CATSNDOGS

.LOL

DevSecOps CTF Runbook

Introduction

Welcome to the security team at CATSNDOGS.LOL. We have launched a massively successful platform for appreciating Cat and Dog memes. The thirst for more features and functionality has caused an explosion in our DevOps teams, and our "agileness" is more agile than ever. We are now forming a well-resourced DevSecOps team to prevent the otherwise inevitable disclosure of sensitive Dog or Cat related material. Our mission is to not slow down development of Cat and Dog related features, while also enforcing a set of security policies.

Now, as we literally just hired you all off of the conference floor at Re:Invent 2017, we need to quickly assess our readiness, and so we are starting off with a simple CTF wargame. Our AWS representatives have volunteered to play the role of the Red Team, crafting CloudFormation templates that breach our security policies in many different ways.

As one of our many loosely coupled Blue Teams, your team is tasked with running a DevSecOps CloudFormation static analysis service, to ensure we accurately block templates that breach our security policy, but also don't impede the good templates that form our organizations competitive advantage.

Unfortunately, we haven't actually written our controls yet, so that is a key part of your short-term backlog to sort out. Fortunately, we have architected a scalable API service for getting CloudFormation templates to AWS Lambda, so that you can apply our security policies quickly. However, things are a bit tight at CATNDOGS right now, so we will get you to run up this in your account.

You may wonder how we form teams (or Squads) in our DevSecOps teams. We believe teams create the best outcome, but individuals sometimes need their own environments. So, your team will actually run an environment per team member, but your performance will be measured on the mean performance of the entire team. As we understand sometimes one environment has a bad day, a team's lowest scoring member score will not be included in the mean.

There is too much to do today, and not enough time to do it, so divide and conquer to be successful.

Good luck.

Capture the Flag scoring

In this CTF Wargame, scoring will be as below

- -1 Points for any general failure to process a template (timeout / non-200)
- -5 Points for any "bad" template which is "passes"
- -10 Points for any "good" template which is "failed"
- +2 Point for any "good" template which is "passed"
- +2 Point for any "bad" template which is "failed", but without accurate violation counts
- +10 Points for any "bad" template which is "failed", and has accurate violation counts

As well as winning critical points, you will be able to measure how often you correctly passed or failed templates, as a percentage. You want this to be as close to 100% as possible.

During the final phase of the challenge, we will the run a final bonus competition. Each team may opt to upload their own CloudFormation template, and fill in accurate violation counts. You will then see a percentage of accounts that failed to correctly block your template. Higher is better. This challenge will require that the attack templates creator accurately define the expected analysis result.

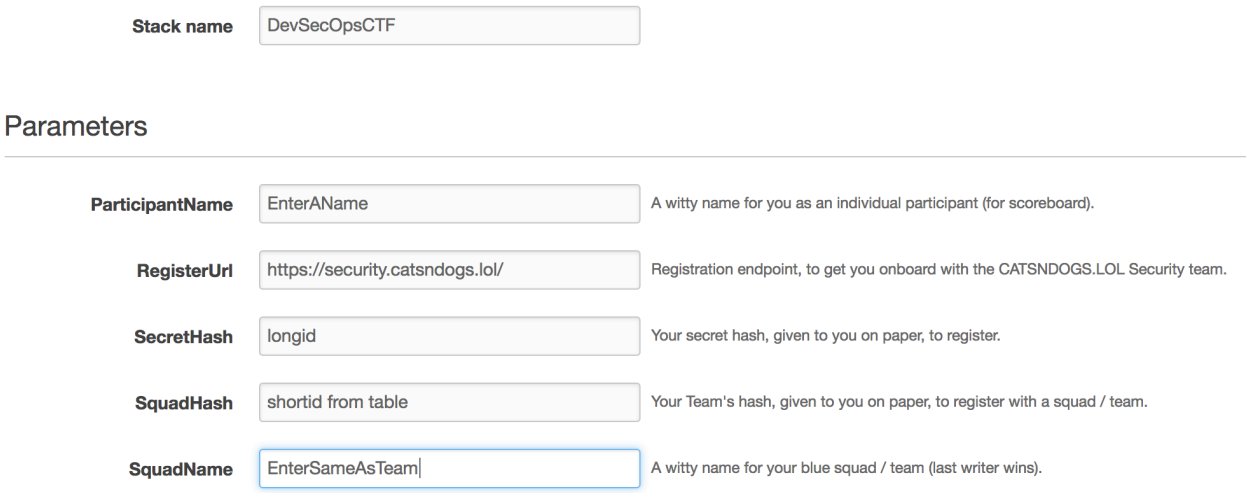
Deploy the infrastructure

Prerequisite's:

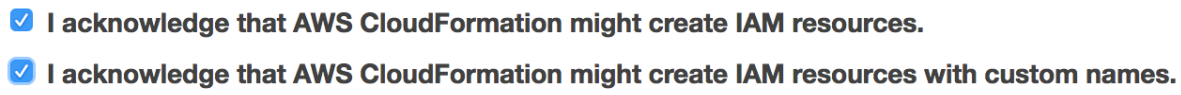
- A Secret Code for you as an individual security practitioner
- A Secret Code for your Squad / Team
- An AWS Account (with IAM rights)
- An AWS Credit Code Voucher will be given to you as you leave the workshop to cover usage

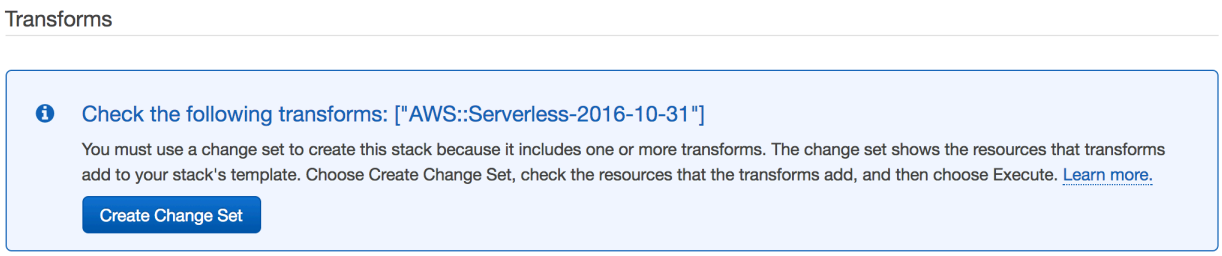
Deploying the infrastructure utilizes an AWS CloudFormation template. You can get the template, and other resources, from the resources link on the scoreboard at:

1. Login to the AWS Console at https://console.aws.amazon.com, and change region to Sydney (top right)
2. Select CloudFormation from the services menu, and select the option to "Create Stack"
3. Upload the devsecops-ctf-template.yml template, and select next.
4. Fill in the required parameters.



5. Leave next page as defaults
6. Select checkboxes, create a Change Set and then execute.





Working as a team

It will be VERY important to work as a team. There are too many rules in our policy to write all of the security controls yourself. So, it's important to have a mechanism to communicate, share snippets, and co-ordinate. This workshop integrates with Slack (the starter code sends debug messages to slack and cloud watch logs).

Its suggested ONE member of the team, creates a slack workspace and slack incoming hook, for your team.

1. Create a slack workspace:

https://slack.com/create#email and follow the steps to create the workspace, and invite team members.

2. Add incoming-webhooks integration:
   Click on "Apps" in slack workspace
   Search for "Incoming Webhooks"

3. Share webhook URL with team, and each team member create a channel for their lambda to log to.

Starter Code

After launching the Cloudformation, you will have 2 lambdas created in your account, which form your defense against our developers. stackname-DevSecOpsStarter-randomid, is your production lambda for static analysis. It receives CloudFormation templates each minute, and returns a pass/fail, and counts of the policy violations.

stackname-TestDevSecOpsStarter-randomid, is an identical function, for if you want to test code changes without breaking your production lambda function.

Configure the starter code by updating the values for SLACK_CHANNEL and HOOK_URL

Testing your code

The AWS Lambda console allows you to create a "test event", which will run your function for testing purposes. After configuring a test event, just use the test button in the lambda console. You can change the test events CloudFormation template, by converting a template to base64 encoding with something like https://www.base64encode.net/, and putting your new base64 into the `b64template` json value, in the test event (see the test event in the next section, Creating the test event).

If you have base64 and curl available on your system, you can also test via the DevSecOps API that launches your DevSecOpsStarter Lambda as part of the CTF scoring process (find the endpoint value in your CloudFormation outputs section):

```
curl --header "Content-Type: application/json" --request POST --data
'{"b64template":"'"$(base64 some_file.yml)"'"}' https://endpoint/api/
```

Or for your test lambda function, add /test to the endpoint:

```
curl --header "Content-Type: application/json" --request POST --data
'{"b64template":"'"$(base64 some_file.yml)"'"}' https://endpoint/api/test
```

Further to testing your function these two ways, the scoreboard has a "DevSecOps API test" function, that is intuitive to use.

Creating the test event

1. Go into the Lambda functions configuration view
2. Select "Select a test event"
3. Select "Configure test events"
4. Leave all details default, and enter a test name
5. Paste the below default test event. This is a base64 encoded YAML CloudFormation template.

{

"b64template":"DQpBV1NUZW1wbGF0ZUZvcm1hdFZlcnNpb246ICIyMDEwLTA5LTA5Ig0KRGVzY3JpcHRpb2
46IFByaXZhdGUgQXBwIFNlcnZlcg0KUmVzb3VyY2VzOg0KICAjTXkgUHJpdmF0ZSBJbnRlcm5hbCBBcHBsaWN
hdGlvbg0KICBNeUluc3RhbmNlOg0KICAgIFR5cGU6ICJBV1M6OkVDMjo6SW5zdGFuY2UiDQogICAgUHJvcGVy
dGllczoNCiAgICAgIEltYWdlSWQ6ICJhbWktZjE3M2NjOTEiDQogICAgICBJbnN0YW5jZVR5cGU6IHQyLm1pY
3JvDQogICAgICBTZWN1cml0eUdyb3VwSWRzOg0KICAgICAgICAtIFJlZjogIk15U2VjdXJpdHlHcm91cCINCi
AgI0kgdGhpbmsgdGhpcyBsZXRzIG1lIGFjY2VzcyBpdA0KICBNeUVJUDoNCiAgICBUeXBlOiAiQVdTOjpFQzI
6OkVJUCINCiAgICBQcm9wZXJ0aWVzOg0KICAgICAgSW5zdGFuY2VJZDogIVJlZiBNeUluc3RhbmNlDQogICNX
YXMgaGFyZCBnZXR0aW5nIHRoaXMgdG8gd29yaywgc28gY2hhbmdlZCBldmVyeXRoaW5nIHRvIDAuMC4wLjAvM
A0KICBNeVNlY3VyaXR5R3JvdXA6DQogICAgVHlwZTogQVdTOjpFQzI6OlNlY3VyaXR5R3JvdXANCiAgICBQcm
9wZXJ0aWVzOg0KICAgICAgR3JvdXBEZXNjcmlwdGlvbjogQWxsb3cgaHR0cCwgdGVsbmV0IGFuZCBTU0gNCiA
gICAgIFZwY0lkOiAidnBjLWVlZDlhNTg5Ig0KICAgICAgU2VjdXJpdHlHcm91cEluZ3Jlc3M6DQogICAgICAt
IElwUHJvdG9jb2w6IHRjcA0KICAgICAgICBGcm9tUG9ydDogJzgwJw0KICAgICAgICBUb1BvcnQ6ICc4MCcNC
iAgICAgICAgQ2lkcklwOiAwLjAuMC4wLzANCiAgICAgIC0gSXBQcm90b2NvbDogdGNwDQogICAgICAgIEZyb2
1Qb3J0OiAnMjMnDQogICAgICAgIFRvUG9ydDogJzIzJw0KICAgICAgICBDaWRySXA6IDAuMC4wLjAvMA0KICA
gICAgLSBJcFByb3RvY29sOiB0Y3ANCiAgICAgICAgRnJvbVBvcnQ6ICcyMScNCiAgICAgICAgVG9Qb3J0OiAn
MjEnDQogICAgICAgIENpZHJJcDogMC4wLjAuMC8wDQogICAgICAtIElwUHJvdG9jb2w6IHRjcA0KICAgICAgI
CBGcm9tUG9ydDogJzIyJw0KICAgICAgICBUb1BvcnQ6ICcyMicNCiAgICAgICAgQ2lkcklwOiAxNzIuMzEuMC
4wLzE2DQogICNOZWVkIGEgcm9sZSBmb3IgaW5zdGFuY2UgdG8gYWNjZXNzIER5bmFtb0RCDQogIFJvb3RSb2x
lOg0KICAgIFR5cGU6ICJBV1M6OklBTTo6Um9sZSINCiAgICBQcm9wZXJ0aWVzOg0KICAgICAgQXNzdW1lUm9s
ZVBvbGljeURvY3VtZW50Og0KICAgICAgICBWZXJzaW9uOiAiMjAxMi0xMC0xNyINCiAgICAgICAgU3RhdGVtZ
W50Og0KICAgICAgICAgIC0NCiAgICAgICAgICAgIEVmZmVjdDogIkFsbG93Ig0KICAgICAgICAgICAgUHJpbm
NpcGFsOg0KICAgICAgICAgICAgICBTZXJ2aWNlOg0KICAgICAgICAgICAgICAgIC0gImVjMi5hbWF6b25hd3M
uY29tIg0KICAgICAgICAgICAgQWN0aW9uOg0KICAgICAgICAgICAgICAtICJzdHM6QXNzdW1lUm9sZSINCiAg
ICAgIFBhdGg6ICIvIg0KICAgICAgTWFuYWdlZFBvbGljeUFybnM6DQogICAgICAgIC0gImFybjphd3M6aWFtO
jphd3M6cG9saWN5L0FtYXpvbkR5bmFtb0RCRnVsbEFjY2VzcyINCiAgICAgIFBvbGljaWVzOg0KICAgICAgIC
AtDQogICAgICAgICAgUG9saWN5TmFtZTogInJvb3QiDQogICAgICAgICAgUG9saWN5RG9jdW1lbnQ6DQogICA
gICAgICAgICBWZXJzaW9uOiAiMjAxMi0xMC0xNyINCiAgICAgICAgICAgIFN0YXRlbWVudDoNCiAgICAgICAg
ICAgICAgLQ0KICAgICAgICAgICAgICAgIEVmZmVjdDogIkFsbG93Ig0KICAgICAgICAgICAgICAgIEFjdGlvb
jogIioiDQogICAgICAgICAgICAgICAgUmVzb3VyY2U6ICIqIg0KDQogIFJvb3RJbnN0YW5jZVByb2ZpbGU6DQ
ogICAgVHlwZTogIkFXUzo6SUFNOjpJbnN0YW5jZVByb2ZpbGUiDQogICAgUHJvcGVydGllczoNCiAgICAgIFB
hdGg6ICIvIg0KICAgICAgUm9sZXM6DQogICAgICAgIC0NCiAgICAgICAgICBSZWY6ICJSb290Um9sZSINCg==
"}

Changing your code

The primary challenge of this CTF, is to transform security policy, into controls. The default starter code is Python 3.6, so you will be using this to create your controls. If you want to use a different lambda runtime (node, java, .net etc), create your new lambda, and add its name or ARN to the LAMBDA environment variable of the APIHandler lambda.

If you continue with Python 3.6, the basic formula for a control would typically be similar to the control the DevSecOpsStarter comes with:

```
if cfn['Resources'][resource]["Type"] == """AWS::EC2::SecurityGroup""":
    if "SecurityGroupIngress" in cfn['Resources'][resource]["Properties"]:
        for rule in cfn['Resources'][resource]["Properties"]['SecurityGroupIngress']:
            if "CidrIp" in rule:
                if (rule["ToPort"] not in SECURE_PORTS or rule["FromPort"] not in SECURE_PORTS) and
rule["CidrIp"] == '0.0.0.0/0':
                    result['pass'] = False
                    result['policy0'] += 1 #Add one to our policy fail counter
                    result["errors"].append("policy0: Port {} not allowed for /0".format(rule["FromPort"]))
```

The same code in pseudo code, for explanation:

```
Is the resource type a Security group?
And does it have an entry for 'SecurityGroupIngress' in its properties.
Lets loop over all the rules in 'SecurityGroupIngress', and from now on call each one ‘rule’
Does ‘rule’ contain a key of ‘CidrIp’? As I don’t want to crash if this rule is based on names.
And, if its port is not in our secure list, and ‘CidrIp’ is 0.0.0.0/0, we need to fail this template.
This template is not a pass
Increment the violation count of the policy we matched
Add error messages for debugging purposes.
```

This same pattern can be used for almost all of our policies, but the complexity varies depending on the CloudFormation structure. Resources that may help with the coding aspect can be found in the “Useful Resources” section.

Deploying new code can be done by:

a. Editing in the lambda console, and then saving in the console (easiest option).
b. Editing using the code in the resources zip, and the “build.sh” script to create a lambda package to upload.
c. Editing using the code in the resources zip, and manually making a package - http://docs.aws.amazon.com/lambda/latest/dg/lambda-python-how-to-create-deployment-package.html

Our DevSecOps Security Policy

Below are the policies we require to be explicitly controlled on AWS CloudFormation templates.

Your goal is to convert these policies into working controls that will correctly identify violations in a scalable API.

Policy 0. Network Access

Subrule1: Any Amazon EC2 cannot be publically accessible via dangerous ports (other than port 443 and 22)

Subrule2: Any Amazon RDS for MySQL can be accessed only by web servers via port 3306

Subrule3: Any Amazon S3 bucket cannot be publically accessible

Policy 1. Identity and Access Control

Subrule1: IAM Policy Elements: Action *, IAM or Organizations cannot be specified in "IAM inline policies" to create IAM users.

Subrule2: Only support or cloudwatch related "IAM managed policies" can be specified to create IAM users.

Subrule3: Any EC2 must be created with IAM role to access other services

Policy 2. Monitoring and Logging

Subrule1: Any ELB or CloudFront distribution, have to be created with logging enabled.

Subrule2: Any EC2 has to have a tags Name, Role, Owner, CostCenter

Policy 3. Data Encryption/Data protection

Subrule1: Any EBS volume for EC2 (except root volume) and RDS needs to be encrypted

Subrule2: Any traffic to Cloudfront needs to be encrypted by using https protocol

Useful Resources

The starter lambda uses Python, and yaml.ruamel to process the CloudFormation templates. Yaml.ruamel was used over the base yaml package as it is easily extendable to cope with Cloudformation functions in yaml (ie !Ref, !Sub).

http://yaml.readthedocs.io/en/latest/overview.html

As far as Python is concerned, if you are unfamiliar you can either replace the lambda that processes the template with another runtime, or get by modifying what you were given. The below tutorial documents focus on the most important language features for our needs.

https://docs.python.org/3/tutorial/

https://docs.python.org/3/tutorial/introduction.html

https://docs.python.org/3/tutorial/controlflow.html

https://docs.python.org/3/tutorial/datastructures.html

Once we have loaded in the template from YAML – it becomes a python dictionary, which we want to look through, and have conditions we look for – so a dictionary tutorial is helpful.

https://www.python-course.eu/dictionaries.php

And - to detect interesting elements in a Cloudformation YAML template, we need to refer to the template reference documentation.

http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-template-resource-type-ref.html

Cheat sheet for working with Lambda: https://github.com/srcecde/aws-lambda-cheatsheet

The end of this document contains an appendix on our DevSecOps policy in more detail, including hints and documentation for the CloudFormation structure.

Appendix 1: DevSecOps Policy

The Security policy

Policy 0. Network Access
Policy 1. Identity and Access Control
Policy 2. Monitoring and Logging
Policy 3. Data Encryption

Policy 0. Network Access

Subrule1:
Any Amazon EC2 cannot be publically accessible via dangerous ports (other than port 443 and 22)
Hint: Validate a security group in templates. "CidrIp: 0.0.0.0/0" cannot be configured on any port except port 443 or 22 in security group, which implies any port range is also not allowed. Note that a security group attached to an instance is also newly created in the same templates, so you can validate those security groups. There might be many security groups and many ingress rules per security group in templates. Each ingress rule needs to be validated. And assume protocol is always "tcp", so it's not necessary to validate which protocol an ingress rule in a security group is for.
Reference:
AWS::EC2::SecurityGroupIngress Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group-ingress.html#aws-resource-ec2-securitygroupingress-syntax
Example (See the YAML format.)
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group-rule.html#w2ab2c21c14d602c11b2

Subrule2:
Only WebServers (the security group named WebServerSecurityGroup) are allowed to access RDS for MySQL instances via port 3306.
Hint: If security groups that allows access to TCP port 3306 are found in templates, they need to be validated whether if WebServerSecurityGroup is specified as source by using "SourceSecurityGroupName. "WebServerSecurityGroup" is the name of the security group attached to web servers. It is not necessary to validate whether if the security groups allowing port 3306 are attached to RDS instances. Note that the security groups allowing 3306 are VPC security group, not DB security group that is used for non VPC environment.
Reference:
AWS::EC2::SecurityGroupIngress Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group-ingress.html#aws-resource-ec2-securitygroupingress-syntax
Example (See the YAML format.)
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-security-group-ingress.html#w2ab2c21c10d428c13b2

Subrule3:
Any Amazon S3 bucket cannot be publically accessible
Hint: Validate S3 canned access control lists (ACL) that are configured for newly created S3 buckets in templates. S3 canned ACL S3 grants predefined permissions to the bucket. ACL cannot be configured as public read and public read write.
Reference:
Access Control List (ACL) Overview - Canned ACL
http://docs.aws.amazon.com/AmazonS3/latest/dev/acl-overview.html#canned-acl
Example (See the YAML format.)
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-s3-bucket.html#w2ab2c21c10d934c17b4

Policy 1. Identity and Access Control

Subrule1:
IAM Policy Elements: Action *, IAM or Organizations cannot be specified in "IAM inline policies" to create IAM users.
Hint: Validate Action elements in IAM inline policies. The list of actions (namespaces) for *, IAM, or Organizations cannot be specified as a value of Action under Statement in IAM inline policies. "NotAction" elements are not in any templates but "Effect" under Statement needs to be validated. Also there may be multiple individual statements in an inline policy so you need to validate each "Effect" and "Action" per individual statement by iterating. To make it simple, only one action (namespace) is specified per "Action", which means an iteration per action is not necessary and the value type is String instead of Array.
Reference:
AWS::IAM::User Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-iam-user.html#aws-resource-iam-user-syntax.yaml
IAM Policy Elements: Action
http://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_elements_action.html
Example of IAM inline policy
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/quickref-iam.html#quickref-iam-example-1.yaml

Subrule2:
Only support or cloudwatch related "IAM managed policies" can be specified to create IAM users.
Hint: Validate Amazon Resource Name (ARN) for managed policies attached to IAM users in templates. IAM managed policies for Support (AWSSupportAccess and SupportUser) and CloudWatch (*CloudWatch*) are only allowed. There might be multiple managed policies specified per IAM user in templates. To find the ARNs for managed policies whose type are either AWS managed or Job function (Job function is part of managed policy), go to the IAM management console. (Go to the IAM console and choose Policies in the navigation pane. Filter menu is also available. Use the search box to filter the list of policies).
Reference:
AWS::IAM::User Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-iam-user.html#aws-resource-iam-user-syntax.yaml
AWS Managed Policies
http://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html#aws-managed-policies

Subrule3:
Any EC2 must be created with IAM role to access other services
Hint: Validate whether if an IAM role is attached to an EC2 instance in templates. It is not necessary to validate the contents of an IAM role itself.
Reference:
AWS::EC2::Instance Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html#aws-properties-ec2-instance-syntax.yaml

Policy 2. Monitoring and Logging

Subrule1:
Any ELB and CloudFront have to be created with logging enabled.
Hint: Validate whether logging is enabled for ELB and CloudFront in templates. Note that logging properties may not exist in a template that creates an ELB or CloudFront because they are not required properties. It is necessary to catch that case to prevent them from passing as well.
Reference:
AWS::ElasticLoadBalancing::LoadBalancer Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-elb.html#aws-resource-elasticloadbalancing-loadbalancer-syntax.yaml

AWS::CloudFront::Distribution Syntax http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-cloudfront-distribution.html#aws-properties-cloudfront-distribution

Subrule2:
Any EC2 has to have tags for Name, Role, Owner & CostCenter
Hint: Validate whether these tags (Name, Role, Owner, CostCenter) are attached to EC2 instances to be created in templates**.** There may be multiple EC2 instances and multiple keys per instance in one template. It is not necessary to validate values of tags but keys. Note tags itself may not be set for an EC2 instance in templates since it is not a required property. It is necessary to catch that case to prevent them from passing as well.
Reference:
AWS::EC2::Instance Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html#aws-properties-ec2-instance-syntax.yaml

Policy 3. Data Encryption

Subrule1: Any EBS volume for EC2 (except root volume) and RDS needs to be encrypted
Hint: For EBS volumes attached to EC2 instances, validate EBS volumes under the type; AWS::EC2::Instance, not AWS::EC2::Volume. There might be multiple EBS volumes attached to one EC2 instance.
Resources:
AWS::EC2::Instance Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html#aws-resource-ec2-instance-syntax
Example - EC2 Instance with an EBS Block Device Mapping
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-ec2-instance.html#w2ab2c21c10d367c15b2
AWS::RDS::DBInstance - Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-rds-database-instance.html#aws-resource-rds-dbinstance-syntax

Subrule2: Any traffic to Clondfront needs to be encrypted by using https protocol
Hint: https-only is the only valid option. This property is under DefaultCacheBehavior.
Resources:
CloudFront DefaultCacheBehavior - Syntax
http://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-properties-cloudfront-defaultcachebehavior.html#w2ab2c21c14d181b5