

Loose Keys Bring These: Attackers + Me's (Incident Responders)

Jonathon Poling

Managing Principal Consultant

Secureworks

~~About Why Me?~~

- A little bit about me... ← blah blah blah
- Why Me? ← this though
 - AWS SME for Secureworks
 - Developed Secureworks' AWS Incident Response Service Line
 - Help SMB through Fortune 10 Customers...
 - Intelligently Configure/Instrument Their Environments
 - Protect Their Infrastructure
 - Effectively Respond to Incidents

Why Did I Put This Together?

After years of doing Incident Response engagements across a variety of clients and verticals, I came to a few realizations:

- 1) There are a LOT of Access Key compromises
- 2) We need to be doing more to both protect against and respond to Cloud incidents (including #1)
- 3) Dev(Sec)Ops is a vastly underutilized and very effective resource (i.e. force multiplier) for doing #2

So, How Will This Help You?

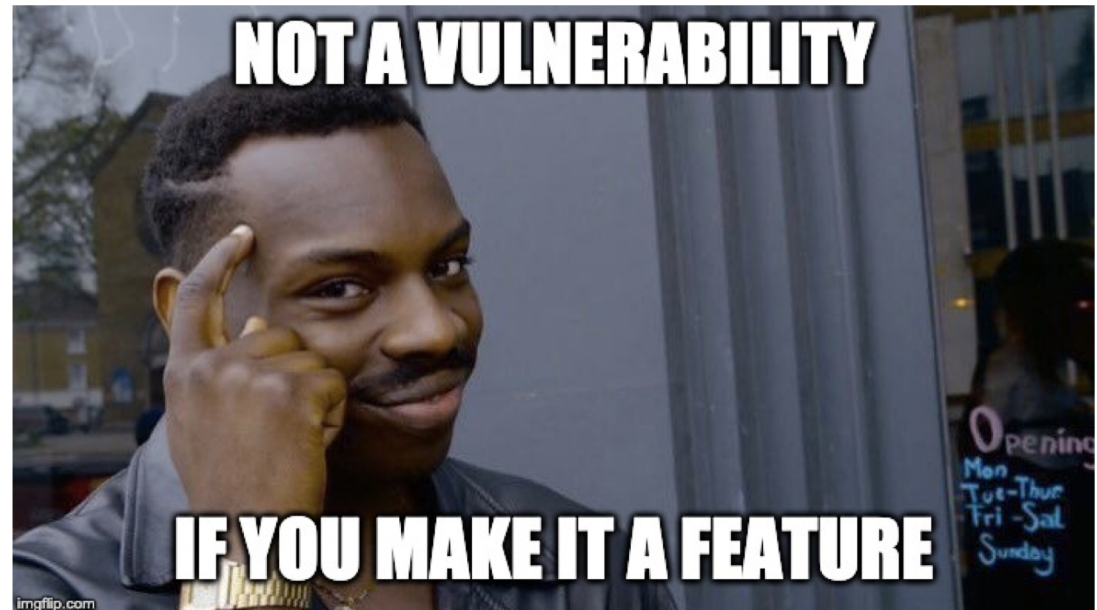
In this talk, you will (hopefully) learn:

- How/why Access Key (AK) compromises occur
- The chronology of an AK compromise
- How to respond to an AK compromise
- Tools/tips/tricks for effective response
- How Dev(Sec)Ops plays a key role in both protecting and responding to threats in the Cloud

Why / How Do Access Key Compromises Occur?

Most Common Reasons These Occur

- Accidental Commit
- Insecure (Local) Storage
- Insecure Transmission
- Command Line History
- AWS Metadata Service*



Accidental Commit

```
$ git commit -am "Reminder: Find New Job."
```

Security

Dev put AWS keys on Github. Then BAD THINGS happened

Fertile fields for Bitcoin yields - with a nasty financial
sting

Spending 100K USD in 4,5 days on Amazon Web Services

*Submitted by **walterheck** on **April 25, 2017***

This post is probably one of the more embarrassing post-mortems I have ever written, but I feel it's important so that we can warn others and hopefully prevent a lot of pain in the process.

Insecure (Local) Storage

14 OCT 2019 NEWS

Stolen Cloud API Key to Blame for Imperva Breach

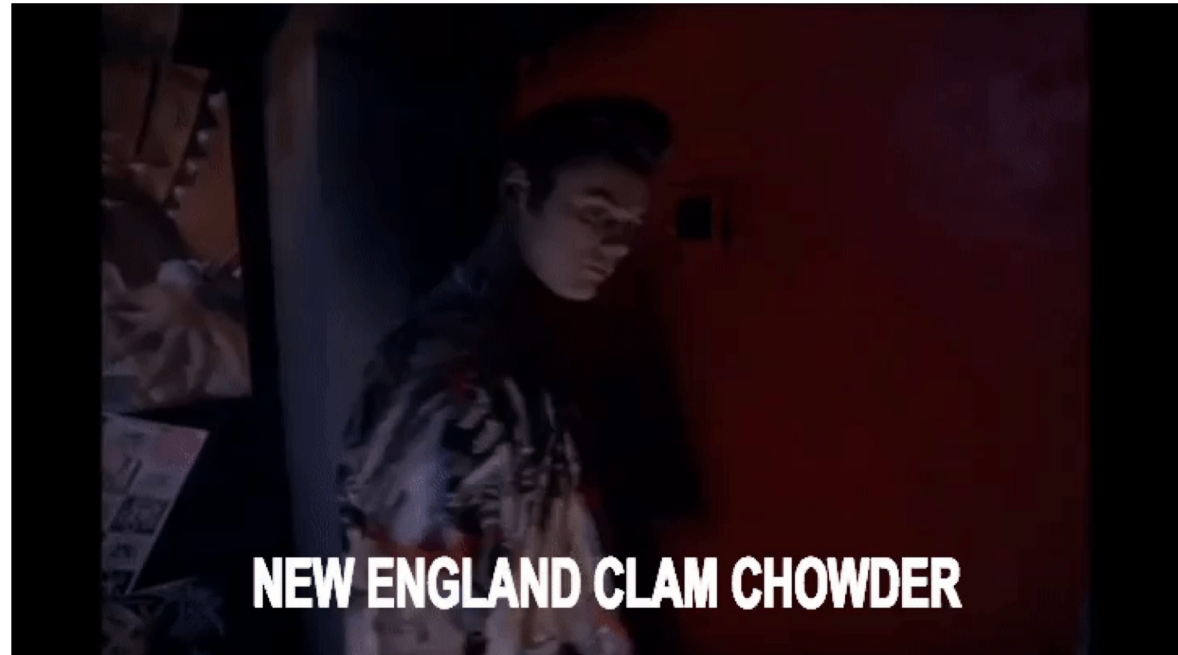
Some key decisions made during the AWS evaluation process, taken together, allowed information to be exfiltrated from a database snapshot. These were: (1) we created a database snapshot for testing; (2) an internal compute instance that we created was accessible from the outside world and it contained an AWS API key; (3) this compute instance was compromised and the AWS API key was stolen; and (4) the AWS API key was used to access the snapshot.

Separately, Imperva's IT team created an internal compute instance containing an AWS administrative API key. Unfortunately, this server was left exposed and subsequently found by a hacker, who stole the all-important key and used it to access the database snapshot, exfiltrating the information in October 2018.

Insecure Transmission

How are you transmitting Access Keys to your users?

- Email
- Chat
- Internal Data Store
- Something else...?



Command Line History

What happens when an attacker compromises an AWS Instance?

- Quick “whoami” + check/collect env var's
 - `$ env | grep "AWS_"`
- Check command line history to:
 - See what this system may be used for
 - Obtain credentials/secrets input via the CL
- Identify currently assigned role/privileges
- Attempt resource enumeration/exploitation
- ...and so on

AWS Metadata Service

Capital One hack highlights SSRF concerns for AWS

Infosec pros warn of server-side request forgery vulnerabilities in AWS following the Capital One data breach, which may have revealed an issue regarding the AWS metadata service.

Pre/Co-Requisites for Success (acquiring Access Key):

- Access to Instance/Resource to make Metadata call(s)
- Knowledge of specific assigned Instance Profile / Role Name
- (Over)Privilege of assigned Role to perform bad things

All that said... it is clearly a successfully utilized attack vector.

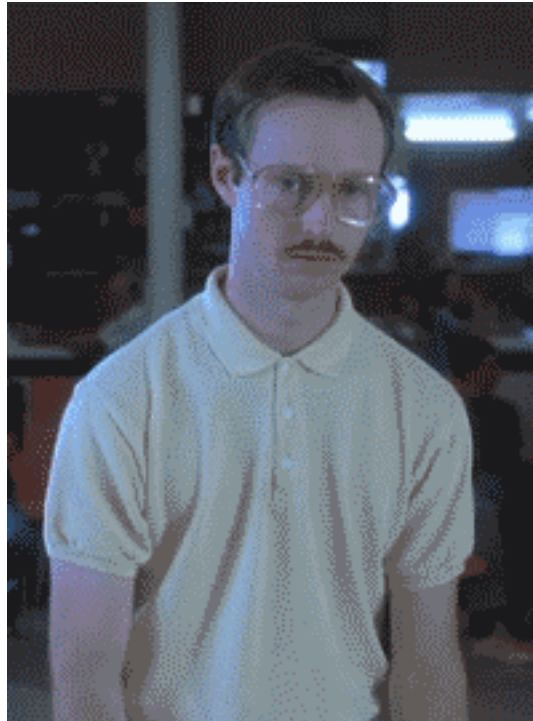
So How Does This Go?

Attackers Be Attackin'

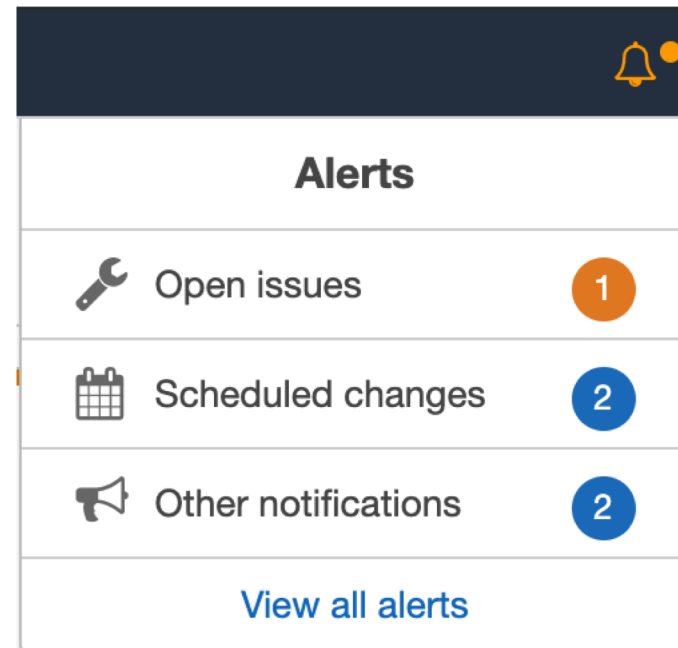
- People are searching and watching GitHub...
 - <https://github.com/dxa4481/truffleHog>
 - “Searches through git repositories for high entropy strings and secrets, digging deep into commit history”
 - <https://github.com/eth0izzle/shhgit/>
 - “Shhgit finds secrets and sensitive files across GitHub code and Gists committed in *near* real time by listening to the [GitHub Events API](#).”
 - <https://github.com/duo-labs/secret-bridge>
 - Duo’s recent release, similar to ShhGit (uses GitHub Events API hooks)
- Attackers are looking in your Command Line History, ~/.aws/, and Documents folders for plaintext credentials

Insecure Credential Storage/Usage

Once credentials are acquired...
it's off to the races.



Initial Alert



OK, this could be anything...

Initial Alert

Dashboard

[Set up notifications with CloudWatch Events](#)



1 Open issues
Past 7 days

2 Scheduled changes

2 Other notifications
Past 7 days

Issues that might affect your AWS infrastructure. [0 issues](#) were resolved in the past 24 hours.

[See all issues](#)



<input type="text"/>					
	Event	Region/AZ	Start time	Last update time	Affected resources
<input type="radio"/>	Risk credentials exposure suspected	us-east-1	2019 at 4:28:28 PM UTC	2019 at 4:28:29 PM UTC	1 key

W'uh oh...

What Does AWS Suggest?

Risk credentials exposure suspected

Close



Details

Affected resources

Your AWS Account may be compromised! Please review the following notice and take immediate action to secure your account. We have also opened an outbound Support Case with additional details. Please check <https://aws.amazon.com/support> and take action immediately.

Your security is important to us. We have detected abnormal activity in your AWS account that matches the pattern of unauthorized access, which may indicate that an access key and the corresponding secret key are compromised.

Please log into your account and check that all resource are legitimate (please check all regions - to switch between regions use the drop-down in the top-right corner of the management console screen) and delete all unauthorized resources. Please pay special attention to IAM users, access keys, EC2 instances, and EC2 Spot Bids. We suggest that you rotate the access key specified above. Unfortunately, deleting the keys from the public website and/or disabling them is NOT sufficient to secure your account.

If the unauthorized usage is not stopped before the deadline specified in the Support Case, in accordance with the AWS Customer Agreement, we may suspend your AWS account. Failure to perform the below security steps within 24 hours may result in the termination of all EC2 Spot instances in addition to any other suspected unauthorized usage on your account. If you believe you've received this note in error, please contact us immediately via the support case.

It is recommended to regularly rotate all credentials including the access keys and account passwords. Also please make sure that you never share an AWS Access Key together with your AWS Secret Key on public web sites.

Detailed instructions are included below for your convenience.

CHECK FOR UNAUTHORIZED USAGE

We strongly encourage you to immediately review your AWS account for any unauthorized AWS usage, suspect running instances, or inappropriate IAM users and policies. To check the usage, please log into your AWS Management Console and go to each service page to see what resources are being used. Please pay special attention to the running EC2 instances and IAM users, roles, and groups. You can also check for any unexpected usage on the "Bills" page in the Billing console.

<https://console.aws.amazon.com/billing/home#/bill>

Please keep in mind that unauthorized usage can occur in any region and that in your console you only see one region at a time. To switch between regions, you can use the dropdown in the top-right corner of the console screen.

What Does AWS Suggest?

→ DELETE THE KEY (ROOT ACCOUNT)

If you are not using the access key, you can simply delete it. To delete the exposed key, visit the "Security Credentials" page here: https://console.aws.amazon.com/iam/home#security_credential. Your keys will be listed in the "Access Keys" section.

→ DELETE THE KEY (IAM USERS)

Navigate to your IAM Users list in the AWS Management Console, here: <https://console.aws.amazon.com/iam/home#users>. Please select the IAM user identified above. Click on the "User Actions" drop-down menu and then click "Manage Access Keys" to show that user's active Access Keys. Click "Delete" next to the access key identified above.

→ ROTATE THE KEY

If your application uses the access key, you need to replace the exposed key with a new one. To do this, first create a second key (at that point both keys will be active) and modify your application to use the new key.

Then disable (but do not delete) the first key. If there are any problems with your application, you can make the first key active again. When your application is fully functional with the first key inactive, please delete the first key.


→ ENABLE AMAZON GUARDDUTY

Amazon GuardDuty is an AWS threat detection service that helps you continuously monitor and protect your AWS accounts and workloads. Enabling Amazon GuardDuty on your accounts gives you further visibility into malicious or unauthorized activity, alerting you to take action in order to reduce the risk of harm. To learn more, visit: <https://aws.amazon.com/guardduty>





Please follow the Best Practices of Managing your Access Keys at <http://docs.aws.amazon.com/general/latest/gr/aws-access-keys-best-practices.html>.

If you have any additional questions or concerns regarding this notification, please reply to the Support Case.


Event data


Event	Risk credentials exposure suspected
Status	Open
Region/AZ	us-east-1
Start time	May 10, 2019 at 4:28:28 PM UTC-7
End time	May 24, 2019 at 4:28:28 PM UTC-7
Event category	 Issue


Which Key is Compromised?

Risk credentials exposure suspected Close    

Details **Affected resources**

 Add filter

Resource ID / ARN
AKIA 



Cool, AWS tells us exactly which key is compromised!

What We Know...

- Credentials Were Leaked (...somewhere)
- Date/Time AWS Became Aware (Alerted Us)
- Access Key ID (AKIA...)

That should be enough to get started investigating...

How Do We Respond
to This?

AWS Credential Prefixes / Formats

In general (the subset we care about for this presentation):

Access Key ID

AKIA[IJ][2-7A-Z]{14}[AQ]

Secret Access Key

[0-9a-zA-Z/+]{40}

AssumeRole Session (STS) Tokens

ASIA[0-9A-Z]{16}

Prefix	Entity Type
AAGA	Action group
ACCA	Context Specific Credential
AGPA	Group
AIDA	IAM user
AIPA	Amazon EC2 instance profile
AKIA	Access key
ANPA	Managed policy
ANVA	Version in a managed policy
APKA	Public key
AROA	Role
ASCA	Certificate
ASIA	Temporary (AWS STS) keys

See links in slide notes to read about all the different types

Identify User Associated with Access Key

```
$ for user in $(aws iam list-users --  
-output text | awk '{print $NF}');  
do aws iam list-access-keys --user  
$user --output text; done | grep  
<AccessKey>
```

Disable / Delete Access Key

Disable Access Key

```
$ aws iam update-access-key --  
access-key-id AKIDPMS9R04H3FEXAMPLE  
--status Inactive [--user-name Bob]
```

Delete Access Key

```
$ aws iam delete-access-key --  
access-key-id AKIDPMS9R04H3FEXAMPLE  
[--user-name Bob]
```

Delete User (Optional)

```
$aws iam delete-user --user-name Bob
```

Disable Temporary Credential Usage

- Attach IAM Policy to User to...
- Deny Temporary Credentials issued before now

```
{  
  "Version": "2012-10-17",  
  "Statement": {  
    "Effect": "Deny",  
    "Action": "*",  
    "Resource": "*",  
    "Condition": {  
      "DateLessThan": {  
        "aws:TokenIssueTime": "2019-11-04T19:35:00Z"  
      }  
    }  
  }  
}
```

Disable Temporary Credential Usage

- Attach IAM Policy to User to... (Cont.)
 - Deny All Actions for 36 hours* (or indefinitely)

*Maximum duration of validity allowed

```
{  
  "Version": "2012-10-17",  
  "Statement": {  
    "Effect": "Deny",  
    "Action": "*",  
    "Resource": "*",  
  }  
}
```

Search Logs (CloudTrail Lookup-Events)

Caveats:

- Limited to events from last 90 days
- Queries must be performed for *each region*

Identify/collect all Actions associated with known compromised
Access Key

```
$ aws cloudtrail lookup-events --lookup-attributes  
AttributeKey=AccessKeyId,AttributeValue=AKIAIOSFODNN  
7EXAMPLE
```


Search Logs (CloudTrail Lookup-Events)

Identify any new User Accounts and/or Access Keys created by attacker for persistence

```
$ aws cloudtrail lookup-events --lookup-attributes  
AttributeKey=AccessKeyId,AttributeValue=AKIAIOSFOD  
NN7EXAMPLE --query  
'Events[?EventName==`CreateUser`]'
```

```
$ aws cloudtrail lookup-events --lookup-attributes  
AttributeKey=AccessKeyId,AttributeValue=AKIAIOSFOD  
NN7EXAMPLE --query  
'Events[?EventName==`CreateAccessKey`]'
```

Search Logs (Athena)

Caveats:

- Presumes you are consolidating CloudTrail logs to a single Bucket
- Requires proper partitioning for performance + \$\$
- Costs \$\$ based on size of data searched (\$5 / TB Scanned)

Identify/collect all actions associated with known compromised Access Key (known compromised after 10/1/2019)

```
> Select *  
From cloudtrail_logs  
Where userIdentity.accessKeyId = 'AKIAIOSFODNN7EXAMPLE'  
AND eventtime >= '2019-10-01T00:00:00Z'
```

Search Logs (Athena)

Identify any new User Accounts and/or Access Keys created by attacker for persistence

```
> Select eventTime, eventName, userIdentity.username,  
userIdentity.accessKeyId, sourceIpAddress  
From cloudtrail_logs  
Where userIdentity.accessKeyId = 'AKIAIOSFODNN7EXAMPLE'  
AND (eventName = 'CreateUser' Or eventName =  
'CreateAccessKey')
```

Search Logs (Athena)

You can also limit your search to a specific region, day/time, etc. to increase performance (and also reduce search costs)

```
> Select eventTime, eventName, userIdentity.username,  
userIdentity.accessKeyId, sourceIpAddress  
From cloudtrail_logs  
Where userIdentity.accessKeyId = 'AKIAIOSFODNN7EXAMPLE'  
AND (eventName = 'CreateUser' Or eventName =  
'CreateAccessKey')  
AND region='us-east-1'  
AND year='2019'  
AND month='10'  
AND day='31'
```

Search Logs (JQ)

JQ is a super powerful command-line tool to ETL all your JSON data to your heart's content

*Examples below are operating on data collected from “aws cloudtrail lookup-events” output.

Identify/collect all Actions associated with known compromised Access Key

```
$ jq -r '.Events[] | select(.AccessKeyId=="<AKIA...>")'  
CloudTrail_Lookup-Events_Output.json
```

Identify any new User Accounts and/or Access Keys created by attacker for persistence

```
$ jq -r '.Events[] | select(.AccessKeyId=="<AKIA...>")'  
| select(.EventName|test("Create(User|AccessKey)"))'  
CloudTrail_Lookup-Events_Output.json
```

Search Logs (...)

And, much much more. But, you get the gist...

How Do We Protect
Against This?

Proactive Protection

- Implement local Key/Secrets Management
 - Practices
 - Don't store credentials in plaintext anywhere!
 - Use/Assume Roles as often as possible
 - Implement and enforce (via IAM Policy):
 - MFA
 - Least Privilege Access
 - Implement filtering/blocking AWS Metadata Proxy
 - Tools (Examples)
 - Git-Secrets, HashiCorp Vault, AWS-Vault, Strongbox, etc.

Proactive Monitoring / Detection

- Proactive Monitoring / Detection
 - GitHub built-in detection/notification
 - AWS built-in detection/notification
 - AWS GuardDuty*
 - *Notifications delayed ~15 min
- Build detections based on experience responding to previous compromises
 - Billing Alerts
 - Anomaly Detections
 - Activity in unutilized Regions
 - ...

Reactive (Response)

- Respond to alerts / findings / issues
 - Services
 - Config Rules
 - CloudWatch Events Rules
 - Lambda
 - ... third-party?
 - Manual
 - Acquire Memory
 - Acquire System Snapshot

But...

Should We Do This All Manually?



Sounds like some DEVELOPMENT opportunities to me...

What is Dev(Sec)Ops'
Role in All This?

How I Build Stuff

This is my formula for building Digital Forensics and Incident Response (DFIR) capabilities:

1. Identify the problem
2. Identify possible solutions
3. **DEVELOP** technical solution(s)
4. Implement technical solution(s)
5. Augment existing solution(s) and/or **DEVELOP** new ones.

So...

Doesn't it make sense to harness the DEVELOPMENT / Security / Operations team(s) in doing all this?

So, why aren't we doing that (better)?

How about we start (getting better) today...

Dev(Sec)Ops Force Multiplication

Security starts in / with Development

So, what do we need to build/implement here?

- Implementing tool(s) for secret usage/management
 - Start with existing tools
 - Git-Secrets, HashiCorp Vault, AWS-Vault, Strongbox, etc.
 - Or, roll your own?
 - Build tools into pipeline for Development Instances / Env's
 - Instance User Data at Launch

Dev(Sec)Ops Force Multiplication

Security starts in / with Development

So, what do we need to build/implement here?

- Building IAM Policies
 - MFA enforcement
 - Require MFA condition(s) present to perform any actions
 - Monitor for accounts with no MFA and disable/remove
 - Least Privilege Access
 - Implement existing tooling to start/test least privilege accesses
 - Work across teams to build least privilege policies for:
 - DevOps Team
 - Security Team
 - Users
 - Instances / Services ...

Dev(Sec)Ops Force Multiplication

Security starts in / with Development

So, what do we need to build/implement here?

- Implementing AWS Metadata Proxy Filtering/Blocking
 - Start With Existing Tools
 - <https://github.com/Netflix-Skunkworks/aws-metadata-proxy>
 - <https://github.com/lyft/metadataproxy>
 - Augment to your needs
 - Build this into your deployment pipeline:
 - CloudFormation Templates
 - OpsWorks Stacks
 - Terraform
 - ...

Dev(Sec)Ops Force Multiplication

Security starts in / with Development

So, what do we need to build/implement here?

- Implementing Proactive Monitoring/Detection
 - CloudWatch Events Rules for CloudTrail
 - Build Alerts for:
 - Specific Access Key Usage
 - Root Access Key
 - Honeytokens
 - Specific API Actions
 - StopLogging, DeleteTrail, Create(User | AccessKey), DeactivateMFADevice ...

Dev(Sec)Ops Force Multiplication

Security starts in / with Development

So, what do we need to build/implement here?

- Implementing AWS GuardDuty
 - Start with Existing Rules
 - Build new Rules based on investigations
 - Build response automations based on real-world use cases
 - CloudWatch Events Rule -> Lambda to respond with <X> actions
- Building Response Automations
 - Access Key / Account Disabling (or Deletion) + MUCH else
 - Be ingenious through Dev(Sec)Ops and Security Team collaborations in coming up with your own!

TL;DR

TL;DR

- Security starts early in the Development process
- Dev(Sec)Ops plays a huge role in both proactive and reactive protection and response
- Collaboration on development, testing, review, and augmentation is key
- Automate, Automate, Automate

The better we build things together...

The sooner we detect the badness...

The faster and better we respond...

The stronger we (and the business) become.

The End

Email: jpoling@secureworks.com

Twitter: @JPoForenso

Blog: <https://www.ponderthebits.com>

