Lego Inventory Tracker

Security and Data Integrity Analysis

Team Name:S2G8

Members: Luke Ferderer, Jamari Morrison, Jason Cramer

Signature Member #1: Luke Ferderer

Signature Member #2: Jamari Morrison

Signature Member #3: Jason Cramer

Table of Contents

1. Executive Summary

The goal of this document is to summarize the privacy commitments, possible threats, and defensive measures for our database's security. It also analyses the integrity of constraints and business rules of the database. The Privacy Analysis identifies protected information and outlines steps taken to ensure that information's integrity. Security Analysis covers some of the vulnerabilities our database has and how they're being addressed. The entity integrity analysis covers the ways we enforce attribute uniformity in tables, while the referential integrity analysis covers the rules for handling foreign key updates. The business rule integrity analysis explains how we handle representing data updates between tables.

2. Privacy Analysis

All usernames will only be accessible to the user and database owner, so users will not be able to view each other's usernames. This will be enforced by requiring all username-related queries, aside from registration and login, to be run while logged into the target account. Keeping usernames private prevents leaks of account-related information. Although it may be the general assumption that usernames are going to be public data, and as such, should not include personal information, we have no need for usernames to be public because there is no interaction between accounts. Because of this, we have decided to treat usernames as private data.

Passwords will be stored as hashes in the database, so no one will be able to access the password of any given user. Tha hashing of passwords helps our database follow up on the generalized expectation that only the owner of an account will be able to log into said account [1]. Passwords are considered private because they block access to the rest of the data on an account. The amount someone wishes their password not to be known is directly proportional to how much they value their account because the leak of a password directly leads to an account breach. Another reason passwords are considered private is because, although people should not, people share their password between many accounts. This means the leak of a password from our database could be disastrous for a user's other accounts. By hashing our passwords, no one is able to know what an account;s password is without directly guessing/brute forcing it [1].

3. Security Analysis

SQL databases are vulnerable to many types of security breach attempts, including SQL Injections [2], user password compromisation, and denial or service attacks [3]; without proper protection in place, these attacks can cripple the database and cause data leaks and destruction.

Prior to storing any user password in our database, we will be using bcrypt [1] to securely hash and store the passwords in the database. The hash will be generated with a minimum cost factor of 13 for the salt. This will ensure that if the database becomes compromised, user information will not be able to accessed. With hashed passwords, the only way to acquire someone's password is through brute force attacks. To prevent this attack, we will limit the amount of failed password attempts to three every 10 minutes. The failed attempts will be stored in a session variable and the server will deauth the client if the failed attempts reach the failure threshold.

SQL injections can lead to unwanted/trash data being added, information being stolen, data deletion, or a variety of other unwanted behavior in our database. [2] To prevent injections, the only way for the backend to interact with the database is through stored procedures. This means nothing a user entered is ever entered into a SQL query directly which prevents SQL injection attacks. On top of this, our front-end checks data inputs into the stored procedures; it also ensures users are logged in and can only run user related queries on their account data.

Denial of service attacks can cause an entire database to go down, possibly causing our Lego inventory tracking service to be unavailable for hours at a time. [1] The express backend will be implementing user sessions to prevent an unauthenticated accessor of our frontend from making modifications to the database. It will also limit the speed at which someone can run stored procedures on the database.

Upon login of a user to our application, a secure session will be generated by our backend and sent to the frontend client. The session will contain a randomly generated key to authenticate that the username requests that are sent to the database come from the correctly authenticated user. This ensures the integrity of the data by preventing people with access to our API from writing invalid POST/PUT/DELETE requests.

4. Entity Integrity Analysis

1. The User table must contain a username that is a custom username type(varchar(20)) and is the primary key. Hashed passwords will be stored as a nvarchar(70). Both fields must be not null.
2. A custom datatype for ImageURL was created to ensure tables with ImageURL fields are varchar(100) that are nullable.
3. The LegoModel table will have an integer ID that is auto-incremented, not null and will be the primary key. The name of the model can be up to 80 characters long and cannot be null. SetID is a foreign key that references the ID of the set it belongs to which can be nullable if the model does not belong to a set.
4. The LegoSet table will contain an integer ID which cannot be null and is the primary key. Name is a varchar(80) and cannot be null
5. The LegoBrick table contains an ID with type BrickID which is up to length 20 and is a numeric string of the format '#####/#####' where the amount of digits before and after the '/' are variable but non-zero. The name field is limited to 80 characters and is not null. Color can be up to 30 characters and is nullable.
6. The ModelContains table contains a Model which is an integer that is a foreign key that references the id of the model and cannot be null. LegoBrick is a BrickID type which references the brick id of the lego brick the model contains. It is a foreign key and cannot be null. The quantity is an integer that is greater than zero and cannot be null. The combination of Model, LegoBrick, and Quantity form the primary key.
7. The Builds table contains a Model which is an integer id that is not null and is a foreign key which references the LegoModel being built. Username which is a username type cannot be null and is a foreign key that references the username of a User entity. The quantity is an integer that cannot be null and must be greater than zero. The Model and Username together form the primary key.
8. The Contains table contains LegoSet which is an integer that is a foreign key that references a lego set id and cannot be null. LegoBrick is a BrickID type that is a foreign key that references a LegoBrick entity and cannot be null. Quantity is an integer that must be greater than zero and cannot be null. LegoSet and LegoBrick from the primary key.
9. The WantsSet table contains a username which is of type username and cannot be null. LegoSet is an integer foreign key that references the id of a LegoSet entity and cannot

be null. Quantity is an integer that must be larger than zero and not null. The combination of Username and LegoSet form the primary key.

10. The WantsBrick table contains a username which is of type username and cannot be null. LegoBrick is a BrickID type foreign key that references the id of a LegoBrick entity and cannot be null. Quantity is an integer that must be larger than zero and not null. The combination of Username and LegoBrick form the primary key.
11. The OwnsSet table contains a username which is of type username and cannot be null. LegoSet is an integer foreign key that references the id of a LegoSet entity and cannot be null. Quantity is an integer that must be larger than zero and not null. The combination of Username and LegoSet form the primary key.
12. The OwnsIndividualBrick table contains a username which is of type username and cannot be null. LegoBrick is a BrickID type foreign key that references the id of a LegoBrick entity and cannot be null. Quantity is an integer that must be larger than zero and not null. The combination of Username and LegoBrick form the primary key.

5. Referential Integrity Analysis

Foreign key relations require a behavior on update and delete. All update operations were set to cascade with deletes being split between cascade and reject. In explaining the reasoning behind the referential integrity behavior choices, tables will be grouped if the reasons for their choices are the same.

1. The WantsSet, WantsBrick, OwnsSet and OwnsBrick tables are almost identical in purpose: they describe a relationship between a user and a Lego product this user wants or owns. If the username is updated, the change is cascaded because the relationship still exists now with a different username representing the same user. If the username is deleted, the user has been deleted from the database and their want and ownership of Lego products is no longer relevant to the database. For this reason, cascade was also chosen on delete. For the relevant Lego product for the table, if the ID is updated, a cascade operation occurs because the wants and owns relation still exists, now with a different ID for the Lego product. On deletion of the Lego Product, the change is rejected because people still want or own that product and it should not be removed from the database until those relations are removed. This prevents users from losing data if a Lego product is removed unexpectedly.

2. The tables Contains and Model Contains tables both describe a relationship between a Lego construction and the bricks it contains. If the Lego construction's ID updated, the update is cascaded because it still contains the same bricks. If the Lego construction is deleted, the delete is also cascaded because the relation defining what bricks it contains is no longer relevant. If the brick ID changes, the change is cascaded because the construction still contains that brick, now with a new ID. If the brick is deleted, the change is rejected because there is still a Lego construction that is using that brick. The brick should be removed from each construction before deletion from the LegoBrick table.
3. The Builds table is the last remaining table with foreing keys. This table represents a relationship between users and a lego model that describes how many of that model they have built. On an update of the model's ID, the change is cascaded because the relationship between the user and model still remains under the new ID. If the ID is deleted, the change is rejected because there is still a user who has constructed this model. A model should not be deleted from the database while it remains built by the user. If the username is updated, a cascade occurs for the same reasoning as cascading on the model ID. If the username is deleted, a cascade also occurs because the relationship this table models is no longer relevant if the user is removed from the database.

6. Business Rule Integrity Analysis

When a set is purchased, each brick that set is added to the user's individual brick inventory by either creating or increasing the quantity of the OwnsIndividualBrick relationship. When a model is deconstructed, the bricks required to build that model have their QuantityInUse decreased in the OwnsIndividualBrick relationship. When a set is removed, each brick from that set is removed from the user's individual brick inventory by either deleting or decreasing the quantity of the OwnsIndividualBrick relationship. When a model is built, the bricks required to build that model have their QuantityInUse increased according to their quantities in the model contents.

7. References

[1] Password security term Definitions: https://www.theguardian.com/technology/2016/dec/15/passwords-hacking-hashing-salting-sha-2#:~:text=Salting%20is%20simply%20the%20addition,same%20salt%20for%20every%20password.
[2] SQL Injection Definition: https://owasp.org/www-community/attacks/SQL_Injection
[3] Denial of Service Attack Definition: https://us-cert.cisa.gov/ncas/tips/ST04-015

8. Index

9. Glossary

Bcrypt - A password hashing algorithm that works using the Blowfish cipher; developed for secure password storage on a database. Blowfish is a symmetric key block cipher that allows information to be encrypted securely without any risks of leaked data.

Denial of Service Attack - An attack that causes legitimate users to be unable to access a system. In this case, we reference a flood-style denial of service attack, where an attacker spams the target server with information in an attempt to take up all available resources.

Hash - A hash refers to a function that maps a string into another usually more condensed value. In terms of password hashing, the function is a unidirectional function that maps exactly one string to a unique expanded hash.

Salt - A randomly generated key used to generate the password hash. Since the hash function is one-directional, salts can be appended to the front of the hashed password and used for the seed to the hash function.

SQL Injection - When an attacker enters malicious SQL scripts into an input field in an attempt to get the server to run the malicious script instead of the intended function.

Lego - Mainly consisting of interlocking plastic bricks, legos are a toy created to allow for building and creation. Also refers to the LEGO Group.

Brick - An individual lego piece which is commonly referred to by its size.

Lego Set - A set of bricks that one buys from Lego that, when the instructions are followed, can be assembled into a specific object. An example of a Lego set would be the Star Wars X-Wing.

Web Scraper - An application that gathers information from websites.