Fast, Flexible and Secure Onloading of Edge Functions using AirBox
Offloading vs Onloading

● Backend driven Cyber foraging:
  – Remote cloud onloads appropriate benefits near end-users

● Compare to Client driven offloading
  – Diverse device space (onchanging mobile platforms)
  – Accurate code profiling needed at mobile to analyze load conditions
  – Continuous monitoring
Offloading vs Onloading

- Inherently handle diversity (server works irrespective of what is the client mobile!)
- Unlimited computational capability (cloud)
- Helps consolidate bandwidth usage by employing ‘service-specific’ logic → reduce traffic over wire
- Best example use-cases: filtering, compression, caching, aggregation etc.
Major contributions

- Design space exploration (extensive!)
- Edge cloud security addressed in detail
  - Hardware level security
- AirBox platform
  - AB console @backend cloud
  - AB Provisioner @Edge cloud
    - 10x faster (compared to JIT cloudlet paper)
    - Scales well with negligible overhead
    - Integrity and security (thanks to SGX)
Nomenclature

- **Edge Functions (EF)**: Onloaded Services running on edge infrastructure
- **Backend service**: `@remote`
- **Infrastructure**: Edge Function Platform (EFP): platform and stack at the edge cloud
Design space considerations – Full Posix support

- VM → cloudlets
  - Disadv: Large size; backend does not directly talk to edge server. adv: but no developer constraints

- OS level containers → Docker
  - Disadv: Underlying OS restrictions; Adv: fast, lightweight

- Sandbox based → Embassies
  - Specific tool chain (limit porting existing applications); adv: no OS constraints
Developer constraints

- VM: No developer constraints.
  - Choose OS, library and application packages as VM and ship it.

- Containers: ‘WIN’ straightforward and ready
  - OS bound (eg. Docker – linux, SGX – Windows (currently)); but claimed its not a significant problem

- Application Sandboxes: learning curve
  eg. Embassies: pico-process libOs + modified libC + elf loader etc.
Security and Privacy

- Integrity verification
- Execution security
- Data Confidentiality

EF cannot trust EFP or privileged system software for security (host OS/ hypervisor)
  - Iago attacks by malicious kernel
EFP requirement

- Fast provisioning (backend load EF quickly)
- Scalability for multiple simultaneous requests
- Security: service specific logic should not be leaked
- Less developer constraints
  common platforms (unlike NaCl etc.)
AirBox

- Model: Docker + Intel’s SGX

- Why? Gives performant EF provisioning: fast and scalable, but needs additional mechanism for integrity verification

- Cannot rely on system components; Intel’s SGX provides processor built-in features for security.
AirBox Design

- Docker + Intel SGX
  - Kernel namespace → application isolation
  - cgroups → tighter hardware control
  - Union FS (layered FS) → lightweight and fast

- SGX (Software Guard Extensions)
  - Secure ISA extension: enclaves – protected exec environment
  - Remote attestation: verify integrity of binary
AirBox Console

- Web based management system
  - Deploy EFs dynamically at remote sites
  - Invokes Docker mechanism
  - Uses Docker registry service
    - pull binary from remote cloud for provisioning
    - Back-end has a EF repository
AirBox provisioner (EFP)

- Calls docker daemon for handling EF creation
- Checks its own integrity with SGX ‘remote attestation’
- EF can relate tolerate temporary unavailability
  - If Edge cloud fails, the client will fall back dealing with remote server directly
- Addresses only a system focused threat model (Iana attacks)?
- Even if EFP is compromised, EF is unexposed
  - Cannot take out subset of instruction set from professor
AirBox Framework overview

● Trusted + untrusted code
● Limit what’s within the enclave (trusted code)
  – To reduce performance impact due to additional SGX processing
● Comments: still scope for breach from system software. [eg., OS handle log requests or packets at incoming port]
AirBox secure Interface

- Remote attestation
  - To verify integrity of the application binary

- Remote authentication
  - To fetch private key of remote (for TLS auth)

- Sealed storage
  - Securely read/write on an insecure disk encrypted using a sealing key

- AirBox provides a wrapper API for SGX functions
EF state confidentiality

- OpenSGX uses a I/O trampoline based approach for communication between Host and Enclave
  - Shared memory allocated between EF’s trusted part and untrusted storage: 5MB limit?
  - For larger memory requirement, slice and reassemble → can have perf. implications
- encrypt/decrypt AES routines (Noise-C)
- 2 phase process: session and sealing keys
Evaluation

- **OpenSGX**: QEMU emulator based

- Overhead of AirBox framework comparing with Docker:
  - SGX attestation overhead
  - Design anatomy overhead
  - Workload overhead

- Attestation overhead reported to be in micro-seconds

- `Memcpy` found to be costly in CPU cycles (b/w Host and enclave)
Discussions

- Restricts to only Intel architecture
- Docker – popular linux only – restricts linux Apps and Windows apps to live in same Edge loud environment
- Container for EFs antivirus, firewall, load balancing etc. difficult
- DPDK extensions (faster networking) impossible
- didn’t talk about migrating containers?
Discussions

● Many other security concerns like DOS are not addressed.
● Multi-tenant system? If we want separate networking for each App, requires external scripting/modules
● 5MB limitation for host and enclave communication
● Limited functions : Compression, filtering etc.