Cybersecurity (CSCI 2413): Final Review

Student Companion Guide

December 11, 2025

1 Introduction to Cybersecurity Review

This companion guide summarizes the critical concepts reviewed for Cybersecurity, CSCI 2413, focusing on foundational knowledge, penetration testing methodologies, common web application vulnerabilities as defined by the OWASP Top Ten (2021), and defensive strategies.

2 Protocols and Ports

A fundamental requirement in networking and security is understanding which common protocols operate on which ports. This knowledge is essential for firewall configuration, intrusion detection, and performing reconnaissance during penetration testing.

Common Protocols and Ports

Protocol	**Port**
FTP (File Transfer Protocol)	20, 21
SSH (Secure Shell)	22
Telnet (Telecommunication Network)	23
SMTP (Simple Mail Transfer Protocol)	25
DNS (Domain Name System)	53
HTTP (Hypertext Transfer Protocol)	80
POP3 (Post Office Protocol version 3)	110
LDAP (Lightweight Directory Access Protocol)	389
HTTPS (HTTP Secure)	443
RDP (Remote Desktop Protocol)	3389

It is crucial to note that services running on low-numbered ports (1-1023) are often privileged and require administrative rights to bind to. Modern secure equivalents, like HTTPS (443) for HTTP (80), or SSH (22) for Telnet (23) and FTP (21, 20), should always be preferred to prevent eavesdropping and data tampering.

3 Types of Hackers and Penetration Testing

The cybersecurity industry categorizes individuals who attempt to bypass security measures into different "hats," reflecting their intent and legality.

3.1 Types of Hackers

- **White Hat Hacker (Ethical Hacker):** This individual is usually referred to as a penetration tester today. They operate with explicit permission from the asset owners to simulate attacks against a target system. Their goal is defensive: to find vulnerabilities before malicious actors do.

- **Black Hat Hacker (Cracker):** These are malicious actors who gain unauthorized access to systems with the intent to cause harm, disruption, or financial gain. They act illegally and without permission.
- **Gray Hat Hacker:** A complex category, the gray hat hacker typically acts as a law-abiding citizen but may sometimes venture into unauthorized activities, often believing their actions are justified, such as exposing vulnerabilities publicly without prior consent from the owner.

3.2 Penetration Testing (PenTester)

A Penetration Tester (PenTester) is an ethical hacker who is formally **AUTHORIZED** to simulate a cyber-attack against a computer or server environment. Pen Testers are typically employed by specialized security firms that perform structured attacks on behalf of a client. Their deliverables include a comprehensive report detailing any vulnerabilities found and concrete recommendations on how to remediate those issues.

3.2.1 Penetration Testing Stages (Conceptual Model)

The penetration testing process is usually broken down into several distinct phases to ensure thoroughness and repeatability. Intuitively, this involves preparation, discovery, exploitation, sustained access, and final reporting/configuration validation.

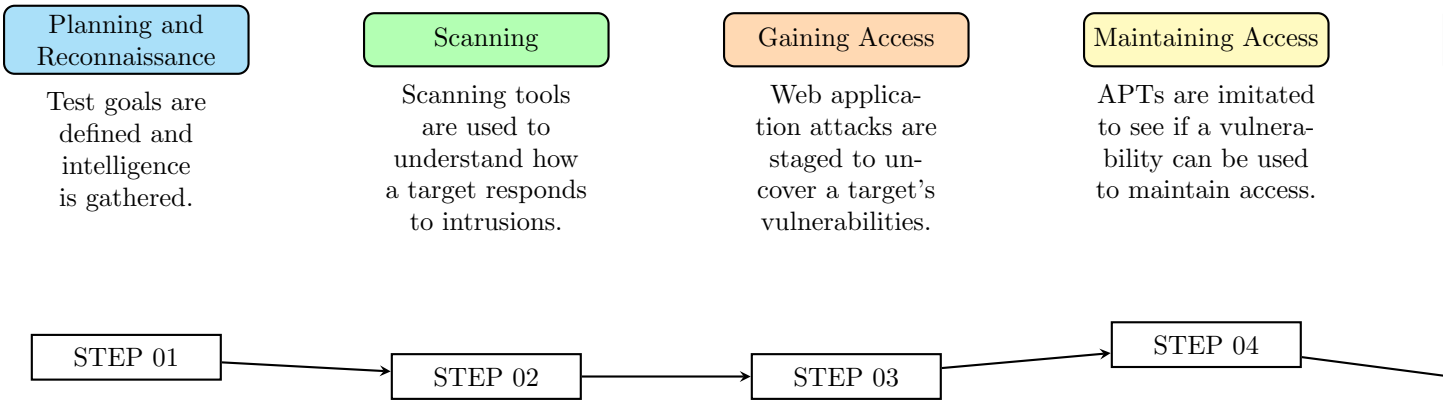

Figure 1: Conceptual Penetration Testing Workflow

3.2.2 Penetration Testing Stages (Technical Model)

A more detailed, technical perspective on the stages of a penetration test reveals an iterative process focusing heavily on discovery and escalation.

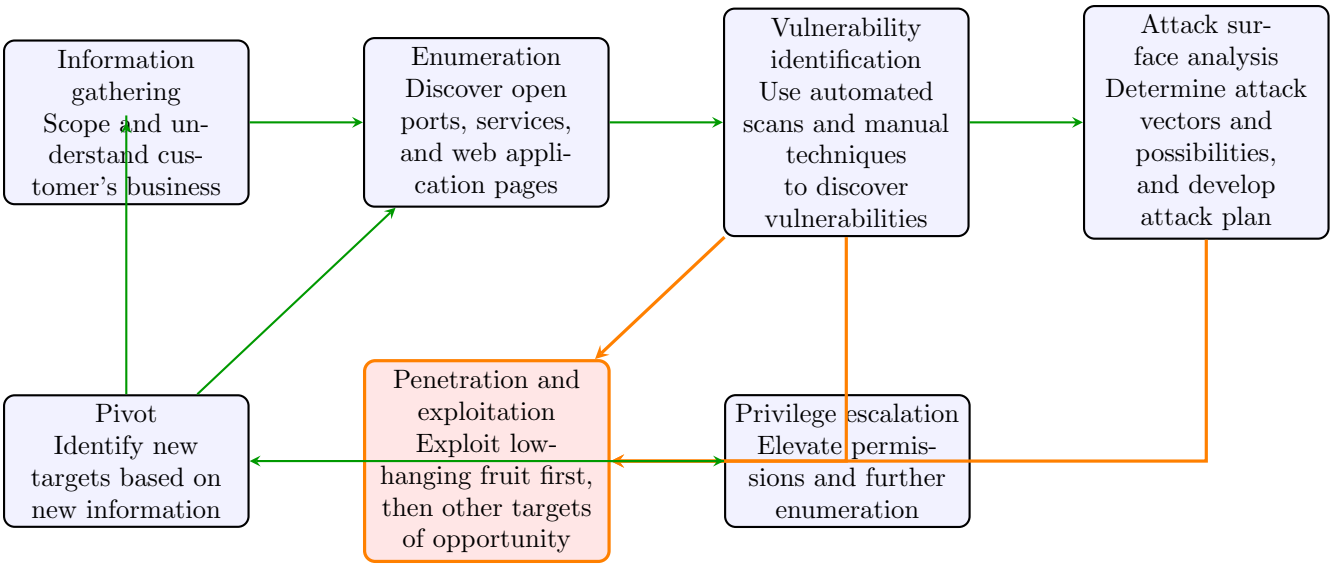

Figure 2: Technical Penetration Testing Stages

4 The Open Web Application Security Project (OWASP)

OWASP is a crucial non-profit foundation dedicated to improving the security of software. They achieve this by providing:

1. Tools and resources for testing and development.
2. A global community platform for networking and collaboration.
3. Education and training materials on secure coding practices.

The OWASP Top Ten is perhaps their most famous publication, outlining the ten most critical web application security risks. This list is released periodically (typically every 3–4 years), with the current version being from 2021.

When determining the criticality and ranking of vulnerabilities, OWASP assesses several criteria: **Exploitability**, **Prevalence**, **Detectability**, and **Technical impact**.

OWASP Top Ten Comparison (2017 vs. 2021)

The 2021 list saw significant reshuffling and the introduction of new categories, reflecting changes in development practices (e.g., component usage, design focus) and attack trends (e.g., SSRF).

2017 Rank and Vulnerability	2021 Rank and Vulnerability
A01:2017-Injection	A01:2021-Broken Access Control (Moved from A05)
A02:2017-Broken Authentication	A02:2021-Cryptographic Failures (Previously A03: Sensitive Data Exposure)
A03:2017-Sensitive Data Exposure	A03:2021-Injection (Moved from A01)
A04:2017-XML External Entities (XXE)	**A04:2021-Insecure Design (New)**
A05:2017-Broken Access Control	A05:2021-Security Misconfiguration (Moved from A06)
A06:2017-Security Misconfiguration	A06:2021-Vulnerable and Outdated Components (Moved from A09)
A07:2017-Cross-Site Scripting (XSS)	A07:2021-Identification and Authentication Failures (Moved from A02)
A08:2017-Insecure Deserialization	**A08:2021-Software and Data Integrity Failures (New)**
A09:2017-Using Components with Known Vulnerabilities	A09:2021-Security Logging and Monitoring Failures (Moved from A10)
A10:2017-Insufficient Logging & Monitoring	**A10:2021-Server-Side Request Forgery (SSRF) (New/Re-entry)**

5 A01: Broken Access Control (2021)

Access control refers to the policies implemented by an application to ensure that users are restricted to acting within their defined and intended permissions. Failures in this area are severe, often leading to unauthorized information disclosure, modification, or outright destruction of data, or allowing a user to perform business functions far outside their security scope.

5.1 Vulnerabilities Associated with Access Control

Broken access control can manifest in numerous ways:

- **Violation of Least Privilege:** Failing to adhere to the principle of least privilege, or not enforcing a "deny by default" approach.
- **Parameter Tampering (Bypassing Checks):** Modifying parameters in the URL (e.g., changing an item ID or user ID) to bypass intended access control checks. This allows users to view or edit someone else's account data.
- **Missing Access Controls in APIs:** Accessing API endpoints (particularly those using methods like `POST`, `PUT`, or `DELETE`) that lack sufficient authorization checks.
- **Privilege Elevation:** A user acting as a highly privileged user without having properly authenticated or having the requisite permissions.
- **Metadata Manipulation:** Tampering with session information or authentication tokens, such as replaying or modifying a JSON Web Token (JWT), to gain unwarranted access.
- **Force Browsing:** Navigating directly to authenticated or privileged pages as an unauthenticated or standard user by simply guessing the URL structure.

5.2 Web HTTP Methods and Status Codes

When discussing web applications and access control, we must understand the core ways a client interacts with a server using HTTP methods (verbs) and how the server responds (status codes).

HTTP Methods

These verbs dictate the intended action on a resource:

- **GET:** Retrieves data from the server.
- **PUT:** Sends updated data to the server (used for modifying an existing resource).
- **POST:** Sends new or created data to the server (used for creating a new resource).
- **DELETE:** Removes data from the server.

HTTP Status Codes

These codes are returned by the server to indicate the outcome of the request:

- **20X (Success):** Indicates successful processing.
  - 200 – OK (Standard success)
  - 201 – Created (Resource creation successful)
  - 204 – No Content (Action successful, but no content returned)
- **30X (Redirection):** Indicates the client needs to take further action.
  - 301 – Moved Permanently
  - 304 – Not Modified
- **40X (Client Error):** Indicates the client made a bad request or lacks authorization.
  - 400 – Bad Request (Server cannot process due to client error)
  - **401 – Unauthorized** (Client must authenticate)
  - **403 – Forbidden** (Client is authenticated but lacks necessary permissions)
  - 404 – Not Found
- **50X (Server Error):** Indicates the server failed to fulfill the request.
  - 500 – Internal Server Error
  - 503 – Service Unavailable

5.3 Prevention: Broken Access Control

To prevent broken access control vulnerabilities, developers should:

- **Deny by Default:** Always assume a user lacks permission unless explicitly granted access.
- **Implement CORS Safely:** Use Cross-Origin Resource Sharing (CORS) only when necessary and configure origins securely to prevent unauthorized domain access.
- **Restrict File Access:** Disable web server directory listing to prevent attackers from browsing sensitive files.
- **Log and Review Failures:** Log all access control failures and review these logs periodically, treating them as potential hack attempts.

- **Validate Permissions Server-Side:** Always ensure the correct user is making the change and that they possess the necessary permissions, regardless of client-side validation.
- **Use Secure Tokens:** Use tokens like JSON Web Tokens (JWT) (`https://jwt.io`) for authentication, but ensure the token data is integrity-protected (signed) and that permissions encoded within it are respected.
- **Avoid Guessable Identifiers:** Make IDs harder to guess, preferably using non-numeric Globally Unique Identifiers (GUIDs) (`https://guidgenerator.com/`) instead of simple sequential integers.

6 A02: Cryptographic Failures (2021)

Previously referred to as A03: Sensitive Data Exposure (2017), this category focuses on protecting data in transit and at rest. This protection is vital for various types of sensitive data:

- Passwords
- Credit Cards (covered under PCI standards)
- Health Records (covered under HIPAA/HIPPA equivalent regulations)
- Personal Information (PII)

6.1 Assessing Vulnerability and Data Classification

To assess whether an application is vulnerable, we must ask critical questions about cryptographic implementation:

- Is any sensitive data transmitted in clear text (unencrypted)?
- Is the cryptography version deprecated or known to be weak (e.g., SHA1, MD5)?
- Are default crypto keys in use, and how were they generated? (Default keys are a huge risk; poor entropy generation leads to weak keys.)
- Is the certificate chain validated, and has the certificate expired?
- Are the same keys being used across multiple, separate servers? (This violates the principle of key segregation.)

Before protecting data, organizations must first classify it to determine its required level of security.

Classification Criteria

- Is the data a company secret?
- Are there specific legal requirements governing the data (e.g., PII, Health Data)?
- Is it highly sensitive authentication data (Password, PIN code, Security Question Answer)?

Classification and Storage Method

Once sensitivity is determined, the storage and transmission method must align with the risk:

- **Passwords, PIN codes:** MUST be Hashed (one-way function, irreversible).
- **User personal information, User health information, Company Private Info:** Must be Encrypted (reversible, but protected).
- **Company Public info, User Public info:** Can often be stored as Plain Text.

It is important to remember that hashing is suitable for comparison and verification (like passwords), while encryption is required if the data must eventually be retrieved in its original form (like health records or credit card numbers).

6.2 Secure Transmission and Storage

Data Transmission Protocols

Non-secure protocols must be avoided in favor of their secure counterparts:

- **HTTP (Web)** is not secure. All websites must use **HTTPS** (Secure Web), which leverages TLS/SSL encryption.
- **FTP (File Transfer)** is not secure. All servers should use **FTPS** (Secure File Transfer Protocol), often utilizing SSL/TLS for encryption.
- **SMTP (Mail)** is inherently insecure. While there isn't one universal secure mail protocol, modern mail solutions (like Gmail) support encrypting and decrypting mail, often via STARTTLS or specialized add-ons utilizing Public Key Infrastructure (PKI).

Secure Email (PKI)

Secure email relies on Public Key Encryption. Both the sender and receiver possess a public key (used to encrypt data for the other party) and a private key (used to decrypt data intended for them). Implementing this successfully requires coordination between the parties to exchange public keys.

Credit Card Data (PCI Compliance)

The Payment Card Industry (PCI) sets strict standards for storing and handling credit cards. Companies are highly discouraged from storing raw credit card data. The recommended security practice for systems requiring recurring payments is to use a secure token (an encrypted, non-sensitive value) provided by a credit card clearing company, ensuring the application itself never holds the sensitive primary account number (PAN).

6.3 Prevention: Cryptographic Failures

Preventing cryptographic failure involves establishing policies around classification and usage:

- Classify data processed, stored, or transmitted into defined sensitivity levels.
- Ensure classification considers all applicable laws and regulations.
- Do not store sensitive data if it is avoidable.
- Encrypt all sensitive data at rest using strong, industry-standard cryptography.
- Use cryptographic standards that are current and up to date.
- **Mandate TLS 1.2 or above** for all data in transit. TLS 1.3 is strongly preferred.
- Disable caching of sensitive data in web browsers or on servers.

7 A03: Injection Attacks (2021)

Injection flaws occur when untrusted data is sent to an interpreter (like a database, command shell, or LDAP server) without proper validation or sanitization. This allows attackers to execute unintended commands or access data without authorization.

While **SQL Injection** is the most commonly discussed type, injection attacks include:

- SQL (Structured Query Language)

- NoSQL (Non-Relational Databases)
- Command Injection (OS commands)
- LDAP (Lightweight Directory Access Protocol)

7.1 Injection Vulnerability Checklist

An application is vulnerable to injection if:

- User-supplied data is **NOT validated** for structure, content, or type.
- Dynamic queries are built using string concatenation rather than being **parameterized** (precompiled statements).
- Hostile data (input containing malicious code) is directly used or concatenated into the final command/query string.

7.2 Relational Databases and SQL Basics

Relational databases structure data efficiently into Tables, which contain columns (fields) and rows (records), similar to a spreadsheet. These tables are managed using Structured Query Language (SQL).

SQL CRUD Keywords

SQL operations map directly to the foundational Create, Read, Update, Delete (CRUD) actions:
- **CREATE** = `INSERT INTO`
- **READ** = `SELECT`
- **UPDATE** = `UPDATE`
- **DELETE** = `DELETE`

READ - SELECT

The basic syntax retrieves columns from a table:

```
SELECT [columns] FROM [table]
```

The columns list can be one or multiple column names separated by commas, or "*" to mean all columns.

Filtering with WHERE

The `WHERE` clause filters the resulting rows based on specific column values:

```
SELECT [columns] FROM [table] WHERE [column] = [value]
```

The `LIKE` operator can be used for pattern matching, such as selecting all names that start with 'J': `WHERE Name LIKE 'J%'`.

CREATE - INSERT INTO

This statement adds new data rows to a table. Note the strict requirement for matching column and value lists.

```
INSERT INTO [table] ([columns]) VALUES ([values])
```

UPDATE - UPDATE

Used to modify existing data. Need to be extremely cautious: without a `WHERE` clause, **all** records in the table will be changed.

```
UPDATE [table] SET [column]=[value] WHERE [column]=[value]
```

DELETE - DELETE

Used to remove data. Need to be **EXTRA** careful: without a `WHERE` clause, **all** records in the table will be deleted.

```
DELETE FROM [table] WHERE [column]=[value]
```

7.3 The Mechanics of SQL Injection

SQL Injection works by exploiting applications that construct queries using unprotected user input concatenated into a string. The core vulnerability involves manipulating string concatenation to inject operational SQL commands.

Consider a dynamic query where an attacker substitutes the password input with: `jmaxwell' or '1'='1' --`

This input modifies the structure of the intended query:

```
SELECT * FROM Users WHERE username='jmaxwell' OR '1'='1' -- AND password='xxx'
```

The single quote (') prematurely closes the password value string. The `OR '1'='1'` condition is then injected, which always evaluates to true. Finally, the double hyphen (`--`) comments out the remainder of the original query structure, forcing the database to return all relevant records without requiring the correct password.

7.4 Prevention: Injection

The industry-standard solution is to use **parameterized queries** (prepared statements). These enforce a strict separation between the SQL command structure and the user-supplied data, ensuring the input is always treated as data, never as executable code.

8 A04: Insecure Design (New in 2021)

Insecure Design is a new, broad category focusing on flaws related to missing or ineffective control design. This vulnerability emphasizes that security must be an integral part of the software development lifecycle (SDLC), not an afterthought. Secure Design involves a fundamental change in organizational thinking, culture, and methodology toward building software and hardware systems securely from the initial design phase.

8.1 Threat Modeling

Threat Modeling is the practice of systematically reviewing application diagrams, data flows, and architecture to proactively identify and mitigate potential security issues before code is written. Key questions during this process focus on:

- How does the data flow?
- Is Encryption being used at appropriate checkpoints?
- Who has access to WHAT? (Establishing authorization policies).

8.1.1 Diagram 1: Application Data Flow (Schematic)

This diagram illustrates key components and trust boundaries in a web application environment, such as the DMZ and LAN, helping visualize potential points of failure.

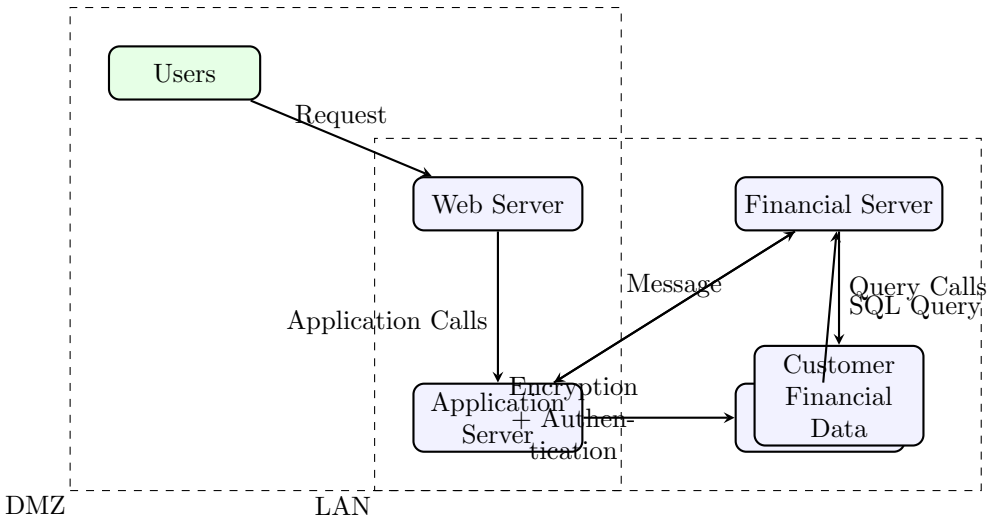

Figure 3: Schematic Application Trust Boundaries and Data Flow

8.1.2 Diagram 2: Banking Transaction Flow (Simplified DFD)

This diagram illustrates a process flow, distinguishing between Authentication (AuthN) and Authorization (AuthZ), and showing how data interacts with policy checks.

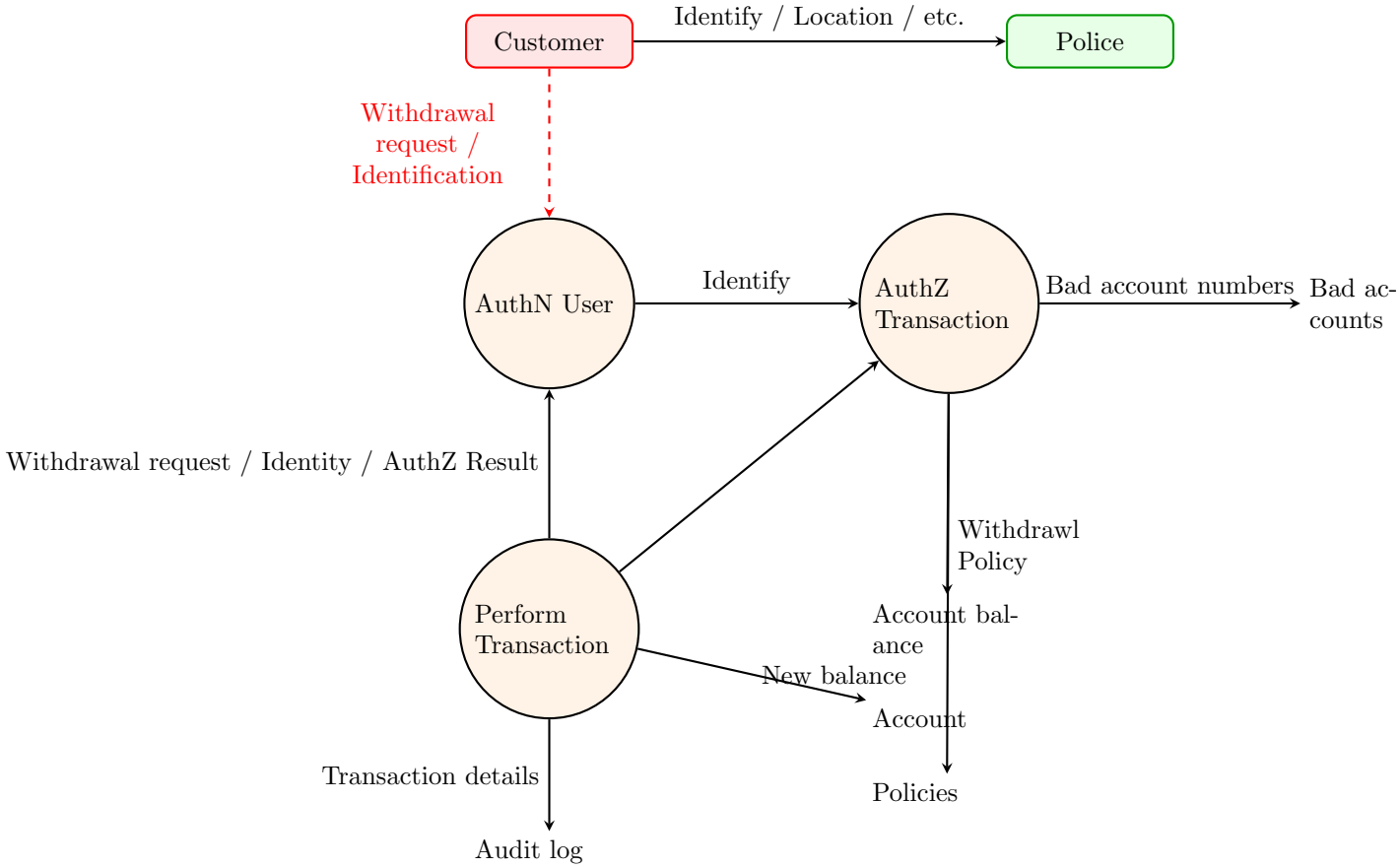

Figure 4: Conceptual Threat Model: Banking Transaction Flow

8.2 Security Assessment and Prevention

The OWASP Software Assurance Maturity Model (SAMM) (`https://owaspsamm.org`) provides a framework for evaluating and improving the security practices of an organization.

To prevent insecure design, security must be integrated:

- AppSec professionals must be involved in the design phase.
- Build a library of Secure Design patterns and reference code.
- Use Threat Modeling for critical systems to diagram authentication, access control, and authorization patterns.
- Document all Data Flows between systems.
- Create User Stories/Requirements to test against known attacks (SQL Injection, XSS, CSRF).
- Have PenTesters validate the design before deployment.

9 A05: Security Misconfiguration (2021)

Security Misconfiguration deals with systems and software that are improperly set up, leading to unnecessary exposure. This includes exploiting unpatched flaws, accessing unprotected files, and leveraging default configurations that grant excessive access.

9.1 Prevention: Security Misconfiguration

- Work with administrators to "Harden" the server ("Lock it down").
- Remove default accounts and credentials.
- Remove any unused services and turn off unnecessary ports.
- Add generalized error handling that hides system information (no stack traces).
- Take a least privilege approach to every component.
- Create a secure baseline (Gold Image) version of each OS.
- Utilize automated Security Testing and tools.
- Log Server and Application errors, and, critically, **review them regularly**.

10 A06: Vulnerable and Outdated Components (2021)

This risk category addresses vulnerabilities introduced by using outdated or flawed third-party components and libraries (3^{rd} party components). Since modern development relies heavily on these components, their security posture directly affects the application's overall security.

10.1 Vulnerability Management Tools

- National Vulnerability Database (NVD): `https://nvd.nist.gov`
- OWASP Dependency Checker: `https://owasp.org/www-project-dependency-check/`
- Snyk.io: `https://snyk.io`

For Node.js projects, the Node Package Manager (NPM) offers:

```
npm ls [package name]
npm audit [fix]
npm update
```

10.2 Prevention: Vulnerable Components

- **Continuous Patching:** Integrate patching as a normal part of the development cycle (CI/CD pipelines).
- Use scanners in build/deployment pipelines to automatically check dependencies.
- Only use components from official sites/sources.
- Use a local, secured repository for development teams to pull vetted updates.
- Maintain scheduled releases for updates (e.g., quarterly).
- Do not allow frameworks to fall more than one or two major versions behind.

11 A07: Identification and Authentication Failures (2021)

This category covers weaknesses in managing user identities and confirming they are who they claim to be. Vulnerabilities often involve weak passwords, poor credential management, or insufficient session handling.

11.1 Key Attack Vectors

- **Credential Stuffing:** Using large lists of previously leaked credentials (stolen from other sites, like those listed on `https://haveibeenpwned.com/`) to attempt logins.
- **Brute Force Attacks:** Trying large volumes of simple or commonly used passwords against accounts.
- **Weak Passwords:** Allowing default or easily guessable passwords (e.g., "Password1").
- **Weak Forgot-Password Processes:** Revealing user existence (enumeration) or emailing cleartext passwords.

11.2 Password Security Protocols

- Plain text and encrypted passwords are fundamentally BAD because encrypted data can be decrypted.
- You must **ALWAYS** Hash passwords with a **RANDOM Salt** value unique for each user.

11.3 Prevention: Broken Authentication

- Implement Complex Password Requirements (length $>$ 15 characters, special characters, mixed case).
- Implement Account Lockout after a small number of failed attempts.
- HASH passwords with a random SALT using a slow, modern algorithm (e.g., Argon2, bcrypt).
- Implement Multi-Factor Authentication (MFA) (e.g., Email, Text, QR code, RSA Token).
- Never email a user's password.
- Prevent user enumeration by giving a generic response to password reset requests (e.g., "If an account exists, instructions were emailed").

12 A08: Software and Data Integrity Failures (2021)

This category addresses the risks associated with code and infrastructure that fail to protect against integrity violations, particularly during updates and deployment.

12.1 Prevention: Data Integrity Failures

- Use Digital Signatures to validate who created software updates.
- Manage internal package repositories for 3rd party controls.
- **Integrity Check:** After deployment, calculate and compare HASHes (checksums) on files/folders to detect tampering.
- Manual Code Reviews of all critical changes.
- Harden CI/CD pipelines to restrict access and avoid storing credentials within them.

13 A09: Security Monitoring and Logging Failures (2021)

This item highlights that insufficient logging, monitoring, and active alerting prevents timely detection of security breaches. Logging must capture all traffic and activity, and logs must be secured away from the monitored system.

13.1 Logging Best Practices

- Log normal traffic (successful paths) and all errors/exceptions.
- Use a centralized tool (like Splunk) to manage and review logs.
- **NEVER Log PII:** Avoid logging sensitive data like PII, Credit Cards, Tokens, or Passwords.
- **Monitoring is Essential:** Logging is **WORTHLESS** unless logs are reviewed daily for errors.
- Implement alerting systems for certain thresholds or abnormal traffic.
- Ensure high-value transactions have detailed, tamper-proof audit trails (e.g., append-only database tables).

14 A10: Server-Side Request Forgery (SSRF) (2021)

SSRF re-entered the Top 10 due to increased exploitation potential, particularly in cloud environments. SSRF occurs when a web application fetches a remote resource (or internal resource) without validating the user-supplied URL.

14.1 Vulnerability and Prevention

- **Vulnerability:** SSRF allows attackers to force the server to connect to internal services or access sites the user shouldn't reach. The Panera Bread Hack (accessing user data by changing a sequential ID) is often a form of SSRF when an API endpoint processes the ID without validation.
- **Prevention:** Need to rigorously validate and sanitize user input that may contain URLs or resource identifiers. Implement strong allow-lists (whitelists) and deny-lists (blacklists) against internal IP ranges.