ISO 27001 Incident Management - SQL Injection in DVWA

Introduction

This document details a security incident identified in the Damn Vulnerable Web Application (DVWA) platform, specifically in the "SQL Injection" module. This incident was caused by the successful execution of an SQL injection attack, which highlights a critical vulnerability in the mechanisms for validating and protecting user inputs. The report includes a description of the incident, its reproduction, potential impact, and remediation measures.

Incident Description

The SQL injection attack was carried out in the "SQL Injection" section of DVWA, a platform designed for vulnerability testing. A payload was introduced into the "User ID" field to manipulate SQL queries and access unauthorized information. This incident reflects the lack of adequate input validation and sanitization mechanisms.

SQL Injection Method Used

To replicate and demonstrate the vulnerability, the following SQL payload was used in the "User ID" field:

Sql

```
1' OR '1'='1
```

By executing the payload, access was gained to data stored in the database, confirming the exploitation of the vulnerability.

Reproduction Steps

1. Log in to DVWA and access the "SQL Injection" module.
2. Set DVWA's security level to "Low" to facilitate the test.
3. Enter the payload `1' OR '1'='1` in the "User ID" field.
4. Press the "Submit" button.
5. Observe the result, which displays sensitive database information, indicating unauthorized access.

Incident Impact

The impact of this vulnerability can be critical in production systems. The identified risks include:

- Unauthorized access to confidential data.
- Possibility of data extraction, modification, or deletion.
- Compromise of system integrity.
- Financial, legal, and reputational risks for affected organizations.

Although DVWA is a platform designed with intentional vulnerabilities, this scenario reflects a real risk that may exist in poorly protected web applications.

Recommendations

1. **Implement input validation:** Use sanitization and validation mechanisms for all user inputs to prevent malicious commands from being processed by the system.
2. **Use parameterized queries:** Replace dynamic SQL queries with prepared statements or stored procedures.
3. **Enable WAF (Web Application Firewall):** Implement a firewall capable of detecting and mitigating SQL injection attacks.
4. **Conduct regular security audits:** Perform regular evaluations to identify and address vulnerabilities in the system.
5. **Train development personnel:** Ensure developers are familiar with secure coding practices.

Conclusions

The SQL injection attack demonstrates the critical importance of implementing robust security controls in web applications. While DVWA is designed for educational and testing purposes, the lessons learned from this incident underscore the necessity of adopting secure coding practices and effective mitigation mechanisms in real-world environments.