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 Result**: 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:

- **Resource access from s3 website**
- **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:

- **Account login**
- **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:

- **File extraction**
- **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:** After initial entry, the users permissions needs to fit the predf
- **Expected Result:**

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

TODO mention use of github for version tracking

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.

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.