Critical Threats in Modern Systems
Step 2 - Conceptual Model & ER Diagram

Jair Ramirez

CS 4332 Introduction to Database Systems

Texas State University

April 6th, 2025

Abstract

This step develops the conceptual schema for our “Critical Threats in Modern Systems” vulnerability database. Building on the normalized tables from Step 1, we identify core entities (e.g. VulnerabilityCategory, VulnerabilityInstance, Product) and their relationships, and draw a crow’s-foot Entity-Relationship diagram to enforce normalization and key constraints. We then define ten non-trivial English queries to illustrate the kinds of reports and lookups our database must support. These queries will guide the SQL implementation in the next phase.

Section A - Normalized Entities and ER Diagram

In this section we translate our logical tables into a conceptual model and ER diagram using crow’s-foot notation—resolving many-to-many relationships, merging one-to-one where appropriate, and explicitly marking primary/foreign keys.

1. VulnerabilityCategories

CategoryID	CategoryName	Description
1	Cross-Site Scripting	Vulnerabilities allowing XSS attacks
2	SQL Injection	Vulnerabilities allowing SQL query manipulation
3	Buffer Overflow	Vulnerabilities causing memory errors
4	Remote Code Execution	Vulnerabilities that enable remote code execution
5	Information Disclosure	Vulnerabilities exposing sensitive data

Figure 1: VulnerabilityCategories Table

Structure:

- **CategoryID** (Primary Key)
- **CategoryName**
- **Description**

2. VulnerabilityInstances

VulnerabilityInstanceID	CVE_ID	CategoryID	CVSS_Score	Severity
101	CVE-2025-2061	1	CVSS 5.3	Medium
102	CVE-2025-2699	1	CVSS 5.1	Medium
103	CVE-2025-2088	2	CVSS 6.9	Medium
104	CVE-2025-2951	2	CVSS 6.3	Medium
105	CVE-2025-2074	2	CVSS 5.3	Medium
106	CVE-2025-1899	3	CVSS 7.1	High
107	CVE-2024-13903	3	CVSS 7.5	High
108	CVE-2024-12035	4	CVSS 8.8	High
109	CVE-2024-50310	5	CVSS 8.7	High

Figure 2: VulnerabilityInstances Table (1)

Description
This vulnerability affects unknown code of the file /passenger.php. The manipulation of the argument name leads to cross site scripting.
Affected by this issue is some unknown functionality of the component Image Handler. The manipulation of the argument onload leads to cross site scripting.
Affected is an unknown function of the file /admin/profile.php. The manipulation of the argument fullname/emailid/mobileNumber leads to sql injection.
Affected is an unknown function of the file /api/data.php. The manipulation of the argument Search leads to sql injection.
Due to insufficient escaping on the user supplied parameter and lack of sufficient preparation on the existing SQL query, "sSearch". This makes it possible for authenticated attackers to append additional SQL queries.
Affected by this vulnerability is an unknown functionality of the file /goform/setPptpUserList. The manipulation of the argument list leads to buffer overflow.
Affected by this vulnerability is the function JS_GetRuntime of the file quickjs.c of the component qjs. The manipulation leads to stack-based buffer overflow.
The CS Framework plugin for WordPress is vulnerable to arbitrary file deletion due to insufficient file path validation in the cs_widget_file_delete() function.
Affected devices do not properly handle authorization.

Figure 3: VulnerabilityInstances Table (2)

Impact

Figure 4: VulnerabilityInstances Table (3)

Mitigation	DiscoveryDate
Implement proper input validation and output encoding on the affected file (/passenger.php). Sanitize all user-supplied data to ensure special characters are handled safely, and apply secure coding practices to prevent script...	2025-03-06
Apply strict input validation and output encoding for the onload attribute in the Image Handler component. Additionally, review and patch the affected component in GetmeUK ContentTools.	2025-03-24
Implement robust input validation and sanitization for the parameters (fullname, emailid, mobileNumber). Use parameterized queries or prepared statements to prevent SQL injection attacks.	2025-03-13
Implement strict input validation and sanitization for the "Search" parameter in the /api/data.php file. Use parameterized queries or prepared statements to prevent SQL injection attacks, and restrict database privileges to th...	2025-03-30
Implement robust input sanitization and parameterized queries to securely handle the "sSearch" parameter. Ensure that any user-supplied data is properly escaped and validated before use in SQL queries.	2025-03-28
Implement strict bounds checking and input validation on the "list" parameter in the /goform/setPptpUserList functionality. Ensure secure memory handling.	2025-03-03
Review and enforce secure coding practices, such as proper input validation and bounds checking, to prevent similar vulnerabilities.	2025-03-21
Enforce strict file path validation and restrict the file deletion functionality to only the intended and authorized directories. Additionally, ensure that file deletion operations are limited to users with the appropriate privileges.	2025-03-07
Enforce network segmentation and restrict remote access to critical systems to mitigate the risk of unauthorized access.	2024-11-12

Figure 5: VulnerabilityInstances Table (4)

Structure:

- **VulnerabilityInstanceID** (Primary Key)
- **CVE_ID**
- **CategoryID** (Foreign Key referencing VulnerabilityCategories)
- **CVSS_Score**
- **Severity**
- **Description**
- **Impact**
- **Mitigation**
- **DiscoveryDate**

3. Products

ProductID	ProductName	Version	Vendor
1	Online Ticket Reservation System	1.0	Code-projects
2	ContentTools	<= 1.6.16	GetmeUK
3	Pre-School Enrollment System	<= 1.0	PHPGurukul
4	Micro Mall	1.0	Bluestar
5	reCAPTCHA plugin for WordPress	1.29	Google
6	TX3	16.03.13.11_multi	Tenda
7	QuickJS	<= 0.8.0	quickjs-ng
8	CS Framework for WordPress	<= 6.9	CS Framework
9	SIMATIC CP 1543-1	4.0	Siemens

Figure 6: Products Table

Structure:

- **ProductID** (Primary Key)
- **ProductName**
- **Version**
- **Vendor**

4. Product_Vulnerability

VulnerabilityInstanceID	ProductID
101	1
102	2
103	3
104	4
105	5
106	6
107	7
108	8
109	9

Figure 7: Product_Vulnerability Table

Structure:

- **VulnerabilityInstanceID** (Foreign Key referencing VulnerabilityInstances)
- **ProductID** (Foreign Key referencing Products)
- **Composite Primary Key:** (VulnerabilityInstanceID, ProductID)

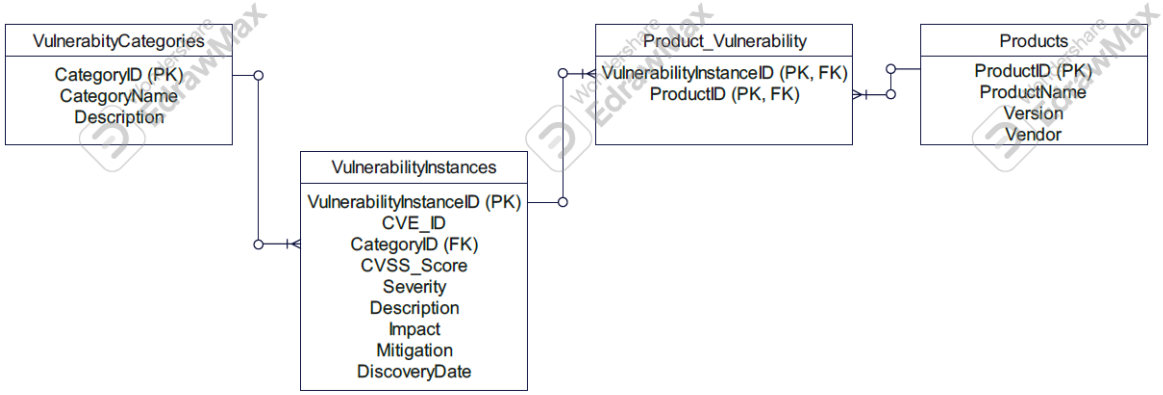

Figure 8: E/R Diagram

Section B - 10 English Queries

Here we list ten representative, user-focused English queries (e.g. “List all high-severity vulnerabilities by discovery date”) that will become our SQL reports in Step 3.

1. ***Query 1:*** Retrieve a list of all vulnerability instances, showing each CVE ID, the corresponding vulnerability type, severity, and discovery date, sorted by discovery date in descending order (most recent first).

2. ***Query 2:*** Display a list of products including the vendor and version, along with the number of vulnerabilities affecting each product, sorted by the vulnerability count in descending order.

3. ***Query 3:*** Retrieve details for all vulnerabilities of type 'SQL Injection'—including the CVE ID, CVSS score, severity, impact, and mitigation details—and list the names of all products affected by each of these vulnerabilities.

4. ***Query 4:*** Retrieve all vulnerabilities that were discovered in March 2025, displaying their CVE IDs, vulnerability types, severity levels, and the names of the products affected.

5. ***Query 5:*** Retrieve all vulnerabilities where the mitigation strategy includes the phrase 'input validation'. For each matching record, display the CVE ID, vulnerability type, detailed mitigation steps, and the names of all products affected by these vulnerabilities.

6. ***Query 6:*** Retrieve all products affected by 'Buffer Overflow' vulnerabilities. For each product, display the product name, vendor, and the count of 'Buffer Overflow' vulnerabilities associated with it, ordered by the vulnerability count in descending order.

7. ***Query 7:*** Retrieve aggregated vulnerability data: for each vulnerability type, display the total number of vulnerabilities and the average CVSS score, sorted by the total count in descending order.

8. ***Query 8:*** Retrieve all vulnerabilities with a severity level of 'High' and display their CVE IDs, vulnerability types, and discovery dates, along with the names of the products affected by them. Sort the results by CVE ID.

9. ***Query 9:*** Retrieve a list of all vulnerabilities along with their CVE IDs, vulnerability types, detailed descriptions, impact, and corresponding mitigation strategies, sorted by the CVE ID.

10. ***Query 10:*** Retrieve a distinct list of mitigation strategies used for vulnerabilities, along with the count of vulnerabilities that use each mitigation strategy, sorted by the count in descending order.