International Journal of Research Publication and Reviews

Journal homepage: www.ijrpr.com ISSN 2582-7421

To Detect and Prevent Vulnerabilities in Microservices and Application Programming Interface

[1]T Anusha, [2]G Naga Manmitha, [3]Sharon Kota, [4]S Laxman Rao, [5]N Ramesh Kumar, [6]P Ganesh

[1,2,3,4,5,6] Department of CSE, GMR Institute of Technology, Rajam. Andhra Pradesh, India

Abstract:

Microservices also known as microservice architecture is a procedure designed for service-oriented architectures (SOA). These services are utilized in the development of software systems which are elastic and can be deployed independently by breaking down an application into various smaller units. The benefits of this mechanism include development of application in a distributed manner, integration of systems which are heterogenous in nature, extensibility, and adaptability. Microservices are completed based on the functioning of APIs. Here, API refers to Application Programming Interface which is an intermediary software between different software or computer programs facilitating communication among each other. For instance, when any kind of application is connected to the Internet through a device (computer, smartphones, etc.), and transmits information to server, the server fetches this data, processes it and remits it to the device. The main motive of this paper is to achieve secured microservices, with the usage of technologies like React JS, Java along with Buckets and Tokens and Spring Boot Framework. It ensures the minimization of harmful effects of an attack. As a result, an application with Microservice framework using Spring Boot is built which provides additional security measures for API vulnerabilities, gives an extra layer of protection and defense to the system, and mitigates the scope of attacks in microservices.

Keywords: Microservices, Kubernetes, API, Spring Boot Framework, Docker, Buckets and Tokens.

I. INTRODUCTION

The service-oriented architecture i.e., the microservice architecture is the sole reason for the change in the development, deployment, and testing of applications. As the total application is broken into multiple fragments of microservices, the system does not allow a single point of entrance. These services function with APIs, which acts as a main source of communication between two or more computer systems or programs. The primary goal of this study is to find a solution that can be implemented in microservices in order to mitigate and prevent attacks and to minimize the exploitation of loopholes in microservices. Throughout this process, Spring Boot Framework is used to develop discrete applications in a simple and effective manner. A web application is developed with the usage of Spring Boot Technology acting as a backend technology. Exploiting the vulnerabilities in the computer system can be minimized by implementing various security mechanisms which can be obtained by using Spring Boot Framework. The outcome of the paper is making and deployment of an e-commerce application, carrying out all manner of attacks to find the loopholes in the system and to provide additional security to the application in order to overcome the vulnerability exploitation. To prevent and mitigate the attacks on microservices, wide range of security mechanisms must be incorporated.

[1] The main contribution in the improvement of service availability is the feature of microservices i.e., the services being lightweight and small. As all the applications which are based on microservices are stateful, every instance of microservice has an independent state, which should be known to other microservice instances to supervise when the system fails. In order to get rid of this problem, stateful microservice based applications are frequently deployed as instances of a microservice which stores the states in external databases making them virtually stateless. Kubernetes is the leading orchestration platform for applications which are containerized. A cluster of Kubernetes can be consisting of different physical or virtual machines and adheres to the master-slave architecture. To ensure the high availability of cluster management, the master node and all the processes should be duplicated. Controllers are entities that manage the pods' lifecycle and are in turn responsible for the creation and maintenance of the number of pods required according to the prerequisites mentioned in the pod template in the resource specification of the controller. Kubernetes comprises of different types of controllers, each of which is used for a distinct purpose and functionality. It cannot differentiate between the failure of a node, network partitioning, and to avoid having multiple pods running on different nodes with the same identity. It does not automatically create a pod to replace the one on the node which is not responsive. Unlike the controllers of a Stateful Set, pods deployed by Deployment controllers are automatically rescheduled on other nodes when their hosts fail. In case of any failure, the failed pod should be repaired for the service to be recovered. Measurement of the availability metrics for the failed pod as well as the scaling time for the deleted pod is performed and compared with the availability metrics of these experiments with those where no scaling event had happened during failover. The evaluations clearly show that integrating this solution improved the recovery of services by 50% on average. Moreover, it is measured that the scaling overhead when integrated with Kubernetes in the range of 7.5% and 10.5%.

[2] The existence of Microservices in software development is due to someone in some business department who found a risk on their articles and place the right question. Digging into the past and analysing the present, it is easy and effortless to perceive why this technology is used extensively and a commercial business to vendors like Microsoft and Amazon. Due to the limitations of conventional methodologies e.g., Monolithic Architecture, the microservices are a success. Despite the colossal usage and years of maturation, scalability was achieved first as a success condition to the prevailing advancement of Microservices, its architecture and technologies. An architecture is something that permits a given subject to evolve in a way which is trustworthy and secure. This allows to extend the gained knowledge base regarding the subject which is given in a way that consists of all the obligatory aspects to a solid and effective transformation. Providing security in Microservices has limitless challenges related to trust and security. It an important aspect that Microservices are designed regularly to trust on their peers. The exploitation of one can result in a full leverage over the information and the disclosure of all the others. It is significant to promote mechanisms and services which enforce the developers, end-users, and administrators to incorporate unique, time stamped and complex passwords. Every character of a potential password must be checked securely to avoid any usage of names, letters, sequence numbers, previous strings, and account names. When dealing with security, it is to be done in a broad sense of its aspects regardless of a risk which is apparently low. Servers need to maintain SSL (Secure Socket Layer) certificates which is troublesome in the case of a multi-machine, and it is essential to handle complex and risky issuing processes and few certificates are difficult to revoke e.g., self-signed certificates. An appropriated solution in Microservices is the use of single-sign-on implementations. SAML and OpenID is perfect for Authentication and Authorization of someone who is on a system but it is also great for service-to-service authentication as well.

[3] This paper presents Microservices Integrated Performance and Reliability Testing (MIPaRT), a cutting-edge methodology and support platform to execute ex-vivo testing sessions for continuous integrated performance and reliability analysis of microservice systems automatically. MIPaRT leverages data in the system and its usage from past Ops phases to automate the formation and implementation of performance and reliability tests at a decision gate. It then estimates and visualizes Key Performance Indicators (KPIs) to pinpoint problems (e.g., faulty microservices). Research and practice typically focus on the assessment of each quality attribute in a separate manner. Recent literature comes up with various ML techniques and symbolic execution methodologies to detect appropriate input values that can determine reliability or performance issues separately. Their application can be very expensive in an environment of DevOps, and sometimes are not more accurate than simple random input generation. Using them for reliability and performance in combination may be even more expensive. Several tools, mechanisms and tehniques have been proposed as Automated reliability testing of microservices is of paramount importance in DevOps. DevOpRET is a technique to estimate reliability at the acceptance testing stage in DevOps cycles. A framework for resilience testing to assess the ability of a microservice system to recover from failures is also proposed. Terminus, to estimate the capacity of a microservice, defined as the maximum number of successfully processed user requests per second, on different deployment configurations via load tests, and fitting a regression model to the acquired performance data. Both new releases (evolutionary) and usage profile (operational) changes are tracked and monitored in a DevOps process. Thus, to automatically generate the operating conditions, we first need the following data in a session log: session identifier, request start time, request end time, request relative path and combinations of valid and invalid values for the arguments of each request. Once the DTMCs are generated, the frequency associated with DTMC is computed as frequencies of sessions in clusters over all sessions. MIPaRT captures variation of such issues under operational changes. With this platform, developers can also have further insights on the nature of the issues of between performance and reliability and how such issues can be affected by a specific operational change.

[4] Microservice architecture (also known as MSA) as an emerging and upcoming development paradigm in software engineering brings new security threats, risks, and vulnerabilities. These threats may be of two types: internal and external threats. The internal threats come from insiders and the external attacks occur due to outsiders. For proper and effective securing of microservice-based systems, all threats, and loopholes, regardless of their origin, need to be detected and prevented using either available mitigation techniques or through advancing innovative solutions. The results show that 63% of primary studies focus on external attacks, only 13% focus on internal attacks and 24% focus on both source of threats. This clearly demonstrates an unbalanced research focus towards external attacks. Relationships between security mechanisms and threats are mandatory and crucial. In addition to this, mapping security mechanisms to their applicability architectural levels and platforms is necessary. Cloud-focused and platform independent solutions are found to have higher rates of 34.78% and 28.26%, respectively. The interest to cloud computing can be understood due to different facilities and provisions supplied to companies by adopting MSA for developing their applications. Adopting MSA for developing applications in the cloud allow companies to integrate existing legacy systems, to grow with demands and to use up-to-date and intuitive interfaces. Solutions provided for IoT applications are also getting more attentions due to the specificity and the growing needs to those applications in the market. However, more attention should be paid to 5G platforms. These are emerging technologies that requires specific attentions. A systematic mapping on securing microservices focusing on threats, nature, applicability platforms, and validation techniques of security proposals is done. The study examined 46 papers published since 2011. The results revealed that unauthorized access, sensitive data exposure and compromising individual microservices are the most treated and addressed threats by contemporary studies. It also revealed that auditing, enforcing access control, and prevention-based solutions are the most proposed security mechanisms. Additionally, we found that most proposed solutions are applicable at soft infrastructure layer of MSA. It is shown that 34.78% of papers proposed MSA security solutions that work for different platforms, the same proportion is noticed for cloud-based solutions. Finally, we found that most verification and validation methods were based on performance analysis, and case studies.

[5] In methods which are object-oriented, separated concerns are modelled as objects and classes, which are generally derived from the entities in the requirement specification and use cases. The key unit of modularity in OOPs is the class. Usually, one class is not enough to provide the module functionality. Even though Object Oriented Programming (OOP) is easy to understand and provides decent decoupling between the objects, it is not enough to address the common concerns that are applicable throughout the application as it affects the entire solution. This principle states that every module should have a single responsibility, with which all its services should be narrowly aligned. One of the major driving forces behind any kind of architectural solution is scalability. There are different ways of decomposing the application into services. One such approach is to use verb-based decomposition and define services that implement a single use case as checkout. The other option is to decompose the application by noun and create services responsible for all operations related to a particular entity such as customer management. An application might also use a combination of verb-based and noun-based decomposition. Data is partitioned (also called as sharded) across a set of servers based on an attribute of each record. A client can authenticate to the API Gateway with a username and password combination using HTTP Basic Authentication. Basic Authentication is not considered to be a secure method of user authentication (unless used in association with some external secure system such as SSL), as the username and password are passed over the network as cleartext. One of the benefits of JWTs (Java Web Token) is that they can be used without a backing store. All the information required to authenticate the user is contained within the token itself. In a distributed microservice world, it becomes easy to not rely on centralized authentication servers and databases. In the JWT, only the Authorization header is base64-encoded and signed, so if anyone were to get the JWT token and request, they can then update the HTTP Request body. To avoid this kind of data manipulation issues, HTTP Signatures are used. The benefit of signing the HTTP message for the purposes of end-to-end message integrity is so that the client can be authenticated using the same mechanism without the need for multiple loops.

II. METHODOLOGY

What is Microservice?

The Concept of microservices refers to applications that are broken up into small (micro) services, rather than being developed as a single large monolith. APIs use these microservices to communicate, forming a smooth user experience. The main reason behind this approach is based on Agile Philosophy - Which is growing in popularity. The development of this method is to help teams deliver value to their customers faster, which makes it's a lot easier to release software updates continuously if the codebase in question is smaller and more straightforward. But today's ever more powerful computer applications don't lend themselves to having small and simple codebase. Microservices' fixes consequently become a huge growth area.

Vulnerabilities:-

A vulnerability of an application is defined as a weakness that can be designed as a flaw or an implementation of a bug, or a configuration of an application, which allows a hacker to cause harm to the stakeholders of the application. The vulnerability in an application can be such by of Lack of input validation on user input, sufficient logging mechanism, Fail-open error handling, or not closing the database connection properly.

Here are some Common Application Vulnerability Exploits:

1. Broken Access Control
2. Cryptographic Failures
3. Injection
4. Insecure Design
5. Security Misconfiguration
6. Vulnerable and Out-dated Components
7. Identification and Authentication Failures
8. Software and Data Integrity Failures
9. Security Logging and Monitoring Failures
10. Server-Side Request Forgery

The Common Weakness Enumeration (CWE) provides information on specific instances of a particular issue.

Broken Access Control

Access control enforces policies such that users cannot act outside of their intended authorization. Failure in such matters typically leads to unauthorized information disclosure, modification, or destruction of all data or performing a business function outside the user's limits, where third party might have access to use them for miscellaneous purposes

Cryptographic Failure

This is where the need to determine the protection of data in transit and at rest. For example, passwords, credit card numbers, health records, personal information, and business information secrets require extra protection, mainly if that data falls under the privacy laws, e.g., EU's General Data Protection Regulation (GDPR) or regulations, financial data protection such as PCI Data Security Standard (PCI DSS).

Injection

An attacker exploits the failure of the web application to filter data provided by users before it inserts that data into a server-side. Due to vulnerabilities in the code of an application, it allows the attacker for invalidated user input. An application is vulnerable to attack when, User-supplied data such as personal details are not confirmed, filtered, or sanitized by the application.

Some of the common injections are SQL, NoSQL, OS commands, ORM, LDAP, EL, or OGNL injection. The concept is identical among all interpreters. Reviewing the applications source code is the best method of detecting if applications are vulnerable to injections or to figure out what are the possibilities for injections. So, automatically testing all the parameters, header URL, cookies, and JSON, SOAP, and XML data inputs is strongly encouraged.

Insecure Design

It's a broad category of representing different weaknesses, expressed as "missing or ineffective control design". It's not the source for all other risk categories. But there is a difference between insecure design and insecure implementation. We differentiate them based on their design flaws and implementations defects for a reason; they have individual root cause and remediation. A secure design can still have implementation defects leading to vulnerabilities that may be exploited. An insecure design cannot be fixed with an implementation as by definition, by means it needs security controls which were never created to defend them against specific attack in the first place.

As for the one of the factors that contribute to insecure design, which lacks the business risk profiling inherent in the software, or system being developed. Thus the failure of the system is determine by what level of security design is required.

Security Misconfiguration

This misconfiguration of security leads the application to be vulnerable to simple and easy attacks by hackers. The Misconfiguration of an application can be anything such as, Unnecessary features being enabled or installed, Default accounts and their passwords are still enabled and unchanged, Error handling reveals stack of traces or other overly informative error message to users, as for the upgraded systems – the latest security features not being enabled or not configured securely.

The application's server not sending security headers or directives by which are supposed to send, or they are not set to secure values. Or having the Software out dated of latest security patches. Without a sold secured configured system, or repeatable application security configuration process the systems that being used are at a considerable risk.

Vulnerable and Out-dated Components

Using out-dated versions of components leads the application to be vulnerable. So, it is important to know the versions of all the components we use or this might lead to have compatibility issues with both Client-side and Server-side. If the software itself is vulnerable, unsupported or out of date. Then it causes the issues to the OS web-application server, database management system, applications, APIs and all components, runtime environment, and library issues.

Identification and Authentication Failures:

The Confirmation of the user's identity, credential authentication, and session management is critical to protect against identity theft or authentication-related attacks. There may be authentication weaknesses if the application allows automated attacks such as credential stuffing, Brute force attacks or any other automated attacks. Authenticated credentials with default, weak or well-known passwords may also cause such sort of attacks to happen. Using weak and ineffective recovery or forgot-password processes such as "knowledge-based answers", are not safe. This sort of attacks may also be caused due to the usage of plain text, encrypted or weakly hashed passwords, data stores, or by having missing, ineffective multi-factor authentication.

Software and Data Integrity Failures

The software and data integrity failures mainly relates to code and infrastructures that does not protect against integrity violations. This might occurs when an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks. An insecure pipeline can introduce the potential risk of unauthorized access with malicious code, or compromising security systems. By using this sort of auto-update functionality where updates are downloaded without checking or performing sufficient integrity verification and allow them to be applied to previously trusted application may give the access to the attackers then attackers could potentially upload their own updates to be distributed as a trusted source and runs on all installations.

Security Logging and Monitoring Failures

This type helps in detecting, escalate, and respond to active breaches. Without proper log data and monitoring date, breaches cannot be detected at any time auditable events, such as logins, failed logins, and high-value transactions, are not logged. Warning error generate no, inadequate, or unclear log messages. Logs of the application and APIs are not monitored for suspicious activity. Logs are only stored locally. A proper alerting thresholds and response escalating processes are not in place which is effective. The penetration testing and scans by dynamic application security testing tools. This kind of application cannot be detected escalate, or alert for active attacks in real-time or near real-time.

Server-Side Request Forgery

The Server-Side Request Forgery flaws occur whenever a web application is fetching a remote resource without confirming the user-supplied URL. Which makes it possible for the attacker to exploit the application and allows sending a crafted request to an unexpected destination at the server, even when the server system is protected by security system, VPN, or another type of network access control list?

As modern web applications provide end-users with convenient features, fetching a URL addresses is a common scenario. As a result, the inside of Stress is indications of SSRF are increasing. And again the severity of SSRF is becoming higher due to cloud services and the complexity of architectures.

How to prevent such vulnerabilities while developing Applications

1. It's crucial to maintain a security-first policy when developing and configuring software.
2. Except for public resources, denying access by default.
3. Making sure of applying protection horizontally and vertically. Vertical protection involves employing the least privilege concept where in access is granted only by them.
4. Centralizing all the authentication decisions to minimize the occurrence of access-related web application vulnerabilities.
5. Using model access control to ensure record ownership instead of granting access for users to freely create, read, modify, or delete any records. Removing their ability to read or modify data from other users.
6. Implementing a one-time access control mechanism in security policies.
7. Setting limitations to API and controller access to reduce the risk of attacks by automated tools.
8. Invalidate JWT tokens and user sessions on the server after logout.
9. Disabling webserver directory listing data transmission and preventing the web roots from storing backup files and file metadata.

We need to be aware of such vulnerabilities and prevent of occurrences while developing an application. Doing so will help in avoiding a large number of security threats. However, this are not only the vulnerabilities that can deal risk to our applications, there are plenty other harmful web application vulnerabilities. Being aware of all the vulnerabilities will help us in enhancing security and protect valuable data against security threats.

III. CONCLUSION

The study completely discussed about the vulnerabilities that are present in the more than 70% of the microservices that are currently in use. These vulnerabilities are basically much more helpful for the attackers to exploit into the web application which may cause much more effect to the users. There are several attacks that are noticed in the microservices now-a-days which were clearly explained above , all these attacks basically uses the vulnerabilities and exploit into ones private premises using the unauthorised authentication credentials and then misuses the usage for gaining the access either to the sever or to the sensitive data of the personal. This causes damage to the CIA (confidentiality, integrity and availability). The study also gives many more ways to protect one's personal web applications or the credentials like there are several ways like using strong passwords, using the up-dated software's etc. The conclusion of the study put us at a point where developing the complete vulnerable free applications the only cause suitable in providing the security to the microservices. The development of the web application must be happened with the framework which can resist any sort of attacks hence using the Spring Boot Framework is the best one in developing the complete vulnerable free application.

References:

[1].Vayghan, L. A., Saied, M. A., Toeroe, M., & Khendek, F. (2021). A Kubernetes controller for managing the availability of elastic microservice based stateful applications. Journal of Systems and Software, 175, 110924.

[2]Mateus-Coelho, N., Cruz-Cunha, M., & Ferreira, L. G. (2021). Security in microservices architectures. Procedia Computer Science, 181, 1225-1236.

[3]Camilli, M., Guerriero, A., Janes, A., Russo, B., & Russo, S. (2022, May). Microservices integrated performance and reliability testing. In Proceedings of the 3rd ACM/IEEE International Conference on Automation of Software Test (pp. 29-39).

[4] Hannousse, A., & Yahiouche, S. (2021). Securing microservices and microservice architectures: A systematic mapping study. Computer Science Review, 41, 100415

[5]Salibindla, J. (2018). Microservices API security. International Journal of Engineering Research & Technology, 7(1), 277-281