

OWASP TOP 10 Documentation

Introduction

This documentation aims to provide a high-level breakdown of how each of the top 10 OWASP vulnerabilities were addressed, if they were relevant e.g. CI/CD prevention methods will not be followed if our development approach was more akin to a waterfall method. There will be dot points simplifying and summing up the OWASP prevention methods as mentioned in the official website. With the text in green, the implementations explained at a high-level.

OWASP #1 Broken Access Control

To quote directly from the website “94% of applications were tested for some form of broken access control”. Which is one of the major factors for why it is ranked as the #1 vulnerability.

To prevent broken access control this is the recommended:

- **Deny access by default, except for public resources:**

By default, no one should have access to resources unless explicitly allowed. Only resources meant to be public should be open to all.

This has been addressed by protected routing, meaning that no one has access to the SIEM pages unless they are logged and have the correct role permissions. Additionally, they cannot navigate to restricted pages by URLs alone.

- **Centralize and reuse access control mechanisms across the application:**

Implement access control rules in one place and apply them consistently throughout the app. Limit cross-site access (CORS) to only what’s necessary to reduce risks.

Only have access to the backend if you’re from ssiem.dev or localhost3000 which is necessary.

- **Enforce ownership and business rules in data models:**

Ensure that users can only manage records they own (e.g., edit or delete their own data).

Users can only access and manage resources they have access to (analyst and admins) with admins only being able to delete and create users.

Logged in as Analyst and do not have access to Admin page.

Logged in as an Admin and do have access to Admin page.

- **Secure the web server by disabling directory listings and protecting sensitive files:**

```

<OpenInvestigationsNotification />
<Routes>
  <Route path="/" element={<LandingPage />} />
  <Route path="/login" element={<Login />} />
  <Route element={<PrivateRoute />>
    <Route element={<ProtectedLayout />>
      <Route path="/dashboard" element={<Dashboard />} />
      <Route path="/investigations" element={<Investigations />} />
      <Route path="/queries" element={<Queries />} />
      <Route path="/reports" element={<Reports />} />
      <Route path="/alerts" element={<Alerts />} />
      <Route path="/preferences" element={<Preferences />} />
      <Route
        path="/admin"
        element={
          <PrivateRoute roles={["ADMIN"]}>
            <AdminPage />
          </PrivateRoute>
        }
      />
    </Route>
  </Route>
</Routes>

```

Figure 1: image

```

# React App
## Need to add Backend API here not Local Host dev environment
REACT_APP_API_URL=http://localhost:8000/api
# Add BACKEND ENV HERE
# REACT_APP_API_URL=

# bypass frontend log in
REACT_APP_BYPASS_AUTH=false

# Security settings
CORS_ALLOWED_ORIGINS=http://localhost:3000,https://ssiem.dev
CSRF_TRUSTED_ORIGINS=http://localhost:3000,http://127.0.0.1:3000,https://ssiem.dev

# JWT Settings ( days is the unit)
JWT_ACCESS_TOKEN_LIFETIME=1
JWT_REFRESH_TOKEN_LIFETIME=1

```

Figure 2: image

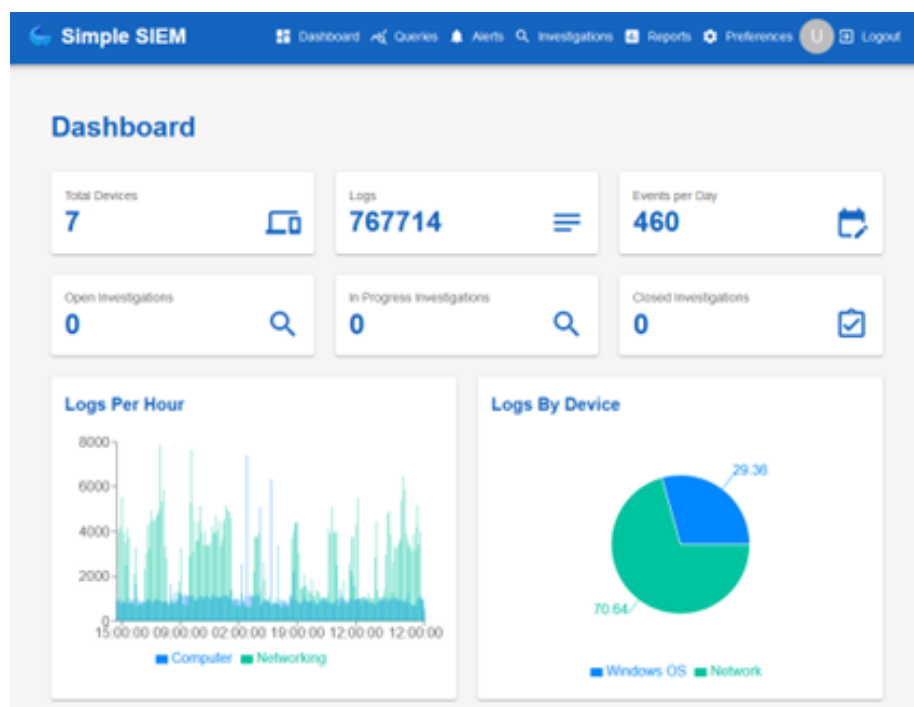


Figure 3: image

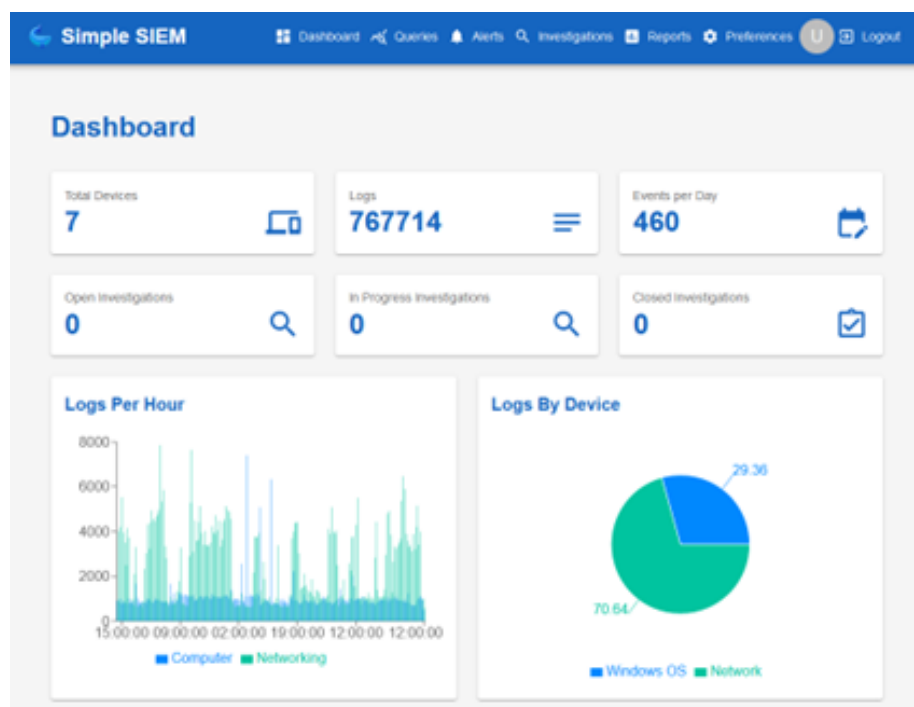
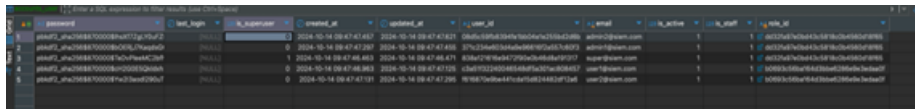


Figure 4: image

Prevent users from seeing the contents of server folders and ensure sensitive files (e.g., backups, metadata like .git) are not publicly accessible.

Log storage and passwords storage is done outside of the application and it privatized.



| pk_login | pk_update | created_at | updated_at | user_id | email | is_active | is_staff | is_superuser |
|---|-----------|-------------------------|-------------------------|----------------------------------|----------------|-----------|----------|--------------|
| pk_login_00000000000000000000000000000000 | | 2024-10-14 08:47:47.457 | 2024-10-14 08:47:47.457 | 00000000000000000000000000000000 | admin@open.com | 1 | 1 | 1 |
| pk_login_00000000000000000000000000000000 | | 2024-10-14 08:47:47.457 | 2024-10-14 08:47:47.457 | 00000000000000000000000000000000 | admin@open.com | 1 | 1 | 1 |
| pk_login_00000000000000000000000000000000 | | 2024-10-14 08:47:47.457 | 2024-10-14 08:47:47.457 | 00000000000000000000000000000000 | admin@open.com | 1 | 1 | 1 |
| pk_login_00000000000000000000000000000000 | | 2024-10-14 08:47:47.457 | 2024-10-14 08:47:47.457 | 00000000000000000000000000000000 | admin@open.com | 1 | 1 | 1 |
| pk_login_00000000000000000000000000000000 | | 2024-10-14 08:47:47.457 | 2024-10-14 08:47:47.457 | 00000000000000000000000000000000 | admin@open.com | 1 | 1 | 1 |

Figure 5: image

- **Monitor access control failures and alert admins when needed:**

Log any failed attempts to access restricted resources and notify admins if there are repeated failures, which could indicate an attack.

The auditing/logging system captures logins/login failures, user creations and editing of user attributes.

REFER TO OWASP #9 FOR MORE INFO.

- **Rate limit API requests and log users out securely:**

Limit the number of API requests users can make to prevent automated attacks. Ensure session IDs are invalidated after logout and make short-lived tokens (like JWTs) to reduce the risk of misuse. For longer-lasting tokens, follow standards to revoke access when necessary.

We have a rate limit in place as well as an expiry of tokens after a day.

GitHub Link line 92-97

OWASP #2 Cryptographic Failures

- **Always use encryption for sensitive data:**

Always encrypt sensitive information (like passwords, credit card numbers) both when it's stored and when it's being sent over the internet.

Passwords are encrypted in the database; this is the only sensitive data we store. Via a built-in function in Django.

- **Avoid outdated or weak algorithms:**

Use modern, secure encryption methods like AES (Advanced Encryption Standard) and avoid old, broken ones like MD5 or SHA-1.

We are not using outdated algorithms or encryptions, we are using PBKDF2, SHA256 hash as built into Django.

For more information refer to the Django documentation: Django Password Documentation

- **Use proper key management:**

Keep encryption keys safe by storing them securely, never hardcoding them into your app, and rotating them regularly.

Environmental variables aka secrets, these values are not hard coded into our application.

GitHub Link

- **Enable HTTPS (SSL/TLS) for all data transmission:**

Always use HTTPS to make sure that data sent between users and your site is encrypted and secure.

Our website has https enabled as well as an SSL cert.

For proof go on to SSIEM and refer to the certificate.

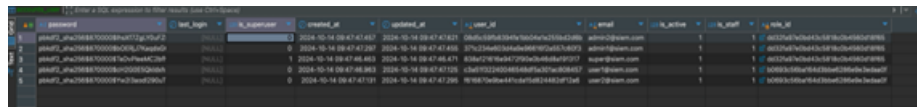


Figure 6: image

- **Do not rely on custom or homegrown cryptography:**

Don't try to create your own encryption methods—use well-tested and trusted libraries and standards.

We are using industry standard encryptions and have not created custom encryptions.

As mentioned earlier we are using Django in-built encryption.

- **Avoid exposing unnecessary sensitive data:**

Only collect and store data that you absolutely need. Don't expose sensitive information unless it's necessary.

We do not store anything that is not needed for the functioning of the website (logs and passwords) directly onto the application itself. No sensitive information is exposed as you are required to login to view logs and alerts.

OWASP #3 Injection

- **Use parameterized queries:**

Avoid putting user input directly into your database queries. Instead, use placeholders (parameters) to keep user input separate from the actual code.

We are using parameterized queries in the backend.

GitHub Link (Entire page has parameterized queries)

- **Use safe APIs and libraries:**

Always use well-established libraries or frameworks that protect against injection attacks. Avoid building your own database queries from scratch.

We are using REST API Framework, Django, React.js, and other well-documented and established tools.

- **Validate and sanitize user inputs:**

Make sure that all data users provide is properly cleaned and checked before it's used in your application (e.g., limiting input to expected characters like letters and numbers).

We are using DOMFury to sanitize the frontend user inputs before they are sent to the backend. In addition to this we have backend validation/sanitization as well. E.g. Date input by users is not valid unless it is in a valid date format.

Below is a quick test of the DOMFury in action:

Within the development environment of the SSIEM, a simple test was made. Two images rendered with unsafe HTML. One is sanitized, and the other is not.

When clicking on the sanitized HTML, nothing appears but when clicking on the unsafe tag, the following alert is shown to demonstrate that possible XSS can be done.

Here is the source code for how the test is done.

- **Disable dangerous database features:**

Turn off or avoid using database features that can be exploited, such as dynamic queries or stored procedures without input validation.

While we are using dynamic queries, they are all parameterized.

GitHub Link

- **Keep your software up to date:**

Regularly update your software (databases, libraries, etc.) to fix known vulnerabilities that could be used in injection attacks.

We are using NPM to ensure our tools are up to date as well as dependabot from GitHub. Although it is important to note, we are shipping this solution as an 'out of box' experience, therefore post-shipping of this application, updating is not our responsibility.

Using NPM, we can see packages that need to be updated.

We can do an automated update of all these packages, when the command is run again it confirms that everything is up to date.

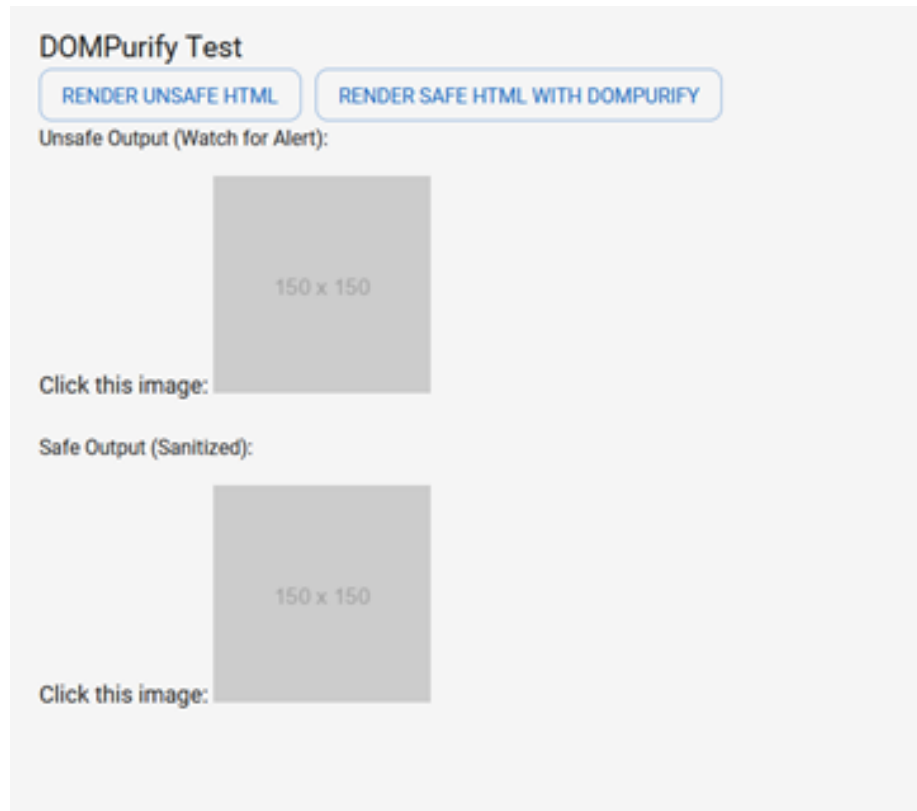


Figure 7: image

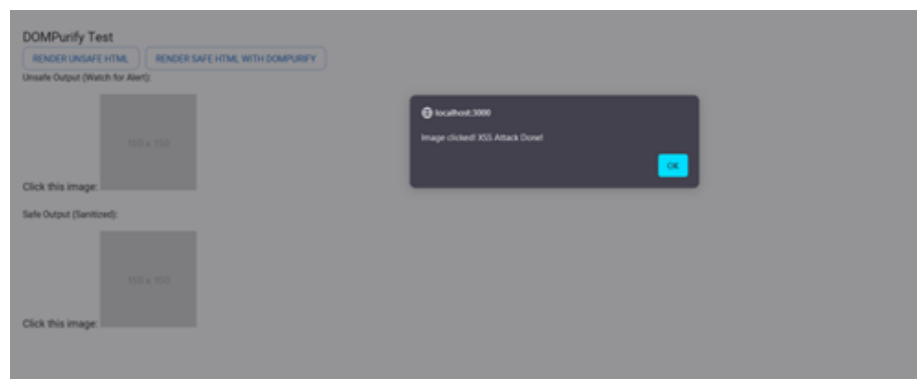


Figure 8: image


```

frontend > src > components > # DOMPurifyTest.js > ...
1 import React from 'react';
2 import DOMPurify from 'dompurify';
3 import { Typography, Button } from '@mui/material';
4
5 const DOMPurifyTest = () => {
6   const unsafeHTML = `<p>Click this image:  {
9     // Directly render the unsafe HTML (this will trigger the onclick event without sanitization)
10    document.getElementById('unsafe-output').innerHTML = unsafeHTML;
11  };
12
13  const handleSafeTest = () => {
14    // Sanitize the HTML with DOMPurify and render the sanitized version (onclick should be removed)
15    const safeHTML = DOMPurify.sanitize(unsafeHTML);
16    document.getElementById('safe-output').innerHTML = safeHTML;
17  };
18

```

Figure 9: image

PS C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\

| Package | Current | Wanted | Latest | | Depended by |
|-----------------------------|---------|--------|--------|--|-------------|
| @mui/icons-material | 5.16.7 | 5.16.7 | 6.1.1 | node_modules/@mui/icons-material | frontend |
| @mui/material | 5.16.7 | 5.16.7 | 6.1.1 | node_modules/@mui/material | frontend |
| @testing-library/jest-dom | 5.17.0 | 5.17.0 | 6.5.0 | node_modules/@testing-library/jest-dom | frontend |
| @testing-library/react | 13.4.0 | 13.4.0 | 16.0.1 | node_modules/@testing-library/react | frontend |
| @testing-library/user-event | 13.5.0 | 13.5.0 | 14.5.2 | node_modules/@testing-library/user-event | frontend |
| date-fns | 3.6.0 | 3.6.0 | 4.1.0 | node_modules/date-fns | frontend |
| web-vitals | 2.1.4 | 2.1.4 | 4.2.3 | node_modules/web-vitals | frontend |

Figure 10: image

```

PS C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\frontend> ncu
Checking C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\frontend\package.json
[=====] 20/20 100%

@emotion/react          ^11.13.0 -> ^11.13.3
@mui/icons-material     ^5.16.6 -> ^6.1.1
@mui/material           ^5.16.6 -> ^6.1.1
@mui/x-data-grid        ^7.15.0 -> ^7.18.0
@mui/x-date-pickers     ^7.15.0 -> ^7.18.0
@testing-library/jest-dom ^5.17.0 -> ^6.5.0
@testing-library/react  ^13.4.0 -> ^16.0.1
@testing-library/user-event ^13.5.0 -> ^14.5.2
axios                   ^1.7.2 -> ^1.7.7
date-fns                 ^3.6.0 -> ^4.1.0
react-router-dom        ^6.25.1 -> ^6.26.2
web-vitals               ^2.1.4 -> ^4.2.3

```

Figure 11: image

```

PS C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\frontend> npm outdated
PS C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\frontend> ncu
Checking C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\frontend\package.json
[=====] 20/20 100%

All dependencies match the latest package versions :)
PS C:\Users\ramj6\Desktop\SIEM - Post Break\i14-ssiem\frontend>

```

Figure 12: image

OWASP #4 Insecure Design

- **User Authentication and Authorization**

Implement strong authentication and authorization mechanisms to ensure that only authorized users can access sensitive data and resources.

All data retrieved by the web app via the API requires a valid token associated with their account, ensuring that data is only accessible to those who are authorized.

```
# JWT settings
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(days=int(os.environ.get('JWT_ACCESS_TOKEN_LIFETIME', 1))),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=int(os.environ.get('JWT_REFRESH_TOKEN_LIFETIME', 1))),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,

    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'AUDIENCE': None,
    'ISSUER': None,

    'AUTH_HEADER_TYPES': ('Bearer',),
    # 'AUTH_HEADER_TYPES': ('JWT',),
    'USER_ID_FIELD': 'user_id',
    'USER_ID_CLAIM': 'user_id',

    'AUTH_TOKEN_CLASSES': ('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(days=1),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}
```

Figure 13: image

- **Encrypted Communication**

Use cryptographic functions and protocols to protect data in transit and at rest, such as HTTPS and encryption.

Traefik manages an ACME DNS challenge to generate a Let's Encrypt certificate for HTTPS communication. The systems administrator can configure LUKS or BitLocker depending on their environment to ensure data is encrypted at rest.

- **Internal Network Isolation**

Ensure internal and external systems are isolated from one another.

External-reaching services are placed behind a reverse proxy and on its own Docker network, while internal services are on a non-exposed Docker

```

traefik:
  image: "traefik:v3.1.2"
  container_name: "traefik"
  command:
    - "--log.level=DEBUG"
    - "--api.insecure=true"
    - "--providers.docker=true"
    - "--providers.docker.exposedbydefault=false"
    - "--entryPoints.web.address=:80"
    - "--entryPoints.websecure.address=:443"
    - "--certificatesresolvers.myresolver.acme.dnschallenge=true"
    - "--certificatesresolvers.myresolver.acme.dnschallenge.provider=cloudflare"
    - "--certificatesresolvers.myresolver.acme.caserver=https://acme-staging-v02.api.letsencrypt.org/directory"
    - "--certificatesresolvers.myresolver.acme.email=${CLOUDFLARE_EMAIL}"
    - "--certificatesresolvers.myresolver.acme.storage=/letsencrypt/acme.json"

```

Figure 14: image

private network.

```

networks:
  internal:
    internal: true
  external:
    external: true

```

Figure 15: image

- **Least Privilege and Separation of Duties**

Least Privilege is a security principle that states that users should only be given the minimum amount of access necessary to perform their job. This means that users should only be given access to the resources they need to do their job, and no more.

Users have limited access; only authenticated users can access and modify data, role-based access control ensures separation of duties by limiting permissions based on user roles. This ensures the principle of Least Privilege and separation of duties, reducing the likelihood of unauthorized access to sensitive resources.

OWASP #5 Security Misconfiguration

- **Keep software and systems up to date:**

Regularly update your software, libraries, and systems to protect against newly discovered vulnerabilities.

As mentioned earlier, tools will be up to date to the point of shipment of this application.

```
# accounts/permissions.py
from rest_framework import permissions

class RoleBasedPermission(permissions.BasePermission):
    def has_permission(self, request, view):
        # Allow all authenticated users to perform read operations
        if request.method in permissions.SAFE_METHODS:
            return request.user.is_authenticated

        # Check if user has the required role for other operations
        required_roles = getattr(view, 'required_roles', [])
        return request.user.is_authenticated and (request.user.role.name in required_roles if request.user.role else False)
```

Figure 16: image

- **Remove or restrict unnecessary access:**

Only give access to those who need it, and remove any user accounts, permissions, or APIs that aren't in use.

To call APIs, tokens are required which need to be authenticated.

- **Don't show too much error detail:**

When errors happen, don't display detailed information that could help attackers understand your system's inner workings.

We have very limited details regarding data, e.g. "Error, data failed to load" which does not explain anything further to a potential bad actor.

OWASP #6 Vulnerable and Outdated Components

- **Keep all software and components up to date:**

Regularly update your app, frameworks, libraries, and any software you rely on to protect against known security flaws.

As mentioned earlier, tools will be up to date to the point of shipment of this application.

- **Remove unused components:**

If you're not using certain libraries, plugins, or services, remove them to reduce the number of things that can go wrong.

We do not have unnecessary components sitting within our final version of the product.

- **Use trusted sources for components:**

Only download software, libraries, and packages from official, trustworthy sources to avoid installing something malicious.

While we used npm to install components, which does not do major checks against components. The components used in this project are from well-established developers with proper documentation of their tools, in addition

to this, the tools used are popular enough to have tutorials and study courses made on them. E.g. MUI.

OWASP #7 Identification and Authentication Failures

- **Use strong, unique passwords:**

Require users to create strong passwords that are hard to guess. Avoid allowing common or weak passwords.

Our password requirements are the following, which can be labelled as strong:

- 16 characters minimum
- 1 special character
- 1 uppercase character
- 1 number

- **Enable multi-factor authentication (MFA):**

Add an extra layer of security by requiring users to verify their identity with something they have (like a phone) in addition to their password.

Due to some time-restrictions, MFA is unable to be implemented, according to the Australian Cyber Security Centre (ACSC), when MFA is unable to be implemented, consider using strong passwords/passphrases.

Source: ACSC - Securing Your Accounts

- **Store passwords securely:**

Never store passwords in plain text. Always hash and salt passwords before saving them, so they're unreadable if accessed.

Passwords are secured and encrypted at rest.

- **Use secure session management:**

Ensure that sessions (when a user is logged in) expire after a period of inactivity. Don't allow session IDs to be reused or hijacked by attackers.

Session tokens only last 1 day, meaning that you need to re-login every day.

GitHub Link

OWASP #8 Software and Data Integrity Failures

- **Use trusted sources for updates and libraries:**

Only download and install software, updates, and libraries from verified, official sources to avoid introducing malicious code.

+ Add New User

Employee User Registration

First Name *

r

Last Name *

r

Email *

r@siem.com

Password *

•

Password must be at least 16 characters long, include 1 uppercase letter, 1 number, and 1 special character.

Employee Role

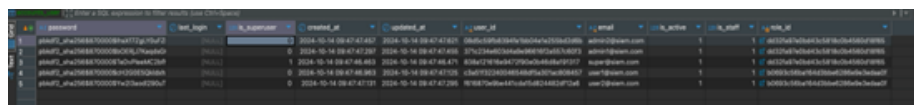
☐ ANALYST

☒ ADMIN

Please fix the password requirements

Submit

Figure 17: image



| | username | email | password | role |
|---|----------|------------------|----------|---------|
| 1 | admin | admin@siem.com | admin | ADMIN |
| 2 | analyst | analyst@siem.com | analyst | ANALYST |
| 3 | analyst | analyst@siem.com | analyst | ANALYST |
| 4 | analyst | analyst@siem.com | analyst | ANALYST |
| 5 | analyst | analyst@siem.com | analyst | ANALYST |

Figure 18: image

```

# JWT settings
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME':
timedelta(days=int(os.environ.get('JWT_ACCESS_TOKEN_LIFETIME',
1))),
    'REFRESH_TOKEN_LIFETIME':
timedelta(days=int(os.environ.get('JWT_REFRESH_TOKEN_LIFETIME',
1))),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
    'UPDATE_LAST_LOGIN': False,

    'ALGORITHM': 'HS256',
    'SIGNING_KEY': SECRET_KEY,
    'VERIFYING_KEY': None,
    'AUDIENCE': None,
    'ISSUER': None,

    'AUTH_HEADER_TYPES': ('Bearer',),
    # 'AUTH_HEADER_TYPES': ('JWT',),
    'USER_ID_FIELD': 'user_id',
    'USER_ID_CLAIM': 'user_id',

    'AUTH_TOKEN_CLASSES':
('rest_framework_simplejwt.tokens.AccessToken',),
    'TOKEN_TYPE_CLAIM': 'token_type',

    'JTI_CLAIM': 'jti',

    'SLIDING_TOKEN_REFRESH_EXP_CLAIM': 'refresh_exp',
    'SLIDING_TOKEN_LIFETIME': timedelta(days=1),
    'SLIDING_TOKEN_REFRESH_LIFETIME': timedelta(days=1),
}

```

Figure 19: image

Tools and software were installed from the official websites or the instructions from the websites were followed for downloading the tools.

E.g. `npm install @mui/material @emotion/react @emotion/styled`

- **Monitor and audit code changes:**

Keep track of all changes made to code and data, and regularly audit those changes to catch any unauthorized or suspicious activity.

We have implemented a logging of the software as mentioned earlier and in OWASP #9.

OWASP #9 Security Logging and Monitoring Failures

- **Logging application events**

Ensure all login, access control, and server-side input validation failures can be logged with sufficient user context to identify suspicious or malicious accounts and held for enough time to allow delayed forensic analysis.

Currently, the system logs account activities such as failed or successful logins, data retrieved by a user and any modifications that an administrator user may do on other user accounts i.e. creation, deletion or modification.

```
logger = logging.getLogger('accounts')

@receiver(post_save, sender=User)
def log_user_save(sender, instance, created, **kwargs):
    if created:
        logger.debug(f'New user created: {instance.email}')
    else:
        logger.debug(f'User updated: {instance.email}')

@receiver(user_logged_in)
def user_logged_in_callback(sender, request, user, **kwargs):
    logger.debug(f'User {user.email} logged in')

@receiver(user_logged_out)
def user_logged_out_callback(sender, request, user, **kwargs):
    if user:
        logger.debug(f'User {user.email} logged out')

@receiver(user_login_failed)
def user_login_failed_callback(sender, credentials, **kwargs):
    logger.debug(f'Login failed for: {credentials.get("email", "unknown")}')
```

Figure 20: image

- **Logging system events.**

Logging server side events that can be used for analysis after breach.

The system also logs any SQL queries that are performed on the database allowing for further visibility and the ability to cross-reference any malicious events that may occur.

```
'loggers': {  
  "django.server": {  
    'handlers': ['file'],  
    "propagate": True,  
    "level": 'DEBUG',  
  },  
  "django.request": {  
    'handlers': ['file'],  
    "propagate": True,  
    "level": 'DEBUG',  
  },  
  "django.security": {  
    'handlers': ['file'],  
    "propagate": True,  
    "level": 'DEBUG',  
  },  
  'accounts': {  
    'handlers': ['file', 'console'],  
    'level': 'DEBUG',  
    'propagate': True,  
  },  
},
```

Figure 21: image

OWASP #10 Server-Side Request Forgery (SSRF)

As stated in the official documentation (OWASP SSRF) for OWASP #10 SSRF “occurs when fetching remote resources”. However, due to the containerized nature of this product, it does not fetch anything remote, the data comes in from trusted APIs and no external resources are accessed. As such this OWASP vulnerability is not relevant to this project. In the future if it is decided that external resources will be used in conjunction with the product then this OWASP should be mitigated as this attack vector opens.