AccTEE: A WebAssembly-based Two-way Sandbox for Trusted Resource Accounting
MIDDLEWARE 2019, UC Davis

David Goltzsche,¹ Manuel Nieke,¹ Thomas Knauth,² and Rüdiger Kapitza¹

goltzsche@ibr.cs.tu-bs.de
@d_goltzsche

¹TU Braunschweig, Germany
²Intel, United States
Offloading Computations

- Offloading computations to remote infrastructure
  - Cloud Computing
  - Volunteer Computing
  - Client-side Web applications

- Reasons:
  - Remotely available resources
  - Moving computations closer to customers

- Usually two entities:
  - Workload provider
  - Infrastructure provider
Offloading Computations

- Offloading computations to remote infrastructure
  - Cloud Computing
  - Volunteer Computing
  - Client-side Web applications

- Reasons:
  - Remotely available resources
  - Moving computations closer to customers

- Usually two entities:
  - Workload provider
  - Infrastructure provider
### Offloading Computations

- Offloading computations to remote infrastructure
  - Cloud Computing
  - Volunteer Computing
  - Client-side Web applications

- Reasons:
  - Remotely available resources
  - Moving computations closer to customers

- Usually two entities:
  - **Workload provider**
  - **Infrastructure provider**

---

Accounting of consumed resources in some cases
Resource Accounting

- Cloud Computing
  - CPU and memory usage, I/O operations

- Volunteer Computing
  - Logging of donated CPU time

- Client-side Web applications
  - No accounting in practice
Resource Accounting in Practice

- Accounting on different levels
  - Task level (e.g. for completed tasks)
  - Hardware level (e.g. CPU usage)

- Resources always accounted by infrastructure provider
Resource Accounting in Practice

- Accounting on different levels
  - Task level (e.g. for completed tasks)
  - Hardware level (e.g. CPU usage)

- Resources always accounted by infrastructure provider

Current approaches of resource accounting **require trust** in the infrastructure provider
Trust Relationship

- Malicious infrastructure provider can ...
  - Spy on provided code or data
  - Fake accounting results (overbilling)

- Malicious workload provider can ...
  - Provide crafted workload to destroy execution environment
  - Trick resource accounting
Trust Relationship

- Malicious infrastructure provider can ...
  - Spy on provided code or data
  - Fake accounting results (overbilling)

- Malicious workload provider can ...
  - Provide crafted workload to destroy execution environment
  - Trick resource accounting

**Problem:** Limited trust between infrastructure and workload provider
Outline

- Design of AccTEE
- Evaluation of AccTEE
- Related Work
- Conclusion
1. Workload provider provides workload
2. Infrastructure provider executes workload in sandbox
3. Sandbox produces mutually trusted resource usage log
1. Workload provider provides workload
2. Infrastructure provider executes workload in sandbox
3. Sandbox produces mutually trusted resource usage log
Approach of AccTEE

1. Workload provider provides workload
2. Infrastructure provider executes workload in sandbox
3. Sandbox produces mutually trusted resource usage log
Approach of AccTEE

1. Workload provider provides workload
2. Infrastructure provider executes workload in sandbox
3. Sandbox produces mutually trusted resource usage log
Approach of AccTEE

1. Workload provider provides workload
2. Infrastructure provider executes workload in sandbox
3. Sandbox produces mutually trusted resource usage log

How do we get an sandbox with mutually trusted resource accounting?
AccTEE’s Sandbox

- Accountable sandbox is a combination of two sandboxes

  - **Execution sandbox**
    - Shields host from workload
    - Shields accounting from workload

  - **Accounting sandbox**
    - Shields workload from host
    - Shields accounting from host
Background: WebAssembly (WASM)

- A platform independent binary instruction format
- Initially designed for computations in browsers
  - Standalone execution emerging
- **Goal:** a safe, fast and portable low-level code
- Application code is **compiled to WASM**
- WebAssembly code executed in sandboxes
  - Based on software fault isolation
Background: Intel SGX

- **x86 instruction set extension**

- Creation of trusted execution environments (TEEs) → **enclaves**

- Execution and data inside enclaves **protected** from privileged software

- Hardware-based **memory integrity protection and encryption**

- Only CPU is trusted

- **Remote attestation** of enclaves

- Limitation: enclave page cache (EPC) size
AccTEE’s Two-way Sandbox

- AccTEE combines two sandboxes
  - **Execution sandbox**
    - Based on **WebAssembly**
  - **Accounting sandbox**
    - Based on **Intel SGX**
    - **Code instrumentation** for resource accounting
AccTEE’s Two-way Sandbox

- AccTEE combines two sandboxes
  - **Execution sandbox**
    - Based on WebAssembly
  - **Accounting sandbox**
    - Based on Intel SGX
    - **Code instrumentation** for resource accounting

AccTEE combines SGX and WebAssembly to create a **two-way sandbox**
AccTEE’s Two-way Sandbox

- AccTEE combines two sandboxes
  - **Execution sandbox**
    - Based on **WebAssembly**
  - **Accounting sandbox**
    - Based on **Intel SGX**
    - **Code instrumentation** for resource accounting

AccTEE combines SGX and WebAssembly to create a two-way sandbox

AccTEE **instruments WebAssembly code** for mutually trusted resource accounting
WebAssembly Code Instrumentation

Goal: Count WebAssembly instructions

- naive instrumentation
  - Based on **basic blocks**
  - Counter incremented at end of block

- flow-based optimization
  - Increment by minimum instruction count
  - Update counter based on **control flow**

- loop-based optimization
  - Identify **loop iterators** with constant increments
  - Increment counter **once** after loop

- Different instruction costs
  - AccTEE uses a **weighted instruction counter**
WebAssembly Code Instrumentation

**Goal:** Count WebAssembly instructions

- **naive instrumentation**
  - Based on **basic blocks**
  - Counter incremented at end of block

- **flow-based optimization**
  - Increment by minimum instruction count
  - Update counter based on **control flow**

- **loop-based optimization**
  - Identify **loop iterators** with constant increments
  - Increment counter **once** after loop

- **Different instruction costs**
  - AccTEE uses a **weighted instruction counter**

```wasm
get_global 12
set_local 3
i32.lt_s
Increment counter by 3>
if (result i32)

global 0
i32.load offset=4
Increment counter by 2>
elset
get_local 4
i32.const 255
i32.and
Increment counter by 3>
end

Increment counter by 3>
tee_local 4
get_local 1
Increment counter by 2>
```

D. Goltzsche, TU Braunschweig, Germany | AccTEE, D. Goltzsche et al. | Page 12
WebAssembly Code Instrumentation

**Goal:** Count WebAssembly instructions

- **naive instrumentation**
  - Based on *basic blocks*
  - Counter incremented at end of block

- **flow-based optimization**
  - Increment by minimum instruction count
  - Update counter based on *control flow*

- **loop-based optimization**
  - Identify *loop iterators* with constant increments
  - Increment counter once after loop

- Different instruction costs
  - AccTEE uses a *weighted instruction counter*

```wasm
get_global 12
set_local 3
i32.lt_s

if (result i32)
    get_local 0
    i32.load offset=4
else
    get_local 4
    i32.const 255
    i32.and
<Increment counter by 1>
end

tee_local 4
get_local 1
<Increment counter by 7>
```
**AccTEE’s Workflow**

1. Workload provider compiles application to WebAssembly
2. WebAssembly is instrumented inside Instrumentation Enclave
   - Instrumentation evidence
   - Instrumented WebAssembly code
3. Accounting Enclave verifies evidence and executes WebAssembly code
4. Result: mutually trusted resource usage log
Example Use Cases

- **Function-as-a-Service**
  - **Trusted** resource accounting in data centers

- **Volunteer Computing**
  - **Trusted** resource accounting at clients

- **Client-side web applications**
  - **Trusted** resource accounting in browsers
  - e.g. for replacing micro payments
PolyBench/C Benchmark Suite

- **Overhead for WASM:** 10%
- **Overhead for WASM-SGX:** $2.1 \times$ (EPC exhaustion)
- **Instrumentation overhead over WASM-SGX:** 4% on average
WebAssembly Instruction Weights

- 74% of instructions need < 10 cycles
- 2% of instructions (e.g. f32.sqrt) > 50 cycles
74% of instructions need < 10 cycles
2% of instructions (e.g. f32.sqrt) > 50 cycles
WebAssembly Instruction Weights

- 74% of instructions need < 10 cycles
- 2% of instructions (e.g. f32.sqrt) > 50 cycles
Function-as-a-Service (FaaS) Use Case

- **Benchmark:** Image resize FaaS function
- **Accounting overhead is negligible**
- **Between 3× and 9× faster than JavaScript baseline**
Related Work

- Combination of Google Native Client (NaCl) and SGX enclaves
  - **MiniBox** (ATC’14), **Ryoan** (OSDI’16)
  - No platform independence
  - No resource accounting

- **S-FaaS** (CCSW’19) Trustworthy and Accountable FaaS
  - Combines SGX and hyper-threading
  - CPU time measured by dedicated timer thread
  - Wastes an entire core to count CPU cycles
Related Work

- **Combination of Google Native Client (NaCl) and SGX enclaves**
  - **MiniBox** (ATC’14), **Ryoan** (OSDI’16)
  - No platform independence
  - No resource accounting

- **S-FaaS** (CCSW’19) Trustworthy and Accountable FaaS
  - Combines SGX and hyper-threading
  - CPU time measured by dedicated timer thread
  - Wastes an entire core to count CPU cycles

AccTEE is the **first two-way sandbox** based on SGX and WebAssembly enabling **mutually trusted resource accounting**
Conclusion

AccTEE’s contributions:

- Implements **two-way sandbox**
- Mutually trusted resource accounting
  - **Instrumentation** of WebAssembly code
  - Platform independent
- More contributions in the paper
  - Volunteer Computing use case
  - Accounting of I/O and memory usage
Conclusion

AccTEE’s contributions:

- Implements **two-way sandbox**
- Mutually trusted resource accounting
  - **Instrumentation** of WebAssembly code
  - Platform independent
- More contributions in the paper
  - Volunteer Computing use case
  - Accounting of I/O and memory usage

Thank you for your time! Questions?
goltzsche@ibr.cs.tu-bs.de
@d_goltzsche
Side-channel Attacks against Intel SGX Enclaves

- Side-channel attacks against SGX:
  - Spectre Attacks: Exploiting Speculative Execution (S&P’19)
  - Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution (USENIX Security’18)
  - ZombieLoad: Cross-Privilege-Boundary Data Sampling (2019)

- All side-channels are not exclusive to SGX!
- All fixed by microcode updates at cost of transition performance