Trustworthy & Accountable Function-as-a-Service

N. Asokan

https://asokan.org/asokan/

@nasokan

Joint work with Fritz Alder, Arseny Kurnikov, Andrew Paverd, Michael Steiner
Function-as-a-Service (FaaS)

Recent instantiation of “serverless computing”
- Customer specifies the function
- Service provider manages runtime, scaling, load-balancing etc.

Differences to Infrastructure-as-a-Service (IaaS)
- Relatively short-running function invocations
- Stateless functions (storage provided by separate service)
Motivation

FaaS is available from established cloud providers

Usual security concerns of cloud computing still apply:
• Confidentiality of data
• Integrity of computation
Motivation

https://www.theregister.co.uk/2018/07/24/apache_ibm_cloud_vulnerable/
Motivation

FaaS is available from established cloud providers

Usual security concerns of cloud computing still apply:
• Confidentiality of data
• Integrity of computation

More accurate resource usage measurements required:
• Sub-second compute time measurements

Currently achieved via existing reputational trust, but can we do better?
Motivation

FaaS can also be provided by non-traditional service providers
• Data centres with spare capacity
• Individuals with powerful PCs (e.g. gamers)

Open source frameworks available

Multiple start-ups in this space

https://golem.network/

https://openwhisk.apache.org/

https://ankr.com
Motivation

FaaS can also be provided by non-traditional service providers
• Data centres with spare capacity
• Individuals with powerful PCs (e.g. gamers)

Heightened security concerns:
• Service provider identity/location may be unknown
• Service provider may not have security expertise

Very few disincentives for cheating:
• Malicious service provider might inflate resource usage measurements

No reputational trust has been established
System Model & Requirements
System model

1. Provision function
2. Inputs
3. Outputs
4. Resource measurements
Adversary model

Two types of adversaries:

Service provider
- Learn inputs and outputs of function invocations
- Modify inputs and outputs, or execute the function incorrectly
- Overcharge the function provider
  - Falsely inflate resource usage measurements
  - Create fake function invocations

Function provider
- Under-pay the service provider for resources used by the function
Requirements

R1 - Security
• Service provider cannot modify inputs or outputs of a function invocation
• Client assured that output is result of correct execution of intended function on supplied inputs

R2 - Privacy
• Service provider cannot learn inputs or outputs of a function invocation

R3 - Measurement accuracy
• Resource measurements must have sufficient accuracy for FaaS billing

R4 - Measurement veracity
• All parties must be able to verify authenticity of resource measurements
Trusted Execution Environments

Hardware support for
- Isolated execution: Isolated Execution Environment
- Protected storage: Sealing
- Ability to convince remote verifiers: (Remote) Attestation

Trusted Execution Environments (TEEs)
Operating in parallel with “rich execution environments” (REEs)
Hardware-assisted TEEs are pervasive

Hardware support for
- Isolated execution: Isolated Execution Environment
- Protected storage: Sealing
- Ability to convince remote verifiers: (Remote) Attestation

Trusted Execution Environments (TEEs)
Operating in parallel with “rich execution environments” (REEs)

Cryptocards
Trusted Platform Modules
ARM TrustZone
Intel Software Guard Extensions

https://www.ibm.com/security/cryptocards/
https://www.infineon.com/tpm
https://www.arm.com/products/security-on-arm/trustzone
https://software.intel.com/en-us/sgx

Background: Intel SGX

CPU enforced TEE (enclave)

Remote attestation

Secure memory
- Confidentiality
- Integrity

https://software.intel.com/sgx
Preliminary design

Execute each function in an SGX enclave

Use remote attestation to establish secure communication channels

Measure resource consumption from within the enclave
Design Challenges
Challenge: Sandboxing untrusted functions

Malicious function provider could attempt to reduce in-enclave measurements

- No protection from code in the same enclave
Challenge: Attesting worker enclaves

Default SGX remote attestation involves multiple message round-trips

• Overhead and latency for short-running functions is too high
• Must be repeated for each enclave
Challenge: Encrypting client input

Function invocation is a one-shot message, including (encrypted) input

- Client must encrypt input before knowing which enclave will run the function
- Cannot rely on service provider to distribute keys to worker enclaves
Challenge: Measuring time in enclaves

SGX enclave cannot reliably measure its own running time

- RDTSC value can be manipulated by VMM
- `sgx_get_trusted_time()` can be arbitrarily delayed
- Enclaves can be transparently interrupted (AEX) and resumed (ERESUME)
Challenge: Measuring time in enclaves

VERICOUNT:
call sgx_get_trusted_time() at ecall start & end

ecall_to_measure()
{
    t1 = sgx_get_trusted_time();
    // [function code]
    // ...
    t2 = sgx_get_trusted_time();
    time = t2 - t1;
}

Tople et al., “VeriCount: Verifiable Resource Accounting Using Hardware and Software Isolation”, ACNS 2018
S-FaaS Architecture
**Architecture overview**

**Worker enclave runs function within a sandbox**
- e.g. Ryoan
- sandboxing interpreters: e.g. for JavaScript

**Challenges**
- C1: Sandboxing
- C2: Attesting enclaves
- C3: Encrypting input
- C4: Measuring time

Architecture overview

Function provisioning

Key Distribution Enclave (KDE)

ka+ ko+ kr+
ka- ko- kr-

Service Provider

Worker Enclave

Sandbox

Function

Resource measurement mechanisms

Client

kc+, \{inputs, h(f), want_receipt, nonce\}_kac

kc+, \{outputs, nonce, [receipt(l,f,O)]_{ko-}\}_kac

[measurements, tag]_{kr-}

ka: enclave’s DH key
ko: output key
kr: resource reporting key
Transitive attestation

Clients and function providers attest worker enclaves indirectly

Client / Function provider
  atests
  distributes public keys

Key Distribution Enclave (KDE)
  atests
  distributes private keys

Worker Enclave

Transitive attestation with key agreement

Challenges
C1: Sandboxing
C2: Attesting enclaves
C3: Encrypting input
C4: Measuring time
Measuring Resource Usage in SGX
Motivation

FaaS is available from established cloud providers

<table>
<thead>
<tr>
<th>Service</th>
<th>Invocations</th>
<th>Time (GHz-s)</th>
<th>Memory (GB-s)</th>
<th>Network (GB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS Lambda</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Azure Functions</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>Google Cloud Functions</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>IBM Cloud functions</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

FaaS billing policies of established cloud providers ($X = explicit; O = implicit$)
## Types of measurements

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
<th>Units</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>Total compute time of the function</td>
<td>multiples of ( T )</td>
</tr>
<tr>
<td>( T )</td>
<td>Duration of each tick in CPU cycles</td>
<td>GHz-s</td>
</tr>
<tr>
<td>( m_{\text{int}} )</td>
<td>Time-integral of memory usage</td>
<td>GB-s</td>
</tr>
<tr>
<td>( m_{\text{max}} )</td>
<td>Maximum memory used by the function</td>
<td>GB</td>
</tr>
<tr>
<td>( \text{net} )</td>
<td>Total number of network bytes sent and received</td>
<td>GB</td>
</tr>
</tbody>
</table>
Measuring compute time

High level idea: two concurrent threads in the enclave (timer & worker)

Worker Enclave

**timer**

Timer thread running a calibrated timing loop

**worker**

Worker thread running the sandboxed function

worker ecall

ecall return
Measuring compute time

High level idea: two concurrent threads in the enclave (timer & worker)

Worker Enclave

How to ensure worker thread has started?

Timer thread running a calibrated timing loop

Worker thread running the sandboxed function

How to detect interrupts?

How to resume from interrupts?

worker ecall

ecall return
Intel SGX internals

Enclave data structures
TCS: Thread Control Structure
(C)SSA: (Current) Save State Area

CPU Registers
- RAX: 0xff...
- RBX: ...
- RSP: ...
- RIP: 0xff...

RIP: Instruction Pointer
RSP: Stack Pointer
Intel Transactional Synchronization Extensions (TSX)

Special instructions enabling Hardware Lock Elision (HLE)

Read set
- Memory addresses read by the transaction (added upon access)
- Transaction will abort if address is concurrently written

Write set
- Memory addresses written by the transaction
- Transaction will abort if address is concurrently read

Roll-back
- All operations since the beginning of the transaction are reverted
Starting a function

1. Acquire mutex
2. Wait on worker
3. Set SSA marker
4. Notify timer, \( \text{processing} := \text{true} \)
5. Start TSX txn
5. Run function
while(processing == true) {
    XBEGIN    // begin TSX txn
    if(worker.ssa == marker)    // add worker.ssa to txn read set
    {
        for(i=0; i<LOOP_COUNT; i++) // LOOP_COUNT depends on Τ
            nop;
        t_internal++;
    }
    XEND     // end TSX txn
    t_external = t_internal    // update external counter
}
Worker thread interrupted

Worker Enclave

1. CPU save registers in SSA

2. Abort TSX txn

3. Modify saved RIP to custom handler

SSA stack

<table>
<thead>
<tr>
<th>Regs</th>
<th>0x00…</th>
</tr>
</thead>
<tbody>
<tr>
<td>RIP</td>
<td>0x89…</td>
</tr>
</tbody>
</table>
Worker thread resumed

1. CPU save registers in SSA
2. Abort TSX txn
3. Modify saved RIP to custom handler
4. Custom ERESUME handler restores SSA marker
5. Start TSX txn

Worker Enclave

timer

worker

AEX

ERESUME

SSA stack
 Marker 0x12..
Custom ERESUME handler

```assembly
.text
.globl custom_eresume_handler
.type custom_eresume_handler,@function
custom_eresume_handler:
push %rax  # Save registers
push %rbx
lea g_worker_ssa_gpr(%rip),%rax  # Load pointer
mov (%rax),%rbx  # Dereference pointer
movl $12345,(%rbx)  # Write SSA marker value
pop %rbx  # Restore registers
pop %rax
jmp *g_original_ssa_rip(%rip)  # Resume execution
```
Completing a function

1. Function completes
2. processing := false
3. Stop timing
4. Read time
5. Return outputs and resource measurements
Measuring Memory and Networking

Memory
• Instrumented allocators used by interpreter
• Measurements updated on every allocation/free

<table>
<thead>
<tr>
<th>$m_{\text{int}}$</th>
<th>Time-integral of memory usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>$m_{\text{max}}$</td>
<td>Maximum memory used by the function</td>
</tr>
</tbody>
</table>

Network
• Payloads measured inside enclave
Integration with OpenWhisk
Integration with OpenWhisk

Proof-of-concept using Duktape JavaScript interpreter in worker enclave

S-FaaS Enclave Service

Worker enclaves

Key distribution enclave(s)

S-FaaS Docker containers

Docker containers

NGINX

Controller

kafka

CouchDB

https://openwhisk.apache.org/documentation.html
Evaluation
Evaluation: Accuracy

Synthetic function with well-defined compute and memory requirements

- fibonacci(k) calculates the first k numbers in the Fibonacci sequence

Compute time

- Expected to be linear in k
- Can be compared with measurement outside the enclave

Memory time-integral

- Expected to be quadratic in k (k-element list pre-allocated at start of function)
- Harder to measure outside enclave
Evaluation: Accuracy
Evaluation: Accuracy

- Memory-time integral ($\tau=630$)
- Memory-time integral ($\tau=6100$)
- Memory-time integral ($\tau=62000$)
- Quadratic fit

Graph showing the relationship between Kilobyte-seconds and Input parameter ($k$) with different lines representing different integral values and a quadratic fit.
Evaluation: Performance

Pre-function latency
• Measure **cold-start** and **warm-start** latency
• Tested using an empty function to isolate pre-function latency
• **Baseline**: equivalent operation (same interpreter) without SGX

Resource measurement overhead
• Measure **overhead** of S-FaaS resource measurement mechanisms
• **Octane JavaScript benchmarks** (excluding graphical tests)
• **Baseline**: equivalent operation without resource measurement

Benchmark environment
• Core i5-6500, 8GB RAM, Ubuntu 16.04, Intel SGX SDK 2.2.1
Evaluation: Pre-function latency

Cold-start
1. Create Docker container
2. Create enclave
3. Provision function
4. Perform key-agreement
5. Return empty response

Baseline: 3179 ms (σ = 40 ms)
S-FaaS: 3249 ms (σ = 38 ms)
Latency increase: ~2%

Warm-start
1. Create Docker container
2. Create enclave
3. Provision function
4. Perform key-agreement
5. Return empty response

Baseline: 204 ms (σ = 106 ms)
S-FaaS: 210 ms (σ = 149 ms)
Latency increase: ~3%
# Evaluation: Resource measurement overhead

<table>
<thead>
<tr>
<th>Function</th>
<th>Baseline</th>
<th>No encryption</th>
<th>Encryption</th>
<th>Encryption &amp; receipt</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box2D</td>
<td>3.019</td>
<td>3.118</td>
<td>3.121</td>
<td>3.135</td>
</tr>
<tr>
<td>DeltaBlue</td>
<td>1.446</td>
<td>1.524</td>
<td>1.529</td>
<td>1.537</td>
</tr>
<tr>
<td>NavierStokes</td>
<td>4.155</td>
<td>4.418</td>
<td>4.447</td>
<td>4.473</td>
</tr>
<tr>
<td>RayTrace</td>
<td>0.779</td>
<td>0.848</td>
<td>0.850</td>
<td>0.852</td>
</tr>
<tr>
<td>Richards</td>
<td>1.719</td>
<td>1.767</td>
<td>1.767</td>
<td>1.799</td>
</tr>
<tr>
<td><strong>Overall</strong></td>
<td>-</td>
<td><strong>5.3%</strong></td>
<td><strong>5.6%</strong></td>
<td><strong>6.3%</strong></td>
</tr>
</tbody>
</table>
Trade-offs and limitations

Need for an additional thread
• State-of-the-art SGX side-channel defences\(^*\) require control of both sibling hyperthreads

Timing granularity
• Choice of T affects extent of under- or over-reporting
• S-FaaS service providers can specify T for each function

Architecture-specific calibration
• Timing loop must be calibrated for different CPU architectures

\(^*\) SGX side-channel defenses:
Cloak: Gruss et al., “Strong and Efficient Cache Side-Channel Protection using Hardware Transactional Memory”, Usenix SEC 2017
HyperRace: Chen et al., “Racing in Hyperspace: Closing Hyper-Threaded Side Channels on SGX with Contrived Data Races”, IEEE S&P 2018
Varys: Oleksenko et al., “Varys: Protecting SGX enclaves from practical side-channel attacks”, Usenix ATC 2018
Suggested SGX enhancements

Secure tick counter
• Provide a trustworthy tick counter that can be accessed without leaving the enclave

Custom ERESUME handlers
• Allow enclaves to specify an in-enclave handler to be called on each ERESUME
• Could also be used to detect frequent AEX events indicative of side-channel attacks
Integration with distributed systems

Smart contracts to pay for outsourced computation
• S-FaaS function receipts and resource measurements can be verified in smart contracts
• Straight-forward integration with payment networks
  - Particularly beneficial to non-traditional service providers

Leader election based on useful work
• Similar to Resource-Efficient Mining for Blockchains (Zhang et al.)
• Uses “useful computation” to determine who mines next block

Zhang et al., “REM: Resource-Efficient Mining for Blockchains”, Usenix SEC 2017
Deployment considerations

Incremental deployment
• Initially, S-FaaS requires no changes on client-side (no client attestation or encryption)
• Clients can individually start to verify attestation and/or encrypt inputs

Implementations with other TEEs
• S-FaaS could be ported to e.g. ARM TrustZone
• TrustZone secure world still requires functions to run in a suitable sandbox, but timing would be simpler because secure world cannot be arbitrarily paused
Conclusions

FaaS increasingly popular with cloud providers and non-traditional service providers
• Requires strong security: data confidentiality and integrity of computation
• Requires accurate and trustworthy resource consumption measurement

S-FaaS demonstrates how to secure current FaaS architectures using Intel SGX
• ACM CCS Cloud Computing Workshop 2019 https://ccsw.io/

Code available on GitHub
https://github.com/SSGAalto/sfaas
https://asokan.org/asokan/research/
What if SGX is broken?

Back to current state of FaaS security and resource measurement

- TEEs useful in two kinds of settings:
  1. improving security
  2. improving other attributes while preserving security

S-FaaS is Type 1. TEE compromise is a bigger concern in Type 2

- Application-specific ways of detecting / mitigating effects of TEE compromise, e.g.,
  - post-mortem auditing of signed receipts
  - statistical mechanisms like in PoET and Zhang et. al.