Security research

TwitterV2

Daniil Blagoev

Contents

[Introduction 2](#_Toc169513516)

[Research questions & methods 2](#_Toc169513517)

[Answers 2](#_Toc169513518)

[What is the current OWASP top 10 list? 2](#_Toc169513519)

[Which threats from the OWASP top 10 list is TwitterV2 vulnerable to? 3](#_Toc169513520)

[How can I secure TwitterV2 against the threats it is vulnerable to? 4](#_Toc169513521)

[Conclusion 7](#_Toc169513522)

[Sources 8](#_Toc169513523)

# Introduction

The purpose of this research is to find out what the latest OWASP top 10 list is, and to make sure that TwitterV2 has the proper security against the threats on the list.

# Research questions & methods

Main research question:

* What can I do to improve TwitterV2’s security?

Sub-questions:

* What is the current OWASP top 10 list?
  + Literature study
    - Googling to find out the latest top 10 list.
* Which threats from the OWASP top 10 list is TwitterV2 vulnerable to?
  + Available product analysis
    - Analyze if TwitterV2 has protective measures against each relevant threat.
* How can I secure TwitterV2 against the threats it is vulnerable to?
  + Literature study
    - Find out how to implement protective measures against threats which TwitterV2 is vulnerable to and implement them.
  + Security test
    - Test if the implementation of protective measures makes TwitterV2 secure against the OWASP top 10.

# Answers

## What is the current OWASP top 10 list?

The latest OWASP top 10 list as of the time I am writing this (24/05/2024) is the following:

1. **Broken Access Control** - Access control enforces policy such that users cannot act outside of their intended permissions. Failures typically lead to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits. Common access control vulnerabilities include violation of the principle of least privilege or deny by default, where access should only be granted for particular capabilities, roles, or users, but is available to anyone; bypassing access control checks by modifying the URL.
2. **Cryptographic Failures** - The first thing is to determine the protection needs of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business secrets require extra protection, mainly if that data falls under privacy laws, e.g., EU's General Data Protection Regulation (GDPR), or regulations, e.g., financial data protection such as PCI Data Security Standard (PCI DSS).
3. **Injection** – An application is vulnerable to this attack when user supplied data is not validated, filtered, or sanitized; hostile data is used within object-relational mapping search parameters to extract additional, sensitive records; hostile data is directly used or concatenated, the SQL or command contains the structure and malicious data in dynamic queries, commands, or stored procedures.
4. **Insecure Design** - One of the factors that contribute to insecure design is the lack of business risk profiling inherent in the software or system being developed, and thus the failure to determine what level of security design is required.
5. **Security Misconfiguration** – The application might be vulnerable if the application has unnecessary features enabled or installed (e.g. unnecessary ports, services, pages, accounts); default accounts and their passwords enabled; error handling which reveals stack traces.
6. **Vulnerable and Outdated Components** – Underlying platforms, frameworks, dependencies, components should be kept up to date and unused ones should be removed.
7. **Identification and Authorization Failures** - Confirmation of the user's identity, authentication, and session management is critical to protect against authentication-related attacks. There may be authentication weaknesses if the application permits brute force or other automated attacks; uses plain text or weakly hashed password data stores; permits automated attacks such as credential stuffing, where the attacker has a list of valid usernames and passwords.
8. **Software and Data Integrity Failures** - Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks (CDNs). An insecure CI/CD pipeline can introduce the potential for unauthorized access, malicious code, or system compromise.
9. **Security Logging and Monitoring Failures** - Returning to the OWASP Top 10 2021, this category is to help detect, escalate, and respond to active breaches. Without logging and monitoring, breaches cannot be detected. Insufficient logging, detection, monitoring, and active response occurs any time warnings and errors generate no, inadequate or unclear log messages; logins, failed logins and high-value transactions are not logged; logs are only stored locally.
10. **Server-Side Request Forgery (SSRF)** - SSRF flaws occur whenever a web application is fetching a remote resource without validating the user-supplied URL. It allows an attacker to coerce the application to send a crafted request to an unexpected destination, even when protected by a firewall, VPN, or another type of network access control list (ACL).[[1]](#footnote-1)

## Which threats from the OWASP top 10 list is TwitterV2 vulnerable to?

1. Broken Access Control – TwitterV2 has authorization & authentication implemented. The frontend needs to be protected against bad actors trying to browse to admin URLs/URLs which would usually require authorization.
2. Cryptographic Failures – TwitterV2 has sufficient data security at rest and during transit. Sensitive data is encrypted and stored accordingly to the GDPR guidelines in Azure’s CosmosDB. Additionally, the data sent around with Kafka is encrypted.
3. Injection – TwitterV2’s users will be communicating with the web application through a React frontend. React sanitizes user input by default and everything is turned into a string before being rendered. Additionally, MongoDB uses parameterized queries, which are safe against SQL injection attacks.
4. Insecure Design – TwitterV2 has unhappy flow tests and expected failure flows are considered.
5. Security Misconfiguration – TwitterV2 does not have unnecessary ports and services. There are no test accounts or pages deployed.
6. Vulnerable and Outdated Components – TwitterV2 currently does not have any outdated components. All frameworks, dependencies and packages are up to date.
7. Identification and Authorization Failures – TwitterV2 is currently not protected against brute force attacks. Credentials are hashed and salted and stored in Azure’s CosmosDB.
8. Software and Data Integrity Failures – TwitterV2’s CI/CD does not use any plugins, libraries, or modules from untrusted sources. It was developed using only widely popular and presumably safe technologies.
9. Security Logging and Monitoring Failures – TwitterV2 uses Azure’s Grafana for logging and monitoring. All logs are stored on the cloud.
10. Server-Side Request Forgery – TwitterV2 does not fetch remote resources at any point during its current workflow.

TwitterV2’s security is vulnerable to threats 1 and 7.

## How can I secure TwitterV2 against the threats it is vulnerable to?

#### Implementing protected routes

/1/ The way to protect TwitterV2 from unauthorized browsing was to implement protected routes. What protected routes do is check whether the person who requested to visit the URL has the needed role. Whenever the user logs in, their role gets saved in the frontend’s local storage. This is where the protected route component checks it from. If the role is not sufficient for what they are trying to access, they are redirected to the login screen.

A screenshot of a computer program

Description automatically generated

Figure 1 User Protected Routes component

A screen shot of a computer code

Description automatically generated

Figure 2 Making use of the User Protected Routes component

#### Implementing rate limiting

/7/ One way to deal with brute force is to limit the number of requests a single client can send. TwitterV2 uses an Ocelot gateway as a middleman between the frontend and the services, so it would make most sense to implement the rate limiter there. Ocelot has a built-in rate limiter, so what I had to do was to enable it and configure it:[[2]](#footnote-2)

A screen shot of a computer program

Description automatically generated

Figure 3 Rate limiter configuration

#### Protected routes

[navigate without role.mp4](navigate%20without%20role.mp4)

#### Rate limiting

A screenshot of a computer

Description automatically generated

Figure 4 Error message when sending too many login requests

# Conclusion

After analyzing today’s biggest threats to web applications and analyzing TwitterV2’s security, I was able to improve it by [preventing brute force attacks](#_Implementing_rate_limiting) and [implementing protected routes](#_Implementing_protected_routes).

# Sources

*OWASP Top Ten | OWASP Foundation*. (n.d.). https://owasp.org/www-project-top-ten/

Wong, C., & Wong, C. (n.d.). *Part Four - Building API Gateway Using Ocelot In ASP.NET Core - Rate Limiting*. https://www.c-sharpcorner.com/article/building-api-gateway-using-ocelot-in-asp-net-core-rate-limiting-part-four/

1. *OWASP Top Ten | OWASP Foundation*. (n.d.). https://owasp.org/www-project-top-ten/ [↑](#footnote-ref-1)
2. Wong, C., & Wong, C. (n.d.). *Part Four - Building API Gateway Using Ocelot In ASP.NET Core - Rate Limiting*. https://www.c-sharpcorner.com/article/building-api-gateway-using-ocelot-in-asp-net-core-rate-limiting-part-four/ [↑](#footnote-ref-2)