Brute-Force Attack Simulation on a Custom Banking API

Author: Muhaned Alhashimy
Date: 17 September 2025

Executive Summary

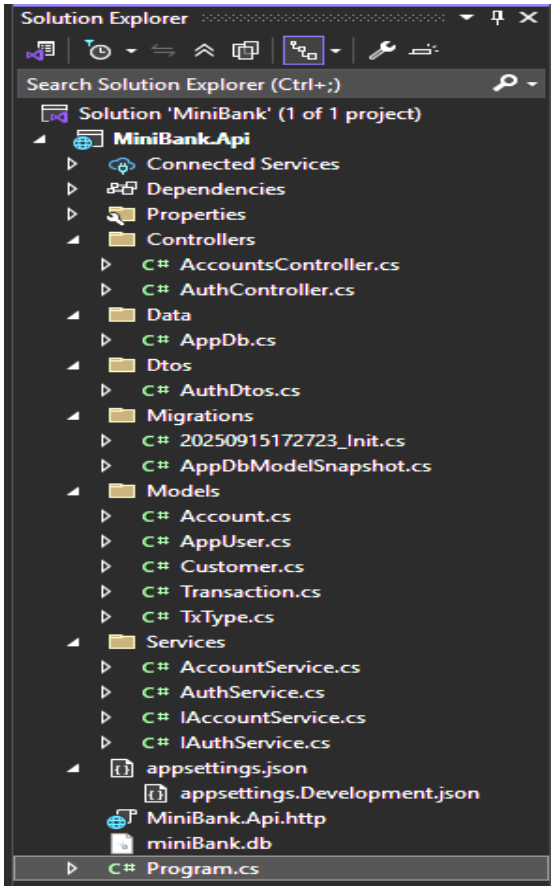
This report documents the development of a custom banking API built with **.NET 9 Web API** and **SQLite** as the backend database. The project simulates a vulnerable financial application and demonstrates how brute-force attacks can compromise weak authentication systems. Using Hydra from Kali Linux, multiple valid passwords were discovered. Evidence was collected from Swagger, curl, tcpdump, Hydra, and application logs. Security recommendations are provided at the end.

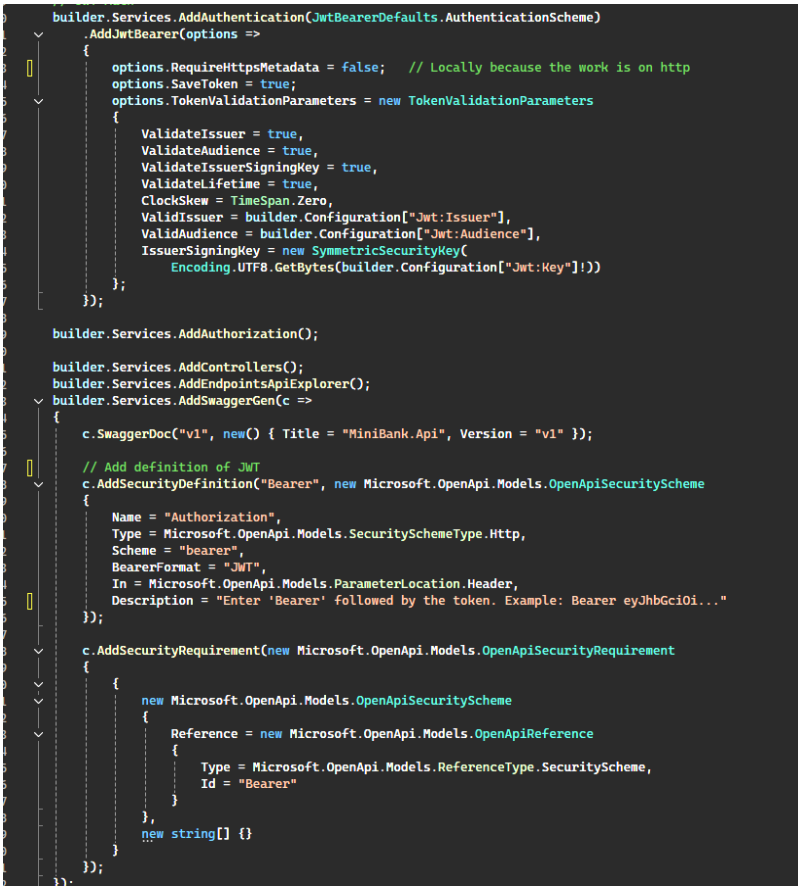
---

1. Application Development

1.1 Project Setup

The application was built in **Visual Studio 2022** as a **.NET 9 Web API solution**. The solution structure included entities, services, and controllers.

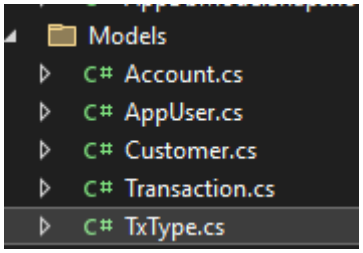


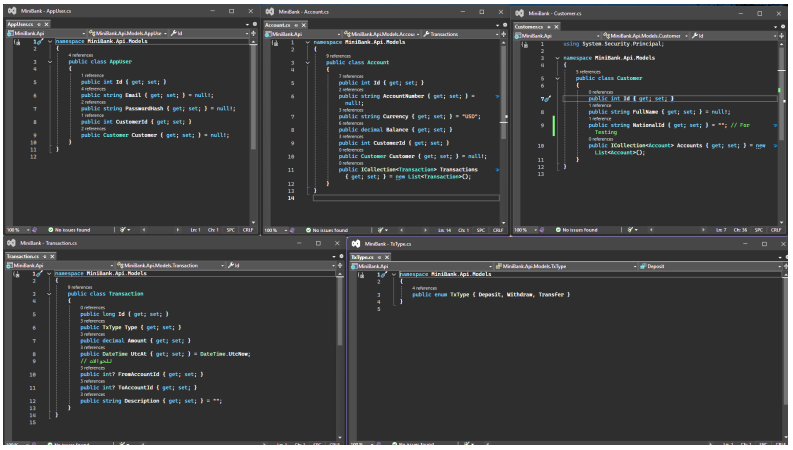


```
builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
    .AddJwtBearer(options =>
    {
        options.RequireHttpsMetadata = false;   // Locally because the work is on http
        options.SaveToken = true;
        options.TokenValidationParameters = new TokenValidationParameters
        {
            ValidateIssuer = true,
            ValidateAudience = true,
            ValidateIssuerSigningKey = true,
            ValidateLifetime = true,
            ClockSkew = TimeSpan.Zero,
            ValidIssuer = builder.Configuration["Jwt:Issuer"],
            ValidAudience = builder.Configuration["Jwt:Audience"],
            IssuerSigningKey = new SymmetricSecurityKey(
                Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]!))
        };
    });

builder.Services.AddAuthorization();

builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(c =>
{
    c.SwaggerDoc("v1", new() { Title = "MiniBank.Api", Version = "v1" });

    // Add definition of JWT
    c.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme
    {
        Name = "Authorization",
        Type = Microsoft.OpenApi.Models.SecuritySchemeType.Http,
        Scheme = "bearer",
        BearerFormat = "JWT",
        In = Microsoft.OpenApi.Models.ParameterLocation.Header,
        Description = "Enter 'Bearer' followed by the token. Example: Bearer eyJhbGciOi..."
    });

    c.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement
    {
        {
            new Microsoft.OpenApi.Models.OpenApiSecurityScheme
            {
                Reference = new Microsoft.OpenApi.Models.OpenApiReference
                {
                    Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,
                    Id = "Bearer"
                }
            },
            new string[] {}
        }
    });
});
```

1.2 Entities

The **Entities** defined the domain models for Users, Accounts, and Transactions.





1.3 Database Context

The application used **Entity Framework Core with SQLite** for persistence. SQLite was selected for its portability and simplicity, suitable for development and testing.
The schema included:

- **Users** table: stored email and password (plain text for demonstration).
- **Accounts** table: linked to users, containing balances.
- **Transactions** table: recorded deposits, withdrawals, and transfers.

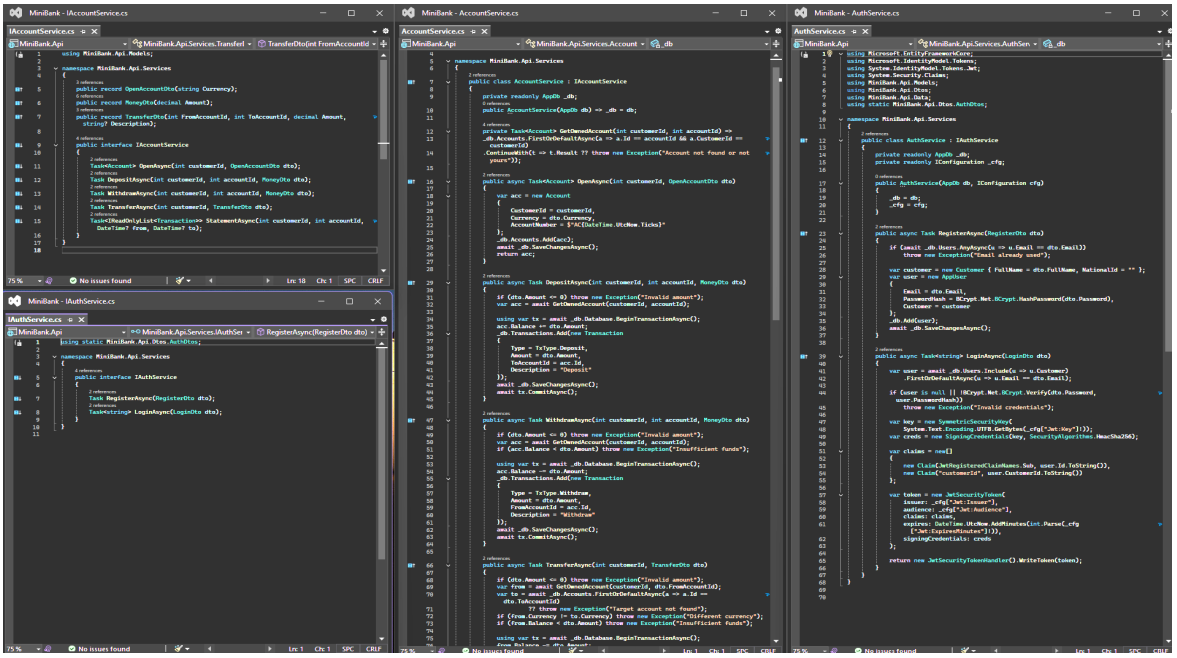
```
using Microsoft.EntityFrameworkCore;
using MiniBank.Api.Models;

namespace MiniBank.Api.Data
{
    public class AppDb : DbContext
    {
        0 references
        public AppDb(DbContextOptions<AppDb> options) : base(options) { }

        2 references
        public DbSet<AppUser> Users => Set<AppUser>();
        0 references
        public DbSet<Customer> Customers => Set<Customer>();
        4 references
        public DbSet<Account> Accounts => Set<Account>();
        4 references
        public DbSet<Transaction> Transactions => Set<Transaction>();

        0 references
        protected override void OnModelCreating(ModelBuilder b)
        {
            b.Entity<Account>().HasIndex(a => a.AccountNumber).IsUnique();
            b.Entity<AppUser>().HasIndex(u => u.Email).IsUnique();
            base.OnModelCreating(b);
        }

    }
}
```

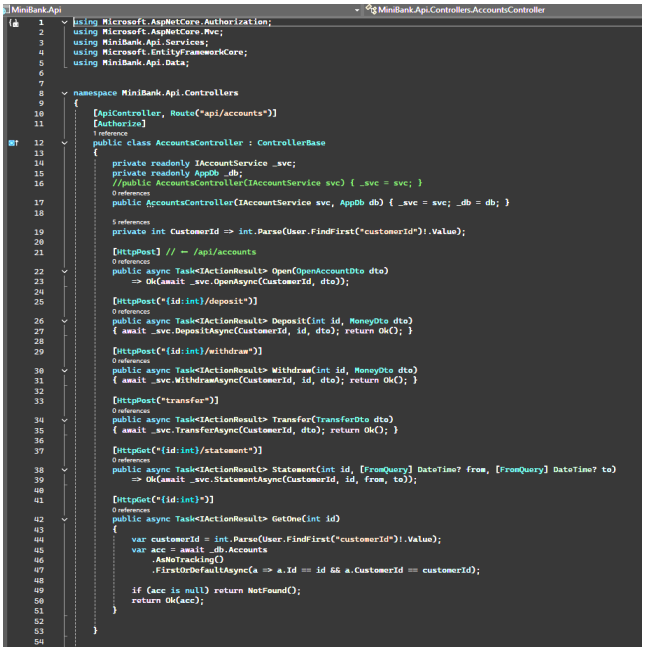
1.4 Services and Controllers

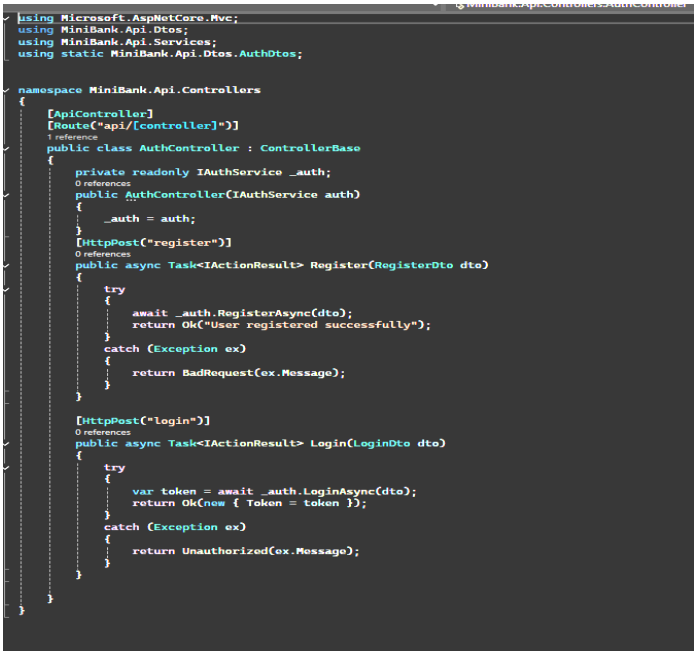
Business logic was implemented in services, while REST endpoints were exposed through controllers.

Endpoints included:

- `POST /api/Auth/login`
- `POST /api/Accounts/open`
- `POST /api/Accounts/deposit`
- `POST /api/Accounts/withdraw`
- `POST /api/Accounts/transfer`
- `GET /api/Accounts/statement`



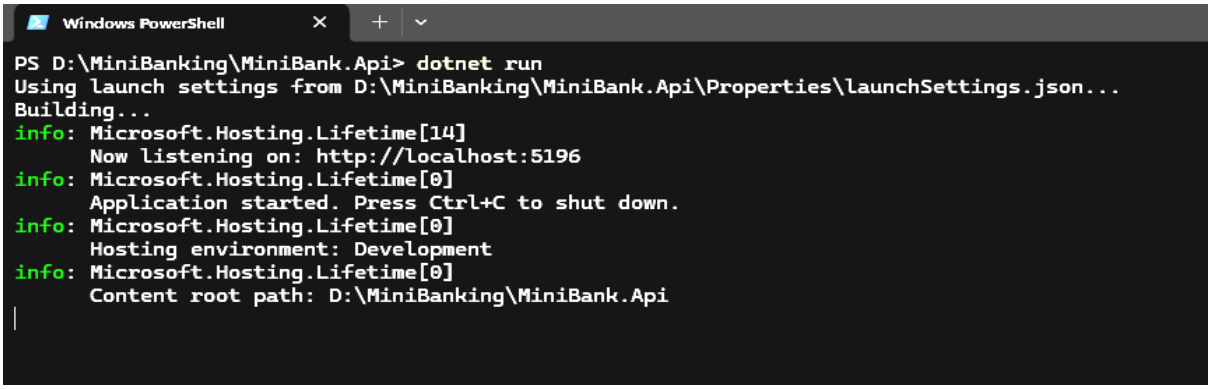


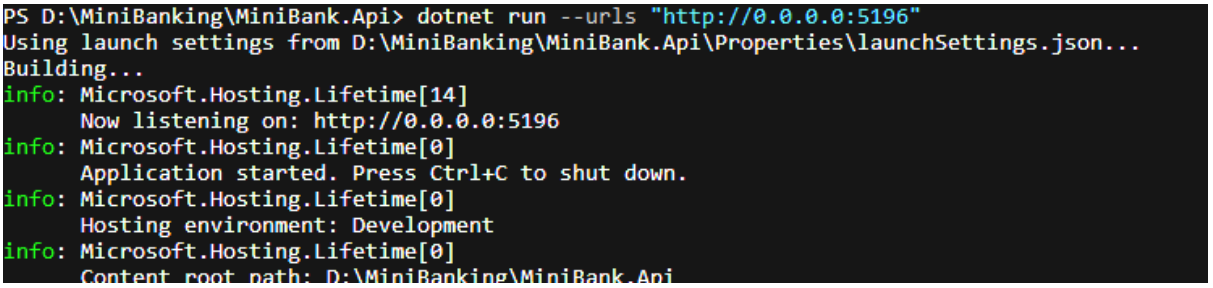


2. Running the Application

The application was started using:

```
dotnet run --urls "http://0.0.0.0:5196"
```

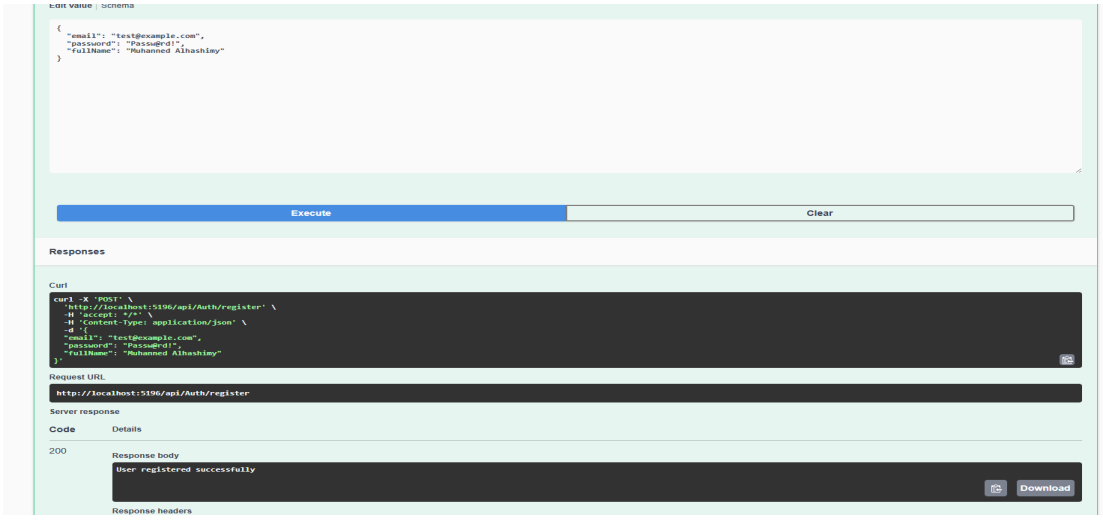


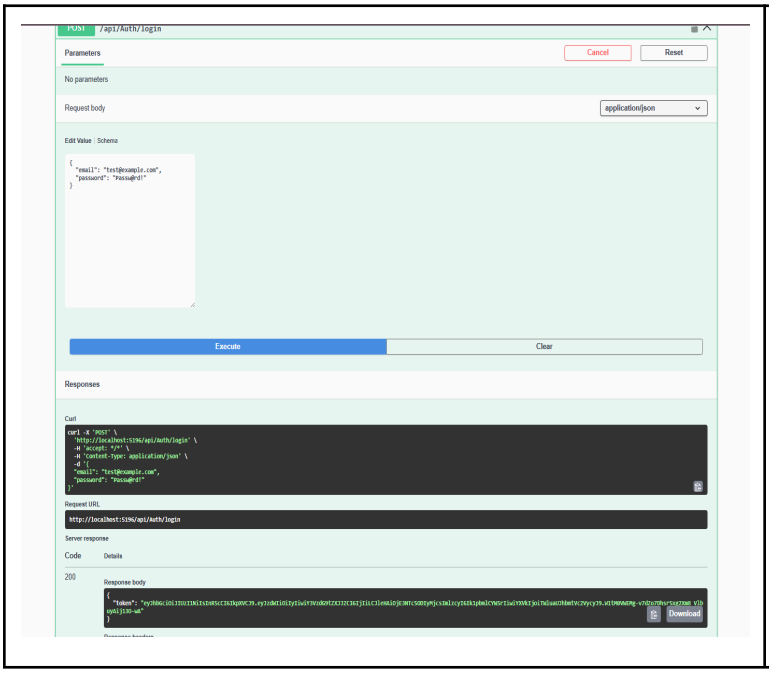


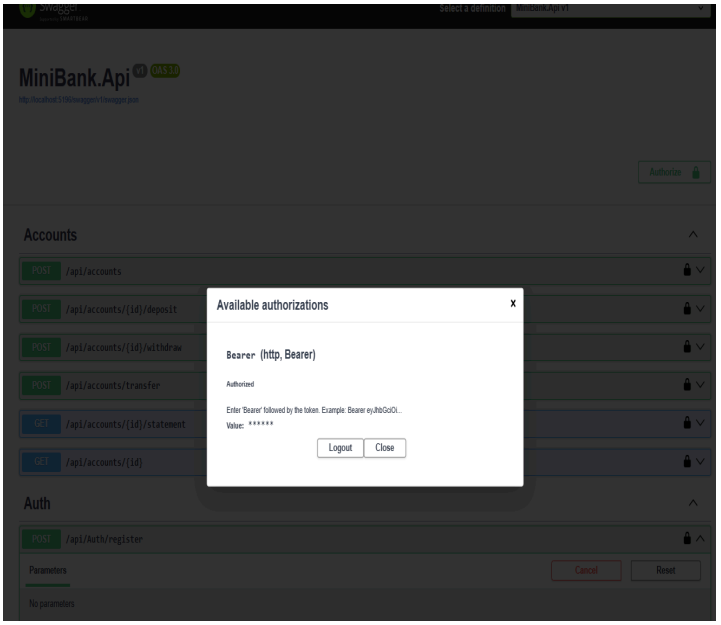
3. API Testing with Swagger

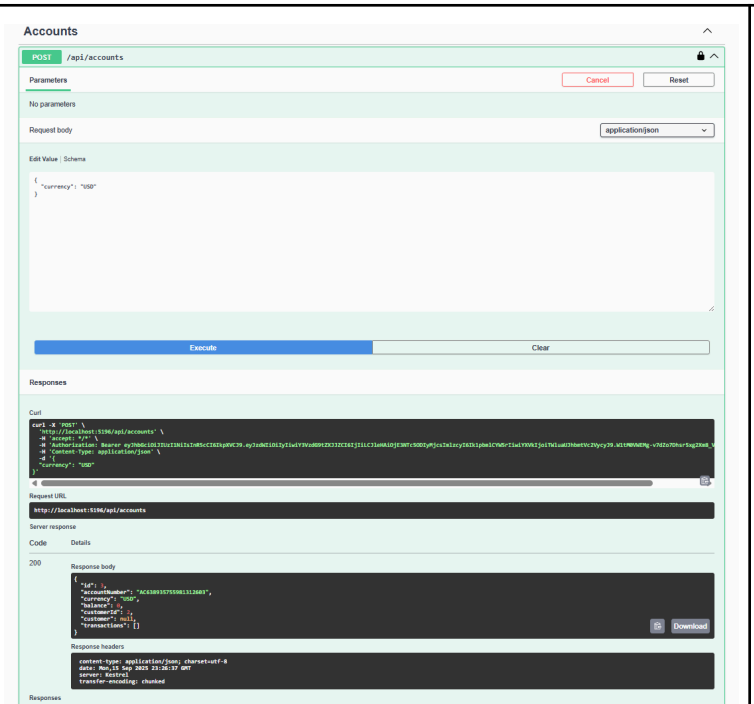
Swagger was used to verify functionality of endpoints:

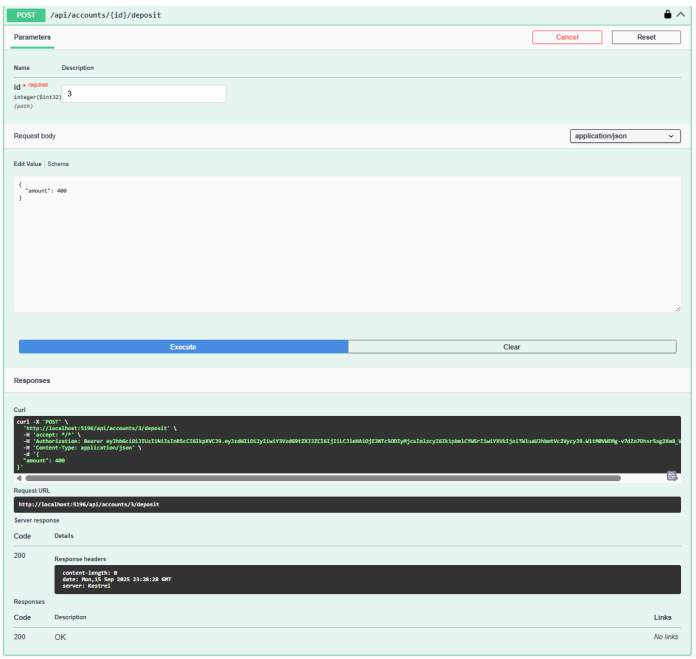
- Registration
- Login
- Authorization
- Account operations (Open, Deposit, Withdraw, Transfer, Statement)

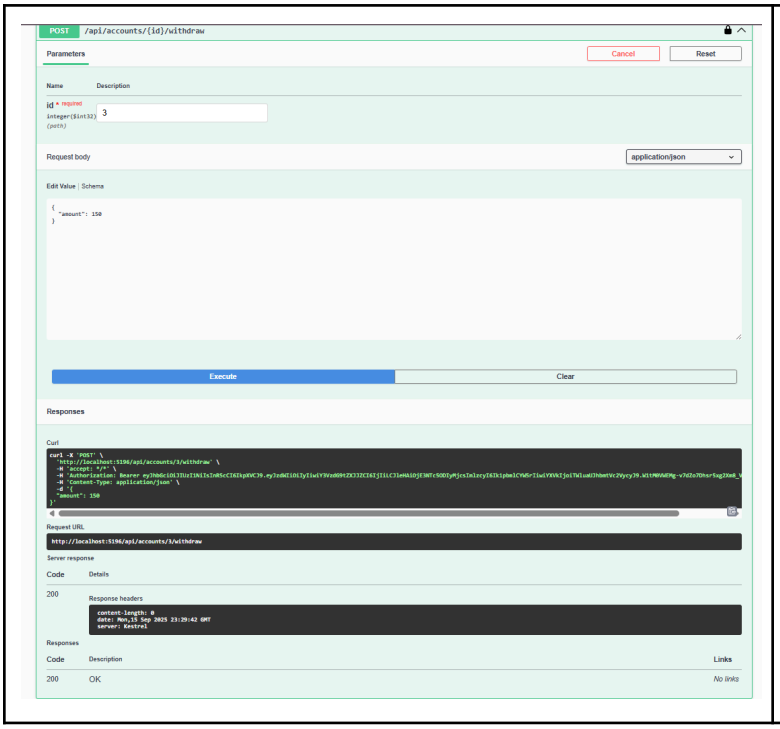


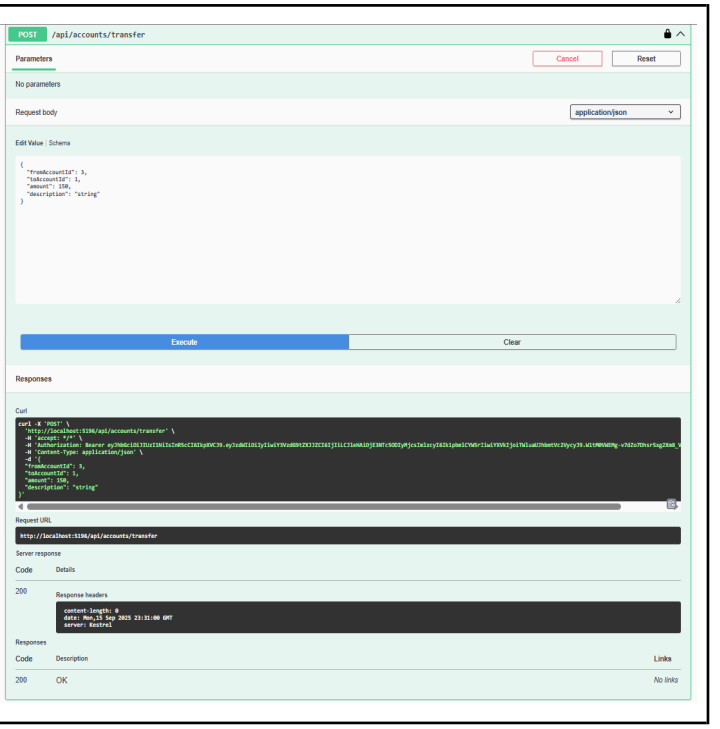

POST /api/Auth/login
Parameters
Cancel
Reset
No parameters
Request body
application/json
Edit Value | Schema
{
"email": "test@example.com",
"password": "Passw@rd!"
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5196/api/Auth/login' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"email": "test@example.com",
"password": "Passw@rd!"
}'
Request URL
http://localhost:5196/api/Auth/login
Server response
Code
Details
200
Response body
Download

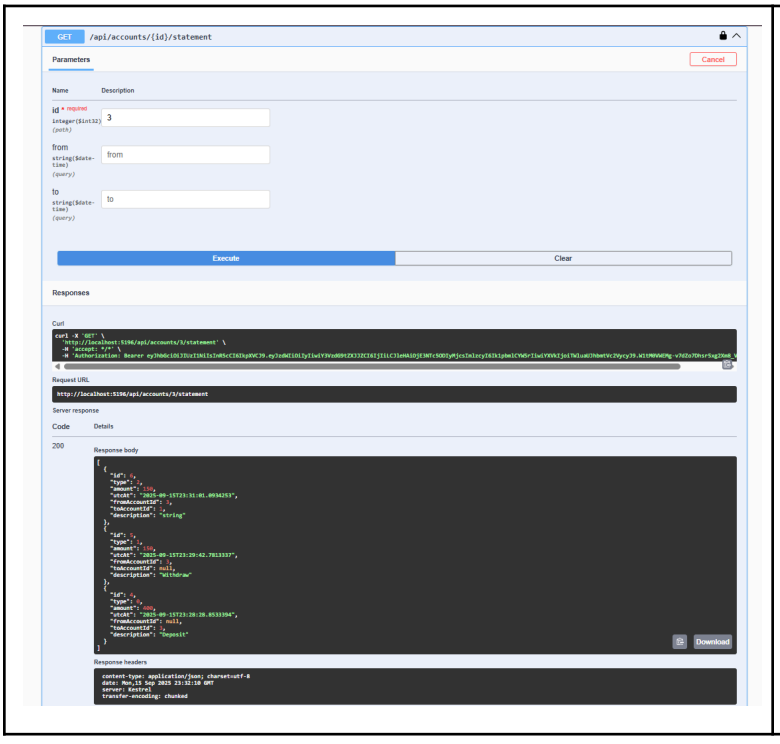

MiniBank.Api
Authorize
Accounts
POST /api/accounts
POST /api/accounts/{id}/deposit
POST /api/accounts/{id}/withdraw
POST /api/accounts/transfer
GET /api/accounts/{id}/statement
GET /api/accounts/{id}
Available authorizations
Bearer (http, Bearer)
Authorized
Enter 'Bearer' followed by the token. Example: Bearer eyJhbGciOi...
Value: ******
Logout
Close
Auth
POST /api/Auth/register
Parameters
Cancel
Reset
No parameters

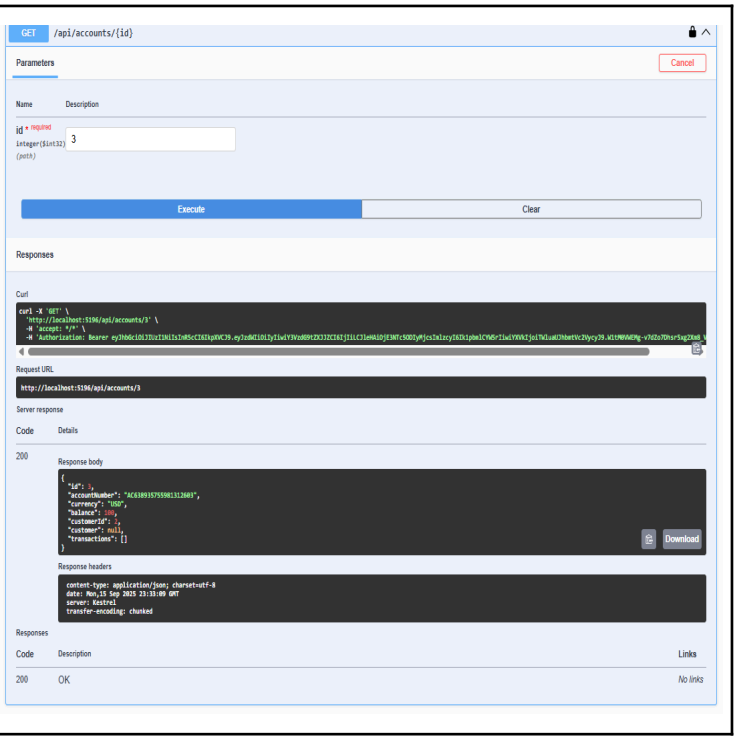

Accounts
POST /api/accounts
Parameters
Cancel
Reset
No parameters
Request body
application/json
Edit Value | Schema
{
"currency": "USD"
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5196/api/accounts' \
-H 'accept: */*' \
-H 'Authorization: Bearer
-H 'Content-Type: application/json' \
-d '{
"currency": "USD"
}'
Request URL
http://localhost:5196/api/accounts
Server response
Code
Details
200
Response body
{
"id": 3,
"accountNumber": "AC6389357550013126B3",
"currency": "USD",
"balance": 0,
"customerId": 2,
"customer": null,
"transactions": []
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Mon,15 Sep 2025 23:26:37 GMT
server: Kestrel
transfer-encoding: chunked
Responses


POST /api/accounts/{id}/deposit
Parameters
Cancel
Reset
Name
Description
id * required
integer($int32)
(path)
3
Request body
application/json
Edit Value | Schema
{
"amount": 400
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5196/api/accounts/3/deposit' \
-H 'accept: */*' \
-H 'Authorization: Bearer
-H 'Content-Type: application/json' \
-d '{
"amount": 400
}'
Request URL
http://localhost:5196/api/accounts/3/deposit
Server response
Code
Details
200
Response headers
content-length: 0
date: Mon,15 Sep 2025 23:38:28 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links


POST /api/accounts/{id}/withdraw
Parameters
Cancel
Reset
Name
Description
id * required
integer($int32)
(path)
3
Request body
application/json
Edit Value | Schema
{
"amount": 150
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5196/api/accounts/3/withdraw' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIiwiYXVkaG9tZXJ2ZCI6IjIiLC3leHAiOjE3NTc5ODIyMjcsImlzcyI6IkipbmlCYW5rIiwiYXVkIjoiTWluaUJhbmtVc2VycyJ9.W1tMNVWdMg-v7dZo7Dhsr5ag2Xm8_U' \
-H 'Content-Type: application/json' \
-d '{
"amount": 150
}'
Request URL
http://localhost:5196/api/accounts/3/withdraw
Server response
Code
Details
200
Response headers
content-length: 0
date: Mon,15 Sep 2025 23:29:42 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links


POST /api/accounts/transfer
Parameters
Cancel
Reset
No parameters
Request body
application/json
Edit Value | Schema
{
"fromAccountId": 3,
"toAccountId": 1,
"amount": 150,
"description": "string"
}
Execute
Clear
Responses
Curl
curl -X 'POST' \
'http://localhost:5196/api/accounts/transfer' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIiwiYXVkaG9tZXJ2ZCI6IjIiLC3leHAiOjE3NTc5ODIyMjcsImlzcyI6IkipbmlCYW5rIiwiYXVkIjoiTWluaUJhbmtVc2VycyJ9.W1tMNVWdMg-v7dZo7Dhsr5ag2Xm8_U' \
-H 'Content-Type: application/json' \
-d '{
"fromAccountId": 3,
"toAccountId": 1,
"amount": 150,
"description": "string"
}'
Request URL
http://localhost:5196/api/accounts/transfer
Server response
Code
Details
200
Response headers
content-length: 0
date: Mon,15 Sep 2025 23:31:00 GMT
server: Kestrel
Responses
Code
Description
Links
200
OK
No links

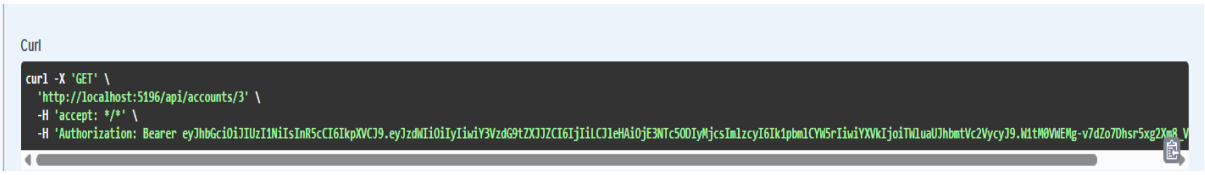

GET /api/accounts/{id}/statement
Parameters
Cancel
Name
Description
id * required
integer($int32)
(path)
3
from
string($date-time)
(query)
from
to
string($date-time)
(query)
to
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:5196/api/accounts/3/statement' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIiwiYXVkaG9tZXJ2ZCI6IjIiLC3leHAiOjE3NTc5ODIyMjcsImlzcyI6IkipbmlCYW5rIiwiYXVkIjoiTWluaUJhbmtVc2VycyJ9.W1tMNVWdMg-v7dZo7Dhsr5ag2Xm8_U'
Request URL
http://localhost:5196/api/accounts/3/statement
Server response
Code
Details
200
Response body
Download
Response headers
content-type: application/json; charset=utf-8
date: Mon,15 Sep 2025 23:32:18 GMT
server: Kestrel
transfer-encoding: chunked


GET /api/accounts/{id}
Parameters
Cancel
Name
Description
id * required
integer($int32)
(path)
3
Execute
Clear
Responses
Curl
curl -X 'GET' \
'http://localhost:5196/api/accounts/3' \
-H 'accept: */*' \
-H 'Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIyIiwiYXVkaG9tZXJ2ZCI6IjIiLC3leHAiOjE3NTc5ODIyMjcsImlzcyI6IkipbmlCYW5rIiwiYXVkIjoiTWluaUJhbmtVc2VycyJ9.W1tMNVWdMg-v7dZo7Dhsr5ag2Xm8_U'
Request URL
http://localhost:5196/api/accounts/3
Server response
Code
Details
200
Response body
Download
Response headers
content-type: application/json; charset=utf-8
date: Mon,15 Sep 2025 23:33:09 GMT
server: Kestrel
transfer-encoding: chunked
Responses
Code
Description
Links
200
OK
No links

4. Connectivity Test (Attacker → Server)

From Kali Linux, connectivity was verified with curl:

`curl http://192.168.1.108:5196/swagger`

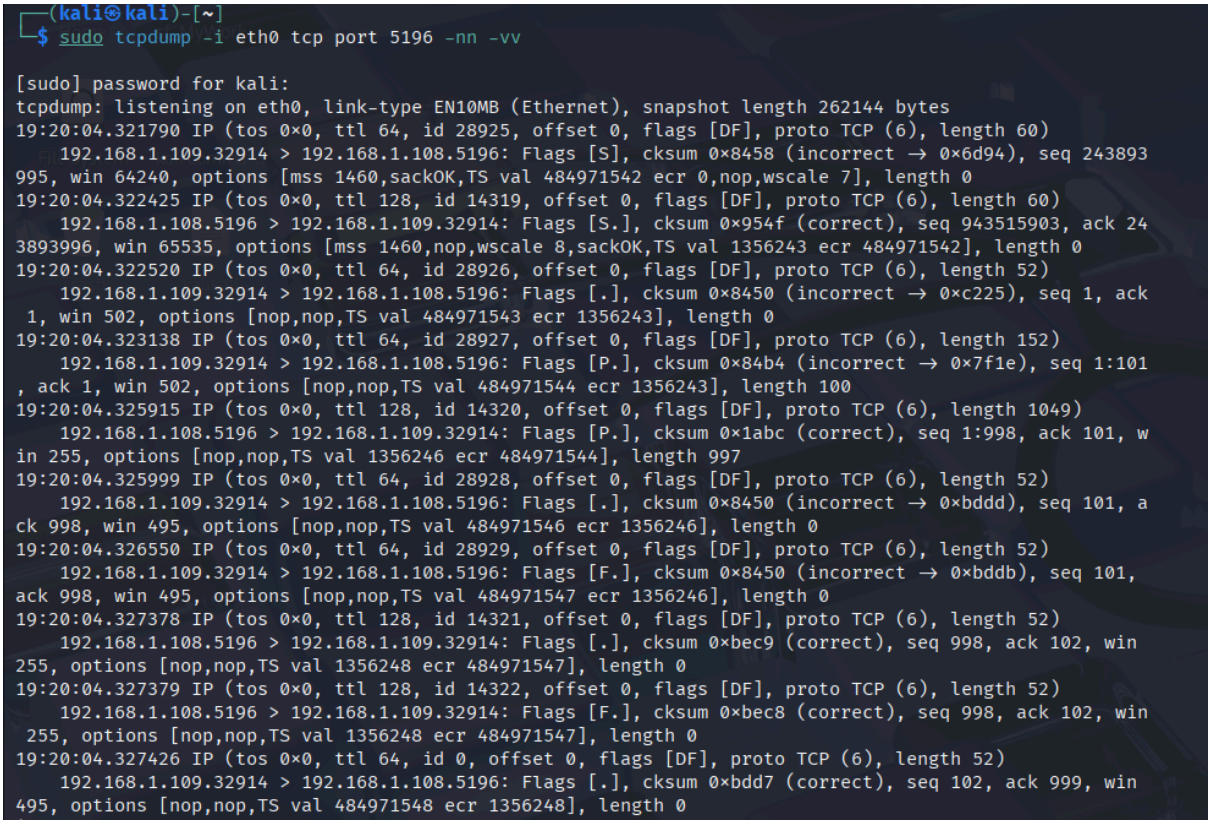



```
┌──(kali㉿kali)-[~]
└─$ curl -v http://192.168.1.108:5196/swagger/index.html

*   Trying 192.168.1.108:5196 ...
* Connected to 192.168.1.108 (192.168.1.108) port 5196
* using HTTP/1.x
> GET /swagger/index.html HTTP/1.1
> Host: 192.168.1.108:5196
> User-Agent: curl/8.14.1
> Accept: */*
>
* Request completely sent off
< HTTP/1.1 200 OK
< Content-Type: text/html;charset=utf-8
< Date: Wed, 17 Sep 2025 23:21:44 GMT
< Server: Kestrel
< Cache-Control: max-age=604800, private
< ETag: W/"XpksuI7SK4ZUIzv9+JEi7bNIY14="
< Transfer-Encoding: chunked
< x-swagger-ui-version: 5.27.1
<
<!-- HTML for static distribution bundle build -->
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Swagger UI</title>
    <link rel="stylesheet" type="text/css" href="./swagger-ui.css">
    <link rel="stylesheet" type="text/css" href="./index.css">
    <link rel="icon" type="image/png" href="./favicon-32x32.png" sizes="32x32" />
    <link rel="icon" type="image/png" href="./favicon-16x16.png" sizes="16x16" />

</head>

<body>
    <div id="swagger-ui"></div>

    <script src="./swagger-ui-bundle.js" charset="utf-8"></script>
    <script src="./swagger-ui-standalone-preset.js" charset="utf-8"></script>
    <script src="index.js" charset="utf-8"></script>
</body>
</html>
* Connection #0 to host 192.168.1.108 left intact
```

5. Packet Capture

Using tcpdump, traffic between the attacker (192.168.1.109) and the server (192.168.1.108) was captured, showing the TCP handshake and HTTP requests.

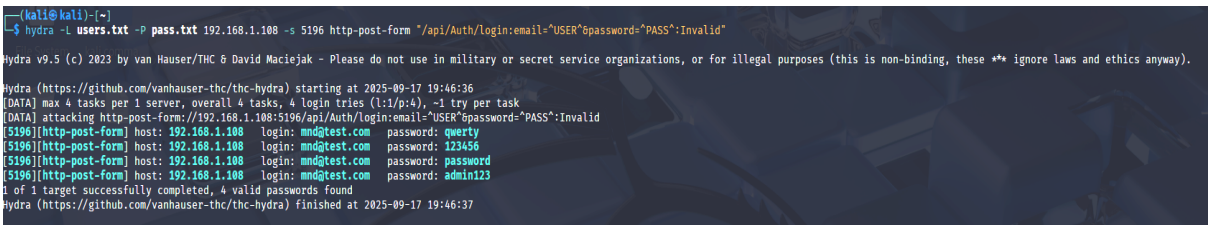
```
sudo tcpdump -i eth0 tcp port 5196 -w capture.pcap
```



6. Brute-Force Attack with Hydra

Hydra was executed against the login endpoint `/api/Auth/login`:

```
hydra -L users.txt -P pass.txt 192.168.1.108 -s 5196 \
      http-post-form
"/api/Auth/login:email=^USER^&password=^PASS^:Invalid"
```



Hydra successfully identified weak passwords:

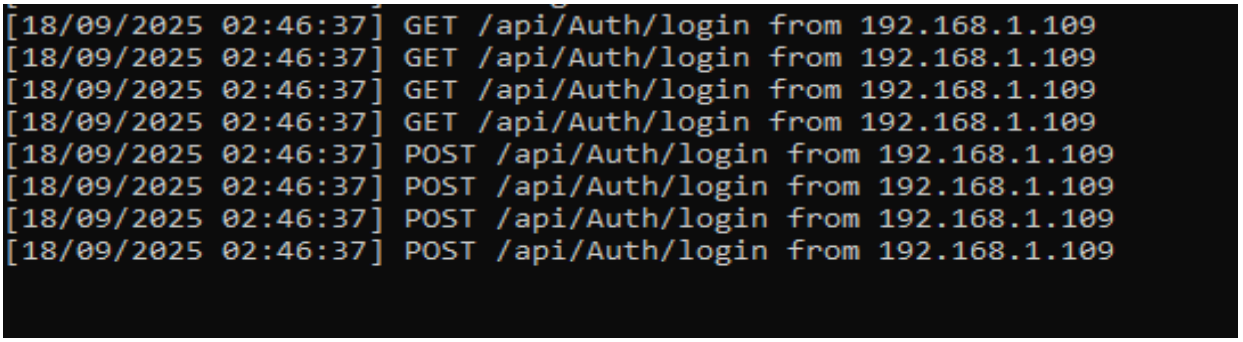
qwerty	123456	password	admin123

---

7. Server Logs

The application logs confirmed multiple requests originating from the attacker's IP (192.168.1.109):

```
[18/09/2025 02:46:37] POST /api/Auth/login from 192.168.1.109
```



---

8. Security Analysis

Weaknesses Identified

1. Weak/default passwords accepted.
2. Passwords stored in plain text.
3. No rate limiting or account lockouts.
4. No multi-factor authentication (MFA).
5. Logs exist but no intrusion alerts.

---

9. Recommendations

- Enforce **strong password policy** (minimum length, complexity).
- Store passwords with **secure hashing algorithms** (bcrypt, Argon2).
- Implement **account lockouts and rate limiting**.
- Add **multi-factor authentication (MFA)**.
- Integrate with **SIEM/IDS tools** to alert on brute-force attempts.
- Conduct **regular penetration tests** to identify vulnerabilities.

---

10. Conclusion

This project demonstrated how a vulnerable banking API built with .NET 9 and SQLite can be exploited through brute-force attacks. The exercise highlights the importance of secure authentication and layered defenses. Implementing strong password policies, rate limiting, hashing, and MFA are critical to preventing such attacks.