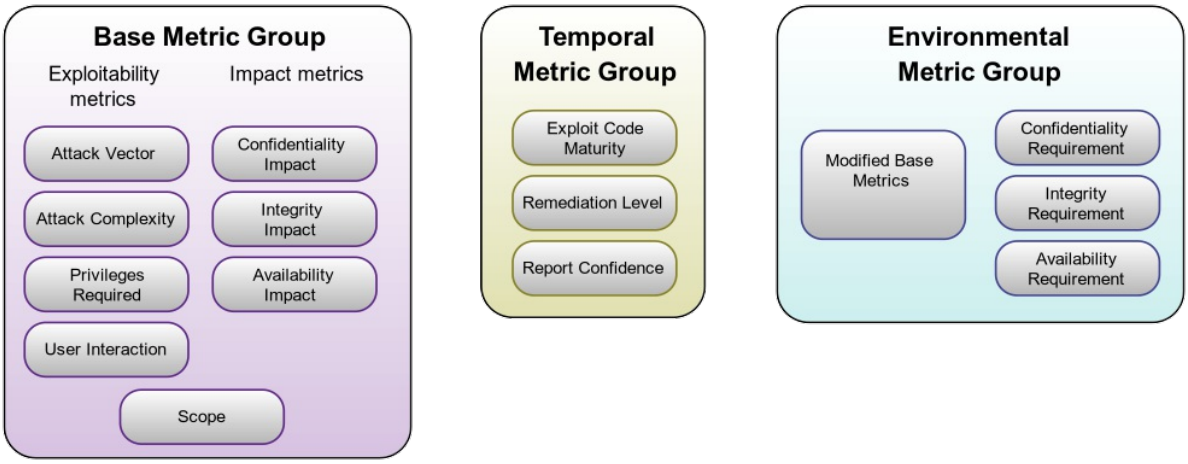
Post Engagement

- Best practice support
- Re-testing

Severity Ratings

The Common Vulnerability Scoring System (CVSS) v3.0 is a framework for rating the severity of security vulnerabilities in software. Operated by the Forum of Incident Response and Security Teams (FIRST), the CVSS uses an algorithm to determine three severity rating scores: Base, Temporal and Environmental. The scores are numeric; they range from 0.0 through 10.0 with 10.0 being the most severe.

CVSS is composed of three metric groups, Base, Temporal, and Environmental, each consisting of a set of metrics, as shown in below figure.



Severity Rating Scale

Findings are grouped into four criticality levels based on their risk.

Rating	CVSS Score
None	0.0
Low	0.1 - 3.9
Medium	4.0 - 6.9
High	7.0 - 8.9
Critical	9.0 - 10.0

Vulnerabilities Summary

S.NO.	VULNERABILITY TITLE	IMPACT	STATE
1	SQL Injection -'GET /product.php?pic=1'	Critical	Open

Appendix A - Vulnerability Summary & Recommendations

#1 SQL Injection -'GET /product.php?pic=1'

Critical SQL Injection

CVSS Score: 9.8

CVSS Vector: CVSS:3.0/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H

DESCRIPTION

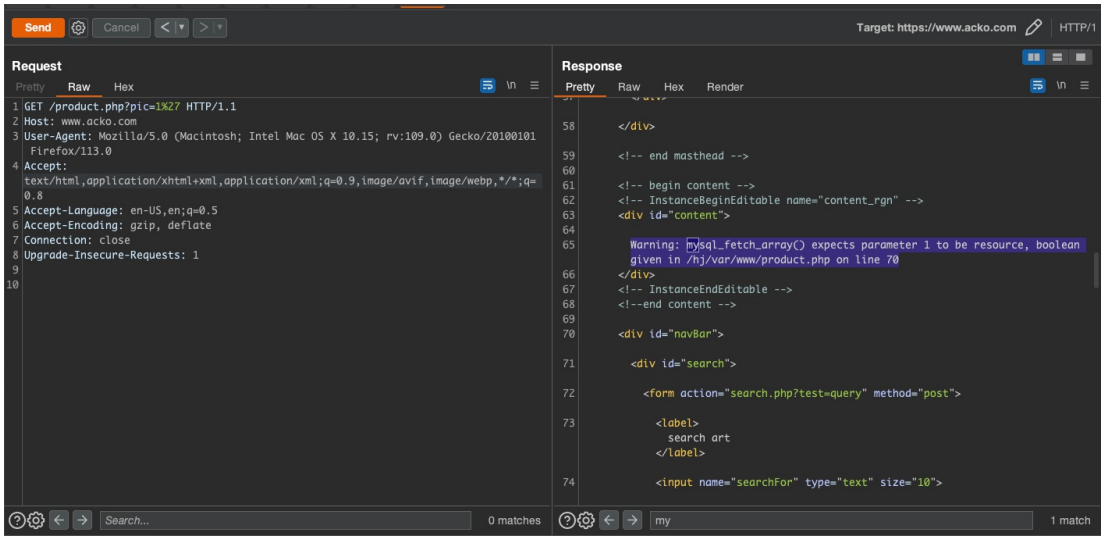
During the security assessment, it has been observed that the application is vulnerable to SQL injection on the 'pic' of the 'GET /product.php?pic=1'. An attacker can inject malicious SQL payloads in the vulnerable parameter and extract the sensitive information from the database.

SQL Injection (SQLi) is an injection attack that makes it possible to execute malicious SQL statements. These statements control a database server behind a web application. Attackers can use SQL Injection vulnerabilities to bypass application security measures. They can go around authentication and authorization of a web page or web application and retrieve the content of the entire SQL database. They can also use SQL Injection to add, modify, and delete records in the database.

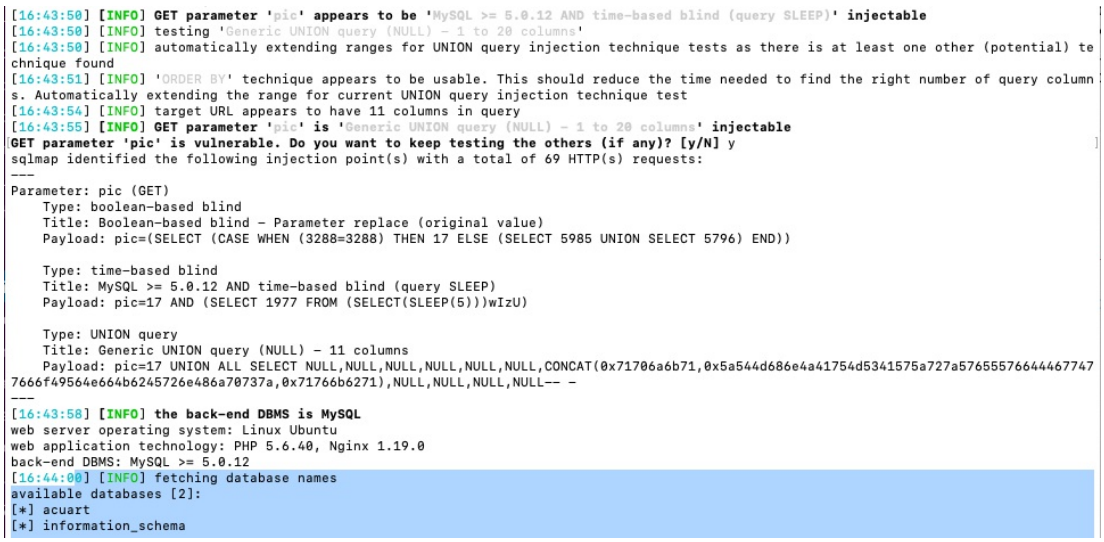
Reference: https://cwe.mitre.org/data/definitions/89.html

STEPS TO REPRODUCE

1. Navigate to the products module.
2. Click on any of the products.
3. Capture the vulnerable request using a proxy tool(Burp Suite).
4. Send the vulnerable request to the Burp Repeater.
5. . Add the single quote character (') in the 'pic' parameter. The application will respond with an SQL error.



Run the following SQLmap command - python3 sqlmap.py -u “<URL>” –dbs. The output will display the database list.



IMPACT

An attacker can get sensitive information from the database by inserting malicious SQL payloads in the vulnerable parameters.

HTTP REQUEST

```
GET /product.php?pic=1%27 HTTP/1.1
Host: www.acme.com
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:109.0)
Gecko/20100101 Firefox/113.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,*/*;q=
0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
```

SUGGESTED FIX

SQL injection can be prevented by using parameterized queries (also known as prepared statements) instead of string concatenation within the query.

Parameterized queries can be used for any situation where untrusted input appears as data within the query, including the WHERE clause and values in an INSERT or UPDATE statement. They can't be used to handle untrusted input in other parts of the query, such as table or column names or the ORDER BY clause. Application functionality that places untrusted data into those parts of the query must take a different approach, such as white-listing permitted input values or using different logic to deliver the required behaviour.

For a parameterized query to effectively prevent SQL injection, the string used in the query must always be a hard-coded constant. It must never contain any variable data from any origin. Do not be tempted to decide case-by-case whether an item of data is trusted, and continue using string concatenation within the query for safe cases. It is too easy to make mistakes about the possible origin of data or changes in other code to violate assumptions about what data is tainted.