Building Cloud-Based Automated Response Systems

Mishka McCowan

Building Cloud-Based Automated Response Systems

GIAC (GMON) Gold Certification

Author: Mishka McCowan, mmcocwan@eagna.net

Advisor: David Hoelzer

Accepted: April 15, 2019

Abstract

When moving to public cloud infrastructures such as Amazon Web Services (AWS), organizations gain access to tools and services that enable automated responses to specific threats. This paper will explore the advantages and disadvantages of using native AWS services to build an automated response system. It will examine the elements that organizations should consider including developing the proper skills and systems that are required for the long-term viability of such a system.

1. Introduction

Today's information security professionals are facing the daunting task of securing an increasing number of systems from a growing number of threats and responding to those threats faster than ever before. To accomplish this, many information security teams are turning to automation to increase their effectiveness. While there is an ecosystem of commercial tools to facilitate automation, budget constraints can force organizations to consider developing their own solutions.

The area in which organizations most often face this dilemma is Infrastructure-as-a-Service cloud providers. Attracted by the flexibility and scalability, companies are rushing to both migrate existing infrastructure from data centers and build new services in the cloud in equal measure. Security professionals are expected to secure these environments just as they did traditional data centers.

The choice to buy or build a solution is often a question of budgetary resources. Building a solution can be very attractive because of the low incremental cost of using the IaaS vendor-supplied tools for scanning, alerting, and automating responses. Using these tools, it is possible to build a system that will detect and respond to specific security events and conditions, relieving security personnel from the pressure to respond quickly and efficiently.

This paper will explore the viability of creating an automated response system in Amazon Web Services (AWS). Automating responses can allow for remediation that is significantly faster and tailored to the specific needs of the organization. Using public cloud resources puts the ability to design, build, and deploy scalable, automated responses within reach of most organizations.

Mishka McCowan, mmccowan@eagna.net

2. Response as Code

2.1. Technical Requirements

There are fundamental features that a public cloud must support for an automated response system to work effectively – a well-defined API, facilities for automation within the cloud itself, and events that can be used to trigger the automation. These features provide both the stimulus that triggers the response as well as the capacity for the response itself. AWS offers a catalog of services that meet these criteria.

A fundamental part of AWS is the Application Programming Interface (API) that is used to control all its services. It is a RESTful API that communicates via standard HTTP verbs such as PUT, GET, and POST. AWS offers several tools that interact with their API's including software development kits (SDK) for languages such as Java, Python, and Node.js and a command line interface (CLI). Developers and administrators can use these tools to create, describe, update, and delete resources within AWS.

AWS offers many services that can be used to create an automated response. Security-specific services such as GuardDuty[1], Macie,[2] and Inspector[3] generate high-value events and require minimal configuration. The events are then consumed by CloudWatch, the AWS service for monitoring cloud resources and applications. Rules can be configured within CloudWatch that evaluate each event against specific criteria. When an event matches, the rule triggers a specific response such as a Lambda[4] function or Batch[5] job which contain custom code that executes the actual response. For example, a certain number of failed login attempts to the AWS management console generates a GuardDuty finding that is logged in CloudWatch. A CloudWatch rule that matches that finding type would trigger a Lambda function that disables the account.

Many different AWS services can handle the actual response. These include Lambda functions, AWS Batch jobs, SNS topics, SQS queues, and many others. They are building blocks that can be used in many different combinations depending on the requirements of the automation. They can be used alone or in conjunction with each other depending on the nature

[1] Amazon GuardDuty is a threat detection service that continuously monitors for malicious activity and unauthorized behavior. https://aws.amazon.com/guardduty/

[2] Macie is a security service that uses machine learning to automatically doscover, classify, and protect sensitive data in AWS. https://aws.amazon.com/macie/

[3] Inspector is an automated security assessment service that helps improve the security and compliance of applications deployed on AWS. https://aws.amazon.com/inspector/

[4] Lambda runs code without provisioning or managing servers. https://aws.amazon.com/lambda/

[5] Batch is a fully managed batch processing AWS service. https://aws.amazon.com/batch/

and complexity of the response required. A simple response might require a single Lambda function while a more complex example may require a queue, multiple Lambda functions controlled by a Step Function.

2.2. Skill Requirements

In the 2008 film Taken, the film's protagonist, played by Liam Neeson, has a phone conversation with his daughter's kidnapper in which he asserts that he has "a very particular set of skills" that will make him a nightmare for the kidnapper. To create an automated system to thwart attackers, information security professionals must similarly develop a very particular set of skills including application development, project management, and cloud infrastructure management. Depending on the backgrounds of the individual team members, existing security teams may already possess some measure of skill in these areas. Additionally, if the organization is already engaged in software development, the security team may be able to leverage those existing competencies to complete the work. However, if the security team does not have the necessary background, and there are no organizational resources, then the team will need to develop those skills.

2.2.1 Application Development Skills

Application development represents both the largest and most technical set of skills for building an automated response capability; it is the umbrella under which other skills such as programming, systems design, and systems integration. Building these competencies within a team requires assessing the current skill levels and establishing proficiency targets. Software development organizations may already have methodologies in place to measure competency. Other organizations can either develop methodologies or use a standard such as the Digital, Data, and Technology Profession Capability Framework (DDTPCF).

The DDTPCF defines four skill levels for developers that describe the requirements for each level. They range from Awareness, "Knows the skill," to Expert, "Has knowledge and experience in the application of this skill." There is a progression

from being aware of the skill to demonstrating a deep level of proficiency that allows for flexibility when assessing an individual's expertise.

The DDTPCF was developed by the UK Government's Government Digital Service for identifying skills gaps and designing specific activities to address them (Ellis, 2017). It first defines a set of essential skills for software developers. These skills include development process optimization, modern standards approach, programming a build (software engineering), service support, systems design, systems integration, and user focus. Additional desirable skills include availability and capacity management, information security, and prototyping. They are designed to encompass the range of skills required for a variety of software development positions.

The skills are then combined with standardized job descriptions and job titles, ranging from Apprentice Developer to Principal Developer, to form a grid. The intersection of each title and skill are assigned one of the four skill levels. The result is a simple, qualitative method for assessing both the current proficiency of an individual and the required level of programming skill required for a project.

When creating and maintaining an automated response system, the standard title of Developer or Senior Developer should be used as a reference for a given skill. Organizations with a lower risk tolerance could adopt titles from the higher end of the scale such as Lead Developer or Principal Developer as their reference, as the higher levels of skill required would reduce the number of programming and design errors. Organizations can also prioritize some skills over others by defining a higher job title. For example, an organization might define Developer as the baseline level of proficiency yet require that skills in information security and systems design be at the Lead Developer level. Thus, the level of skill can be tailored to the specific amount of risk the organization is willing to accept.

2.2.2 Software Testing

Closely related to application development is quality assurance testing, which also includes security testing. Testing is an integral part of the Software Development Life Cycle (SDLC) and examines a combination of people, process, and technology in the

application or system. A comprehensive testing regime will include threat modeling, code reviews, and penetration testing (OWASP, 2014).

While the application developers conduct a certain amount of testing, a separate software testing function is essential. It allows the code to be inspected by someone with fresh eyes and a different perspective. The effect is similar to the difficulty a writer experiences when trying to edit their work. When they try to edit something that they have just finished writing, the tendency is to skip over errors because they are too familiar with what they want it to say (The Writing Center). An application developer can experience the same effect. By knowing what the code should do, they may miss seeing bugs that have been introduced.

The DDTPCF defines the role for a test engineer who "undertakes test planning activity including discovery, capture, definition of tests, and estimating test effort as part of a broader risk-based approach" (Digital, Data and Technology Profession, 2017). Like the skills required for application development, the level of skill required for software testing will be a function of the risk appetite of an organization. The DDTPCF has a grid of skills and job titles that can be used to select the appropriate level of skill if the organization does not already have a method for doing so.

2.2.3 Other Skills

In addition to application development and testing, building an automated response system also requires some proficiency with project management, change management, and release management. Many security teams will have some level of experience with these disciplines, though their role in the process may be different. For example, in the change management process, the role of the security team is to reduce the risks posed by the proposed changes. By deploying and updating an automated response system, the security team becomes the system owner. While this should not be an unfamiliar role for the security team, it is worth noting because of the additional overhead it requires.

Security teams will likely either have access to a project management function within the organization or will have some level of expertise developed through implementing other security projects. If there is no in-house expertise, a decision is

needed regarding the methodology. Will the team take a waterfall approach or an agile approach? If it selects agile, will the team use Scrum, Kanban, Lean, or one of the other frameworks for agile development (Wikipedia, 2019)? The pro and cons of each are beyond the scope of this paper. The security team should seek information on the various types and select the one that best fits their culture and goals.

Release management is also a discipline the security team may have experience with if there is an in-house software development function. In those cases, they may be able to leverage the existing processes that are already in place. As with change management, their role in the process will change as they move from being responsible for reducing risk in the process of deploying code.

2.2.4 Security Considerations

If the responsibility for building, managing, and deploying the automated response system falls to the security team, the team will need to examine the controls in place. Is there adequate separation of duty between the various roles in the development process (AICPA, n.d.)? Are those who develop code also responsible for deploying it? Who is providing the security reviews? For a small team, developing answers to these questions can be challenging.

Depending on the risk appetite of the organization, several different compensating controls can be used. Automated deployments can include static and dynamic code scans. They can also be configured to create highly detailed logs of what was deployed and who deployed it. Additionally, users can be granted very limited roles for deploying code into production.

2.2.5. Long-Term Maintenance

The final consideration is the cost of long-term maintenance of the solution. In March of 2019, AWS released more than 120 updates to its various services (Amazon Web Services 2019). With changes occurring at a rate of nearly four per day, it can be a daunting task to ensure that the changes do not affect the application. Additionally, changes in the environment by other developers and administrators can similarly impact the efficiency and operation of the solution.

3. Building the Automation

Automating a response within AWS can be broken down into several use cases. Two of the most common involve responding to an event generated by a native AWS service and responding to a custom event defined in CloudWatch. Creating automated responses for these two use cases can provide a template for responding to similar events in AWS.

The examples will make use of the following AWS resources and services:

Figure 1: AWS Resources

Resource	Description
GuardDuty	A threat detection service that continuously monitors for malicious activity and unauthorized behavior. https://aws.amazon.com/guardduty/
CloudWatch	A repository for logs and metrics for AWS resources running on AWS. Alarms can be triggered by CloudWatch events. https://aws.amazon.com/cloudwatch/
IAM	The Identity and Access Management service is used to securely manage access to AWS services and resources. https://aws.amazon.com/iam/
Lambda function	Runs code without provisioning or managing servers. https://aws.amazon.com/lambda/

3.1. Example: Responding to a GuardDuty Alert

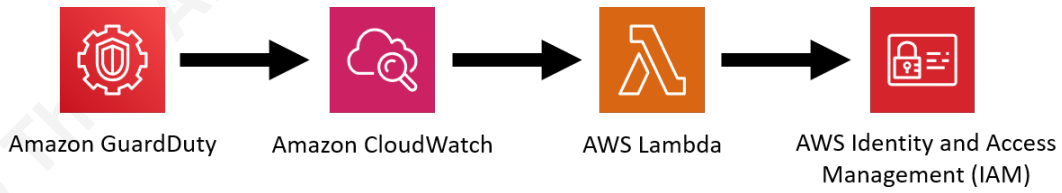
The goal of this example is to respond to an alert generated by a native AWS service, GuardDuty. According to AWS, GuardDuty is "a threat detection service that continuously monitors for malicious activity and unauthorized behavior." Once enabled, it will generate alerts for a predefined set of findings. There are currently eleven finding types – backdoor, behavior, cryptocurrency, pentest, persistence, policy, recon, resource consumption, stealth, trojan, and unauthorized access.

This example will make use of the unauthorized access finding for an unusual console login. The AWS designation for this finding is UnauthorizedAccess:IAMUser/ConsoleLogin. It detects when a login by a principal differs from its established baseline. For instance, a user logging in from an unusual geographic location would generate this finding. The automated response would be to place an account that has been flagged by this finding by removing all its privileges, effectively disabling the account until the finding can be investigated.

3.1.1. Building the Response

In this example, unauthorized access will trigger a finding in GuardDuty, which will, in turn, cause a CloudWatch rule to fire. The rule will call a Lambda function that contains the logic for the response. In this case, it will first determine the type of GuardDuty finding, extract the user from the finding data, and then apply a policy that denies all permissions to that user. Figure 2 shows the overall flow of the response:

Figure 2: Response Flow



3.1.2. IAM Policies and Roles

The first step is to create an IAM policy that explicitly denies all privileges. In IAM, an explicit deny for a permission will override an allow permission. By creating a policy that denies all permissions, any IAM account it is attached to will no longer be able to view or change information through the AWS console or CLI.

The policy can be created by pasting the following text into the JSON tab on the Create Policy tab of the Create Policy wizard in IAM. It contains four major components: the SID, Effect, Action, and Resource. The SID is an optional identifier that can be provided to help classify the policy. The Effect specifies whether the statement allows or denies the Action. The Action element describes the action that a user will or will not be

allowed to perform. Each AWS service defines a set of actions. Examples of actions include changing passwords in IAM or listing the active instances in EC2. The Resource element defines one or more objects that the statement applies to, such as which users or groups can change passwords in IAM.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": " DenyAllPrivsPolicy ",
            "Effect": "Deny",
            "Action": "*",
            "Resource": "*"
        }
    ]
}
```

Step-by-step directions for creating the policy can be found in the AWS documentation at: https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_create.html#access_policies_create-json-editor. In this example the policy will be named DenyAllPrivsPolicy though this is not a requirement.

Once DenyAllPrivsPolicy has been created, a second policy and a role will need to be created. The new policy will be attached to a role which will, in turn, be attached to the Lambda function. This will grant permissions to the Lambda function to attach a policy to a user and to create logs in CloudWatch. The first permission will allow the Lambda to attach the DenyAllPrivsPolicy to the user identified by the GuardDuty finding. The other permissions will allow it to write log entries to CloudWatch, which is essential for both troubleshooting and auditing.

To create the new policy, use the following JSON and name it AttachPolicyForLambda:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
```

```
            "Sid": " AttachPolicyForLambda ",
            "Effect": "Allow",
            "Action": "iam:AttachUserPolicy",
            "Resource": "arn:aws:iam::*:user/*"
        },
        {
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:PutLogEvents"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

Next, create a role using the instructions in the AWS documentation found at https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_create_for-service.html. Name the role AttachPolicyRole and, when prompted, associate the AttachPolicyForLambda policy with the role.

3.1.3. Lambda Function

The Lambda function contains the logic for the response. It first extracts the type of event and name of the user attempting the login from the GuardDuty finding. It then attaches the DenyAllPrivsPolicy to that user.

To create the Lambda, click Create Function in the AWS console. Next, click the "Author from scratch" button at the top of the screen. Enter the following information in the form that appears and then click the Create Function button.

Field	Value
Function Name	removePrivsFunction

Runtime	Python 3.6
Execution Role	Use an existing role
Existing Role	AttachPolicyRole

In the screen that appears next, scroll down to the Function code section of the page. The highlighted portion of the code below needs to be changed to the AWS account number where the account will be run. Once that is done, replace the default code in the editor with the code below:

```
import json
import boto3

def lambda_handler(event, context):
    evt = json.dumps(event)
    j=json.loads(evt)
    eventType=j[0]['type']
    user=j[0]['resource']['accessKeyDetails']['userName']
    print("eventType: "+eventType)
    print("user: "+user)
    iam = boto3.client('iam')
    response = iam.attach_user_policy(
    UserName='Bob',
    PolicyArn='arn:aws:iam::111111111111:policy/
DenyAllPrivsPolicy)
    return {
        'statusCode': 200,
        'body': json.dumps('OK')
    }
```

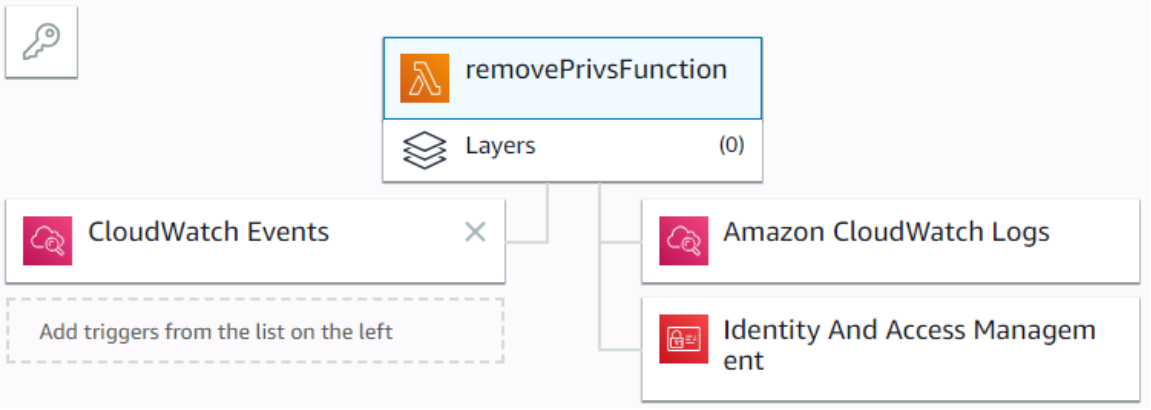
3.1.4. Configure CloudWatch

The last step in the setup process is to configure CloudWatch. In the AWS console, go to the CloudWatch configuration screen and click on the Rules link under Events in the left-hand navigation. Next, click the Create Rule button at the top of the

screen. Set the value of Service Name to GuardDuty and the Event Type to All Events. On the left-hand side of the screen, click the Add target button in the Targets section. Select "Lambda function" in the first drop-down. Underneath should appear a drop-down named Function. Use it to select the name of the Lambda function that was created in the previous section, removePrivsFunction. When this is complete, click the Configure details button in the lower right-hand corner of the screen. In the screen that follows enter triggerResponse in the Name field and click the Create rule button in the lower right-hand corner of the screen. The screen will return to the list of CloudWatch rules in the account.

To validate that the CloudWatch event has been configured correctly, return to the Lambda service in the AWS console and click on the name of the Lambda created in the previous section: removePrivsFunction. The page should show CloudWatch Events as an input to removePrivsFunction and CloudWatch Logs and IAM as outputs.

Figure 3: Lambda Console



3.1.5. Testing the Response

The automated response was tested using the AWS CLI to generate test findings from GuardDuty. The CLI method is very efficient because it can generate the specific finding needed to trigger the response without having to set up an artificial situation in which a console login would trigger GuardDuty. It should be noted that in the Lambda function, the username was hard coded to bob. In a production environment, this value would be extracted from the finding. For testing purposes, it was hard coded to ensure that a production account would not be accidentally locked out.

The testing process examined two attributes of this solution – responsiveness and consistency. The solution needs to trigger the response consistently and should do so quickly. In 2018, the time to compromise a system was measured in minutes to hours (Verizon, 2018) and the median dwell time for an attacker on a network was 71 days (Fireeye, 2019). An automated response can limit that exposure significantly by immediately reacting to a threat.

The testing itself was split into two portions. The first tested the effectiveness of the solution by confirming that applying the policy would remove all permissions for that user even if they were currently logged in. The second portion of the test measured the response time of the solution. The AWS CLI was used to submit a test finding to GuardDuty. The elapsed time between the submission of the finding and the application of the policy to the test account was measured. The process was repeated a hundred times to measure the average response time. The tests were conducted in small batches on different days and at different times to accurately measure overall effectiveness.

The first step to set up the test is to create a test user. Go to IAM in the AWS console, click on Users in the left-hand navigation then click the "Add user" button at the top of the screen. On the next screen, enter "bob" as the user name and check the "AWS Management Console access" box before clicking next. On the Set permissions screen, click the "Attach existing policies direct" button at the top of the screen. In the lower portion of the screen, put a check next to the ReadOnlyAccess policy and click the "Next: Tags" button. Click the "Next: Review" button on the "Add tags" screen and then "Create User" on the final review screen. On the confirmation screen, click the "Download .csv" button to download the password for bob.

The next step is to configure the AWS CLI according to the instructions on the AWS web site at https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-configure.html. Once the CLI is configured, test it by listing the policies attached to bob using the following command:

```
C:\>aws iam list-attached-user-policies --user-name bob
```

If the user and CLI are both set up correctly, the following output should appear, showing that the AWS managed policy ReadOnlyAccess is attached to bob.

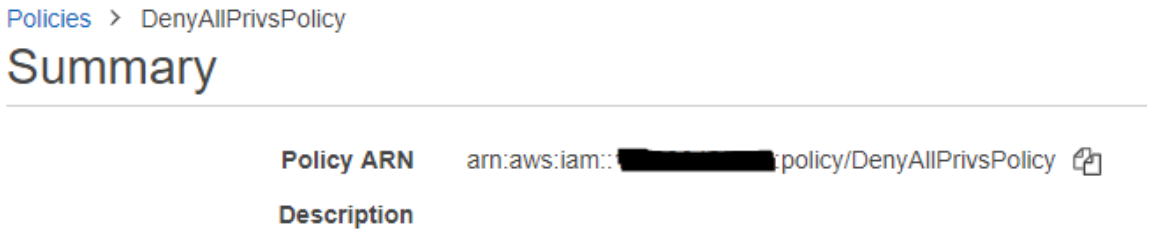
Mishka McCowan, mmccowan@eagna.net

```
{
 "AttachedPolicies": [
  {
   "PolicyName": "ReadOnlyAccess",
   "PolicyArn": "arn:aws:iam::aws:policy/ReadOnlyAccess"
  }
 ]
}
```

The first test can then be performed. Log into the AWS console using bob's credentials. Navigating to IAM in the console shows that bob can view other users, groups and roles. However, attempting to create or modify any of those elements results in an error saying that the user does not have permissions to perform that action. This validates that bob has read-only permissions for the console.

The next step is to manually apply the DenyAllPrivsPolicy previously created to bob using the CLI. First, the policy ARN for the custom policy is needed. In IAM, click on "Policies" in the left-hand navigation. In the screen that follows, filter for DenyAllPrivsPolcy and then click the name. The "Policy ARN" is displayed at the top of the screen, as shown below. Either click the icon to the left of the ARN to copy it or highlight the text and copy it manually. The ARN contains the account number of the AWS account (obfuscated below) and should be treated as sensitive information as seen in the figure below.

Figure 4: The ARN Displayed In IAM



The following CLI command is used to attach the policy to bob. The ARN should be replaced with the ARN copied in the previous step.

```
aws iam attach-user-policy --policy-arn
arn:aws:iam::1234567890:policy/DenyAllPrivsPolicy --user-
name bob
```

Once the policy has been attached, it can be verified using the list-attached-user-policies command below:

```
aws iam list-attached-user-policies --user-name bob

{
  "AttachedPolicies": [
    {
      "PolicyName": "ReadOnlyAccess",
      "PolicyArn": "arn:aws:iam::aws:policy/ReadOnlyAccess"
    },
    {
      "PolicyName": "DenyAllPrivsPolicy",
    "PolicyArn":
"arn:aws:iam::1234567890:policy/DenyAllPrivsPolicy"
    }
  ]
}
```

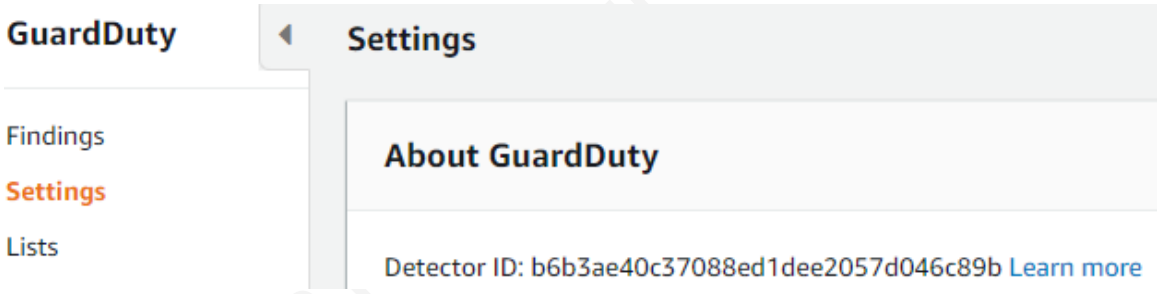
After confirming that both policies have been applied to bob's account, return to the browser session where bob is the active user or log in as bob. Visiting the same IAM pages results in an error message: "You do not have the permission required to perform this operation. Ask your administrator to add permissions." The explicit deny permissions in the DenyAllPrivsPolicy override the permissions granted in the ReadOnlyAccess policy. They take effect immediately and are applied to any current open sessions.

The next phase of the test is to validate that the automation works as expected. A test request will be submitted to GuardDuty, which will cause the automation to apply DenyAllPrivsPolicy to the bob account. In order to submit a test request, the `aws guardduty create-sample-findings` command will be used from the CLI. The command takes two arguments: detector-id and finding-types. The detector-id can be found in the AWS console, and the findings-type is

UnauthorizedAccess:IAMUser/ConsoleLogin as found in the AWS documentation at https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_unauthorized.html.

The detector-id is the unique identifier assigned by AWS to an active GuardDuty detector. Since GuardDuty can be activated per region, the detector-id will be different in each region under a given AWS account. To find the detector-id, open GuardDuty and click on Settings in the left-hand navigation. The detector-id for that region will be displayed at the top of the page.

Figure 5: GuardDuty Console



Once the detector-id has been located, the `create-sample-findings` command can be executed in the CLI.

```
aws guardduty create-sample-findings --detector-id
b6b3ae40cb37088ed1de32057d046c89b --finding-types
UnauthorizedAccess:IAMUser/ConsoleLogin
```

When run, the command will trigger a test finding in GuardDuty. The finding will be logged in CloudWatch where the previously created rule will forward it to the Lambda function for processing. After a short delay, the action will be logged into CloudWatch. These logs can be viewed by going to CloudWatch and clicking on “Logs.” On the right-hand portion of the screen, click on the name of the Log Group, /aws/lambda/removePrivsFunction, and then click on the most current log stream to see the log entries.

18:31:12	START RequestId: 3b3dcac3-be48-41d9-8547-45da368a31af Version: $LATEST
18:31:12	finding: {"version": "0", "id": "cb3c523e-f32a-a98e-07ba-05b6844674e7", "detail-type":
18:31:12	eventType: UnauthorizedAccess:IAMUser/ConsoleLogin
18:31:12	user: bob
18:31:13	END RequestId: 3b3dcac3-be48-41d9-8547-45da368a31af
18:31:13	REPORT RequestId: 3b3dcac3-be48-41d9-8547-45da368a31af Duration: 569.46 ms

Mishka McCowan, mmccowan@eagna.net

According to the log, the Lambda function was successfully executed. The time for the function to execute was 569.46 ms. This is notable because there was a much longer delay between the test request being submitted and the appearance of the logs in CloudWatch. Executing the `create-sample-findings` command followed by the `list-attached-user-policies` command shows that the DenyAllPrivsPolicy policy has not been attached to the bob account. However, subsequent executions of `list-attached-user-policies` showed that the policy is eventually attached.

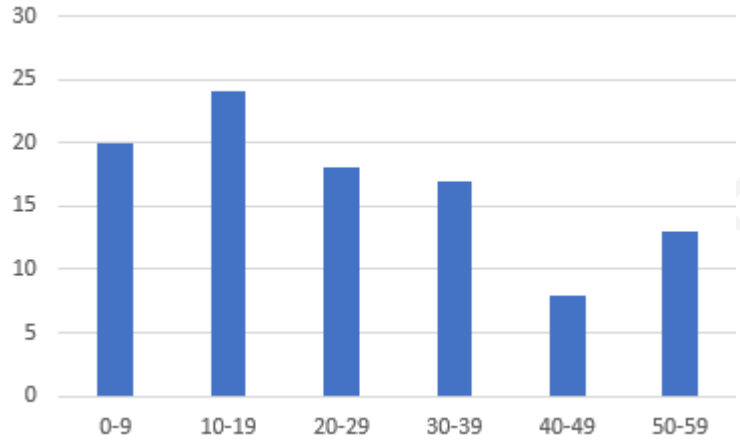
Because the timeliness of the response is critical, the final phase of testing worked to determine if the expected delay between the finding and the execution of the Lambda function. To measure the amount of time taken to complete the response, many test findings were submitted for processing. The starting time for each request was recorded. The end time was recorded from the CloudWatch logs. The difference between the two was then calculated for each submission.

The beginning time was recorded by using a small script that displayed the current time by echoing the Windows variable %TIME% and then executing the `create-sample-findings` command. The result was that the current time was displayed down to the hundredth of a second before executing the command. The time was recorded in an Excel spreadsheet. CloudWatch was then monitored for the START and END entries that denote the execution of the Lambda function. The timestamp on the END entry was then converted from UTC to EDT and recorded in the spreadsheet. The `list-attached-user-policies` command was executed to confirm that the policy had been attached. The process was repeated in batches of 10 on different days and at different times of day to capture any variation that might occur because of load on the underlying platform.

In all, 100 samples were taken over 10 days. The average time between the submission of the test request and the completion of the Lambda function was 29 seconds. The longest execution time was 59 seconds, and the shortest was 4 seconds. There was no appreciable variation in the timings based on the day or time of day that the requests were made. As can been seen in the chart below, overall, the timing tends to skew slightly towards 30 seconds or less.

Mishka McCowan, mmccowan@eagna.net

Figure 6: Response Times



The two most likely points at which latency is introduced into the process is the time it takes for the notification to move between components and the time to start up Lambda function. In the first case, there is latency in the processing messages between services. According to the AWS documentation, “Due to the distributed nature of the CloudWatch Events and the target services, the delay between the time the scheduled rule is triggered and the time the target service honors the execution of the target resource might be several seconds” (Amazon Web Services, n.d.). The second source of latency is often referred to as the “Cold Start” issue for Lambda functions (Malishev, 2018). According to the AWS documentation, “It takes time to set up an execution context and do the necessary "bootstrapping," which adds some latency each time the Lambda function is invoked” (Amazon Web Services, n.d.). Essentially extra time is required in order to instantiate the function the first time it is run. The instance remains in place for subsequent instantiations, so the “cold start” penalty is not assessed.

While an average response time of 29 seconds is longer than would be expected in a non-distributed system, it significantly improves on the compromise time of minutes to hours reported by Verizon (Verizon, 2018).

4 Conclusion

Building an automated system using the tools provided in AWS is a viable solution for decreasing the time needed to respond to an alert. The building blocks for creating the system are in place and require a relatively low level of skill to implement a

basic response. However, more robust solutions have additional considerations that organizations should be aware of before implementation.

As with any solution, the organization should be aware of the commitment to the long-term maintenance required when creating a robust, production-ready solution. A level of proficiency in software development and testing is required to ensure that the solution does not introduce additional vulnerabilities. The high rate of change in AWS requires constant monitoring and adjustments to maintain the viability of the system. Lastly, smaller teams will need to address the question of separation of duties as the lines between system administrator, developer, and security blur.

For organizations with the resources to create and maintain the system, building an automated response system can be a very effective way to both reduce the time needed to address an alert and to tailor the response to the specific organizational needs.

Mishka McCowan, mmccowan@eagna.net

References

AICPA (n.d.). Segregation of Duties. Retrieved from https://www.aicpa.org/interestareas/informationtechnology/resources/value-strategy-through-segregation-of-duties.html

Amazon Web Services (n.d.). Amazon GuardDuty. Retrieved from https://aws.amazon.com/guardduty/.

Amazon Web Services (2016, May 10). Automating Your AWS Security Operations. Retrieved from https://www.slideshare.net/AmazonWebServices/automating-your-aws-security-operations-61879668

Amazon Web Services (n.d.). AWS Lambda Execution Context. Retrieved from https://docs.aws.amazon.com/lambda/latest/dg/running-lambda-code.html

Amazon Web Services (2019, April). What's New. Retrieved from https://aws.amazon.com/about-aws/whats-new/2019/

Amazon Web Services (n.d.). Schedule Expressions for Rules. Retrieved from https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/ScheduledEvents.html

Chuvakin, Anton (2018, February 22). Our Security Orchestration and Automation (SOAR) Paper Publishes. Retrieved from https://blogs.gartner.com/anton-chuvakin/2018/02/22/our-security-orchestration-and-automation-soar-paper-publishes/

Duffy, Stan (2017, May). Why your QA team really should be part of your application security team. Retrieved from https://puppet.com/blog/why-your-qa-team-really-should-be-part-your-application-security-team.

Ellis, Holly (2017, July). Building digital, data, and technology capability for government. Retrieved from https://www.gov.uk/government/publications/senior-developer-skills-they-need/senior-developer-skills-they-need

Engelbrecht, Stan (2018, July 27). The Evolution of SOAR Platforms. Retrieved from https://www.securityweek.com/evolution-soar-platforms

FireEye (2019). M-Trends 2019. Retrieved from https://content.fireeye.com/m-trends

Mishka McCowan, mmccowan@eagna.net

Hainly, Jeremiah (2017, March). Auto-Nuke It from Orbit: A Framework for Critical Security Control Automation. Retrieved from https://www.sans.org/reading-room/whitepapers/incident/auto-nuke-orbit-framework-critical-security-control-automation-37687

Hulme, George V. (2014, May 14). DevOps: Getting Past Audit. Retrieved from: https://devops.com/devops-getting-past-audit/

Malishev, Nathan (2018, April 19). Lambda Cold Starts, A Language Comparison. Retrieved from https://medium.com/@nathan.malishev/lambda-cold-starts-language-comparison-%EF%B8%8F-a4f4b5f16a62

OWASP (2014). OWASP Testing Guide v4, accessed at https://www.owasp.org/index.php/OWASP_Testing_Guide_v4_Table_of_Contents.

Radichel, Teri. (2016, March). Balancing Security and Innovation with Event Driven Automation. Retrieved from https://www.sans.org/reading-room/whitepapers/incident/balancing-security-innovation-event-driven-automation-36837

Verizon (2018). 2018 Data Breach Investigations Report, 11th Edition. Retrieved from https://enterprise.verizon.com/resources/reports/dbir/

Wikipedia (2019, April). Agile software development. Retrieved from https://en.wikipedia.org/wiki/Agile_software_development.

The Writing Center, University of North Carolina at Chapel Hill. (n.d.) Editing and Proofreading. Retrieved from https://writingcenter.unc.edu/tips-and-tools/editing-and-proofreading/.

Digital, Data, and Technology Profession. (2017, March). Test Engineer: Role Description. Retrieved from https://www.gov.uk/government/publications/test-engineer-role-description/test-engineer-role-description

Digital, Data, and Technology Profession. (2017, April). Software Developer Roles: Skill Levels. Retrieved from https://www.gov.uk/government/publications/software-developer-roles-skill-levels/software-developer-roles-skill-levels

Last Updated: August 24th, 2019

Upcoming SANS Training

Click here to view a list of all SANS Courses

SANS Tampa-Clearwater 2019	Clearwater, FLUS	Aug 25, 2019 - Aug 30, 2019	Live Event
SANS Copenhagen August 2019	Copenhagen, DK	Aug 26, 2019 - Aug 31, 2019	Live Event
SANS Philippines 2019	Manila, PH	Sep 02, 2019 - Sep 07, 2019	Live Event
SANS Munich September 2019	Munich, DE	Sep 02, 2019 - Sep 07, 2019	Live Event
SANS Brussels September 2019	Brussels, BE	Sep 02, 2019 - Sep 07, 2019	Live Event
SANS Canberra Spring 2019	Canberra, AU	Sep 02, 2019 - Sep 21, 2019	Live Event
SANS Network Security 2019	Las Vegas, NVUS	Sep 09, 2019 - Sep 16, 2019	Live Event
SANS Oslo September 2019	Oslo, NO	Sep 09, 2019 - Sep 14, 2019	Live Event
SANS Dubai September 2019	Dubai, AE	Sep 14, 2019 - Sep 19, 2019	Live Event
SANS Paris September 2019	Paris, FR	Sep 16, 2019 - Sep 21, 2019	Live Event
Oil & Gas Cybersecurity Summit & Training 2019	Houston, TXUS	Sep 16, 2019 - Sep 22, 2019	Live Event
SANS Rome September 2019	Rome, IT	Sep 16, 2019 - Sep 21, 2019	Live Event
SANS Raleigh 2019	Raleigh, NCUS	Sep 16, 2019 - Sep 21, 2019	Live Event
SANS Bahrain September 2019	Manama, BH	Sep 21, 2019 - Sep 26, 2019	Live Event
SANS San Francisco Fall 2019	San Francisco, CAUS	Sep 23, 2019 - Sep 28, 2019	Live Event
SANS London September 2019	London, GB	Sep 23, 2019 - Sep 28, 2019	Live Event
SANS Dallas Fall 2019	Dallas, TXUS	Sep 23, 2019 - Sep 28, 2019	Live Event
SANS Kuwait September 2019	Salmiya, KW	Sep 28, 2019 - Oct 03, 2019	Live Event
SANS Northern VA Fall- Reston 2019	Reston, VAUS	Sep 30, 2019 - Oct 05, 2019	Live Event
SANS Cardiff September 2019	Cardiff, GB	Sep 30, 2019 - Oct 05, 2019	Live Event
SANS Tokyo Autumn 2019	Tokyo, JP	Sep 30, 2019 - Oct 12, 2019	Live Event
SANS DFIR Europe Summit & Training 2019 - Prague Edition	Prague, CZ	Sep 30, 2019 - Oct 06, 2019	Live Event
Threat Hunting & Incident Response Summit & Training 2019	New Orleans, LAUS	Sep 30, 2019 - Oct 07, 2019	Live Event
SANS Riyadh October 2019	Riyadh, SA	Oct 05, 2019 - Oct 10, 2019	Live Event
SIEM Summit & Training 2019	Chicago, ILUS	Oct 07, 2019 - Oct 14, 2019	Live Event
SANS October Singapore 2019	Singapore, SG	Oct 07, 2019 - Oct 26, 2019	Live Event
SANS Lisbon October 2019	Lisbon, PT	Oct 07, 2019 - Oct 12, 2019	Live Event
SANS San Diego 2019	San Diego, CAUS	Oct 07, 2019 - Oct 12, 2019	Live Event
SANS Baltimore Fall 2019	Baltimore, MDUS	Oct 07, 2019 - Oct 12, 2019	Live Event
SANS Doha October 2019	Doha, QA	Oct 12, 2019 - Oct 17, 2019	Live Event
SANS Denver 2019	Denver, COUS	Oct 14, 2019 - Oct 19, 2019	Live Event
SANS Seattle Fall 2019	Seattle, WAUS	Oct 14, 2019 - Oct 19, 2019	Live Event
SANS New York City 2019	OnlineNYUS	Aug 25, 2019 - Aug 30, 2019	Live Event
SANS OnDemand	Books & MP3s OnlyUS	Anytime	Self Paced