Who done it:
Gaining visibility and accountability in the cloud

By Ryan Nolette
Marta Gomez
$whoami

**Ryan Nolette**
15+ year veteran of IT, Security Operations, Threat Hunting, Incident Response, Threat Research, and Forensics
GitHub - [https://github.com/sonofagl1tch](https://github.com/sonofagl1tch)
Career highlight - Time's person of the year 2006

**Marta Gómez**
IT Security Developer at Wazuh where she is their leading python and AWS subject matter expert.
Software engineer and diversity-in-tech activist at Yes We Tech and many other local tech organizations.
GitHub - [https://github.com/mgmacias95](https://github.com/mgmacias95)
Agenda

● Overview
  ■ Who did what and when?
  ■ Common Techniques
  ■ Common Visibility Tools and Their AWS Equivalent
● Increasing visibility until you have accountability
  ■ Common Logs to fetch
  ■ Fetching logs from AWS
● End to end example
  ■ Detonation Lab
  ■ Logging Pipelines and Services
● Finding What Matters
● We are giving away a full detonation lab to learn with

https://github.com/sonofagl1tch/AWSDetonationLab
Who did what and when

- These are the 3 pillars of each stage of scoping the event
- Will be modified for each iteration
- The analysts should be able to start at any of the stages and complete the cycle
Common Techniques
What’s Their Goal?

- OS hardening
- Config management
- Identity Management
- Process monitoring

Visibility → Accountability

Who  What  When
### Common Visibility Tools and Their AWS Equivalent

<table>
<thead>
<tr>
<th>Traditional Tool</th>
<th>AWS equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Pane of Glass (SPG) for Findings</td>
<td>Security Hub</td>
</tr>
<tr>
<td>IDS/IPS</td>
<td>GuardDuty</td>
</tr>
<tr>
<td>DLP</td>
<td>Macie</td>
</tr>
<tr>
<td>Netflow</td>
<td>VPCFlow</td>
</tr>
<tr>
<td>DNS</td>
<td>Route53</td>
</tr>
<tr>
<td>Access and authentication auditing</td>
<td>CloudTrail</td>
</tr>
<tr>
<td>Active Directory</td>
<td>Directory Service</td>
</tr>
<tr>
<td>Identity Management</td>
<td>IAM</td>
</tr>
<tr>
<td>Single Sign On</td>
<td>AWS SSO</td>
</tr>
<tr>
<td>Vulnerability scanner</td>
<td>Inspector</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>AWS config</td>
</tr>
<tr>
<td>Logging Services Pipeline</td>
<td>Cloudwatch + Firehose + Lambda</td>
</tr>
</tbody>
</table>
End to end example

DevOps Installs new tool that is backdoored with a CryptoMiner

Static defense controls do not stop miner installation

SecOps remains vigilant

SecOps Defends

Always Squirrel away your logs!!!
Detonation Lab Topology
<table>
<thead>
<tr>
<th>Log Service</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CloudFormation</td>
<td>CloudFormation allows you to use a simple text file to model and provision, in an automated and secure manner, all the resources needed for your applications across all regions and accounts.</td>
</tr>
<tr>
<td>S3</td>
<td>Amazon S3 is object storage built to store and retrieve any amount of data from anywhere.</td>
</tr>
<tr>
<td>VPC flow</td>
<td>Amazon Virtual Private Cloud (Amazon VPC) lets you provision a logically isolated section of the AWS Cloud where you can launch AWS resources in a virtual network that you define.</td>
</tr>
<tr>
<td>IAM</td>
<td>AWS Identity and Access Management (IAM) enables you to manage access to AWS services and resources securely. Using IAM, you can create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources.</td>
</tr>
<tr>
<td>Cloudtrail</td>
<td>AWS CloudTrail is a service that enables governance, compliance, operational auditing, and risk auditing of your AWS account. With CloudTrail, you can log, continuously monitor, and retain account activity related to actions across your AWS infrastructure.</td>
</tr>
<tr>
<td>Macie</td>
<td>Amazon Macie is a security service that uses machine learning to automatically discover, classify, and protect sensitive data in AWS.</td>
</tr>
<tr>
<td>Inspector</td>
<td>Amazon Inspector is an automated security assessment service that helps improve the security and compliance of applications deployed on AWS.</td>
</tr>
<tr>
<td>Cloudwatch</td>
<td>Amazon CloudWatch is a monitoring and management service built for developers, system operators, site reliability engineers (SRE), and IT managers.</td>
</tr>
<tr>
<td>Guardduty</td>
<td>Amazon GuardDuty is a threat detection service that continuously monitors for malicious or unauthorized behavior to help protect your AWS accounts and workloads.</td>
</tr>
</tbody>
</table>
Wazuh is an open-source host-based intrusion detection system (HIDS).
How do we ingest AWS logs?

Tools:
- python
- boto3

Steps:
- Iterates over all files in the bucket.
- Parses files content.
- Send parsed alerts to Wazuh.
- Raise an alert and index it in Elasticsearch.
How to iterate over files

```python
#!/usr/bin/python3
import boto3
client = boto3.client('s3')
bucket_files = client.list_objects_v2(Bucket='a_bucket_name')
if 'Contents' not in bucket_files:
    raise Exception("No files available in bucket")
for bucket_file in bucket_files['Contents']:
    print(bucket_file['Key'])
```

**Existing challenges with proposed solution**
- Filter by region/account id?
- Iterates over all files in the bucket from the very beginning.
- Only returns 1000 first files, what about the others?
Filtering by account id and region

This is how an AWS log filename looks:

```
AWSLogs/123456789/CloudTrail/us-east-1
/2018/09/27/123456789_CloudTrail_us-east-1_20180927T1520Z_uF5y0kjRnBDK1oT.json.gz
```

It includes account id and region. That can be used as filter: filtering by a prefix including the desired region and account id.

```python
account_id = "123456789"
region = 'us-east-2'
AWSprefix = f'AWSLogs/{account_id}/CloudTrail/{region}/'
client = boto3.client('s3')

bucket_files = client.list_objects_v2(Bucket='a_bucket_name', Prefix=AWSprefix)
```

When are your logs from?
Select where to start processing logs

- **StartAfter** parameter in `list_objects_v2` function.
- This parameter doesn't accept a date, it must be a *filename*.
- But Amazon organizes these buckets by dates directories, using this we can get the first key from an specified date and use it as **StartAfter** parameter.

```python
account_id = "123456789"
region = 'us-east-1'
start_date = "2019/03/14"
AWSprefix = f'AWSLogs/{account_id}/CloudTrail/{region}/'
client = boto3.client('s3')

bucket_files = client.list_objects_v2(Bucket='a_bucket_name', Prefix=AWSprefix, StartAfter=AWSprefix + start_date)
```

AWSLogs/123456789/CloudTrail/us-east-1/2018/09/27/123456789_CloudTrail_us-east-1_20180927T1520Z_uF5y0kJRNhBDKloT.json.gz

AWSLogs/123456789/CloudTrail/us-east-1/2019/03/14/123456789_CloudTrail_us-east-1_20190314T0830Z_46eUvDSkGl95G.json.gz

AWSLogs/123456789/CloudTrail/us-east-1/2019/03/14/123456789_CloudTrail_us-east-1_20190314T0830Z_EQSwZlYCLgoYk.json.gz

[...]

Iterating over all accounts and regions

Get all account ids and regions from filenames. Instead of iterating over all filenames and manually grouping, Amazon can group common prefixes in filenames for us:

```python
def get_common_prefixes(prefix):
    return [os.path.basename(os.path.abspath(x['Prefix'])) for x in client.list_objects_v2(
        Bucket='a_bucket_name', Prefix=prefix, Delimiter='/')]['CommonPrefixes']
```

```python
>>> get_common_prefixes('')
['AWSLogs']
>>> get_common_prefixes('AWSLogs/123456789')
['CloudTrail']
>>> get_common_prefixes('AWSLogs/123456789/CloudTrail/)
>>> get_common_prefixes('AWSLogs/123456789/CloudTrail/ap-northeast-1/')
['2018', '2019']
>>> get_common_prefixes('AWSLogs/123456789/CloudTrail/ap-northeast-1/2018/')
['10', '11', '12']
```
Iterating over all accounts and regions

Iterate over the common prefixes returned from the `get_common_prefixes` function:

```python
for account in get_common_prefixes(prefix='AWSLogs/ '):
    for region in get_common_prefixes(prefix=f'AWSLogs/{account}/CloudTrail/ '):
        AWSprefix = f'AWSLogs/{account}/CloudTrail/{region}/'
        bucket_files = client.list_objects_v2(Bucket='a_bucket_name', StartAfter=AWSprefix + start_date,
                                        Prefix=AWSprefix)
```

**YO DAWG, I HEARD YOU LIKE LOOPS**

**SO WE PUT A LOOP IN YOUR LOOP SO YOU CAN LOOP WHILE YOU LOOP**
Fetching all files and not just the first 1000

The `list_objects_v2` function only returns 1000 elements on each call. To know if there are more elements to iterate use `IsTruncated` field in the return value. `ContinuationToken` parameter and `NextContinuationToken` return attribute are used to paginate over results.

```python
kwargs = { 'Bucket': 'a_bucket_name', 'StartAfter': AWSprefix + start_date, 'Prefix': AWSprefix}
continuation_token, is_truncated = None, True

while is_truncated:
    if continuation_token:
        kwargs['ContinuationToken'] = continuation_token

    bucket_files = client.list_objects_v2(**kwargs)
    is_truncated = bucket_files.get('IsTruncated', False) # Same as bucket_files['IsTruncated']
    continuation_token = bucket_files.get('NextContinuationToken') # Same as .get('Next..', None)
```

**Python Hacks:**
- Dynamically define arguments of `list_objects_v2` function:
  - [https://docs.python.org/3.7/glossary.html#term-argument](https://docs.python.org/3.7/glossary.html#term-argument)
- Return a default value if a key isn’t found in a dictionary using `.get` method:
  - [https://docs.python.org/3.7/library/stdtypes.html#dict.get](https://docs.python.org/3.7/library/stdtypes.html#dict.get)
Challenges with parsing:

- Each service logs in a different data structure:
  - All logs coming from Firehose are JSON joined without any separation
    - This is not a valid JSON, how to split and parse it?
  - VPC flow logs are CSV
  - CloudTrail is valid JSON
- Some keys have the same name but different type -> causes elasticsearch to fail.
- Amazon aggregates some events
- Logs can be stored as uncompressed, gzip, zip, or snappy.
- They must be uncompressed before parsing.

Working Production Example:
Parse log files: CloudTrail
Parse log files:
Firehose:
https://docs.python.org/3.7/library/json.html#json.JSONDecoder.raw_decode
Simple Dashboards

VPCFlow-Top10ExternalDestinationIP-Table

<table>
<thead>
<tr>
<th>Top 10 External Destination IP</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>54.239.31.225</td>
<td>4,406</td>
</tr>
<tr>
<td>54.239.25.71</td>
<td>4,282</td>
</tr>
<tr>
<td>54.239.25.60</td>
<td>4,150</td>
</tr>
<tr>
<td>54.239.30.177</td>
<td>2,478</td>
</tr>
<tr>
<td>54.239.30.195</td>
<td>2,465</td>
</tr>
<tr>
<td>45.127.112.2</td>
<td>1,709</td>
</tr>
<tr>
<td>54.239.29.61</td>
<td>1,469</td>
</tr>
<tr>
<td>66.241.101.63</td>
<td>477</td>
</tr>
<tr>
<td>209.141.60.238</td>
<td>291</td>
</tr>
</tbody>
</table>

GuardDuty-BreakdownOfAlertsPerInstancePerVPC-Pie

GuardDuty-MostCommonAccountID-Table

<table>
<thead>
<tr>
<th>Account ID</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>9638</td>
<td>40</td>
</tr>
</tbody>
</table>
Finding what matters

External Traffic Destinations

China

Massive spike in traffic to China

Traffic volume by destination IP

58.218.205.75

Most common destination
Finding what matters

58.218.205.75

Most common destination

Logs with Destination

Ec2 Instance involved

Hostname of findings

All Guardduty Findings for instance
Finding what matters

Most common destination

58.218.205.75

File responsible for outbound connection

VT Results
How did they get in?

```
[ec2-user@redTeam ~]$ nikto -host 172.16.0.22
Nikto v2.1.6
---------------------------------------------------------------------------
+ Target IP:          172.16.0.22
+ Target Hostname:    172.16.0.22
+ Target Port:        80
+ Start Time:         2019-04-15 15:19:57 (GMT0)
+ Server: Apache/2.4.37 () PHP/5.4.16
+ Server leaks inodes via ETags, header found with file /, fields: 0x1160x5861ba997b7ef
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion to the MIME type
+ Retrieved x-powered-by header: PHP/5.4.16
+ Multiple index files found: /index.php, /index.html
+ PHP/5.4.16 appears to be outdated (current is at least 5.6.9). PHP 5.5.25 and 5.4.41 are also current.
+ Allowed HTTP Methods: GET, POST, OPTIONS, HEAD, TRACE
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ /index.php?page=../../../../../../../../../../etc/passwd: The PHP Nuke Rocket add-in is vulnerable to file traversal, allowing an attacker to view any file on the host. (probably Rocket, but could be any index.php)
+ 8345 requests: 0 error(s) and 18 item(s) reported on remote host
+ End Time:           2019-04-15 15:20:09 (GMT0) (12 seconds)
+ 1 host(s) tested
```

### CVE-2001-1204

**Learn more at National Vulnerability Database (NVD)**
- CVSS Severity Rating
- Fix Information
- Vulnerable Software Versions
- SCAP Mappings
- CPE Information

**Description**

Directory traversal vulnerability in phprocketadd in Total PC Solutions PHP Rocket Add-in for FrontPage 1.0 allows remote attackers to read arbitrary files via a .. (dot dot) in the page parameter.

**References**

- **Note:** References are provided for the convenience of the reader to help distinguish between vulnerabilities. The list is not intended to be complete.
  - BID:3751
  - URL:http://www.securityfocus.com/bid/3751
  - BUGTRAQ:200111228 PHP Rocket Add-in (file transversal vulnerability)
  - URL:http://www.securityfocus.com/archive/1/247559
  - XF:phprocket-directory-traversal(7749)
  - URL:https://exchange.xforce.ibmcloud.com/vulnerabilities/7749

**Assigning CNA**

MITRE Corporation

**Date Entry Created**

20020315

**Disclaimer:** The **entry creation date** may reflect when the CVE ID was allocated or reserved, and does not necessarily indicate when this vulnerability was discovered, shared with the affected vendor, publicly disclosed, or updated in CVE.

**Phase (Legacy)**

Modified (20050705)

**Votes (Legacy)**

MODIFY(1) Frech
NOOP(5) Cole, Foat, Green, Wall, Ziese

**Comments (Legacy)**

Frech >:X:phprocket-directory-traversal(7749)

**Proposed (Legacy)**

20020315
Recap

Visibility

Accountability

<table>
<thead>
<tr>
<th>Traditional Tool</th>
<th>AWS equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single Pane of Glass (SPG) for Findings</td>
<td>Security Hub</td>
</tr>
<tr>
<td>IDS/IPS</td>
<td>GuardDuty</td>
</tr>
<tr>
<td>DLP</td>
<td>Macie</td>
</tr>
<tr>
<td>Netflow</td>
<td>VPCFlow</td>
</tr>
<tr>
<td>DNS</td>
<td>Route53</td>
</tr>
<tr>
<td>Access and authentication auditing</td>
<td>CloudTrail</td>
</tr>
<tr>
<td>Active Directory</td>
<td>Directory Service</td>
</tr>
<tr>
<td>Identity Management</td>
<td>IAM</td>
</tr>
<tr>
<td>Single Sign On</td>
<td>AWS SSO</td>
</tr>
<tr>
<td>Vulnerability scanner</td>
<td>Inspector</td>
</tr>
<tr>
<td>Configuration Management</td>
<td>AWS config</td>
</tr>
<tr>
<td>Logging Services Pipeline</td>
<td>Cloudwatch + Firehose + Lambda</td>
</tr>
</tbody>
</table>
Attacker Life Cycle

Personal
Flag it, Tag it, and Bag it.

Fear my squirrely wrath!

Fear it!