Security Report for application

Summary

This security report was conducted on 01/04/2020 at 01:30:10 (UTC+0). A total of 3 issue(s) were found, 2 of which may require immediate attention.

This report is produced by running automated security scanning tools, which will likely not detect all vulnerabilities present.

It is not a replacement for a manual analysis of the application.

The following technical impacts may arise if an adversary successfully exploits one of the issues found by this scan.

- **Confidentiality**: Read Application Data
- **Integrity**: Gain Privileges or Assume Identity
- **Availability**: DoS: Crash, Exit, or Restart

Statistics

This report found issues with the following severities.

Critical: 2 | **High** 0 | **Medium** 0 | **Low** 0 | **Informational** 0 | **Unknown** 1

To gain a better understanding of the severity levels please see the appendix.

Contents

Overview of Issues

Uncaught Exception

An exception is thrown from a function, but it is not caught.

When an exception is not caught, it may cause the program to crash or expose sensitive information.

Consequences

Using a vulnerability of this type an attacker may be able to affect the system in the following ways.

- **Availability**: DoS: Crash, Exit, or Restart
- **Confidentiality**: Read Application Data

An uncaught exception could cause the system to be placed in a state that could lead to a crash, exposure of sensitive information or other unintended behaviors.

For more information see CWE-248.

Sensitive Cookie Without 'HttpOnly' Flag

The product uses a cookie to store sensitive information, but the cookie is not marked with the HttpOnly flag.

An HTTP cookie is a small piece of data attributed to a specific website and stored on the user's computer by the user's web browser. This data can be leveraged for a variety of purposes including saving information entered into form fields, recording user activity, and for authentication purposes. Cookies used to save or record information generated by the user are accessed and modified by script code embedded in a web page. While cookies used for authentication are created by the website's server and sent to the user to be attached to future requests. These authentication cookies are often not meant to be accessed by the web page sent to the user, and are instead just supposed to be attached to future requests to verify authentication details.

The HttpOnly flag directs compatible browsers to prevent client-side script from accessing cookies. Including the HttpOnly flag in the Set-Cookie HTTP response header helps mitigate the risk associated with Cross-Site Scripting (XSS) where an attacker's script code might attempt to read the contents of a cookie and exfiltrate information obtained. When set, browsers that support the flag will not reveal the contents of the cookie to a third party via client-side script executed via XSS.

Consequences

Using a vulnerability of this type an attacker may be able to affect the system in the following ways.

- **Confidentiality**: Read Application Data

If the HttpOnly flag is not set, then sensitive information stored in the cookie may be exposed to unintended parties.

- **Integrity**: Gain Privileges or Assume Identity

If the cookie in question is an authentication cookie, then not setting the HttpOnly flag may allow an adversary to steal authentication data (e.g., a session ID) and assume the identity of the user.

For more information see CWE-1004.

Vulnerabilities

Critical Severity

test-issue-title (version 1.3.24)

Severity: Critical | **Type**: dependency | **Fix**: Unknown | **Found By**: test-scanner

test-issue-description with excluding term: block list

Evidence

The following evidence of this vulnerability was found in the application.

Example Web Request

In PDF format only a single example web request is included, for a more complete view, check the html, markdown or json report options.

Curl

```
curl -o - -i \
  -X POST \
  --data 'words=ZAP' \
  -H "cache-control: no-cache" \
  -H "content-type: application/x-www-form-urlencoded" \
  -H "host: localhost:3000" \
  -H "pragma: no-cache" \
  -H "referer: http://localhost:3000" \
  -H "user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36" \
  "http://localhost:3000"
```

Request

A `POST` request to the target `http://localhost:3000` with the following body:

```
"words=ZAP"
```

Response

The server responded with a `200` status code.

The following headers were returned.


```
{
  "Connection": "keep-alive",
  "Content-Length": "939",
  "Content-Type": "text/html; charset=utf-8",
  "Date": "Wed, 05 Jul 2023 11:49:16 GMT",
  "ETag": "\"3147526947+ident\"",
  "Keep-Alive": "timeout=5",
  "X-Powered-By": "Express"
}
```


Along with the response body:


```
{
  "menu": [
    {
      "value": "New",
      "onclick": "CreateNewDoc()"
    },
    {
      "value": "Open",
      "onclick": "OpenDoc()"
    },
    {
      "value": "Close",
      "onclick": "CloseDoc()"
    }
  ]
}
```


References

CVE-2022-25772 | CWE-1004 | CWE-248 | GHSA-f9xv-q969-pqx4

test-issue-title

Severity: Critical | **Type**: code smell | **Fix**: Unknown | **Found By**: test-scanner

test-issue-description

Evidence

The following evidence of this vulnerability was found in the application.

`scripts/update-cwe.js` (starting on line: 5)

```
3| /* eslint-disable @typescript-eslint/no-var-requires */
4| const {createWriteStream} = require('fs');
5| const {get} = require('https');
6| const {Parse} = require('unzip-stream');
7| const {minifyStream} = require('minify-xml');
```

References

CWE-1004 | GHSA-f9xv-q969-pqx4

Unknown Severity

test-issue-title

Severity: Unknown | **Type**: code smell | **Fix**: Unknown | **Found By**: test-scanner

test-issue-description

References

CWE-248

Additional Information

What are severity levels?

Issue severity is scored using the Common Vulnerability Scoring System (CVSS) where such data is available. Severity levels do not represent the risk associated with an issue as risk depends on your specific context. Severity scoring does however give an indication of the ease of exploitation and potential scope of an attacks effect on an application.

Critical

Exploitation will likely lead to an attacker gaining administrative access to the application and infrastructure that supports it. Exploiting critical vulnerabilities is usually trivial and will generally not require prior access to the application. **A development team should aim to resolve these issues immediately by mitigating or directly resolving the issue**.

High

Exploitation could lead to an attacker gaining elevated access to the application and the infrastructure that supports it. It is likely that an attacker will not find exploitation trivial. Such exploitation could lead to significant data loss or downtime.

Medium

Exploitation could lead to an attacker gaining limited access to the application. Exploiting vulnerabilities may require an attacker to manipulate users to gain access to their credentials. Such exploitation could lead to limited data loss or downtime.

Low

Exploitation will likely have very little impact on the application, and it is unlikely that an attacker will gain any meaningful access to the application. Exploiting an issue of this severity will potentially require physical access to the infrastructure that supports the application.

Informational

While not part of the CVSS scoring specification, several security analysis tools use this severity level to indicate that an issue is a matter of best practice. It is extremely unlikely that issues with this severity will lead to an attacker gaining access to any application components.

Unknown

This severity level is used when the analysis tool used to perform a scan of the application does not associate any kind of severity level with the issues or vulnerabilities it finds. Issues with an unknown severity should be investigated by application developers and project stakeholders to establish the ease of exploitation, scope of any potential impact and the specific risks associated.