THREAT MODEL - MOOSYNC

Table of Contents

SCOPE OF THE WORK

Threat Model Information

Application Version: 10.3.2 latest version released on Jan 30th 2024
Description: Moosync is an Electron based simple music player with a primary goal to provide a clean and easy interface. Through Moosync you can easily listen to music from your desktop, YouTube or Spotify. It is scalable and allows users to add their own APIs and customize the experience even more. Main users who will be included for this application are -

1. User who want to use the application (direct use or develop APIs and use custom version)
2. Users who wish to contribute for the project's development (open source Github project)
3. Main developers who update the project

Document Owner: Jui Bangali, Pamela Cheema, Rithi Afra Jerald Jothi, Lakshana Jeyakumar
Participants: Jui Bangali, Pamela Cheema, Rithi Afra Jerald Jothi, Lakshana Jeyakumar
Reviewer: Professor

External Dependencies

External dependencies should be documented as follows:

1. ID: A unique ID assigned to the external dependency.
2. Description: A textual description of the external dependency.

Even if this is a github open source project, the tool has its own website and has desktop versions to download for Windows, linux and Mac OS.

Considering this, the external dependencies could be -

ID	Description
1	The website of this tool must be running on a web server. This server will be hardened per the developer's hardening standard that they finalize. This includes the installation of the latest operating system and application security patches.

2	There are features in the tool which allow users to create their own playlist from music they get from youtube or spotify, which means there is a database where the information gets stored and retrieved as needed. This is a local database and not hosted on any server, hence it must be hardened as per the developer's policies and installed with the latest OS and application security patches.
3	The web server is behind a firewall and the only communication available is TLS.
4	The tool has an option to add APIs and perform third-party integrations with youtube, spotify and discord. The generation of public API key for the integrations is dependent on the user's account & authentication with the third-party applications. Usage and creation of the integration is included in scope of the project, but having user authentication with third-party tools is not in the scope of this project.
5	Integration with external applications like spotify and youtube available, authentication with those applications and applying mechanisms to ensure only authenticated users should integrate with this tool.
6	Libraries used for developing the project, example yarn, installing the dependencies from other programming languages will be external dependency, the security of the libraries maintained in the programming languages would not fall under application's threat model.

Entry Points

Entry points show where data enters the system (i.e. input fields, methods) and exit points are where it leaves the system (i.e. dynamic output, methods), respectively. Entry and exit points define a trust boundary (see Trust Levels).

Entry points should be documented as follows:

ID: A unique ID assigned to the entry point. This will be used to cross-reference the entry point with any threats or vulnerabilities that are identified. In the case of layered entry points, a major.minor notation should be used.

Name: A descriptive name identifying the entry point and its purpose.
Description: A textual description detailing the interaction or processing that occurs at the entry point.
Trust Levels: The level of access required at the entry point. These will be cross-referenced with the trust levels defined later in the document.

ID	Name	Description	Trust Levels
1	Playlist data access	The application provides the functionality of curating the user's own playlist. This data will be stored in a database. All the liked songs of the associated user can be accessed through this entry point.	(1) Remote Anonymous User (3) Authenticated user (authenticated through Github)
2	Extension API	The Extension Application of this API allows developers to create custom extensions on top of Moosync and publish these extensions in store.	(1) Remote Anonymous User (3) Authenticated User
3	Spotify and Youtube Url	Moosync allows users to add tracks and playlists via Spotify and Youtube URLs.	(1) Remote Anonymous User (Youtube integration is only required to access private videos or playlists) (3) Authenticated User (Authentication via Spotify and/or Youtube)
5	Codebase on Github	Moosync is an open-source application that is hosted on Github. Anyone can analyze the source code for potential vulnerabilities as well as contribute to the project.	1) Remote Anonymous User (3) Users who wish to contribute to project development (Authenticated by Github) (5) General Contributors
6	Desktop Permissions and Local File Access	Users provide audio files from their desktop to play.	(1) Remote Anonymous User (3) Authenticated User

Exit Points

Explained along with entry points.

Exit points should be documented as follows:

ID: A unique ID assigned to the exit point. This will be used to cross-reference the exit point with any threats or vulnerabilities that are identified. In the case of layered exit points, a major.minor notation should be used.
Name: A descriptive name identifying the exit point and its purpose.
Description: A textual description detailing the interaction or processing that occurs at the exit point.
Trust Levels: The level of access required at the exit point. These will be cross-referenced with the trust levels defined later in the document.

ID	Name	Description	Trust Levels
1	Codebase on Github	Moosync is an open-source application that is hosted on Github. Anyone can download the source code and any vulnerabilities will be included along with it.	(1) Remote Anonymous User (3) Authenticated User (authenticated by app/Github)
2	Extension API	Users creating a custom extension on Moosync will also inherit the possible vulnerabilities or security implications of the extended app.	(1) Remote Anonymous User (3) Authenticated User
3	Spotify and Youtube Url	Some things to consider for the URLs are data transmissions between these apps and Moosync as well as user session handling.	(1) Remote Anonymous User (Youtube integration is only required to access private videos or playlists) (3) Authenticated User (Authentication via Spotify and/or Youtube)
4	3rd Party Integration	There is an API generated to used third party apps. (Youtube, Spotify)	(1) Remote Anonymous User (Youtube integration is only required to access private videos

			or playlists) (3) Authenticated User (Authentication via Spotify and/or Youtube accounts to generate API key)
5	Playlist data access	The exit point for the user playlist needs security controls to maintain confidentiality and integrity.	(1) Remote Anonymous User (3) Authenticated user
6	Desktop Permissions and Local File Access	The exit points to play the audio file from their desktop must have security controls.	(1) Remote Anonymous User (3) Authenticated User

Assets

Assets are documented in this sample threat model as follows:

- **ID**: A unique ID is assigned to identify each asset. This will be used to cross-reference the asset with any threats or vulnerabilities that are identified.
- **Name**: A descriptive name that clearly identifies the asset.
- **Description**: A textual description of what the asset is and why it needs to be protected.
- **Trust Levels:** The level of access required to access the entry point is documented here. These will be cross-referenced with the trust levels defined in the next step

ID	Name	Description	Trust Levels
1	User	Assets that relate to a user	
1.1	Authentication Token	The token generated as a result of valid client ID. On successful verification of the client ID, the authentication token gets generated.	1) Remote anonymous user. 2) User with valid Client ID 3) An authenticated user.. 4) Administrator
1.2	Client ID	The client ID associated with a user for the respective platform. This can be used to connect the	1) Remote anonymous user. 2) User with valid

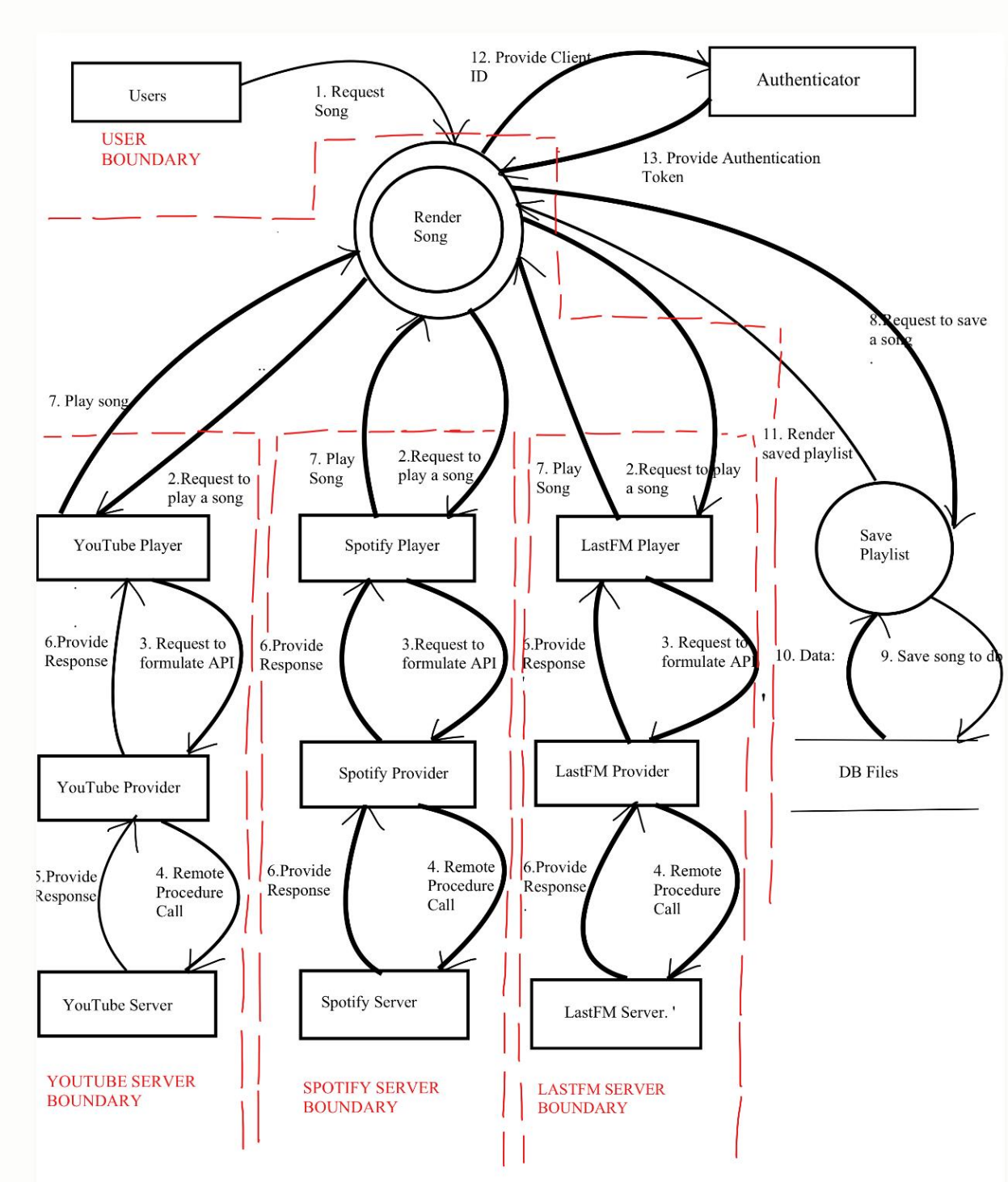
		user's account with Moosync	Client ID 3) An authenticated user 4) Administrator
1.3	Playlist db	The user's liked list of songs. This could get valuable insights on user's preference, listening preferences etc.	1) Remote anonymous user. 2) User with valid Client ID 3) An authenticated user 4) Administrator 5) General Contributor
2	Process	Assets that relate to the process running the app	
2.1	Ability to execute code as the identity of the associated server	This is the ability to execute source code as the associated server.	1) Remote anonymous user. 2) User with valid Client ID 3) An authenticated user
2.2	Ability to Execute query as a Database Read User	This is the ability to execute select queries on the dblite database, and thus read the playlist information.	1) Remote anonymous user. 2) User with valid Client ID 3) An authenticated user
2.3	Ability to Execute query as a Database Read/Write User	This is the ability to execute select queries on the dblite database, and thus update the playlist information.	1) Remote anonymous user. 2) User with valid Client ID 3) An authenticated user
2.4	Availability	The application must be available all the time whenever user requests for a particular song	1) Remote anonymous user. 2) User with valid Client ID 3) An authenticated user 4) Administrator 5) General

			Contributor

Trust Levels

ID	Name	Description
1	Remote Anonymous User	A user who is connected to the app but has not provided a valid Client ID to connect with external platforms.
2	User with valid Client ID	A user who has provided a valid Client ID.
3	An authenticated user	A user who has provided a Client ID and received an authentication token.
4	Administrator	An administrator that can configure the application
5	General Contributors	A developer who contributes to the source code

Data Flow Diagrams



DETERMINE THREATS

We choose the framework STRIDE to perform our analysis for determining the potential threats that the application might possess. One of the main reasons for choosing this is because this framework covers the CIA Triad along with Authentication, all of which are the foundation of security.

Below is the STRIDE Framework description [5] -

Threat	**Desired property**	**Threat Definition**
Spoofing	Authenticity	Pretending to be something or someone other than yourself
Tampering	Integrity	Modifying something on disk, network, memory, or elsewhere
Repudiation	Non-repudiability	Claiming that you didn't do something or were not responsible; can be honest or false
Information disclosure	Confidentiality	Someone obtaining information they are not authorized to access
Denial of service	Availability	Exhausting resources needed to provide service
Elevation of privilege	Authorization	Allowing someone to do something they are not authorized to do

Spoofing

Spoofing is a deceptive practice where an entity masquerades as another by falsifying data to gain an illegitimate advantage. Common types include:

- IP Spoofing: Forging the sender IP address in network packets.
- Email Spoofing: Sending emails with forged sender addresses.
- Caller ID Spoofing: Falsifying the phone number displayed on the recipient's caller ID.
- Website Spoofing: Creating a fake website that mimics a legitimate one to deceive users.

In context of this application, which is an open source github project where users are allowed to use the project, contribute directly to the project, resolve issues, report bugs and also integrate the application with other third-party applications. Every detail mentioned here offers an opportunity for the attackers to execute spoofing.

Addressing these one by one -

1. **Commit Spoofing**: A malicious user could attempt to make commits that appear to come from another developer by falsifying the author information in the commit metadata. This could be used to mislead others about who made changes to the project.

2. **Pull Request Spoofing**: Similar to commit spoofing, a malicious actor might create a pull request that appears to come from a trusted contributor by manipulating their Git configuration to use the trusted contributor's name and email.
3. **Account Spoofing**: An attacker could create a GitHub account with a username similar to a legitimate contributor to trick others into believing they are interacting with the trusted contributor.
4. **Repository Spoofing**: An attacker could clone the repository and make malicious changes, then promote the fake repository to deceive users into downloading and using the compromised code.
5. **Discord access**: An attacker can access the discord server which is mentioned on the website and github link for the project to help contributors communicate better, however, if an attacker is able to pretend to be someone else and jump on the server, he will be able to access all the communication and resources that contributors have, directly impacting the confidentiality of data. Furthermore, an attacker can use this data to his advantage and make changes in the code to install backdoors, inject malware etc.
6. **Owners/contributors profile spoofing**: There is a lot of user data accessible from Github. Owner and developer's profile and their personal information (which they choose to add) on github is visible. An attacker can utilize this information and execute identity spoofing, with respect to this application or even others. Recon through Github could lead to spoofing on this project or others.

For an open-source project, spoofing will be one of the biggest threats since accessibility of the project is very open. Hardening the security around this will be crucial to maintain the authenticity of the application and project.

Tampering

Tampering refers to making unauthorized alterations to the application which will directly impact the integrity of that application. In the context of an open source github project, Tampering can be a severe threat considering there are ways in which an attacker can access the code, drop malicious dependencies on the repository, change configurations etc.

Addressing these one by one -

1. **Code Tampering:**
    a. Directly modifying code: Unauthorized users might gain access to the repository and make changes to the source code, introducing bugs, vulnerabilities, or malicious features.
    b. Pull Request Tampering: Malicious actors might submit pull requests containing seemingly benign but actually harmful changes. If not reviewed thoroughly, these changes could be merged into the main codebase.
2. **Commit History Tampering:**
    a. Rewriting Commit History: If an attacker gains sufficient access, they could alter the commit history to hide malicious changes or remove important contributions.

    b. Force Pushes: Unauthorized force pushes can rewrite the repository's history, potentially introducing tampered commits and losing important historical data.
3. **Dependency Tampering**:
    a. Malicious Dependencies: Introducing or updating dependencies to malicious versions within the project's dependency management files (like package.json, requirements.txt, etc.).
4. **Configuration and Setting Tampering**:
    a. Repository Settings: Changing repository settings such as branch protection rules, access permissions, or webhooks to weaken security and facilitate further attacks.
    b. CI/CD Pipeline Tampering: Modifying continuous integration and deployment scripts to introduce malicious actions or disable critical checks.

Tampering directly affects the integrity aspect from the CIA Triad. Based on the above potential threats, the impact they will have in the context of this application considering integrity are -

1. **Introducing Malicious Code**: Introducing backdoors or security flaws that can be exploited later, leads to creating security vulnerabilities.
2. **Operational Disruptions by corrupting data:** Breaking functionality, causing outages, or disrupting the development workflow, changes that introduce bugs or corrupt data, leading to malfunctioning software.
3. **Data Loss**: Erasing or corrupting important code and data, which could lead to significant setbacks in development.
4. **Trust Erosion**: Undermining the trust of users and contributors if tampered code makes it into production or releases, causes direct impact on business by losing the trust of users and stakeholders.
5. **Altering Commit History**: Manipulating the commit history to hide changes, making it difficult to track what has been altered and by whom, this is a type of tampering which assists the attacker in repudiation, meaning, the attacker can deny he made changes as they are not traceable to him anyway.

Repudiation

Repudiation occurs when someone denies an action or agreement they previously acknowledged or performed. This lack of accountability arises when the user fails to accept responsibility for their initial involvement.

In our open-source GitHub project, repudiation can pose a threat by allowing harmful actions to be carried out using the app, followed by an inability to trace them back to their source.

- This creates a scenario where attackers can engage in malicious activities without the fear of consequences, as there's no clear path for legal accountability.
- Additionally, if a user is unaware of their app's vulnerabilities and inadvertently exposes others to threats, the app owner may later refuse responsibility in the absence of evidence tying the actions to them.

Based on the above scenarios of how Repudiation can occur in this open source application, below are the possible strategies this could be done in :

1. **Code Contributions:** The contributors to this application can either deny ownership of the code or claim a user's contribution was done without permission, which can be done with or without the code causing malicious harm.
2. **License Agreements:** The contributors can later deny having agreed to the terms of the open-source license. They can do this by stating that their project contribution was made under different circumstances, which may not be mentioned in the agreement, or that the project was never intended to be shared.
3. **Commit History:** The contributor can repudiate the claim that the changes made in the project’s commit histoy were actions performed because their github account was compromised. They can deny all changes attributed to them.
4. **Bug Reports:** The application contributors can deny bug reports that attribute to them by suggesting that their account was unauthorizedly accessed or that the bug report is fabricated.

Note: Evidence is required for the above threat actions, to prove their lack of accountability. For example, to claim an account was compromised, there must be minimum indicators of compromise for it or information on how their specific account details could have been accessed in the first place.

Information Disclosure

Information disclosure as a threat, directly impacts the confidentiality aspect of the CIA triad. This can occur by a user gaining access to unauthorized information or access to data in transit.

In terms of the open source github project, below are the potential risks associated with information disclosure :

1. **Hardcoded Secrets:** Developers inadvertently include sensitive information within their code. For example, credentials could be hardcoded and left there in the open source code forgetfully. (if they were used by developers during testing).
2. **Repository Misconfigurations:** When the repository has improper configurations, it may inadvertently expose sensitive information. Misconfigured repository permissions could allow unauthorized access to issues, pull requests, or other repository settings. This leads to revealing internal aspects or sensitive contributor/project details.
3. **Code Comments:** Comments within the codebase could unintentionally disclose sensitive information. Developers may leave behind implementation details that others can get design details from. If comments indicate deprecated or insecure functionality, they can inadvertently expose potential attack vectors.
4. **Accidental Commit of Confidential Data:** Developers may accidentally commit confidential information in code, including their GitHub credentials. This can be done by hardcoding as mentioned above. These commits can be accessed by anyone browsing the repository in the project’s history.

5. **External Dependencies:** This app depends on Youtube and Spotify optionally(based on user choices). Vulnerabilities found in these external applications can disclose information if not promptly addressed. Outdated dependencies may inadvertently expose sensitive information.
6. **Sensitive Issues Discussion:** If not careful, sensitive information can be disclosed on the project, its users or infrastructure through discussions within github issues. This can also happen through the Discord channel link the contributors have provided. Vulnerabilities can be exposed through these discussions as well.

Denial of Service

Denial of Service is a cyber-attack in which the attacker tends to make a resource unavailable to its users by disrupting services of a host connected to a network. This could be done by overwhelming it with a flood of requests. The goal of a DoS attack is to disrupt the normal flow of operations and eventually damage the system's reputation.

The open source project Moosync, does not involve any server of its own to handle its functions. So the possibilities of performing a Denial of Service attack lies in exploiting the vulnerabilities in code to make the application crash. As the application stores data in a local database, it can also be done by manipulating the database and making the application crash eventually.

1. **Manipulating the application's API:** An attacker might contribute to the application, a malicious script which makes the system perform intensive tasks or makes numerous API calls. This will degrade the performance of the application or make the application unavailable.
2. **Automated builds:** Since this is an open source application, the attacker can craft a script that runs excessive builds or automated tasks. When the code is then executed, whenever a pull request is submitted, the CI/CD pipeline is triggered. The malicious code will then make the app perform the same process multiple times. This will consume excessive resources. This will then make the application unavailable for all the users who download it.
3. **Database manipulation:** The application involves the usage of an SQLite local database. An attacker can try to inject malformed data that has the potential to corrupt database files. This might make the data unreadable from the database and eventually cause the application to crash when it tries to read the corrupted data and make the application unavailable for the users.
4. **File manipulation:** An attacker can try injecting code that will try to manipulate the files that are a part of the application and try to fill up the disk space. This will eventually consume a lot of system resources and make the application crash for users.
5. **Exhausting the CPU:** Attackers might introduce code to the open source project such that it requires the system to perform CPU intense activities. This will make the user's CPU resources to be fully utilized and thus lead to slow responsiveness and the application might become inaccessible to users.

Elevation of Privilege

Elevation of Privilege (EoP) refers to when someone gains access or privileges to resources that should not be made available to them. This poses a threat as they may be able to perform restricted actions. An example of this occurrence is a buffer overrun attack that attempts to write executable code.

There are two primary types of Privilege Escalation Attacks:

1. **Vertical Privilege Escalation**: Attackers attempt to gain access from a standard user account to an account with higher privileges such as admin privileges which would grant them access to restricted areas and allow them to control parts of the overall system such as modifying configurations.
2. **Horizontal Privilege Escalation**: Instead of an attacker attempting to gain access to a higher level privilege, the attacker attempts to gain access to part of the system on the same privilege level. For example, trying to gain access from an employee's stolen credentials.

In the context of our open-source application, anyone can analyze the source code for vulnerabilities as well as contribute to the project itself and introduce malicious code. As defined earlier, there are three types of users that are relevant:

1. User who want to use the application (direct use or develop APIs and use custom version)
2. Users who wish to contribute for the project's development (open source Github project)
3. Main developers who update the project

As a result, there is no need for escalation directly, however, there are still EoP vulnerabilities that exist:

1. **Credential Exploitation**: An attacker can accomplish credential theft via password reuse attacks. If an attacker gains access to passwords of an administrator Github account, then they can compromise configurations or user data.
2. **Inadequate Permission Management:** Inadequate procedure to delegate permissions for who can modify code or how permissions can be granted can lead to EoP vulnerabilities. An attacker can abuse this vulnerability to compromise an account with editing permissions and introduce malicious code to exploit unauthorized access. This applies to both the permissions needed in editing code as well as permissions needed to approve code changes.
3. **Code Contributions:** Attackers can contribute code to the project that may appear benign if not carefully reviewed, but actually exploit EoP vulnerabilities.
4. **Extension API Vulnerabilities**: Moosync allows anyone to develop custom apps on top of Moosync Extension API and publish them to the store. Malicious extensions developed could exploit vulnerabilities to gain unauthorized access to data or other functionalities.

Threat analysis and ranking

For analyzing and ranking the threats which are determined above using the STRIDE model, this section will include usage of the DREAD model for assessing the risks associated with every threat outlined above. The ranking will not be numeric in nature even though the DREAD model

is referred to. This model is used to structure the findings but determination of the risk associated with threats will be categorized as - High, medium and low. [Qualitative analysis]

The questions used for determining the ranking are:

- **D**amage – how bad would an attack be?
- **R**eproducibility – how easy is it to reproduce the attack?
- **E**xploitability – how much work is it to launch the attack?
- **A**ffected users – how many people will be impacted?
- **D**iscoverability – how easy is it to discover the threat?

Final table containing potential threats and their rankings on qualitative analysis using DREAD model -

Index	Threat	Ranking
1	Pull Request Spoofing	High
2	Discord Access	High
3	Code tampering	High
4	Commit history tampering	High
5	Dependency tampering	High
6	Configuration and settings tampering	High
7	Code Contributions	High
8	Credential exploitation	High
9	Manipulation the application's API	High
10	Automated builds	High
11	Commit history	High
12	Repository Misconfigurations	High
13	Commit spoofing	Medium
14	Account Spoofing	Medium

15	Owners/contributors profile spoofing	Medium
16	Extension API Vulnerabilities	Medium
17	Hardcoded Secrets	Medium
18	Code Comments	Medium
19	Accidental Commit of Sensitive Data	Medium
20	External Dependencies	Medium
21	File Manipulation	Low
22	Database Manipulation	Low
23	Exhausting the CPU	Low
24	License Agreements	Low
25	Bug Reports	Low
26	Sensitive Issues Discussion	Low

DETERMINE COUNTERMEASURES AND MITIGATION

To address the threats outlined above, we will assist the application team by providing detailed guidance on implementing appropriate controls and measures. This process will involve:

1. **Risk Assessment**: Evaluating the associated risks and business impacts.
2. **Resource Allocation**: Analyzing the trade-offs of the resources required to handle or mitigate these threats.

Once we identify the potential impacts, we can address the risks using the following strategies:

- **Accept**: Determine that the business impact is acceptable and document the decision-maker responsible for accepting the risk.
- **Eliminate**: Remove components that contribute to the vulnerability.
- **Mitigate**: Implement checks or controls to reduce the risk's impact or likelihood of occurrence.
- **Transfer**: Shift the risk to an insurer or customer.

This structured approach will ensure that we effectively manage and mitigate the identified threats.

Spoofing

1. **Commit Signature Verification: Mitigate**
    - Contributors should cryptographically sign commits to add a layer of verification.
    - Developers can enable vigilant mode to display the verification status of all commits, aiding in the detection of impersonation attempts.
2. **Diligent Verification: Mitigate**
    - Verify the identity of contributors and do not rely solely on metadata.
    - Implement manual introspection alongside automated tools to validate pull requests.
3. **Pull Request Review Process: Mitigate**
    - Ensure all contributions from less-trusted parties are reviewed by at least two maintainers.
    - Validate the presence of a Contributor License Agreement (CLA) and ensure code passes unit tests, style requirements, and security checks.
4. **Red Team Exercises: Mitigate/Eliminate**
    - Conduct “red team” exercises to test the effectiveness of code review processes in detecting malicious, obfuscated changes.

Tampering

1. **Automated Security Checks: Mitigate**
    - Run security tools on all pull requests before merging to detect vulnerabilities.
    - Use tools like Microsoft Application Inspector and the NPM Security Insights API for thorough reviews of changes to the component's attack surface or core characteristics.
2. **Validation of Build Configurations: Mitigate**
    - Validate any changes to build configurations similar to other code changes.
    - Limit the duration and frequency of pull request validation routines to avoid denial of service or resource exhaustion.
3. **Package Management Security: Mitigate**
    - Use package signing mechanisms (e.g., Ubuntu PPAs, NuGet Package Signing) to ensure authenticity.
    - Employ repository signing to mitigate risks from compromised mirrors.
    - The Go ecosystem’s notary mechanism ensures cryptographic checksums of modules do not change after initial publication.
4. **Static and Dynamic Analysis: Eliminate**
    - Conduct static and dynamic analysis within the publishing pipeline to detect unwanted activity.

- Analyze installation script code and behavior, leveraging maintainer reputation as an input.
- Extend analysis to include the entire package.

5. **Communication and Automated Analyses: Mitigate/Eliminat**e
    - Maintain regular communication between security teams across major package management systems.
    - Conduct automated analyses for malicious code patterns upon package publication.

Repudiation

Preventing repudiation in Moosync, an open-source application on GitHub, can be achieved by implementing methods to ensure that actions performed within the app can be tracked and verified. This will prevent users from denying their actions:

1. **Code Contributions:** Signing GitHub commits using GPG keys helps verify that the commits are from a trusted source. Similarly, signing release tags with GPG ensures the integrity of the release.
2. **License Agreements:** The use of Contributor License Agreements (CLAs) will help formalize contributions and ensure that contributors acknowledge their actions. This can be further secured with digital signatures to verify the sender's identity.
3. **Commit History:** Multi-factor authentication for contributors adds an extra layer of security, ensuring that only authorized users can perform actions. Clearly defining roles and permissions for all users and contributors is essential to avoid confusion, especially since more than one contributor is involved in the Moosync app.
4. **Bug Reports:** The regular review of reports can help handle reports from being ignored. When filing a bug report, ensure the contributors are logged into their github accounts before submission.

Information Disclosure

Adopting certain habits can prevent information disclosure:

1. **Hardcoded Secrets:** This includes regular code audits to help identify and remove hardcoded secrets to prevent leaks.
2. **Repository Misconfigurations:** Mindful management of repository permissions limits access to sensitive information. Being aware and implementing the required configurations in github that help secure the app.
3. **Accidental Commit of Confidential Data:** Sensitive data like log files, configuration files or any other files can be excluded when pushing into the github repository by using a .gitignore extension on those files/directories.
4. **Code Comments and Accidental Commit of Confidential Data :** Following clear guidelines of what is to be in the comments and not, and enforcing them with regular checks would help in avoiding unintentional disclosure of sensitive information through

comments. Auto-sanitization scripts can be implemented to clean comments when sensitive data is found. This can be an automated process triggered by new commits.

5. **External Dependencies:** Information disclosure through third party apps such as Youtube and Spotify can be prevented by reviewing the third party app limitations on the Moosync app. The use of vulnerability scanning tools are also helpful. Ensuring APIs accessed by third party apps are secured by strong authorization.
6. **Sensitive Issues Discussion:** There should be contribution guidelines for what information can be exposed publicly and what cannot. Examples of sensitive information are API keys, personal data, and proprietary code. Discord channels used for this app must allow users to enter the channel on permission or maintain a private communication channel with trusted members only for sensitive information discussions.

Denial of Service

1. **Rate Limiting the requests: Mitigate**
    - Implement code that would check the number of API requests made in a particular time frame.
    - If the number of requests are above the threshold value, the requests must be blocked.
2. **Build Optimization: Mitigate**
    - Implement mechanisms to handle the automated builds in an effective way to consume minimal resources.
    - As it is an open source project, proper code review must be done to optimize the CI/CD pipeline to handle the automated builds.
3. **Input Sanitization: Mitigate**
    - Implement sanitization mechanisms to avoid code that would modify the data that is stored in the database.
    - Integrity checks must be implemented at regular intervals to make sure the existing data is not corrupted.
4. **File Access Controls: Mitigate**
    - Implement proper access control mechanisms for accessing the files associated with the application.
    - Integrity checks must be implemented to ensure that the files are not tampered with corrupt data.
5. **CPU Monitoring: Mitigate**
    - Alerts should be implemented to notify when there are changes in the trends of CPU usage.
    - Implement mechanisms to limit the resource consumption of CPU to a predefined limit.

Elevation of Privilege

Since Moosync is an open source project with three main types of users, the most effective mitigation and countermeasure are thorough code reviews to maintain the integrity of the

application. A strong review process with multiple developers of the application should be established to approve pull requests before code is merged. This would reduce the chances of malicious code being introduced into the codebase. Other measures such as multi-factor authentication (MFA) can be used for main developers to prevent unauthorized approvals of code merges. Establishing a strong review process is the best way to maintain community engagement on an open source project while preventing harmful code being introduced to the codebase. Additionally, a proper procedure needs to be put in places for requesting and granting permissions. Permissions should be granted on the basis of least privilege and only be given the necessary permissions for the task at hand.

1. **Code Reviews: Mitigate**
   Since this is an open source project, thorough code reviews are crucial in maintaining the integrity of the application. A strong review process with multiple developers of the application should be established to approve pull requests before code is merged. This would reduce the chances of malicious code being introduced into the codebase.
2. **Multi-factor authentication (MFA): Mitigate**
   Implement MFA for roles of higher-privilege such as administrative roles or trusted developer to approve code changes as a countermeasure to credential exploitation.
3. **Run with least privilege: Mitigate**
   Grant users with minimum necessary permissions for task completion. This will apply more to the main developers of the application and the special permissions needed access sensitive data for configurations. This will also include who has permissions to approve pull requests from outside contributors.
4. **Sandboxing: Mitigate**
   Running the extensions in a sandboxed environment with limited permissions and access to unauthorized resources will limit the potential damage of malicious extensions.

REFERENCES

1) Main Application website: https://moosync.app/wiki/
2) Github open source project link: https://github.com/Moosync
3) Privacy policy for the application: https://www.iubenda.com/privacy-policy/70189564/full-legal
4) OWASP Threat Modelling guide: https://owasp.org/www-community/Threat_Modeling_Process
5) STRIDE Model: https://en.wikipedia.org/wiki/STRIDE_model
6) DREAD Model: https://en.wikipedia.org/wiki/DREAD_(risk_assessment_model)
7) https://resources.github.com/security/open-source/how-github-secures-open-source-software/
8) https://www.darkreading.com/application-security/how-attackers-could-dupe-developers-into-downloading-malicious-code-from-github
9) https://github.com/ossf/wg-metrics-and-metadata/blob/main/publications/threats-risks-mitigations/v1.1/Threats%2C%20Risks%2C%20and%20Mitigations%20in%20the%20Open%20Source%20Ecosystem%20-%20v1.1.pdf
10) https://learn.microsoft.com/en-us/windows-hardware/drivers/ifs/elevation-of-privilege