Security Scan Findings Report

Generated for: https://example.com

Scan Information

URL: https://example.com
Scan ID: eb4cacbf-0f29-4149-90f9-b079c247b508
Date of scan: September 09, 2025 at 03:07:10

Analysis & Recommendations

Summary

The application has been found to have several security vulnerabilities that need to be addressed. These vulnerabilities include XSS, CSRF, and file upload issues, among others. It is essential to prioritize and fix these issues to prevent potential attacks and ensure the security of the application.

Issues

Missing Security Headers

Explanation: The website is missing important security headers, such as Content-Security-Policy, that help prevent XSS attacks. This makes the application vulnerable to XSS attacks, which can lead to unauthorized access to user data and system compromise.
Fix Recommendation:

1. Step 1: Configure Next.js to include the necessary security headers in the response. This can be done by using the next.config.js file to add custom headers.
2. Step 2: Use Vercel's built-in support for security headers by configuring the vercel.json file to include the necessary headers. **Possible Fix**:

```
// next.config.js
module.exports = {
//... other configurations ...
async headers() {
return [
{
source: '/:path*',
headers: [
{
key: 'Content-Security-Policy',
value: "default-src 'self'; script-src 'self' https://cdn.example.com; object-src 'none'",
},
],
},
];
},
};
```

Priority: High

CSRF Vulnerabilities

Explanation: The application has been found to have CSRF vulnerabilities, which can allow an attacker to perform unauthorized actions on behalf of a user.
Fix Recommendation:

1. Step 1: Implement CSRF protection using a library like csurf in Next.js. This can be done by creating a middleware function that checks for the CSRF token in incoming requests.
2. Step 2: Configure the csurf library to use a secret key stored in an environment variable, and set up the necessary cookies and headers. **Possible Fix**:

```
// pages/api/csrf.js
import csrf from 'csurf';
import { NextApiRequest, NextApiResponse } from 'next';

const csrfProtection = csrf({ cookie: true });

const handler = async (req: NextApiRequest, res: NextApiResponse) => {
//... other handler code ...
await csrfProtection(req, res);
//... other handler code ...
};

export default handler;
```

Priority: Medium

DOM-based XSS Vulnerabilities

Explanation: The application has been found to have DOM-based XSS vulnerabilities, which can allow an attacker to inject malicious code into the application.
Fix Recommendation:

1. Step 1: Use a library like DOMPurify to sanitize user input and prevent XSS attacks.

2. Step 2: Configure the DOMPurify library to use a whitelist of allowed tags and attributes.
   Possible Fix:

```
// components/Input.js
import DOMPurify from 'dompurify';

const Input = ({ value }) => {
const sanitizedValue = DOMPurify.sanitize(value, {
ALLOWED_TAGS: ['p', 'span'],
ALLOWED_ATTR: ['style'],
});

return <div dangerouslySetInnerHTML={{ __html: sanitizedValue }} />;
};

export default Input;
```

Priority: High

File Upload Vulnerabilities

Explanation: The application has been found to have file upload vulnerabilities, which can allow an attacker to upload malicious files and execute them on the server.
Fix Recommendation:

1. Step 1: Use a library like multer to handle file uploads and validate the uploaded files.
2. Step 2: Configure the multer library to use a whitelist of allowed file types and sizes.
   Possible Fix:

```
// pages/api/upload.js
import multer from 'multer';
import { NextApiRequest, NextApiResponse } from 'next';

const upload = multer({
dest: './uploads/',
limits: { fileSize: 1024 * 1024 * 5 }, // 5MB
fileFilter(req, file, cb) {
if (!file.originalname.match(/\.(jpg|jpeg|png)$/)) {
return cb(new Error('Only image files are allowed'));
}
cb(null, true);
},
});

const handler = async (req: NextApiRequest, res: NextApiResponse) => {
//... other handler code ...
upload(req, res, (err) => {
if (err) {
//... handle error ...
} else {
//... handle uploaded file ...
}
});
};

export default handler;
```

Priority: High

Path Traversal Vulnerabilities

Explanation: The application has been found to have path traversal vulnerabilities, which can allow an attacker to access sensitive files and directories.
Fix Recommendation:

1. Step 1: Use a library like path to normalize and validate file paths.
2. Step 2: Configure the path library to use a whitelist of allowed directories and files.
   Possible Fix:

```
// utils/path.js
import path from 'path';

const normalizePath = (filePath) => {
const normalizedPath = path.normalize(filePath);
const allowedDirs = ['public', 'uploads'];
const allowedFiles = ['index.html', 'image.jpg'];

if (!allowedDirs.includes(normalizedPath) && !allowedFiles.includes(normalizedPath)) {
throw new Error('Access denied');
}

return normalizedPath;
};

export default normalizePath;
```

Priority: High