Volume: 10 Issue: VIII Month of publication: August 2022

DOI: https://doi.org/10.22214/ijraset.2022.45147

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 10 Issue VIII Aug 2022- Available at www.ijraset.com

REST APIs Cloud Service Security Checks

A. Sanjana[1], M. Anusha[2], G. Pravallika[3], Mrs. S. Radhika[4]
[1, 2, 3]*B-Tech, CSE, Sridevi Women's Engineering College, Hyderabad*
[4]*Assistant Professor*

Abstract: REST APIs are used by the majority of modern cloud and web services. This paper explains how an attacker can take advantage of REST API flaws to gain access to a service. We introduce four new security rules that take advantage of REST API's and then demonstrate how a stateful REST API fuzzer can be extended with active property checkers that automatically test and identify violations of these rules. Then we talk about how to implement such checkers in both ways modularly and efficiently. Using these tools, we discovered new bugs in a number of production Azure and Office365 cloud services and discussed their security implications and resolve all these issues.

I. INTRODUCTION

The use of cloud computing is exploding rapidly. Over the course of the past few years, thousands of new cloud services have been put into operation by cloud platform providers like Amazon Web Services [2] and Microsoft Azure [13], as well as by their customers who are "digitally transforming" their businesses by modernising their workflows while simultaneously collecting and analysing a wide variety of new data.

As of today, the majority of cloud services can be accessed via REST APIs [9]. Cloud resources can be created (PUT/POST), monitored (GET), managed (PUT/POST/PATCH) and deleted(DELETE) using REST APIs, which are built on top of the HTTP/S protocol.

The Swagger (recently renamed OpenAPI) interface description language can be used by cloud service developers to document their REST APIs and generate sample client code [25]. For example, it specifies the format of responses and the types of requests that a cloud service can handle in its REST API, which can be accessed via Swagger.

How safe are these APIs? Most people haven't figured it out yet. In the early stages of development, REST API testing tools are still in their infancy. Some tools for testing REST APIs collect live API traffic before parsing, fuzzing, and replaying it in the hopes of detecting flaws [4], [21], [6], [26], [3]. As of late, stateful REST API fuzzing [5] has been proposed as a way to better test REST API-based services. A Swagger specification of a REST API is used to automatically generate sequences of requests, rather than single requests, in order to thoroughly test the cloud service deployed behind that API, with the goal of finding unhandled exceptions (service crashes) that can be detected by a test client as "500 Internal Server Errors". Despite the encouraging results of that research, its scope is limited to the identification of unhandled exceptions.

In this work, we present four security principles for REST APIs and services that capture desired qualities.

1) *Use-after-free Rule:* A resource that has been deleted must no longer be accessible.
2) *Resource-leak Rule:* A resource that was not created successfully must not be accessible and must not "leak" any side-effect in the backend service state.
3) *Resource-hierarchy Rule:* A child resource of a parent resource must not be accessible from another parent resource.
4) *User-namespace Rule:* A resource created in a user namespace must not be accessible from another user namespace.

Violations of these rules provide an attacker the ability to take control of shared cloud resources, or steal data from other users in the event of an attack, or to corrupt the status of the backend service.

Using a stateful REST API fuzzer, we demonstrate how these rules may be tested and detected. The active property checker for each rule creates API queries to test for particular rule violations and discovers any rule violations. Aside from checking for any rule violations, each checker actively attempts to breach its own rule. A modular approach is discussed so that checkers don't interact with one another.

Beyond "500 Internal Server Errors" stateful REST API fuzzer can check uncovered security rule violations with constructing. We discovered new bugs in numerous live Azure and Office 365 cloud services by using these checkers and fix them.

The following are the contributions of this paper:
1) Here in the paper, we present a set of rules that describe REST API security.
2) To ensure that these rules are being followed, we build and install active checks.
3) A number of Azure and Office 365 cloud services were identified to have new security vulnerabilities by using these checkers.

II. OBJECTIVES

A. We define tools for testing cloud services automatically via their REST APIs and checkers property.
B. Check whether the services are reliable and secure.
C. We also identify bugs and fix them using REST API's.

III. METHODOLOGY

A. Stateful Rest API Fuzzing

As far as we're concerned, REST APIs are the best way to reach the cloud by providing security. A client programme sends requests to a service and gets replies back in the form of responses from the service. A secure protocol called HTTP/S is used to transmit these types of communications. When a response is received it is connected with a specific HTTP status code that is either2xx, 3xx, 4xx, or 5xx .

Swagger is a interface description language famously known as OpenAPI which is used to describe REST API's. For RESTful services, Swagger specifies what kind requests may be made and what replies can be returned. The response format is also specified.

Requests are defined as part of the REST API. In order to fulfil the needs of the user, each request is made up of the following four elements <a, t, p, b> where
1) a is an authentication token,
2) t is the request type,
3) p is a resource path, and
4) b is the request body.

B. Security Checkers for Rest APIS

We present four security rules for REST APIs and services that capture their desired qualities. All four rules are based on real-world errors discovered via manual penetration testing or root cause analysis of customer-visible issues in previously deployed cloud services. New problems discovered in production Azure and Office 365 services as a result of rule violations will be discussed and the bugs will be fixed.

With the help of active checkers we apply the rules. As the main driver of stateful REST API fuzzing explores state space, an active checker offers additional tests for ensuring that certain rules are not broken. Therefore, an active checker broadens the scope of the search area by carrying out additional tests with the intention of breaking certain rules.

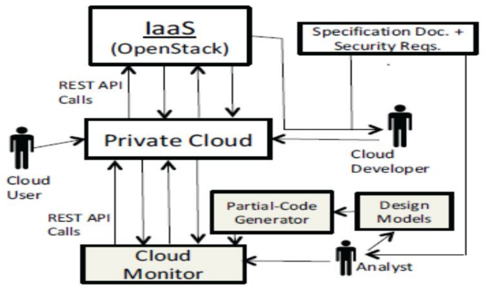
We create active checkers using a modular layout that corresponds to the following two principles:
1) Checkers are not dependant on the main contributor of stateful REST API fuzzing, and their presence has no impact on the latter's state space exploration.
2) Checks are not reliant on one another in any way, and they produce tests by examining the requests that are carried out by the main driver while ignoring the requests that are carried out by other checkers.

- Use-after-free checker
- Resource-Leak Checker
- Resource-Hierarchy Checker
- User-namespace checker

Combination of all these checkers along with REST API Fuzzer executes the code. Stateful REST API fuzzing is extended by checkers as follows:
- ➢ Expansion of the state space by conducting extra tests,
- ➢ By looking for replies other than 5xx that might be a sign of rule-violation issues. Consequently, the main driver's bug-finding skills are significantly improved using checkers.

IV. RESEARCH AND DESIGN

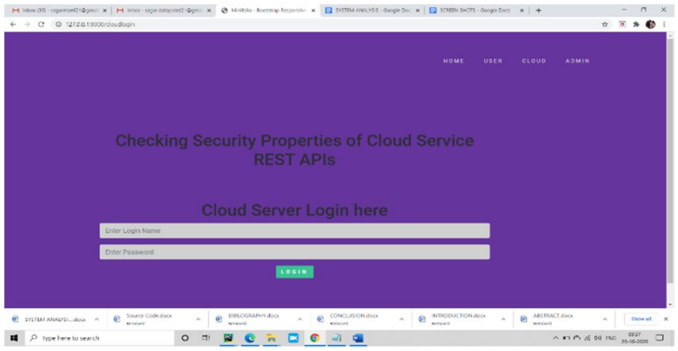
In the project cloud developer develops a private cloud using Infrastructure as a service(Iaas) through REST API's. For each and every request and response patter the request is evaluated using checkers property for building a model then the request is analysed. Based upon the analysed result the partial code will be generated. After the successful verification the cloud monitor provides the access to the cloud user considering it as a valid request through REST API'S in the form of a response i.e in 2xx format. If it is a invalid request then the cloud monitor restricts from access for unauthorized users by giving a response 3xx /4xx format. If a bug is found it produces 5xx response format.



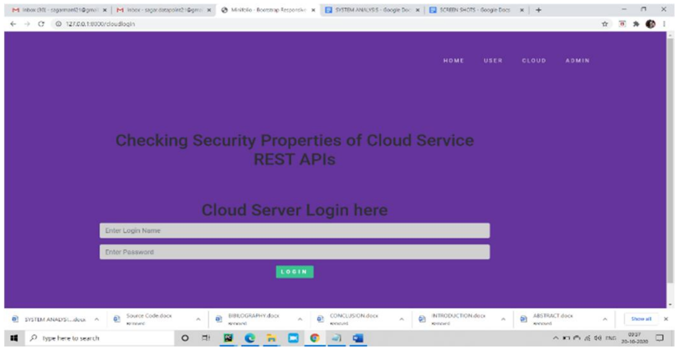
V. RESULT AND DISCUSSION

A. Result

Here we are running the code to display the page, where the user can register with the required credentials. Then the user details are approved and activated by the admin. Admin consists of user details, which are given by the user. User can also create an app to store his/her text files or documents. User has get approved from the cloud to activate the app.



This is the home page of user. Here, User can upload the files in app. And also can view the files. User can only upload files, edit/update and access the files only in user page. User is not allowed to edit or upload files except download files from cloud and admin.



This is the cloud server login page. Cloud server activate the user app and generates secret key. Every user has unique secret key. By the key user's data is protected by the cloud server from other users.

B. Discussion

The REST API'S is the latest emerging technology and have no much information about the technology. Besides the lack of information there is a huge usage of it. And our work in this paper purely represents the creativity to solve new problems.

VI. CONCLUSION AND FUTURE SCOPE

A. Conclusion

In the paper we discussed four new security rules along with REST APIs. Also we discussed about how stateful REST API's along with active checkers property automatically test and detect violations of the rules. Using the fuzzer and checkers that are detailed in this article, our team has successfully tested and fuzzed many Azure and Office 365 cloud services. Our fuzzing was successful in finding roughly a handful of newly introduced flaws in each of these services in virtually all of the situations. Approximately one third of those issues are violating the defined rules that were reported by our new security checkers, and others are 500 Internal Server Error bugs. We conclude that proprietors of the service about all of these issues, and they have now got resolved. These, violations of the four security standards outlined in this study represent a possible security risk. And the current bug discovery ratio is 100% and are fixed immediately.

B. Future Scope

In future, Fuzzing additional REST APIs and inspecting more rules will help us find out if there are any other kind of flaws or security issues. Its remarkable that so little information exists on how to use REST APIs safely in light of the recent development of REST APIs for cloud and online applications. Toward that end, this research contributes four security-relevant criteria that makes difficult for an attacker break the rules.

REFERENCES

[1] S. Allamaraju. RESTful Web Services Cookbook. O'Reilly, 2010.
[2] Amazon. AWS. https://aws.amazon.com/.
[3] APIFuzzer. https://github.com/KissPeter/APIFuzzer.
[4] AppSpider.https://www.rapid7.com/products/appspider.
[5] V. Atlidakis, P. Godefroid, and M. Polishchuk. RESTler: Stateful REST API Fuzzing. In 41st ACM/IEEE International Conference on Software Engineering (ICSE'2019), May 2019.
[6] BooFuzz. https://github.com/jtpereyda/boofuzz.
[7] Burp Suite. https://portswigger.net/burp.
[8] D. Drusinsky. The Temporal Rover and the ATG Rover. In Proceedings of the 2000 SPIN Workshop, volume 1885 of Lecture Notes in Computer Science, pages 323–330. Springer-Verlag, 2000.
[9] R. T. Fielding. Architectural Styles and the Design of Network-based Software Architectures. PhD Thesis, UC Irvine, 2000.
[10] P. Godefroid, M. Levin, and D. Molnar. Active Property Checking. In Proceedings of EMSOFT'2008 (8th Annual ACM & IEEE Conference on Embedded Software), pages 207–216, Atlanta, October 2008. ACM Press.
[11] K. Havelund and G. Rosu. Monitoring Java Programs with Java PathExplorer. In Proceedings of RV'2001 (First Workshop on Runtime Verification), volume 55 of Electronic Notes in Theoretical Computer Science, Paris, July 2001.
[12] R. Lammel and W. Schulte. Controllable Combinatorial Coverage in Grammar-Based Testing. In Proceedings of TestCom'2006, 2006.
[13] Microsoft. Azure. https://azure.microsoft.com/en-us/.
[14] Microsoft. AZURE DNS Zone REST API https://docs.microsoft.com/en-us/rest/api/dns/zones/get.
[15] Microsoft. Microsoft Azure Swagger Specifications. https://github.com/Azure/azure-rest-api-specs.
[16] Microsoft. Office. https://www.office.com/.
[17] S. Newman. Building Microservices. O'Reilly, 2015.
[18] OAuth. OAuth 2.0. https://oauth.net/.
[19] OWASP (Open Web Application Security Project). https://www.owasp org.
[20] Peach Fuzzer. http://www.peachfuzzer.com/.
[21] Qualys Web Application Scanning (WAS). https://www.qualys.com/apps/web-app-scanning/.
[22] SPIKE Fuzzer.http://resources.infosecinstitute.com/fuzzer-automation-with-spike/.
[23] Sulley. https://github.com/OpenRCE/sulley.
[24] M. Sutton, A. Greene, and P.Amini. Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley, 2007.
[25] Swagger. https://swagger.io/.
[26] TnT-Fuzzer. https://github.com/Teebytes/TnT-Fuzzer.
[27] M. Utting, A. Pretschner, and B. Legeard. A Taxonomy of Model-Based Testing Approaches. Intl. Journal on Software Testing, Verification and Reliability 2012.
[28] M. Yann kakis and D. Lee. Testing Finite-State Machines. In Proceedings of the 23rd Annual ACM Symposium on the Theory of Computing 1991.