L04: Ready, Set, Stop That Threat!

Course: Cybersecurity Threat Modeling Lab
Lab: L04 - Threat Modeling Analysis
Student: [Your Name]
Date: January 2025 (America/Los_Angeles)

Task 1: Software Selection

Project Name: Polr

Description: Polr is an open-source, minimalist URL shortener that allows users to host their own URL shortening service. Built with PHP and utilizing either MySQL or SQLite databases, Polr offers a sleek interface and robust API for easy integration. Typically deployed on LAMP or LEMP stacks, it provides features like custom short URLs, detailed analytics, and a user-friendly dashboard for managing shortened links. The application serves both individual users and organizations looking to create branded short URLs while maintaining full control over their link shortening infrastructure.

Official Repository: https://github.com/cydrobolt/polr

Task 2: Data Flow Diagram (DFD)

DFD Rationale

- **Trust Boundaries**: Internet-to-Edge and App-to-DB boundaries clearly separate external users from internal systems
- **Data Flows**: HTTPS POST for URL submission and SQL queries for database operations represent critical data movement

- **Components**: External entities (users), processes (reverse proxy, app server), and data stores (database) model the complete system
- **Security Focus**: Trust boundaries highlight where authentication and authorization controls must be implemented
- **Compliance**: Follows OWASP threat modeling guidelines for identifying potential attack vectors

DFD Diagram

```mermaid graph TD %% External Entities User[("User Browser (External Entity)")]

```
%% Trust Boundary 1: Internet to Edge
subgraph TB1 ["🌐 Internet to Edge Trust Boundary"]
    ReverseProxy["Reverse Proxy
(Process)"]
end

%% Trust Boundary 2: App to DB
subgraph TB2 ["🏢 App to Database Trust Boundary"]
    AppServer["Polr Application Server
(Process)"]
    Database[("URL Database
(Data Store)")]
end

%% Data Flows
User -->|"1. HTTPS POST: URL + Auth Token
(Data Flow)"| ReverseProx
ReverseProxy -->|"2. HTTP: Forwarded Request
(Data Flow)"| AppServe
AppServer -->|"3. SQL Query: Check Existing URL
(Data Flow)"| Datab
Database -->|"4. SQL Response: URL Data
(Data Flow)"| AppServer
AppServer -->|"5. HTTP: Shortened URL Response
(Data Flow)"| Revers
ReverseProxy -->|"6. HTTPS: Final Response
(Data Flow)"| User

%% Additional Data Flows for URL Access
User -->|"7. HTTPS GET: Access Short URL
(Data Flow)"| ReverseProxy
ReverseProxy -->|"8. HTTP: Forward Access Request
(Data Flow)"| App
AppServer -->|"9. SQL Query: Retrieve Original URL
(Data Flow)"| Da
Database -->|"10. SQL Response: Original URL
(Data Flow)"| AppServe
AppServer -->|"11. HTTP 302: Redirect Response
(Data Flow)"| Revers
```
```

```
ReverseProxy -->|"12. HTTPS: Final Redirect<br/>(Data Flow)"| User

%% Styling
classDef external fill:#e1f5fe
classDef process fill:#f3e5f5
classDef datastore fill:#e8f5e8
classDef trustboundary fill:#fff3e0,stroke:#ff9800,stroke-width:3px,str

class User external
class ReverseProxy,AppServer process
class Database datastore
class TB1,TB2 trustboundary
```

```

---

# Task 3: Threat Tree Diagram (TTD)

## TTD Rationale

- **Root Threat**: Compromise of user data or account integrity captures the primary security concern
- **Vulnerabilities**: Three distinct attack vectors covering authentication, input validation, and authorization weaknesses
- **Countermeasures**: Each vulnerability has specific, actionable security controls mapped to mitigate the risk
- **Attack Paths**: Clear logical progression from high-level threat to specific technical vulnerabilities
- **OWASP Alignment**: Follows OWASP Top 10 and threat modeling best practices for web applications

## TTD Diagram

```mermaid graph TD %% Root Threat Root["🔥 Compromise of User Data
or Account Integrity
(Root Threat)"]

```
%% Level 1 - Attack Categories
AuthWeak["🔐 Weak Authentication
(Vulnerability)"]
InputVal["⚠️ Input Validation Issues
(Vulnerability)"]
AuthzWeak["🛡️ Authorization Weaknesses
(Vulnerability)"]

%% Level 2 - Specific Vulnerabilities
BruteForce["💥 Brute Force Attacks
(Specific Vulnerability)"]
SessionFix["🔗 Session Fixation
(Specific Vulnerability)"]
XSS["🌐 Cross-Site Scripting (XSS)
(Specific Vulnerability)"]
SQLi["🗡️ SQL Injection
(Specific Vulnerability)"]
IDOR["🎯 Insecure Direct Object References
(Specific Vulnerability)"
PrivilegeEsc["⬆️ Privilege Escalation
(Specific Vulnerability)"]

%% Countermeasures
RateLimit["🚦 Rate Limiting
(Countermeasure)"]
MFA["🔒 Multi-Factor Authentication
(Countermeasure)"]
SecureSession["🔐 Secure Session Management
(Countermeasure)"]
InputSanit["🧹 Input Sanitization
(Countermeasure)"]
ParamQueries["📝 Parameterized Queries
(Countermeasure)"]
AuthChecks["✅ Authorization Checks
(Countermeasure)"]

%% Connections - Root to Categories
Root --> AuthWeak
Root --> InputVal
Root --> AuthzWeak

%% Connections - Categories to Specific Vulnerabilities
AuthWeak --> BruteForce
AuthWeak --> SessionFix
InputVal --> XSS
InputVal --> SQLi
AuthzWeak --> IDOR
AuthzWeak --> PrivilegeEsc

%% Countermeasures mapped to vulnerabilities
RateLimit -.->|"Mitigates"| BruteForce
MFA -.->|"Mitigates"| BruteForce
```

```
SecureSession -.->|"Mitigates"| SessionFix
InputSanit -.->|"Mitigates"| XSS
ParamQueries -.->|"Mitigates"| SQLi
AuthChecks -.->|"Mitigates"| IDOR
AuthChecks -.->|"Mitigates"| PrivilegeEsc

%% Styling
classDef root fill:#ffebee,stroke:#f44336,stroke-width:4px
classDef vulnerability fill:#fff3e0,stroke:#ff9800,stroke-width:2px
classDef specific fill:#f3e5f5,stroke:#9c27b0,stroke-width:2px
classDef countermeasure fill:#e8f5e8,stroke:#4caf50,stroke-width:2px

class Root root
class AuthWeak,InputVal,AuthzWeak vulnerability
class BruteForce,SessionFix,XSS,SQLi,IDOR,PrivilegeEsc specific
class RateLimit,MFA,SecureSession,InputSanit,ParamQueries,AuthChecks co
```

```

---

# Task 4: Use & Misuse Diagram (UMD)

## UMD Rationale

- **Use Cases**: Three core legitimate user actions covering authentication, content creation, and account management
- **Misuse Cases**: Three corresponding attack scenarios that threaten each legitimate use case
- **Threaten Connections**: Clear mapping showing how misuse cases undermine legitimate functionality
- **Mitigation Controls**: Specific security measures that counteract each misuse case
- **Actor Modeling**: Four distinct actors representing different user roles and threat sources
```

UMD Diagram

```mermaid graph TB %% Actors RegisteredUser[("👤 Registered User (Actor")] AnonymousUser[("🌐 Anonymous Visitor (Actor")] Attacker[("⚠️ Malicious Attacker (Actor")] Admin[("👑 System Administrator (Actor")]

```
%% Use Cases
SignIn["🔐 Sign In to Account
(Use Case)"]
CreateShortURL["✂️ Create Short URL
(Use Case)"]
ManageAccount["⚙️ Manage Account Settings
(Use Case)"]
ViewAnalytics["📊 View URL Analytics
(Use Case)"]
AdminPanel["🛠️ Access Admin Panel
(Use Case)"]

%% Misuse Cases
CredentialStuffing["💀 Credential Stuffing Attack
(Misuse Case)"]
MaliciousURL["☠️ Shorten Malicious URLs
(Misuse Case)"]
DataHarvesting["🕷️ Harvest User Data
(Misuse Case)"]
AdminPrivilege["👑 Unauthorized Admin Access
(Misuse Case)"]

%% Mitigation Controls
MFA["🔒 Multi-Factor Authentication
(Mitigation)"]
URLValidation["✅ URL Validation & Filtering
(Mitigation)"]
AccessControl["🛡️ Strict Access Controls
(Mitigation)"]
RBAC["🎭 Role-Based Access Control
(Mitigation)"]

%% Legitimate Actor Connections
RegisteredUser --> SignIn
RegisteredUser --> CreateShortURL
RegisteredUser --> ManageAccount
RegisteredUser --> ViewAnalytics
AnonymousUser --> CreateShortURL
Admin --> AdminPanel

%% Threaten Connections (Red dashed lines)
CredentialStuffing -.->|"🚫 THREATENS"| SignIn
```
```

```
MaliciousURL -.->|"🚫 THREATENS"| CreateShortURL
DataHarvesting -.->|"🚫 THREATENS"| ViewAnalytics
AdminPrivilege -.->|"🚫 THREATENS"| AdminPanel

%% Attacker to Misuse Cases
Attacker --> CredentialStuffing
Attacker --> MaliciousURL
Attacker --> DataHarvesting
Attacker --> AdminPrivilege

%% Mitigation Connections (Green solid lines)
MFA -->|"✅ MITIGATES"| CredentialStuffing
URLValidation -->|"✅ MITIGATES"| MaliciousURL
AccessControl -->|"✅ MITIGATES"| DataHarvesting
RBAC -->|"✅ MITIGATES"| AdminPrivilege

%% Legend
subgraph Legend ["📋 Legend"]
    Threaten["🚫 THREATENS<br/>(Dashed Red Line)"]
    Mitigates["✅ MITIGATES<br/>(Solid Green Line)"]
    Normal["➡️ NORMAL FLOW<br/>(Solid Blue Line)"]
end

%% Styling
classDef actor fill:#e3f2fd,stroke:#2196f3,stroke-width:2px
classDef usecase fill:#e8f5e8,stroke:#4caf50,stroke-width:2px
classDef misuse fill:#ffebee,stroke:#f44336,stroke-width:2px
classDef mitigation fill:#fff3e0,stroke:#ff9800,stroke-width:2px
classDef legend fill:#f5f5f5,stroke:#9e9e9e,stroke-width:1px

class RegisteredUser,AnonymousUser,Attacker,Admin actor
class SignIn,CreateShortURL,ManageAccount,ViewAnalytics,AdminPanel usec
class CredentialStuffing,MaliciousURL,DataHarvesting,AdminPrivilege mis
class MFA,URLValidation,AccessControl,RBAC mitigation
class Legend legend
```

References

- OWASP Threat Modeling Guide: https://owasp.org/www-community/Threat_Modeling
- OWASP Threat Modeling Cheat Sheet: https://cheatsheetseries.owasp.org/cheatsheets/Threat_Modeling_Cheat_Sheet.html
- OWASP Top 10 Web Application Security Risks: https://owasp.org/www-project-top-ten/

End of Submission