Historical Archived IP List: Leveraging AWS to Persist Slack Security Data

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Jason Yu

Fall, 2022

On my honor as a University Student, I have neither given nor receivseed unauthorized aid on this assignment as defined by the Honor Guidelines for Thesis-Related Assignments

Advisor

Roseanne Vrugtman, Department of Computer Science

Technical Report

Historical Archived IP List: Leveraging AWS to Persist Slack Security Data

CS4991 Capstone Report, 2022

Jason Yu
Computer Science
The University of Virginia
School of Engineering and Applied Science
Charlottesville, Virginia USA
jiy6krm@virginia.edu

ABSTRACT

Slack security harnesses a tool called RAINS (Rapid Analysis, Internal Network Scan) to provide visibility into Slack's AWS (Amazon Web Services) infrastructure in order to alert engineers about unauthenticated services and defend against subdomain takeover attacks. However, RAINS does not keep an historical log of facts and findings, making it difficult to determine the cause of a potential security incident. To solve this problem, I implemented HAIL (Historical Archived IP List), consisting of a database to store RAINS findings and a backend API, allowing users to query past results. I leveraged AWS RDS (Relational Database Service) for the data layer, created the backend service using Flask, and used AWS Lambda and S3 (Simple Storage Service) in conjunction with the Risk and Compliance team's Security Data Warehouse project, enabling users to easily view RAINS results. I also modified the RAINS codebase (written in Go) to call the Flask backend and pass the findings to RDS. By the end of the internship, I successfully deployed HAIL to a production environment and configured metrics and dashboards using Prometheus and Grafana. One area for future improvement of the project is configuring default querying options within Security Data Warehouse, simplifying the process of extracting insights from historical RAINS results.

1. INTRODUCTION

Slack is a messaging application designed for workplaces and office productivity. Because Slack customers primarily include businesses, Slack is a likely target for cyberattacks. To defend against certain attacks, Slack owns a tool called RAINS (Rapid Analysis, Internal Network Scan) that provides visibility into Slack's AWS infrastructure, alerting engineers about cyber risks such as services lacking proper authentication and dangling DNS entries.

1.1. Unauthenticated Services

Slack manages a number of internal services intended to be accessed only by authorized Slack employees. To ensure security, these sites must be placed behind an authentication portal. If RAINS detects an internal service that does not require authentication, then RAINS sends an alert to Slack security engineers. Unauthenticated internal services allow attackers to steal sensitive information or company secrets.

1.2. Dangling DNS Entries

Slack routinely spins up and tears down thousands of hosts on a daily basis, and each host is associated with a human-readable CNAME (canonical name) via DNS (Domain Name System). A dangling DNS entry occurs when Slack tears down a host but does not remove the corresponding DNS record. In

some cases, a dangling CNAME record in one of Slack's subdomains is all that an attacker needs to take control of the content served by the subdomain in question. Such an attack is known as a subdomain takeover, and the potential risks include company defacement and stealing session cookies, allowing attackers to pose as Slack employees.

1.3. RAINS Limitations

RAINS solves the problem of detecting unauthenticated services and dangling DNS entries by scanning Slack's AWS infrastructure every 30 minutes. Indeed, Slack has historically paid tens of thousands of dollars through its bug bounty program due to such risks. However, RAINS does not keep a historical log of facts and findings, a limitation that makes it difficult to determine the source of a possible breach. To remedy this problem, HAIL (Historical Archived IP List) is a project designed to store RAINS scan results for future reference.

2. RELATED WORKS

Several existing projects accomplish similar goals to those of RAINS and HAIL. Nmap ("Network Mapper") is a free and open-source utility for network discovery and security auditing. Nmap works by checking a network for hosts and services (Lyon, 2009). Once found, the software platform sends a packet over TCP (Transmission Control Protocol) to those hosts and services which then respond. Nmap reads and interprets the response that comes back and uses the information to create a map of the network. The functionality of Nmap is akin to RAINS in that RAINS sends packets over the top 100 ports to hosts in Slack's AWS infrastructure, reading and interpreting the response to determine if a given host requires authentication [1].

Bhartiya (2021) developed an open-source tool written in Go that scans any number of user-specified domains for dangling DNS entries [2]. This tool allows users to check whether a subdomain can be overtaken for reasons ranging from a CNAME pointing to a CMS (content management system) provider (e.g., Heroku) that can be taken over to a dangling CNAME pointing to a non-existent domain name.

While a service that stores the same type of information as HAIL is not known to exist, the pattern of using a backend API to store and retrieve data in a relational database is a common use case for Flask and AWS RDS (Wattamwar, 2020) [3].

3. PROPOSED DESIGN

The system design for HAIL is shown in Figure 1.

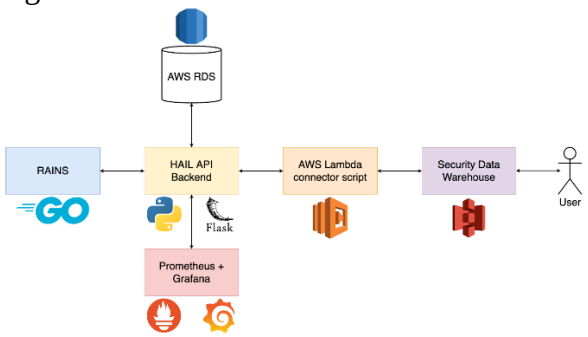

Figure 1. System Architecture

3.1. Data Layer

For the data layer, I used AWS RDS (MySQL 5.7), a fully managed, open-source cloud database service. RDS is an appropriate choice because the type of data that RAINS collects can be easily represented in a relational database model. Additionally, I decomposed the tables in order to conserve space. Specifically, I first assigned an incrementing, numerical value for every full scan of Slack's AWS infrastructure, occurring every 30 minutes). HAIL stored every RAINS finding in the database along with a start scan ID and an end scan ID, indicating the time range for which the given RAINS finding persisted. If a given RAINS scan result persisted from the previous RAINS scan to the current RAINS

scan, then instead of storing an entirely new record with the same data, the existing record's end scan ID was simply updated with the current scan ID. The final schema statements are shown below:

Run(run_id, time)
IPScanResult(id, ip_address, start_run_id, end_run_id, instance_name, instance_id, account_name, account_id, is_auth)
DNSScanResult(id, ip_address, cname, start_run_id, end_run_id, is_dangling)

I specified and deployed the desired RDS instance using HashiCorp Terraform, an open-source infrastructure-as-code tool.

3.2. Backend API

I used Flask, a Python web application microframework, to create a RESTful API to insert data into and retrieve data from the RDS database. I used the package PyMySQL to allow the HAIL API to programmatically communicate with the database by simply passing in the database account credentials. Additionally, I used SQLAlchemy, a Python SQL toolkit and Object Relational Mapper that automatically converts Python code into SQL operations, forgoing the need to write SQL code and mitigate related security issues such as SQL injections. After creating Python class models representing the database tables, I coded a number of API endpoints that serve GET and POST requests for each of the database tables. For each of the endpoints, the HAIL API returned the relevant records along with a status code. HAIL returned a status code of 200 for successful GET requests, a status code of 201 for successful POST requests, a status code of 404 in the case of an invalid user input, and a code of 500 in the case of some other error.

3.3. RAINS Modifications

After coding the API endpoints, I modified the RAINS tool to send POST requests to the HAIL API with RAINS scan findings. In order to allow RAINS to communicate with HAIL, I onboarded both services to Nebula, a Slack-owned, open-source global overlay network. To minimize traffic from RAINS to HAIL, configured the HAIL API endpoints and the corresponding RAINS code to accept lists of up to 1000 records of RAINS findings rather than calling the HAIL API for every individual RAINS finding, reducing the total volume of traffic by a factor of approximately 1000. For every full RAINS scan, RAINS collects approximately 100,000 scan results.

3.4. Security Data Warehouse

In order to present RAINS findings in a human-readable format, I harnessed the Security Data Warehouse, an existing project managed by the Risk and Compliance team. The Security Data Warehouse ingests security-related data from an AWS S3 (Simple Storage Service) bucket and exposes the data on a webpage in which authorized users can write SQL queries to retrieve data. To transfer RAINS results from RDS to S3, I wrote a Python script that calls GET request endpoints from the HAIL API and transforms the data into JSON (JavaScript Object Notation) format. This script was run via AWS Lambda, a serverless, event-driven compute service allowing users to run code on a periodic basis. I configured this Lambda script to run once a day, automatically retrieving the most up-to-date RAINS results from RDS and storing it in S3 to be ingested by the Security Data Warehouse.

3.5. Testing

To ensure code quality and correctness, I set up automated test cases using the PyTest tool. Such test cases included unit tests and integration tests, resulting in the entire HAIL codebase having complete code coverage. In order to ensure that database transactions were working properly, I created a mock SQLite database and tested the API endpoints against

the mock database, setting up mock tables at the beginning of the tests and removing the tables at the end of the tests so that each run of the test cases began with a clean start. Furthermore, I configured the Slack continuous deployment pipeline to run the test cases anytime a contributor pushed a new commit to the GitHub repo.

3.6. Metrics and Dashboards

To provide visibility into the health of the HAIL service over time, I configured metrics such as request rates, request counts, and latencies for each of the API endpoints using Prometheus, a software application used for event monitoring and alerting. In addition to gross metrics, I also recorded P50, P90, and P99 metrics for each of the aforementioned statistics, presenting the corresponding graphs in a Grafana dashboard.

3.7. Security

I incorporated security best practices into my design decisions. For example, instead of hard-coding database credentials in the HAIL codebase, I stored the secrets in HashiCorp Vault, which secures, stores, and tightly controls access to tokens and passwords. When HAIL needed to connect to RDS, it first called Vault to retrieve the secrets and subsequently gained access to the database. Additionally, I placed the HAIL API behind authentication, ensuring that only employees with a secret access token could call the API.

4. RESULTS

By the end of my internship at Slack, I successfully deployed HAIL and RAINS modifications to a production environment, thereby persisting RAINS scan results from Slack's actual AWS infrastructure in real time. I also presented HAIL to the security team at the end of the internship.

5. CONCLUSION

I successfully deployed HAIL, a project completed for the Slack product security team. HAIL keeps a historical record of facts and findings related to Slack's AWS infrastructure, providing valuable insight into Slack's attack surface and security posture.

6. FUTURE WORK

One area for future improvement of the project is configuring default querying options within Security Data Warehouse, simplifying the process of extracting insights from historical RAINS results.

7. UVA EVALUATION

My UVA coursework prepared me very well for this internship program. One helpful course was CS 3240 (Advanced Software Development Techniques), which taught me principles of software engineering and gave me experience with Django, a Python web application framework with many similarities to Flask. By taking CS 4750 (Databases), I learned about database design, schema statements, table decomposition, indexing, and writing SQL queries, all of which proved helpful during my internship. CS 4457 (Networks) taught me about DNS as well as the mechanics behind network scanning, which allowed me to easily understand the RAINS codebase and the rationale behind defending against subdomain takeover attacks. Finally, CS 4740 (Cloud Computing) provided me an introduction to AWS and gave me a head start when deciding which AWS services to use for HAIL.

REFERENCES

[1] Lyon, G. 2009. Nmap Network Scanning. Nmap Project.

[2] Bhartiya, A. 2021. Tko-subs. tko-subs, (2009), GitHub repository. Retrieved October 10, 2022 from https://github.com/anshumanbh/tko-subs

[3] Wattamwar, A. 2020. AWS RDS with MySQL using Flask. (August 2020). Retrieved September 8, 2022 from https://medium.com/aws-rds-with-mysql-using-flask-f1c6d8cc7ef