February 30, 2024

SECURITY AUDIT REPORT FOR

XYZ Consulting

NEXUSCRYPT

DOCUMENT DETAILS

Title	Details
TEST TYPE:	WAPT (Web Application Penetration Testing)
STARTED ON:	30 February, 2024
COMPLETED ON:	31 February, 2024
DURATION:	30 Days

CONTACT INFORMATION

XYZ Consulting	
John J.	john@example.com
George L.	george@example.com
NexusCrypt	
Samyak Jain	samyak@nexuscrypt.com
Ajay Jain	ajay@nexuscrypt.com

Table of Contents

1. Executive Summary

This document contains the initial security assessment report for :

XYZ Consulting Dasbboard

The purpose of this assessment was to point out security loopholes, business logic errors, and missing best security practices. The tests were carried out assuming the identity of an attacker or a malicious user but no harm was made to the functionality or working of the application/network.

1.1 Scope of Testing

Security assessment includes testing for security loopholes in the scope defined below. Apart from the following, no other information was provided. Nothing was assumed at the start of the security assessment.

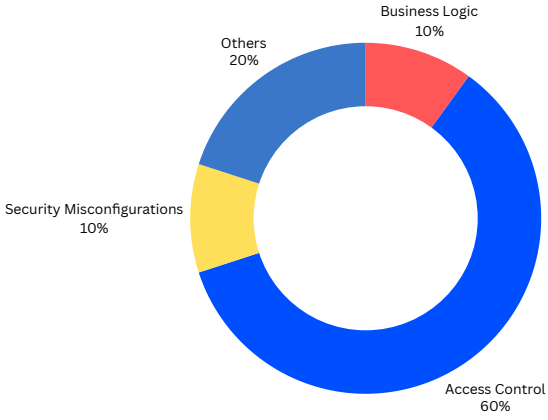
The following was the scope covered under the security audit:

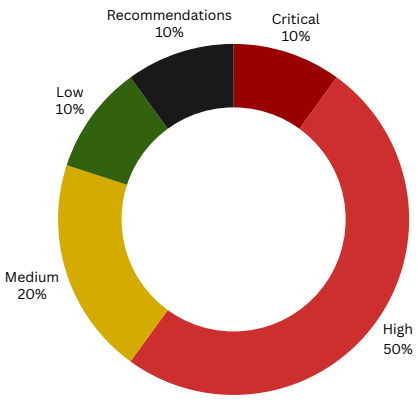
Application 1: auth.example.com

Application 2: api.example.com

1.2 Graphical Summary

The below graphical representations will provide you an overall summary of the security audit scan results, including, vulnerabilities discovered, severity, respective CVSS Score, and other vulnerability details such as its impact, detailed PoC, steps to reproduce, affected URLs/network parameters, and recommended fixes.





1.3 List of Vulnerabilities

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
1	2	2	2	1

FINDING	SEVERITY
SQL Injection x 3	CRITICAL
IDORs	HIGH
Stored XSS	HIGH
Reflected XSS	MEDIUM
Information Disclosure	MEDIUM
Server-Side Request Forgery (SSRF)	LOW
Cross-Site Request Forgery (CSRF)	LOW
Lack of Current Password	INFORMATIONAL

2. Discovered Vulnerabilities Details

Vulnerability #1

SQL Injection

Severity: **Status:**

Critical **Unresolved**

Affected URLs:

https://example.com/?test={payload}

https://example.com/about

https://example.com/contact

Details of Vulnerability:

SQL injection vulnerabilities arise when user-controllable data is incorporated into database SQL queries in an unsafe manner. An attacker can supply crafted input to break out of the data context in which their input appears and interfere with the structure of the surrounding query.

Steps to reproduce:

1. Go to: https://example.com/?test={payload}

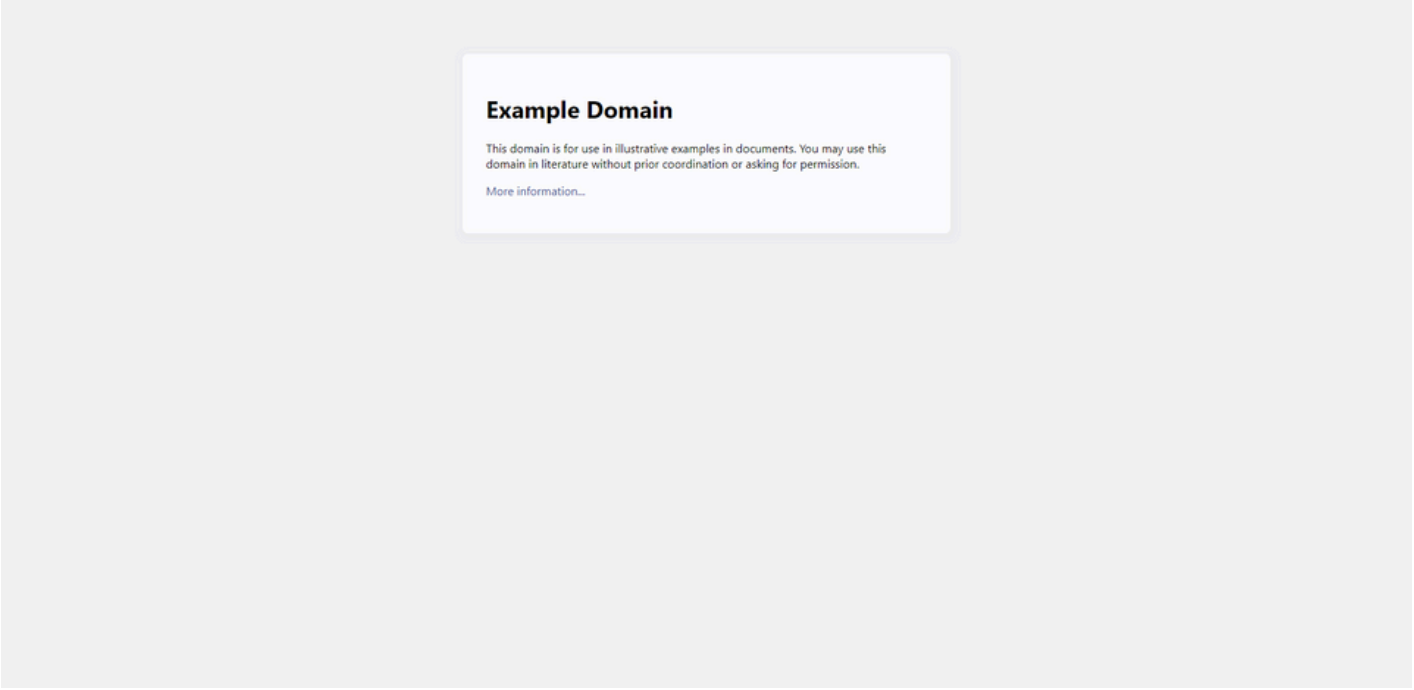
2. Proxy your data throw burp suite and turn interception on.

3. After forwarding few requests, you will get a request with the following endpoint, send it to repeater: contactus/?test=

4. In the “?test=” parameter, add the following SQL Payload: 'SELECT *’

5. The payload contains a sleep action, which will make the database sleep for around 20 milliseconds, send the request.

6. After sending the request, in the right hand down side, notice that the request took 20 milliseconds to complete, confirming our payload has been successfully executed which confirms the SQL Injection vulnerability.



Note:

Please find the requests for all three vulnerable endpoints and video pocs in resources folder.

Suggested Fix:

The most effective way to prevent SQL injection attacks is to use parameterized queries (also known as prepared statements) for all database access. This method uses two steps to incorporate potentially tainted data into SQL queries: first, the application specifies the structure of the query, leaving placeholders for each item of user input; second, the application specifies the contents of each placeholder. Because the structure of the query has already been defined in the first step, it is not possible for malformed data in the second step to interfere with the query structure. You should review the documentation for your database and application platform to determine the appropriate APIs which you can use to perform parameterized queries.

Additional References:

- https://example.com/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html
- https://www.example.com/websitesecurity/sql-injection/

3. List of VAPT Tests Performed

The following lists of tests are suggestive & not limited to the ones listed. Most importantly, every test case has multiple sub-test cases ranging from a few to sometimes 1000+ sub tests.

Additional test cases will be performed based on factors such as:

1. Technology Stack
2. Server Side Programming Language, Front-end frameworks
3. Framework/CMS/APIs
4. Type of application (Payment integrations, external integrations)

3.1 OWASP Top 10

#	OWASP Top 10
	for **Web Applications**
1	SQL Injection
2	Broken Authentication
3	Sensitive Data Exposure
4	XML External Entities (XXL)
5	Broken Access Control
6	Security Misconfiguration
7	Cross-Site Scripting (XSS)
8	Insecure Deserialization
9	Using Components with Known Vulnerabilities
10	Insufficient Logging and Monitoring
	for **Mobile Applications**
1	Improper Platform Usage
2	Insecure Data Storage
3	Insecure Communication
4	Insecure Authentication
5	Insufficient Cryptography
6	Insecure Authorization
7	Client Mode Quality
8	Code Tampering
9	Reverse Engineering
10	Extraneous Functionality

3.2 SANS 25 Software Errors/Tests

#	SANS 25
1	Improper Restriction of Operations within the Bounds of a Memory Buffer
2	Improper Neutralization of Input During Web Page Generation ('XSS')
3	Improper Input Validation
4	Information Exposure
5	Out-of-bounds Read
6	Improper Neutralization of Special Elements used in an SQL Command (SQLi)
7	Use After Free
8	Integer Overflow or Wraparound
9	Cross-Site Request Forgery (CSRF)
10	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')
11	Improper Neutralization of Special Elements used in an OS Command
12	Out-of-bounds Write
13	Improper Authentication
14	NULL Pointer Dereference
15	Incorrect Permission Assignment for Critical Resource
16	Unrestricted Upload of File with Dangerous Type
17	Improper Restriction of XML External Entity Reference
18	Improper Control of Generation of Code ('Code Injection')
19	Use of Hard-coded Credentials
20	Uncontrolled Resource Consumption
21	Missing Release of Resource after Effective Lifetime
22	Untrusted Search Path
23	Deserialization of Untrusted Data
24	Improper Privilege Management
25	Improper Certificate Validation

3.3 174 Other Test Cases

#	Other Tests	Typical Severity
1	OS Command Injection	High
2	SQL Injection (Second Order)	High
3	XML External Entity Injection	High
4	LDAP Injection	High
5	XPath Injection	High
6	XML Injection	High
7	ASP.NET Debugging Enabled	High
8	DoS Locking Customer Accounts	Medium
9	DoS Buffer Overflows	Medium
10	Storing too much data in session (DoS)	High
11	Writing user-provided data to disk (DoS)	High
12	HTTP Insecure methods available on Server	High
13	Out of band resource load (HTTP)	High
14	File path manipulation	High
15	Server-site JavaScript code injection	High
16	Perl code injection	High
17	Ruby code injection	High
18	Python code injection	High
19	Expression Language injection	High
20	Unidentified code injection	High
21	Server-side template injection	High
22	SSL injection	High
23	Stored XSS	High
24	HTTP response header injection	High
25	Reflected XSS	High
26	Client-side template injection	High
27	DOM-based XSS	High
28	Reflected DOM-based XSS	High
29	Stored DOM-based XSS	High
30	DOM-based JavaScript Injection	High
31	Reflected DOM-based JavaScript Injection	High
32	Stored DOM-based JavaScript Injection	High
33	Path-relative style sheet import	Information
34	Client-side SQLi (DOM-based)	High
35	Client-side SQLi (Reflected DOM-based)	High
36	Client-side SQLi (Stored DOM-based)	High

THE END

Questions? Contact us at
contact@nexuscrypt.com