SQL Injection, DNS Tunnelling, and CSRF Threat Scenarios

Following is a detailed description of all possible threat scenarios related to SQL Injection, DNS Tunnelling, and Cross-Site Request Forgery that this monitoring solution shall help detect, mitigate, and respond against.

SQL Injection (SQLi) Threat Scenarios

1. Authentication Bypass

Scenario: Attackers inject SQL code into login forms to always evaluate true, allowing unauthorised access to user accounts.

Impact: Unauthorised access to user accounts and sensitive information.

Monitoring and Alerting:

- **Detection:** Monitor web server logs for suspicious login patterns or anomalous queries in the login endpoint.
- **Alert:** Trigger an alert when multiple failed logins are followed by a successful login without password change.

2. Data Extraction

Scenario: Attackers inject SQL code to extract sensitive data such as usernames, passwords, and credit card details.

Impact: Exposure of confidential information, potential identity theft, and financial loss.

Monitoring and Alerting:

- **Detection:** Monitor database logs for UNION SELECT patterns in queries.
- **Alert:** Trigger an alert when UNION SELECT queries are detected from user inputs.
-

3. Error-Based SQL Injection

Scenario: Attackers use error messages to gather information about the database structure.
Impact: Provides attackers with valuable information for further exploitation.

Monitoring and Alerting:

- **Detection:** Monitor for specific error messages in web server logs.
- **Alert:** Trigger an alert when error messages related to SQL syntax or type conversion are detected.

4. Second-Order SQL Injection

Scenario: Insert malicious SQL code through one function to be executed later.

Impact: Delayed execution of attacks, potentially causing significant damage.

Monitoring and Alerting:

- **Detection:** Monitor for suspicious SQL queries being stored in the database.
- **Alert:** Trigger an alert when patterns indicating second-order SQL injection are detected.

5. Command Execution

Scenario: Execute commands on the database server, leading to further system compromise.

Impact: Full control over the server, leading to data theft, destruction, or unauthorised use.

Monitoring and Alerting:

- **Detection:** Monitor for SQL queries that attempt to execute system commands.
- **Alert:** Trigger an alert when system command execution via SQL is detected.

6. Data Manipulation

Scenario: Inject SQL code to modify or delete data in the database.

Impact: Data corruption, loss of data integrity, and potential business impact.

Monitoring and Alerting:

- **Detection:** Monitor for SQL UPDATE or DELETE queries originating from user inputs.
- **Alert:** Trigger an alert when unauthorised data modification patterns are detected.

7. Database Fingerprinting

Scenario: Inject SQL code to identify database type and version to tailor attacks.

Impact: Customised attacks based on specific database vulnerabilities.

Monitoring and Alerting:

- **Detection:** Monitor for SQL queries that attempt to retrieve database metadata.
- **Alert:** Trigger an alert when queries aimed at database fingerprinting are detected.

8. Privilege Escalation

Scenario: Inject SQL code to elevate database user privileges.

Impact: Unauthorised access to administrative functions and sensitive data.

Monitoring and Alerting:

- **Detection:** Monitor for SQL queries that attempt to access or modify user privileges.
- **Alert:** Trigger an alert when privilege escalation attempts are detected.

9. Enumeration of Database Schema

Scenario: Inject SQL code to list tables and columns to understand the database structure.
Impact: Detailed knowledge of the database structure, facilitating further attacks.
Monitoring and Alerting:

- **Detection:** Monitor for SQL queries that access information_schema or metadata tables.
- **Alert:** Trigger an alert when schema enumeration attempts are detected.

DNS Tunnelling Threat Scenarios

1. Data Exfiltration

Scenario: Use DNS queries to transfer data out of the network.

Impact: Unauthorised data transfer, leading to data breaches and loss of sensitive information.

Monitoring and Alerting:

- **Detection:** Monitor DNS logs for unusually large volumes of DNS queries or responses containing encoded data.
- **Alert:** Trigger an alert when patterns indicative of data exfiltration via DNS are detected.

2. Command and Control (C2) Communication

Scenario: Use DNS queries to establish a command and control channel.

Impact: Persistent control over infected devices, enabling further malicious activities.

Monitoring and Alerting:

- **Detection:** Monitor DNS logs for query patterns associated with C2 traffic.
- **Alert:** Trigger an alert when DNS queries/responses match known C2 communication patterns.

3. Malware Download

Scenario: Use DNS queries to download malware or additional payloads.

Impact: Further infection of systems, leading to broader attacks and increased damage.

Monitoring and Alerting:

- **Detection:** Monitor DNS logs for queries that result in large data transfers or contain encoded payloads.
- **Alert:** Trigger an alert when DNS queries indicative of malware download are detected.

4. DNS Amplification Attack

Scenario: Use DNS queries to amplify traffic directed at a target.

Impact: Disruption of services, affecting availability and performance.

Monitoring and Alerting:

- **Detection:** Monitor DNS logs for patterns of queries that result in significantly larger responses.
- **Alert:** Trigger an alert when amplification patterns are detected.

5. Evasion of Network Controls

Scenario: Use DNS tunnelling to bypass network security controls and filters.

Impact: Unmonitored and unauthorised communication, increasing the difficulty of detection and mitigation.

Monitoring and Alerting:

- **Detection:** Monitor DNS logs for unusual query patterns or encoded data.
- **Alert:** Trigger an alert when DNS queries that appear to tunnel traffic are detected.

Cross-Site Request Forgery (XSRF/CSRF) Threat Scenarios

1. Unauthorised Actions

Scenario: Trick authenticated users into executing unauthorised actions.

Impact: Unauthorised changes to user settings, financial transactions, or other sensitive operations.

Monitoring and Alerting:

- **Detection:** Monitor web server logs for unusual patterns of form submissions or requests originating from third-party sites.
- **Alert:** Trigger an alert when suspicious patterns indicative of CSRF are detected.

2. Data Modification

Scenario: Force users to submit changes to their personal data or application settings.
Impact: Unauthorised modification of user data, leading to potential identity theft or data corruption.

Monitoring and Alerting:

- **Detection:** Monitor web server logs for patterns of data modification requests that deviate from normal user behaviour.
- **Alert:** Trigger an alert when unauthorised data modification attempts are detected.

3. Account Takeover

Scenario: Change the email address or password associated with a user account.

Impact: Loss of account control, leading to unauthorised access to sensitive data and services.

Monitoring and Alerting:

- **Detection:** Monitor web server logs for patterns of account credential changes, particularly those that are unexpected or unauthorised.
- **Alert:** Trigger an alert when suspicious account takeover attempts are detected.

4. Permission Changes

Scenario: Change the permissions or roles of a user within an application.

Impact: Unauthorised access to restricted areas or functionalities, leading to potential data breaches.

Monitoring and Alerting:

- **Detection:** Monitor logs for patterns of permission changes that deviate from normal administrative behaviour.
- **Alert:** Trigger an alert when unauthorised permission change attempts are detected.

1. Microsoft. "Application Security Scenarios (SQL Server)." Microsoft Learn, https://learn.microsoft.com/en-us/sql/connect/ado-net/sql/application-security-scenarios-sql-server?view=sql-server-ver16.
2. OWASP Foundation. "SQL Injection Prevention Cheat Sheet." OWASP, https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html.
3. Redgate. "SQL Server Vulnerabilities and Assessment." Redgate, https://www.red-gate.com/simple-talk/databases/sql-server/security/sql-server-vulnerabilities-and-assessment/.
4. N-able. "SQL Injection Examples." N-able, https://www.n-able.com/pt-br/blog/sql-injection-examples.
5. Imperva. "SQL Injection (SQLi)." Imperva, https://www.imperva.com/learn/application-security/sql-injection-sqli/.

6. OWASP Foundation. "SQL Injection." OWASP, https://owasp.org/www-community/attacks/SQL_Injection.

1. Authentication Bypass

Implementation Example:

- **Scenario:** A login form asks for a username and password.
- **Attack:** The attacker enters `admin' OR '1'='1` in the username field and anything (or nothing) in the password field.
- **Resulting Query:** `SELECT * FROM users WHERE username = 'admin' OR '1'='1' AND password = ''`
- **Effect:** The condition `OR '1'='1'` is always true, so the database returns the first user's record (often an admin).

Detection and Mitigation:

- **Input Validation:** Ensure inputs match expected formats (e.g., no SQL special characters). Validate and sanitize all user inputs to prevent the insertion of malicious code.

Parameterized Queries: Use SQL parameters to separate code from data, which prevents SQL injection attacks by ensuring that user input cannot alter SQL code. For example, in Python with SQLite:
python
Copy code
```
cursor.execute("SELECT * FROM users WHERE username = ? AND password = ?", (username, password))
```

-
- **Web Application Firewalls (WAF):** Deploy WAFs to detect and block common SQL injection patterns by filtering out malicious traffic before it reaches the application.

2. Data Extraction

Implementation Example:

- **Scenario:** A search form that displays product details.
- **Attack:** The attacker modifies the search input to `1' UNION SELECT username, password FROM users--`.
- **Resulting Query:** `SELECT * FROM products WHERE product_id = 1 UNION SELECT username, password FROM users--`
- **Effect:** The query returns both product details and user credentials.

Detection and Mitigation:

- **Least Privilege:** Restrict database user privileges to limit what data can be accessed by any particular user. Ensure that database users have only the minimum necessary privileges.
- **Error Handling:** Customize error messages to avoid revealing database details. Generic error messages should not give hints about the database structure.
- **Database Monitoring:** Use database activity monitoring tools to detect unusual query patterns. Tools like SQLmap can help simulate attacks to test the effectiveness of security measures.

3. Error-Based SQL Injection

Implementation Example:

- **Scenario:** A website displays product prices.
- **Attack:** The attacker inputs `1' AND 1=CONVERT(int, (SELECT @@version))--`.
- **Resulting Query:** `SELECT * FROM products WHERE product_id = 1 AND 1=CONVERT(int, (SELECT @@version))--`
- **Effect:** If the conversion fails, the error message may reveal the database version.

Detection and Mitigation:

- **Error Handling:** Ensure generic error messages without database details. Implement detailed error logging that is not accessible to end users but can be reviewed by administrators to identify potential attacks.
- **Input Validation:** Sanitize and validate all user inputs.

4. Second-Order SQL Injection

Implementation Example:

- **Scenario:** An application saves user inputs, like a username, for later use.
- **Attack:** The attacker enters `John'--`.
- **Result:** The username `John'--` is stored.
- **Later Query:** `SELECT * FROM users WHERE username = 'John'--'`
- **Effect:** The comment `--` causes the rest of the query to be ignored, potentially bypassing checks.

Detection and Mitigation:

- **Sanitization:** Clean inputs both at the time of storage and retrieval. Ensure that stored data cannot be interpreted as SQL commands when retrieved.

5. Command Execution

Implementation Example:

- **Scenario:** An application allows administrators to run database commands.
- **Attack:** The attacker injects `1; DROP TABLE users--`.
- **Resulting Query:** `SELECT * FROM products WHERE product_id = 1; DROP TABLE users--`
- **Effect:** The `DROP TABLE` command deletes the users table.

Detection and Mitigation:

- **Permissions:** Restrict the execution of commands to trusted users only. Implement role-based access control (RBAC) to ensure that only authorized users can execute sensitive commands.
- **Parameterized Queries:** Always use parameterized queries to prevent command injection.

6. Data Manipulation

Implementation Example:

- **Scenario:** An input form allows users to update their profiles.
- **Attack:** The attacker changes their input to `1; UPDATE users SET password='hacked' WHERE username='admin'--`.
- **Resulting Query:** `SELECT * FROM users WHERE user_id = 1; UPDATE users SET password='hacked' WHERE username='admin'--`
- **Effect:** The attacker changes the admin's password.

Detection and Mitigation:

- **Input Validation:** Validate and sanitize all inputs. Use white-listing to ensure that only expected input types and values are accepted.
- **Parameterized Queries:** Use parameterized queries to ensure that user input cannot alter SQL code.

7. Database Fingerprinting

Implementation Example:

- **Scenario:** An attacker wants to know the database type to tailor further attacks.
- **Attack:** The attacker inputs `1' AND 1=2 UNION SELECT NULL, @@version--`.
- **Resulting Query:** `SELECT * FROM products WHERE product_id = 1 AND 1=2 UNION SELECT NULL, @@version--`
- **Effect:** The query returns the database version.

Detection and Mitigation:

- **Input Validation:** Restrict input types and lengths to prevent SQL code from being executed.
- **Database Monitoring:** Monitor for queries that attempt to retrieve database metadata.

8. Privilege Escalation

Implementation Example:

- **Scenario:** An attacker aims to gain administrative privileges.
- **Attack:** The attacker injects `1; GRANT ALL ON *.* TO 'attacker'@'%'--`.
- **Resulting Query:** `SELECT * FROM users WHERE user_id = 1; GRANT ALL ON *.* TO 'attacker'@'%'--`
- **Effect:** The attacker gains full privileges.

Detection and Mitigation:

- **Permissions:** Use the principle of least privilege to restrict user access. Ensure that even if an account is compromised, it has minimal impact.
- **Role-Based Access Control:** Implement RBAC to control access to sensitive commands and data.

9. Enumeration of Database Schema

Implementation Example:

- **Scenario:** An attacker wants to understand the database structure.
- **Attack:** The attacker inputs `1 UNION SELECT table_name, column_name FROM information_schema.columns--`.
- **Resulting Query:** `SELECT * FROM products WHERE product_id = 1 UNION SELECT table_name, column_name FROM information_schema.columns--`
- **Effect:** The query reveals the names of all tables and columns.

Detection and Mitigation:

- **Input Validation:** Limit the scope of user inputs to prevent SQL code from being executed.
- **Database Monitoring:** Monitor for queries accessing information_schema or metadata tables.

1. Data Exfiltration

Implementation Example:

- **Scenario:** An attacker uses DNS queries to exfiltrate data from a network.
- **Attack:** The attacker encodes data into DNS queries. For example, sensitive data is divided into small chunks, base64-encoded, and included in subdomains of DNS requests. These requests are sent to an attacker-controlled DNS server.
- **Example Query:** `sensitive_data.example.com` where `sensitive_data` is the encoded data.
- **Result:** The attacker-controlled DNS server receives the queries, decodes the data, and reconstructs the stolen information.

Detection and Mitigation:

- **Monitor DNS Traffic:** Use tools like Splunk or ELK Stack to analyze DNS traffic patterns. Look for unusual volumes of DNS queries or queries to suspicious or uncommon domains.
- **DNS Filtering:** Implement DNS filtering to block known malicious domains and prevent communication with attacker-controlled servers.
- **Rate Limiting:** Apply rate limiting on DNS queries to detect and prevent data exfiltration attempts.

2. Command and Control (C2) Communication

Implementation Example:

- **Scenario:** An attacker establishes a C2 channel using DNS.
- **Attack:** The attacker sends commands to a compromised device through DNS queries. The infected device sends DNS requests containing encoded commands to an attacker-controlled domain.
- **Example Query:** `cmd.attackerserver.com` where `cmd` contains the encoded command.
- **Result:** The compromised device decodes the commands and executes them, then sends the results back using DNS queries.

Detection and Mitigation:

- **Anomaly Detection:** Monitor DNS traffic for patterns typical of C2 communication, such as regular intervals of unusual DNS queries.
- **DNS Security Solutions:** Deploy DNS security solutions that can detect and block suspicious DNS activity.
- **Behavioral Analysis:** Use behavioral analysis to detect deviations from normal DNS query patterns.

3. Malware Download

Implementation Example:

- **Scenario:** An attacker downloads malware onto a compromised device using DNS.
- **Attack:** The attacker encodes the malware payload into DNS responses. The compromised device sends DNS queries, and the attacker-controlled DNS server responds with encoded malware chunks.
- **Example Query:** `malware.attackerserver.com`
- **Result:** The device decodes the malware chunks from DNS responses and reassembles the malware for execution.

Detection and Mitigation:

- **DNS Inspection:** Inspect DNS responses for unusually large payloads or encoded data that could indicate malware transfer.

- **Endpoint Security:** Use endpoint security solutions to detect and block the execution of downloaded malware.

4. DNS Amplification Attack

Implementation Example:

- **Scenario:** An attacker uses DNS queries to amplify traffic towards a target.
- **Attack:** The attacker sends small DNS queries with spoofed source IP addresses, causing the DNS server to send large responses to the target IP.
- **Example Query:** `large_response_query.attackerserver.com`
- **Result:** The target IP is overwhelmed by the amplified traffic, resulting in a denial-of-service (DoS) condition.

Detection and Mitigation:

- **Rate Limiting:** Limit the rate of DNS responses to prevent amplification.
- **IP Blacklisting:** Block IPs known to be sources of DNS amplification attacks.
- **Response Size Limits:** Configure DNS servers to limit the size of responses.

5. Evasion of Network Controls

Implementation Example:

- **Scenario:** An attacker uses DNS tunneling to bypass network security controls.
- **Attack:** The attacker encodes traffic in DNS queries to evade detection. This allows the attacker to communicate with the compromised device without triggering security alerts.
- **Example Query:** `encoded_traffic.attackerserver.com`
- **Result:** The attacker can covertly communicate and transfer data without detection.

Detection and Mitigation:

- **DNS Anomaly Detection:** Monitor for DNS queries that deviate from normal patterns, such as those containing encoded data.
- **Network Segmentation:** Segment the network to limit the spread of malicious traffic and isolate compromised devices.
- **DNS Tunneling Detection Tools:** Use specialized tools designed to detect and block DNS tunneling activities.

Cross-Site Request Forgery (CSRF) Threat Scenarios Explained

1. Unauthorized Actions

Implementation Example:

- **Scenario:** An attacker tricks a user into executing unauthorized actions on a website where they are authenticated.

- **Attack:** The attacker sends the user a malicious link or form submission that, when clicked or submitted, triggers an action on a site where the user is already logged in. For example, changing the user's email address.
- **Example:** A user receives an email with a link to `http://victimsite.com/change_email?email=attacker@example.com`. When the user clicks the link while logged into `victimsite.com`, their email is changed to `attacker@example.com`.
- **Result:** The attacker gains control over the user's account settings without their consent.

Detection and Mitigation:

- **Anti-CSRF Tokens:** Implement unique tokens in forms that are verified on the server. Each request should include a token that the server verifies before processing the request.

Example in Code:
html
Copy code
```
<input type="hidden" name="csrf_token" value="{{csrf_token}}">
```
python
Copy code
```
if request.POST['csrf_token'] != session['csrf_token']:
    return "CSRF attack detected!"
```

-
- **SameSite Cookies:** Use the `SameSite` attribute to restrict cookies to same-site requests, preventing them from being sent along with cross-site requests.

Example in Code:
python
Copy code
```
response.set_cookie('sessionid', session_id, samesite='Strict')
```

-

2. Data Modification

Implementation Example:

- **Scenario:** An attacker forces a user to submit changes to their personal data or application settings.
- **Attack:** The attacker crafts a malicious form or URL that, when the user interacts with it, modifies data on the server. For instance, updating the user's address.
- **Example:** The attacker sends a link to `http://victimsite.com/update_address?address=attacker_address`. If the user clicks it while logged in, their address is changed.
- **Result:** The user's data is modified without their knowledge or consent.

Detection and Mitigation:

- **User Confirmation:** Require users to confirm changes through a secondary action such as entering a password or receiving a confirmation email.

Example in Code:
python
Copy code
```
if not confirm_password(input_password):
    return "Password confirmation required!"
```

-
- **Referer Header Check:** Verify that requests originate from trusted sources by checking the `Referer` header.

Example in Code:
python
Copy code
```
if request.headers['Referer'] != 'https://trustedsite.com':
    return "Invalid Referer!"
```

-

3. Account Takeover

Implementation Example:

- **Scenario:** An attacker changes the email address or password associated with a user account.
- **Attack:** The attacker tricks the user into submitting a request that changes their account credentials.
- **Example:** The user clicks on a link `http://victimsite.com/change_password?new_password=attacker_password` while logged into their account.
- **Result:** The attacker changes the user's password, gaining control over their account.

Detection and Mitigation:

- **Multi-Factor Authentication (MFA):** Require an additional verification step, such as a code sent via SMS, for critical actions like changing passwords.

Example in Code:
python
Copy code
```
if not verify_otp(user_input_otp):
    return "OTP verification required!"
```

-
- **Login Alerts:** Notify users via email or SMS when account settings are changed.

Example in Code:
python
Copy code
```
send_email(user_email, "Your password has been changed.")
```

-

4. Subscription Manipulation

Implementation Example:

- **Scenario:** An attacker alters the subscription settings of a user.
- **Attack:** The attacker tricks the user into sending a request that changes their subscription status.
- **Example:** The user clicks a link to `http://victimsite.com/change_subscription?status=canceled`.
- **Result:** The user's subscription is changed without their consent.

Detection and Mitigation:

- **User Confirmation:** Require users to confirm subscription changes through a secondary action, such as a confirmation email.

Example in Code:
python
Copy code
```
if not confirm_email_link(user_input_link):
    return "Email confirmation required!"
```

-
- **Referer Header Check:** Verify the origin of requests to ensure they come from trusted sources.

Example in Code:
python
Copy code
```
if request.headers['Referer'] != 'https://trustedsite.com':
    return "Invalid Referer!"
```

-

5. Permission Changes

Implementation Example:

- **Scenario:** An attacker changes the permissions or roles of a user within an application.
- **Attack:** The attacker tricks the user into submitting a request that changes their user role.
- **Example:** The user clicks a link `http://victimsite.com/change_role?role=admin` while logged in.
- **Result:** The user's role is changed, potentially giving them unauthorized access to sensitive functions.

Detection and Mitigation:

- **Anti-CSRF Tokens:** Implement unique tokens in forms that are verified on the server.

Example in Code:
html
Copy code
```
<input type="hidden" name="csrf_token" value="{{csrf_token}}">
```
python
Copy code
```
if request.POST['csrf_token'] != session['csrf_token']:
    return "CSRF attack detected!"
```

-
- **SameSite Cookies:** Use the `SameSite` attribute to restrict cookies to same-site requests.

Example in Code:
python
Copy code
```
response.set_cookie('sessionid', session_id, samesite='Strict')
```

-

Summary Table: Threats, Test Scenarios, and Monitoring Tools

Threat Type	Test Scenarios	Monitoring Tool
SQL Injection	- **Authentication Bypass:** Test by inputting SQL code in login forms.	**Splunk Enterprise**
	- **Data Extraction:** Test by using SQL injection to retrieve sensitive data.	
	- **Error-Based SQLi:** Trigger error messages to gather database info.	
	- **Second-Order SQLi:** Inject SQL code that gets stored and executed later.	
	- **Command Execution:** Inject commands to alter/delete records in the database.	
DNS Tunneling	- **Data Exfiltration:** Encode data in DNS queries to an attacker-controlled server.	
	- **Command and Control (C2):** Use DNS queries to communicate with compromised devices.	
	- **Malware Download:** Encode malware payloads in DNS responses.	
	- **DNS Amplification Attack:** Use DNS queries to amplify traffic and overwhelm a target.	
	- **Evasion of Network Controls:** Encode traffic in DNS queries to bypass security.	
CSRF (XSRF)	- **Unauthorized Actions:** Trick users into executing unintended actions.	
	- **Data Modification:** Force users to submit changes to their data.	
	- **Account Takeover:** Change user credentials without consent.	
	- **Subscription Manipulation:** Alter user subscription status.	

	- **Permission Changes:** Change user roles or permissions.	