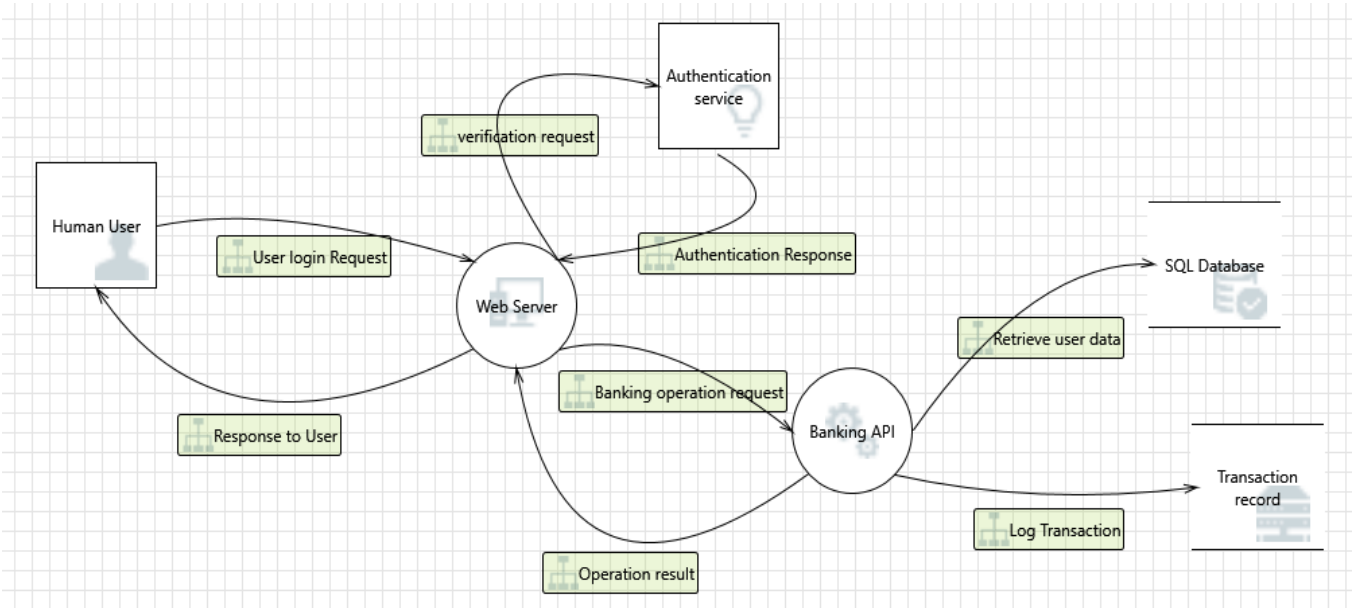
Threat Modeling Report

Created on 24/05/2025 2:06:29 PM

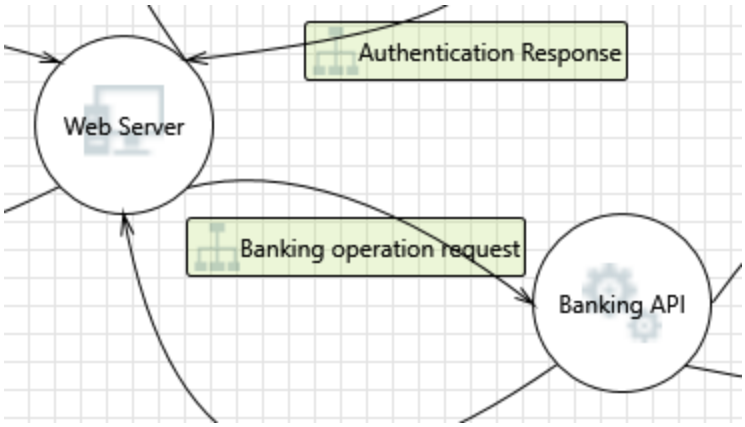
Threat Model Summary:

Not Started	23
Not Applicable	0
Needs Investigation	0
Mitigation Implemented	0
Total	23
Total Migrated	0

Diagram: Diagram 1



Interaction: Banking operation request



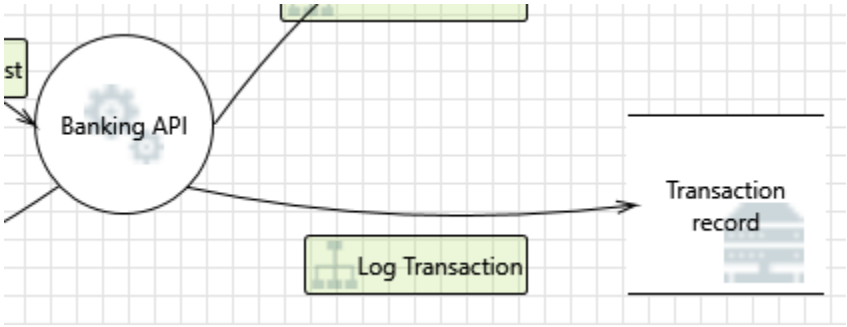
1. Elevation Using Impersonation [State: Not Started] [Priority: High]

Category: Elevation Of Privilege

Description: Banking API may be able to impersonate the context of Web Server in order to gain additional privilege.

Justification: To prevent the Banking API from impersonating the Web Server, we apply strict mutual authentication using service-level tokens or certificates and enforce least privilege access between service

Interaction: Log Transaction



2. Spoofing of Destination Data Store Transaction record [State: Not Started] [Priority: High]

Category: Spoofing

Description: Transaction record may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of Transaction record. Consider using a standard authentication mechanism to identify the destination data store.

Justification: We mitigate spoofing by implementing strong authentication mechanisms between the application and data store, applying network-level access controls, and monitoring for unusual destination access patterns.

3. Risks from Logging [State: Not Started] [Priority: High]

Category: Tampering

Description: Log readers can come under attack via log files. Consider ways to canonicalize data in all logs. Implement a single reader for the logs, if possible, in order to reduce attack surface area. Be sure to understand and document log file elements which come from untrusted sources.

Justification: We mitigate tampering in logs by sanitizing input before logging, using a secure single log reader, and reducing the attack surface with canonicalized log formats.

4. Lower Trusted Subject Updates Logs [State: Not Started] [Priority: High]

Category: Repudiation

Description: If you have trust levels, is anyone other outside of the highest trust level allowed to log? Letting everyone write to your logs can lead to repudiation problems. Only allow trusted code to log.

Justification: We restrict log writing permissions to only highly trusted components, ensuring only authorized and trusted code can write to the logs to avoid repudiation risks.

5. Data Logs from an Unknown Source [State: Not Started] [Priority: High]

Category: Repudiation

Description: Do you accept logs from unknown or weakly authenticated users or systems? Identify and authenticate the source of the logs before accepting them.

Justification: We identify and authenticate all log sources before accepting log data, preventing repudiation risks from unknown or unauthenticated systems.

6. Insufficient Auditing [State: Not Started] [Priority: High]

Category: Repudiation

Description: Does the log capture enough data to understand what happened in the past? Do your logs capture enough data to understand an incident after the fact? Is such capture lightweight enough to be left on all the time? Do you have enough data to deal with repudiation claims? Make sure you log sufficient and appropriate data to handle a repudiation claims. You might want to talk to an audit expert as well as a privacy expert about your choice of data.

Justification: We enhance auditing by ensuring all critical events are logged with sufficient detail, and that logs are designed for lightweight, continuous operation to support incident investigation and repudiation claims.

7. Potential Weak Protections for Audit Data [State: Not Started] [Priority: High]

Category: Repudiation

Description: Consider what happens when the audit mechanism comes under attack, including attempts to destroy the logs, or attack log analysis programs. Ensure access to the log is through a reference monitor, which controls read and write separately. Document what filters, if any, readers can rely on, or writers should expect

Justification: We protect audit data by ensuring all access goes through a reference monitor, using tamper-evident logging, and clearly documenting filters and access controls.

8. Authorization Bypass [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Can you access Transaction record and bypass the permissions for the object? For example by editing the files directly with a hex editor, or reaching it via filesharing? Ensure that your program is the only one that can access the data, and that all other subjects have to use your interface.

Justification: To prevent unauthorized access, we enforce that all interactions with transaction records go through validated interfaces, block direct file access, and apply strict database permissions.

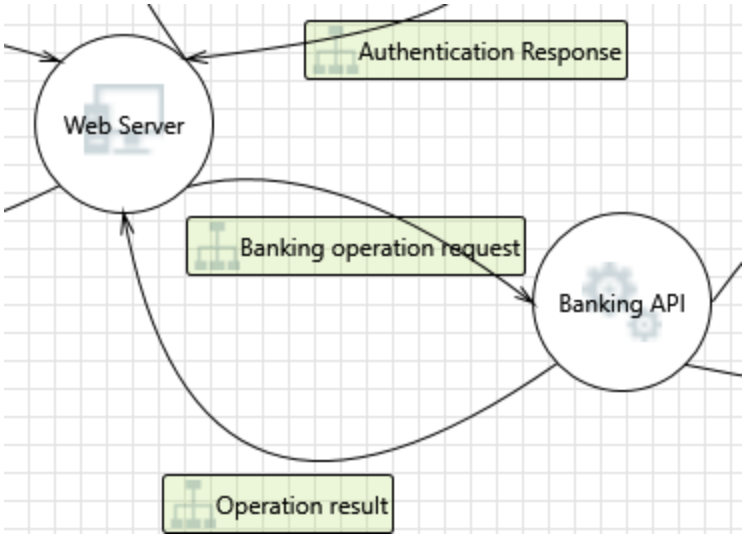
9. Potential Excessive Resource Consumption for Banking API or Transaction record [State: Not Started] [Priority: High]

Category: Denial Of Service

Description: Does Banking API or Transaction record take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.

Justification: We mitigate DoS risks by applying API rate limiting, timeouts, and resource quotas, and ensuring system-level protections are in place.

Interaction: Operation result



10. Replay Attacks [State: Not Started] [Priority: High]

Category: Tampering

Description: Packets or messages without sequence numbers or timestamps can be captured and replayed in a wide variety of ways. Implement or utilize an existing communication protocol that supports anti-replay techniques (investigate sequence numbers before timers) and strong integrity.

Justification: We prevent replay attacks by introducing nonces, timestamps, or sequence numbers in all sensitive communications and using secure communication protocols with anti-replay protections.

11. Collision Attacks [State: Not Started] [Priority: High]

Category: Tampering

Description: Attackers who can send a series of packets or messages may be able to overlap data. For example, packet 1 may be 100 bytes starting at offset 0. Packet 2 may be 100 bytes starting at offset 25. Packet 2 will overwrite 75 bytes of packet 1. Ensure you reassemble data before filtering it, and ensure you explicitly handle these sorts of cases.

Justification: We mitigate collision attacks by correctly reassembling packets before processing, validating all incoming data, and explicitly handling overlapping data scenarios.

12. Cross Site Scripting [State: Not Started] [Priority: High]

Category: Tampering

Description: The web server 'Web Server' could be a subject to a cross-site scripting attack because it does not sanitize untrusted input.

Justification: To prevent XSS, we apply rigorous input sanitization, escape all untrusted output, and implement Content Security Policy (CSP) headers to limit script execution sources.

13. Weak Authentication Scheme [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Custom authentication schemes are susceptible to common weaknesses such as weak credential change management, credential equivalence, easily guessable credentials, null credentials, downgrade authentication or a weak credential change management system. Consider the impact and potential mitigations for your custom authentication scheme.

Justification: We replace custom or weak authentication mechanisms with industry-standard protocols like OAuth 2.0, enforce MFA, and implement strong credential handling to reduce unauthorized access risks.

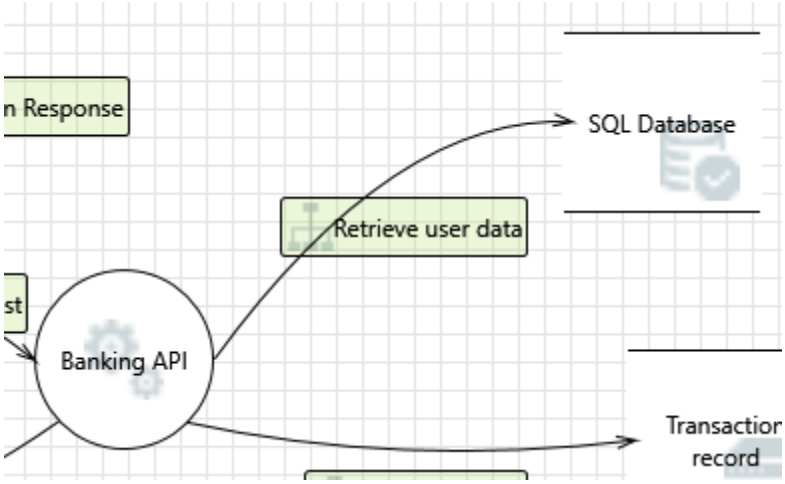
14. Elevation Using Impersonation [State: Not Started] [Priority: High]

Category: Elevation Of Privilege

Description: Web Server may be able to impersonate the context of Banking API in order to gain additional privilege.

Justification: We separate roles and identities clearly, apply strong service-to-service authentication, and enforce session binding to prevent privilege escalation between components.

Interaction: Retrieve user data



15. Spoofing of Destination Data Store SQL Database [State: Not Started] [Priority: High]

Category: Spoofing

Description: SQL Database may be spoofed by an attacker and this may lead to data being written to the attacker's target instead of SQL Database. Consider using a standard authentication mechanism to identify the destination data store.

Justification: We mitigate spoofing by using secure, authenticated connections to the SQL database, enforcing firewall and network access controls, and validating all database endpoints.

16. Potential SQL Injection Vulnerability for SQL Database [State: Not Started] [Priority: High]

Category: Tampering

Description: SQL injection is an attack in which malicious code is inserted into strings that are later passed to an instance of SQL Server for parsing and execution. Any procedure that constructs SQL statements should be reviewed for injection vulnerabilities because SQL Server will execute all syntactically valid queries that it receives. Even parameterized data can be manipulated by a skilled and determined attacker.

Justification: We prevent SQL injection by using parameterized queries or prepared statements, avoiding direct string concatenation in SQL, and validating all user inputs.

17. Authorization Bypass [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Can you access SQL Database and bypass the permissions for the object? For example by editing the files directly with a hex editor, or reaching it via filesharing? Ensure that your program is the only one that can access the data, and that all other subjects have to use your interface.

Justification: We eliminate bypass risks by blocking direct access to the database files and ensuring all access flows through controlled application interfaces with proper permissions.

18. Weak Credential Storage [State: Not Started] [Priority: High]

Category: Information Disclosure

Description: Credentials held at the server are often disclosed or tampered with and credentials stored on the client are often stolen. For server side, consider storing a salted hash of the credentials instead of storing the credentials themselves. If this is not possible due to business requirements, be sure to encrypt the credentials before storage, using an SDL-approved mechanism. For client side, if storing credentials is required, encrypt them and protect the data store in which they're stored

Justification: We secure credentials by using salted hashes (bcrypt, scrypt, Argon2) on the server side and applying encryption plus local store protections on the client side.

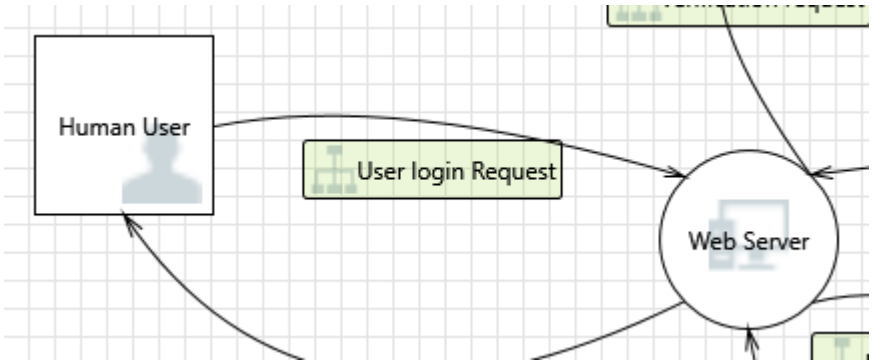
19. Potential Excessive Resource Consumption for Banking API or SQL Database [State: Not Started] [Priority: High]

Category: Denial Of Service

Description: Does Banking API or SQL Database take explicit steps to control resource consumption? Resource consumption attacks can be hard to deal with, and there are times that it makes sense to let the OS do the job. Be careful that your resource requests don't deadlock, and that they do timeout.

Justification: We mitigate resource exhaustion by applying query limits, connection pool caps, input validation, and OS-level resource constraints

Interaction: User login Request



20. Spoofing the Human User External Entity [State: Not Started] [Priority: High]

Category: Spoofing

Description: Human User may be spoofed by an attacker and this may lead to unauthorized access to Web Server. Consider using a standard authentication mechanism to identify the external entity.

Justification: We prevent user impersonation through strong authentication (passwords + MFA), CAPTCHA or bot detection, and monitoring of login patterns for anomalies.

21. Cross Site Scripting [State: Not Started] [Priority: High]

Category: Tampering

Description: The web server 'Web Server' could be a subject to a cross-site scripting attack because it does not sanitize untrusted input.

Justification: We prevent XSS through input sanitization, output escaping, and the application of CSP headers to control executable content.

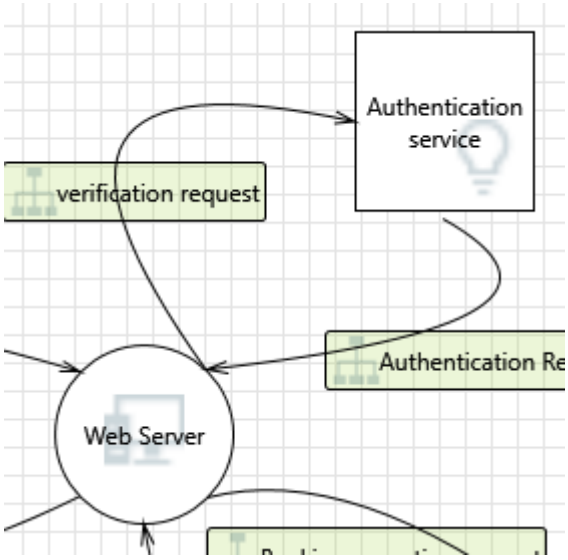
22. Elevation Using Impersonation [State: Not Started] [Priority: High]

Category: Elevation Of Privilege

Description: Web Server may be able to impersonate the context of Human User in order to gain additional privilege.

Justification: We mitigate impersonation risks by enforcing strict identity separation, binding sessions to device/IP, and using strong service authentication measures.

Interaction: verification request



23. Weakness in SSO Authorization [State: Not Started] [Priority: High]

Category: Elevation Of Privilege

Description: Common SSO implementations such as OAUTH2 and OAUTH Wrap are vulnerable to MitM attacks.

Justification: We strengthen SSO by implementing PKCE with OAuth2, enforcing TLS across all flows, validating all tokens, and using short-lived access tokens to reduce MitM risks.