Software vulnerability report

Version 0.1

june15th, 2021

Document history

Versions

Version	Date	Author(s)	Changes	Status
0.1	03-03-2021	Kevin Heijboer	OWASP checklist	Concept

Table of Contents

Document history 2

Versions 2

1 OWASP vulnerabilities 6

1.1 AppDOS 6

Application Flooding 6

Application Lockout 6

1.2 AccessControl 7

Parameter Analysis 7

Authorization 7

Authorization Parameter Manipulation 7

Authorized pages/functions 8

Application Workflow 8

1.3 Authentication 9

Authentication endpoint request should be HTTPS 9

Authentication bypass 9

1.4 Authentication.User 10

Credentials transport over an encrypted channel 10

Default Accounts 10

Username 10

Password Quality 11

Password Reset 11

Password Lockout 11

Password Structure 12

Blank Passwords 12

1.5 Authentication.SessionManagement 13

Session Token Length 13

Session Timeout 13

Session Reuse 13

Session Deletion 13

Session Token Format 14

1.6 Configuration.Management 14

HTTP Methods 14

Virtually Hosted Sites 14

Known Vulnerabilities / Security Patches 15

Back-up Files 15

Web Server Configuration 15

Web Server Components 15

Common Paths 16

Language/Application defaults 16

1.7 Configuration.Management.Application 16

Infrastructure Admin Interfaces 16

1.8 Error Handling 17

Application Error Messages 17

User Error Messages 17

1.9 DataProtection 18

Sensitive Data in HTML 18

Data Storage 18

1.10 DataProtection.Transport 18

SSL Version 18

SSL Key Exchange Methods 18

SSL Algorithms 19

SSL Key Lengths 19

Digital Certificate Validity 20

1.11 InputValidation 20

Script Injection 20

1.12 InputValidation.SQL 21

SQL Injection 21

1.13 InputValidation.OS 21

OS Command Injection 21

1.14 InputValidation.LDAP 22

LDAP Injection 22

1.15 InputValidation.XSS 22

Cross Site Scripting 22

1.16 BufferOverflow 23

Overflows 23

Heap Overflows 23
Stack Overflows 23
Format Strings 23
2 OWASP checklist 24

1 OWASP vulnerabilities

1.1 AppDOS

Application Flooding

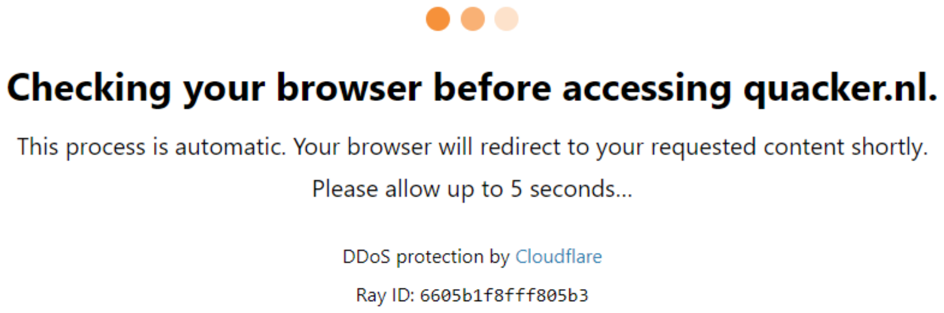
Objective

Ensure that the application functions correctly when presented with large volumes of requests, transactions and / or network traffic.

Notes: Use various fuzzing tools to perform this test (e.g. SPIKE)

Status

Quacker is deployed with both vertical and horizontal autoscaling to handle large amounts of requests and traffic. Autoscalers have limits and do not prevent DDoS attacks. Cloudflare is used as a proxy/CDN on top of the regular DigitalOcean server. Cloudflare has options for DDoS protection such as whitelisting/blacklisting IP addresses and a "I'm under attack" option. This will prompt the following message when entering Quacker.



Checking your browser before accessing quacker.nl.

This process is automatic. Your browser will redirect to your requested content shortly.

Please allow up to 5 seconds…

DDoS protection by Cloudflare

Ray ID: 6605b1f8fff805b3

Figure 1. Example of Cloudflare's DDoS protection

Application Lockout

Objective

Ensure that the application does not allow an attacker to reset or lockout user's accounts.

Status

This vulnerability is not applicable to Quacker as there is no lockout mechanism implemented. If implemented, brute force attacks are easy to conduct. The lockout should automatically unlock after 10-15 minutes so that users are not bothered as much.

1.2 AccessControl

Parameter Analysis

Objective

Ensure that the application enforces its access control model by ensuring that any parameters available to an attacker would not afford additional service.

Notes: Typically this includes manipulation of form fields, URL query strings, client-side script values and cookies.

Status

The way Quacker is built, all data is stored and retrieved from the back end. Invalid data is also validated and checked for existence in the back end. The front end merely serves as a template to show the data. Editing URL parameters or HTML/JavaScript values will not affect any data or backend processes.

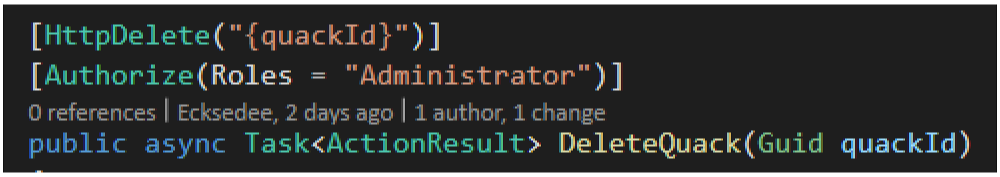
Authorization

Objective

Ensure that resources that require authorization perform adequate authorization checks before being sent to a user.

Status

Quacker has role-based authorization. Every resource has an authorization check that will validate the users, JWT and validate if the user has sufficient permissions.



```
[HttpDelete("{quackId}")]
[Authorize(Roles = "Administrator")]
0 references | Ecksedee, 2 days ago | 1 author, 1 change
public async Task<ActionResult> DeleteQuack(Guid quackId)
```

Figure 2. Example of back-end authorization check

Authorization Parameter Manipulation

Objective

Ensure that once valid user has logged in it is not possible to change the session ID’s parameter to reflect another user account

Notes: i.e. accountnumber, policynumber,usernr etc.

Status

All user data such as IDs are stored in a JWT format. Users will need to know the JWT secret which is stored in the back end before they can manipulate the JWT.

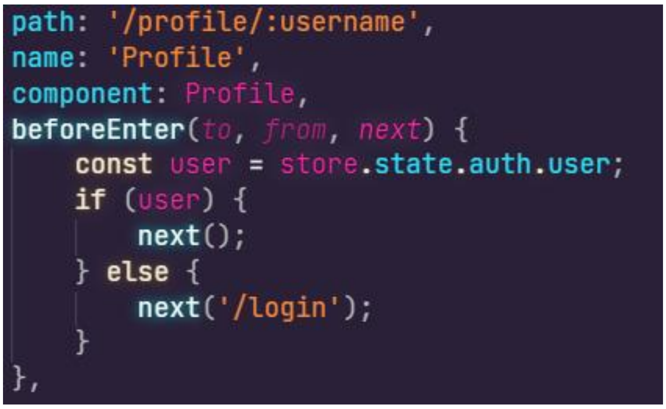
Authorized pages/functions

Objective

Check to see if it's possible to access pages or functions which require logon but can be bypassed

Status

Quacker has so called "route guards" on protected pages. These route guards check for the user's permissions which are stored in cookies. If the guard is somehow bypassed, the attack will not be able to see any vulnerable data as the front end only serves as a template.



```
path: '/profile/:username',
name: 'Profile',
component: Profile,
beforeEnter(to, from, next) {
    const user = store.state.auth.user;
    if (user) {
        next();
    } else {
        next('/login');
    }
},
```

Figure 3. Example of front-end route guard

Application Workflow

Objective

Ensure that where the application requires the user to perform actions in a specific sequence, the sequence is enforced.

Status

This vulnerability is not applicable. Quacker has no workflows that require a specific sequence of actions.

1.3 Authentication

Authentication endpoint request should be HTTPS

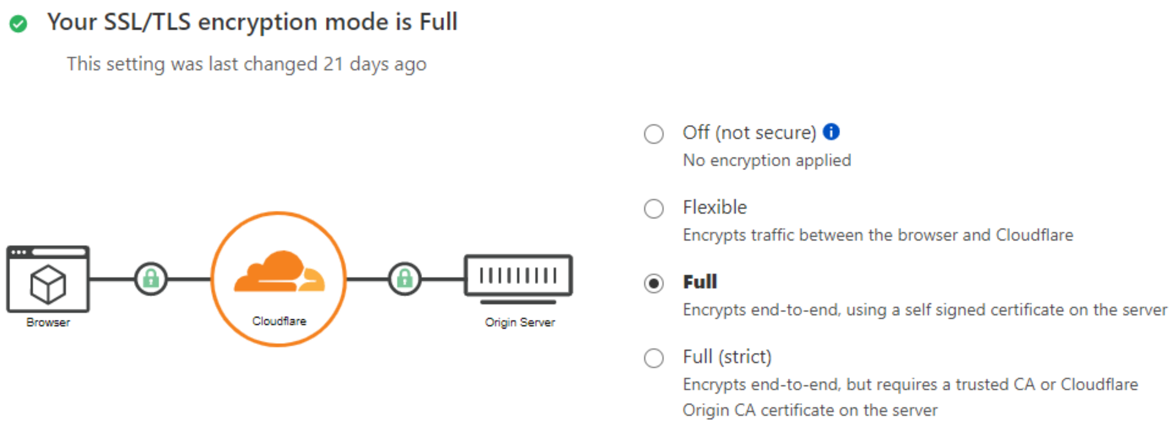
Objective

Ensure that users are only asked to submit authentication credentials on pages that are served with SSL.

Notes: This ensures that the user knows who is asking for his / her credentials as well as where they are being sent.

Status

Both the front-end website as the back-end API are provided with SSL encryption. The certificates are managed by Cloudflare.



✓ Your SSL/TLS encryption mode is Full

This setting was last changed 21 days ago

○ Off (not secure)
No encryption applied

○ Flexible
Encrypts traffic between the browser and Cloudflare

● **Full**
Encrypts end-to-end, using a self signed certificate on the server

○ Full (strict)
Encrypts end-to-end, but requires a trusted CA or Cloudflare Origin CA certificate on the server

Figure 4. Cloudflare's SSL settings

Authentication bypass

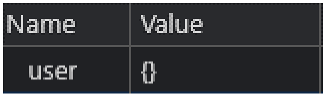
Objective

Ensure that the authentication process cannot be bypassed.

Notes: Typically this happens in conjunction with flaws like SQL Injection.

Status

After some testing, it is possible to view pages by adding an empty "user" cookie.



Name	Value
user	{}

Figure 5. Example of an empty user cookie

This cookie allows an attacker to bypass page guards and view pages even when they are not authenticated. There is no data present on the page as the back end authorization fails. A possible solution for this is redirecting users after a 401 unauthorized status is received from the back end.

1.4 Authentication.User

Credentials transport over an encrypted channel

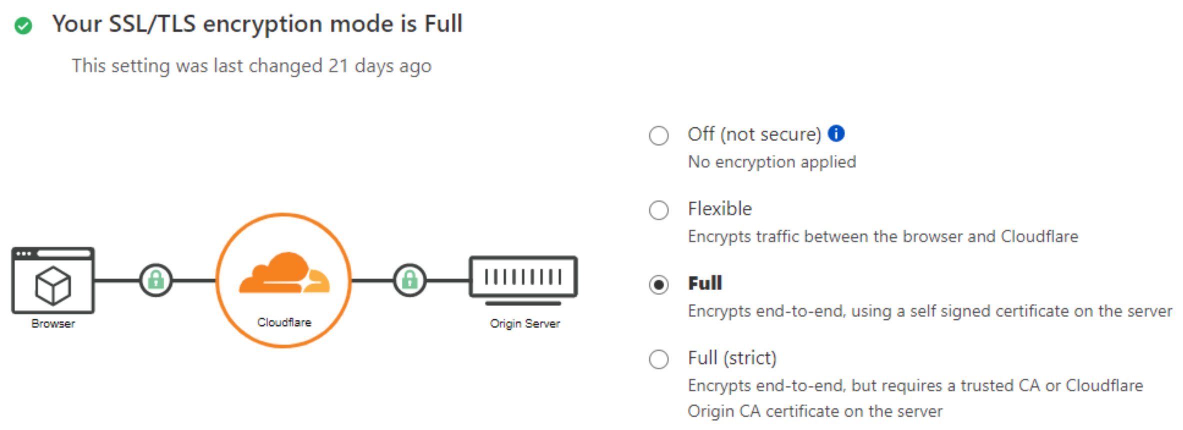
Objective

Ensure that usernames and passwords are sent over an encrypted channel.

Notes: Typically this should be SSL.

Status

Both the front-end website as the back-end API are provided with SSL encryption. The certificates are managed by Cloudflare.



✓ Your SSL/TLS encryption mode is Full

This setting was last changed 21 days ago

○ Off (not secure)
No encryption applied

○ Flexible
Encrypts traffic between the browser and Cloudflare

● **Full**
Encrypts end-to-end, using a self signed certificate on the server

○ Full (strict)
Encrypts end-to-end, but requires a trusted CA or Cloudflare Origin CA certificate on the server

Figure 6. Cloudflare's SSL settings

Default Accounts

Objective

Check for default account names and passwords in use.

Status

This is a high-risk vulnerability that is currently active in Quacker. On initialization, a default “admin” account is seeded. This admin account has an equally easy-to-guess password. A way to fix this vulnerability is to force the admin user to change the default credentials after the first log in.

Username

Objective

Ensure that the username is not public (or “wallet”) information such as email or SSN.

Status

A user has two ways to authenticate with Quacker. Their username and email. A user’s email is not public information and is never shown on their profile. Users also have a display name which they can change to show on their profile.

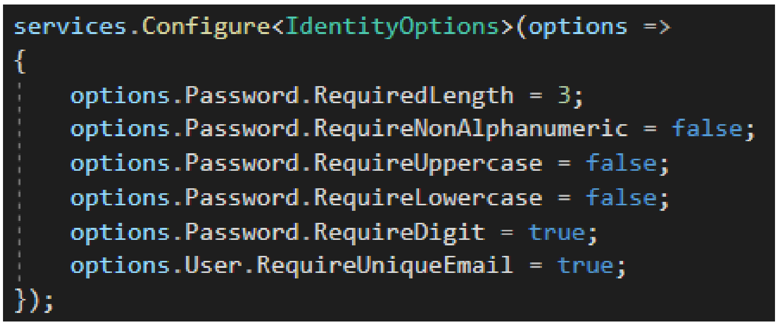
Password Quality

Objective

Ensure that the password complexity makes guessing passwords difficult.

Status

Quacker enforces password complexity. For development purposes, this is currently set to a simple ruleset. This ruleset can be changed in the authentication microservice.



```
services.Configure<IdentityOptions>(options =>
{
    options.Password.RequiredLength = 3;
    options.Password.RequireNonAlphanumeric = false;
    options.Password.RequireUppercase = false;
    options.Password.RequireLowercase = false;
    options.Password.RequireDigit = true;
    options.User.RequireUniqueEmail = true;
});
```

Figure 7. Password complexity for Quacker

Password Reset

Objective

Ensure that user must respond to a secret answer / secret question or other predetermined information before passwords can be reset.

Notes: Ensure that passwords are not sent to users in email.

Status

This vulnerability is not applicable. Quacker does not provide the ability to reset a password because it is out of scope for this project. The Microsoft Identity framework that is used in Quacker has support for sending emails for resetting passwords.

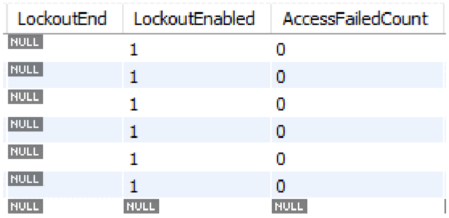
Password Lockout

Objective

Ensure that the users account is locked out for a period of time when the incorrect password is entered more that a specific number of times (usually 5).

Status

This vulnerability is not applicable. Quacker does not provide the ability to lock users out because it is out of scope for this project. The Microsoft Identity framework that is used in Quacker has support for user lockouts. The below image shows the Identity-generated columns that exist for this reason.



LockoutEnd	LockoutEnabled	AccessFailedCount
NULL	1	0
NULL	1	0
NULL	1	0
NULL	1	0
NULL	1	0
NULL	1	0
NULL	NULL	NULL

Figure 8. Lockout related columns generated by the Identity framework

Password Structure

Objective

Ensure that special meta characters cannot be used within the password

Notes: Can be useful when performing SQL injection

Status

After performing a test by registering a user with the password: ' or 1=1; truncate table AspNetUsers; -- it does not seem to affect anything in the applications. Since the authentication and user management is mostly handled by Microsoft's Identity Framework, many potential vulnerabilities such as special meta characters and SQL injection are protected by the ORM.

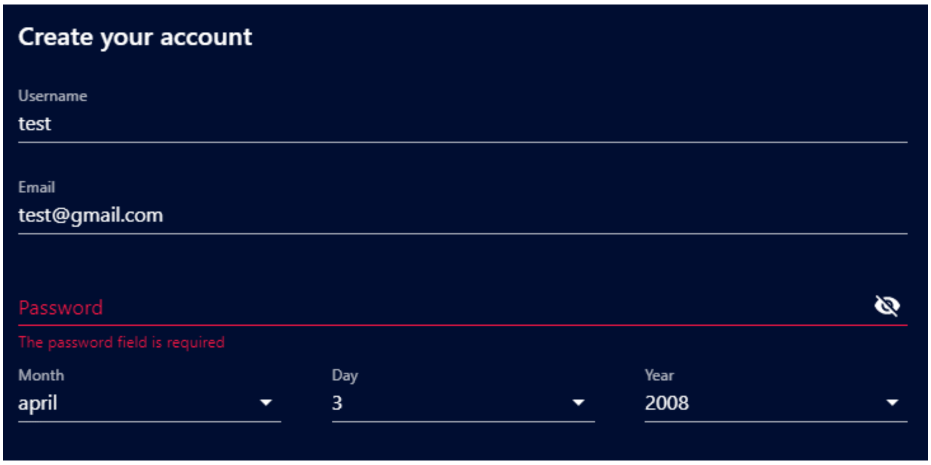
Blank Passwords

Objective

Ensure that passwords are not blank

Status

It is ensured in both the front end and the back end that passwords are not blank when registering an account.



Create your account

Username: test

Email: test@gmail.com

Password

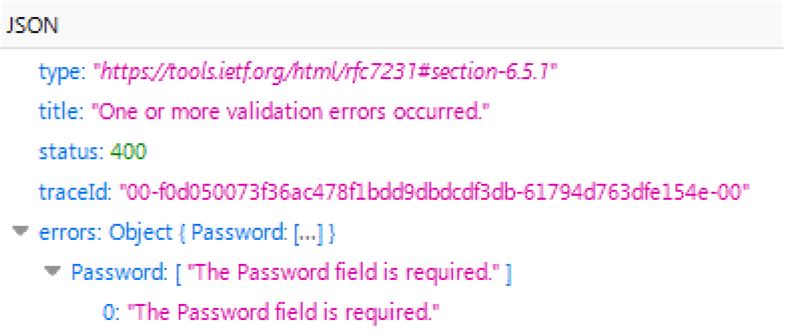
The password field is required

Month: april

Day: 3

Year: 2008

Figure 9. Front-end password check



```
JSON
    type: "https://tools.ietf.org/html/rfc7231#section-6.5.1"
    title: "One or more validation errors occurred."
    status: 400
    traceId: "00-f0d050073f36ac478f1bdd9dbdcdf3db-61794d763dfe154e-00"
  ▼ errors: Object { Password: […] }
      ▼ Password: [ "The Password field is required." ]
            0: "The Password field is required."
```

Figure 10. Response when registering directly with the back end

1.5 Authentication.SessionManagement

Session Token Length

Objective

Ensure that the session token is of adequate length to provide protection from guessing during an authenticated session.

Status

Not applicable as Quacker uses JWT authentication instead of session tokens. The user's state is not on the server, but instead the JWT is sent with every request.

Session Timeout

Objective

Ensure that the session tokens are only valid for a predetermined period after the last request by the user.

Status

The JWT tokens have an expiry date of two days. This expiry date is the same as the cookie expiry date. After two days, a user will have to login again.

Session Reuse

Objective

Ensure that session tokens are changed when the user moves from an SSL protected resource to a non-SSL protected resource.

Status

Not applicable as Quacker uses JWT authentication instead of session tokens.

Session Deletion

Objective

Ensure that the session token is invalidated when the user logs out.

Status

Not applicable as Quacker uses JWT authentication instead of session tokens. JWTs can still be invalidated on logout. Ways too do this would be by creating a token blocklist or keeping a version number on the JWT and changing this on every logout.

Session Token Format

Objective

Ensure that the session token is non-persistent and is never written to the browsers history or cache

Status

Not applicable as Quacker uses JWT authentication instead of session tokens.

1.6 Configuration.Management

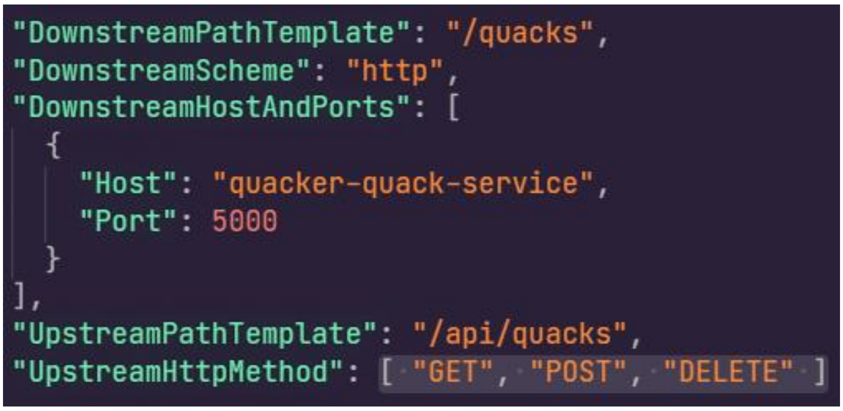
HTTP Methods

Objective

Ensure that the web server does not support the ability to manipulate resources from the Internet (e.g. PUT and DELETE)

Status

Since the back end is an API, it requires some HTTP methods to be supported. The API gateway handles all requests and carefully determines which HTTP methods are allowed for which endpoint.



```
"DownstreamPathTemplate": "/quacks",
"DownstreamScheme": "http",
"DownstreamHostAndPorts": [
  {
    "Host": "quacker-quack-service",
    "Port": 5000
  }
],
"UpstreamPathTemplate": "/api/quacks",
"UpstreamHttpMethod": [ "GET", "POST", "DELETE" ]
```

Figure 11. API gateway configuration

Virtually Hosted Sites

Objective

Try and determine if site is virtually hosted.

Notes: If there are further sites, they could be vulnerable and lead to the compromise of the base server

Status

Quacker is solely hosted on a Kubernetes cluster. This cluster is not shared with any other websites or applications. Furthermore, Cloudflare is used to cover the origin IP of the cluster.

Known Vulnerabilities / Security Patches

Objective

Ensure that known vulnerabilities which vendors have patched are not present.

Status

In order to truly discover vulnerabilities, Quacker should be penetration tested. This is best done when the tester is provided with internal information of the software. This allows the tester to analyze what vulnerabilities might be present in the architecture and how they can affect the application itself. When possible, these vulnerabilities can be tested to determine their real effects and to detect if there might be any external elements that might reduce or negate the possibility of successful exploitation.

Back-up Files

Objective

Ensure that no backup files of source code are accessible on the publicly accessible part of the application

Status

The code is stored in private repositories on GitHub. All repositories are private for security reasons.

Web Server Configuration

Objective

Ensure that common configuration issues such as directory listings and sample files have been addressed

Status

Nginx is used as web server for quacker. Quacker currently uses a basic Nginx configuration which has not been completely tested for vulnerabilities.

Web Server Components

Objective

Ensure that web server components like Front Page Server Extensions or Apache modules do not introduce any security vulnerabilities

Status

This vulnerability is not applicable as no web server components are used. Only the base NGINX server is used.

Common Paths

Objective

Check for existence of common directories within the application root

Notes: /backup & /admin may contain information

Status

There are currently no common directories that contain vulnerable information. Furthermore, the user will be redirected and will have no access to the resource without valid permissions.

Language/Application defaults

Objective

J2EE environmental quirks e.g Availability of snoop.jsp /*Spy.jsp and loaded modules

Status

The languages used in Quacker have previously been researched and graded on their vulnerabilities/security. Both the front-end and back-end languages have no significant vulnerabilities or quirks that could be exploited in Quacker.

1.7 Configuration.Management.Application

Infrastructure Admin Interfaces

Objective

Ensure that administrative interfaces to the applications are not accessible to the Internet.

Status

Administrative interfaces such as monitoring software (Grafana, Kiali, etc.) are not publicly accessible. A user would need the credentials for the Kubernetes cluster in order to see these interfaces and then port-forward the internal applications to their system.

1.8 Error Handling

Application Error Messages

Objective

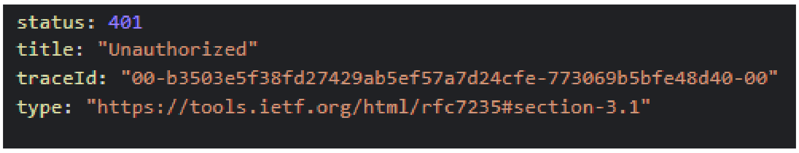
Ensure that the application does not present application error messages to an attacker that could be used in an attack.

Notes: This typically occurs when applications return verbose error messages such as stack traces or database errors.

Status

Standard error handling is handled by the .NET Core framework. The error messages that are producer do not contain verbose error messages such as stack tracer or database errors by nature. This information is logged, but not sent to the user.

.NET Core error messages do contain a trace id.



```
status: 401
title: "Unauthorized"
traceId: "00-b3503e5f38fd27429ab5ef57a7d24cfe-773069b5bfe48d40-00"
type: "https://tools.ietf.org/html/rfc7235#section-3.1"
```

Figure 12. Error message containing trace id

Trace ids cannot be used in attacks. They are unique for every request and can be used to uniquely identify the request in logs.

User Error Messages

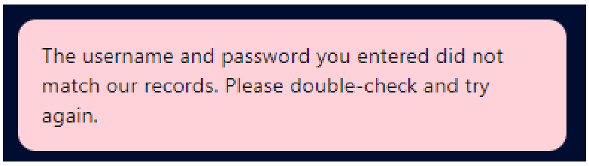
Objective

Ensure that the application does not present user error messages to an attacker that could be used in an attack.

Notes: This typically occurs when applications return error messages such as "User does not exist" or "User Correct, Password Incorrect"

Status

For user messages, Quacker tries to avoid giving vulnerable information. For example, Quacker will prompt the following error on an invalid login:



The username and password you entered did not match our records. Please double-check and try again.

Figure 13. Error when attempting to log in with invalid credentials

This error message lets the user know that their password is incorrect without letting potential attackers know that the username/password exists.

1.9 DataProtection

Sensitive Data in HTML

Objective

Ensure that there is no sensitive data in the HTML (cached in the browser history) that could lead an attacker to mount a focused attack.

Notes: This typically occurs when developers leave information in html comment or the application renders names and addresses in HTML.

Status

There is no sensitive data in the HTML as all data is dynamically loaded from the back end. The HTML merely acts as a template for the data.

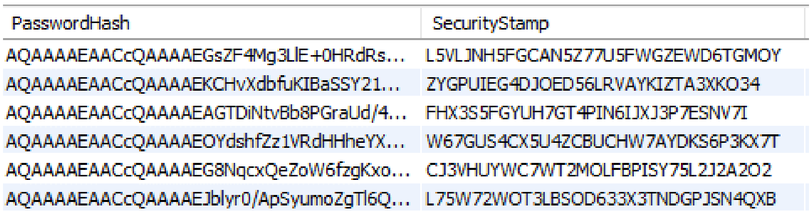
Data Storage

Objective

Ensure where required, data is protected to protect its confidentiality and integrity.

Status

All data is stored in a managed database cluster from DigitalOcean. Passwords are hashed by the Identity framework in order to protect their confidentiality.



PasswordHash	SecurityStamp
AQAAAAEAACcQAAAAEGsZF4Mg3LlE+0HRdRs...	L5VLJNH5FGCAN5Z77U5FWGZEWD6TGMOY
AQAAAAEAACcQAAAAEKCHvXdbfuKIBaSSY21...	ZYGPUIEG4DJOED56LRVAYKIZTA3XKO34
AQAAAAEAACcQAAAAEAGTDiNtvBb8PGraUd/4...	FHX3S5FGYUH7GT4PIN6IJXJ3P7ESNV7I
AQAAAAEAACcQAAAAEOYdshfZz1VRdHHheYX...	W67GUS4CX5U4ZCBUCHW7AYDKS6P3KX7T
AQAAAAEAACcQAAAAEG8NqcxQeZoW6fzgKxo...	CJ3VHUYWC7WT2MOLFBPISY75L2J2A2O2
AQAAAAEAACcQAAAAEJblyr0/ApSyumoZgTl6Q...	L75W72WOT3LBSOD633X3TNDGPJSN4QXB

Figure 14. Example of password hashing and security stamps

1.10 DataProtection.Transport

SSL Version

Objective

Ensure that SSL versions supported do not have cryptographic weaknesses.

Status

The SSL encryption and version are handled by Cloudflare and cannot be decided by Quacker.

SSL Key Exchange Methods

Objective

Ensure that the web server does not allow anonymous key exchange methods.

Notes: Typically ADH Anonymous Diffie Hellman.

Status

The SSL encryption is handled by Cloudflare and cannot be decided by Quacker.

SSL Algorithms

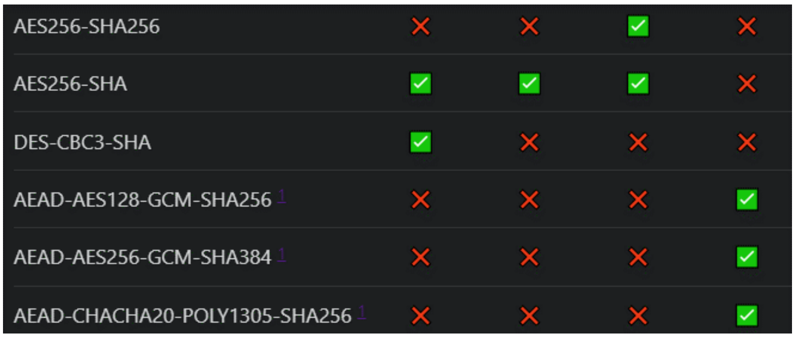
Objective

Ensure that weak algorithms are not available.

Notes: Typically algorithms such as RC2 and DES.

Status

The SSL ciphers are determined by Cloudflare and cannot be decided by Quacker.



AES256-SHA256	✗	✗	✓	✗
AES256-SHA	✓	✓	✓	✗
DES-CBC3-SHA	✓	✗	✗	✗
AEAD-AES128-GCM-SHA256 [1]	✗	✗	✗	✓
AEAD-AES256-GCM-SHA384 [1]	✗	✗	✗	✓
AEAD-CHACHA20-POLY1305-SHA256 [1]	✗	✗	✗	✓

Figure 15. Cloudflare's supported cipher suites by protocol

It can be seen that Cloudflare frequently updates their supported cipher suites. Quacker uses TLS 1.3 which means that one of the bottom three cipher suites are used. It should be noted that Quacker uses the free version of Cloudflare’s SSL.

SSL Key Lengths

Objective

Ensure the web site uses an appropriate length key.

Notes: Most web sites should enforce 128 bit encryption

Notes

The SSL encryption is handled by Cloudflare and cannot be decided by Quacker.

Digital Certificate Validity

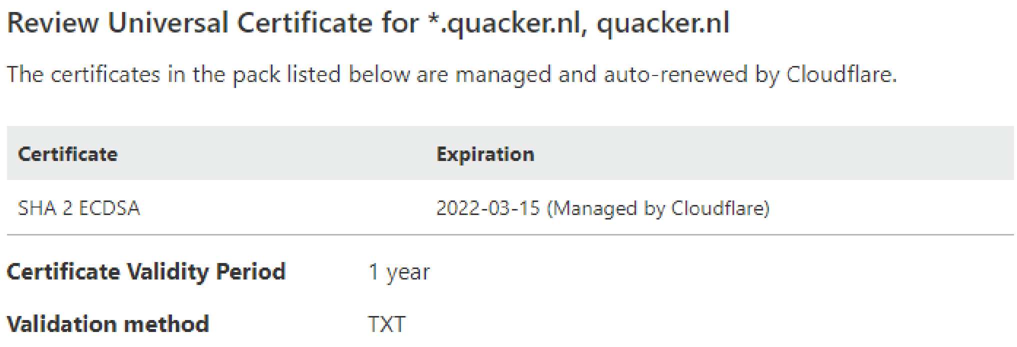
Objective

Ensure the application uses valid digital certificates.

Notes: Ensure that the digital certificate is valid, that is to say its signature, host, date etc are all valid.

Notes

Certificates for Quacker are managed by Cloudflare. The certificates for Quacker.nl expire on 2022-03-15



Review Universal Certificate for *.quacker.nl, quacker.nl

The certificates in the pack listed below are managed and auto-renewed by Cloudflare.

Certificate	Expiration
SHA 2 ECDSA	2022-03-15 (Managed by Cloudflare)

Certificate Validity Period	1 year
Validation method	TXT

Figure 16. Cloudflare's SSL certificate review

1.11 InputValidation

Script Injection

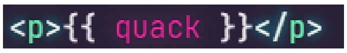
Objective

Ensure that any part of the application that allows input does not process scripts as part of the input.

Notes: Classic case of Cross Site Scripting but includes other scripting as well.

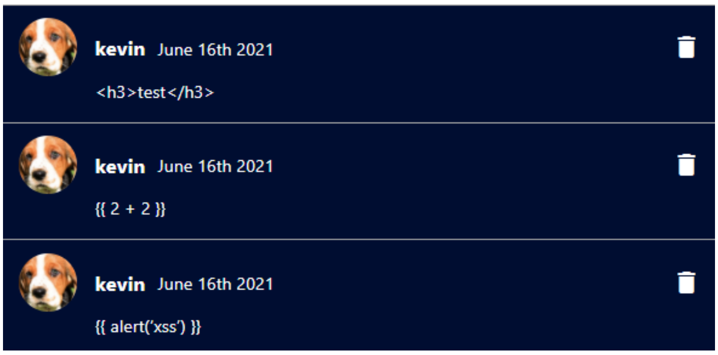
Status

The front end is built with Vue.js. Interpolated expressions are used for showing all data. These expressions are stringified and cannot be executed by the browser.



```
<p>{{ quack }}</p>
```

Figure 17. Example of interpolated expression for showing quacks



kevin June 16th 2021

<h3>test</h3>

kevin June 16th 2021

{{ 2 + 2 }}

kevin June 16th 2021

{{ alert('xss') }}

Figure 18. Example of how HTML and JavaScript are stringified and displayed

1.12 InputValidation.SQL

SQL Injection

Objective

Ensure the application will not process SQL commands from the user.

Status

For communication with the database, the ORM EntityFramework is used. EntityFramework is used in combination with C#'s LINQ queries. LINQ queries are protected from SQL Injection because parameterized values are used in the translation process from C# to SQL. This provides the same type of protection that Stored Procedures would normally provide.

LINQ is the only way of communicating that is used in Quacker. There are no occurrences of manual SQL execution.

1.13 InputValidation.OS

OS Command Injection

Objective

Ensure the applications will not process operating system commands from the user.

Notes: This typically includes issues such as path traversal, spawning command shells and OS functions.

Status

This vulnerability is not applicable for Quacker as there are no occurrences where system commands are executed.

1.14 InputValidation.LDAP

LDAP Injection

Objective

Ensure the application will not process LDAP commands form the user.

Status

This vulnerability is not applicable for Quacker as there are no occurrences where LDAP statements are executed. Therefore, injecting them will not cause any harm.

1.15 InputValidation.XSS

Cross Site Scripting

Objective

Ensure that the application will not store or reflect malicious script code.

Status

There are currently no filters for malicious script code. Any malicious script will not be executed due to protection of EntityFramework and Vue.js. However, there is currently nothing that prevents malicious code from being stored in the database as strings. In order to prevent thing, a filter can be designed that checks for certain characters and code.

1.16 BufferOverflow

Overflows

Objective

Ensure that the application is not susceptible to any buffer overflows.

Notes: Fuzzing tools help with testing all components of an application for this issue.

Status

Due to the nature of C#, buffer overflows can only with the usage of certain unsafe constructs, and not with "normal" C# code. This is not the case in Quacker; thus, buffer overflows are not a vulnerability. If a buffer overflow does appear, it will throw an exception instead of crashing the system.

Heap Overflows

Objective

Ensure that the application is not susceptible to any heap overflows.

Status

As a heap overflow is a form of buffer overflow, the chance of it happening is little to none, due to the nature of C# as a language. If it were to happen, the system would not crash, but instead throw an exception.

Stack Overflows

Objective

Ensure that the application is not susceptible to any stack overflows.

Status

According to the Microsoft docs: *StackOverflowException is thrown for execution stack overflow errors, typically in case of a very deep or unbounded recursion. So make sure your code doesn't have an infinite loop or infinite recursion.*

Quacker uses no form of recursion at all. The chance of a stack overflow happening is low to zero. Like the other overflows, this would not crash the system.

Format Strings

Objective

Ensure that the application is not susceptible to any format string overflows.

Status

C# code is not susceptible to memory address overwrites as process memory is managed by the CLR (common language runtime) rather than it being the responsibility of the developer.

2 OWASP checklist

🟩	The vulnerability has been fixed
🟨	The vulnerability has been researched but no fix is implemented
🟪	The vulnerability is not applicable
🟥	The vulnerability has not been researched or fixed

Category	Name	Objective	Notes	Status
AppDOS	Application Flooding	Ensure that the application functions correctly when presented with large volumes of requests, transactions and / or network traffic.	Use various fuzzing tools to perform this test (e.g. SPIKE)	🟩
	Application Lockout	Ensure that the application does not allow an attacker to reset or lockout user's accounts.		🟪
AccessControl	Parameter Analysis	Ensure that the application enforces its access control model by ensuring that any parameters available to an attacker would not afford additional service.	Typically this includes manipulation of form fields, URL query strings, client-side script values and cookies.	🟩
	Authorization	Ensure that resources that require authorization perform adequate authorization checks before being sent to a user.		🟩
	Authorization Parameter Manipulation	Ensure that once valid user has logged in it is not possible to change the session ID's parameter to reflect another user account	i.e. accountnumber, policynumber,usernr etc	🟩

Category	Name	Objective	Notes	Status
	Authorized pages/functions	Check to see if it’s possible to access pages or functions which require logon but can be bypassed		🟩
	Application Workflow	Ensure that where the application requires the user to perform actions in a specific sequence, the sequence is enforced.		🟩
Authentication	Authentication endpoint request should be HTTPS	Ensure that users are only asked to submit authentication credentials on pages that are served with SSL.	This ensures that the user knows who is asking for his / her credentials as well as where they are being sent.	🟩
	Authentication bypass	Ensure that the authentication process can not be bypassed.	Typically this happens in conjunction with flaws like SQL Injection.	🟨
Authentication. User	Credentials transport over an encrypted channel	Ensure that usernames and passwords are sent over an encrypted channel.	Typically this should be SSL.	🟩
	Default Accounts	Check for default account names and passwords in use		🟨
	Username	Ensure that the username is not public (or “wallet”) information such as email or SSN.		🟩
	Password Quality	Ensure that the password complexity makes guessing passwords difficult.		🟩
	Password Reset	Ensure that user must respond to a secret answer / secret question or other predetermined information before passwords can be reset.	Ensure that passwords are not sent to users in email.	🟪

Category	Name	Objective	Notes	Status
	Password Lockout	Ensure that the users account is locked out for a period of time when the incorrect password is entered more that a specific number of times (usually 5).		🟨
	Password Structure	Ensure that special meta characters cannot be used within the password	Can be useful when performing SQL injection	🟩
	Blank Passwords	Ensure that passwords are not blank		🟩
Authentication. SessionManagement	Session Token Length	Ensure that the session token is of adequate length to provide protection from guessing during an authenticated session.		🟪
	Session Timeout	Ensure that the session tokens are only valid for a predetermined period after the last request by the user.		🟩
	Session Reuse	Ensure that session tokens are changed when the user moves from an SSL protected resource to a non-SSL protected resource.		🟪
	Session Deletion	Ensure that the session token is invalidated when the user logs out.		🟨
	Session Token Format	Ensure that the session token is non-persistent and is never written to the browsers history or cache		🟪

Category	Name	Objective	Notes	Status
Configuration Management	HTTP Methods	Ensure that the web server does not support the ability to manipulate resources from the Internet (e.g. PUT and DELETE)		🟩
	Virtually Hosted Sites	Try and determine if site is virtually hosted.	If there are further sites, they could be vulnerable and lead to the compromise of the base server	🟩
	Known Vulnerabilities / Security Patches	Ensure that known vulnerabilities which vendors have patched are not present.		🟨
	Back-up Files	Ensure that no backup files of source code are accessible on the publicly accessible part of the application		🟨
	Web Server Configuration	Ensure that common configuration issues such as directory listings and sample files have been addressed		🟨
	Web Server Components	Ensure that web server components like Front Page Server Extensions or Apache modules do not introduce any security vulnerabilities		🟪
	Common Paths	Check for existence of common directories within the application root	/backup & /admin may contain information	🟩
	Language/Application defaults	I.e. J2EE environmental quirks e.g Availability of snoop.jsp /*Spy.jsp and loaded modules		🟩

Category	Name	Objective	Notes	Status
Configuration. Management. Application	Application Admin Interfaces	Ensure that administrative interfaces to the applications are not accessible to the Internet.		🟩
Error Handling	Application Error Messages	Ensure that the application does not present application error messages to an attacker that could be used in an attack.	This typically occurs when applications return verbose error messages such as stack traces or database errors.	🟩
	User Error Messages	Ensure that the application does not present user error messages to an attacker that could be used in an attack.	This typically occurs when applications return error messages such as “User does not exist” or “User Correct, Password Incorrect”	🟩
DataProtection	Sensitive Data in HTML	Ensure that there is no sensitive data in the HTML (cached in the browser history) that could lead an attacker to mount a focused attack.	This typically occurs when developers leave information in html comment or the application renders names and addresses in HTML.	🟩
	Data Storage	Ensure where required, data is protected to protect its confidentiality and integrity.		🟩

Category	Name	Objective	Notes	Status
DataProtection Transport	SSL Version	Ensure that SSL versions supported do not have cryptographic weaknesses.		🟨
	SSL Key Exchange Methods	Ensure that the web server does not allow anonymous key exchange methods.	Typically ADH Anonymous DiffieHellman.	🟥
	SSL Algorithms	Ensure that weak algorithms are not available.	Typically algorithms such as RC2 and DES.	🟥
	SSL Key Lengths	Ensure the web site uses an appropriate length key.	Most web sites should enforce 128 bit encryption	🟥
	Digital Certificate Validity	Ensure the application uses valid digital certificates.	Ensure that the digital certificate is valid, that is to say its signature, host, date etc are all valid.	🟩
InputValidation	Script Injection	Ensure that any part of the application that allows input does not process scripts as part of the input.	Classic case of Cross Site Scripting but includes other scripting as well.	🟩
InputValidation. SQL	SQL Injection	Ensure the application will not process SQL commands from the user.		🟩
InputValidation. OS	OS Command Injection	Ensure the applications will not process operating system commands from the user.	This typically includes issues such as path traversal, spawning command shells and OS functions.	🟨
InputValidation. LDAP	LDAP Injection	Ensure the application will not process LDAP commands form the user.		🟨
InputValidation. XSS	Cross Site Scripting	Ensure that the application will not store or reflect malicious script code.		🟩

Category	Name	Objective	Notes	Status
BufferOverflow	Overflows	Ensure that the application is not susceptible to any buffer overflows.	Fuzzing tools help with testing all components of an application for this issue.	🟨
	Heap Overflows	Ensure that the application is not susceptible to any heap overflows.		🟨
	Stack Overflows	Ensure that the application is not susceptible to any stack overflows.		🟨
	Format Strings	Ensure that the application is not susceptible to any format string overflows.		🟨