RSAConference2018
San Francisco | April 16 – 20 | Moscone Center

MATTERS NOW

SESSION ID: LAB2-W03

CLOUD DEFENDER: DETECTING AND RESPONDING TO ADVERSARIES IN AWS

Martin Holste

CTO, Cloud
FireEye, Inc.
@mcholste

Agenda

#RSAC

1. Overview and intro
2. Setup the demo environment
3. Understand the demo apps you are protecting
4. Red Team Activities (Offense)
   - A. Reconnaissance
   - B. Exploitation
5. Blue Team Activities (Defense)
   - A. Detection
   - B. Response

OVERVIEW AND INTRODUCTION

The importance of “red teaming”

Practicing adversarial situations is critical for validating security defenses.

Red Team:
Plan and execute exercises employing threat models

VS

Blue Team:
Continually executes detection and response

Benefits

This process is essential for a number of reasons:

- It gets the org thinking like an adversary and remembering that you're fighting against humans on the other end of the keyboard.
- It validates security solutions.
- The SOC knows there is someone targeting them at all times and keeps the analysts vigilant.

As the red team, you will…

1. Map out the cloud environment
2. Exploit the app using the provided script
3. Find and download the machine learning data
4. Alter machine learning data
5. Attack the business

As the blue team, you will…

1. Get familiar with available security controls
2. Centralize available logs
3. Hunt through logs for evidence of attack
4. Prepare a report
5. Create protective countermeasures
6. Create reactive detection capabilities

At the end you will…

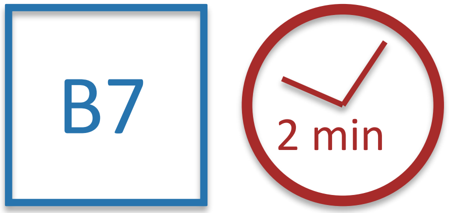
1. Review attack successes and failures
2. Review security controls and implementations
3. Understand the process of a targeted attacker and the challenges in defending against one.

GETTING STARTED

Take a Card

There is a card at your place which has lab login information. The username has a number corresponding to your role in it, e.g. LabUser1.

Each person will have a task to perform during the Red and Blue team sections, along with a time estimate for how long the task should take, e.g.:



which means LabUser7 has a Blue Team task that will take 2 minutes.

Completing Tasks

A note about completing tasks: Try to complete the task in the allotted amount of time, but don’t worry if you cannot. If you’re having trouble:

- Get help from teammates.
- Make use of the facilitators and ask questions.
- For code tasks, the working answer can be found in the lab materials (either the “red” or “blue” folder) in the Cloud9 environment. Use this if you get stuck.

Login

1. Login to the AWS account using the credentials on the card.
2. Go to the Cloud9 console in the AWS account. Open the lab environment under “shared with me.” The Cloud9 environment will have all lab materials and a terminal shell with which to run commands.
3. Each task will be performed by the designated person, but everyone should watch and take notes. For scripting, you can see live updates and help your teammate in the Cloud9 console. Collaborate as much as possible!
4. Open the “variables.txt” file in the Cloud9 console. It contains the values for <LAB API URL> and <TRAINING BUCKET>. Replace those placeholders with the values in the variables file whenever you encounter them in the lab materials.

Review the Environment

The lab consists of an imaginary Example.com production environment:

- Two websites with dynamic and static content running on S3 and API Gateway
- S3 buckets
- Lambdas
- CloudWatch Log Groups

RED TEAM

Making the Attack

Attack Scenario

Example.com is an imaginary financial institution that has millions of dollars of assets.

You are assuming the role of an advanced thief interested in compromising the integrity of the fraud alerting business process in order to steal money by making fraudulent transfers without being noticed.

Our mission

1. Find and understand the machine learning data underlying example.com.
2. Alter the training data example.com uses for fraud alerting.
3. Execute fraudulent transactions.

Reconnaissance

Our recon team scoured publicly available data on Google and social media for company references. They found:

- Example.com transactions page that we are trying to hack
- Example.com company “careers” web site with a resume submission capability

Reconnaissance

Take a look at the existing transactions page by opening a browser and going to <LAB API URL>/transactions.

Enter in any account number.

Try to issue a transaction for $1.

Try to issue a transaction for $1,000,000.

R1

5 min

Reconnaissance

During reconnaissance, the team found a reference to static.example.com in the HTML source code for the web page.

That is a CNAME to example-com-prod-static-assets.s3.amazonaws.com which means we have found a bucket named example-com-prod-static-assets.

There appears to be a naming convention prefix of example-com-prod-

Reconnaissance

You can determine the existence of an S3 bucket with a DNS query.

This allows for faster discovery of assets, especially if a naming convention is discovered.

Given the prefix example-com-prod-, the team used trial and error to find the bucket containing training materials.

- ✗ example-com-prod-training
- ✗ example-com-prod-fraud-training
- ✓ example-com-prod-fraud-alert-training

Reconnaissance

Now that we know our target (data in the discovered bucket example-com-prod-fraud-alert-training), we need to find a vector to get at it.

We check it to ensure that it is not publicly readable/writeable (keep in mind this is on a per-object basis, so there may be sub objects that are public).

Now we need to find our way in. Go to the index of the careers site (<API URL>/) and ”view source” to see if there is anything useful, then inform your team.

R2

5 min

Reconnaissance

We find this HTML source code of the URL “careers.example.com:”

```
<form action="resumes/upload" method="POST"
enctype="multipart/form-data">
…
<input name="resume" type="file"></input>
```

Reconnaissance

There’s another interesting section of the HTML which shows a different form parameter containing the relative URL path:

```
<form action=“resumes/preview” method="POST"
enctype="multipart/form-data">
<input name=“filename”></input>
```

This appears to potentially be a way to read files available to the backend Lambda if ‘filename’ isn’t sanitized.

Planning the Attack

Given this form, we see that there may be a file upload available, which can be abused if not properly protected.

The CNAME on the uploads site indicates this is hosted on AWS API Gateway because it ends with `.cloudfront.net`.

The assumption is that there is a Lambda handling the upload. We want to find out if it is vulnerable to a parameter injection.

Planning the Attack

Our goal is to be able to read and write data on S3. There are two parameter injections we want to succeed:

- Injecting an arbitrary file path into the preview function so we can read any file in example.com's S3 buckets
- Injecting an arbitrary file path into the upload function so we can write to any file in example.com's S3 buckets

The plan is to scope out the environment by abusing the preview function and alter the training data using the upload function.

Executing Parameter Injection

Lambda function code is often reused for various purposes, and as part of that process, a function will often allow itself to be invoked using different parameters depending on how it was triggered. One common pattern is to check if certain parameters were given.

```
if params.has_key(“Records”):
  # This was an S3 file notification trigger
  params[“bucket”] = params[“Records”][0][“bucket”][“name”]
  params[”key”] = params[“Records”][0][“s3”][“object”][“key”]
upload_to_bucket(params)
```

An example data structure for an S3 trigger

```
{"Records": [
  {
    "eventVersion": "2.0",
    "eventTime": "1970-01-01T00:00:00.000Z",
    "requestParameters": {"sourceIPAddress": "127.0.0.1"},
    "s3": {
      "configurationId": "testConfigRule",
      "object": {
        "eTag": "0123456789abcdef0123456789abcdef",
        "sequencer": "0A1B2C3D4E5F678901",
        "key": "HappyFace.jpg",
        "size": 1024
       },
     "bucket": {
       "arn":bucketarn,
       "name": "sourcebucket",
       "ownerIdentity": {"principalId": "EXAMPLE"}},
       "s3SchemaVersion": "1.0"
     },
     "responseElements": {
       "x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH",
       "x-amz-request-id": "EXAMPLE123456789"
     },
     "awsRegion": "us-east-1",
     "eventName": "ObjectCreated:Put",
     "userIdentity": {"principalId": "EXAMPLE"},
     "eventSource": "aws:s3"
  }
]}
```

Reading the Training Files

We found through brute-force that there is a bucket called com-prod-fraud-alert-training (actually <TRAINING BUCKET>) and we need to see what files are in it.

Craft a curl command to exploit the parameter in the preview function to try to read a file from <TRAINING BUCKET> called ”index.html.” It should use the JSON record format shown previously, which can be found in the lab materials folder under “red/reference/s3_record.json.”

It needs to execute a POST with application/json as the Content-Type.

R3

5 min

Reading the Training Files

```
curl -XPOST -H ”Content-Type: application/json”

https://<LAB API URL>/Prod/resumes/preview

-d '{"Records": [ { "s3": { "object": { "key":
”index.html" }, "bucket": { "name": ”<training
bucket>" } } } ] }'
```

Reading the Training Files

If you get a 4xx response, it means the file ‘index.html’ doesn’t exist. A 5xx response means we found a permission problem.

We can use brute-force with scripting to check for many possible options, trying different combinations until we get a 200 OK back.

Use the script “red/scripts/brute_find.py” to find the training file in the training bucket.

R4

5 min

Reading the Training Files

The script makes several assumptions:

- They are using a data-based folder system
- They are using a filename with some sort of naming convention.
- They are using a standard file extension.

Eventually, the script finds the full file path to download the latest fraud alert training file: 2018/04/18/fraud-training.csv.gz.

Altering the Data

Opening the CSV file shows thousands of entries of the form:

date, account-number, zip, amount

We want to be able to make random, large withdrawals, so we want to add entries for many dates with many zip codes for many account numbers for the category and amounts over $1,000,000.

We write a small script to create 1000 fake entries, concatenate with the downloaded file, and store the output locally as training.gz.

Now we need to upload it.

R5
R6

10 min

Executing Parameter Injection

Now that we have our altered training file, we need to overwrite the legitimate training file being used by the transaction system.

Craft a curl command to upload the altered file using the “resume,” “key,” and “bucket” form params.

R7

5 min

Executing Parameter Injection

The curl command should look like:

```
curl

-F resume=@CSV.gz

-F key=2018/04/18/training.csv.gz

-F bucket=<TRAINING BUCKET>

https://<LAB API URL>/resumes/upload
```

Verifying the Attack Worked

We can verify that we successfully wrote a new file by using the file viewing exploit from earlier when we were able to download the legitimate file:

```
curl -XPOST -H ”Content-Type: application/json”

https://<LAB API URL>/Prod/resumes/preview

-d '{"Records": [ { "s3": { "object": { "key":
”2018/04/18/training.csv.gz" }, "bucket": {
"name": ”<training bucket>" } } ] }’
```

The altered data should now take effect and we are free to make large transfers without example.com being alerted!

Steal a Lot of Money

#RSAC

Now that you have altered the file, steal all the money!

Take the curl command that succeeded and write a small script that will repeat the withdrawal 100 times.

R8

5 min

BLUE TEAM

Defending the Attack

Alerting

How do we know that something is wrong?

- Endpoint or network-based security?
  - No, this is serverless, and there are no exploits.
- Platform alerts (GuardDuty, etc.)?
  - This isn't originating from known-bad IP space.
- SIEM?
  - This doesn't match known-bad rules

Outside Notification

Sadly, in most events like this, initial awareness of the occurrence comes from external notification, often law enforcement.

In our scenario, we'll assume that the FBI has notified us that fraudulent transfers have occurred, and we need to investigate.

Finding the Problem

Why weren’t these transactions getting flagged?

After hours of debugging, the incident response team notices that the training file was last altered at an odd time of day (not midnight).

This centers the investigation around the tampered training file.

Investigation Tasks

1. Find who altered the training file last and when.
2. Find the differences between the latest training file and the previous version.

Investigate Altered Training Files

We will need to use AWS Athena to search our CloudTrail records for S3 object access.

1. Open the CloudTrail console
2. In the navigation pane, choose **Event history**.
3. In **Event history**, choose **Run advanced queries in Amazon Athena**.
4. For **Storage location**, choose the Amazon S3 bucket where log files are stored for the trail to query.
5. Choose **Create table**. The table is created with a default name that includes the name of the Amazon S3 bucket.
6. Click the “Go to Athena” button.

B1

1 min

Investigate Altered Training Files

Now we have our CloudTrail audit data in a format we can programmatically query. Let’s find out who modified our training file and when:

1. Click the “default” database
2. Click the three vertical dots next to the table name “cloudtrail_logs…” and choose “Preview table”
3. Edit the query to have a WHERE clause like the following:

B1

1 min

Investigate Altered Training Files

```
WHERE eventsource='s3.amazonaws.com’

AND json_extract_scalar(requestparameters,
'$.bucketName') LIKE ‘%fraudalert%’

AND eventname LIKE 'Put%’
```

Run the query and note the results.

B1

3 min

Download Latest and Previous Versions

Use the S3 console to download the latest version of the fraud alert training file in the fraud alert bucket, then use the version selector for the key to download the previous version.

Compare the two and note the differences.

B2

5 min

Backtrace the Altered File to Find Root Cause

Use API Gateway and Lambda logs to show the correlation between the assumed role that altered the file and the external IP that hit the API and invoked the Lambda.

1. Go to the S3 console, select the clouddefender-logging-… bucket, and choose permissions, then click "Bucket Policy." Add the policy located in blue/snippets/s3_log_export_policy.txt, updating the bucket to match the existing policy.
2. In the CloudWatch console, go to "Logs," and select the group starting "API-Gateway…" and click "Actions", then "Export to S3"
3. Export to the "clouddefender-logging-…" bucket with the prefix "apigateway"
4. Do the same for each of the log groups "/aws/lambda/CloudDefender-*" with the prefix "lambda"
5. In the Athena console, create two new tables using the SQL in blue/sql/api_gateway_table.sql and lambda_table.sql
6. Use the query template found in blue/sql/correlate.sql to isolate the Lambda invocations.

B3

5 min

Incident Report

Using the text editor of your choice, collect and organize the result logs from Athena that show a complete picture of what happened:

- API Gateway
- Lambda
- S3
- IAM/STS

B4

5 min

Implement Preventative Countermeasures

These are the remediation tasks for the security team:

1. Fix code
2. Lockdown Lambda permissions
3. Lockdown S3 resource permissions
4. Lockdown API Gateway parameter template
5. Add infringing IP addresses to WAF blacklist
6. Add infringing IP addresses to custom GuardDuty threat list

Implement Detection Techniques

How can we detect similar (but not exact) attacks?

We can't reliably detect on:

- Attacker IP
- Parameters or simple rules
- Attacks abusing other kinds of resources (DynamoDB, RDS, etc.)

Detection Through Business Logic Anomalies

This app has never:

- Transferred so much total money
- Made so many large transactions versus small

Detection Through Anomalies

This Lambda has never:

- Read from another bucket
- Written to another bucket
- Written so many bytes
- Been invoked so frequently

We need to detect and correlate these anomalies.

How Do We Find These Anomalies?

1. Instrument the app to be conducive to business logic anomaly detection by adding CloudWatch metrics and alarms
2. Get all logs writing to S3
3. Use Athena to produce reporting
4. Invoke Athena queries and take actions from a scheduled Lambda and use SNS for alerting.

Instrument the App with CloudWatch

Add a CloudWatch metric to the fraud detection app.

1. In Cloud9, find the FraudCheck function using the right-hand “AWS Resources” and ”Remote Functions” list.

2. Add in a line to write a metric for the amount transferred (in Cloud9 blue/snippets/cloudwatch.py):

```
boto3.client(‘cloudwatch’).put_metric_data(
    Namespace=‘FraudAlertApp’,
    MetricData={
            ’MetricName’: ‘Transactions’,
            ‘Dimensions’: [
                    { ‘Name’: ‘Amount’, ‘Value’: amount } ],
            ‘Unit’: ‘Dollars’ }
)
```

B5

5 min

3. Highlight the FraudCheck function in “Local Functions” and deploy the modified code using the “up arrow.”

Create a CloudWatch Alarm

Create a CloudWatch alarm for when the total amount transacted goes above $50,000 per 5 minutes.

1. Go to the CloudWatch console
2. Click ”Create Alarm”
3. Choose the metric category “FraudAlertApp”
4. Choose the metric “Transactions”
5. Choose “Sum” for the statistic
6. For the period, choose 5 minutes
7. Click next and name the alert
8. Specify 50000 as the threshold

B6

5 min

Centralized Logging

Stream CloudWatch logs for API Gateway and Lambda to Firehose to S3.

1. Create a Firehose stream
   1. Go to the Firehose console
   2. Choose "Create Delivery Stream"
   3. Choose an S3 destination
   4. Choose a new S3 bucket
   5. On the next page, use the default IAM settings
2. Stream the CloudWatch log group to Firehose
   1. Go to CloudWatch
   2. Select the API Gateway log group and click "Actions"
   3. Choose stream to Kinesis Firehose as the subscription

B7

5 min

Create Detection Lambda

This Lambda function will run some automated queries for correlation on a schedule.

Use the script in Cloud9 located at blue/lambda/investigate.py as your Lambda.

Create a new Lambda function with this file and run it using "Test."

B8

5 min

Investigation Lambda

The Lambda will:

1. Make Athena query for first-seen bucket reads
2. Make Athena query for first-seen bucket writes
3. Make Athena query for deviant counts of bytes by Lambda
4. Make Athena query for deviant counts of Lambda invocations
5. Determine if any anomalies coincide, and if so, take action.

RSAConference2018

NOW MATTERS

#RSAC

CONCLUSION

Summary

- Cloud removes a lot of inherent vulnerabilities but can create knowledge gaps with developers, security staff, and operations leading to mistakes in coding and/or deployment.
- Exploiting these mistakes means thinking like devops.
- Defending these cloud apps means understanding the business logic and thinking like an attacker.

Action Items

1. Review and inventory your cloud presence.
2. Ensure that you are logging everything centrally and instrumenting your apps for tracking business logic.
3. Ensure that developers are trained on proper permissions.
4. Ensure that proper permissions are being followed using built-in platform tools like Config and Inspector.
5. Ensure that fundamental security for network, endpoint, and email are in place.
6. Ensure that analytics-driven security is in place.