Hypersafe

By
Tony Iacobelli, Anthony Burchette, and Peter Johnson

Submitted to:
Faculty of the School of Information Technology
In Partial Fulfillment of the Requirements for
The Degree of Bachelor of Science
In Information Technology

Tony Iacobelli - Project Manager/Chief Architect — Date 4/23/2019

Anthony Burchette - Lead Back-End Developer — Date 4/23/19

Peter Johnson - Lead Front-End Developer — Date 4/23/2019

Bogdan Vykhovanyuk - Advisor — Date 4/23/19

University of Cincinnati
College of
Education, Criminal Justice, and Human Services

April 2019

Table of Contents

Section **Page**

List of Illustrations

List of Tables

Abstract

Phishing Emails are a problem as ubiquitous as email itself. According to Digital Guardian, over 90% of all cyber attacks now start with a phishing message (Guest, 2017). Additionally, with over 12% of targets clicking on any given phishing message (Verizon, 13), and 95% of all attacks on enterprise networks begin as a result of spear phishing, it is critical to not only know who clicks on these links, but also when a link was clicked. While there are many tools trying to prevent the message from ever getting in, only Hypersafe will enable instant visibility and scoping when a message inevitably gets through. This is accomplished by intercepting all inbound messages to a target's email system, and then scanning messages for spam and malicious content, and then replacing every link contained within the message with a custom link that points to the Hypersafe Link Resolution Service. When a user clicks on a link, they first go through the Link Resolution Service to ensure that the link has not been identified as a phishing or otherwise malicious message. If the link is clean, then the user is seamlessly redirected to their intended target. If the link is malicious, the user is redirected to an education and awareness page, thus hindering the attack. In order to ensure reliable service, Hypersafe is built atop the proven Amazon Web Services Platform to enable best in class availability, scalability, and security. The email processing and link resolution services are serverless, enabling Hypersafe to scale without any administrative burden and also cost very little to operate. Finally, there is a web administration portal that administrators can use to interact with the service to manually flag links, senders, and domains as malicious as well as view statistics on who has clicked on a certain link. By taking the uncertainty of email away from the user, Hypersafe will be an invaluable partner in each customer's defense toolkit.

Problem Statement

1.1 Introduction

Phishing emails are a hindrance on industries and individuals. Costing millions of dollars per incident, multiple organizations have been affected by this nuisance and are in need of a proactive defense. According to the latest Verizon Data Breach Report, most phishing attacks are not reported until they are hundreds of emails in. Instead of trying to focus on the message from ever getting in, Hypersafe takes a different approach where each link is tracked and able to be neutralized. Hypersafe aims to detect when one of these attacks inevitably end up centered on an organization, track every link coming into an organization, educate the customers' end-users about the dangers of phishing attacks, and provide deep visibility into these attacks to aid in rapid scoping of the incident.

1.2 Problem

With over 30% of targets opening a phishing message, 12% clicking on a phishing message, (Verizon, 13), and 95% of all attacks on enterprise networks being a result of spear phishing (Network World), it is critical to not only know about every click from every user, but also remove the user's ability to get to a malicious link. As phishing rates against institutions continue to rise (Verizon, 11), the cost of a single incident also continues to rise to an average of $1.6 million for a mid-sized business (Cofense, 2).

This leads to a disturbing reality: these attacks are not only costing organizations more, but they are happening more often.

1.3 Solution

Hypersafe is a service that will process all inbound emails to a customer. During processing, all messages will be checked for spam content and malware, and links contained within the message will be detected and replaced with a unique link that resolves to the Hypersafe Link Resolution Service. Messages that are identified as spam, malware, or phishing will not be delivered to the customer. When a user clicks on a link, the user will first be routed to the Hypersafe Link Resolution Service. The link that the user clicked on has the necessary information in it for the link resolution service to look up if the link, sender, or domain has been identified as malicious. If the link has not been identified as malicious, the user is redirected to the intended website, and the access to the site is logged. If the link has been identified as malicious, the user will be redirected to an education and awareness page explaining why the user was redirected to that site, and also provide educational material to help them avoid phishing messages in the future. All link processing and resolution is done completely serverless, ensuring the highest levels or availability, redundancy, and security for customers.

1.4 Project Goals/Methodology

The primary goal of this project is to provide peace of mind to customers that their organization is more resilient to one of the most common attack vectors against their digital assets. The following sub-goals enable the fulfillment of the main goal

- Design, test, and implement a serverless link parsing and replacement service
- Design, test, and implement a serverless link resolution service
- Design, test, and implement a web application to enable customers to manually flag malicious content and further provide customers the visibility into which links have been clicked on

The guiding methodology used in the development of this platform was to make the platform “secure first.” As this is a tool to help organizations secure themselves, it was of utmost importance to the development team to ensure that this product would not introduce additional vulnerabilities or additional risk to our customers. Keeping this in mind, the team diligently tested each component and architected the solution to uphold the highest levels of Confidentiality, Integrity, and Availability.

1.5 Overview

The rest of the below report will explore how this project was completed in detail. The following sections are included within this document: design objectives, methodology, budget, timeline, problems encountered, and technical documentation.

Discussion

2.1 Project Concept

The idea for this project came from the group member's collective experiences dealing with a near constant stream of highly complex, automated phishing attacks against their employer's email infrastructure. As their respective units (Information Security, Identity Management, and Integrated Service Desk) all have a hand in the detection and remediation of compromised accounts due to phishing, the idea to proactively alert on and disable links, while providing visibility into who has click on any given link was born. By ensuring that every click from every user on every device was safe, the ability for the large-scale success of a phishing attack would be significantly diminished, and thus result in a more sustainable workload of each member of the project team when dealing with phishing attacks.

2.2 Design Objectives

The primary goals for Hypersafe are to provide easy to use user-level phishing protection through controlled Uniform Resource Locator (URL) resolution, provide a level of privacy to the end users when links are encoded, and provide ease of administration in URL management and visibility.

By parsing, encoding and then replacing links in each email that passes through Hypersafe the end user is not required to change their day to day processes to be protected. When Hypersafe encodes the links, it parses in each email only the minimum amount of information is encoded for administrators to know which email the link originated from. Only the original link, the recipient of the email, and a timestamp of when the link was encoded will be in the encoded link for administrators to see. Administrative overhead is minimized with Hinder, the streamlined administration portal that provides functionality to add, edit, remove, enable, and disable links.

As with any project, there were a small subset of goals that never made it to the development stage due to time constraints. These goals were:

- Integration of artificial intelligence to determine the validity of phishing messages.
- Integration with external threat intelligence feeds to disable link resolution.

2.3 Methodology/Technical Approach

Being primarily a security tool, Hypersafe's methodology was to not introduce any new risk to an organization by use of this tool. Hypersafe is designed to enable the highest levels of Confidentiality, Integrity, and Availability. Across the board, by being a serverless application, Hypersafe is able to be immune to many of the vulnerabilities found in every server operating system, can easily scale up to the required demand, and also be highly available in multiple data centers across the world. Furthermore, the SANS institute describes the best Cyber Security Controls as transparent to the genuine

user. With this principle in mind, Hypersafe was designed to ensure that end-users do not need to change their current email workflows. By integrating with the provider's email-service, and re-writing the links on the back-end, without relying on any parts of an end-user's device, Hypersafe is able to achieve this transparency.

2.4 User Profiles

The end user will never knowingly interact with our service, as the Hypersafe methodology of protection is transparent to the genuine user. The user would continue to interact with their email regularly and click on the links within as they normally would. Because of this, clients would be able to continue working through their emails undisturbed. Figure 2.4.1 below describes these user profiles.

User Profiles
Project: Hypersafe - A phishing email protection service
Potential Users: - Anyone with an email account that has a desire to remove the access a person would have to interact with phishing email links.
Experience for the User: - The user will only know that Hypersafe is protecting them if they interact with an identified phishing message and are redirected to the education and awareness page. Outside of that scenario, Hypersafe's protection is transparent to the genuine user.
Experience with Other Applications: - There are no other applications that a person encounters on a regular basis that would interact with our service.
Task Experience: - Using the email application: Gmail, Office 365, Hotmail, Corporate email. A customer will be able to access their email account and begin reading emails inside of their mailbox as normal. Before they have even looked into their email. The web service will have scanned the contents of the email and disabled malicious email links, determined by our application. When a customer clicks on the link, if it is safe then it will work as advertised. If the link is not deemed safe, then the link will be routed to a warning page with information for prevention and screening.
Frequency of Use: - Hypersafe will be used any time a customer clicks on a link within their email. With the application being a web service, the implementation would be across as many platforms and email services that opt for it. Phishing emails can come from many sources and will cause a system to be compromised.

Figure 2.4.1 ***User Profile Diagram***

2.5 Use Case Diagram

The diagram that is listed in Figure 2.5.1 is meant to show the interface between the users and the Administrators:

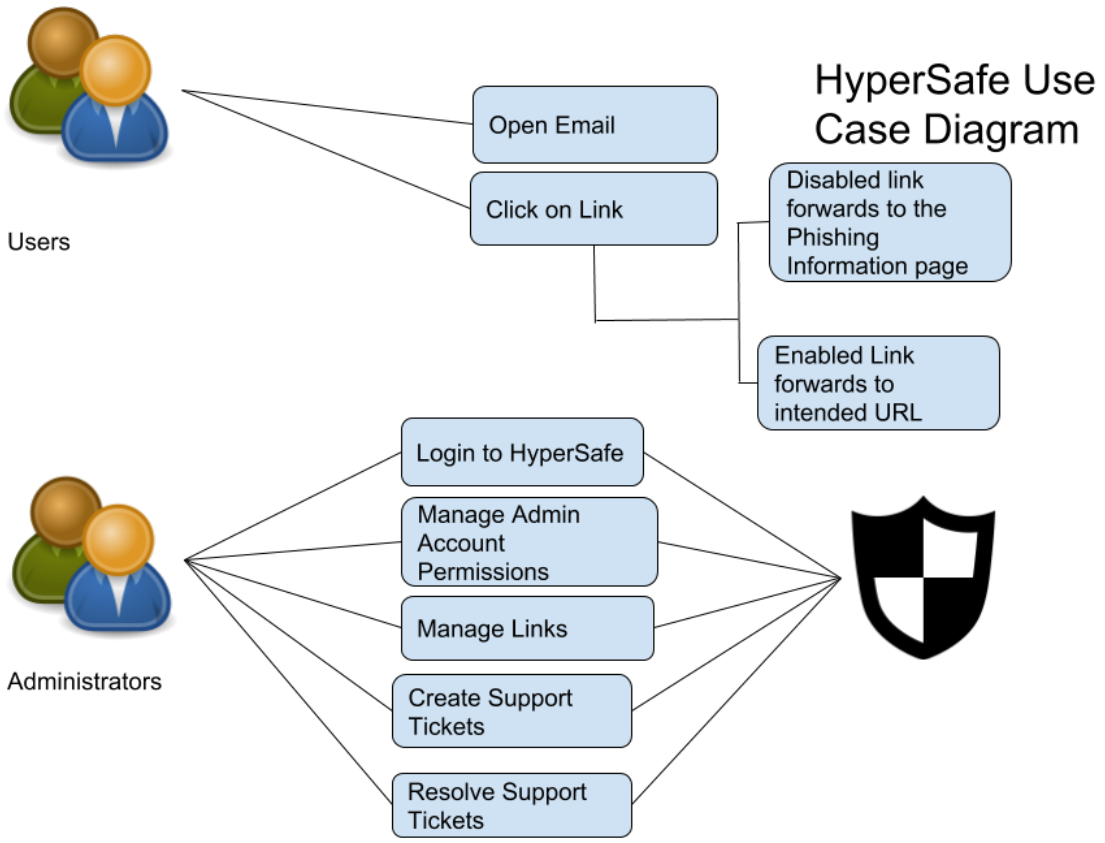


Figure 2.5.1 ***Use Case Diagram***

2.6 Technical Discussion

Hypersafe is a highly distributed serverless application. There are four main services (Email Delivery Service, Link Resolution Service, Hinder- Web Administration Portal, and the informational website). Each of the services will be described in detail

below. Definitions for all Amazon Web Services (AWS) offerings used are provided in Appendix C: *Amazon Web Services Definitions.*

Additionally, the benefits of a serverless deployment are massive as comparted to a traditional architecture. The project team was able to realize the decreased management load of services allowing for further development of each respective service, and the low total cost to implement and sustain this project. On the security front, since there are no servers to manage or interact with, risk is reduced since all patching and management of the low-level resources is handled by AWS. The depth of the project was only able to be completed with this serverless model, as there simply would not have been enough time to complete a project of this scope if there were multiple additional management layers to each of the technical components.

2.6.1 Email Delivery Service

The primary goal of the Email Delivery Service is to provide an initial layer of filtering for all messages coming in and also parse and replace all links within a message. Figure 2.6.1.1 illustrates how this service works.

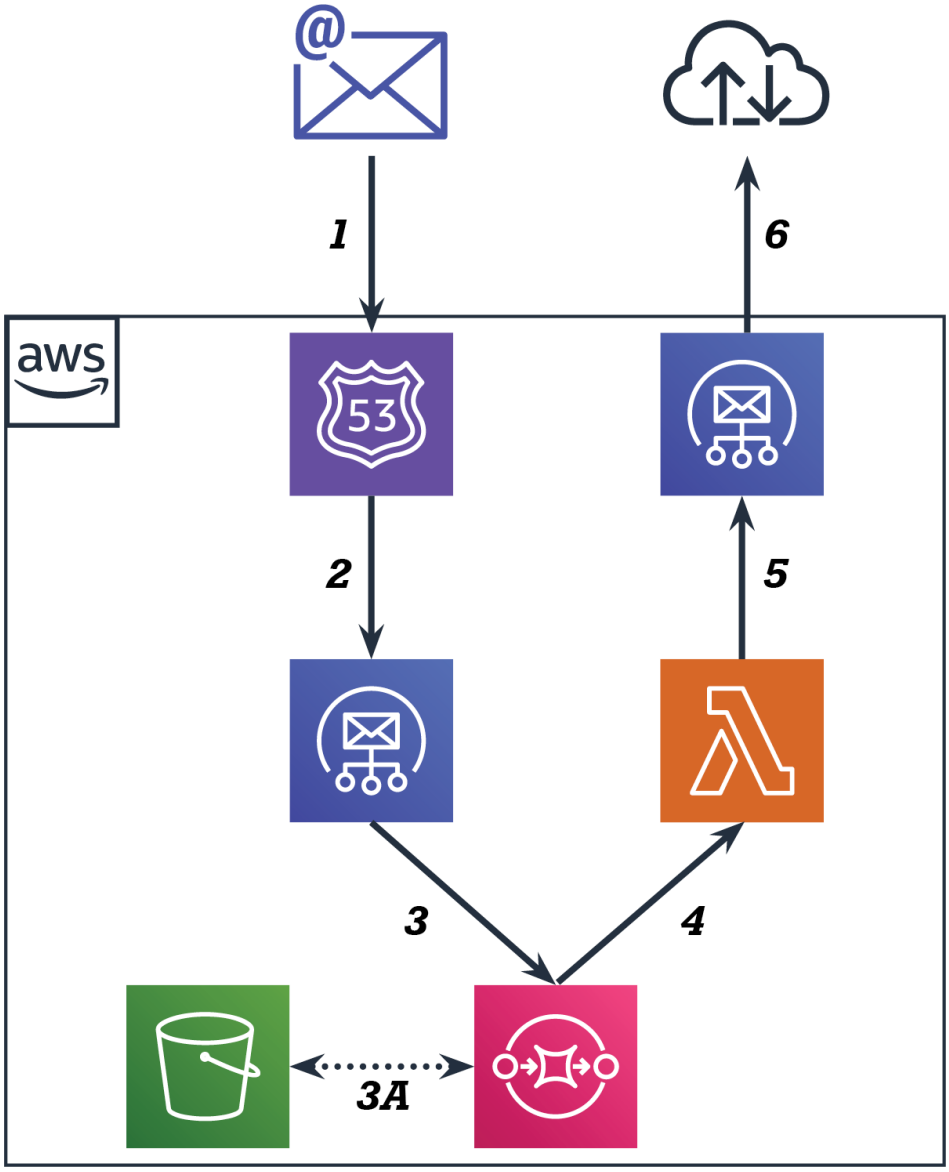


Figure 2.6.1.1 ***Email Delivery Service Technical Diagram***

1. An email message is sent to an organization that is protected by Hypersafe. Due to Mail Exchange (MX) record weighting, the message is delivered to Hypersafe's Infrastructure. All of Hypersafe's Domain Naming System (DNS) infrastructure is hosted by Route 53.
2. Hypersafe's mail relay (Simple Email Service, or SES) scans the message for malware, checks Sender Policy Framework (SPF) and Domain Keys Identified

Mailer (DKIM) records, and sender reputation databases. If these checks fail, the message is dropped, all other processing stops, and the message is not delivered to the recipient.

3. Once the preliminary checks are complete, the message is placed into a First-in-First-Out (FIFO) processing queue, managed by Amazon Simple Queue Service (SQS).
    a. SQS places a copy of every message into a Simple Storage Service (S3) bucket to allow for analysis of the original messages.
4. SQS feeds messages to a Hypersafe link processor, which are Lambda Functions. The link processors parse each message, and replaces each link per message, per user. Up to 1,000 link processors can run concurrently per availability zone.
5. Once each link is parsed and replaced, the link processor sends the processed message back to SES for the message to be sent to its final destination.
6. Hypersafe's exit mail relay (SES) sends the message to its intended destination, whether it be an Exchange Server, Office 365, Amazon WorkMail, Gsuite, or any other mail provider.

It is important to note that all link processing Lambda Functions are written in Python and use a regular expression to detect and validate links found within a message. When a link is replaced, all of the information other Hypersafe services need to decode and inspect the original destination are contained within this replaced link. This ensures that all links will work years in the future, as resolution to the final designation is not dependent on any database, schema, or other external data outside of all of the information contained within the replaced link.

2.6.2 Link Resolution Service

The Hypersafe Link Resolution Service comes into play when a user clicks on a link. The purpose of this service is to redirect the user's device to either the intended target of the original link or to the Hypersafe Education and Awareness Page if the link is malicious, as illustrated in Figure 2.6.2.1.

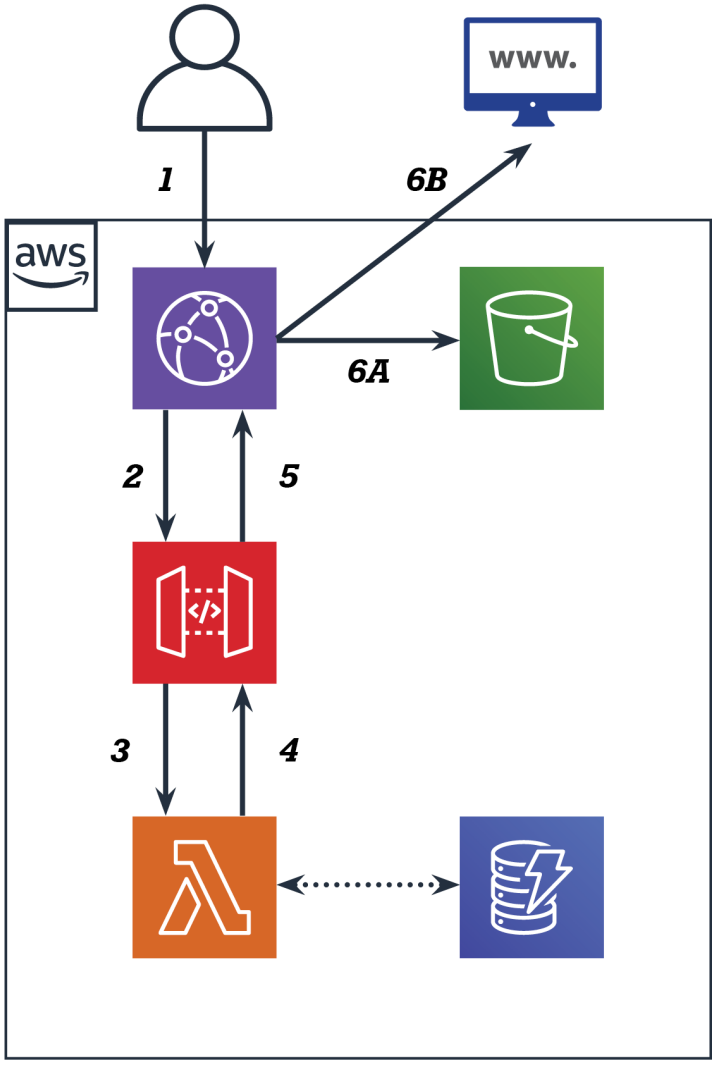


Figure 2.6.2.1 ***Link Resolution Service Technical Diagram***

1. A user clicks on a link within an email message that was previously processed by Hypersafe. The link resolves to the Hypersafe Link Resolution Service, accelerated by CloudFront.

2. CloudFront passes the request to the Hypersafe Link Resolution Application Programming Interface (API), built atop the AWS API Gateway. If the request is identical to a request received in the past 5 minutes and the link is malicious, the previous cached result is returned to the user, and no further processing is done.
3. The Hypersafe Link Resolution API proxies a Lambda Function. The function decodes the link and checks if any of the link's information (content, sender, etc.) is contained within a Dynamo DB (database) of known malicious indicators.
4. The link analysis Lambda Function returns the redirect target, based on the analysis performed in step 3. If the link is malicious, the redirect target is the education and awareness page. If the link is valid, the redirect target is the original destination. The API gateway turns this information into a 301 redirect to be interpreted by the user's browser.
5. The Link Resolution API passes the result to CloudFront. CloudFront caches results for malicious links for 5 minutes.
6. The user's browser is redirected to one of two places:
    a. An education and awareness page (hosted in an S3 bucket) stating that the link was malicious, and that Hypersafe Protected them.
    b. The original destination if the link is safe.

Additionally, it is worth noting that the Hypersafe Link Resolution API is able to proxy multiple Lambda Functions, based on the resource (e.g. /check-link) and the method (e.g. POST). This allows a multitude of functionality, such as adding links, searching links, and also removing links to be accomplished via this API.

By accelerating all API calls via CloudFront, every user gets a quick response, no matter where they are in the world, and a cost savings is able to be obtained due to caching results, as those requests to not invoke the API Gateway, Lambda Functions, or DynamoDB lookups.

2.6.3 Hinder (Web Administration Service)

Hinder is Hypersafe's web-based administration service. Hinder is a native Flask application, and Bootstrap is used for managing the layout and design of Hinder. Multiple themes are able to be applied, so that the experience can be branded to each tenant's linking without removing any functionality whatsoever. This ability is demonstrated in Figures 2.6.3.1 and 2.6.3.2 on the following pages.

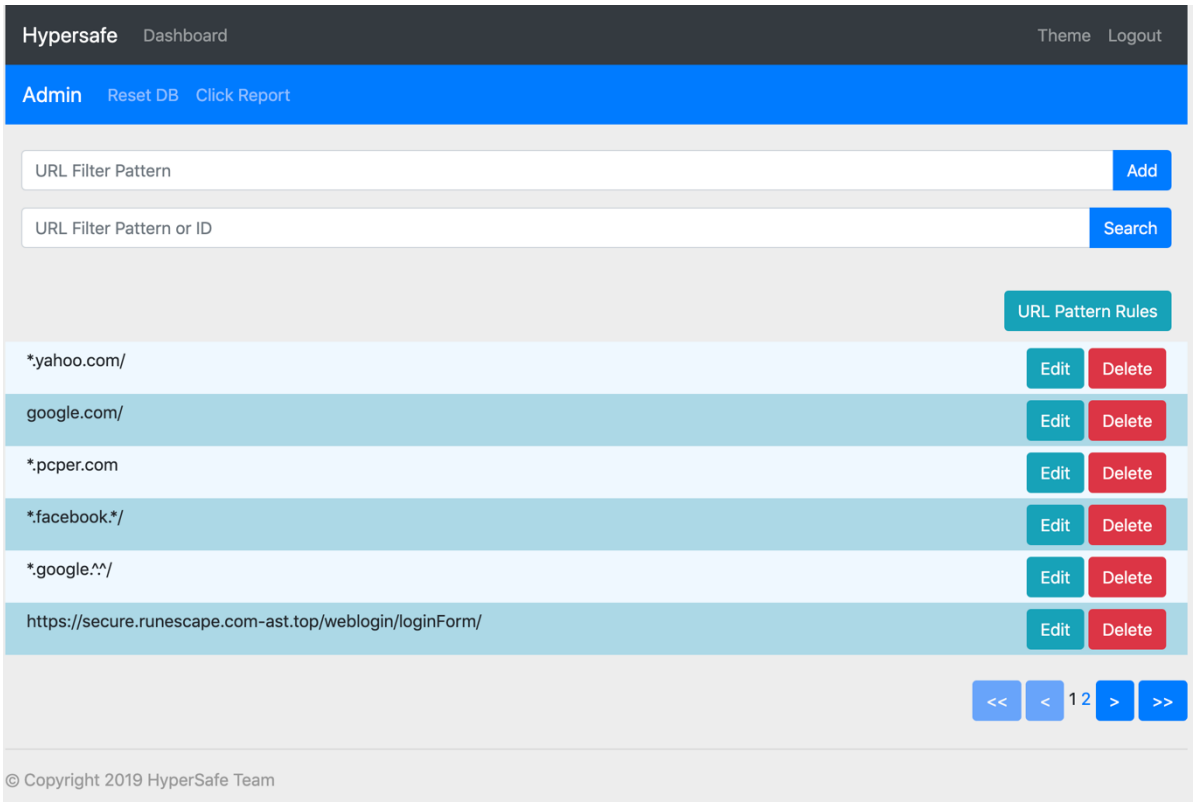


Figure 2.6.3.1 ***Default Hypersafe Theme***

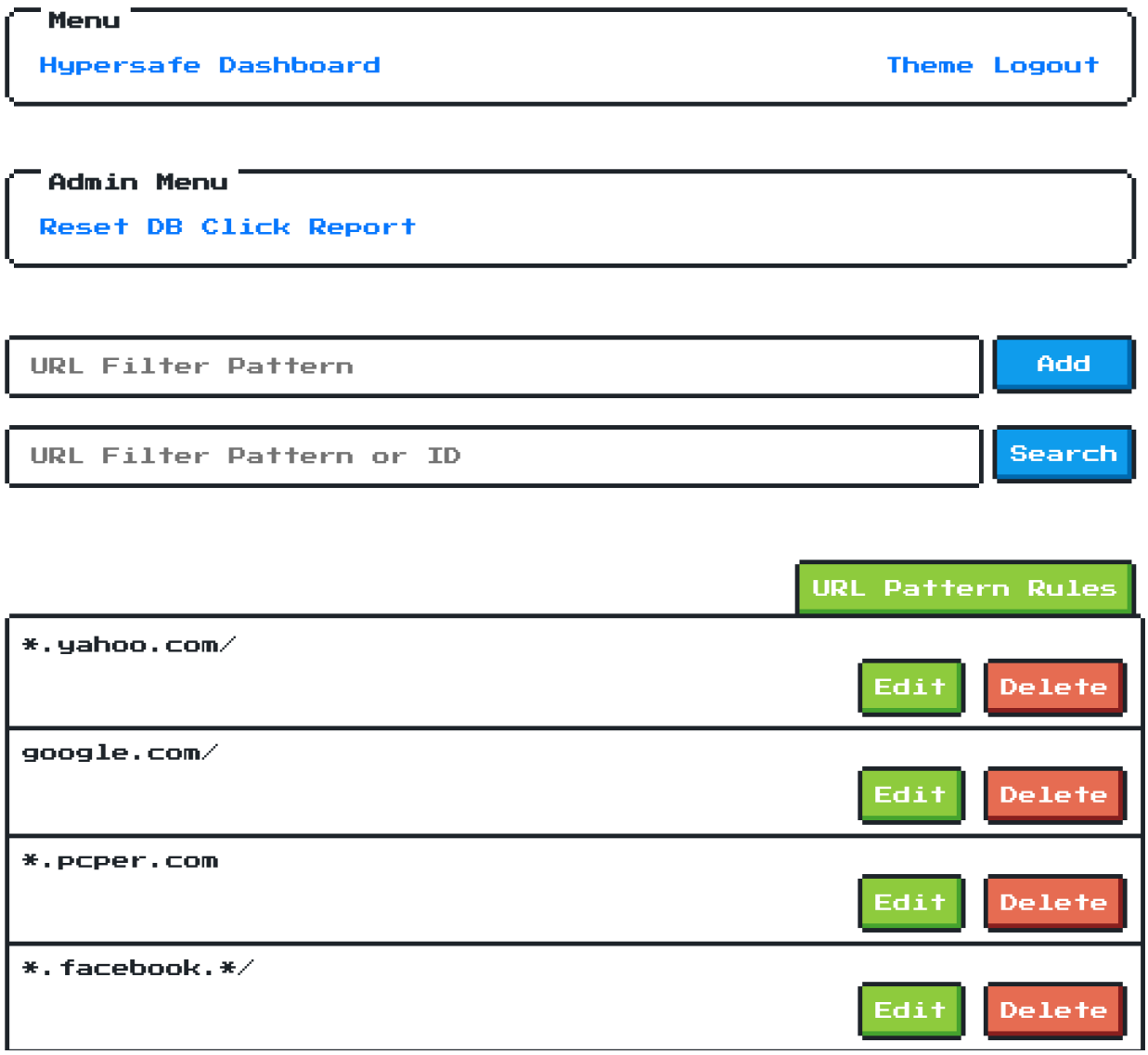


Figure 2.6.3.2 ***Hypersafe Super Nintendo Theme***

Furthermore, Hinder is the most technically complex part of Hypersafe. This is due to needing to provide functionality to control all other Hypersafe services, and also do so in

an easy to use and secure manner. Figure 3.6.3.3 illustrates this interaction at a high level.

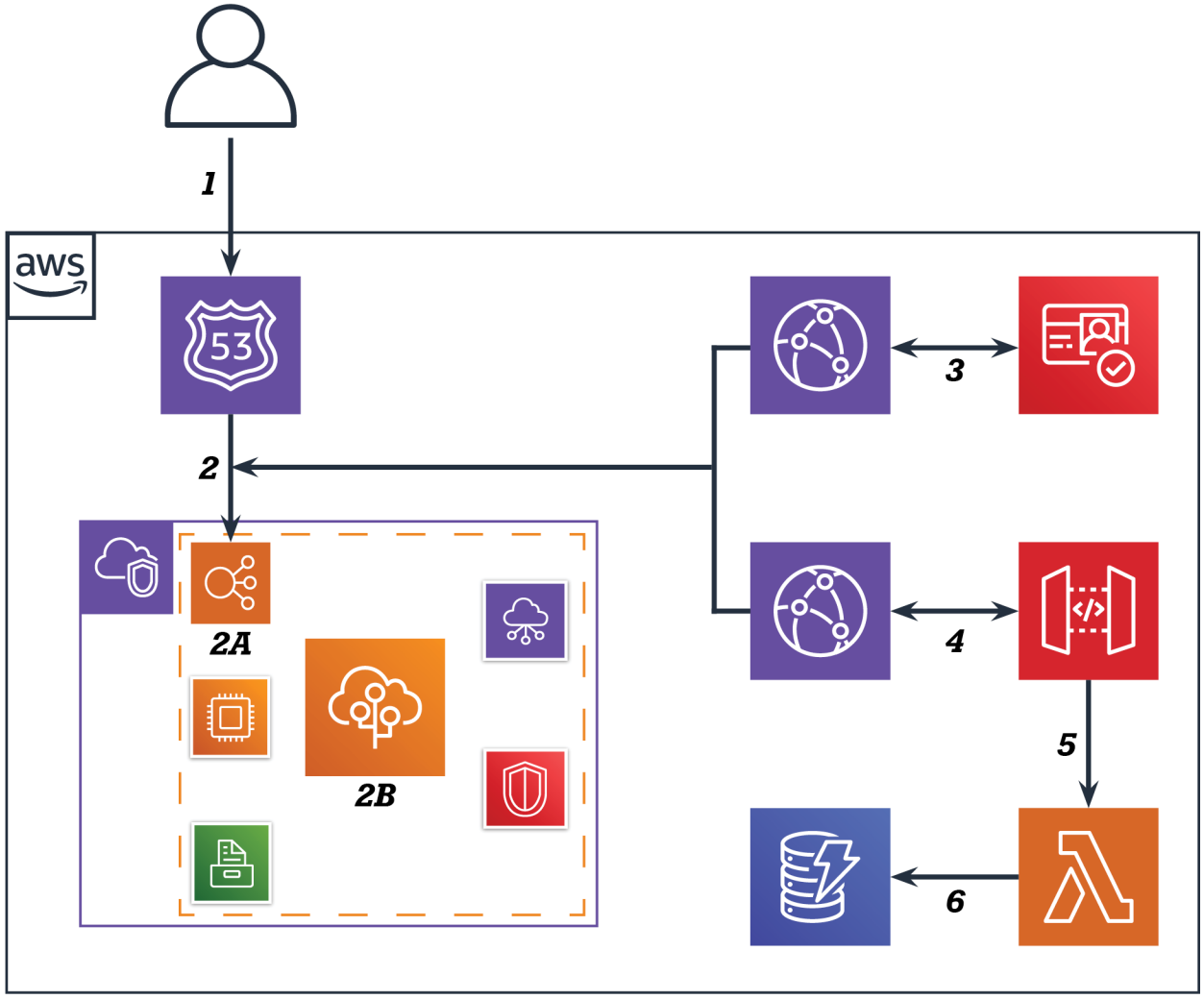


Figure 2.6.3.3 ***Hinder Technical Diagram***

1. The user navigates to Hinder via their preferred web browser.
2. Route 53 routes the request inside the Hinder Virtual Private Cloud (VPC).
    a. An application load balancer redirects all insecure traffic to use secure protocols and is the only publicly accessible resource within the application.

    b. Elastic Beanstalk orchestrates and manages all compute, store, network, and security resources to run Hinder.
3. Accelerated by CloudFront, the user logs in to Hinder using Cognito or a federated social provider of their choice.
4. Also accelerated by CloudFront, all changes the user makes within Hinder invoke the administration API to make changes.
5. The Administration API gateway proxies multiple Lambda Functions to make the appropriate changes within the database, based on what the user clicks within Hinder.
6. Once syntax of the request is checked, the Lambda Function commits the changes to DynamoDB.

By running Hinder in a VPC, the utmost levels of confidentiality, as the VPC is isolated from all other AWS tenants and resources. Only authorized flows are permitted to authorized services (e.g. Hinder can only view logs, but not erase logs within Cloud Watch).

Furthermore, significant time and resources were spent on developing AWS Cognito to be the identity provider for Hinder. Cognito handles all access and authorization, allowing for multiple roles to be defined across multiple login providers. Users are able to sign into Hinder via Cognito using either a Federated login from the University of Cincinnati (via shibboleth), their Google, Amazon, or Facebook accounts (via openID), or by creating an account. If the user creates an account within cognito, multiple security processes are applied to ensure that all logins are genuine. First, the password the user sets is automatically checked against a database of known compromised and

easy to guess passwords. The user is prompted for a password reset upon their next sign in if their password has been detected as compromised. Secondly, all sign ins from a new device send an email alert to the user provided and verified at signup. If the user did not authorize the sign in, a link is included within the message to lock down the account. Finally, all users are able to voluntarily enroll in multi-factor authentication. All of these security processes are part of the AWS Cognito service offering.

2.6.4 Informational Website

The project team thought it would be a good idea to create a simple website to better explain and showcase the project. This site has no functional use to the Hypersafe platform outside of marketing purposes. The website was built upon WordPress using a LAMP (Linux, Apache, MariaDB, and PHP) stack. This is the only component of the entire project that is not serverless. Figure 2.6.4 details the architecture of this website on the next page.

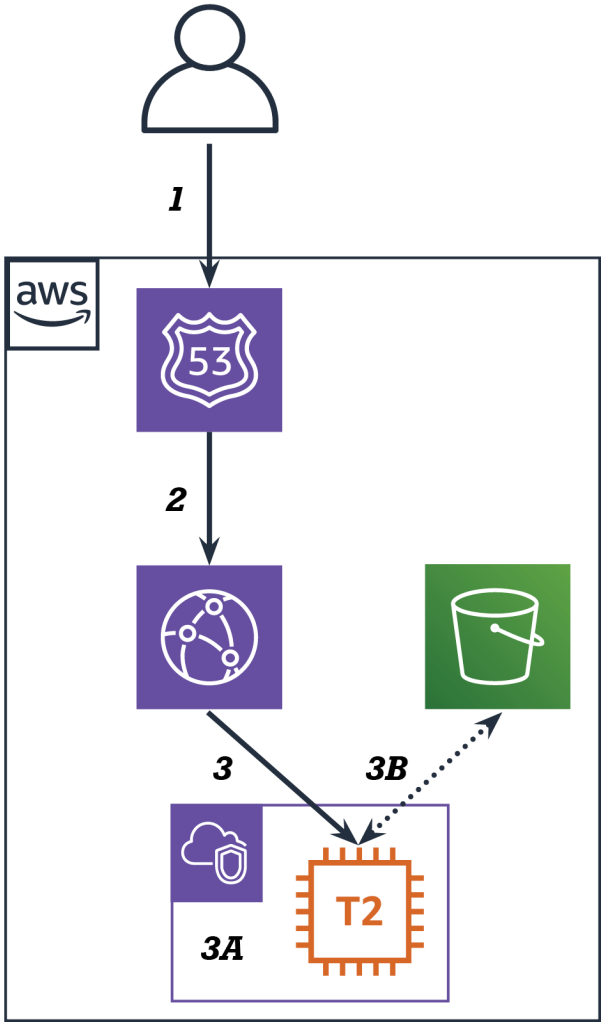


Figure 2.6.4.1 ***Hypersafe Informational Website Technical Diagram***

1. The user navigates to the Hypersafe website.
2. Route 53 routes the request to the nearest edge location in CloudFront.

   CloudFront redirects all insecure connections and also terminates Transport Layer Security (TLS).
3. If the request is not already cached, CloudFront will proxy the request to the web server running on a T2.micro Elastic Compute Cloud (EC2) instance.

a. The web server is located within its own VPC to isolate it from the rest of the Hypersafe platform and ensure that the website poses no additional risk to the safe operation of Hypersafe.
b. An S3 Bucket is used for content storage.

A significant cost savings was able to be achieved via accelerating the website with CloudFront, as a smaller server was able to be used, and it provides for one network surcharge instead of a distinct surcharge for each service. Another added benefit to acceleration is decreased site response time.

2.6.5 Other AWS Services Globally Used

There are a multitude of AWS services used across every service that makes up Hypersafe. While not reflected in any of the above diagrams, a listing of each service (and what it does) is provided below.

- **Amazon Certificate Manager** is a fully managed service for requesting and renewing TLS certificates with AWS Services.
- **Amazon Simple Notification Service** enables users, devices, and services to publish and receive notifications from the cloud.
- **AWS Cost Management** provides detailed usage and billing reports pertaining to the use of AWS services and infrastructure.
- **Key Management Service** facilitates the generation and use of unique security keys across various AWS services.
- **AWS Management Console** allows for web-based access to the management consoles for all AWS resources.

- **CloudTrail** is a managed logging service that collects logs from various AWS Services. All Hypersafe services (and their underlying AWS components) log using CloudTrail by default.
- **CloudWatch** is a service that allows for monitoring and alarms based on CloudTrail data and other metrics.
- **Identity and Access Management** allows for management of permissions that dictate how AWS services interact with each other. Each Hypersafe Service (and its underlying AWS service(s)) have custom roles that dictate how that service can interact with other components within the entire ecosystem. These roles ensure that the most restrictive permissions for each service to work are applied (e.g. due to IAM configurations, only search API calls can search the database, and they are not permitted to update or change any information).

2.7 Budget Management

One of the benefits of having a largely serverless deployment within a cloud service is the minimization of costs, and the ability to pinpoint and exactly estimate what the costs will be going forward. Using the AWS Cost Management Tools, the initial budget the project team forecasted was correct, even though the conclusion of this project. Table 2.7.1 shows the budget:

Service Type	Components	Region	Component Price	Service Price
Amazon S3 Service				$0.16
	S3 Standard	US East (N.	$0.12	

	Storage:	Virginia)		
	S3 Standard Put Requests:	US East (N. Virginia)	$0.03	
	S3 Standard Other Requests:	US East (N. Virginia)	$0.01	
Amazon Route 53 Service				$1.90
	Hosted Zones:	Global	$1.50	
	Standard Queries:	Global	$0.40	
Amazon DynamoDB Service				$2.69
	Provisioned Throughput Capacity:	US East (N. Virginia)	$2.38	
	Indexed Data Storage:	US East (N. Virginia)	$0.31	
Amazon SES Service				$2.10
	Send Messages from email clients	US East (N. Virginia)	$0.50	
	Attachment from email clients	US East (N. Virginia)	$1.20	
	Receive Messages	US East (N. Virginia)	$0.40	
AWS Data Transfer In				$0
	US East (N. Virginia) Region:	Global	$0	
AWS Data Transfer Out				$0.36
	US East (N. Virginia) Region:	Global	$0.36	
AWS Lambda	US East (N. Virginia) Region:	Global	$0.36	
AWS API Gateway	US East (N. Virginia) Region:	Global	$3.50	
AWS Support (Basic)				$0
	Support for all		$0	

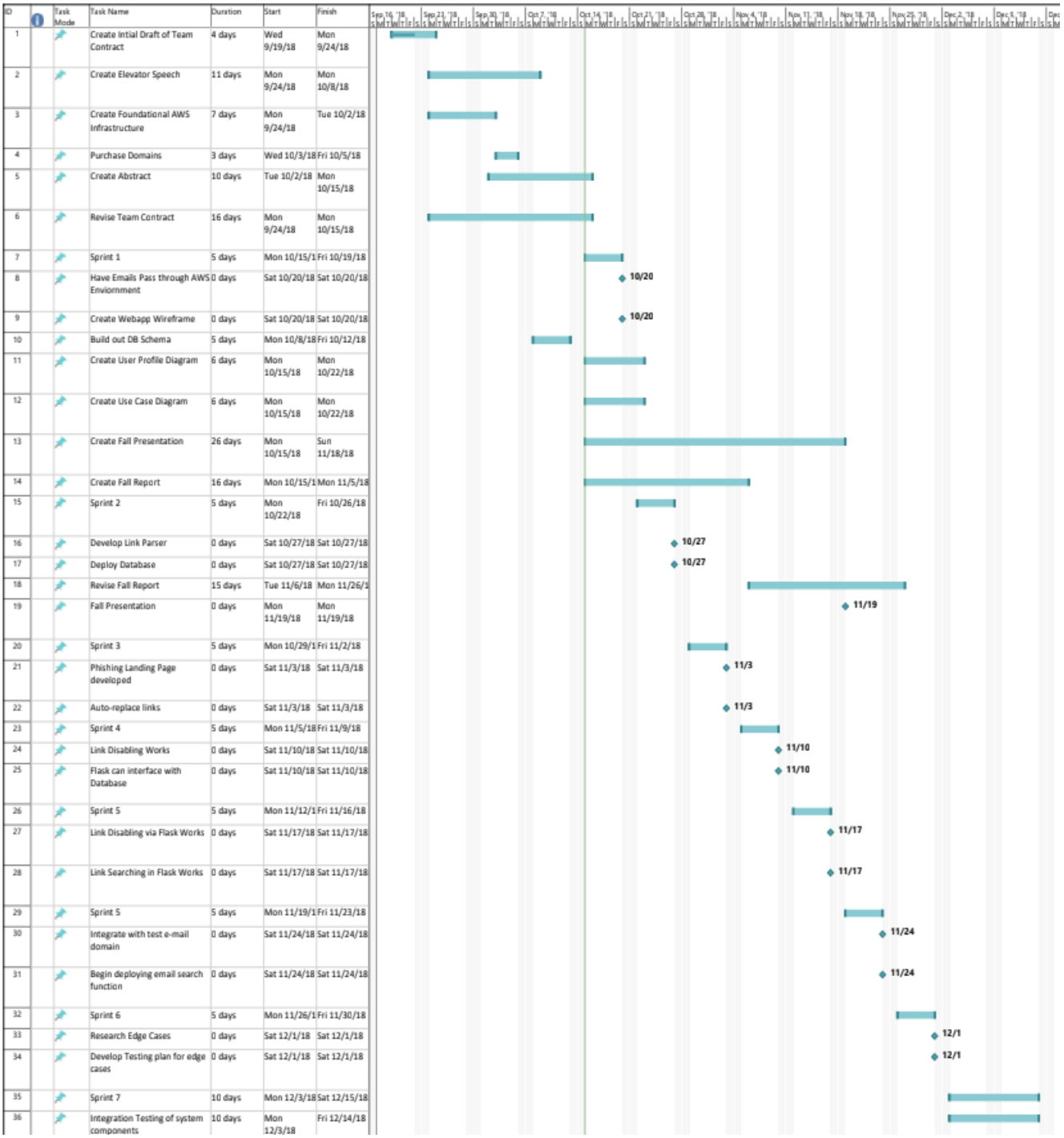
	AWS services:			
	Total Monthly Payment by Service:		$11.07	$7.21
		Total Cost:	$18.28	
		Total Budgeted Amount:	$20.11	

Table 2.7.1: **Project Budget**

The ability to exactly pinpoint all costs is another major benefit to the serverless deployment strategy. The native toolset within AWS also provides for the ability to accurately forecast costs based on demand, and also retroactively examine usage and bills to ensure that services are delivered in the most cost-effective way possible.

2.8 Project Schedule

Hypersafe was developed within the time constraints of the Information Technology Senior Design Practicum series, which occurs over two semesters. This timeline is illustrated in figure 2.8.1 on the next page:



ID	Task Mode	Task Name	Duration	Start	Finish
1		Create Intial Draft of Team Contract	4 days	Wed 9/19/18	Mon 9/24/18
2		Create Elevator Speech	11 days	Mon 9/24/18	Mon 10/8/18
3		Create Foundational AWS Infrastructure	7 days	Mon 9/24/18	Tue 10/2/18
4		Purchase Domains	3 days	Wed 10/3/18	Fri 10/5/18
5		Create Abstract	10 days	Tue 10/2/18	Mon 10/15/18
6		Revise Team Contract	16 days	Mon 9/24/18	Mon 10/15/18
7		Sprint 1	5 days	Mon 10/15/1	Fri 10/19/18
8		Have Emails Pass through AWS Enviornment	0 days	Sat 10/20/18	Sat 10/20/18
9		Create Webapp Wireframe	0 days	Sat 10/20/18	Sat 10/20/18
10		Build out DB Schema	5 days	Mon 10/8/18	Fri 10/12/18
11		Create User Profile Diagram	6 days	Mon 10/15/18	Mon 10/22/18
12		Create Use Case Diagram	6 days	Mon 10/15/18	Mon 10/22/18
13		Create Fall Presentation	26 days	Mon 10/15/18	Sun 11/18/18
14		Create Fall Report	16 days	Mon 10/15/1	Mon 11/5/18
15		Sprint 2	5 days	Mon 10/22/18	Fri 10/26/18
16		Develop Link Parser	0 days	Sat 10/27/18	Sat 10/27/18
17		Deploy Database	0 days	Sat 10/27/18	Sat 10/27/18
18		Revise Fall Report	15 days	Tue 11/6/18	Mon 11/26/1
19		Fall Presentation	0 days	Mon 11/19/18	Mon 11/19/18
20		Sprint 3	5 days	Mon 10/29/1	Fri 11/2/18
21		Phishing Landing Page developed	0 days	Sat 11/3/18	Sat 11/3/18
22		Auto-replace links	0 days	Sat 11/3/18	Sat 11/3/18
23		Sprint 4	5 days	Mon 11/5/18	Fri 11/9/18
24		Link Disabling Works	0 days	Sat 11/10/18	Sat 11/10/18
25		Flask can interface with Database	0 days	Sat 11/10/18	Sat 11/10/18
26		Sprint 5	5 days	Mon 11/12/1	Fri 11/16/18
27		Link Disabling via Flask Works	0 days	Sat 11/17/18	Sat 11/17/18
28		Link Searching in Flask Works	0 days	Sat 11/17/18	Sat 11/17/18
29		Sprint 5	5 days	Mon 11/19/1	Fri 11/23/18
30		Integrate with test e-mail domain	0 days	Sat 11/24/18	Sat 11/24/18
31		Begin deploying email search function	0 days	Sat 11/24/18	Sat 11/24/18
32		Sprint 6	5 days	Mon 11/26/1	Fri 11/30/18
33		Research Edge Cases	0 days	Sat 12/1/18	Sat 12/1/18
34		Develop Testing plan for edge cases	0 days	Sat 12/1/18	Sat 12/1/18
35		Sprint 7	10 days	Mon 12/3/18	Sat 12/15/18
36		Integration Testing of system components	10 days	Mon 12/3/18	Fri 12/14/18

Figure 2.8.1 ***Fall Semester Project Schedule and Gantt Chart***

The focus of the Fall Semester WAS on developing and implementing the foundational architecture, as well as developing a working proof of concept. Goals were significantly shorter, as there was much more work to do at a lower difficulty level.

37		Winter Break	21 days	Mon 12/17/18	Sat 1/12/19
38		Bug Repair	10 days	Mon 1/14/19	Fri 1/25/19
39		Sprint 8	5 days	Mon 1/28/19	Fri 2/1/19
40		Flask TLD Searching works	0 days	Fri 2/1/19	Fri 2/1/19
41		Integrate a second e-mail domain	0 days	Fri 2/1/19	Fri 2/1/19
42		Full Stack Integration Testing	20 days	Mon 2/4/19	Fri 3/1/19
43		Prepare for Final Presentation	21 days	Mon 3/4/19	Mon 4/1/19
44		Prepare for IT Expo	21 days	Mon 3/4/19	Mon 4/1/19
45		Prepare Final Paper	21 days	Mon 3/4/19	Mon 4/1/19

Figure 2.8.2 ***Spring Semester Project Schedule and Gantt Chart***

The Spring Semester Gantt Chart has significantly longer goals due to the longer iterations of testing, and also identification and development of edge cases.

2.9 Testing

This section explains in detail the testing that was conducted for development of Hypersafe and each of its three core services (Email Delivery, Link Resolution, and the Web Application). Every effort was given to ensure that each of the three distinct services that make up the Hypersafe platform works as intended in as many conditions as possible.

2.9.1 Testing Methodology

Since Hypersafe is built upon the AWS platform, much of the underpinning infrastructure is code. Due to this, all Hypersafe development is done using an Agile methodology, regardless of which part of the project is being used. This ensures that Hypersafe can remain nimble given any set new requirements to be presented into the platform. Furthermore, many of the facts of Hypersafe are entirely managed by AWS (e.g. mail routing), ensuring that the project team can focus on making the best application possible. Given this, the methodology was to test at the lowest level possible (unit testing) since all integration and system testing is handled by AWS, as most

components are either in a functional state, or a dysfunctional state upon instantiation within the platform.

2.9.2 Testing Scope

The scope of this testing covers edge to end delivery of an email message, and also device to edge resolution of a link. This ensures that our platform can receive messages, parse and replace links, and finally resolve links to their intended target. Since Hypersafe is built upon the AWS platform, all edge routing and the last hop (from an external mailer) is out of scope as that is exclusively handled by AWS. End-user mail server configuration is also out of scope, as no end-users directly interact with the platform.

2.9.3 Testing Objective

As with any service, the objective of Hypersafe in this regard is to work in the intended fashion 100% of the time with the least amount of added burden to the user. Given the variable nature of email messages and lack consistently applied standard protocols, this will unfortunately not always be the case. Furthermore, Hypersafe needs to remain stable and nimble while under load, due to the ubiquitous nature of email in the modern workplace. Simply put, the objective of our testing is to ensure that the service works as intended and as quickly as possible in as many situations as possible.

2.9.4 Email Delivery Service Testing

This service has two primary goals: ensure that incoming messages are being accepted and forwarded to the intended destinations, and also to find and replace links within the message. Email acceptance and delivery is the responsibility of AWS, and as the Figure 2.9.4.1 shows, our service has remained stable, as there have been no

bounces (Hypersafe delivering messages to mailboxes that don't exist), rejects (downstream mail servers refusing to accept mail from Hypersafe), or user complaints due to our system being in place:

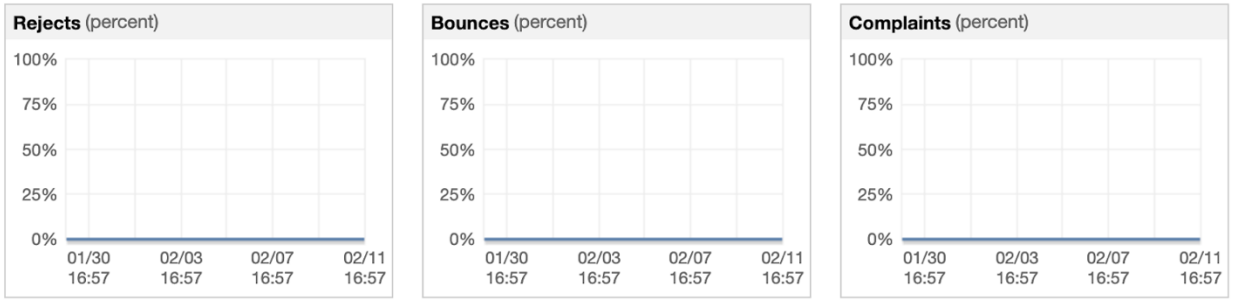


Figure 2.9.4.1: ***Email Reject, Bounce, and Complaint Rates***

Fail conditions for this portion of the service are as follows:

- The Reject Rate rises above 5% of all messages sent
- The Bounce Rate rises above 5% of all messages sent
- The Complaint Rate rises above 0.005% of all messages sent

If a fail condition were to appear, the project team would need to interface with AWS directly to solve the problem at hand.

The second major component of this service, finding and replacing links, has also shown promising results. It is impossible to know every link that will ever come through our platform, however the *Alexa top 500 List* contains a representative sample of the most popular sites on the web. To test this, the project team ran these 500 sites through

the function that detects links in the Email Delivery Service. As evidenced in Figure 2.9.5.2, Hypersafe was able to detect all links present in the test.

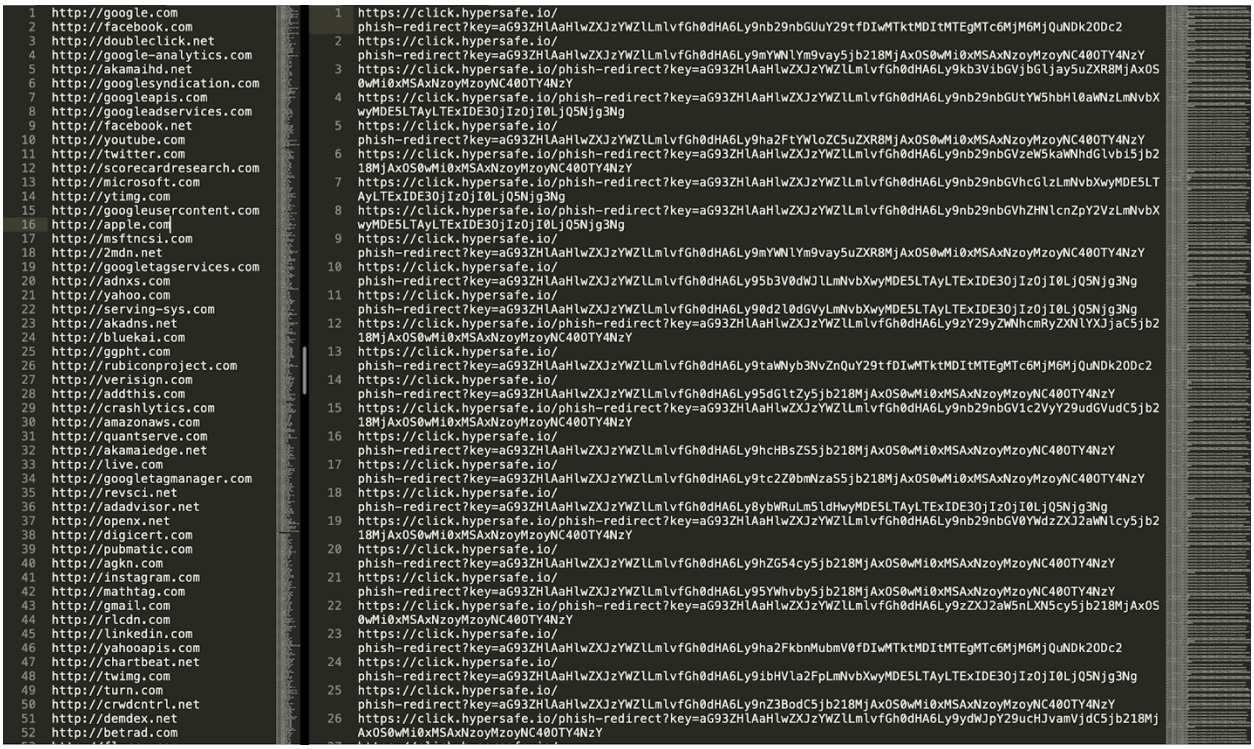


Figure 2.9.5.2: ***Link Detection Test Results***

To ensure that the Email Delivery Service is servicing requests in an appropriate time, the timing of the entire processing batch is recorded. Figure 2.9.4.3 shows the processing time of the entire detection test, with Hypersafe averaging 0.00008 seconds per link:

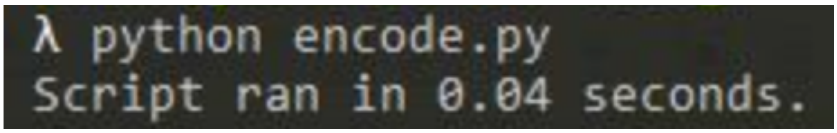


Figure 2.9.4.3: ***Total Test Runtime***

This test is run every time any change is made to the detect function of the Email Delivery Service, and any results outside of the below parameters are considered a failure:

- Less than 99% of the links were detected
- Processing time is greater than 0.00025 seconds per link

When a fail condition manifests, Hypersafe employs platform debugging tools (such as AWS X-Ray) to pinpoint the root cause to enable a speedy resolution of the problem.

2.9.5 Link Resolution Service Testing

Unlike the Email Delivery Service, the Link Resolution Service only has one goal: resolve links to their intended target or resolve the end-user to the education and awareness page. This service relies most heavily on service offerings that are 100% managed by AWS, thus reducing the scope of testing drastically. For example, the education and awareness page's uptime, stability, and distribution is handled by AWS, and thus out of scope. Figure 2.5.9.1 shows that there have been no AWS-side issues since Hypersafe's instantiation that have had a negative service effect on the platform,

due to the redundant nature of both AWS, and also Hypersafe spanning multiple Availability Zones:

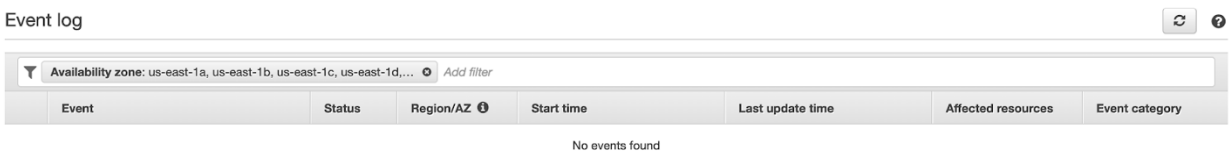


Figure 2.9.5.1: *AWS Service Health Dashboard*

Since the majority of this service is AWS-managed, the Hypersafe-specific components are inherently binary in testing (i.e. if the link resolves, the service/test is in a passing state, if the link does not resolve, the service/test is in a failed state). AWS runs service status checks constantly and will alert Hypersafe Administrators at the first signs of trouble.

2.9.6 Hinder Testing

The Web Application is the most open-ended component of Hypersafe as it is the only component that non-Hypersafe administrators directly interact with. Since there is a fundamentally different set of needs here, the testing objectives and strategy have shifted to directly reflect this change.

First, extra attention was given to cross-platform interoperability, so extensive testing was done in Edge, Chrome and Firefox to ensure that each engine will render the application correctly. This testing was achieved by manually matching the design draft to what was presented within each browser. A consistent experience within each

browser is a passing condition, while any inconsistencies in the user experience would be considered a fail. Further testing was done to ensure that the design is responsive, and Figure 2.5.9.1 shows the desktop experience in Firefox, while figure 2.5.9.2 shows the mobile experience in Chrome. Note the overall consistency in the design elements that prevail through two different rendering engines and screen sizes.

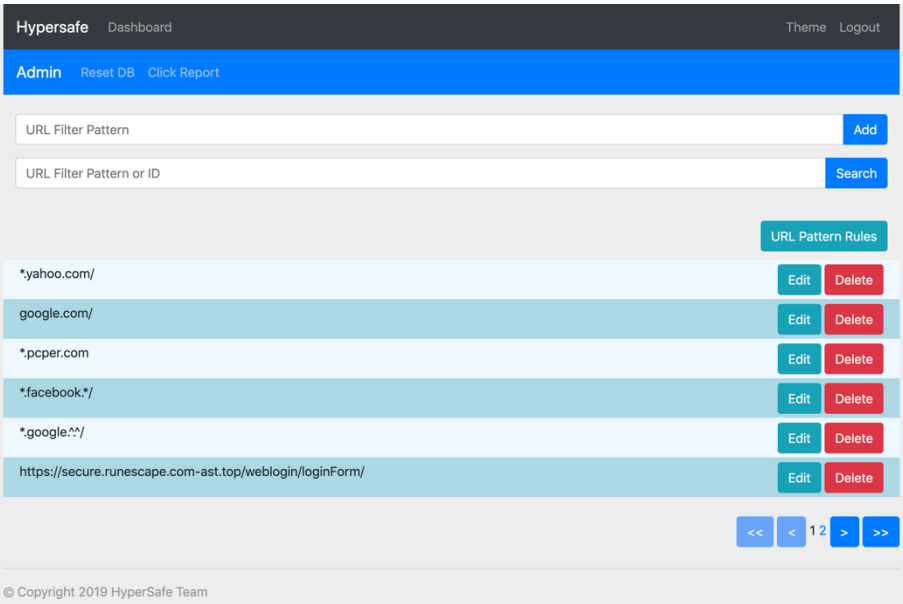


Figure 2.5.9.1 ***Full Size Desktop Experience in Firefox***

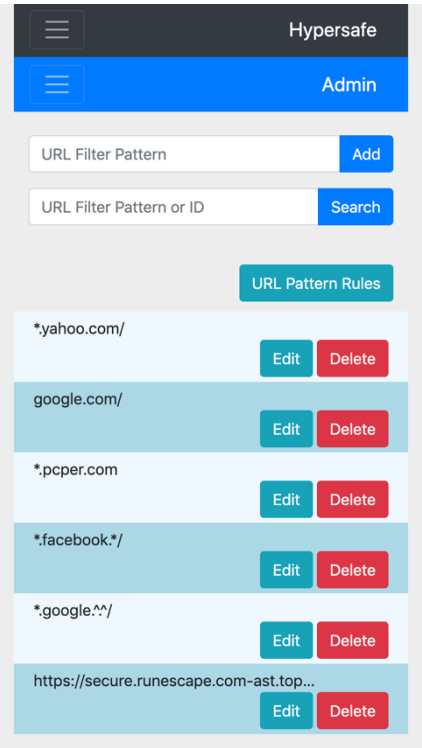


Figure 2.5.9.2 *Mobile Experience in Chrome*

All functional tests for the site are performed on code commits, and each test will fail unless the following criteria are met:

- There are no dead links on the site
- A known input is placed into all areas of user input. If anything other than the expected output is returned, the test will fail.

Finally, the project team wanted to ensure that the use cases of the Web Application are actually being fulfilled. With this, members of the team manually assigned the different roles to themselves within the system (user, administrator) and then ensured that all critical functionality (such as adding a link to a blacklist or inspecting email traffic) worked in the intended manner.

2.10 Problems Encountered

While the development of Hypersafe has seen a very low number of issues arise due to all of the infrastructure being code, and the binary nature of the AWS service offerings, that is not to say that the platform has been without issues. Most of the issues encountered were when services were initially instantiated, and once the service started “working”, the system would be in what is a considered a working state. Below is an example of one of the more novel issues that the project team has dealt with:

When testing the Email Delivery Service with the Alexa top 500 list, finding and encoding links in Hypertext Markup Language (HTML) based emails caused validation or viewing issues in the HTML after the links have been encoded. To resolve this issue a more complete regular expression was developed to better identify links and logic was included to not encode links that are specified in a HTML tags “src” attribute.

Additionally, the AWS learning curve was a substantial problem for the project team. Given that two of the three project team members had limited to no interaction with the platform before, learning the syntax, vernacular, and concepts in addition to linking services together was a significant challenge. Once those specific challenges were overcome, then understanding the data and developing acceptable pass/fail metrics was also a challenge, as AWS will let a lot of crazy things happen within the platform, so while a specific component may work, ensuring that it is optimized and working in the most appropriate manner was a continuing learning exercise for the team.

Finally, the project team found the documentation for AWS Cognito to be severely lacking. The most significant development hurdle was figuring out how this service works and needs to interact with the rest of the technology stack. Once this issue was solved though, future integrations were trivial to implement.

2.11 Future Plans/Improvements

If the project were to be started again today, more serious consideration would need to be given to Microsoft Azure. At the time of the beginning of this project, Azure lacked the services and resources to be able to effectively execute this project in a serverless manner successfully. Given the ever-changing nature of public cloud providers, and the meteoric rise in popularity of Office 365, potential performance and costs savings can be achieved by building this product on Azure. Additionally, the ability to filter outbound messages would be a nice improvement to make as well, as only inbound messages are currently filtered and processed.

If there was more time, the project team would have attempted to integrate artificial intelligence (AI) in order better determine the validity of a message, and to use that AI to contribute to automatically block link resolution.

Finally, the project team does plan to meet with staff members of the 1819 Innovation Hub to assess the viability of commercializing this project.

Conclusion

Phishing messages are a problem that all organizations face, and this problem is not going away any time soon. Hypersafe takes a novel and innovative approach in

protecting organizations from this menace via a completely serverless architecture and ensuring that every click on every device for every user is protected. This tool is a valuable asset in each customer's tool box to ensure resiliency against these kinds of attacks.

3.1 Lessons Learned During Spring Semester

Perhaps the biggest lesson learned during the spring semester was learning time management. Every member of the project team had significant competing time commitments, yet the project was still able to be completed on time and on budget. Additionally, the team was able to fully learn how bootstrap works, and how AWS Cognito works.

3.2 Lessons Learned During Fall Semester

The project team was able to learn how AWS works on a foundational level, and how it is possible to intercept email messages via MX record weighting. Additionally, the project team was able to learn how to create an API using the Amazon API Gateway.

3.3 Skills Developed

The majority of the project team was able to learn how to deploy highly available applications within AWS, and how over 20 AWS services work. Secondly, the team learned how to manage a project from start to finish. Finally, all project team members learned how bootstrap works to control how to format a webapp.

3.4 Items Completed Since Fall Semester

Most of the end-user facing components of Hypersafe were completed in the spring semester. All of Hinder, the informational website, Content Delivery Network (CDN) acceleration, and technical documented was completed in this semester.

3.5 Lessons Learned from IT Expo

IT Expo was an extremely valuable experience where the project team was able to get practice not only showcasing this project, but also explain this project to the non-technical person. Many people within the IT industry can easily grasp the issue, however the layperson may not even be aware of the problem, let alone a solution to it. Finally, the project team was also able to assess that this project would be a prime candidate for commercialization from the feedback from the community at IT Expo.

Appendix A: Works Cited

Widup, Suzanne & Spitler, Marc & Hylender, David & Bassett, Gabriel. (2018). 2018 Verizon Data Breach Investigations Report.

Cofense. "Enterprise Phishing Resiliency and Defense Report." PDF file, 2017.

Amazon Inc. (2019, April 15). AWS Glossary. Retrieved from AWS Documentation: https://docs.aws.amazon.com/general/latest/gr/glos-chap.html

Wineburg, Niel. "How to blunt spear phishing attacks." Network World, IDG, 6 Mar. 2013, www.networkworld.com/article/2164139/network-security/how-to-blunt-spear-phishing-attacks.html.

Appendix B: List of Acronyms

- **AI-** Artificial Intelligence
- **API-** Application Programming Interface
- **CDN-** Content Delivery Network
- **DKIM-** Domain Keys Identified Mailer
- **DNS**- Domain Naming System
- **FIFO-** First in First Out
- **HTML-** Hypertext Markup Language
- **LAMP-** Linux, Apache, MariaDB, PHP
- **MX-** Mail Exchange
- **URL-** Uniform Resource Locator
- **TLS-** Transport Layer Security
- **SPF-** Sender Policy Framework

Appendix C: Amazon Web Services Definitions

Below is a list of all of the Amazon Web Services (AWS) that are used within Hypersafe. All definitions are directly from Amazon's Service Definition Glossary.

- **Amazon Certificate Manager** is a fully managed service for requesting and renewing TLS certificates with AWS Services.
- **Amazon Simple Notification Service** enables users, devices, and services to publish and receive notifications from the cloud.
- **Amazon Simple Queue Service (SQS)** is a reliable and scalable hosted queues for storing messages as they travel between computers.
- **Amazon API Gateway** is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale.
- **Amazon Web Services (AWS)** is an infrastructure web services platform in the cloud for companies of all sizes.
- **AWS Cost Management** provides detailed usage and billing reports pertaining to the use of AWS services and infrastructure.
- **AWS Key Management Service** facilitates the generation and use of unique security keys across various AWS services.
- **AWS Management Console** allows for web-based access to the management consoles for all AWS resources.
- application load balancer
- **Availability Zones** are distinct locations within a Region that is insulated from failures in other Availability Zones, and provides inexpensive, low-latency network connectivity to other Availability Zones in the same Region.
- **Cognito** is a web service that makes it easy to save mobile user data, such as app preferences or game state, in the AWS cloud without writing any back-end code or managing any infrastructure. Amazon Cognito offers mobile identity management and data synchronization across devices.
- **CloudFront** is an AWS content delivery service that helps you improve the performance, reliability, and availability of your websites and applications.
- **CloudTrail** is a web service that records AWS API calls for your account and delivers log files to you. The recorded information includes the identity of the API caller, the time of the API call, the source IP address of the API caller, the request parameters, and the response elements returned by the AWS service.
- **CloudWatch** is a web service for monitoring and troubleshooting systems and applications from existing system, application, and custom log files. Existing log files to CloudWatch Logs can be send and monitor these logs in near real-time.
- **Dynamo DB** is a fully managed NoSQL database service that provides fast and predictable performance with seamless scalability.

- **Elastic Beanstalk** is a web service for deploying and managing applications in the AWS Cloud without worrying about the infrastructure that runs those applications.
- **Elastic Block Store (EBS)** is a service that provides block level storage volumes for use with EC2 instances.
- **Elastic Compute Cloud (EC2)** is a web service that enables you to launch and manage Linux/UNIX and Windows server instances in Amazon's data centers.
- **Identity and Access Management** is a web service that enables Amazon Web Services (AWS) customers to manage users and user permissions within AWS.
- **Lambda** is a web service that lets you run code without provisioning or managing servers. You can run code for virtually any type of application or back-end service with zero administration. You can set up your code to automatically trigger from other AWS services or call it directly from any web or mobile app.
- **Network ACLs** are an optional layer of security that acts as a firewall for controlling traffic in and out of a subnet. You can associate multiple subnets with a single network ACL, but a subnet can be associated with only one network ACL at a time.
- **Regions** are a named set of AWS resources in the same geographical area. A Region comprises at least two Availability Zones.
- **Route 53** is a web service you can use to create a new DNS service or to migrate your existing DNS service to the cloud.
- **Simple Storage Service (S3)** Storage for the internet. You can use it to store and retrieve any amount of data at any time, from anywhere on the web.
- **Simple Email Service (SES)** is an easy-to-use, cost-effective email solution for applications.
- **Virtual Private Cloud (VPC)** is a web service for provisioning a logically isolated section of the AWS cloud where AWS resources can be launched in a virtual network that the customer defines. The customer controls their virtual networking environment, including selection of their own IP address range, creation of subnets, and configuration of route tables and network gateways.

Appendix D: Tech Expo Poster

University of CINCINNATI

Hypersafe

College of Education,
Criminal Justice,
and Human Services
School of Information Technology

Tony Iacobelli | Anthony Burchette | Peter Johnson | Advisor: Bo Vykhovanyuk | Team 02

What is Hypersafe?

» Hypersafe is a collection of three distinct services that helps bolster your organization's phishing defenses.

The Phish Problem

» It is not a matter of if, but when your organization will be phished. The average user receives an average of 16 malicious emails a month, costing organizations $675 million annually.

How Hypersafe Works

Email Delivery

Someone sends a message to a recipient protected by Hypersafe

Hypersafe processes the message and makes the links safe

Hypersafe delivers the message to the intended recipient with the safe links

Link Resolution

A User clicks on a link in an email

Hypersafe examines the link, redirecting the user based on its validity

Safe link – User redirected to intended website
Malicious Link – User redirected to education and awareness resources

Web Administration

Administrators log into Hinder-Hypersafe's Web Admin Portal

Administrators can view and control all aspects of link resolution

All settings are synchronized across all Hypersafe services

Try Hypersafe

» Navigate to **hypersafe.io/demo** to see how easy Hypersafe makes protecting your organization from phishing.

Under the Hood

» Hypersafe is built upon the proven Amazon Web Services (AWS) platform. Hypersafe is modular and entirely serverless, allowing for a cost-efficient experience that is fast, secure, and scalable.

Appendix E: Code Digest

Below is the source code for all Lambda Functions within the Hypersafe Platform.

SimpleEmailFilter.py

```
from __future__ import print_function

import boto3
import json
import os
import re
import base64
import email
from email.mime.text import MIMEText
import email.encoders
from datetime import datetime

# old pattern. Don't use. Eats trailing html tags
# pattern = '(http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-
fA-F][0-9a-fA-F]))+)'
# other old pattern. Don't use. Does not match things like uc.edu (No
leading http:// or https://)
# pattern = '(?i)(src=\"|xmlns(:.)?=\")?(https?:[;\/?\\@&=+$,\[\]A-Za-z0-
9\-_\.\!\~\*\'\(\)%][\;\/\?\:\@\&\=\+\$\,\[\]A-Za-z0-9\-
_\.\!\~\*\'\(\)%#]*|[KZ]:\\*.*\w+)'
# pattern =
'(?i)(src=\"|xmlns(:.)?=\")?(http:\/\/www\.|https:\/\/www\.|http:\/\/|http
s:\/\/)?[a-z0-9]+([\-\.]{1}[a-z0-9]+)*\.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?'
pattern = '(?i)(src=\"|style=\"|xmlns(:.)?=\")?((?:[a-z][\w-
]+:(?:/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-
z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]
+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:\'\".,<>?]))'


def encoder(data):
    encoded = base64.urlsafe_b64encode(data)
    tmp = encoded.replace('=', '')
    return tmp


# Update the emailDomain environment variable to the correct domain, e.g.
<MYDOMAIN>.com
EMAIL_DOMAIN = os.environ['emailDomain']


def print_with_timestamp(*args):
    print(datetime.utcnow().isoformat(), *args)


def lambda_handler(event, context):
    print_with_timestamp('Starting - inbound-ses-spam-filter')
```

```
    ses_notification = event['Records'][0]['ses']
    message_id = ses_notification['mail']['messageId']
    receipt = ses_notification['receipt']

    print_with_timestamp('Processing message:', message_id)

    # Check if any spam check failed
    if (receipt['spfVerdict']['status'] == 'FAIL' or
            receipt['dkimVerdict']['status'] == 'FAIL' or
            receipt['spamVerdict']['status'] == 'FAIL' or
            receipt['virusVerdict']['status'] == 'FAIL'):

        send_bounce_params = {
            'OriginalMessageId': message_id,
            'BounceSender': 'mailer-daemon@{}'.format(EMAIL_DOMAIN),
            'MessageDsn': {
                'ReportingMta': 'dns; {}'.format(EMAIL_DOMAIN),
                'ArrivalDate': datetime.now().isoformat()
            },
            'BouncedRecipientInfoList': []
        }

        for recipient in receipt['recipients']:
            send_bounce_params['BouncedRecipientInfoList'].append({
                'Recipient': recipient,
                'BounceType': 'ContentRejected'
            })

        print_with_timestamp('Bouncing message with parameters:')
        print_with_timestamp(json.dumps(send_bounce_params))

        try:
            ses_client = boto3.client('ses')
            bounceResponse = ses_client.send_bounce(**send_bounce_params)
            print_with_timestamp('Bounce for message ', message_id, '
sent, bounce message ID: ', bounceResponse['MessageId'])
            return {'disposition': 'stop_rule_set'}
        except Exception as e:
            print_with_timestamp(e)
            print_with_timestamp('An error occurred while sending bounce
for message: ', message_id)
            raise e
    else:
        s3 = boto3.client('s3')
        to_email = ses_notification['mail']['commonHeaders']['to'][0]
        from_address = ses_notification['mail']['commonHeaders']['from']
        response = s3.get_object(Bucket='store.email.hypersafe.io',
Key=message_id)
        emailcontent = response['Body'].read().decode('utf-8')
        emailcontent = re.sub(r"^Return-Path:.*$", 'Return-Path:
<howdy@hypersafe.io>', emailcontent, flags=re.MULTILINE)
        emailcontent = re.sub(r"^[Ff][Rr][Oo][Mm]:.*$", 'From: Hypersafe
Team <howdy@hypersafe.io>', emailcontent, flags=re.MULTILINE)
```

```
        emailcontent = re.sub(r"^[Tt][Oo]:.*$", "To: " + from_address[0],
emailcontent, flags=re.MULTILINE)
        # FIXED: Wonky looking links seem to be coming from trying to
parse the links from a MIME block instead of straight text.
        #        The MIME blocks can cut links in half causing the search
and replace to fail. Need to find a way to convert the multi
        #        parts (text and html) of the MIME block to straight text,
process them, and convert them back.
        # TODO:  Discuss what scenarios links should be replaced. Look at
Duke Energy email Anthony sent through.
        #        It has broken images because links in <img src=? tags
were replaced.
        msg = email.message_from_string(emailcontent)
        base_url = 'https://click.hypersafe.io/check-link?key='
        # base_url = 'https://click.hypersafe.io/phish-redirect?key='
        orig_emailcontent = emailcontent
        for part in msg.walk():
            if part.get_content_type().startswith('text/'):
                part_content = part.get_payload(decode=True)
                orig_part_content = part_content
                for m in re.finditer(pattern, part_content):
                    if m:
                        if not m.group(0).startswith('src="') and not
m.group(0).startswith('xmlns') and not m.group(0).startswith('style="'):
                            url = re.sub('<.*>', '', m.group(0),
flags=re.MULTILINE)
                            print_with_timestamp('Found URL:
{0}'.format(url))
                            # data = '|'.join([str(to_email), str(url),
str(datetime.now())])
                            data = '|'.join([str(from_address[0]),
str(url), str(datetime.now())])
                            enc = encoder(data)
                            user_url = base_url + enc
                            part_content = part_content.replace(url,
user_url, 1)
                            # email[:m.start()] + email[m.end():]
                            # offset = abs(len(user_url) - len(url))
                            # part_content = part_content[:m.start()] +
user_url + part_content[m.end():]
                            print_with_timestamp('Replacing {0} with
{1}'.format(url, user_url))
                part.set_payload(part_content)
                email.encoders.encode_quopri(part)
                part.__delitem__('Content-Transfer-Encoding')
                part.add_header('Content-Transfer-Encoding', 'quoted-
printable')
        s3.put_object(Bucket='store.email.hypersafe.io',
                      Key=message_id + '_mod',
                      Body=msg.as_string())
        try:
            ses_client = boto3.client('ses')
            source = 'Hypersafe Team <howdy@hypersafe.io>'
```

```
            ses_client.send_raw_email(RawMessage={'Data': msg.as_string(),
},
                                      Destinations=from_address,
                                      Source=source)
        except Exception as e:
            print_with_timestamp(e)
            raise e
```

Searchlinks.py

```
import json
import boto3
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError


# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)


def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('hypersafe_master')

    try:
        fields = ['id', 'url']
        items = []
        hashes = []
        q = event['query']
        if len(q) > 0:
            fe = 'id = :x OR contains(#u, :x)'
            eav = {':x': event['query']}
            ean = {'#u': 'url'}
            response = table.scan(FilterExpression=fe,
                                  ExpressionAttributeValues=eav,
                                  ExpressionAttributeNames=ean)
            items = response['Items']
            while 'LastEvaluatedKey' in response:
                esk = response['LastEvaluatedKey']
                response = table.scan(ExclusiveStartKey=esk,
                                      FilterExpression=fe,
                                      ExpressionAttributeValues=eav,
                                      ExpressionAttributeNames=ean)
                items.extend(response['Items'])
        else:
            response = table.scan()
            items = response['Items']
```

```
            while 'LastEvaluatedKey' in response:
                esk = response['LastEvaluatedKey']
                response = table.scan(ExclusiveStartKey=esk)
                items.extend(response['Items'])
    except ClientError as e:
        print(e.response['Error']['Message'])
        return 500
    else:
        # print("searchLinks succeeded:")
        # print(json.dumps(items, indent=4, cls=DecimalEncoder))
        return items
```

getClickReport.py

```
import json
import boto3
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError


# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)


def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('hypersafe_click_report')

    try:
        items = []
        response = table.scan()
        items = response['Items']
        while 'LastEvaluatedKey' in response:
            esk = response['LastEvaluatedKey']
            response = table.scan(ExclusiveStartKey=esk)
            items.extend(response['Items'])
    except ClientError as e:
        print(e.response['Error']['Message'])
        return 500
    else:
        return items
```

editItem.py

```
import json
import boto3
import decimal
from boto3.dynamodb.conditions import Key, Attr
```

```
from botocore.exceptions import ClientError


# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)


def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('hypersafe_master')

    try:
        for i in event:
            # print(i['id'])
            response = table.update_item(
                Key={
                    'id': i['id']
                },
                UpdateExpression="set #u = :r",
                ExpressionAttributeValues={
                    ':r': i['url'],
                }, ExpressionAttributeNames={
                    '#u': 'url',
                },
                ReturnValues="UPDATED_NEW"
            )
    except ClientError as e:
        print(e.response['Error']['Message'])
        return 500
    else:
        print("RemoveFromDB succeeded:")
        print(json.dumps(event, indent=4, cls=DecimalEncoder))
        return 200
```

verifyToken.py

```
import urllib
import json


def lambda_handler(event, context):
    print(event)
    return {"statusCode": 200, "body": str(event)}
```

searchClickReport.py

```
import json
import boto3
```

```
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError


# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)


def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('hypersafe_click_report')

    try:
        fields = ['id', 'url', 'Redirected', 'owner', 'time_clicked']
        items = []
        hashes = []
        q = event['query']
        if len(q) > 0:
            fe = 'id = :x OR contains(#u, :x) OR contains(#r, :x) OR
contains(#o, :x) OR contains(#tc, :x)'
            eav = {':x': event['query']}
            ean = {'#u': 'url', '#r': 'Redirected', '#o': 'owner', '#tc':
'time_clicked'}
            response = table.scan(FilterExpression=fe,
                                  ExpressionAttributeValues=eav,
                                  ExpressionAttributeNames=ean)
            items = response['Items']
            while 'LastEvaluatedKey' in response:
                esk = response['LastEvaluatedKey']
                response = table.scan(ExclusiveStartKey=esk,
                                      FilterExpression=fe,
                                      ExpressionAttributeValues=eav,
                                      ExpressionAttributeNames=ean)
                items.extend(response['Items'])
        else:
            response = table.scan()
            items = response['Items']
            while 'LastEvaluatedKey' in response:
                esk = response['LastEvaluatedKey']
                response = table.scan(ExclusiveStartKey=esk)
                items.extend(response['Items'])
    except ClientError as e:
        print(e.response['Error']['Message'])
        return 500
    else:
        # print("searchLinks succeeded:")
```

```
        # print(json.dumps(items, indent=4, cls=DecimalEncoder))
        return items
```

removeFromDB.py

```
import json
import boto3
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError


# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)


def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('hypersafe_master')

    try:
        for i in event:
            response = table.delete_item(Key=i)
    except ClientError as e:
        print(e.response['Error']['Message'])
        return 500
    else:
        print("RemoveFromDB succeeded:")
        print(json.dumps(event, indent=4, cls=DecimalEncoder))
        return 200
```

listAllLinks.py

```
import json
import boto3
import decimal
from boto3.dynamodb.conditions import Key, Attr
from botocore.exceptions import ClientError


# Helper class to convert a DynamoDB item to JSON.
class DecimalEncoder(json.JSONEncoder):
    def default(self, o):
        if isinstance(o, decimal.Decimal):
            if o % 1 > 0:
                return float(o)
            else:
                return int(o)
        return super(DecimalEncoder, self).default(o)
```

```
def lambda_handler(event, context):
    dynamodb = boto3.resource('dynamodb')
    table = dynamodb.Table('hypersafe_master')

    try:
        response = table.scan()
        print(response)
    except ClientError as e:
        print(e.response['Error']['Message'])
    else:
        # print(response)
        items = response['Items']
        print("list-all succeeded:")
        print(json.dumps(items, indent=4, cls=DecimalEncoder))
        return items
```

demo_simple_email_filter.py

```
from __future__ import print_function

import boto3
import json
import os
import re
import base64
import email
from email.mime.text import MIMEText
import email.encoders
from datetime import datetime

# old pattern. Don't use. Eats trailing html tags
# pattern = '(http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-
fA-F][0-9a-fA-F]))+)'
# other old pattern. Don't use. Does not match things like uc.edu (No
leading http:// or https://)
# pattern = '(?i)(src=\"|xmlns(:.)?=\")?(https?:[;\/?\\@&=+$,\[\]A-Za-z0-
9\-_\.\!\~\*\'\(\)%][\;\/\?\:\@\&\=\+\$\,\[\]A-Za-z0-9\-
_\.\!\~\*\'\(\)%#]*|[KZ]:\\*.*\w+)'
# pattern =
'(?i)(src=\"|xmlns(:.)?=\")?(http:\/\/www\.|https:\/\/www\.|http:\/\/|http
s:\/\/)?[a-z0-9]+([\-\.]{1}[a-z0-9]+)*\.[a-z]{2,5}(:[0-9]{1,5})?(\/.*)?'
pattern = '(?i)(src=\"|style=\"|xmlns(:.)?=\")?((?:[a-z][\w-
]+:(?:/{1,3}|[a-z0-9%])|www\d{0,3}[.]|[a-z0-9.\-]+[.][a-
z]{2,4}/)(?:[^\s()<>]+|\(([^\s()<>]+|(\([^\s()<>]+\)))*\))+(?:\(([^\s()<>]
+|(\([^\s()<>]+\)))*\)|[^\s`!()\[\]{};:\'\".,<>?]))'


def encoder(data):
    encoded = base64.urlsafe_b64encode(data)
    tmp = encoded.replace('=', '')
    return tmp
```

```
# Update the emailDomain environment variable to the correct domain, e.g.
<MYDOMAIN>.com
EMAIL_DOMAIN = os.environ['emailDomain']


def print_with_timestamp(*args):
    print(datetime.utcnow().isoformat(), *args)


def lambda_handler(event, context):
    print_with_timestamp('Starting - inbound-ses-spam-filter')

    ses_notification = event['Records'][0]['ses']
    message_id = ses_notification['mail']['messageId']
    receipt = ses_notification['receipt']

    print_with_timestamp('Processing message:', message_id)

    # Check if any spam check failed
    if (receipt['spfVerdict']['status'] == 'FAIL' or
            receipt['dkimVerdict']['status'] == 'FAIL' or
            receipt['spamVerdict']['status'] == 'FAIL' or
            receipt['virusVerdict']['status'] == 'FAIL'):

        send_bounce_params = {
            'OriginalMessageId': message_id,
            'BounceSender': 'mailer-daemon@{}'.format(EMAIL_DOMAIN),
            'MessageDsn': {
                'ReportingMta': 'dns; {}'.format(EMAIL_DOMAIN),
                'ArrivalDate': datetime.now().isoformat()
            },
            'BouncedRecipientInfoList': []
        }

        for recipient in receipt['recipients']:
            send_bounce_params['BouncedRecipientInfoList'].append({
                'Recipient': recipient,
                'BounceType': 'ContentRejected'
            })

        print_with_timestamp('Bouncing message with parameters:')
        print_with_timestamp(json.dumps(send_bounce_params))

        try:
            ses_client = boto3.client('ses')
            bounceResponse = ses_client.send_bounce(**send_bounce_params)
            print_with_timestamp('Bounce for message ', message_id, '
sent, bounce message ID: ', bounceResponse['MessageId'])
            return {'disposition': 'stop_rule_set'}
        except Exception as e:
            print_with_timestamp(e)
            print_with_timestamp('An error occurred while sending bounce
for message: ', message_id)
            raise e
```

```
    else:
        s3 = boto3.client('s3')
        to_email = ses_notification['mail']['commonHeaders']['to'][0]
        from_address = ses_notification['mail']['commonHeaders']['from']
        response = s3.get_object(Bucket='store.email.hypersafe.io',
Key=message_id)
        emailcontent = response['Body'].read().decode('utf-8')
        emailcontent = re.sub(r"^Return-Path:.*$", 'Return-Path:
<demo@hypersafe.io>', emailcontent, flags=re.MULTILINE)
        emailcontent = re.sub(r"^[Ff][Rr][Oo][Mm]:.*$", 'From: Hypersafe
Team <demo@hypersafe.io>', emailcontent, flags=re.MULTILINE)
        emailcontent = re.sub(r"^[Tt][Oo]:.*$", "To: " + from_address[0],
emailcontent, flags=re.MULTILINE)
        # FIXED: Wonky looking links seem to be coming from trying to
parse the links from a MIME block instead of straight text.
        #        The MIME blocks can cut links in half causing the search
and replace to fail. Need to find a way to convert the multi
        #        parts (text and html) of the MIME block to straight text,
process them, and convert them back.
        # TODO:  Discuss what scenarios links should be replaced. Look at
Duke Energy email Anthony sent through.
        #        It has broken images because links in <img src=? tags
were replaced.
        msg = email.message_from_string(emailcontent)
        # base_url = 'https://click.hypersafe.io/check-link?key='
        base_url = 'https://click.hypersafe.io/phish-redirect?key='
        orig_emailcontent = emailcontent
        for part in msg.walk():
            if part.get_content_type().startswith('text/'):
                part_content = part.get_payload(decode=True)
                orig_part_content = part_content
                for m in re.finditer(pattern, part_content):
                    if m:
                        if not m.group(0).startswith('src="') and not
m.group(0).startswith('xmlns') and not m.group(0).startswith('style="'):
                            url = re.sub('<.*>', '', m.group(0),
flags=re.MULTILINE)
                            print_with_timestamp('Found URL:
{0}'.format(url))
                            # data = '|'.join([str(to_email), str(url),
str(datetime.now())])
                            data = '|'.join([str(from_address[0]),
str(url), str(datetime.now())])
                            enc = encoder(data)
                            user_url = base_url + enc
                            part_content = part_content.replace(url,
user_url, 1)
                            # email[:m.start()] + email[m.end():]
                            # offset = abs(len(user_url) - len(url))
                            # part_content = part_content[:m.start()] +
user_url + part_content[m.end():]
                            print_with_timestamp('Replacing {0} with
{1}'.format(url, user_url))
                part.set_payload(part_content)
```

```
                email.encoders.encode_quopri(part)
                part.__delitem__('Content-Transfer-Encoding')
                part.add_header('Content-Transfer-Encoding', 'quoted-
printable')
        s3.put_object(Bucket='store.email.hypersafe.io',
                      Key=message_id + '_mod',
                      Body=msg.as_string())
        try:
            ses_client = boto3.client('ses')
            source = 'Hypersafe Team <demo@hypersafe.io>'
            ses_client.send_raw_email(RawMessage={'Data': msg.as_string(),
},
                                      Destinations=from_address,
                                      Source=source)
        except Exception as e:
            print_with_timestamp(e)
            raise e
```