ASSIGNMENT

Database security

Database security assignment to fulfill the criteria for the

Database Fundamentals course in the Informatics program

Pramudya Arya Wicaksana
2242805

Teknik Informatika
STMIK AMIK Bandung

March 11, 2023

Contents

1 Sql Injection **2**
1.1 How to prevent SQL Injection 2

2 Cross-site Scripting **5**
2.1 How to prevent XSS 6

3 Denial-of-Service **8**
3.1 How to prevent Database DDOS 8

4 Bruteforce Attack **10**
4.1 How to prevent Database Bruteforce attack 10

1 Sql Injection

SQL injection is a type of attack on a database-driven application that allows an attacker to execute unauthorized SQL commands by injecting malicious SQL statements into an application's input fields. SQL injection occurs when an attacker can input data into an application without proper input validation, such as input sanitization, and then that input is passed to the application's SQL query.

Example of popular SQL Injection code

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

This code is designed to manipulate the application's SQL query by injecting a condition that is always true, such as 1=1. The double dash at the end is used to comment out the rest of the SQL code, effectively neutralizing it. This SQL code would then be executed by the application, potentially granting the attacker access to sensitive data or control over the database.

SQL Injection attacks can be prevented through various measures such as using prepared statements with parameterized queries, validating and sanitizing user inputs, limiting database privileges, and avoiding dynamic SQL. Web developers and database administrators should take appropriate security measures to protect against SQL Injection attacks.

1.1 How to prevent SQL Injection

Preventing SQL Injection attacks requires a multi-layered approach, involving both web developers and database administrators. Here are some measures that can be taken to prevent SQL Injection:

- Parameterized queries and prepared statements: Web developers should use parameterized queries and prepared statements instead of dynamically building SQL statements with user inputs. Parameterized queries separate the SQL code from the input values, preventing malicious input from modifying the SQL statement. Prepared statements precompile the SQL statement, which can improve performance and prevent SQL Injection attacks.

Golang example for parameterized query

```
// Correct format for executing an SQL statement with parameters.
rows, err := db.Query("SELECT * FROM user WHERE id = ?", id)
// SECURITY RISK!
rows, err := db.Query(fmt.Sprintf("SELECT * FROM user WHERE id = %s", id))
```

Listing 1: Golang code example

- Input validation and sanitization: Web developers should validate and sanitize all user inputs to ensure that they conform to expected formats and do not contain malicious code. This can include filtering out special characters, checking input length, and enforcing input types.

- Least privilege: Database administrators should limit the privileges of database users and ensure that they only have access to the data and functionality they need. This can prevent attackers from using SQL Injection to gain access to sensitive data or perform unauthorized actions.

- Avoid dynamic SQL: Web developers should avoid dynamically building SQL statements from user inputs whenever possible. This can reduce the risk of SQL Injection attacks.

- Keep software up-to-date: Both web developers and database administrators should keep their software up-to-date with the latest security patches and updates to prevent known vulnerabilities.

- Use a web application firewall (WAF): A WAF can help protect against SQL Injection attacks by filtering out malicious inputs and preventing unauthorized access to the database.

By implementing these measures, web developers and database administrators can significantly reduce the risk of SQL Injection attacks and improve the overall security of their web applications, most importantly tell the developer to properly code the application, not just let it be if it's done, do some unit testing also

2 Cross-site Scripting

Cross-site scripting (XSS) is a type of security vulnerability that can occur in web applications, including those that use databases. XSS attacks involve the insertion of malicious code, typically in the form of a script, into a web page viewed by other users. When a user visits the affected web page, the malicious script is executed in the user's web browser, allowing the attacker to steal sensitive data, such as login credentials or personal information, or perform unauthorized actions on behalf of the user.

XSS attacks can occur in web applications that use databases when the application fails to properly validate user input or sanitize data before displaying it in a web page. For example, if a web application allows users to submit comments that are displayed on a web page, an attacker could insert a script in the comment that is executed when other users view the page.

To prevent XSS attacks in database-driven web applications, web developers should implement proper input validation and sanitization techniques. This can include filtering out special characters, checking input length, and enforcing input types. Additionally, web developers should use encoding techniques to ensure that user input is properly displayed as plain text, rather than being interpreted as executable code.

Database administrators can also play a role in preventing XSS attacks by ensuring that the database is properly configured and secure. This can include limiting user privileges, encrypting sensitive data, and monitoring database activity for signs of unauthorized access or malicious activity.

Overall, preventing XSS attacks requires a multi-layered approach that involves both web developers and database administrators working together to ensure that web applications are secure and resistant to attacks.

Example of common XSS pattern

```
<script type=”text/javascript”>
var test=’../example.php?cookie_data=’+escape(document.cookie);
</script>
```

2.1 How to prevent XSS

Preventing cross-site scripting (XSS) attacks requires a multi-layered approach that involves both web developers and database administrators. Here are some measures that can be taken to prevent XSS attacks:

- Input validation and sanitization: Web developers should validate and sanitize all user inputs to ensure that they conform to expected formats and do not contain malicious code. This can include filtering out special characters, checking input length, and enforcing input types.

- Output encoding: Web developers should use output encoding techniques to ensure that user input is properly displayed as plain text, rather than being interpreted as executable code. This can include HTML encoding, URL encoding, and JavaScript encoding.

- Content Security Policy (CSP): A Content Security Policy (CSP) is a security mechanism that can be used to prevent XSS attacks by allowing web developers to specify the sources of content that are allowed to be loaded on a web page. CSP can be used to block the execution of malicious scripts and other types of content that could be used in an XSS attack.

- HTTP-only cookies: Cookies that are marked as HTTP-only cannot be accessed by JavaScript, preventing attackers from stealing sensitive data such as session IDs or authentication tokens.

- Keep software up-to-date: Both web developers and database administrators should keep their software up-to-date with the latest security patches and updates to prevent known vulnerabilities.

- Use a web application firewall (WAF): A WAF can help protect against XSS attacks by filtering out malicious inputs and preventing unauthorized access to the database.

By implementing these measures, web developers and database administrators can significantly reduce the risk of XSS attacks and improve the overall security of their web applications.

3 Denial-of-Service

A database DDoS (Distributed Denial of Service) attack is a type of cyber attack in which a large number of requests are sent to a database server with the intent of overwhelming its resources and rendering it unavailable to users.

In a database DDoS attack, the attacker typically uses a network of compromised devices (also known as a botnet) to send a massive number of requests to the targeted database server. This flood of requests can quickly consume the server's resources, causing it to become unresponsive or crash.

Database DDoS attacks can have serious consequences, including loss of data, business disruption, and financial loss. They can be difficult to detect and mitigate, as they often appear as legitimate traffic until the attack reaches a critical level.

To prevent database DDoS attacks, database administrators should implement strong security measures, such as firewalls and intrusion detection systems, to monitor incoming traffic and block malicious requests. Additionally, regular backups should be made to ensure that data can be restored in the event of a successful attack.

3.1 How to prevent Database DDOS

Preventing database DDoS (Distributed Denial of Service) attacks requires a multi-layered approach that involves both database administrators and network security professionals. Here are some measures that can be taken to prevent database DDoS attacks:

- Use firewalls: A firewall can help protect the database from DDoS attacks by filtering out incoming traffic and blocking malicious requests. Firewalls should be configured to allow only authorized traffic to reach the database server.

- Implement intrusion detection/prevention systems (IDS/IPS): An IDS/IPS can help detect and prevent DDoS attacks by monitoring network traffic and identifying abnormal patterns or behaviors that may indicate an attack.

- Use rate limiting: Rate limiting can help prevent DDoS attacks by limiting the number of requests that are allowed to be made to the database server from a single IP address or user account.

- Monitor network traffic: Database administrators should regularly monitor network traffic to identify and respond to any suspicious activity or unusual traffic patterns that may indicate an ongoing DDoS attack.

- Implement load balancing: Load balancing can help distribute traffic across multiple servers, reducing the impact of a DDoS attack on any single server.

- Keep software up-to-date: Both database administrators and network security professionals should keep their software up-to-date with the latest security patches and updates to prevent known vulnerabilities that can be exploited in a DDoS attack.

By implementing these measures, database administrators and network security professionals can significantly reduce the risk of DDoS attacks and improve the overall security of their databases.

4 Bruteforce Attack

A database brute force attack is a type of cyber attack in which an attacker attempts to gain unauthorized access to a database by trying a large number of password combinations until the correct one is found.

In a brute force attack, the attacker uses automated tools or scripts to rapidly generate and test different password combinations until the correct one is found. This method is time-consuming and requires a significant amount of computing power, but it can be effective against weak or easily guessed passwords.

Once the attacker gains access to the database, they may be able to steal sensitive information, modify or delete data, or use the compromised account to launch further attacks. To prevent database brute force attacks, it is important to use strong passwords, limit login attempts, and implement other security measures such as two-factor authentication and encryption.

4.1 How to prevent Database Bruteforce attack

There are several ways to prevent database brute force attacks:

- Use strong passwords: Make sure that all user accounts in the database have strong passwords that are difficult to guess. A strong password should be at least 8-12 characters long, include a mix of upper and lower case letters, numbers, and special characters, and should not include common words or phrases.

- Implement account lockout policies: Set up your database to lock out users after a certain number of failed login attempts. This will prevent an attacker from using automated tools to repeatedly try different passwords.

- Use two-factor authentication: Two-factor authentication requires users to provide a second form of authentication, such as a security token or biometric factor, in addition to a password. This can significantly reduce the risk of a successful brute force attack.

- Use intrusion detection and prevention systems: These systems can detect and prevent brute force attacks by analyzing login attempts and blocking traffic from known malicious sources.

- Implement encryption: Encrypting sensitive data in the database can prevent an attacker from accessing it even if they do manage to gain access to the database.

- Regularly update and patch your database: Keep your database software up to date with the latest security patches to prevent known vulnerabilities from being exploited.

By implementing these measures, you can significantly reduce the risk of a successful database brute force attack.