SQL Injection Demo
Security Assessment Findings Report

Business Confidential

Date: January 8, 2026
Version 1.0

Contents

Confidentiality statement

This document is the exclusive property of Andrew Myrden (canandrew). This document contains hypothetical - FOR DEMONSTRATION, NOT ACTUAL - proprietary and confidential information. Duplication, redistribution, or use, in whole or in part, in any form, requires consent from Andrew.

This document may be shared with auditors under non-disclosure agreements to demonstrate penetration test requirement compliance.

Disclaimer

A penetration test is considered a snapshot in time. The findings and recommendations reflect the information gathered during the assessment and not any changes or modifications made outside of that period.

Time-limited engagements do not allow for a full evaluation of all security controls. The assessment is prioritized to identify the weakest security controls an attacker would exploit. We recommend conducting similar assessments on an annual basis by internal or third-party assessors to ensure the continued success of the controls.

Assessment overview

From January 5th, 2026 to January 8th, 2026, an evaluation of the security posture and infrastructure compared to current industry best practices that included an external penetration test. All testing performed is based on the NIST CSF and ISO 27001 guidelines.

Phases of penetration testing activities include the following:

- Planning – Goals are gathered and rules of engagement obtained.
- Discovery – Perform scanning and enumeration to identify potential vulnerabilities, weak areas, and exploits.
- Attack – Confirm potential vulnerabilities through exploitation and perform additional discovery upon new access.
- Reporting – Document all found vulnerabilities and exploits, failed attempts, and company strengths and weaknesses.

Assessment components

External penetration test

An external penetration test emulates the role of an attacker attempting to gain access to the fake login page, representing a SQL injection in order to bypass website security. The attacker (Andrew) attempts to gather common SQL injection methods used in modern attacks, including 'In-band SQLi' which involves the common website communication channel. Andrew also performs scanning and reads through source code (available through developer tools) to look for vulnerabilities in code.

Finding severity ratings

The following table defines levels of severity and corresponding CVSS score range that are used throughout the document to assess vulnerability and risk impact.

Severity	CVSS V3 score range	Definition
Critical	9.0 – 10.0	Exploitation is straightforward and usually results in system-level compromise. It is advised to form a plan of action and patch immediately.
High	7.0 – 8.9	Exploitation is more difficult but could cause elevated privileges and potentially a loss of data or downtime. It is advised to form a plan of action and patch as soon as possible.
Moderate	4.0 – 6.9	Vulnerabilities exist but are not exploitable or require extra steps such as social engineering. It is advised to form a plan of action and patch after high-priority issues have been resolved.
Low	0.1 – 3.9	Vulnerabilities are non-exploitable but increase an organisation's attack surface. It is advised to form a plan of action and patch during the next maintenance window.
Informational	N/A	No known vulnerability exists. Additional information is provided regarding items noticed during testing, strong controls, and additional documentation.

Scope

Scope exclusions

As common in the workforce, no Denial of Service attacks were attempted in order to maintain availability.

Client allowances

N/A; no allowances were needed to assist the testing.

Executive summary

Andrew evaluated the security posture of the sample login page through a series of basic SQL inject attempts. Out of surprise, efforts failed to bypass the login system, even with how unsecure and horribly coded it was. After some research, it appears that the Apache server stack 'XAMPP' – which was used to host this simulation – contains built-in security measures to protect form-input from containing SQL commands. Although the goal of this project (gaining access through SQL injections) failed, it highlights the importance and effectiveness of modern host solutions having the ability to protect against basic attacks.

Attack summary

The following list describes how login access would be gained in a SQL injection (with a vulnerable host), step by step:

1. Access the site webpage.
2. Locate the login form, read through the source code to find vulnerabilities relating to input sanitation and/or filters.
3. Build an SQL command with the goal of tricking the program into evaluating the login attempt as 'TRUE', such as `' OR 1=1 --`
    - The sample command above works from the "1=1" being evaluated to TRUE; making the logical OR condition evaluate to TRUE overall.
4. Paste the SQL command into the password input, input any single character as the username input.
    - Many outdated websites only care to encrypt users' passwords, so the username isn't the target of the SQL injection in this hypothetical.
5. Gain access, or if failure, study any error messages to build a new SQL command and try again.

Security strengths

Host Built-in Prevention Measures

During the assessment, it became clear that MariaDB (through XAMPP's MySQL hosting) contains numerous built-in measures to prevent basic SQL injection attacks. It appears that MariaDB treats standard SQL terminators such as `--`, `/*`, and `#` as literal characters when used in string contexts. This prevents attackers from using the comment to ignore trailing quotes in the code.

Security weaknesses

Missing input sanitation

Login validation scripts (JavaScript) were missing input sanitation. Without the built-in prevention measures from MariaDB, this would allow non-strings to be inputted into the source code, enabling attacks such as SQL injections, or even throwing unhandled errors.

Lack of username encryption

Though passwords *were* encrypted during MySQL databse storage, usernames were not. It's always good practice to encrypt database information, especially when it contains crucial data including login credentials. This would not slow down logins whatsoever, and there is no reason *not* to implement this.

Unrestricted login attempts

Multiple brute-force attacks against login forms were able to be executed. For all logins, unlimited attempts were allowed, which permits the idea of brute force attacking. Denial of service attacks could also be carried out through this weakness.

External penetration test findings

Lack of Input Sanitation & Filtering – Login Form (Critical)

Exploitation proof of concept

Andrew gathered common SQL injection attack string sequences. These can easily be used to trick a vulnerable program into reading a password input containing keywords such as 'TRUE', then giving complete access to sensitive information that is meant to be protected by such a login form.

Remediation

Item 1: Login did not require Multi-Factor Authentication (MFA). Andrew recommends implementing and enforcing MFA across all external-facing login services.

Item 2: Unlimited login attempts were permitted. Andrew recommends implementing some sort of service rule restricting number of login attempts.

Item 3: No encryption of usernames. Having a second piece of information which is encrypted makes it exponentially harder for attackers to breach the login form.