A.I.-enhanced Cyber Resiliency Evaluation System (ACRES) Database

Kristian Alleyne, Richard Flores, Claire Kamobaya, Matthew Penn

Dr. Xiang Liu

IT 310 A - Database Technology

December 10, 2024

TABLE OF CONTENTS

I. BUSINESS PROFILE

1. Introduction and Business Summary

CyberGuard Solutions is an innovative, small start-up focused on creating a cybersecurity assessment tool for IT teams. Our tool helps organizations identify cybersecurity gaps, provides tailored recommendations for remediation, and tracks security data for detailed analysis and reporting. By empowering teams to make informed, proactive decisions, we aim to enhance their overall security posture. Our core team includes Python developers who build and optimize the tool, a SCRUM master guiding agile development, technical writers ensuring clear documentation, researchers tracking cybersecurity trends, and a database architect managing data storage and reporting functions. Together, we bring a streamlined, accessible cybersecurity solution to a growing market.

2. Database Purpose and Function

The database is built to help IT teams assess cybersecurity by keeping track of essential information about infrastructure, software, and potential threats. Its functions include tracking definitions of critical node functions, mapping these functions to specific infrastructure nodes, and assigning qualitative names to quantitative criticality scores. Additionally, the database organizes infrastructure rack categories and network nodes, logs software names and versions, and maps software to corresponding infrastructure nodes. It tracks detected vulnerabilities, mapping each to the affected software versions, and maintains a list of potential APT (Advanced Persistent Threat) groups by ID and name. An AI-based threat scoring feature links APT groups to vulnerabilities, providing reasoning for each threat level. Each of these functions is represented by a dedicated table, ensuring a structured and efficient system for cybersecurity analysis.

3. Anticipated Users

The anticipated end users of this database are Cybersecurity Analysts, Network Engineers, and IT Staff. Cybersecurity Analysts will use the system to identify, assess, and respond to vulnerabilities within the infrastructure. Network Engineers will benefit from the ability to map critical functions to infrastructure nodes and manage network assets efficiently. IT Staff will use the database to monitor software versions, track infrastructure categories, and oversee overall network health. By centralizing essential cybersecurity and infrastructure data, the system allows each user group to quickly access relevant information, making it easier to collaborate on addressing security gaps and improving system resilience.

4. Project Inspiration and Origin

This project idea originated from a group of students at Marymount University who participated in the CRAM challenge, a cybersecurity competition hosted by the U.S. Navy. During Phase III of the challenge, we explored the idea of transforming our data into a structured database to enhance our cybersecurity tool's functionality. Initially, we tracked critical information using JSON files, including definitions and levels of critical node functions (ranked 1-3), mappings of functions to network nodes, infrastructure lists, installed software names and versions, and detected vulnerabilities. Vulnerabilities were identified through network scans, manual checks of vendor announcements, and NIST CPE database queries. Our current approach loads JSON files into our Python tool as lists of dictionaries, but shifting to a database would provide more efficient data management by allowing direct querying rather than iterating through all records. We propose designing a database structure for future iterations of our tool and populating it with data from one of the systems evaluated during the challenge. This structured database could also serve as a demonstration dataset for technical sales teams when presenting the tool to potential clients, showcasing its capabilities in a more streamlined and professional way.

II. ENTITY RELATIONSHIP DESIGN

1. Entity Descriptions
    a. APT_GROUPS:
        i. This entity stores information about each Advanced Persistent Threat (APT) groups such as their names, any aliases they go by, and a brief description of them.
            1. Apt_group - Primary Key, The unique name for apt groups
            2. Alias_names - Any alternate names the APT group goes by. For example APT 3 is also known as Gothic Panda.
            3. Description - detailed description of the APT group and their tactics, techniques, and procedures (TTP's)
    b. CRITICALITY_DEFINITIONS
        i. This entity stores the criticality values for the three different criticalities, or how important, different systems are.
            1. Criticality_values - Primary Key, Unique values that describe the criticality of systems.
            2. Criticality_name - the name of the criticality level.
            3. Downtime_allowed - the acceptable amount of downtime that can be permitted for each criticality level.
    c. VULNERABILITIES_DATA
        i. This entity contains all of the information about a particular CVE. It is heavy in attributes, but the majority of these attributes are used in our final calculation for a system score.
            1. Cve_number - primary key, unique identifier, common vulnerabilities and exposure (CVE). There can not be repeated CVE numbers.

2. Nvd_score - the score of the vulnerability from the NVD database. This describes how critical the vulnerability is.
3. Cvss_version - this is the version of the CVSS standard used for each vulnerability. This is relevant because different versions will have different stored information.
4. Vector_string - this is the attack vector details from CVSS, it shows all of the metrics in one long string.
5. Attack_vector - Part of the CVSS, and our scoring algorithm. This shows the level of access required. Four options, Network, Adjacent, Local, and Physical
6. Attack_complexity - Part of the CVSS, and our scoring algorithm. This measures the difficulty of the exploit. Two levels, Low or High.
7. Privilege_required - Part of the CVSS, and our scoring algorithm. Level of privilege required for the exploitation to be carried out. Three options, None, Low, High.
8. User_interaction_required - Part of the CVSS, and our scoring algorithm. This measures if a user is required for this exploitation, for example if a user must click a link. Two options, None or Required
9. Scope_changed -Part of the CVSS, and our scoring algorithm. This measures if the exploitation will affect systems outside of the initial exploitation environment. Two options
10. Impact_confidentiality - Part of the CVSS, and our scoring algorithm. This measures if confidentiality, or only allowing authorized users access to data, has been impacted. Three metrics, High impact, Low impact, or No impact

11. Impact_integrity - Part of the CVSS, and our scoring algorithm. This measures if the integrity, or unauthorized modification of system data has been impacted by and exploitation. Three metrics, High impact, Low impact, or No impact.
12. Impact_avaliability - Part of the CVSS, and our scoring algorithm. This measures if the availability, or access to data or systems has been impacted. Three metrics for this attribute, High impact, Low impact, or No impact.
13. Base_score - Part of the CVSS, and our scoring algorithm. This is the base score of the CVE. We use this score as a baseline for our algorithm to calculate cyber resiliency.
14. Base_severity - Part of the CVSS, and our scoring algorithm. This is based on the base score. If the score is between 0.1-3.0 it is LOW, 4.0-6.9 MEDIUM, 7.0-8.9 HIGH, 9.0-10 CRITICAL. This severity level can be used to show differences in systems when for example, all CRITICAL vulnerabilities are patched.
15. Exploitability_score - Part of the CVSS, and our scoring algorithm. The overall score for how exploitable this particular vulnerability is.
16. Impact_score - Part of the CVSS, and our scoring algorithm. The overall score for what this particular vulnerability will impact.
17. Description - Gives a brief description about what each particular CVE is, what is being exploited and how it is being exploited.

d. INFRASTRUCTRURE_CATEGROIES
    i. This entity categorizes and links the categories of the workstations to the employee endpoints in use.

1. Category_id - Primary Key, the unique identifier for the different types of categories
2. Category_name - what the categories have been named by the company.

e. HARDWARE_MAPPING
   i. This is an associative entity that maps, or connects, the hardware being used on each category and subsequently each endpoint.
      1. Hardware_id - Primary Key and Foreign Key. This attribute originates in the HARDWARE table and is used as a part of a unique composite primary key.
      2. Serial_number - Primary Key. This is the serial number on each piece of hardware. The reason we used a composite key is that the hardware will have the same serial number, but it will be unique with the hardware_id included.
      3. Category_id - Foreign Key. Shows the relationship between the INFRASTRUCTURE_CATEGORIES table and this table.
      4. Endpoint_id - Foreign key. Originated in the ENDPOINT_NODES table, is used here to link the category as well as the hardware running on the endpoint.

f. ENDPOINT_NODES
   i. This table represents the endpoints, or what the actual employees are working on.
      1. Endpoint_id - Primary Key. This is the unique identifier for each endpoint.
      2. Endpoint_name - Name of the endpoints.

g. SOFTWARE_FIRMWARE

    i. This table tracks the software and firmware installed on each and every node.
        1. Software_id - Primary Key. Unique identifier for the software installed. This is auto-generated.
        2. Software_make - the vendor or the creator of the software. For example, Cisco.
        3. Software_name - The title for the specific software. For example the Catalyst 2960-X, which is a Cisco network switch.
        4. Software_version - Represents the version or the release of the software/firmware. For example, IOS 15.2(1)E.

h. SYSTEM_SCORING
    i. This table tracks and stores all scores of system security against potential threats posed by a given APT group. Tracks metrics that CVSS can not such as personnel scores.
        1. Apt_group - Primary Key, Foreign Key. This attribute references the specific APT group under assessment. Originated in the APT_GROUPS entity.
        2. Score_name - Primary Key. This attribute defines the specific area of assessment. This categorizes scores for things such as personnel, physical security, and policies.
        3. Score - the actual numeric value which indicates the systems vulnerability or potential readiness in the specified score name.
        4. Reasoning - This gives a reasoning behind the given score so it is not arbitrary.

i. FUNCTION_DEFINITION

    i. This entity will describe organizational functions that link to the criticality values in the CRITICALITY_DEFINITION table.
        1. Function_number - Primary Key. This is an auto-generated number for each function. This ensures that each unique function can be referenced in other tables.
        2. Function_name - This is a clear and descriptive name for the function. For example, data input/processing. This helps define the function's purpose and goals.
        3. Work_area - This attribute specifies the department or work area that is responsible for the function.
        4. Criticality_value - Foreign Key. This is the attribute that links to important criticality values in another table.

j. FUNCTION_MAPPING
    i. This entity maps the endpoints to the functions that they are running on. This allows for the identification of critical systems required.
        1. Endpoint_id - Primary Key, Foreign Key. This references the specific endpoint performing the given function. Originates in the ENDPOINT_NODES entity.
        2. Function_number - Primary Key, Foreign Key. This attribute references the function being performed by the endpoint. This links to the FUNCTION_DEFINITION entity.

k. SOFTWARE_FIRMWARE_MAPPING
    i. This entity associates the endpoints that the employees will be using and their installed software and firmware.

1. Endpoint_id - Primary Key, Foreign Key. This attribute references the endpoint that is hosting the software. Originates in the ENDPOINT_NODES entity.
2. Software_id - Primary Key, Foreign Key. This attribute references the software that is installed on each endpoint. This attribute originates from the SOFTWARE_FIRMWARE entity.

l. VULNERABILITY_INSTANCES

i. This entity records instances where each particular CVE affects a specific software or firmware.

1. Cve_number - Primary Key, Foreign Key. This attribute is part of a composite primary key, and this is the unique CVE number. Originates in the VULNERABILITIES_DATA table.
2. Software_id - Primary Key, Foreign Key. This attribute is another part of a composite primary key, and is the unique software ID defined in the SOFTWARE_FIRMWARE table.

m. APT_CVE_SCORING

i. This entity evaluates the likelihood and severity of exploitation of specific vulnerabilities by a given APT group.

1. Cve_number - Primary Key, Foreign Key. This attribute is part of a composite primary key, and this is the unique CVE number. Originates in the VULNERABILITIES_DATA table.
2. Apt_group - Primary Key, Foreign Key. This attribute references the specific APT group under assessment and how they would impact a given vulnerability. Originated in the APT_GROUPS entity.

3. Score - This attribute is the numerical representation of the given APT groups likelihood to exploit the vulnerability under assessment.
4. Reasoning - This describes the logic and reasoning behind the score attribute so it is not just an arbitrary value.

2. Database Purpose and Function
    a. APT_GROUPS and SYSTEM_SCORING
        i. This is a 1:M relationship. Each APT group can have multiple scores across the 3 P's (physical, personnel, and policies), however each assessment relates to one singular APT group
    b. APT_GROUPS and APT_CVE_SCORING
        i. This is a 1:M relationship. Each singular APT group will have a score for each vulnerability. This means that each group will have a unique score per CVE, making this a 1:M relationship. For example, since we are only testing for one APT group against a system at a time this will be relevant for one group at a time.
    c. APT_CVE_SCORING and VULNERABILITIES_DATA
        i. This is a mandatory many to mandatory one (1:M) relationship. The CVE connects the tables and for each CVE there is one entry in the vulnerabilities data table.
    d. VULNERABILITIES_DATA and VULNERABILITY_INSTANCES
        i. This is a mandatory 1 to optional many relationship (1:M). This is because for each vulnerability (CVE) there can be multiple instances of it across the entire system since there is reuse of software and hardware.
    e. VULNERABILITY_INSTANCES and SOFTWARE_FIRMWARE

   i. This is a mandatory one to optional many relationship (1:M). The reason for this is that each software or firmware could be impacted by many different vulnerabilities, but each vulnerability and software pair is unique.

f. SOFTWARE_FIRMWARE and SOFTWARE_FIRMWARE_MAPPING
   i. This is a mandatory one to mandatory many relationship (1:M). This relationship maps or connects the software and the firmware to the endpoint nodes by creating an associative entity. So each software or firmware can be installed on multiple endpoints, but the software and firmware running on an endpoint is unique.

g. SOFTWARE_FIRMWARE_MAPPING and ENDPOINT_NODES
   i. This is a mandatory one to optional many relationship (1:M). Similar to the previous relationship this serves as mapping between endpoints and the software on the endpoints. Each singular endpoint can have many different software running on it.

h. ENDPOINT_NODES and FUNCTION_MAPPING
   i. This is a mandatory one to optional many relationship. This is an associative relationship where each endpoint is being mapped to the functions that the endpoint would fall under. Each endpoint can fall into multiple functions.

i. ENPOINT_NODES and HARDWARE_MAPPING
   i. This is a mandatory one to optional many relationship. This serves as a link between endpoints and the hardware that is running on a particular endpoint. This is an identifying relationship as HARDWARE_MAPPING would not exist without ENDPOINT_NODES and the other table.

j. FUNCTION_MAPPING and FUNCTIONS_DEFINITION

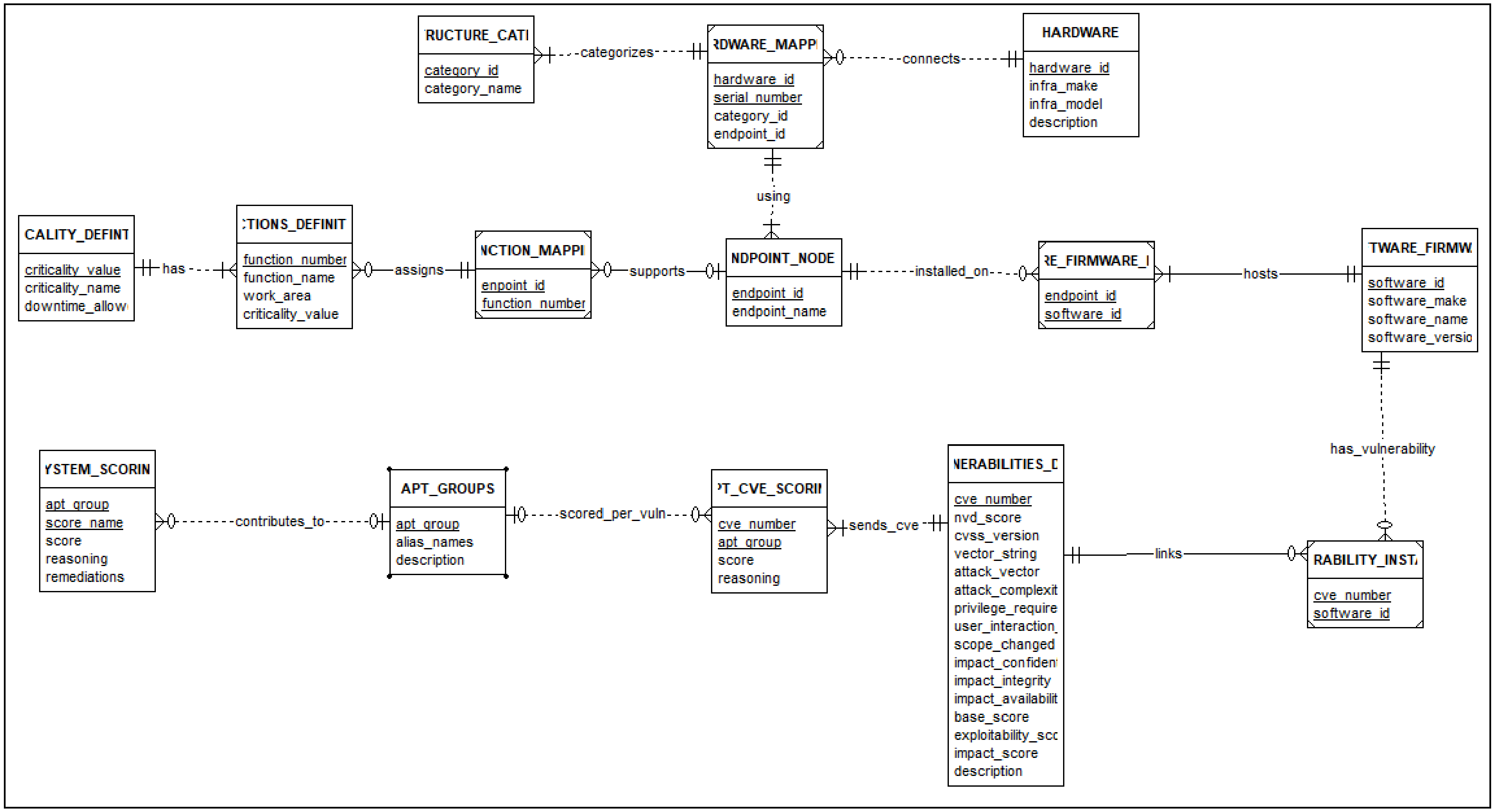
   i. This is a mandatory one to optional many relationship (1:M). Each function can be mapped to multiple endpoints through this table. FUNCTION_MAPPING is an associative entity. Each mapping in the table corresponds to a single function, making this 1:M.

k. FUNCTIONS_DEFINITION and CRITICALITY_DEFINITION
   i. This is a mandatory one to mandatory many relationship (1:M). Each function is assigned a criticality level of low, medium, or high. A single criticality value can be used for many different functions, but each function can only have one criticality level.

l. HARDWARE_MAPPING and INFRASTRUCURE_CATEGORY
   i. This is a mandatory one to optional many relationship (1:M). This relationship maps the hardware used on each endpoint to the category that the hardware falls under. Each singular hardware instance had a category but could be included in more than one category making this 1:M.

m. HARDWARE_MAPPING and HARDWARE
   i. This relationship is a mandatory one to optional many (1:M). Each unique hardware id is mapped to an endpoint id and category id. Each individual hardware piece can be on multiple endpoints or categories, making this a 1:M relationship.

3. Entity Relationship (E-R) Diagram



III. RELATIONAL SCHEMA

1. Rationale for Additional Tables

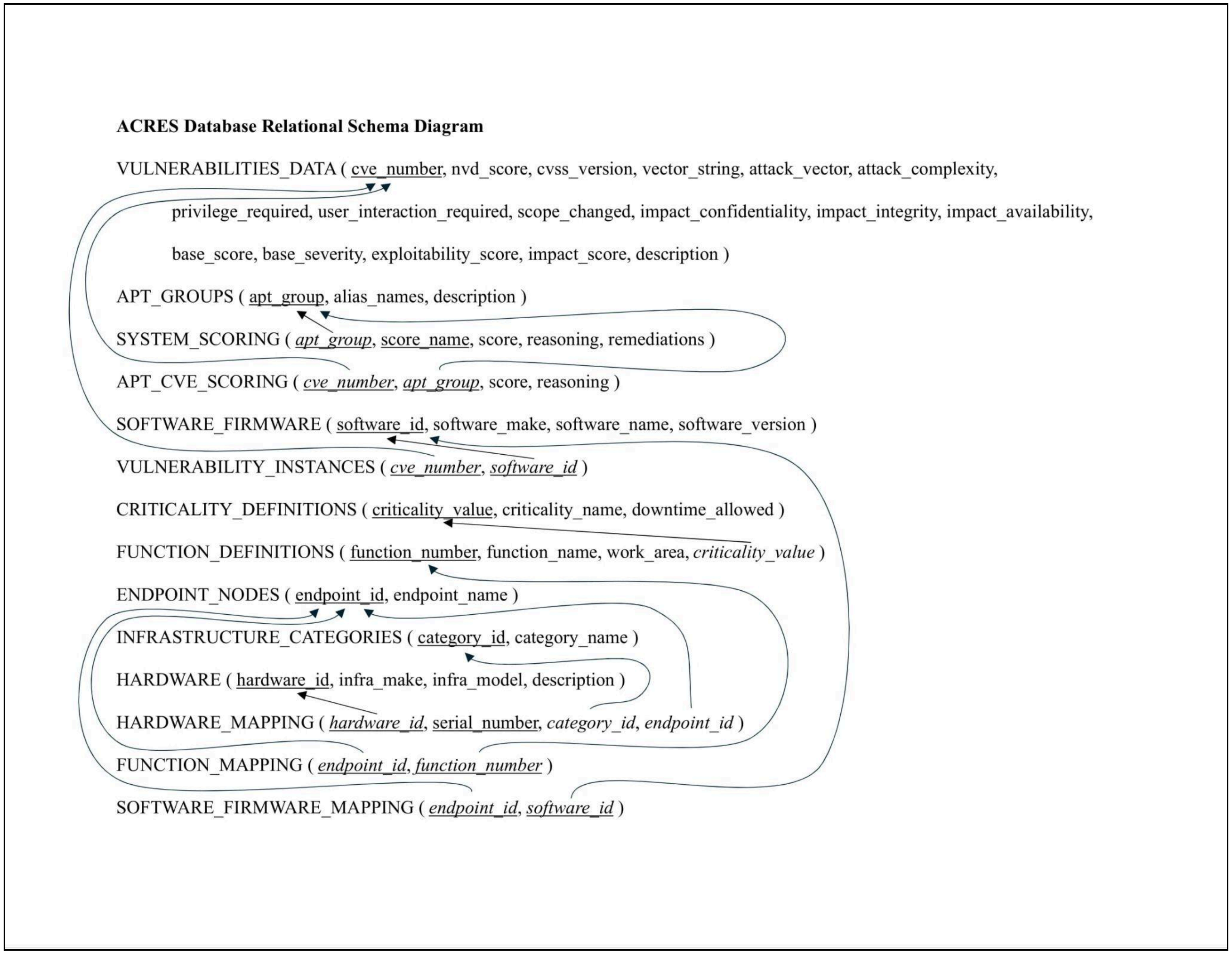
The following tables were identified as needing additional rationale for their necessity:

a. APT_CVE_SCORING: In this table, a score with associated reasoning is stored for each CVE detected. This data could be stored in the VULNERABILITIES_DATA table, but the user would be limited to only storing the scores for a single APT group evaluated. By having a separate table and using a composite key of cve_number and apt_group, all APT groups evaluated to date by the software can have their scoring stored.

b. VULNERABILITY_INSTANCES: This table is required since even though a detected CVE will be associated with a piece of software or firmware within the system, and therefore could be stored in the SOFTWARE_FIRMWARE table, a detected vulnerability may effect multiple versions of software or firmware that are all present in the same system, and a piece of software or firmware could and likely does have multiple vulnerabilities, not just one.

c. SOFTWARE_FIRMWARE_MAPPING, HARDWARE_MAPPING, and FUNCTION_MAPPING: These three tables map either software/firmware, hardware, or function roles to the different endpoints. While some of this information could be saved to the ENDPONT_NODES table, separating the components of the endpoints is required as endpoints may and likely will have multiple types of software/firmware, hardware, and/or function roles associated with each of them, and the same types or versions of these four objects can also exist on multiple endpoints at the same time, for example, an individual server may perform more than one role, and will also have multiple types of software installed on it.

2. Relational Schema Diagram



ACRES Database Relational Schema Diagram

VULNERABILITIES_DATA (cve_number, nvd_score, cvss_version, vector_string, attack_vector, attack_complexity, privilege_required, user_interaction_required, scope_changed, impact_confidentiality, impact_integrity, impact_availability, base_score, base_severity, exploitability_score, impact_score, description)

APT_GROUPS (apt_group, alias_names, description)

SYSTEM_SCORING (*apt_group*, score_name, score, reasoning, remediations)

APT_CVE_SCORING (*cve_number*, *apt_group*, score, reasoning)

SOFTWARE_FIRMWARE (software_id, software_make, software_name, software_version)

VULNERABILITY_INSTANCES (*cve_number*, *software_id*)

CRITICALITY_DEFINITIONS (criticality_value, criticality_name, downtime_allowed)

FUNCTION_DEFINITIONS (function_number, function_name, work_area, *criticality_value*)

ENDPOINT_NODES (endpoint_id, endpoint_name)

INFRASTRUCTURE_CATEGORIES (category_id, category_name)

HARDWARE (hardware_id, infra_make, infra_model, description)

HARDWARE_MAPPING (*hardware_id*, serial_number, *category_id*, *endpoint_id*)

FUNCTION_MAPPING (*endpoint_id*, *function_number*)

SOFTWARE_FIRMWARE_MAPPING (*endpoint_id*, *software_id*)

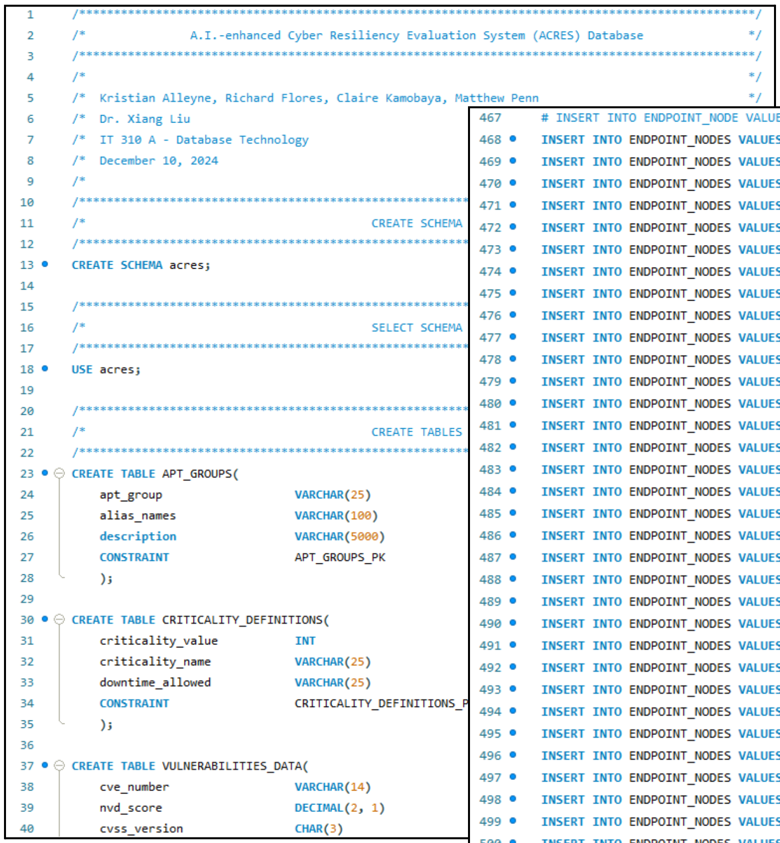
IV. PROJECT IMPLEMENTATION

1. Implementation Summary

Our Data Definition Language (DDL) script can be broken down into four sections. The first two are very short; CREATE SCHEMA creates the schema, and USE selects it as the working schema. Next, the CREATE TABLE section creates the 13 tables that compose our database. Finally, the INSERT data section populates our tables with some data. We successfully converted our JSON files representing the data from the original challenge by strategically utilizing the find and replace features in a combination of VS Code and Excel, supplemented with manual cleanup and handcrafting a few tables.

2. Link to DDL File

https://github.com/RogueFlotilla/ACRES.git



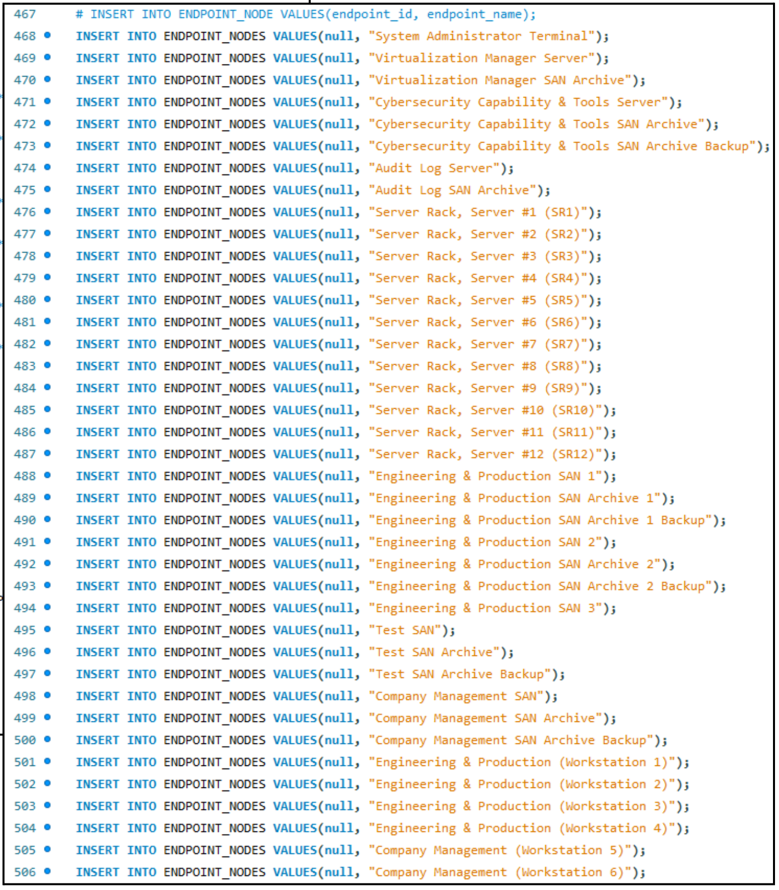
```
/*****************************************************************************************/
/*              A.I.-enhanced Cyber Resiliency Evaluation System (ACRES) Database        */
/*****************************************************************************************/
/*                                                                                       */
/*  Kristian Alleyne, Richard Flores, Claire Kamobaya, Matthew Penn                      */
/*  Dr. Xiang Liu
/*  IT 310 A - Database Technology
/*  December 10, 2024
/*
/**********************************************************
/*                                            CREATE SCHEMA
/**********************************************************
CREATE SCHEMA acres;

/**********************************************************
/*                                            SELECT SCHEMA
/**********************************************************
USE acres;

/**********************************************************
/*                                            CREATE TABLES
/**********************************************************
CREATE TABLE APT_GROUPS(
    apt_group                   VARCHAR(25)
    alias_names                 VARCHAR(100)
    description                 VARCHAR(5000)
    CONSTRAINT                  APT_GROUPS_PK
    );

CREATE TABLE CRITICALITY_DEFINITIONS(
    criticality_value           INT
    criticality_name            VARCHAR(25)
    downtime_allowed            VARCHAR(25)
    CONSTRAINT                  CRITICALITY_DEFINITIONS_P
    );

CREATE TABLE VULNERABILITIES_DATA(
    cve_number                  VARCHAR(14)
    nvd_score                   DECIMAL(2, 1)
    cvss_version                CHAR(3)
```



```
# INSERT INTO ENDPOINT_NODE VALUES(endpoint_id, endpoint_name);
INSERT INTO ENDPOINT_NODES VALUES(null, "System Administrator Terminal");
INSERT INTO ENDPOINT_NODES VALUES(null, "Virtualization Manager Server");
INSERT INTO ENDPOINT_NODES VALUES(null, "Virtualization Manager SAN Archive");
INSERT INTO ENDPOINT_NODES VALUES(null, "Cybersecurity Capability & Tools Server");
INSERT INTO ENDPOINT_NODES VALUES(null, "Cybersecurity Capability & Tools SAN Archive");
INSERT INTO ENDPOINT_NODES VALUES(null, "Cybersecurity Capability & Tools SAN Archive Backup");
INSERT INTO ENDPOINT_NODES VALUES(null, "Audit Log Server");
INSERT INTO ENDPOINT_NODES VALUES(null, "Audit Log SAN Archive");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #1 (SR1)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #2 (SR2)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #3 (SR3)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #4 (SR4)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #5 (SR5)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #6 (SR6)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #7 (SR7)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #8 (SR8)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #9 (SR9)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #10 (SR10)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #11 (SR11)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Server Rack, Server #12 (SR12)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN 1");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN Archive 1");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN Archive 1 Backup");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN 2");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN Archive 2");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN Archive 2 Backup");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production SAN 3");
INSERT INTO ENDPOINT_NODES VALUES(null, "Test SAN");
INSERT INTO ENDPOINT_NODES VALUES(null, "Test SAN Archive");
INSERT INTO ENDPOINT_NODES VALUES(null, "Test SAN Archive Backup");
INSERT INTO ENDPOINT_NODES VALUES(null, "Company Management SAN");
INSERT INTO ENDPOINT_NODES VALUES(null, "Company Management SAN Archive");
INSERT INTO ENDPOINT_NODES VALUES(null, "Company Management SAN Archive Backup");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production (Workstation 1)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production (Workstation 2)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production (Workstation 3)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Engineering & Production (Workstation 4)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Company Management (Workstation 5)");
INSERT INTO ENDPOINT_NODES VALUES(null, "Company Management (Workstation 6)");
```

V. SAMPLE SQL QUERIES

1. Built-in function Query

```sql
SELECT

      base_severity,

      AVG(nvd_score) AS average_nvd_score

FROM

      VULNERABILITIES_DATA

WHERE

      base_severity = 'HIGH'

GROUP BY

      base_severity;
```

This query's purpose is to calculate the average National Vulnerability Database (NVD) score for vulnerabilities with a 'HIGH' base severity. It then groups the vulnerabilities by their base severity level and only pays attention to those with the 'HIGH' severity classification, and then computes the mean of the associated NVD scores. This calculation can help with understanding the average threat level of high severity vulnerabilities in the system which helps cybersecurity analysts to prioritize their remediation efforts.

2. Using WHERE to filter results

```
SELECT

      vd.cve_number,

      vd.description,

      vd.nvd_score

FROM

      VULNERABILITIES_DATA vd

JOIN

      CRITICALITY_DEFINITIONS cd

      ON vd.base_severity = cd.criticality_name

WHERE

      vd.nvd_score > 8.0

      AND cd.criticality_name = 'HIGH';
```

This query retrieves detailed information about vulnerabilities with an NVD score exceeding 8.0 that are classified with a criticality level of "HIGH." By joining the VULNERABILITIES_DATA and CRITICALITY_DEFINITIONS tables, the query ensures the accurate mapping of vulnerability data to their corresponding criticality levels. The output includes the CVE number, a description of the vulnerability, and its NVD score, providing analysts with the information needed to address the most severe vulnerabilities effectively.

3. Using INNER JOIN to join two tables together

```
select ag.apt_group, vd.cve_number, acs.score as "exploitation likelihood",

vd.description as vulnerability_desc

from

      apt_groups ag

inner join

      apt_cve_scoring acs on ag.apt_group = acs.apt_group

inner join

      vulnerabilities_data vd on acs.cve_number = vd.cve_number

where acs.score >= .6;
```

This query aims to identify the most likely vulnerabilities to be exploited by particular groups of Advanced Persistent Threat (APT). It correlates APT grouping data with vulnerability exploitation scores, highlighting vulnerabilities with a likelihood score of 0.6 or above (critical for immediate patching). The resulting data includes the corresponding CVE number, the APT group name, and a description of the vulnerability, as well as the exploitation likelihood score, which can be used to take targeted actions to reduce the risk of being attacked.

4. Process using one-sided OUTER JOIN

```
SELECT

      sf.software_id,

      vd.nvd_score, vd.cve_number

FROM

      SOFTWARE_FIRMWARE sf

LEFT OUTER JOIN

      VULNERABILITY_INSTANCES vi

      ON sf.software_id = vi.software_id

LEFT OUTER JOIN

      VULNERABILITIES_DATA vd

      ON vi.cve_number = vd.cve_number

WHERE

      vd.base_severity = 'CRITICAL' OR vd.base_severity = 'HIGH' OR

vd.base_severity IS NULL

ORDER BY

      sf.software_id;
```

This query lists all software IDs and associated vulnerabilities which are classified as "HIGH" or "CRITICAL," and identifies software without any detected vulnerabilities. The query will make sure that even if a software is not recorded to have any vulnerability it is still returned in results, via left outer joins. The ordered output gives IT staff a complete overview, identifying critical software components that need prioritized updates or maintenance, and identifying secure systems that do not need immediate action.