IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

EMPIRICAL STUDY ON API SECURITY THREATS & EXPLOITATION OF RATE LIMITING FLAW

[1]Amit Gawande, [2]Aniruddha Gayake, [3]Mohit Charkha, [4]Saraunsh Shewale, [5]Prof. Kirti Wanjale

[1-4]B. Tech Student, [5]Associate Professor

[1-5]Department of Computer Engineering,

[1-5]Vishwakarma Institute of Information Technology, Pune, India

Abstract: APIs today securely and safely exchange a vast amount of data and resources, including payment information, banking information, healthcare information, and databases of all sorts. APIs are the means by which developers use data and functionality to build new apps and integrate with digital ecosystems, and they're critical to delivering the modern digital experiences that consumers and partners expect. Since it's arguably a requirement for businesses to expose their services through APIs in order to stay relevant and competitive in today's market, it's important that they do so safely — by limiting who has access to APIs, monitoring and managing how APIs are used, deploying features that can detect suspicious activity and otherwise make monitoring data actionable, and so on. Bad actors can easily launch denial-of-service (DDoS) campaigns and other similar attacks, but rate limiting can help an organization keep attackers at bay.

Keywords - API, Security, Rate Limiting, REST, SOAP, Injection Attacks, Man-in-the-Middle.

I. INTRODUCTION

A software middleman called an Application Programming Interface (API) allows your applications to communicate with one another. It contains protocols, routines, and tools for software developers to use while constructing applications., as well as allowing for the easy extraction and sharing of data. While Web APIs are interfaces that enable apps to interact with other services or platforms, such as social networks, games, databases, and gadgets. Internet of Things (IoT) applications and devices also use APIs to collect data and even control other devices. For example, a power company, might utilize an API to save energy by altering the thermostat's temperature. SOAP and REST are two well-known API implementation methods.

SOAP (Simple Object Access Protocol) is an XML-based communications protocol for exchanging data between computers. In SOAP's built-in WS-Security standard, XML Encryption, XML Signature, and SAML tokens are used to resolve concerns about the security of transactional communications. SOAP also follows the OASIS and W3C standards.

REST uses the JSON protocol to receive API payloads, which speeds up data transmission across browsers. REST is stateless, which means that each HTTP request contains all necessary data, and neither the client nor the server must save any data in order to complete the request. Unlike SOAP, which requires parsing and routing for each request in order for a local web service to function, HTTP does not require this.

II. ABBREVIATIONS & ACRONYMS

IoT: Internet of Things, API: Application Programming Interface, HTTP: Hyper-Text Transfer Protocol, REST: Representational State Transfer, SOAP: Simple Object Access Protocol, MITM: Man in The Middle, XSS: Cross Site Scripting, SQLi: Structured Query Language Injection.

III. API SECURITY THREATS

APIs most of the times have self-document information concerning its implementation and internal structure, which can be used for cyber-intelligence. API attacks are vulnerable mentioned below due to additional weaknesses. Inadequate authentication, lack of encryption, business logic problems, and unsafe endpoints are just a few examples to quote.

3.1 Man in the Middle Attack

In a man in the middle (MITM) attack, an attacker discreetly relays, intercepts, or modifies communications between two parties, including API messages, in order to get sensitive information. For instance, a criminal can play the role of a man in the middle between an API issuing a session token in an HTTP header and a user's browser. Intercepting that session token would grant access to the user's account, which might include personal details, such as information of the credit card and login information(credentials).

3.2 API Injection Attacks

An injection of code assault involves injecting malicious code into a software application that is vulnerable. Cross-site scripting (XSS) or SQL injection (SQLi). Are examples of attacks. To conduct an XSS attack against end users' browsers, a malicious script can be injected into a vulnerable API by a perpetrator, such as one that fails to execute the necessary filter input and output, or one that fails to execute the right escape output (FIEO).

Additionally, malicious commands, for example an SQL Query to delete the tables from the database, could be included inside an API message. Any online API that uses parsers or processors is at risk of being hacked. For instance, a code generator that includes JSON code processing, and does not sanitize input properly, is vulnerable to executable code that executes in the development environment being injected.

3.3 Rate Limiting

Rate limitation is a technique for controlling the amount of traffic flowing in and out of a network. For example, let us say you are using a service's API that is configured to allow 100 requests/minute. An error will be raised if the number of requests you make exceeds that limit. The goal of putting rate limitations in place is to improve data flow while also increasing security by preventing DDoS attacks.

Rate restriction is also useful if a network user makes a mistake in their request, causing the server to obtain a large amount of data, perhaps overloading the network for everyone. These kind of problems or attacks are significantly more manageable with rate restriction in place.

When it comes to setting rate restrictions, there are numerous ways and factors that can be used. The rate limit approach you should choose will be determined by your goals as well as how restrictive you want to be. The next section describes three different sorts of rate restriction techniques that you might use.

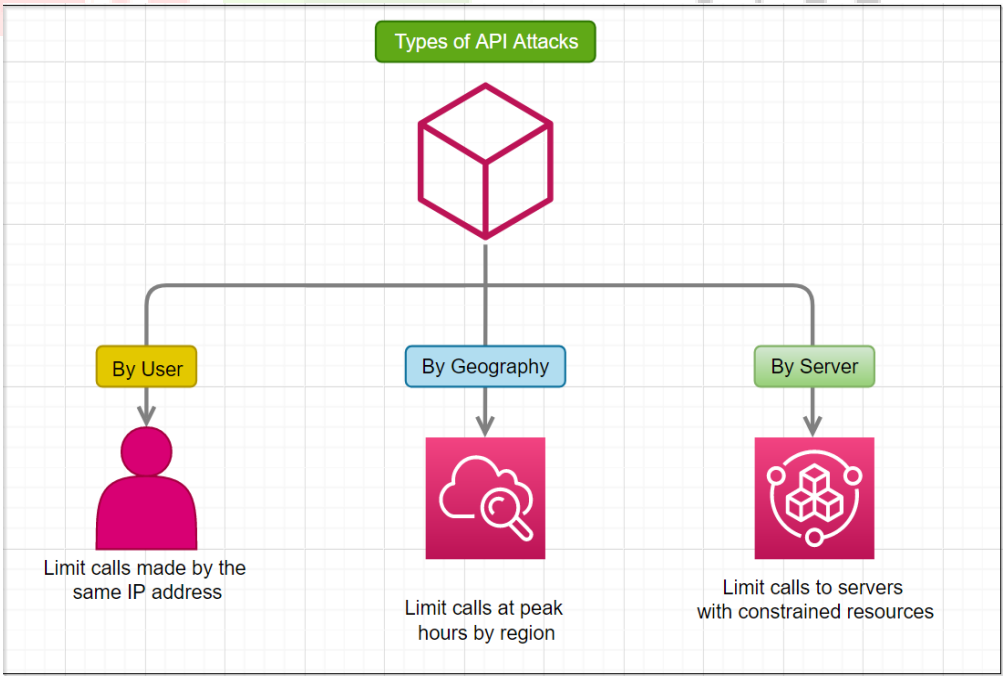

Figure 1: Types of API Rate Limiting Attacks

i. User rate limitation: User rate limiting is the most common sort of rate limiting. This identifies how many requests a user has made to their API key or IP address (depending on which method you use). As a result, if a user exceeds the rate limit, any subsequent requests will be rejected unless the user contacts the developer to seek a higher limit or wait until the rate limit timeframe refreshes.

ii. Geographic rate limitation: Developers can define rate limits for specific regions and time periods to strengthen security in certain geographic areas. For example, if a developer understands that users in a given location will be less active from midnight to 8:00 a.m., they can set lower rate limitations for that time period. This could be utilized as a precaution to assist lessen the danger of attacks or suspicious behavior.

iii. Server rate limitation: If a developer has designated specific servers to handle specific portions of their application, at the server level, rate limitations can be imposed. This allows developers to lower traffic limitations on server A while raising them on server B. (a more commonly used server).

IV. Exploitation Of The Bug

Rate limiting is a technique for limiting the amount of traffic flowing into and out of a network. Let us assume you're using an API for a specific service that's set to allow 100 requests per minute. An error will be triggered if the number of requests you make exceeds that limit. There is no way to defend against the requests you made in a limited period of time if there is no rate threshold.

In this context, we have a forgot password functionality which is very mainstream in modern designed web as well as mobile applications. The feature prompts the user for the registered email address which is part of that platform. Once the email address is stated, application validates if the user is registered on the system and if it is not registered then application will throw an error and request will be dropped out. However, if the email address is valid and registered, the application simply sends out an email with password reset token link to the specified email address.

Now to check if the application has some sort of rate limiting constrains deployed, we can attempt to register user with any random email address as the web application does not asks for any sort of email verification. So, any number of dummy accounts can be created on this vulnerable application without even verifying integrity of the user. After successful registration of the user on the application, let us verify if there's rate limiting constrains deployed on forgot password functionality. In order to verify this, first setup a web intercepting proxy tool which can intercept HTTP requests. In this case, we have set up a proxy tool called as burp suite. Now, enter victim's email address and click on submit button. Send intercepted request by burp suite proxy tool to the intruder tab and set repetition count for the same request as 50 repetitions of the request in one go.

After sending these many requests, check the email inbox to verify if it has received 50 password reset tokens. As there is no constraint on the API usage, requests will be approved by the server. Due to this critical misconfiguration, a denial of service or (DoS) state can be introduced while keeping legitimate users away from using the resources of the application and thus causing huge capital loss for business corporations.

All of these manual checks can be automated using a simple python program that includes the victim's email address and the amount of password reset tokens that need to be sent to his inbox. The python program acquires the email address and count to request the vulnerable API service. Count can be any non-negative number ranging from 0 to any large number. In return to the API requests, successful response is returned by the associated application server.

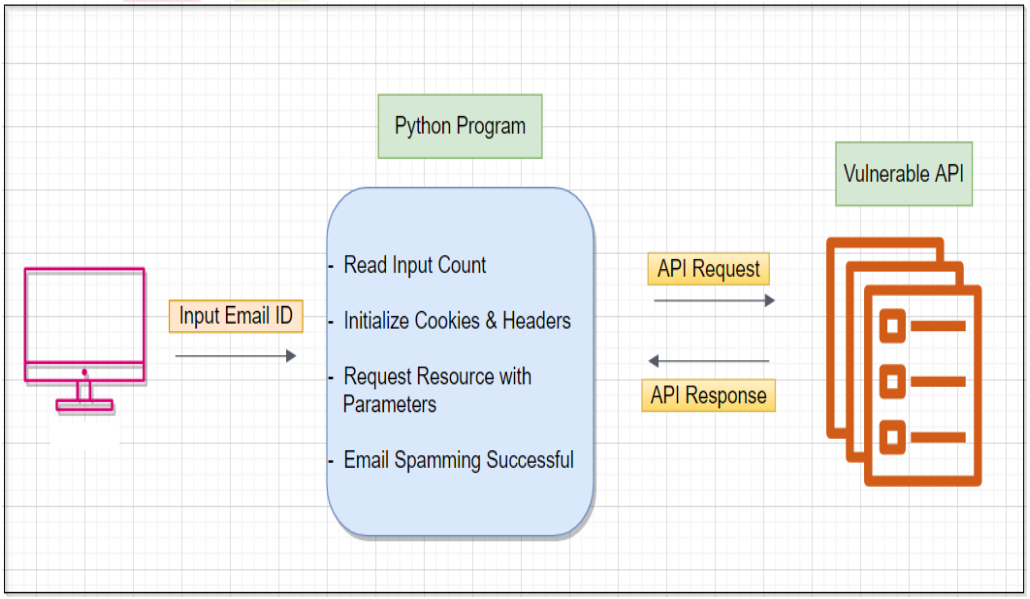

Figure 2: Types of API Rate Limiting Attacks

V. PREVENTING API ATTACKS

A well-designed API security toolset provides protection in depth against attack threats. API Gateway comes with a content firewall that can detect malicious content including viruses and malformed JSON or XML data structures. The Gateway reduces the possibility of parameter attacks, business logic attacks, SQL injection, and XSS attacks by detecting and blocking these troublesome API calls. The Gateway may also build whitelists to reduce the chances of an attack from an untrustworthy source. Attack avoidance happens on two occasions in the case of Denial of Service.

API providers should set limits on the amount and frequency of API calls made by a single app. If the API becomes overwhelmed with requests or is subject to unusual patterns of API calls from the network, they can track usage and submit warnings to system administrators. The API Gateway not only has these features, but it also has a licensing feature that allows API owners to establish contractual agreements with apps, including pay-for-API-use terms. DoS attacks are significantly reduced when licensing is combined with rate limiting and QoS control.

5.1 Encrypting the Message Channel

Encryption for API security must be widely used and adaptable. SSL/TLS encryption should be the default configuration for all APIs, according to API security providers. All API providers must enable SSL as it provides highly effective security against "man in the middle" attacks. A built-in PKI and key distribution scheme can secure the privacy of user data with advanced encryption and signing features.

5.2 Integrating Secure API Practices into Software Development & Deployment

Authentication and authorization allow a company to determine who can use which API. What about the number of times your API is utilized by a given app or user? What's more important, how is it being put to use? Excessive API usage can be just as destructive and costly to an API provider as not using it at all. Without the capacity to monitor the API's activities and usage, it's impossible to create the capacity to control the API's Quality of Service (QoS). You should be able to monitor an API's performance and availability, as well as that of its users, using your API management system.

5.3 Monitoring & Auditing API Traffic

API security will not be ideal without a complete, policy-based strategy, which should be obvious to infosec specialists. Using a disjointed toolkit and haphazard security standards would almost surely result in security vulnerabilities and unneeded risks. Organizational APIs, as well as the apps that connect to them, should be managed by a consistent set of security policies across their entire life cycle. Architects and developers should consider the dependencies, authentication concerns, data integrity concerns, and other difficulties that may affect the API once it is constructed and deployed at the planning stage. The API policies defined during development should be followed.

VI. CONCLUSION

Despite the fact that API risks may have a significant economic impact, they can be addressed. For firms that are serious about information security, protecting APIs from threats is a manageable task. API security will, in most circumstances, be an extension of existing security policies and controls, with some new components thrown in for good measure. APIs do create certain uncommon security concerns, but they are unable to address them due to the planned nature of the access they provide to external users.

With the right API security toolkit, even these threats can be prevented. Security managers can effectively mitigate API risk by managing API access, limiting API usage, monitoring quality of service, and encrypting API requests and responses when a company's attention is directed to API security. It's a procedure that should cover the complete API lifecycle, from planning to development to deployment.

REFERENCES

[1] Riccardo Focardi, Flaminia L. Luccio, "An Introduction to Security API Analysis" Jan 2011.
[2] R. Anderson "The correctness of crypto transaction sets. In 8th International Workshop on Security Protocols", April 2000.
[3] R. Anderson. "What we can learn from API security (transcript of discussion). In Security Protocols, pages 288–300". Springer, 2003.
[4] R. Anderson "Security Engineering Wiley, 2nd edition" 2007.
[5] A. Armando and L. Compagna "SAT-based model-checking for security protocols analysis. Int. J. Inf. Sec., 7(1):3–32" 2008.
[6] O. Berkman and O. M. Ostrovsky, 11th International Conference, Financial Cryptography and Data Security (FC 2007), Scarborough, Trinidad and Tobago, pages 224–238" February 2007.
[7] B. Blanchet. "From secrecy to authenticity in security protocols. In M. Hermenegildo and G. Puebla, editors, International Static Analysis Symposium (SAS'02), volume 2477 of Lecture Notes in Computer Science, pages 342–359 Madrid, Spain" September 2002.
[8] M. Bond and R. Anderson "API level attacks on embedded systems. IEEE Computer Magazine, 34 67–75" October 2001.
[9] C. Cachin and J. Camenisch. "Encrypting keys securely. IEEE Security & Privacy, 8(4):66–69, 2010.
[10] J. Clulow. "The design and analysis of cryptographic APIs for security devices. Master's thesis, University of Natal, Durban" 2003.
[11] D. Longley and S. Rigby "An automatic search for security flaws in key management schemes. Computers and Security, 11(1):75–89" March 1992.
[12] D. Knuth "The Computer as a Master Mind. Journal of Recreational Mathematics, 9:1–6" 1976.
[13] G. Keighren "Model checking security APIs. Master's thesis, University of Edinburgh" 2007.
[14] J.A. Goguen and J " Meseguer. Security policies and security models. In IEEESymposium on Security and Privacy, pages 11–20" 1982.
[15] B. Schneier "Applied Cryptography. John Wiley and Sons, 2nd edition" 1996.