Fast Enclave Launch: Efficient Enclave Launching for Secure Serverless Cloud Computing

Sadman Sakib Akash, Naveed Ul Mustafa, and Yan Solihin

University of Central Florida, Orlando, FL

Introduction

Serverless Computing:

- Applications are decomposed into independent and stateless *functions* and *events* [1].
- Function execution is triggered by event(s), e.g., HTTP requests.
- Cloud provider manages the resources, without involvement of the application developer [2].
- What if client trusts only its own functions and nothing else (i.e., OS, hypervisor, other applications running on the cloud server)?

FaaS Needs Security Protection:

- Memory Integrity
- Memory encryption
- Replay attack

Enclave:

- A hardware enclave (e.g., intel SGX) allocates a set of physical addresses accessible only from the program.
- Enclave can be used to isolate a function's execution from system/application software.

Motivation

- Multiple but identical functions to serve multiple requests.
- One enclave per function for security protection.
- Function have typically short life cycle (<1s).
- A few seconds to create an enclave [3].
- Enclave initialization cost = Copy Latency + Measurement Latency.

Problem Statement

"Enable security protections for serverless computing with a lower enclave initialization cost"

Approach

Avoid the Measurement Latency:

- Create, attest and save a *template* enclave.
- Instantiate *child* enclaves by copying template into children.
- Attest once, use multiple times.

Lower the Copy Latency:

- Avoid page table walk, when possible, during copy operation.
- Copy on demand and at finer granularity.

Threat Model

- **Attacker:** Cloud Service Provider, Function Provider.
- **Goal of Attacker:** Break confidentiality or alter the state of the program
- **Attacker can** control privileged software, tamper in-memory values, and perform physical attack.
- **Attacker cannot** access on-chip caches or registers.

Enclave Memory Encryption

- Encrypt/Decrypt data blocks crossing secure chip-boundary.
- $Ciphertext = Plaintext \oplus OTP$
- $OTP = Block\_Cipher(Seed, SecretKey)$
- $Seed = PageAddress + BlockOffset + Counter + Padding$
- Block-level counter is incremented each time block is written to memory

Enclave Memory Integrity

- Data blocks are protected by their MAC.
- MAC computed over counter, physical address and data of a block.
- Bonsai Merkel Tree (BMT) built only over counters.
- Root hash is kept on-chip, protected from attackers

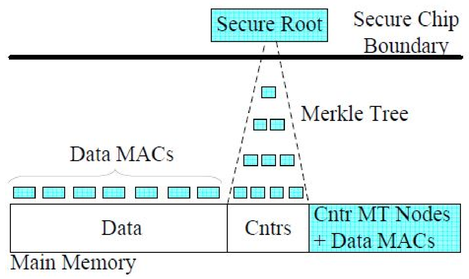

Figure 1. Bonsai Merkle Tree.[4]

Base Design

Full Enclave Copy Without Page Table

- Contiguous and memory resident *template* enclave.
- Decrypt all data blocks from template and copy them in *child* as *dirty*.
- Build child's page table during copy operation.
- Counters remain unchanged for the child.
- Compute BMT root for child and compare with that of the template.
- Child's MACs will be recomputed with new address (same counter)

Pros:
+ Avoid attestation
+ No Page Table Walk (PTW) over template

Cons:
- Template must be resident/contiguous
- High copy latency

EID	Root	Base	Range	Ready Execute	Pin	ParentID
124	x	B1	R1	1	1	
366	x	CR3		0	0	124

Table 1. Enclave Table for Base Design.

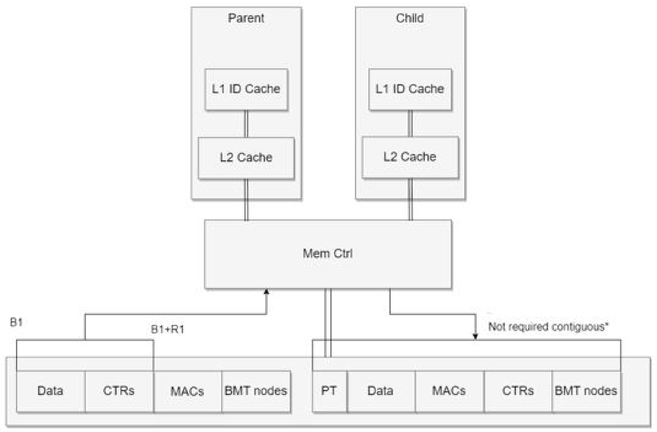

Figure 2. Block Diagram for Full Enclave Copy.

Full Copy with Page Table

- Walks template's page table for copying each page

Pros:
+ Template enclave does not need to be contiguous or memory resident

Cons:
- Copies everything, not selective
- High copy latency

Page Level Copy with Page Table

- Child copies only accessed-pages from template.
- Child's PT is updated.
- Child's MAC are computed using new address.
- Child's BMT is computed using split counters i.e., some in child (for copied pages), and some in template (for non-copied) pages.

Pros:
+ Fast enclave start
+ Memory efficient

Cons:
- Needs to track counter location.
- Re-encryption of full page could be slow

EID	Root	Ready Execute	Pin	Page Table
124	x	1	1	Ptr to CR3
366	x	0	0	||

Table 2. Enclave Table for Page-Level Copy

References

[1] Theo Lynn, Pierangelo Rosati, Arnaud Lejeune, and Vincent Emeakaroha. A preliminary review of enterprise serverless cloud computing (function-as-a-service) platforms. In *2017 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 162–169. IEEE, 2017.

[2] Mohammad Shahrad, Jonathan Balkind, and David Wentzlaff. Architectural implications of function-as-a-service computing. In *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, pages 1063–1075, 2019.

[3] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, Xueqiang Jiang, Yubin Xia, Binyu Zang, and Haibo Chen. Scalable memory protection in the {PENGLAI} enclave. In *15th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 21)*, pages 275–294, 2021.

[4] Brian Rogers, Siddhartha Chhabra, Milos Prvulovic, and Yan Solihin. Using address independent seed encryption and bonsai merkle trees to make secure processors os-and performance-friendly. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 183–196. IEEE, 2007.

[5] Dayeol Lee, Kevin Cheang, Alexander Thomas, Catherine Lu, Pranav Gaddamadugu, Anjo Vahldiek-Oberwagner, Mona Vij, Dawn Song, Sanjit A Seshia, and Krste Asanovic. Cerberus: A formal approach to secure and efficient enclave memory sharing. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 1871–1885, 2022.

Email: sa717931@ucf.edu