International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

EVALUATING SECURITY PROPERTIES OF CLOUD SERVICE REST APIS FOR ENHANCED PROTECTION

[1]L. Priyanka, [2]V. Yogesh

[1]Assistant Professor, Department of MCA Student, Sree Chaitanya College of Engineering, Karimnagar

[2]MCA Student, Department of MCA Student, Sree Chaitanya College of Engineering, Karimnagar

ABSTRACT

REST APIs are used to programmatically access the majority of contemporary cloud and online applications. This article explains how a service might be compromised by an attacker taking advantage of flaws in its REST API. In order to capture the desirable characteristics of REST APIs and services, we present four security criteria. We next demonstrate how active property checks may be added to a stateful REST API fuzzer to automatically test and identify rule violations. We go over how to efficiently and modularly construct such checks. We addressed the security implications of the new issues we discovered using these checks in a number of production Azure and Office 365 cloud services that had been deployed. These bugs have all been resolved.

I. INTRODUCTION

Cloud computing is exploding. Over the last few years, thousands of new cloud services have been deployed by cloud platform providers, like Amazon Web Services and Microsoft Azure , and by their customers who are “digitally transforming” their businesses by modernizing their processes while collecting and analyzing all kinds of new data. Today, most cloud services are programmatically accessed through REST APIs . REST APIs are implemented on top of the ubiquitous HTTP/S protocol, and offer a uniform way to create (PUT/POST), monitor (GET), manage (PUT/POST/PATCH) and delete (DELETE) cloud resources. Cloud service developers can document their REST APIs and generate sample client code by describing their APIs using an interface-description language such as Swagger (recently renamed OpenAPI) . A Swagger specification describes how to access a cloud service through its REST API, including what requests the service can handle, what responses may be received, and the response format. How secure are all those APIs? Today, this question is still largely open. Tools for automatically testing cloud services via their REST APIs and checking whether these services are reliable and secure are still in their infancy. Some tools available for testing REST APIs capture live API traffic, and then parse, fuzz, and replay the traffic with the hope of finding bugs . Recently, stateful REST API fuzzing was proposed to specifically test more deeply services deployed behind REST APIs. Given a Swagger specification

of a REST API, this approach automatically generates sequences of requests, instead of single requests.

II. LITERATURE SURVEY

1) Model driven security for web services

AUTHORS: MM Alam et al.

Model driven architecture is an approach to increase the quality of complex software systems based on creating high level system models that represent systems at different abstract levels and automatically generating system architectures from the models. We show how this paradigm can be applied to what we call model driven security for Web services. In our approach, a designer builds an interface model for the Web services along with security requirements using the object constraint language (OCL) and role based access control (RBAC) and then generates from these specifications a complete configured security infrastructure in the form of Extended Access Control Markup Language (XACML) policy files. Our approach can be used to improve productivity during the development of secure Web services and quality of resulting systems.

2) Run-time generation, transformation, and verification of access control models for self-protection

AUTHORS: Chen, Bihuan; Peng, Xin; Yu, Yijun; Nuseibeh, Bashar and Zhao, Wenyun (2014).

A self-adaptive system uses runtime models to adapt its ar-chitecture to the changing requirements and contexts. How-ever, there is no one-to-one mapping between the require-ments in the problem space and the architectural elements in the solution space. Instead, one refined requirement may crosscut multiple architectural elements, and its realization in volves complex behavioral or structural interactions manifested as architectural design decisions. In this paper we pro-pose to combine two kinds of self-adaptations: requirements-driven self-adaptation, which captures requirements as goal models to reason about the best plan within the problem space, and architecture-based self-adaptation, which cap-tures architectural design decisions as decision trees to search for the best design for the desired requirements within the contextualized solution space. Following these adaptations, component-based architecture models are reconfigured using incremental and generative model transformations. Com-pared with requirements-driven or architecture-based approaches, the case study using an online shopping bench-mark shows promise that our approach can further improve the effectiveness of adaptation (e.g. system throughput in this case study) and offer more adaptation flexibility

3. Towards development of secure systems using umlsec.

AUTHORS: Jan J¨urjens

We show how UML (the industry standard in object-oriented modelling) can be used to express security requirements during system development. Using the extension mechanisms provided by UML, we incorporate standard concepts from formal methods regarding multi-level secure

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

systems and security protocols. These definitions evaluate diagrams of various kinds and indicate possible vulnerabilities.On the theoretical side, this work exemplifies use of the extension mechanisms of UML and of a (simplified) formal semantics for it. A more practical aim is to enable developers (that may not be security specialists) to make use of established knowledge on security engineering through the means of a widely used notation.

III. SYSTEM ANALYSIS

EXISTING SYSTEM:

Scanning of Swagger-based Representational State Transfer (REST) APIs - In addition to scanning Simple Object Access Protocol (SOAP) web services, Qualys WAS leverages the Swagger specification for testing REST APIs. Users need to only ensure the Swagger version 2.0 file (JSON format) is visible to the scanning service, and the APIs will automatically be tested for common application security flaws. - Enhanced API Scanning with Postman Support - Postman is a widely-used tool for functional testing of REST APIs. A Postman Collection is a file that can be exported from the tool that clubs together related requests (API endpoints) and shares them with other users. These collections are exported in JSON format. With the release of Postman Collection support in Qualys WAS, customers have the option to configure their API scans using the Postman Collection for their API.

DISADVANTAGES OF EXISTING SYSTEM:

- SOAP APIs are largely based and use only HTTP and XML.
- On other hand Soap API requires more resources and bandwidth as it needs to convert the data in XML which increases its payload and results in the large sized file.
- On other hand SOAP cannot make use of REST since SOAP is a protocol and REST is an architectural pattern.

PROPOSED SYSTEM:

REST APIs are implemented on top of the ubiquitous HTTP/S protocol, and offer a uniform way to create (PUT/POST), monitor (GET), manage (PUT/POST/PATCH) and delete (DELETE) cloud resources. Cloud service developers can document their REST APIs and generate sample client code by describing their APIs using an interface-description language such as Swagger (recently renamed OpenAPI) [25]. A Swagger specification describes how to access a cloud service through its REST API, including what requests the service can handle, what responses may be received, and the response format

ADVANTAGES OF PROPOSED SYSTEM:

- REST APIs are usually simple to build and adapt.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

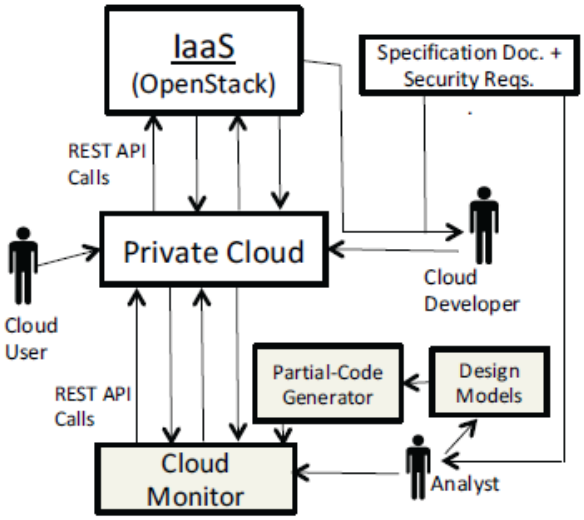
- With the initial URI, the client does not require routing information.

- Tools for automatically testing cloud services via their REST APIs and checking whether these services are reliable and secure are still in their infancy. Some tools available for testing REST APIs capture live API traffic, and then parse, fuzz, and replay the traffic with the hope of finding bugs.

- Clients can have a generic 'listener' interface for notifications.

-->The approach is implemented as a semi-automatic code generation tool in Django, a Python web framework.

SYSTEM DESIGN



IV. IMPLEMENTATION

MODULES:

- user
- cloud
- admin
- REST API.

User

It defines the access rights of the cloud users. A volume can be created, if the it has not exceeded its quota of the permitted volumes and a user Authorization is an important security concern in cloud computing environments. a POST request from the authorized user on the volumes resource would create a new volume. a DELETE request on the volume resource by an authorized user would delete the volume . if the user of the service is authorized to do so, and the volume is not attached to any instance .It aims at regulating an access of the users to system resources.

Cloud

.The cloud monitors contain contracts used to automatically verify the implementation .

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal www.ijarst.in
ISSN: 2457-0362

A cloud developer uses IaaS to develop a private cloud for her/his organization that would be used by different cloud users within the organization. In some cases, this private cloud may be implemented by a group of developers working collaboratively on different machines. We use Django web framework to implement cloud monitor and OpenStack to validate our implementation.

Admin

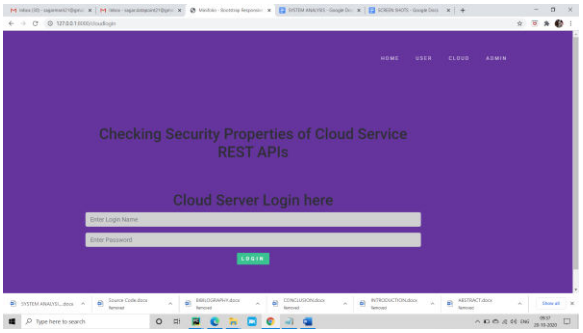
the cloud administrator using Keystone and users or user groups are assigned the roles in these projects. It defines the access rights of the cloud users in the project. A volume can be created, if the project has not exceeded its quota of the permitted volumes and a user is authorized to create a volume in the project. Similarly, a volume can be deleted, if the user of the service is authorized to do so, and the volume is not attached to any instance, i.e., its status is not in-use.

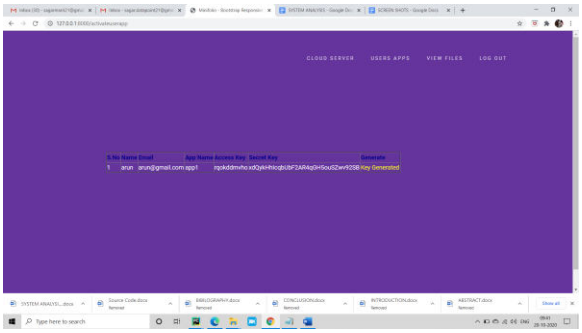
REST API:

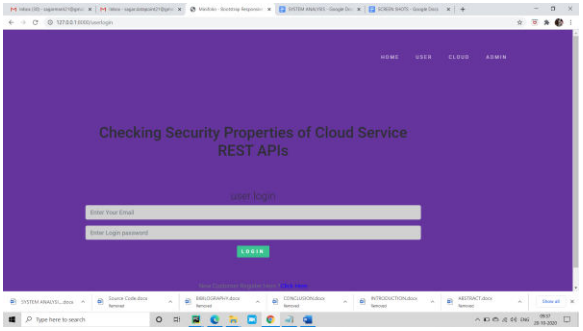
REST is an acronym for **RE**presentational **S**tate **T**ransfer. It is an architectural style for **distributed hypermedia systems**.By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components. Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client. Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests. By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resource through representations; self-descriptive messages; and, hypermedia as the engine of application state.The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting. REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

V. RESULTS

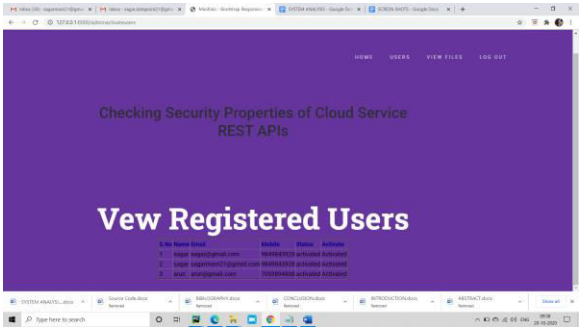
International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
IJARST
ISSN: 2457-0362



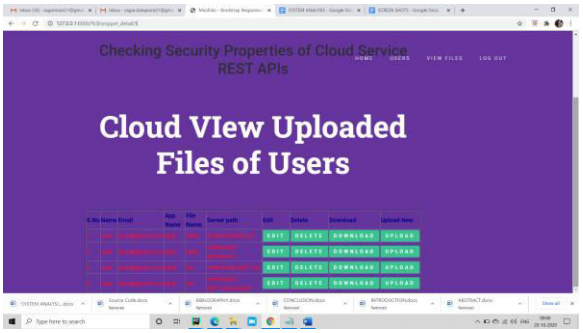




User details:



View-files:



V. CONCLUSION

In order to capture the desired characteristics of REST APIs and services, we created four security principles. We then demonstrated how active property checkers, which automatically test and identify breaches of these rules, may be added to a stateful REST API fuzzer. Using the fuzzer and checkers outlined in this work, we have so far fuzzed about a dozen production Azure and Office-365 cloud services. Our fuzzing was able to identify a few additional issues in each of these services in nearly every instance. Approximately one-third of those problems are rule violations that our new security checkers have discovered, and the other two-thirds are "500 Internal Server Errors." All of these issues have been resolved when we informed the service owners about them. In fact, it is evident that possible security vulnerabilities exist when the four security standards presented in this study are broken. Our current bug "fixed/found" ratio is over 100%, indicating that the service owners have treated the issues we discovered seriously. Furthermore, fixing these issues is safer than running the risk of an unknowable live event that may be purposefully started by an attacker or accidentally triggered. Lastly, the ease of reproducibility of these

problems and the fact that our fuzzing technique does not generate false alerts are helpful. How broadly applicable are these findings? We must examine more attributes and fuzz more services via their REST APIs in order to identify various bugs and security flaws. Surprisingly little advice exists about the use of REST APIs from a security perspective, despite the recent boom of REST APIs for cloud and web services. By proposing four criteria whose infractions are security-relevant and difficult to verify and comply with, our study takes a step in that direction.

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/enus/rest/api/dns/zones/get.

[15] Microsoft. Microsoft Azure Swagger Specifications. https://github.com/Azure/azure-rest-api-specs.