'lambdaScanner' – Scan & Secure Toolkit

Footprint, Enumerate, Scan, Tunnel, Trace, Watch & Protect Lambda

Table of Contents

Objective

'lambdaScanner' is a toolkit which has a combination of scripts for performing penetration testing of lambda functions. The scripts available in the toolkit help assessing the lambda functions from a security standpoint. It helps the tester to discover vulnerabilities in deployment as well as code. It aids in checking vulnerabilities like improper permissions, SQL injections, command executions etc. to name a few. This is not an automated scanner, but a toolkit that helps pen-testers to perform the testing of functions, so it needs to be used wisely by crafting customized requests and payloads. The lambda functions are invoked through various events encompassing AWS like S3, DynamoDB, SQS etc. so the scripts in the toolkit are very helpful in evaluating functions as well as directly testing with various sets of payloads. All these scripts are written in python by using boto3 APIs. The toolkit also has a package called 'lambdaProtect' which can be integrated with an existing lambda function to guard both incoming event stream as well as outgoing response.

This toolkit is "in progress/prototype" and would be enhanced with time by an addition of various functionalities.

Here is a diagram, which describes 'lambdaScanner': -

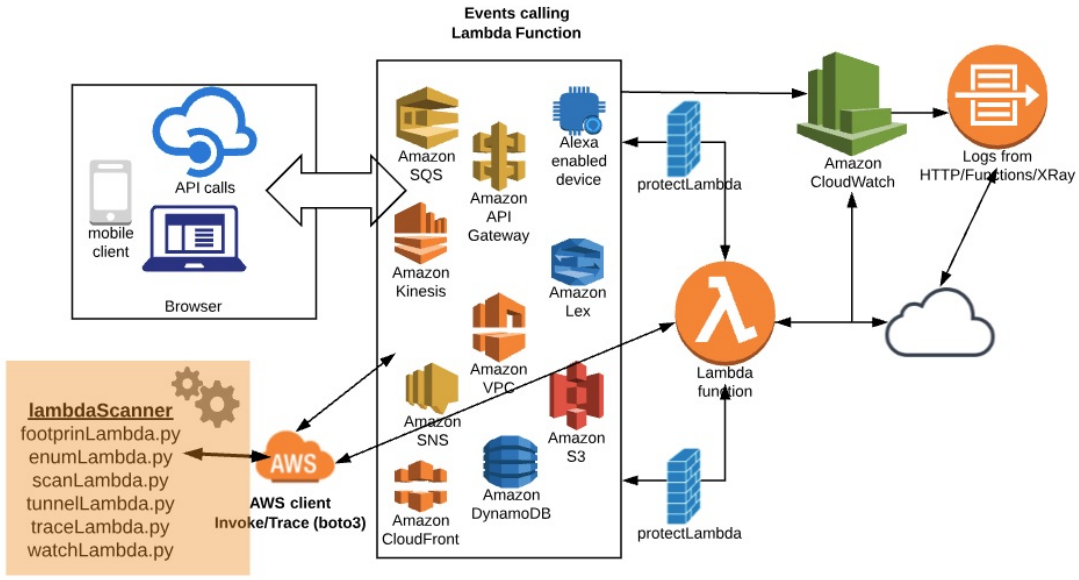


Figure 1 – Lambda Function Architecture and Testing Methodology using 'lambdaScanner'

As shown in the figure, the lambda function, which is well integrated to all components, can be accessed through various devices like mobile, browser or even APIs. The web application or other components call the lambda function and this fires a set of events from time to time. At the same time, since it makes changes to various places at every point of execution, we can access all these footprints via APIs. With the use of boto3 client, we are directly accessing and testing the lambda function as shown in the figure. We can also deploy 'lambdaProtect' for securing the events as well as output.

Requirements

To run 'lambdaScanner' scripts, one needs to install the following: -

(+) Python3
(+) Boto3 (Library for AWS)

For the installation of boto3, you can find more information and guidance from here: - https://pypi.org/project/boto3/

Setting up AWS Credentials for Testing:

You can either directly write your credentials to the configuration file in your shell or use AWS client to set it up: - https://docs.aws.amazon.com/cli/latest/userguide/cli-chap-getting-started.html

You can find more information and options over here: - https://boto3.readthedocs.io/en/latest/guide/configuration.html

Once credentials are set, you can start using 'lambdaScanner' toolkit. It has the following seven scripts in place: -

1. footprintLambda
2. enumLambda
3. scanLambda
4. tunnelLambda
5. traceLambda
6. watchLambda
7. protectLambda

Let's go through each one of them and understand its usage.

footprintLambda

'footprintLambda' searches the footprints of the lambda function across various AWS components like S3, DynamoDB, SQS etc. The script will go through all the data and establish a footprinting relationship with the lambda function and other components. It helps in identifying events supported by that lambda function.

Following is the basic help: -

```
$ python3 footprintLambda.py

================================================================
footprintLambda - Lambda Function Footprinting Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

usage: footprintLambda.py [-h] [-p]

optional arguments:
  -h, --help  show this help message and exit
  -p          use for footprint and mapping
$
```

Let's use "–p" to start footprinting.

```
$ python3 footprintLambda.py -p

================================================================
footprintLambda - Lambda Function Footprinting Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

(+) Collecting list of Lambda functions for footprinting...
        (-)Lambda: getFileObj - arn:aws:lambda:us-east-2:313588302550:function:getFileObj
        (-)Lambda: listener - arn:aws:lambda:us-east-2:313588302550:function:listener
        (-)Lambda: cuttrace - arn:aws:lambda:us-east-2:313588302550:function:cuttrace
        (-)Lambda: xlogin - arn:aws:lambda:us-east-2:313588302550:function:xlogin
        (-)Lambda: login - arn:aws:lambda:us-east-2:313588302550:function:login
        (-)Lambda: processInvoice - arn:aws:lambda:us-east-2:313588302550:function:process
```

Here, we can see that the tool discovered that the "processInvoice" function is triggered by SQS service. Hence, now we can fuzz the function with SQS event structure.

```
(+) Collecting list of configured Dynamodb tables for footprinting...

(+) ==> Final Footprint and Mapping Results...
        (-)getFileObj (arn:aws:lambda:us-east-2:313588302550:function:getFileObj) -
        (-)listener (arn:aws:lambda:us-east-2:313588302550:function:listener) -
        (-)cuttrace (arn:aws:lambda:us-east-2:313588302550:function:cuttrace) -
        (-)xlogin (arn:aws:lambda:us-east-2:313588302550:function:xlogin) -
        (-)processInvoice (arn:aws:lambda:us-east-2:313588302550:function:processInvoice) - arn:aw
s:sqs:us-east-2:313588302550:processInvoice
        (-)trace (arn:aws:lambda:us-east-2:313588302550:function:trace) -
        (-)getuserinfo (arn:aws:lambda:us-east-2:313588302550:function:getuserinfo) -
$
```

enumLambda

'enumLambda' helps in identifying all the functions deployed in the environment and then one can identify key attributes of any targeted function. The tool will fetch and enumerate the function.

Following is the basic help: -

```
$ python3 enumLambda.py

============================================================
enumLambda - Lambda Function Enumeration Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

usage: enumLambda.py [-h] [-f FUNCTION_NAME] [-e]

optional arguments:
  -h, --help        show this help message and exit
  -f FUNCTION_NAME  use function name for enumeration
  -e                list all functions
$
```

First just use "-e" switch and enumerate the list of functions deployed.

```
$ python3 enumLambda.py -e

============================================================
enumLambda - Lambda Function Enumeration Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

(+) Lambda functions ...
   (-)getFileObj
   (-)listener
   (-)cuttrace
   (-)xlogin
   (-)login
   (-)processInvoice
   (-)env
   (-)trace
   (-)getuserinfo

$
```

Now, we can enumerate the function "login" if interested in testing that function. We can run the following command using "-f" switch.

```
$ python3 enumLambda.py -f login

============================================================
enumLambda - Lambda Function Enumeration Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

(+) Fetching Lambda function login...
       (+) Platform: nodejs6.10
       (+) Permission: arn:aws:iam::313588302550:role/service-role/access
       (+) Permission Policy: access
                ==> {"Effect": "Allow", "Action": ["xray:PutTraceSegments", "xray:PutTelemetryReco
rds"], "Resource": ["*"]}
                ==> [{"Effect": "Allow", "Action": "logs:CreateLogGroup", "Resource": "arn:aws:log
s:us-east-2:313588302550:*"}, {"Effect": "Allow", "Action": ["logs:CreateLogStream", "logs:PutLogE
vents"], "Resource": ["arn:aws:logs:us-east-2:313588302550:log-group:/aws/lambda/login:*"]}]
       (+) xRay-Tracing: Active
       (+) Code-Location: https://awslambda-us-east-2-tasks.s3.us-east-2.amazonaws.com/snapshots/3
```

```
---Function Mapping---
[]
---Function Policy---
   (+)Point of Service: apigateway.amazonaws.com
        (-)Entry Point: arn:aws:execute-api:us-east-2:313588302550:          /*/POST/
   (+)Point of Service: apigateway.amazonaws.com
        (-)Entry Point: arn:aws:execute-api:us-east-2:313588302550:          /*/POST/login
   (+)Point of Service: apigateway.amazonaws.com
        (-)Entry Point: arn:aws:execute-api:us-east-2:313588302550:          /*/POST/login
   (+)Point of Service: apigateway.amazonaws.com
        (-)Entry Point: arn:aws:execute-api:us-east-2:313588302550:          /*/POST/login
   (+)Point of Service: apigateway.amazonaws.com
        (-)Entry Point: arn:aws:execute-api:us-east-2:313588302550:          /*/OPTIONS/login
$
```

We can get very critical information like technology stack, role, permission, policy for the role, location to get source code, mapping and integration with API gateway etc. This set of information can help in identifying security issues and allow one to invoke and scan the function.

scanLambda

'scanLambda' helps in invoking and fuzzing the function with different values being injected into the event stream. We can do full scanning by passing different values and see its impact on the function. Based on the error messages or behaviour we can discover vulnerabilities within the function.

Following is the basic help: -
It has two main switches "-i" and "-s".

```
$ python3 scanLambda.py

============================================================
scanLambda - Lambda Scanner Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

usage: scanLambda.py [-h] [-i] [-f FUNCTION] [-s] [-e EVENT]

optional arguments:
  -h, --help   show this help message and exit
  -i           use for invoke the function
  -f FUNCTION  use function for scan/invoke
  -s           use for scan the function
  -e EVENT     supply event file
$
```

We have a file called "event.txt", as shown below, which has a stream of events written in it. These events can be of any type - be it a message, S3, Alexa or simple name value pairs. We can pass this file along with the target function.

```
$ ls
enumLambda.py           readme.txt              traceLambda.py
events                  scan-config             watch-config
footprintLambda.py      scanLambda.py           watchLambda.py
$ cd events
$ ls
event.txt
$ cat event.txt
{
        "login":"john",
        "password":"letmein"
}$
```

As shown below, we are using "-i" to invoke the function, selecting the "login" function to be invoked and passing our "event.txt" file. This command will invoke the function, show its response, logs and the response ID which can be used to trace the request/response in the logs.

```
$ python3 scanLambda.py -i -f login -e ./events/event.txt

============================================================
scanLambda - Lambda Scanner Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

(+)Configuring Invoking ...
        (-) Loading event from file ...
        (-) Request Id ==> 41e55821-a2f0-11e8-8e8f-6d45acdcb248
        (-) Response ==>
        (-) {"status":"success"}
        (-) Log ==>START RequestId: 41e55821-a2f0-11e8-8e8f-6d45acdcb248 Version: $LATEST
2018-08-18T14:09:01.885Z        41e55821-a2f0-11e8-8e8f-6d45acdcb248    { login: 'john', pass
 'letmein' }
END RequestId: 41e55821-a2f0-11e8-8e8f-6d45acdcb248
REPORT RequestId: 41e55821-a2f0-11e8-8e8f-6d45acdcb248  Duration: 62.06 ms      Billed Durati
00 ms   Memory Size: 128 MB     Max Memory Used: 20 MB
```

Now let's move ahead and scan this function with different values (fuzz). We have a folder called "scan-config". This folder has two files – "payload.txt" and "regex.txt". The payloads or the fuzzing values can be defined in the "payload.txt" file in separate lines as shown below. At the same time, we can put regular expressions which we want to check against in the "regex.txt" file.

```
$ ls
enumLambda.py           readme.txt              traceLambda.py
events                  scan-config             watch-config
footprintLambda.py      scanLambda.py           watchLambda.py
$ cd scan-config/
$ ls
payload.txt     regex.txt
$ cat payload.txt
john
\"
'
1 and 1=1$ cat regex.txt
.*error.*$
$
```

In the "event.txt" file, we can define injection points with $fuzz$. The script will take values from the "payload.txt" file and change at this position before invoking the function.

```
$ ls
enumLambda.py           readme.txt              traceLambda.py
events                  scan-config             watch-config
footprintLambda.py      scanLambda.py           watchLambda.py
$ cd events/
$ ls
event.txt
$ cat event.txt
{
        "login":"$fuzz$",
        "password":"letmein"
}$
```

We can run and see the output on the screen or dump it to a file. The analysis will show if a regex matches to the output. It helps in defining and discovering vulnerabilities within the function. This is how we can scan functions.

```
$ python3 scanLambda.py -s -f login -e ./events/event.txt

============================================================
scanLambda - Lambda Scanner Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

(+)Configuring scanner ...
        (-)Loading payload from file ...
        (-)Loading regex from file ...
        (-)Loading event from file ...
(+) ==> ----------------------------
        (-) Injecting payload - john
        (-) Request Id ==> 27b2b005-a2f1-11e8-9dbf-b56ebd42ea67
        (-) Response Analysis
{"status":"success"}
(+) ==> ----------------------------
        (-) Injecting payload - \"
        (-) Request Id ==> 27f07caa-a2f1-11e8-a08b-b91e43454c44
        (-) Response Analysis
{"status":"Error fetching value from table"}
        (-) Regex found: .*error.*
        (-) {"status":"Error fetching value from table"}
        (-) Log ==>START RequestId: 27f07caa-a2f1-11e8-a08b-b91e43454c44 Version: $LATEST
2018-08-18T14:15:27.562Z        27f07caa-a2f1-11e8-a08b-b91e43454c44    { login: '"', passworc
etmein' }
END RequestId: 27f07caa-a2f1-11e8-a08b-b91e43454c44
```

tunnelLambda

'tunnelLambda' helps in establishing a tunnel from your shell to a targeted lambda function. It helps in sending HTTP traffic to the selected port, which will automatically tunnel to the test function. Hence, now we can use some standard HTTP tools like Burp or ZAP to test the lambda function.

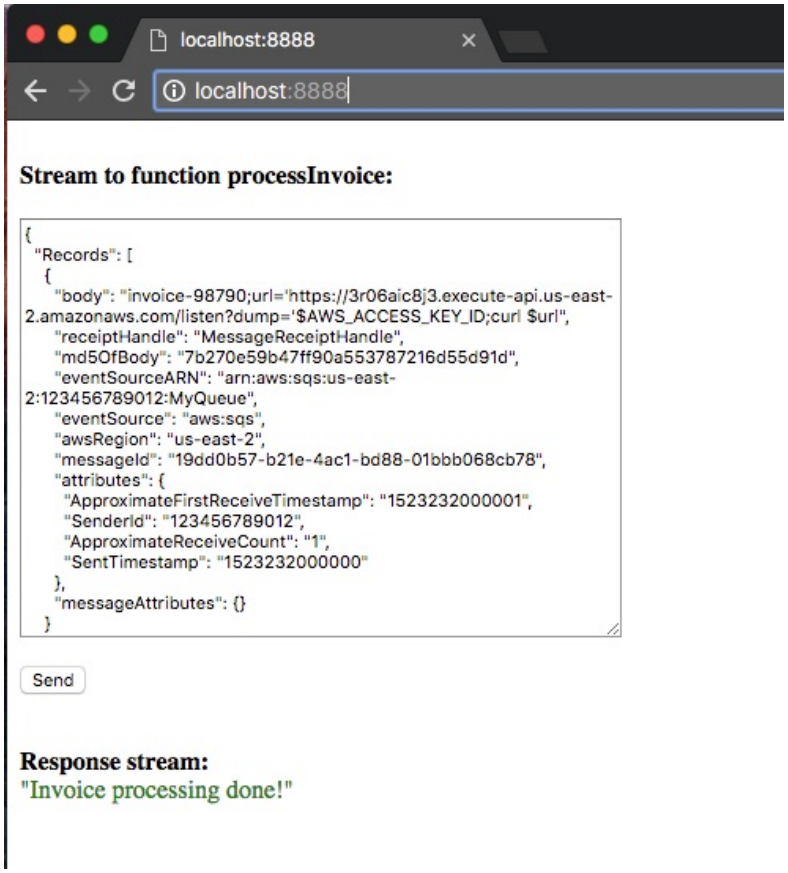
Following is the basic help: -
We have to pass two parameters – function and port using "-f" and "–p" switches.

```
[bliss$python3 tunnelLambda.py

============================================================
tunnelLambda - Lambda Tunneling Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

usage: tunnelLambda.py [-h] [-f FUNCTION] [-p PORT]

optional arguments:
  -h, --help   show this help message and exit
  -f FUNCTION  use function for http tunneling
  -p PORT      supply port for listening
bliss$
```

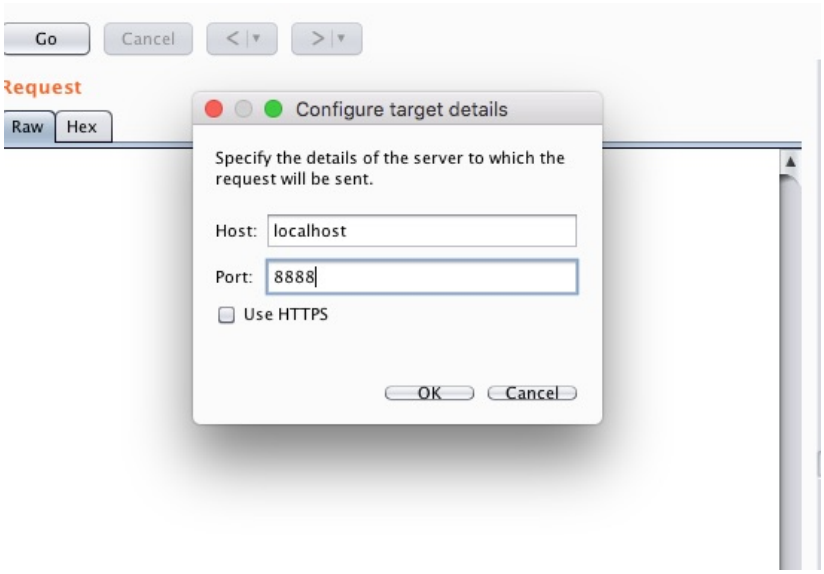
Once it is set, the script will listen on the target port for both GET and POST requests as shown below: -

```
[bliss$python3 tunnelLambda.py -f processInvoice -p 8888

============================================================
tunnelLambda - Lambda Tunneling Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

Established tunnel for lambda function -  processInvoice
Listening on Port -  8888
127.0.0.1 - - [25/Aug/2018 10:18:29] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [25/Aug/2018 10:18:29] "GET /favicon.ico HTTP/1.1"
127.0.0.1 - - [25/Aug/2018 10:18:41] "POST / HTTP/1.1" 200 -
```

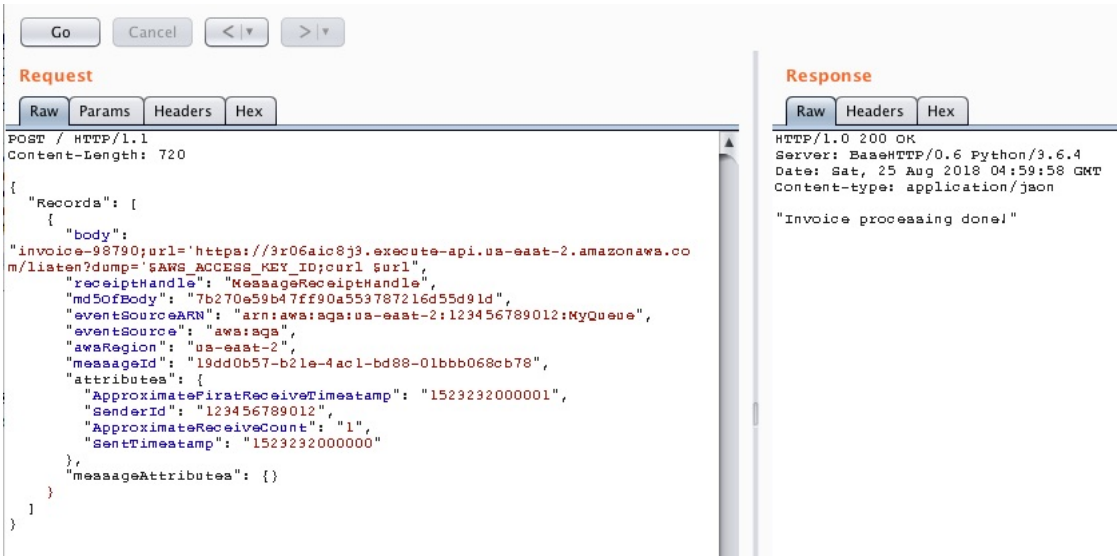
When you make a GET request it will serve a simple HTML page which can be used to interact with the lambda function as shown in the below figure. We can just open the page in a browser, put the event stream and click on "Send" button. It will show the output once it is invoked.



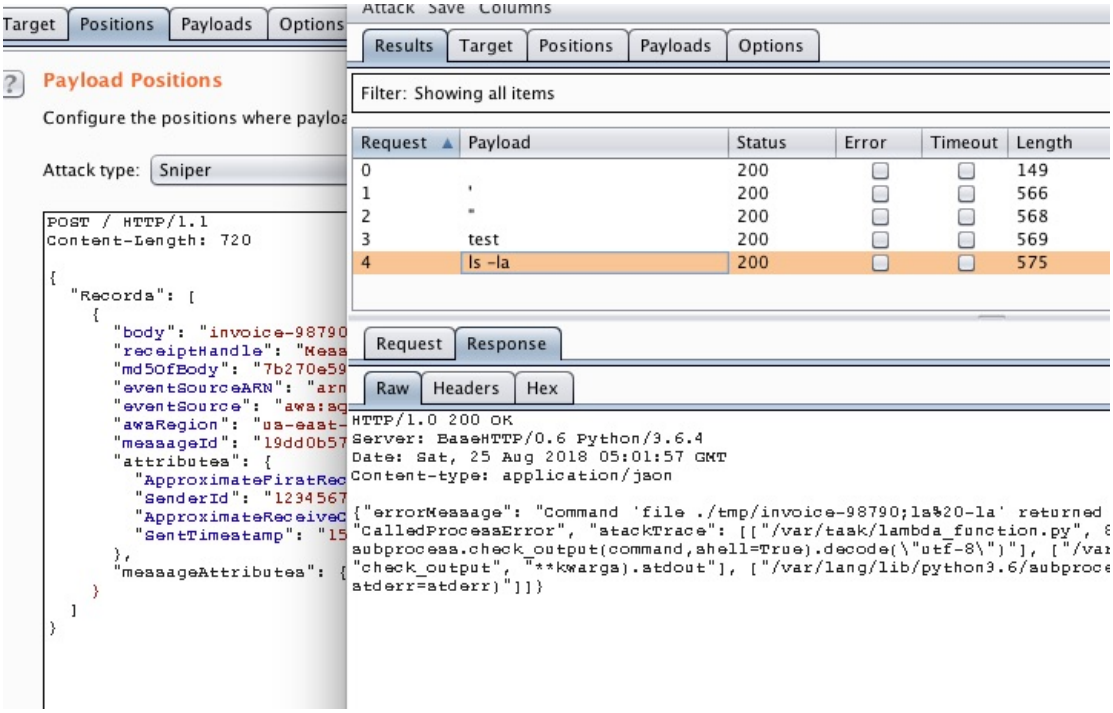
Also, we can use Burp or any other tool to make a POST request directly. We need to configure the details in Burp repeater as shown below: -



Once it is set, we can make the call as shown below: -



Next, we can simply send the request to intruder and run attacks as shown below: -



traceLambda

'traceLambda' helps in tracing the request ID within various logs on CloudWatch.

Following is the basic help: -

```
bliss$python3 traceLambda.py

================================================================
traceLambda - Lambda Function Tracing Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

usage: traceLambda.py [-h] [-i ID] [-f FUNCTION] [-x]

optional arguments:
  -h, --help   show this help message and exit
  -i ID        use id for trace fetching
  -f FUNCTION  use function for trace fetching
  -x           use xRay to get trace of 120 seconds
bliss$
```

As shown below, we can first get the request ID from the invoke call. Also, we can get a similar ID from the HTTP call made through an API gateway, as a part of HTTP response.

```
$ python3 scanLambda.py -i -f login -e ./events/event.txt

================================================================
scanLambda - Lambda Scanner Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

(+)Configuring Invoking ...
        (-) Loading event from file
        (-) Request Id ==> 6d578438-a2f1-11e8-b8f0-39cb969a61f8
        (-) Response ==>
        (-) {"status":"Error fetching value from table"}
        (-) Log ==>START RequestId: 6d578438-a2f1-11e8-b8f0-39cb969a61f8 Version: $LAT
2018-08-18T14:17:24.025Z        6d578438-a2f1-11e8-b8f0-39cb969a61f8    { login: '$fuz
d: 'letmein' }
END RequestId: 6d578438-a2f1-11e8-b8f0-39cb969a61f8
REPORT RequestId: 6d578438-a2f1-11e8-b8f0-39cb969a61f8  Duration: 5.02 ms       Billed
00 ms   Memory Size: 128 MB     Max Memory Used: 29 MB

$
```

Once we have the ID, it can be passed with "-i" switch to the script and it will grab the log entries from CloudWatch logs as shown below: -

```
bliss$python3 traceLambda.py -i 6d578438-a2f1-11e8-b8f0-39cb969a61f8

============================================================
traceLambda - Lambda Function Tracing Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

(+)Searching logs in /aws/lambda/cuttrace
(+)Searching logs in /aws/lambda/env
(+)Searching logs in /aws/lambda/getFileObj
(+)Searching logs in /aws/lambda/hemil
(+)Searching logs in /aws/lambda/listener
(+)Searching logs in /aws/lambda/login
------------------------------------------------
    (-)START RequestId: 6d578438-a2f1-11e8-b8f0-39cb969a61f8 Version: $LATE

------------------------------------------------
    (-)2018-08-18T14:17:24.025Z 6d578438-a2f1-11e8-b8f0-39cb969a61f8    { '
 '$fuzz$', password: 'letmein' }

------------------------------------------------
    (-)END RequestId: 6d578438-a2f1-11e8-b8f0-39cb969a61f8
```

Next, we can get all logs for a specific function. Here, we are getting the logs for "login" function.

```
bliss$python3 traceLambda.py -f login

============================================================
traceLambda - Lambda Function Tracing Script (beta)
(c) Blueinfy solutions pvt. ltd.
============================================================

(+)Dumping traces from /aws/lambda/login
------------------------------------------------
    (-)START RequestId: 2014c680-9961-11e8-9971-93303b726338 Version: $LATE

------------------------------------------------
    (-)2018-08-06T10:11:46.676Z 2014c680-9961-11e8-9971-93303b726338    {}

------------------------------------------------
    (-)END RequestId: 2014c680-9961-11e8-9971-93303b726338

------------------------------------------------
    (-)REPORT RequestId: 2014c680-9961-11e8-9971-93303b726338   Duration: 5
ms      Billed Duration: 100 ms         Memory Size: 128 MB     Max Memory
 20 MB
```

Lambda is having support for instrumentation and extension of the function via xRay services. It helps in tracking various things at runtime for each invoke. It provides traces along with external calls like time taken to access DynamoDB or S3 etc. It helps in discovering vulnerabilities as well. Also, developer/pentester can inject code to fetch information at runtime as well. We can use "-x" switch to fetch last few traces (120 seconds).

```
bliss$python3 traceLambda.py -x

================================================================
traceLambda - Lambda Function Tracing Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

------------------------------------------------
1-5b80fa00-f3466790ba424e8a8be04055
{
   "ResponseMetadata": {
      "RequestId": "df4561f1-a831-11e8-870a-6b99d510e0bd",
      "HTTPStatusCode": 200,
      "HTTPHeaders": {
         "date": "Sat, 25 Aug 2018 06:41:18 GMT",
         "content-type": "application/json",
         "content-length": "1183",
         "connection": "keep-alive",
         "x-amzn-requestid": "df4561f1-a831-11e8-870a-6b99d510e0bd"
      },
      "RetryAttempts": 0
```

watchLambda

'watchLambda' enables investigating and monitoring of attacks on the lambda function. We can make a list of attack signatures in the rules file and run it against the logs.

Following is the basic help: -

```
[$ python3 watchLambda.py

================================================================
watchLambda - Lambda Function Log Scanning Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

None
None
usage: watchLambda.py [-h] [-f FUNCTION] [-r RULES]

optional arguments:
  -h, --help   show this help message and exit
  -f FUNCTION  use function for scan/invoke
  -r RULES     supply regex rule file
$
```

As shown below, the "rules.txt" file has a set of regex. These patterns get compared within the logs of the targeted function.

```
[$ ls
enumLambda.py           readme.txt              traceLambda.py
events                  scan-config             watch-config
footprintLambda.py      scanLambda.py           watchLambda.py
[$ cd watch-config/
[$ ls
rules.txt
[$ cat rules.txt
.*1=1*.
.*letmein*.
.*curl *.$
```

Now, by running the tool as shown below, we can get a list of points where the pattern is matching. It helps in discovering possible attacks. We can look into the logs and further analyse a specific instance.

```
[$ python3 watchLambda.py -f login -r ./watch-config/rules.txt

================================================================
watchLambda - Lambda Function Log Scanning Script (beta)
(c) Blueinfy solutions pvt. ltd.
================================================================

./watch-config/rules.txt
login
(+)Configuring Watcher ...
        (-)Loading rules from file ...
(+)Scanning for signatures in /aws/lambda/login
(Found ==> ).*letmein*.
2018-08-17T10:11:39.286Z        ee3e5aa3-a205-11e8-b8af-493a384821bc    { user: 's
rd: 'letmein' }

(Found ==> ).*letmein*.
```

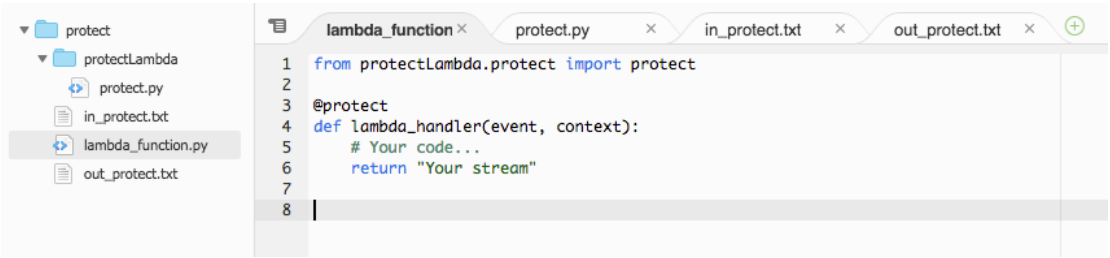
Here, for example we can see an instance of a blind SQL injection probe.

```
(Found ==> ).*1=1*.
2018-08-18T08:12:33.784Z        75bfa5dc-a2be-11e8-ae66-95d9863498c9    { login: '1 and 1=1', p
word: 'letmein' }

(Found ==> ).*letmein*.
2018-08-18T08:12:33.784Z        75bfa5dc-a2be-11e8-ae66-95d9863498c9    { login: '1 and 1=1', p
word: 'letmein' }
```

protectLambda

'protectLambda' is a lambda protection function that needs to be deployed with the function code.

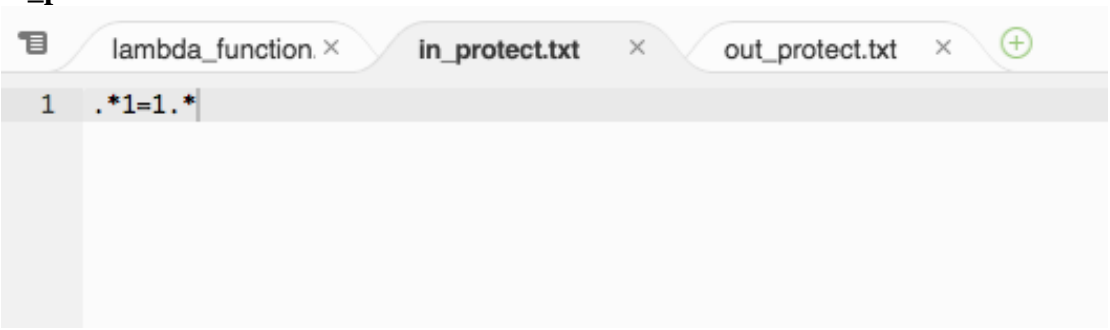
We need to add this folder in the project and add the following lines of code for protection as shown below: -



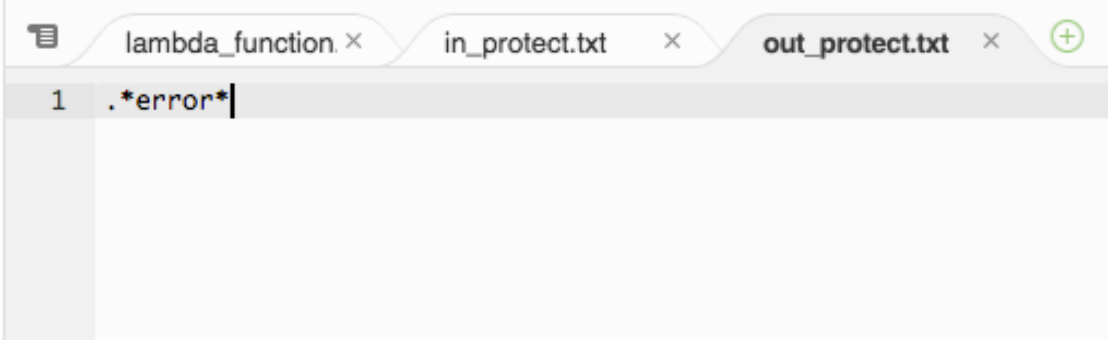

```
from protectLambda.protect import protect

@protect
def lambda_handler(event, context):
```

Once this is done, we can add rules for both incoming event stream as well as outgoing stream by regex patterns. We have two files added to the project – "in_protect.txt" and "out_protect.txt" as shown below with dummy/real rules.

in_protect.txt



out_protect.txt



Now, if we try to inject a function with 1=1 payload then we get the following output: -

Request Stream:

Output:

This way we can secure both incoming and outgoing streams for lambda function.

Release Notes

Date: 18th August, 2018
Version 1.0 (beta) – Basic Release