

Security Analysis of Serverless Functions



sdmay24-26 (Dillon H, Cameron H, Michael G, Samuel P, Trent W)
Client and Advisor: Dr. Berk Gulmezoglu

Background

- What is a serverless function?
- What are some use cases for Serverless Functions?

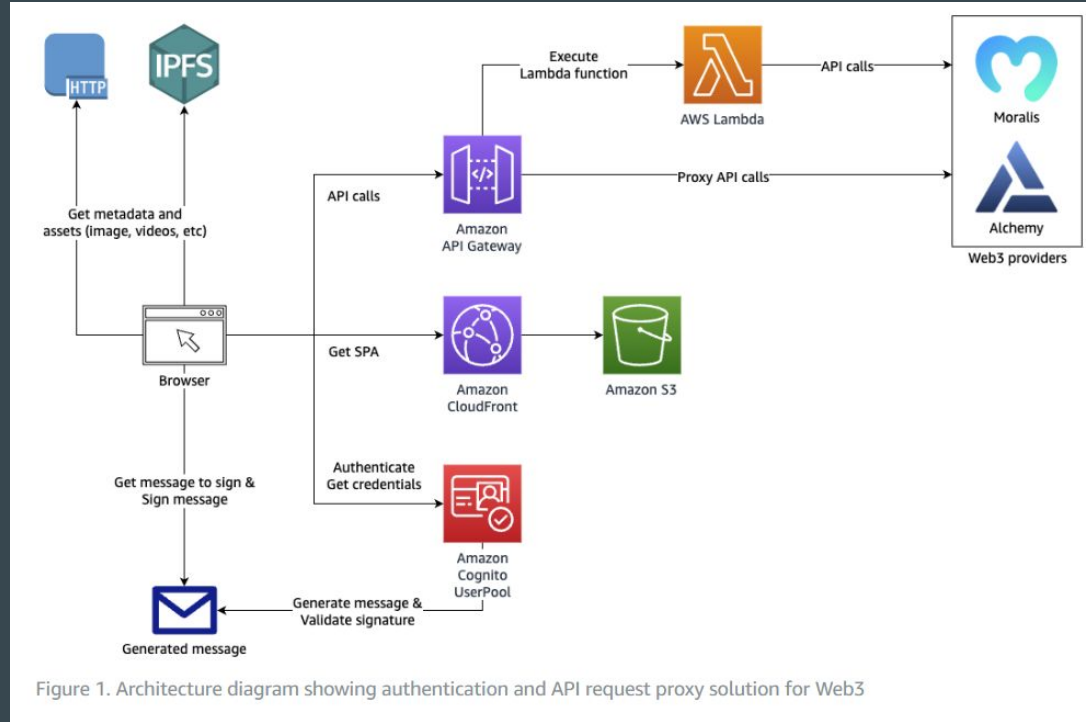
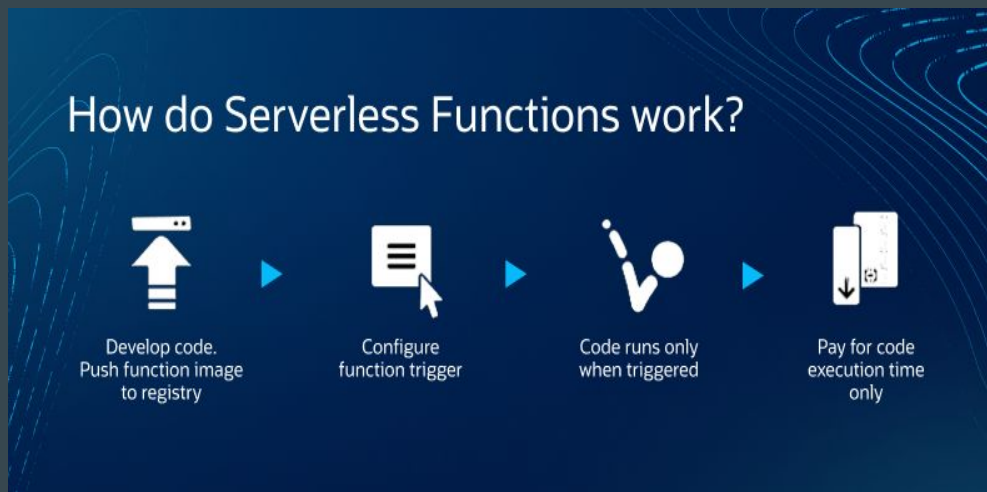


Figure 1. Architecture diagram showing authentication and API request proxy solution for Web3

Background

- How do Serverless Functions work
 - Code Deployment
 - Event Trigger
 - Resource Allocation
 - Code Execution
 - Response
 - Resource Deallocation
- Why are these functions advantageous?
 - Cost efficient
 - Scalability
 - Developer Productivity
 - Reduced Overhead



<https://developer.oracle.com/learn/technical-articles/serverless-functions>

Project Requirements

- Functional
 - Documentation, vulnerabilities
- Resources
 - Server, environment setup, Lambdas
- Legal requirements
- Performance
 - Memory, processing, minimum output constraints
- Maintainability
- Testing requirements
 - Environment should mimic AWS as much as possible
 - Consistency of attack code output



Project Design - Backend

- Automated Lambda Creation
- Provide additional AWS services
- Routing of function requests
- Dynamic Networking
- Changeable MicroVM settings
- Uses firecracker for quick VM startup
- Automated enumeration of lambda functions
- Dynamically builds runtime image
- Starting and stopping MicroVMs from an API

The screenshot displays a web interface for managing MicroVMs and a terminal window showing system metrics.

Web Interface Tables:

VM Name	IP Address	Status	PID
big-core2	192.168.24.106	<input type="button" value="Stop Function VM"/>	

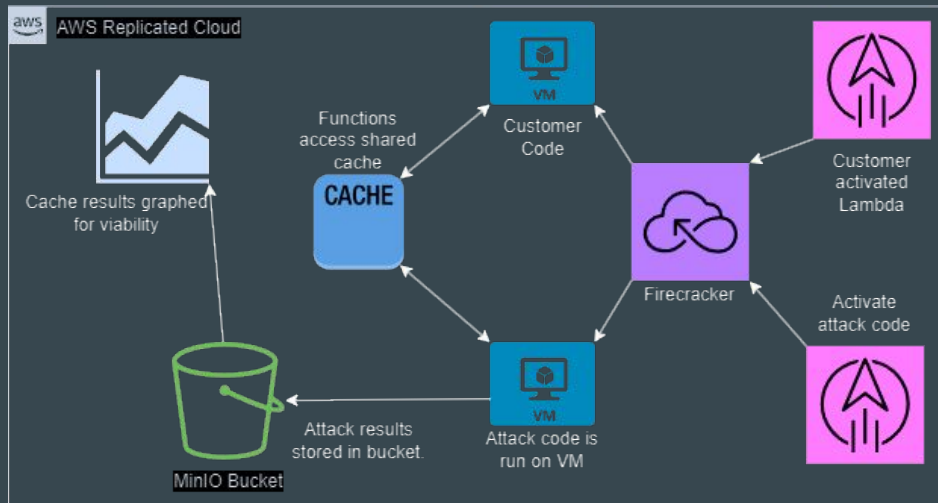
Function Name	Runtime	Has Image	Start
big-core2	python3.9 8192 MB 6 CPU cores	True <input type="button" value="Rebuild"/>	<input type="button" value="Start"/>

Terminal Window:

```
trwbox -- snmay24_16@berk: -- ssh senior-design -- 168x29
1 [|||||||||||||||||98.1%] 17 [|||||||||||||||||100.0%] 33 [||||| 1.3%] 49 [||||| 0.0%]
2 [||||| 0.0%] 18 [||||| 0.0%] 34 [||||| 0.0%] 50 [||||| 0.0%]
3 [||||| 0.0%] 19 [||||| 0.0%] 35 [||||| 0.0%] 51 [||||| 76.5%]
4 [||||| 0.0%] 20 [||||| 0.0%] 36 [||||| 0.0%] 52 [||||| 0.0%]
5 [||||| 0.0%] 21 [||||| 0.0%] 37 [||||| 0.0%] 53 [||||| 0.0%]
6 [||||| 0.0%] 22 [||||| 0.0%] 38 [||||| 0.0%] 54 [||||| 0.0%]
7 [||||| 0.0%] 23 [||||| 0.0%] 39 [||||| 0.0%] 55 [||||| 0.0%]
8 [||||| 0.0%] 24 [||||| 98.0%] 40 [||||| 0.0%] 56 [||||| 1.9%]
9 [||||| 0.0%] 25 [||||| 0.0%] 41 [||||| 0.0%] 57 [||||| 0.0%]
10 [||||| 0.0%] 26 [||||| 0.0%] 42 [||||| 0.0%] 58 [||||| 0.0%]
11 [||||| 0.0%] 27 [||||| 100.0%] 43 [||||| 0.0%] 59 [||||| 0.0%]
12 [||||| 0.0%] 28 [||||| 0.0%] 44 [||||| 0.0%] 60 [||||| 0.0%]
13 [||||| 0.0%] 29 [||||| 2.0%] 45 [||||| 0.0%] 61 [||||| 0.0%]
14 [||||| 0.0%] 30 [||||| 0.0%] 46 [||||| 0.0%] 62 [||||| 0.0%]
15 [||||| 76.5%] 31 [||||| 100.0%] 47 [||||| 0.0%] 63 [||||| 0.0%]
16 [||||| 0.0%] 32 [||||| 0.0%] 48 [||||| 0.0%] 64 [||||| 0.0%]
Mem[||||| 5.076/2526] Tasks: 179, 666 thr; 6 running
Swp[||||| 2.08M/2.08G] Load average: 2.14 1.32 1.10
```

Project Design - Frontend

- Lambda
 - Attack code
 - “Customer” functions
- Data collection and transfer
- Graphing - troubleshooting
 - Helped identify viable data to be used in a machine learning model
- Issues faced
 - Multiple languages with different timing mechanisms
 - Customer cache footprinting
 - Running Lambda functions on the same cores/CPU



Running Attack Code on Server Directly

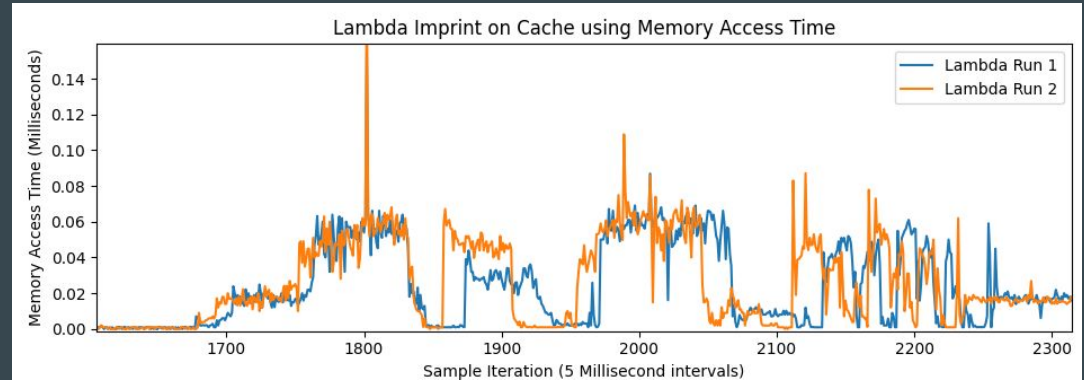
Results - Testbench

Initial Testing Phase

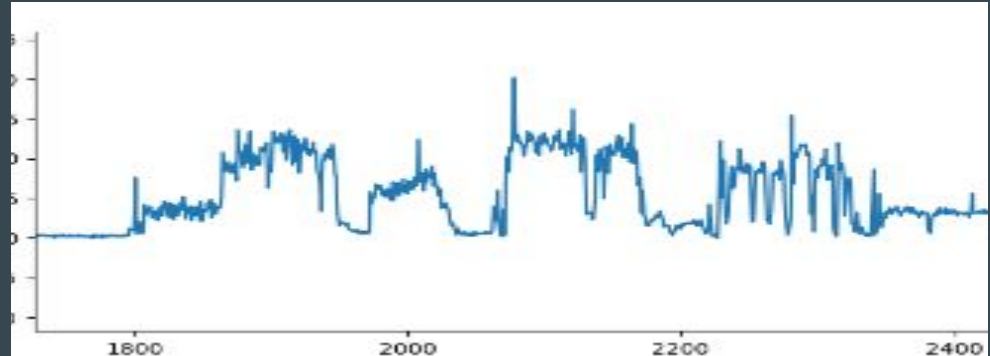
- Attack code viability
- Useful imprint on cache
- Repeatable

Attack Code as Lambda

- Similar results
- Still using testbench directly



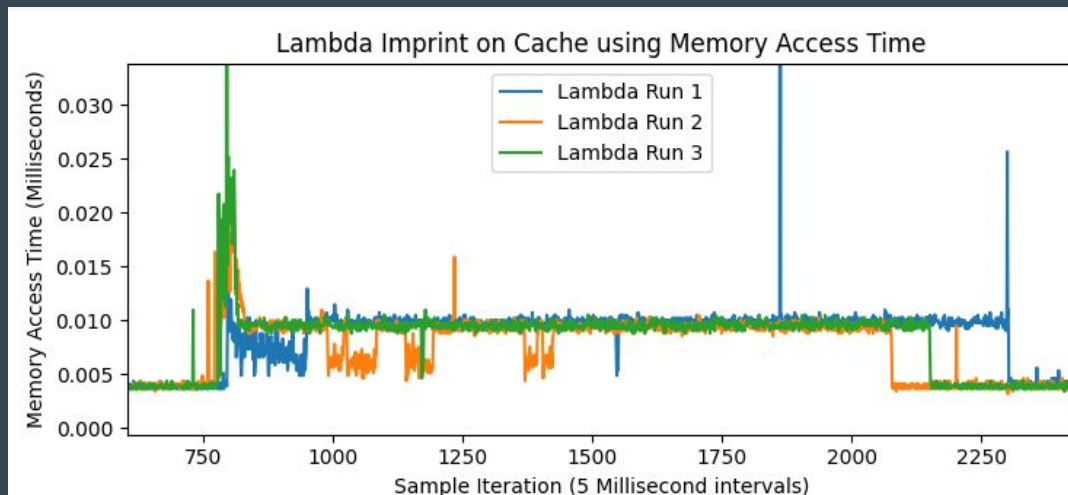
Running Attack Code as a Lambda



Results - Webserver

Attack Code and a Lambda

- Balance between number of cores and imprint
- Finding similarities
- Finding viable candidates for lambda
- Identifiable?
- Clear Start and Stop?
- Same Scale?



Subsequent Actions

- Train and test a ML model for classification of functions
- Testing attack code in real environment (AWS)
- Investigation into minimal viable capture amount
 - How many cores produce viable data?
- Expand interpreted languages used

Conclusion / Demo