CORE

WHITE PAPER

API SECURITY:

COMMON VULNERABILITIES, BEST PRACTICES, AND SOLUTIONS

Application Programming Interface (API) is essential to many business operations. APIs connect different systems and enable applications to communicate with each other, making it easier for businesses to share data and provide services to their customers. However, APIs can also pose significant security risks if not adequately secured.

In this whitepaper, we will discuss the best practices for API security that can help businesses protect their systems and data from malicious attacks.

May 2023 | gcore.com

Understanding APIs

API stands for Application Programming Interface. In the context of APIs, an "application" refers to any software with a specific function. An API interface can be considered a service contract between two applications. This contract defines how applications interact with each other using requests and responses. The API architecture is usually explained from the client and server perspective. The application that sends the request is called the client, and the application that sends the response is called the server.

APIs can be classified into different categories based on their application areas:

- **Private APIs:** Internal APIs of organizations are only used to connect systems and data within the business.
- **Public APIs:** APIs with public access that anyone can use.
- **Partner APIs:** APIs available only to authorized third-party developers to assist in partner relationships between businesses.
- **Composite APIs:** APIs that combine two or more different APIs to meet complex system requirements or behaviors.

Best Practices for API Security

1. Educate on Common Vulnerabilities and Attacks

It is crucial to be aware of common vulnerabilities and attacks that APIs are susceptible to. Educating your development and security teams on these issues can help them design, build, and maintain secure APIs:

- Manipulating query parameters
- Extracting excess data from responses
- Disrupting service in the absence of rate limiting or response limiting constraints
- Accessing unauthorized administrate functions by guessing endpoints
- Stealing credentials
- Exploiting security misconfigurations
- Injecting malicious code
- Gaining entry through test versions or deprecated API versions
- Exploiting Detection Delays

2. Start With Secure API Design

One of the earliest stages in the API lifecycle is its design. You must ensure your model does not expose confidential data to applications and potential attackers. To do this, ensure your API only provides users with what they need. This can mean reducing the amount of information sent in responses but also reducing information in requests.

Incorporating security measures into API design is a more practical approach than implementing them on an already operational API. The OWASP Application Security Verification Standard (ASVS) is a good source for all types of application designs, and it is a useful resource to help guide secure API design. **Be sure to follow its recommendations, including:**

- Draft security requirements for building and integrating APIs.
- Include business logic in design reviews.
- Draft secure coding and configuration practices relevant to your technology stacks.

3. Prepare API Documentation

Documentation is vital for the application and API teams building or integrating APIs. Sufficient documentation also benefits a range of activities, including design reviews, security testing, operations, and protection.

To make documentation more efficient and effective, it is recommended to use machine-readable formats such as the OpenAPI Specification (OAS). This enables the use of automated tools in the API development process, making it easier for developers to validate and test their APIs. In addi-tion, the API schema can be reused as a basic testing and protection approach, saving time and resources during the development process.

However, it is essential to have a contingency plan for documentation discrepancies and API drift. Documentation discrepancies can arise from API changes, leading to testing and protection issues. Therefore, it is crucial to have a plan in place to ensure that documentation remains up-to-date and accurate.

4. Implement Security Testing of APIs

Testing the security of your API is crucial in identifying vulnerabilities and ensuring the security of your system. You can use traditional security testing tools to check for known vulnerabilities, but remember these tools have limitations. No scanner can analyze business logic, making organizations vulnerable to API abuse.

You should also perform automatic static code analysis of your API within the version control and CI/CD framework. Check for known vulnerable dependencies in your API code.

Perform dynamic analysis and fuzzing of deployed APIs to identify usable code during runtime. This process involves generating many random inputs to the API to identify potential vulnerabilities that may not have been identified during the development or testing phases. By performing dynamic analysis and fuzzing, you can identify potential security vulnerabilities and ensure the overall security of your API.

5. Choose the Authentication Method

Several authentication methods are available, and choosing the right one for your organization is critical to the security of your API. Some of the most common authentication methods are:

- **Basic HTTP Authentication:** A simple authentication method that sends a username and password with each API request. This method is easy to implement but does not have robust security features and is, therefore, unsuitable for high-risk applications.
- **Access Tokens:** A commonly used method that generates a token passed with each API request to authenticate the user. This method is more secure than simple HTTP authentication but requires additional infrastructure to manage and revoke access tokens as needed.
- **API Key Authentication:** This method uses a unique key provided to each user and must be supplied with each API request. This method is easy to implement and provides a level of security but is less secure than other methods, such as access tokens.
- **JSON Web Tokens (JWT):** A popular method that uses a digitally signed token to authenticate the user. This method provides strong security features and is suitable for high-risk applications but can be more complex to implement.
- **OAuth 2.0:** A protocol that allows for delegated access to APIs. This method enables users to grant third-party applications access to their API resources without sharing their credentials. This method is commonly used by social media platforms and other applications that allow for integration with third-party applications.
- **OpenID:** A protocol for authentication that allows for single sign-on across multiple applications. This method is commonly used by enterprise applications that require users to authenticate across various systems.
- **SAML/SAML 2.0:** A protocol for exchanging authentication and authorization data between parties. This method is commonly used in enterprise environments and allows secure communication between systems.

When choosing an authentication method, it is essential to consider factors such as ease of use, security, scalability, and compatibility with your existing systems. By selecting the proper authentication method, you can ensure the security of your API and protect against potential security threats.

6. Implement Access Control and Management

To ensure the security of your APIs, it's crucial to implement effective access control and management. In addition to traditional authentication methods, we recommend adding extra layers of access control by implementing access control management at the API level.

This is particularly important for modern applications and systems that handle a mix of confidential, non-confidential, and highly confidential data.

One effective way to accomplish this is through API access management, which allows you to apply additional authentication factors when a consumer attempts to use a particular API that could be responsible for transmitting sensitive data. In the case of banking applications, for example, where the stakes are high, additional access control management can help protect sensitive customer data.

You can more effectively control who has access to what data and how that data can be used by using access control management, reducing the risk of unauthorized access or data breaches. With proper access control, you can more confidently manage the security of your APIs and protect your business from costly security incidents.

7. Ensure Proper API Inventory Management

API discovery and inventory management are critical steps to ensuring the security of your APIs. It is essential to identify all APIs that are currently in use, both in production and development environments, to manage and secure them effectively.

Organizations can use automated tools to discover and create an inventory of APIs. Once you have identified all APIs, tracking them throughout their lifecycle, from development to retirement, is crucial to ensure they remain secure and properly documented. This can help prevent the launch of phantom APIs and reduce the risk of vulnerabilities being introduced into your system.

Additionally, conducting regular security assessments and penetration testing can help identify potential vulnerabilities in your APIs, including those that may have been introduced through undocumented or unsanctioned API deployments.

8. Set Up API Monitoring

We recommend ensuring you have set up API monitoring to detect anomalous traffic, identify suspicious consumer behavior, and alert system administrators of potential risks.

- Record all authentication and authorization failures.
- Log request details that can be used to quickly identify the source of attacks using API security tools.
- Properly format logs so they can be filtered and reported through a log management platform.
- Treat logs as confidential data, as they contain information about both your users and API vulnerabilities.
- Continuously monitor infrastructure and generate monitoring reports prioritizing information most important to API security.

9. Ensure Unified API Protection Across the Entire Organization

Ensuring unified API protection across the entire organization, including managing synchronous and asynchronous APIs, is crucial. Organizations must be able to apply access control, authentication, monitoring, and other security measures to both their synchronous and asynchronous APIs.

Synchronous APIs, which are used for real-time communication between systems, and asynchronous APIs, which are used for delayed or batched communication, require adequate protection measures to mitigate potential security risks. This includes implementing strong access control mechanisms to restrict access to APIs based on the roles and privileges of users or applications, as well as using effective authentication methods such as OAuth or JWT to verify the identity of consumers and prevent unauthorized access.

By implementing a unified approach to API protection across your organization, you can ensure that all APIs are properly secured and monitored, reducing the risk of data breaches or other security incidents that could impact your business operations or reputation.

10. Follow OWASP API Security Top 10 Best Practices

OWASP API Security Top 10 is a comprehensive list of modern API-driven applications' most critical security risks. We recommend you follow these security measures:

- **A1: Broken Object Level Authorization.** Ensure secure object-level authorization by implementing checks based on user policies and hierarchy, avoiding client IDs in favor of session-stored object IDs, hardening server configuration, checking authorization for each client request to access the database, and avoiding random guessable IDs (UUIDs).
- **A2: Broken Authentication.** To prevent broken authentication, verify all API authentication methods, ensure password reset APIs and one-time links are protected and allow authentication, implement standard authentication with token generation, secure password storage and multi-factor authentication, use short-lived access tokens, and implement stricter rate-limiting, lockout policies, and weak password checks for authentication.
- **A3: Excessive Data Exposure.** Ensure secure data exposure by avoiding client-side data filtering, adapting API responses to consumer needs, defining request and response schemas in the API specification, clearly defining error responses, justifying sensitive or PII information, and enforcing response checks to prevent accidental data and exception leaks.
- **A4: Lack of Resources and Rate Limiting.** Configure rate limiting appropriately for API method, client, and addresses while considering payload limit, compression ratio, and computing/container resources.
- **A5: Broken Function Level Authorization.** Ensure correct implementation of functional level authorization by denying all access by default, disabling unnecessary features, not relying on the application to enforce admin access, granting roles based on specific roles, verifying rate limiting, and correctly implementing authorization in the API.
- **A6: Mass Assignment.** Verify that APIs precisely define accepted request schemas, types, and patterns at design time, do not automatically bind incoming data to internal objects, explicitly define expected parameters and payload, and use the readOnly set to true for all properties that can be retrieved via APIs but should never be modified.
- **A7: Security Misconfiguration.** Verify that APIs are securely configured with repeatable hardening and patching processes, automated configuration flaw detection, disabled unnecessary features, restricted administrative access, and defined and enforced outputs and authorization.
- **A8: Injection.** Verify API is not trusting internal API consumers, strictly define input data and validate, filter, and sanitize all incoming data, and limit and enforce API outputs to prevent data leaks.
- **A9: Improper Assets Management.** Verify proper asset management by creating an inventory of all API hosts, limiting access to non-public resources, segregating production and non-production data, implementing external controls like API firewalls, retiring old versions, and implementing strict authentication and redirects.
- **A10: Insufficient Logging and Monitoring.** Ensure proper logging and monitoring of API activity, including failures, validation errors, and security policy checks, with logs formatted for easy consumption, protected as sensitive data, and integrated with SIEM and monitoring tools for identification and alerting purposes.

By implementing Gcore Web Security and its Web Application Firewall, your API will receive comprehensive protection against all of these critical security risks.

How to Apply an API Security Checklist in Practice

Now that you have a security checklist that can serve as a basis for assessing the security of your API, here's how to apply this knowledge in practice:

1. Review past incidents and/or attack attempts.
2. Stay in touch with the security team to identify existing potential vulnerabilities.
3. Develop and agree on appropriate security measures to address potential vulnerabilities.
4. Check which tools and/or new features are necessary to implement proper security measures.
5. Implement security measures.
6. Regularly monitor the security status of your API using the steps outlined above.