Understanding data leakage in cloud from Mobile Apps

Devin Sunil Lilaramani‡
School of Computing
Clemson University
Clemson, South Carolina
dlilara@g.clemson.edu

Shriya Reddy Surusani‡
School of Computing
Clemson University
Clemson, South Carolina
shriyas@g.clemson.edu

Anoop Kakkireni‡
School of Computing
Clemson University
Clemson, South Carolina
akakkir@g.clemson.edu

I. Introduction

With organizations moving their infrastructure to the cloud, there comes in a major question in to play, What about data security? A cloud leak is when sensitive business data stored in a private cloud instance is accidentally exposed to the internet. The cloud is part of the internet. The difference is that "the cloud" offers pockets of privatized space that can be used to carry out enterprise scale IT operations. Our project focuses on a tool called "Leak Scope" which uses the process of reverse engineering to find vulnerabilities in applications that use cloud APIs to develop the mobile apps..

II. Background

Widespread Adoption of Cloud APIs:
Before the advent of cloud technologies, the entire mobile application infrastructure was built from scratch which comprised of the front-end and back-end servers. Even after releasing the app, ensuring the availability and scalability of the application to accomodate varying user traffic was a manual task. Cloud service providers then started providing Mobile Back-end as a Service (mBaaS) to automate this process. With it came the cloud APIs to interact with the back-end. Different types of keys are now used to identify the source, authenticate the request and check access level of the source of the API calls. Most cloud providers make use of two different keys namely:
Root key: It is the master key which essentially has permissions to access all the back-end resources. It is advisable to keep this key a secret and only resevre it for use by the system adiministrators.
App key: These keys have limited permissions and it is a way for the cloud to identify the source of the API calls and only provide the information that is publicly available for that particular app.

With the introduction of cloud APIs, althougth the work of a developer has been minimized, so has the room for error. The developers are responsible for the proper authentication and authorizations to prevent security breaches.

Here are some of the biggest cloud data leaks in recent history

- Verizon Partner Nice Systems Exposes 6 Million Customer Records to Internet : This information, leaked from an exposed Amazon S3 bucket, included customer details such as name, address, and phone number, as well as some account PINs, used to verify identity with Verizon.
- RNC Vendor Data Root Analytics Exposes 198 Million Voter Records : One of the largest leaks of all time was discovered when an exposed cloud system was found, containing both collected and modeled voter data from Data Root Analytics, a firm contracted by the RNC for data driven political strategy. Over 198 million unique individuals were represented in the data set, with personal details, voter information, and modeled attributes including probable race and religion.

III. Problem

There are two major problems that are associated with the use of mBaaS from a data leakage standpoint. The first problem arises from the misuse of keys in authenticaing an API call. Developers tend to overlook the differences between an app key and a root key and this improper assignment leads to unwanted data leaks. This issue can also arise from improper guidance from the cloud providers. The second problem is associated with the misconfigurations of user permissions. Authentication only identifies who the user is whereas authroization decides what resources the user has to. Since there are multiple access levels defined by the developers, impropoer authorizations are highly susceptible to attackers and incrase the chances of data leakage.

IV. Related Work

We have studied various papers that align with our line of research. TaintDroid [5], PiOS [6], and AndroidLeaks[7] focused on privacy leakage such as GPS coordinates and address book. CHEX [8] and Harehunter [9] have been developed to identify component and code hijacking vulnerabilities. ConfErr [10], ConfAid [11] and SPEX[12] are configuration testing tools to reveal misconfigurations and improper authorizations. We intened to take inspiration from most of these studies to improve the tool "Leak Scope" and to better understand the vulnerbilities in mobile applications hosted on the cloud.

V. Our Plan

From the paper that we are reproducing [1], we see that the tool developed by the authors is used to scan the mobile apps statically though automation in such a way that the results of the following are directly achieved:

- Identifying Cloud API's used by the app. Along with the obfuscated API's
- Returning the key used in the mobile app
- Sending requests with the key to an invalid ID in the cloud service used by the app, with which they determine what type of key is the app hosting.

We aim to use the tool and reproduce our own experimental results, with a much lesser dataset than what has been used in the original paper[1]. Our focus would be on evaluating the working of the tool along with checking for any possibility of alternate approaches to perform each step that the tool is intended for. How we plan on moving forward with this is as follows:

- Finding a small dataset (few mobile apps) that use Azure, Microsoft or firebase along with the ones which do not use any cloud service for the backend.
- Execute the first part of the tool to find the API's in our dataset and to check for any false positives or false negatives.
- The second part of the tool that uses static analysis to identify the keys in the apps and retrieves it. We plan to manually do the dynamic analysis on the apps to find the key manually and compare if this approach has a better chance on finding the key than the static analysis.
- Next we can move on checking if the key is root key or app key by using their python script to Check if it's a trial and error hit.

VI. Timeline

Checkpoint 1: First we gather the requirements to execute the code followed by understanding the underlying implementation of it. Then, collect the sample set of Mobile apps using AWS, Azure, Firebase or any other cloud services for backend along with the ones which do not use any cloud services.

Checkpoint 2: Checking which cloud Mbaas service the app is using, with the help of API's that are retrieved from the apps. Followed by getting the key from the app manually, dynamically [2].

Checkpoint 3: Finally, Check if the key is root key or app key [4] and evaluate how effective the tool is regarding this and check if this key from the mobile app could causes the data leakage in cloud. Then, we evaluate how effectively the authorization rules are written for them [3]. Followed by writing the final paper.

References

[1] Zuo, Chaoshun, Zhiqiang Lin, and Yinqian Zhang. "Why does your data leak? uncovering the data leakage in cloud from mobile apps." 2019 IEEE Symposium on Security and Privacy (SP). IEEE, 2019.

[2] Serrão, Francisco Miguel Gouveia. "SMART: Static and Dynamic Analysis to Reverse Engineer Android Applications." (2021).

[3] Masood, Rahat, et al. "Cloud authorization: exploring techniques and approach towards effective access control framework." Frontiers of Computer Science 9.2 (2015): 297-321.

[4] Wang, Hui, et al. "Vulnerability assessment of oauth implementations in android applications." Proceedings of the 31st annual computer security applications conference. 2015.

[5] W. Enck, P. Gilbert, B. Chun, L. Cox, J. Jung, P. McDaniel, and A. Sheth, "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in OSDI, 2010

[6] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications," in NDSS, 2011.

[7] C. Gibler, J. Crussell, J. Erickson, and H. Chen, "Androidleaks: automatically detecting potential privacy leaks in android applications on a large scale," in Trust, 2012.

[8] L. Lu, Z. Li, Z. Wu, W. Lee, and G. Jiang, "Chex: statically vetting android apps for component hijacking vulnerabilities," in Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012, pp. 229–240.

[9] Y. Aafer, N. Zhang, Z. Zhang, X. Zhang, K. Chen, X. Wang, X. Zhou, W. Du, and M. Grace, "Hare hunting in the wild android: A study on the threat of hanging attribute references," in Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security. ACM, 2015, pp. 1248–1259.

[10] L. Keller, P. Upadhyaya, and G. Candea, "Conferr: A tool for assessing resilience to human configuration errors," in Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on. IEEE, 2008, pp. 157–166.

[11] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation, ser. OSDI'10, Vancouver, BC, Canada, 2010, pp. 237–250.

[12] T. Xu, J. Zhang, P. Huang, J. Zheng, T. Sheng, D. Yuan, Y. Zhou, and S. Pasupathy, "Do not blame users for misconfigurations," in Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles, ser. SOSP '13, Farminton, Pennsylvania, 2013, pp. 244–259.