Levi Crosier

Professor Nalubandhu

SWENG 837 – Software System Design

April 26, 2024

Course Project – Authentication System

Revision 1

Introduction

This document is intended to serve as a guidance document for an authentication system design. Among other things, the document includes a high level domain model for authentication necessities and interactions; base skeleton class implementations; and potential deployment recommendations to guide development of the system. This is the first iteration of the system, and many references to another iteration were discovered well into the course. There was not enough time to always reiterate on the system, and so they are merely documented as improvement points.

Business Goals

Problem Statement

The world has seen a large rise in the focus on security and security requirements. Companies and businesses utilize email systems, messaging systems, and many additional services and products to conduct their necessary operations. Access to these systems needs to be restricted. Utilizing the built in authentication mechanisms of these systems can be cumbersome and does not scale, especially in large environments where employees are constantly joining and leaving the organization.

The new system will allow other systems to authenticate users and allow access to those systems. It will support multiple authentication techniques including password authentication, cipher key authentication, and token authentication.

In today's security landscape, it is important to learn how unauthorized access is gained and find the entry points of opponents. It is imperative that companies know when old passwords are compromised and to keep proper logs to understand entry points.

Actor Identification

The following table identifies primary, secondary, and offstage actors that interact directly or closely, yet indirectly, with the system:

Type	Actor	Comment
Primary	Employees	These are employees that are employed by the owner of the system. This encompasses full time, part time, and contracted employees.
	Administrators	System owners and super users that will perform administrative tasks on the system
	Customers	Clients of the owners that will use external applications of the owners and require authentication
Secondary	Log System	Logging system or subsystem to allow for error tracing or to help identify malicious intent
	Data Storage System	Storage system or subsystem that will be used to store information from the system.
	MFA Provider	Multi-factor authentication provider that will be utilized to support MFA/2FA
Offstage	Authorization System	External system/subsystem that will apply appropriate permissions following successful authentication
	External Applications	Applications that will utilize this system for authentication

Use Cases

This section covers the initial use cases of the system. These use cases are used throughout this document to build an initial set of requirements that influence the design of the system. Adding, removing, or modifying these use cases will modify everything else that follows in this document. This primarily includes the Domain Model and System Operation Contracts that influence other aspects of the document. Each use case is followed by an appropriate system sequence diagram to demonstrate how the system interacts with the outside world.

Table 1: Use Case 1

Use Case Section	Description
Use Case Name	Authenticate Login Credentials
Scope	Secure Authentication System
Level	Sub function
Primary Actor	Employee
Stakeholders and Interests	Employee: Wants to log into email client to view and send an email Data Store: Storage location for authentication credentials MFA Subsystem: Provides OTP for User External Application: The email client that uses the system for authentication
Preconditions	-
Success Guarantee	Employee is authenticated
Main Success Scenario	1. The email client establishes a secured, private connection to the system 2. Email client provides the system with username and password 3. The system verifies that the user exists in its data store 4. The system verifies that the supplied password is equivalent to the password hash stored in the data store with the user 5. The system asks the client for a OTP 6. The email client asks the Employee for a OTP, and the employee provides the OTP to the email client 7. The email client provides the OTP to the system 8. The system creates a OTP request to the MFA subsystem and provides the OTP to the MFA subsystem 9. The MFA subsystem responds indicating that the OTP is correct 10. The system informs the email client that the authentication was successful
Extensions	User does not exist: 3a. The system does not find the User in the data store 3b. The system returns an error to the client indicating that the user does not exist User password is incorrect: 4a. The system returns an error to the client indicating that the password is incorrect User provides incorrect OTP:

	9a. The MFA subsystem responds indicating an incorrect OTP 9b. The system informs the email client of the incorrect OTP
Special Requirements	-

Table 2: Use Case 2

Use Case Section	Description
Use Case Name	Authenticate SSH Keys
Scope	Secure Authentication System
Level	Sub function
Primary Actor	Employee
Stakeholders and Interests	Employee: Wants to log into a network switch to diagnose network troubles Data Store: Storage location for SSH keys External Application: The SSH Server that will be allowing the user to connect to the switch
Preconditions	-
Success Guarantee	Public key is added to SSH Server's Trust Store
Main Success Scenario	1. The SSH Server establishes a secured, private connection to the system 2. The SSH Server provides the system with the employee's username and public key 3. The system verifies that the user exists in the data store 4. The system verifies that the user has a public key in the data store 5. The system provides a public key to the SSH Server 6. The SSH Server then adds the public key to its local trust store 7. The SSH Server performs SSH Key authentication with the employee
Extensions	User does not exist: 3a. The system does not find the User in the data store 3b. The system returns an error to the client indicating that the user does not exist User public key is incorrect: 4a. The system returns an error to the client indicating that the public key is incorrect and should not be added to the trust store
Special Requirements	-

Table 3: Use Case 3

Use Case Section	Description
Use Case Name	Connect API
Scope	Secure Authentication System
Level	Sub function
Primary Actor	System Administrator
Stakeholders and Interests	System Administrator (Admin): Responsible for administering the system, such as integrating external services Employee: Developer with a new API that requires tokens in order to use Data Store: Storage location for App authentication information External Application (App): Created by developer that will request new tokens from the system
Preconditions	• System Administrator is logged in • Employee's application is ready to request and receive tokens
Success Guarantee	System administrator retrieves ID and Secret for App
Main Success Scenario	1. Admin opens the "Administration" page 2. Admin enters App name 3. Admin designates the app as an API that will need Tokens 4. Admin enters the app resources that require authentication 5. Admin selects 'Save' 6. System generates ID and Secret 7. System adds salt to ID and Secret, and then the system hashes the ID and Secret. 8. The system saves the ID, Secret, App Name, and resources to the Data store 9. System displays the original ID and Secret to Administrator
Extensions	-
Special Requirements	OAuth 2.0 Authentication

Table 4: Use Case 4

Use Case Section	Description
Use Case Name	Generate Access Tokens
Scope	Secure Authentication System
Level	Sub function
Primary Actor	System Administrator
Stakeholders and Interests	Employee: Developer with a new API that requires tokens in order to use Data Store: Storage location for tokens and App authentication information External Application (App): Application that is using the system to generate and manage access tokens
Preconditions	• The app is connected to the system
Success Guarantee	App receives an access token
Main Success Scenario	1. The app requests a new token from the system 2. The system requests authentication details from the app 3. The app sends the authentication details to the system 4. The system verifies the authentication details of the app 5. The system generates a new access token and provides it to the app
Extensions	-
Special Requirements	OAuth 2.0 Authentication

Table 5: Use Case 5

Use Case Section	Description
Use Case Name	Account Creation
Scope	Secure Authentication System
Level	Sub function
Primary Actor	System Administrator
Stakeholders and Interests	System Administrator (Admin): Developer with a new API that requires tokens in order to use Data Store: Storage location for accounts and account information External Application (App): Application that is using the system for authentication Employee: New hire that needs access to company assets
Preconditions	System Administrator is logged in
Success Guarantee	The new account is saved to the data store
Main Success Scenario	1. The admin opens the “Administration” page 2. The admin selects “New User” 3. The system returns the New User form 4. The admin enters the employees necessary information into the form including full name, job title, organization, username and password 5. The admin indicates that the password needs changed on login 6. The admin indicates that the user will use MFA 7. The admin clicks ‘Save’ 8. The system saves the users information to the database
Extensions	-
Special Requirements	-

Table 6: Use Case 6

Use Case Section	Description
Use Case Name	Forgotten Password
Scope	Secure Authentication System
Level	Sub function
Primary Actor	Customer
Stakeholders and Interests	Customer: Customer who forgot their password Data Store: Storage location for accounts and account information
Preconditions	- The customer is at the login screen - The customer has MFA enabled and configured properly
Success Guarantee	The new password is saved to the data store
Main Success Scenario	1. The customer clicks ‘Forgot Password’ 2. The system requests a Username from the customer, and the customer provides it 3. The system submits a One-Time Password (OTP) request to the MFA system 4. The system asks the customer for a OTP 5. The customer provides the OTP to the system 6. The system verifies the OTP is correct with the MFA system 7. The system presents the user for a ‘New Password’ form 8. The customer enters in their new password 9. The system saves the new password to the database for the customer
Extensions	-
Special Requirements	-

Domain Information

The following section contains information related to building an initial domain model for the authentication system. The first table in this section contains the class concepts derived from the Use Cases. The second table is a slit list of classes that are kept for the domain model and classes that are pruned from the model. During the initial revision, many conceptual classes are being treated as methods, while many Roles have been determined through proper authorization. As authorization is not currently being included within this system, these aspects are currently being pruned. It is possible that a later revision could deem them necessary.

During the construction of the system operation contracts and domain class diagram, it was noticed that the SSH Server may need to remain as an conceptual or potentially physical object as they have their own authentication mechanisms. Reevaluation of the implementation of the SSH Server authentication aspects of the system should be the primary goals of the next revision.

Table 7: Conceptual Classes

Conceptual Class Categories	Examples
Physical or Tangible Objects	SSH Server
Specifications, Designs, or Descriptions of Things	User Information, Credential, One-time Password, ~~Admin Panel,~~ User
Places	-
Transactions	~~Authenticate, Request, Save~~
Transaction Line Items	~~Username~~, One-time Password
Roles of People	~~Developer~~, System Administrator, Employee, Customer
Containers of Other Things	~~Form, Page~~
Things in a Container	~~Form~~, User Information, Application Information, SSH Server Information
Other Computers/Systems (external)	Multi-factor Authentication Subsystem, Data Store Subsystem
Abstract Noun Concepts	Application, Form, Credential, Token, Key
Organizations	-

Events	~~Save, Submit, Request, Authentication, Update, New User, New Application, Get User, Add Employee~~
Processes	~~Request One-time Password~~
Rules and Policies	-
Catalogs	User Catalog, Token Catalog, Key Catalog, Application Catalog, SSH Server Catalog
Records of Finance, Work, Contracts, Legal Matters, etc.	-
Financial Instruments and Services	-
Manuals, books, Documents, Reference Papers	-

Table 8: Pruned Classes

Good Classes (Retained)	Bad Classes (Pruned)
User	Admin Panel
User Information	Request One-time Password
User Catalog	Save
Key	Submit
Key Catalog	Request
Token	Authentication
Token Catalog	Update
Credential	New User
Credential Catalog	New Application
Application	Get User
Application Catalog	Add Employee
Application Information	Form
One-time Password	Page
Customer	Multi-factor Authentication Subsystem
System Administrator	Data Store Subsystem
Employee	Developer
SSH Server	
SSH Server Catalog	
SSH Server Information	

System Operation Contracts

Next is an evaluation of many of the system operation contracts that can be found through the use cases and system sequence diagrams of earlier sections. This list of contracts is the base list required for minimal functionality. Each contract includes a visual diagram to demonstrate how the systems classes from the domain model interact with each other. This assists in building and visualizing the domain class diagram in the following section.

One important note about this implementation is that the SSH key authentication mechanism would require a form of agent on the client to intercept the SSH authentication requests from users. This type of implementation requires an entire new system to simply perform the SSH authentication. This is not an ideal solution, but it can be remedied in the next iteration of the system. This can be achieved by taking a preemptive approach to SSH key management. Instead of sending a key upon request, which does not necessarily follow the SSH protocol, the system could be designed to remotely deploy and install public keys into their trust store using a service account. This would be performed ad hoc by a system administrator or when a user is added to the system. However, this could potentially begin to cross the boundary into authorization which is not a goal of this system.

Table 9: System Operation Contract 1

Field	Comment
Name of Operation	authenticateUser(username: integer, password: string)
Responsibilities	Check that the provided credentials (username/password) are correct and check that OTP is correct
Type	System
Cross Reference	Use Case: Authenticate Login Credentials
Exceptions	• If data store is unreachable, indicate there was an error • If user credentials are not found, indicate there was an error • If user credentials are invalid, indicate there was an error • If OTP is invalid, indicate there was an error
Pre-Conditions	User knows credentials and has the OTP device (whether physical or software)
Post-Conditions	-

Table 10: System Operation Contract 2

Field	Comment
Name of Operation	requestPublicKey(username: string)
Responsibilities	Check that the user exists and has a public key. Then, provide the public key
Type	System
Cross Reference	Use Case: Authenticate SSH Keys
Exceptions	• If data store is unreachable, indicate there was an error • If username is not found, indicate there was an error • If public key for user is not available
Pre-Conditions	User has a public/private key pair
Post-Conditions	Public key is provided to requesting SSH Server

Table 11: System Operation Contract 3

Field	Comment
Name of Operation	addApiResource(resourceUri: string)
Responsibilities	Adds the resource URI to the specified API object to allow for token authentication
Type	System
Cross Reference	Use Case: Connect API
Exceptions	• If data store is unreachable, indicate there was an error • If application is not found, indicate there was an error
Pre-Conditions	API is configured in the system
Post-Conditions	API resource is updated with a new resource URI

Table 12: System Operation Contract 4

Field	Comment
Name of Operation	updatePassword(username: string, password: string)
Responsibilities	Updates the password of a user account in the event of a forgotten password or a simple password change
Type	System
Cross Reference	Use Case: Forgotten Password
Exceptions	If data store is unreachable, indicate there was an error
Pre-Conditions	• The username exists • The user has MFA enabled and configured • The user has submitted their OTP correctly
Post-Conditions	The new credentials are associated with the appropriate User Record

Table 13: System Operation Contract 5

Field	Comment
Name of Operation	generateToken(application: string, resource: string, secret: string)
Responsibilities	Generates a token for use by an API resource
Type	System
Cross Reference	Use Case: Generate Access Tokens
Exceptions	If data store is unreachable, indicate there was an error
Pre-Conditions	The application exists
Post-Conditions	The new token is associated with the appropriate application

Domain Class Design

This section contains the domain class diagram for the authentication system. The domain class diagram is derived from the domain model and the system operation contracts. It establishes the basic concepts for classes within the system, which translate to the skeleton classes that follow. The skeleton classes are displayed in a C# format, but they can be easily translated to another language of choice.

Within the domain class diagram, you will also notice an abstract class not seen in the domain model called the ‘AuthHandler’. This Authentication Handler is intended to be the entry point into the system that determines the type authentication required, perform intermediary actions such as hashing or salt generation, and then pass the act of authentication onto the required class. If desired, the ‘AuthHandler’ can also be used to performed the required authentication.

The skeleton classes are followed by the initial data schema that can achieve functionality with the authentication system. One downside of this implementation is that there are very few foreign key uses, so as the system grows, query performance may degrade. As this is the first revision of the schema, it is intended as a base schema to be improved upon within the next revisions. This will allow for coupling of the system to be closely monitored and to prevent unintentional complexity.

Skeleton Classes

```
class AuthHandler {
        private int id;

        public bool compareCreds(string u1, string p1, string u2, string p2){};
        public string generateOTP(string username){};
        public string generateSecret(){};
        public string generatePasswordSalt(){};
        public string generatePasswordHash(string password, string salt){};
}

class User {
        private int id;
        private string name;
        private string password;

        public void forgotPassword(){};
}

class UserCatalog {
        private int id;
        public List<User>;

        public bool UserExists(string username){};
}


class OneTimePassword {
        private int id;
        private string data;

        public bool verifyOtp(string username, string otp){};
}

class Token {
        private int id;
        private string data;
}

class TokenCatalog {
        private int id;
        public List<Token> tokens;

        public Token generateToken(string application, string resource){};
}
```

```
class Resource {
        private int id;
        private string uri;
}

class Application {
        private int id;
        private string name;
        public int resource;

        public void addApiResource(string resourceUri){};
}

class ApplicationCatalog {
        private int id;
        public List<Application> applications;

        public void updateApplicationRecord(string resoruceUri, string secret){};
        public bool applicationExists(string application, string resource){};
        public bool verifySecret(string resource, string secret){};
}

class Key {
        private int id;
        private string name;
        private string data;
}

class KeyCatalog {
        private int id;
        public List<Key> keys;

        public Key getKey(string username){};
        public void generateKey(string username){};
}

class Credential {
        private int id;
        private string data;
}

class CredentialCatalog {
        private int id;
        public List<Credential> credentials;

        public Credential getCredentials(string username){};
        public void generateCreds(string username, string password){};
}
```

Table Schemas

Tokens

Field	Data Type	Comment
id	integer	Primary key
data	varchar	Base64 encoded data of the token

Application

Field	Data Type	Comment
id	integer	Primary key
name	varchar	Name of the application for identification
resources	integer	Foreign Key to the resource table’s ID field

Resource

Field	Data Type	Comment
id	integer	Primary key
uri	varchar	URI for the resource that requires a token

Credential

Field	Data Type	Comment
id	integer	Primary key
data	varchar	Credential data created from Username and Password

User

Field	Data Type	Comment
id	integer	Primary key
name	varchar	Username of the user
password	varchar	Salted and hashed password for the user

Key

Field	Data Type	Comment
id	integer	Primary key
name	varchar	Username associated with the public key
data	varchar	Public key RSA data

Components and Deployment

This final section of the document contains for a component diagram and a deployment diagram. The component diagram demonstrates the interconnections of different system components both for the authentication system itself and external subsystem requirements such s the MFA subsystem. An important note of the MFA subsystem is some basic current assumptions about the subsystem:

- There is an interface for generating one-time passwords
- There is an interface for sending the one-time password to a user such as email or SMS
- The MFA subsystem performs the verification required to complete authentication

The deployment diagram displays a high level separation of different aspects of the system within an overall infrastructure. This diagram, while simple, makes deployment flexible in different types of environments. For example, if the data stores are all databases, a simple REST API could be implemented for the ‘AuthHandler’ in AWS Elastic Beanstalk. The API could be configured to connect to an AWS managed RDS for storing and retrieving data, and sending one-time password requests to an MFA provider.