Report

This lab emphasises the importance of deprecating old API endpoints, removing debug code, and ensuring that developers are aware of and follow relevant standards (in this case, the Payment Card Industry Data Security Standard (PCI-DSS)).

Vulnerabilities relating to HTTPS/TLS have not been raised in this report. The issues highlighted are not necessarily an exhaustive list of security issues affecting the target site.

This is my first Express application, so I suspect I'm using anti-patterns and unconventional coding practices, which I would endeavour to correct for production ready challenges.

I have other vulnerable labs available at vulnerable-labs.

Vulnerabilities and Compliance Issues

This web application contains a number of security vulnerabilities and standards violations, including:

1. Invalid JWTs (JSON Web Tokens) were accepted by the application

Description
It was possible to modify the JWT such that a signature was not required or could be invalid, while still being accepted by the application. The JWT could also be expired and still accepted by the application. This vulnerability could have allowed attackers to impersonate other users and retrieve sensitive information, such as card payment information.

Recommendations
Debug code was present in the application which decoded JWTs that were invalid, and this code should have been removed prior to go live. The affected file was **routes/middlewares/auth.js:16**.

This issue highlights the importance of unit testing, because if an invalid JWT could have been mocked and tested against the application, this issue could have been identified before being pushed to production.

Another way that this issue could have been avoided would have been to ensure that any code being pushed live was subject to peer review prior to deployment. Any debug code could have been wrapped in a function which checks the environment type (development, UAT, production etc.) prior to execution.

References

- Bathla, S. (2021). Hacking JWT Tokens: The None Algorithm - Pentester Academy Blog. Medium. Retrieved from https://blog.pentesteracademy.com/hacking-jwt-tokens-the-none-algorithm-67c14bb15771
- JWT none algorithm supported. (2023, February 06). Retrieved from https://portswigger.net/kb/issues/00200901_jwt-none-algorithm-supported

2. Full Payment Account Number (PAN) returned by API (/api/v1/cards)

Description
The application API returned a full PAN, which should not be displayed back to customers in accordance with PCI DSS Requirement 3.3. The code can be found in **routes/index.js:51**.

Recommendations
In line with PCI DSS Requirement 3.3, PANs should be masked (with a maximum of the first six and last four digits being displayed to customers), or truncated.

If a full PAN was returned by the API, this could be indicative of a miscommunication or misunderstanding of the requirements of the project by the developers or project managers. If an issue like this is identified in an application, it may be symptomatic of a larger issue based around communication and/or understanding of the requirements by relevant teams.

References

- PCI Security Standards Council (2023). Payment Card Industry Data Security Standard. Retrieved from https://docs-prv.pcisecuritystandards.org/PCI%20DSS/Standard/PCI-DSS-v4_0.pdf
- PCI Security Standards Council (2023). PCI DSS Quick Reference Guide. Retrieved from https://listings.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf

3. Card Verification Value (CVV) returned by API /api/v1/cards

Description
The application API returned the CVV, which should not be stored after authorisation in accordance with PCI DSS requirement 3.2. The code can be found in **routes/index.js:51**.

Recommendations
The application should not store the CVV after authorisation, and this endpoint should not return the CVV for cards. This functionality could reasonably lead to the organisation losing PCI accreditation, and is indicative of a poor understanding or communication of requirements.

To mitigate this issue, developers need to understand the requirements of the application and check that Sensitive Authentication Data (SAD) is not stored after an attempted purchase.

References
- PCI Security Standards Council (2023). PCI DSS Quick Reference Guide. Retrieved from https://listings.pcisecuritystandards.org/documents/PCI_DSS-QRG-v3_2_1.pdf

4. Payment Account Number (PAN) was not encrypted

Description
While not demonstrable via the web app itself, a cursory glance over the source code will highlight the fact that PANs were not encrypted, which is in violation of PCI DSS Requirement 3.4. This issue can be seen in the SQLite database (db.sqlite), and in the code responsible for adding new cards to an account (**routes/index.js:201**). No attempt is made to encrypt the CVV parameter prior to the INSERT.

Recommendations
To address this issue, PANs should be encrypted at rest to ensure compliance with PCI standards. Other mechanisms can also be deployed to adhere to relevant PCI standards, including implemmentation of a one-way hash function on the PAN, truncation, or index tokens.

To stop unprotected PANs from being disclosed in future, locations where PANs could be disclosed (in a database or web logs, for example), could be monitored for Luhn passing card numbers.

References
- Baykara, S. (2022). How can you make stored PAN information unreadable? - PCI DSS GUIDE. PCI DSS GUIDE. Retrieved from https://www.pcidssguide.com/how-can-you-make-unreadable-stored-pan-information

5. Incrementing user ID in session JWT

Description
An identifier in the JWT could be used to estimate the number of users of the application, which could be considered commercially sensitive. The ID is added to the JWT in **routes/index.js:176**.

Recommendations
If the number of users of the target application was considered commercially sensitive, an incrementing integer should not be utilised in order to uniquely identify accounts. A universally unique identifier (UUID) or hash could be more appropriate, as it does not indicate how many accounts exist in the target application, and cannot be enumerated.

6. Time-based username enumeration

Description
The login page was affected by a time-based username enumeration issue. The average response time for a non-existent user on the login HTTP POST request was ~10ms. For an existing user this increased to ~67ms.

This issue exists because different code branches are taken depending on whether the user exists or not. If the user exists, a bcrypt function is executed, while this is not the case if the user doesn't exist, contributing to the variance in execution times. The function responsible for logging in a user is available at **routes/index.js:143**.

Recommendations
The response time for valid and invalid users should be indistinguishable to mitigate this risk. This risk may be acceptable to the organisation, especially if mitigations are in place to prevent password stuffing or brute force attacks against accounts.

References
- Time based username enumeration | Pen Test Partners. (2023, February 05). Retrieved from https://www.pentestpartners.com/security-blog/time-based-username-enumeration

7. Cross-Site Request Forgery (CSRF)

Description
The add card functionality provided by the site was affected by a CSRF vulnerability. If this issue was exploited, an attacker may be able to add superfluous cards to a target account.

Recommendations
Traditionally, a CSRF token was used to mitigate this issue. A CSRF token could be sent as a key value in the form data, in a request header, or as a cookie. By whichever means the CSRF token was transmitted, the server would set and subsequently check this value to ensure that it was present and correct. If it was not present and correct, the form data would not be accepted by the target application.

Alternatively, the session cookie could be set with the SameSite attribute set to Strict. This attribute instructs the browser to only send cookies to the target application in a first party context. Cookies set with this attribute will not be sent by third party websites which initiate the request.

In absence of the session cookie in a request, any attempted actions performed as a result of the third party iniatiated request should not be fullfilled in the context of an authenticated user of the application.

To address this issue in Express, the developer should use a CSRF middleware, of which examples are available at https://www.npmjs.com/search?q=express%20csrf.

References

- SameSite cookies - HTTP | MDN. (2023, February 04). Retrieved from https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie/SameSite
- Cross-Site Request Forgery Prevention - OWASP Cheat Sheet Series. (2023, February 05). Retrieved from https://cheatsheetseries.owasp.org/cheatsheets/Cross-Site_Request_Forgery_Prevention_Cheat_Sheet.html

8. Clickjacking

Description
The target website was affected by clickjacking, an attack that allows attackers to entice victims to perform actions on a seamingly unrelated website, which results in actions being performed on the target website.

An example of this type of attack could be a malicious website which includes an invisible iframe which is overlayed. When the victim interacts with the attacker's website, they believe they are performing actions on the malicious website, and not the target website.

An attacker could entice the user to perform actions on the target site without the victim being aware. By strategically placing elements on the page, the mouse clicks and other events performed by the user could be passed to the target application. This could lead to an attacker compromising the target user account by making them change their password or associating other accounts with the victims account.

Recommendations
Anti-clickjacking mechanisms can be implemented via specific framing headers (for example, X-Frame-Options) and/or a suitable Content Security Policy (specifically, by using the frame-ancestors directive). Using the SameSite=Strict attribute in the session cookie will also mitigate against this risk, because actions performed by the user as a result of the clickjacking attack will not be performed in the context of the authenticated user on the target application.

For Express, a CSP middleware could be utilised (such as https://www.npmjs.com/package/express-csp-header) to address this issue.

References

- Clickjacking | OWASP Foundation. (2023, February 06). Retrieved from https://owasp.org/www-community/attacks/Clickjacking