Boomerang: Exploiting the Semantic Gap in Trusted Execution Environments

Aravind Machiry, Eric Gustafson, Chad Spensky, Chris Salls, Nick Stephens, Ruoyu Wang, Antonio Bianchi, Yung Ryn Choe, Christopher Kruegel, and Giovanni Vigna

UCSB

Sandia National Laboratories

Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SANDNO. 2017-XXXXP

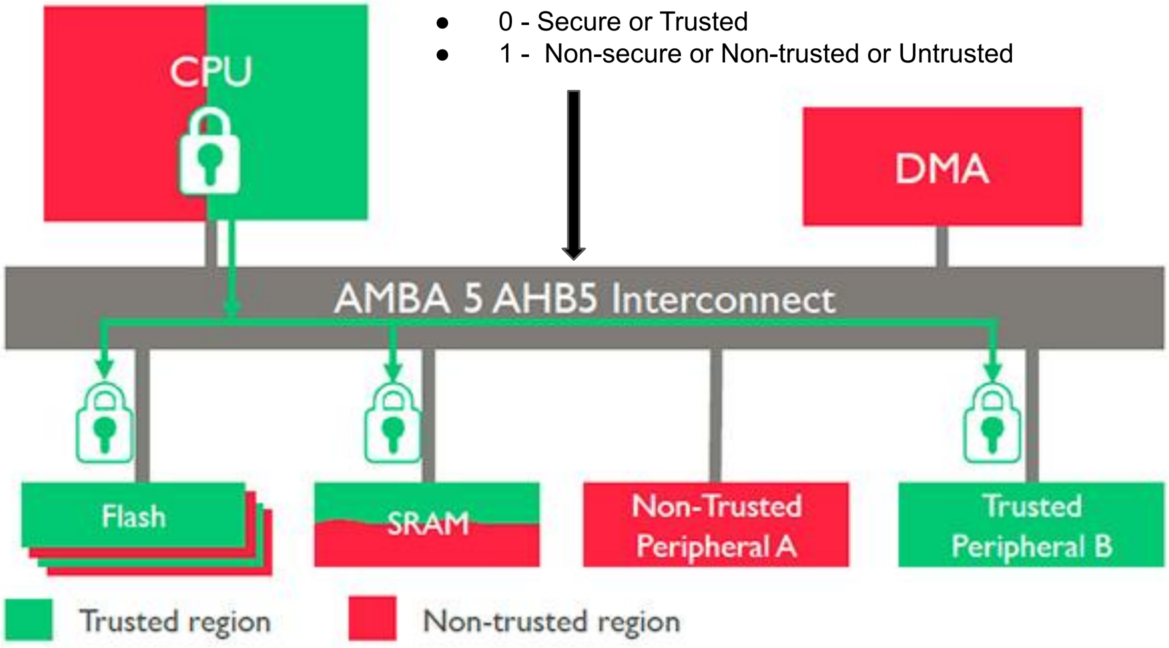
Trusted Execution Environment (TEE)

- Hardware-isolated execution environments (e.g., ARM TrustZone)
  - Non-secure world
    - Untrusted OS and untrusted applications (UAs) (e.g., Android and apps)
  - Secure world
    - Higher privilege, can access *everything*
    - Trusted OS and trusted applications (TAs).

ARM TrustZone

NS Bit

- 0 - Secure or Trusted
- 1 - Non-secure or Non-trusted or Untrusted



Picture reused from arm.com

Untrusted OS ↔ Trusted OS

- Untrusted applications (UAs) request trusted applications (TAs) to perform privileged tasks.

- TAs should verify the request and perform it only if the request is valid.
  - **Example:** Sign the contents of a memory region
    - TA should check if the **requested memory region belongs to untrusted OS** before computing the signature of it.

Untrusted OS ↔ Trusted OS

NETFLIX

Non-Secure World

Secure World

Untrusted Application (UA)

Trusted Application (TA)

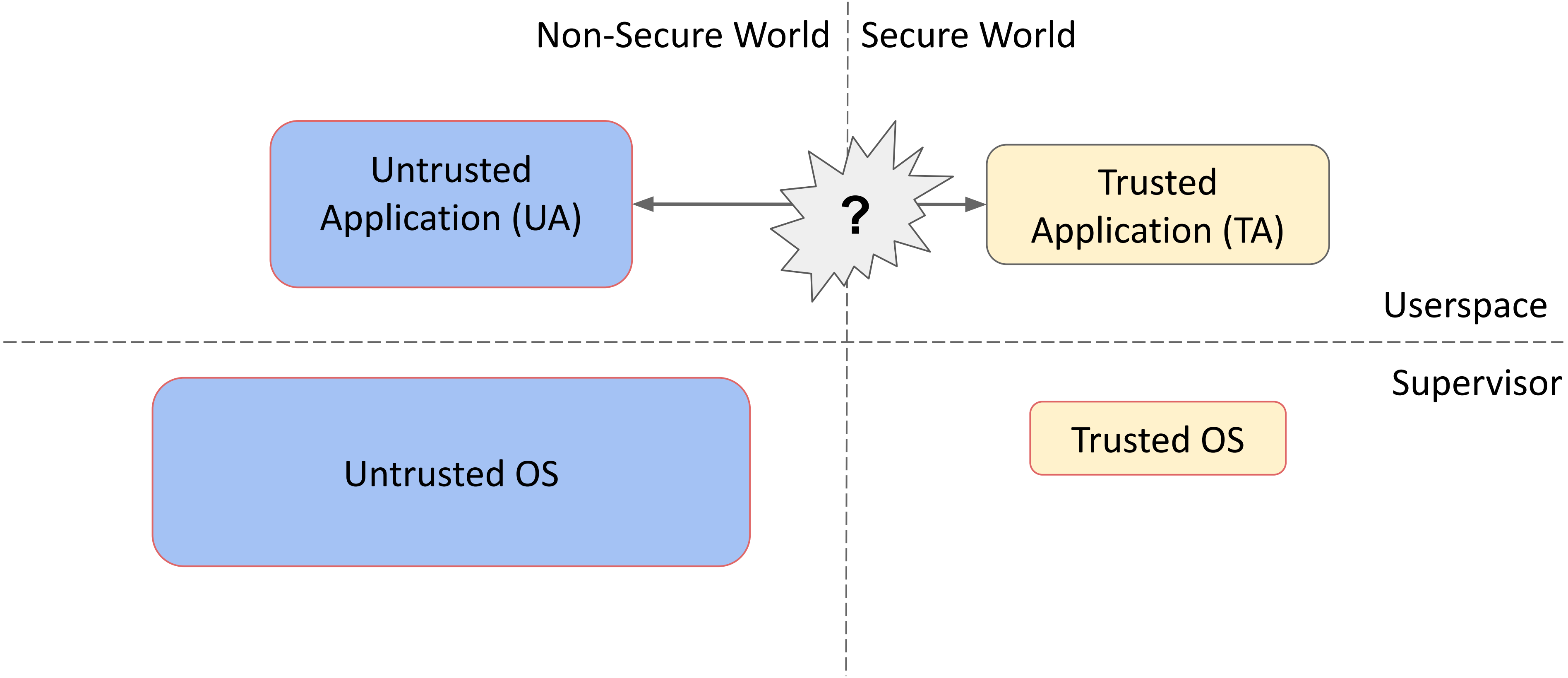
Userspace

Supervisor

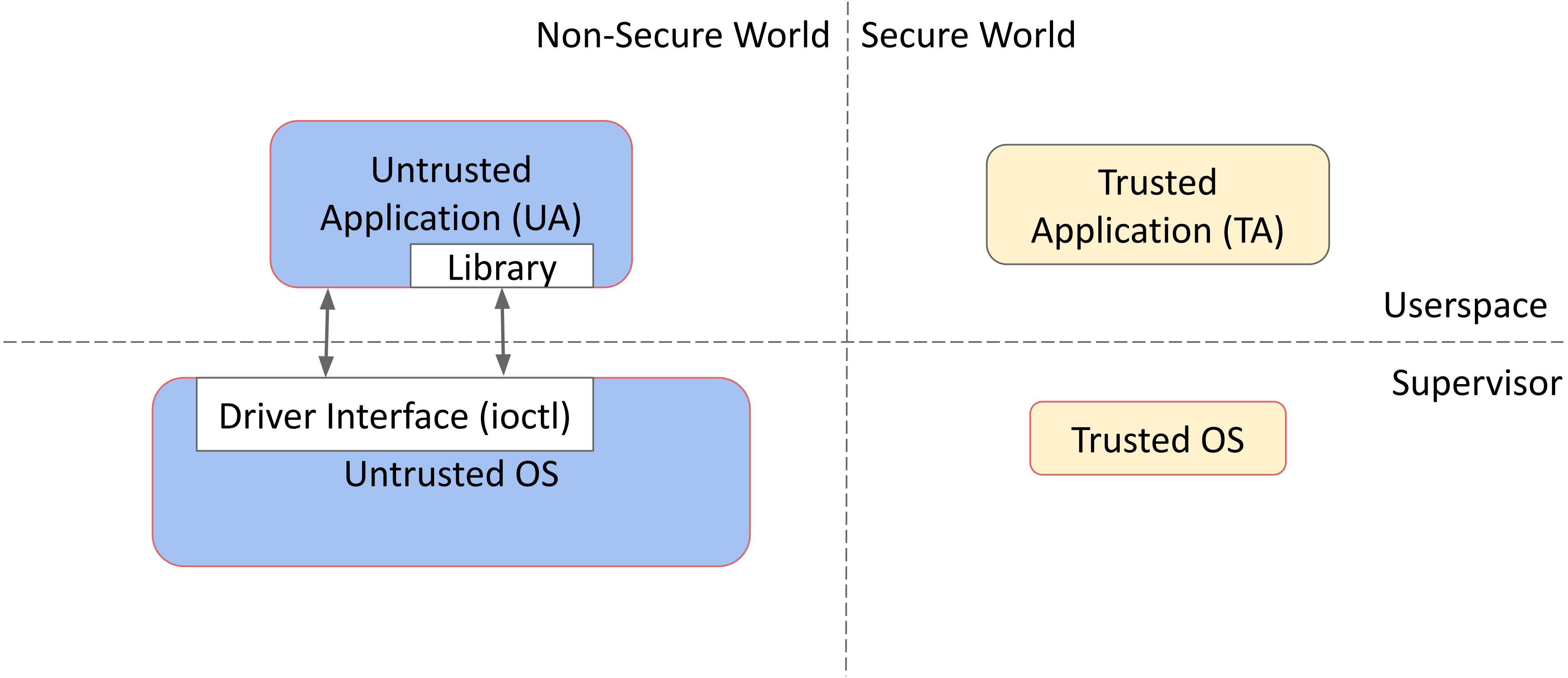
Untrusted OS

Trusted OS

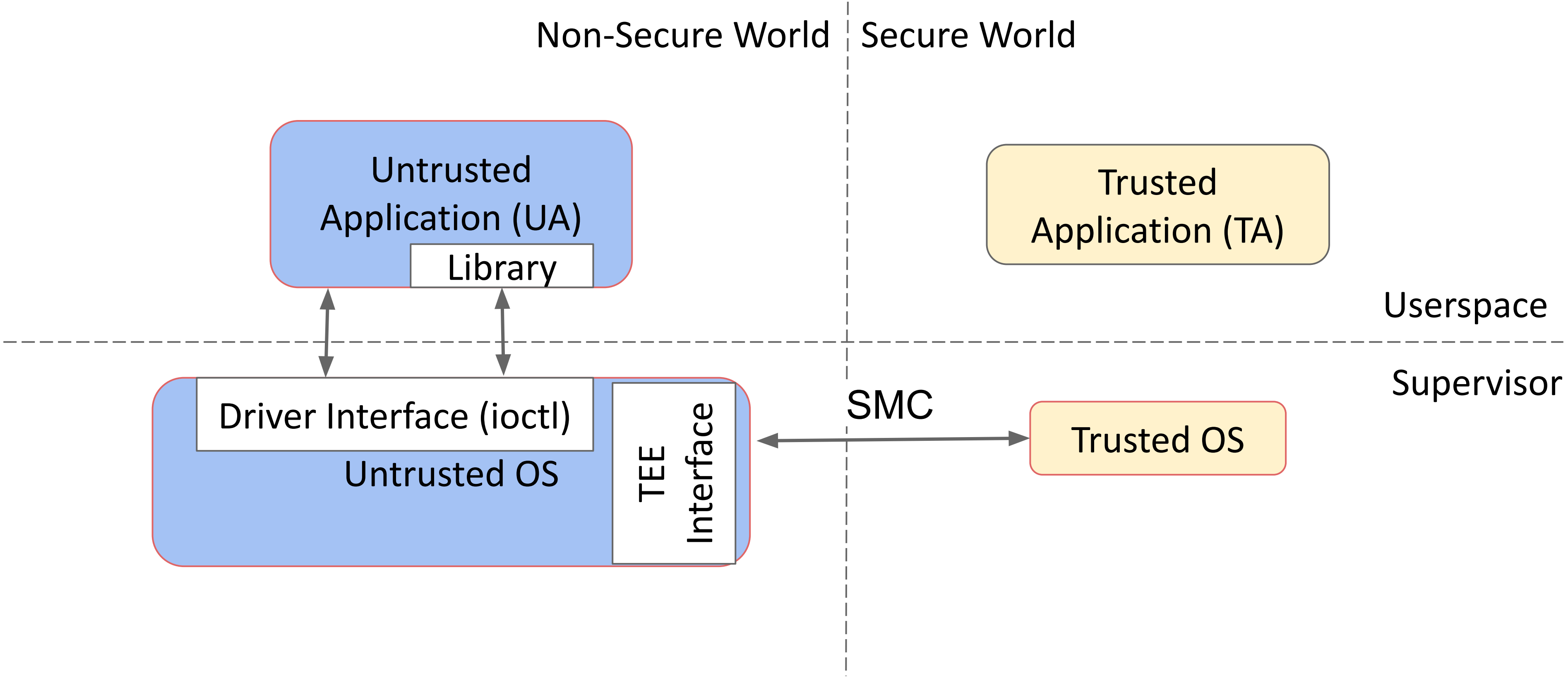
Untrusted OS ↔ Trusted OS



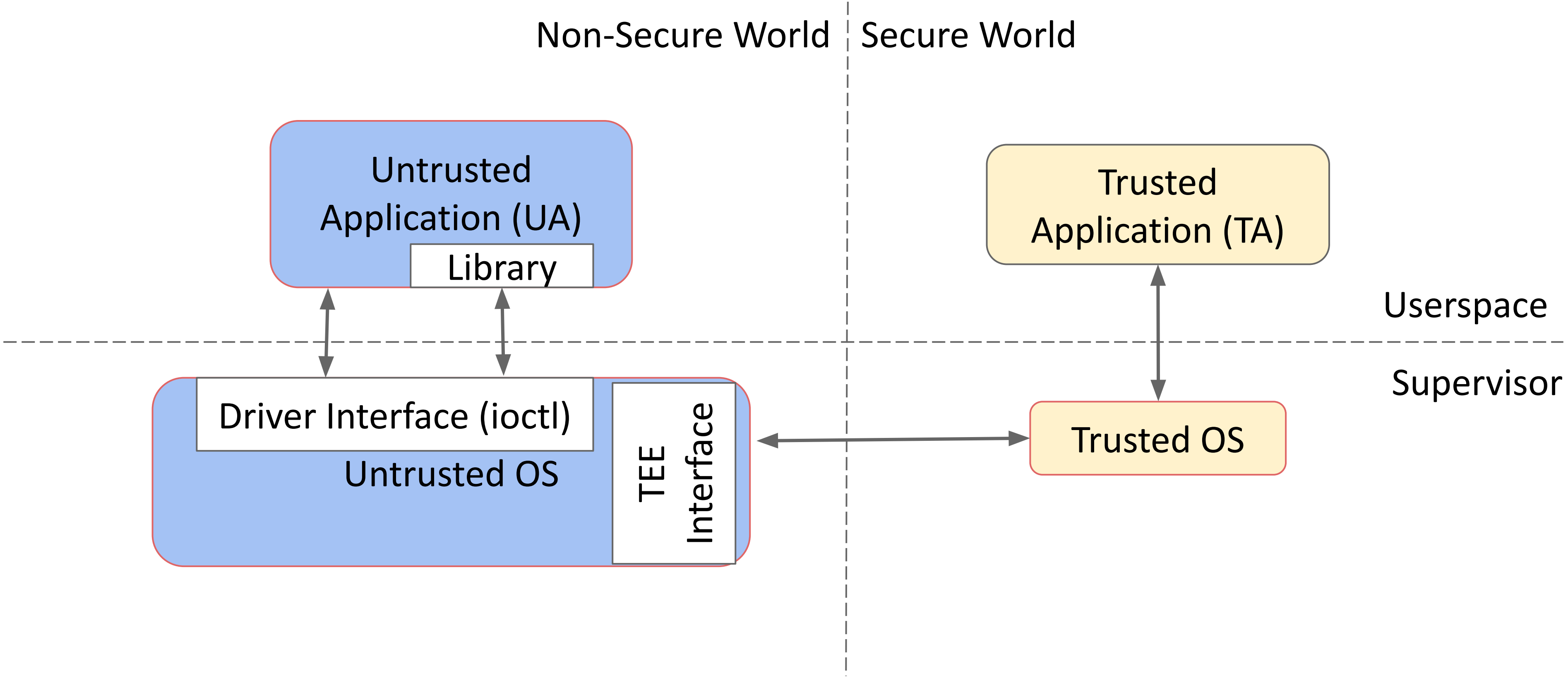
Untrusted OS ↔ Trusted OS



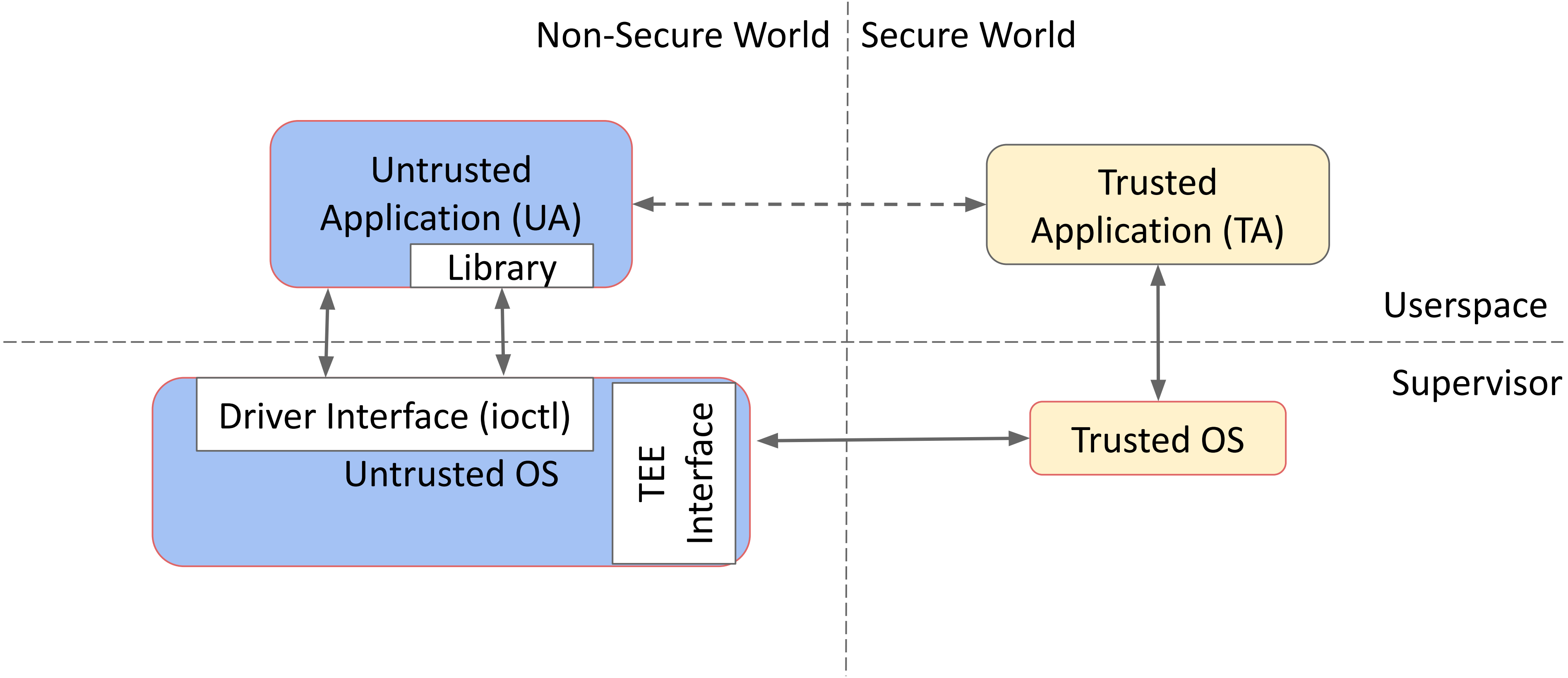
Untrusted OS ↔ Trusted OS



Untrusted OS ↔ Trusted OS



Untrusted OS ↔ Trusted OS



Communication with TA

- Requests to TA can contain pointers.

```
struct keymaster_sign_data_cmd {
    uint32_t  data_ptr; // Pointer to the data to sign
    size_t    dlen; // length of the data to sign
};
```

Structure of a sign request to KeyMaster TA.

Pointer translation and sanitization in untrusted OS

- Memory model could be different in untrusted and trusted OSes.

- One should use physical address for all pointer values between trusted and untrusted OSes.

Pointer translation and sanitization in untrusted OS

- *Sanitization:* Untrusted OS should check that the UA has access to the pointer provided in the request.
- *Translation:* Convert the virtual address to physical address.
- We call this **functionality in untrusted OS as PTRSAN**.

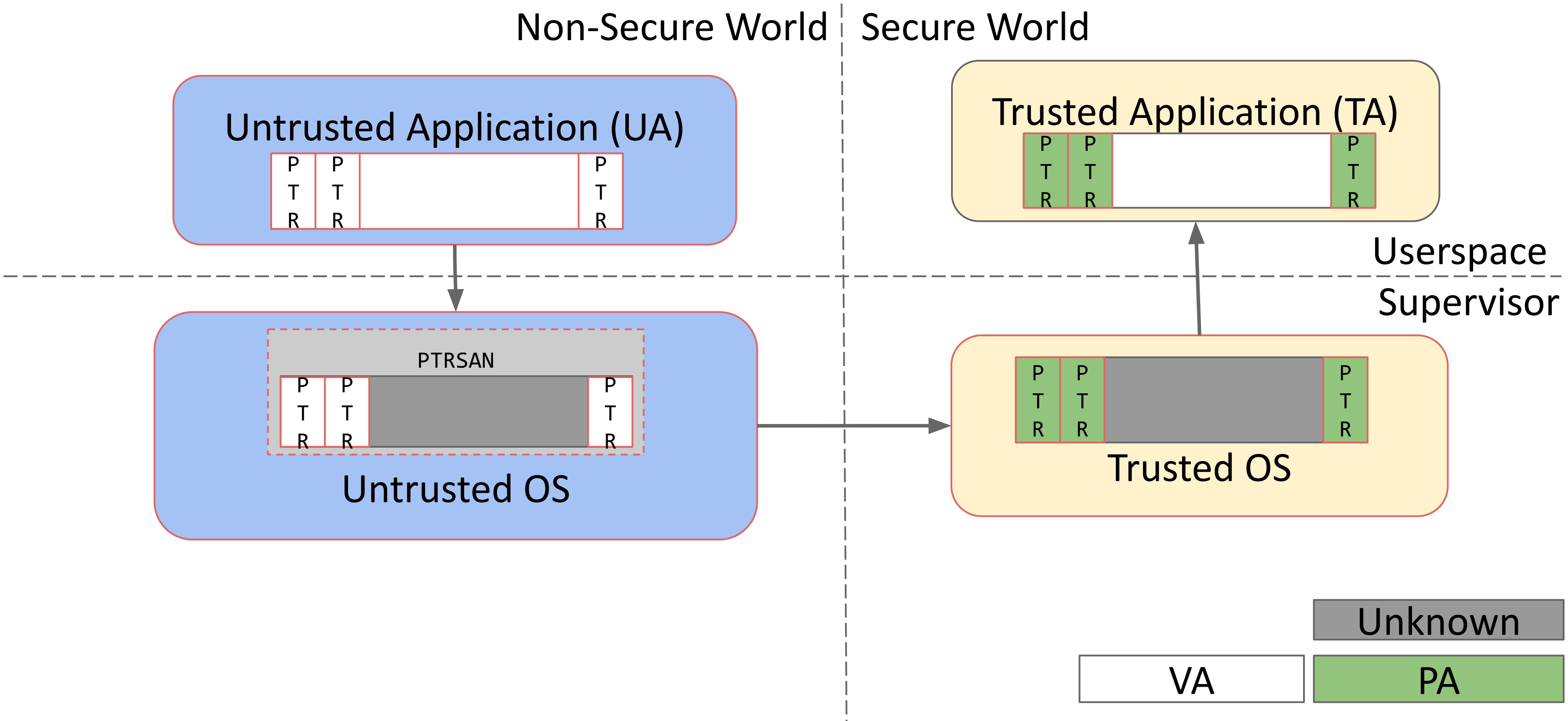
Example PTRSAN

```
int ptr_san(void *data, size_t len, phy_t *target_phy_addr)
{
    if(!access_ok(VERIFY_WRITE, data, len)) {
        return -EINVAL;
    }
    *target_phy_addr = get_physical_address(data);
    return 0;
}
```

Sanitization

Translation

PTRSAN


Non-Secure World
Secure World
Untrusted Application (UA)
PTR
PTR
PTR
Trusted Application (TA)
PTR
PTR
PTR
Userspace
Supervisor
PTRSAN
PTR
PTR
PTR
Untrusted OS
PTR
PTR
PTR
Trusted OS
Unknown
VA
PA

Handling untrusted pointers in trusted OS

- Check if the physical address indicated by the pointer belongs to the non-secure memory.
  - Protect trusted OS against untrusted OS

- Trusted OS (or TA) has no information about the UA which raised the request.

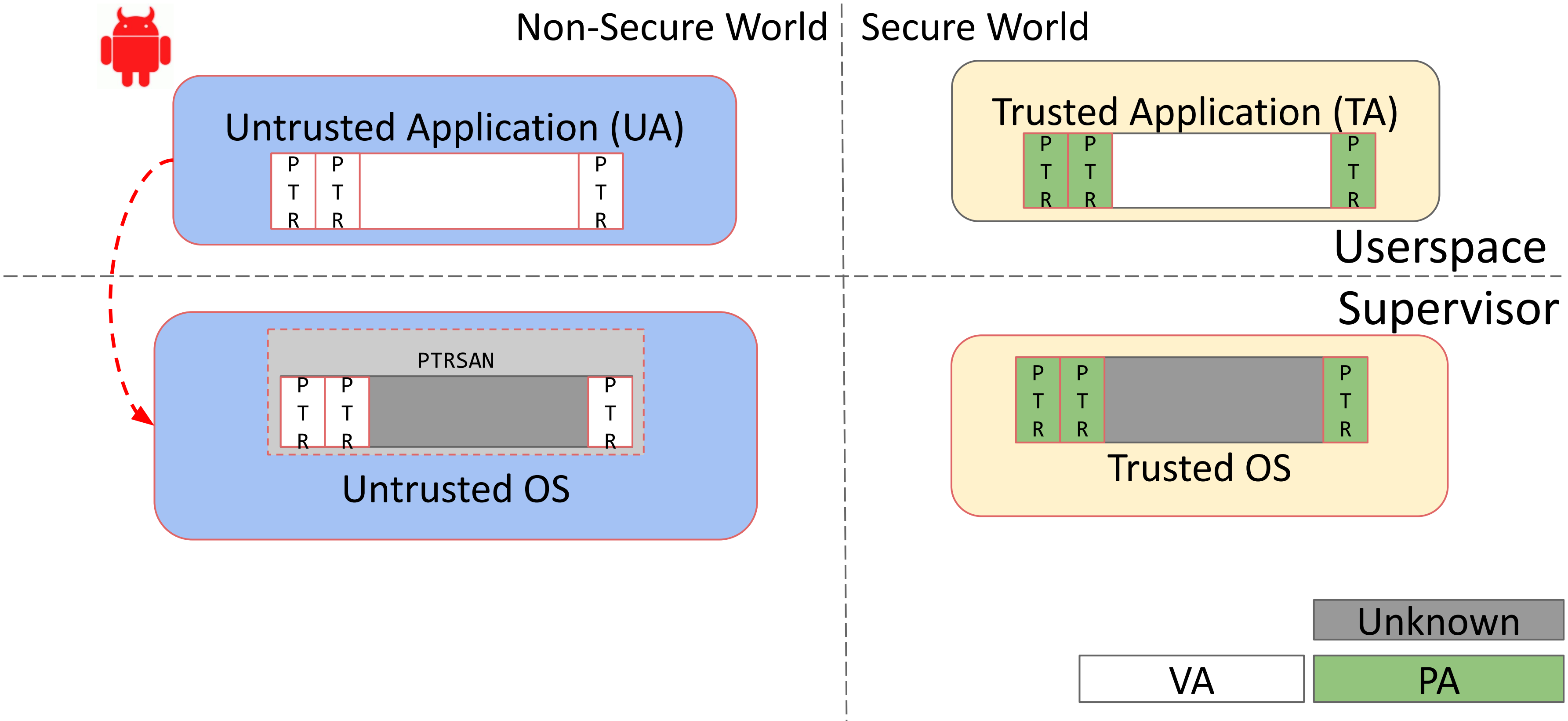
Handling untrusted pointers in trusted OS

- Check if the physical address indicated by the pointer belongs to the non-secure memory.
  - Protect trusted OS against untrusted OS

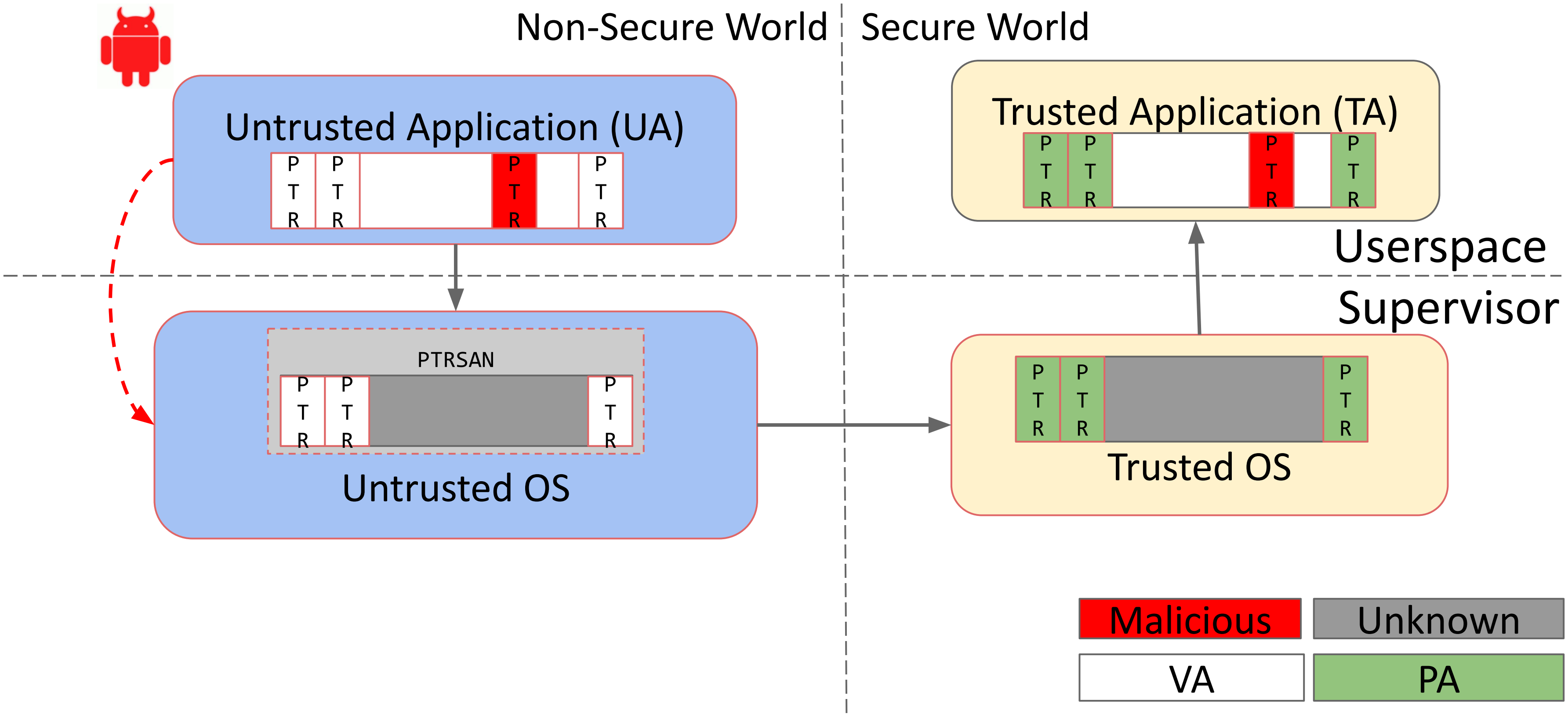
Semantic Gap

- Trusted OS (or TA) has no information about the UA which raised the request.

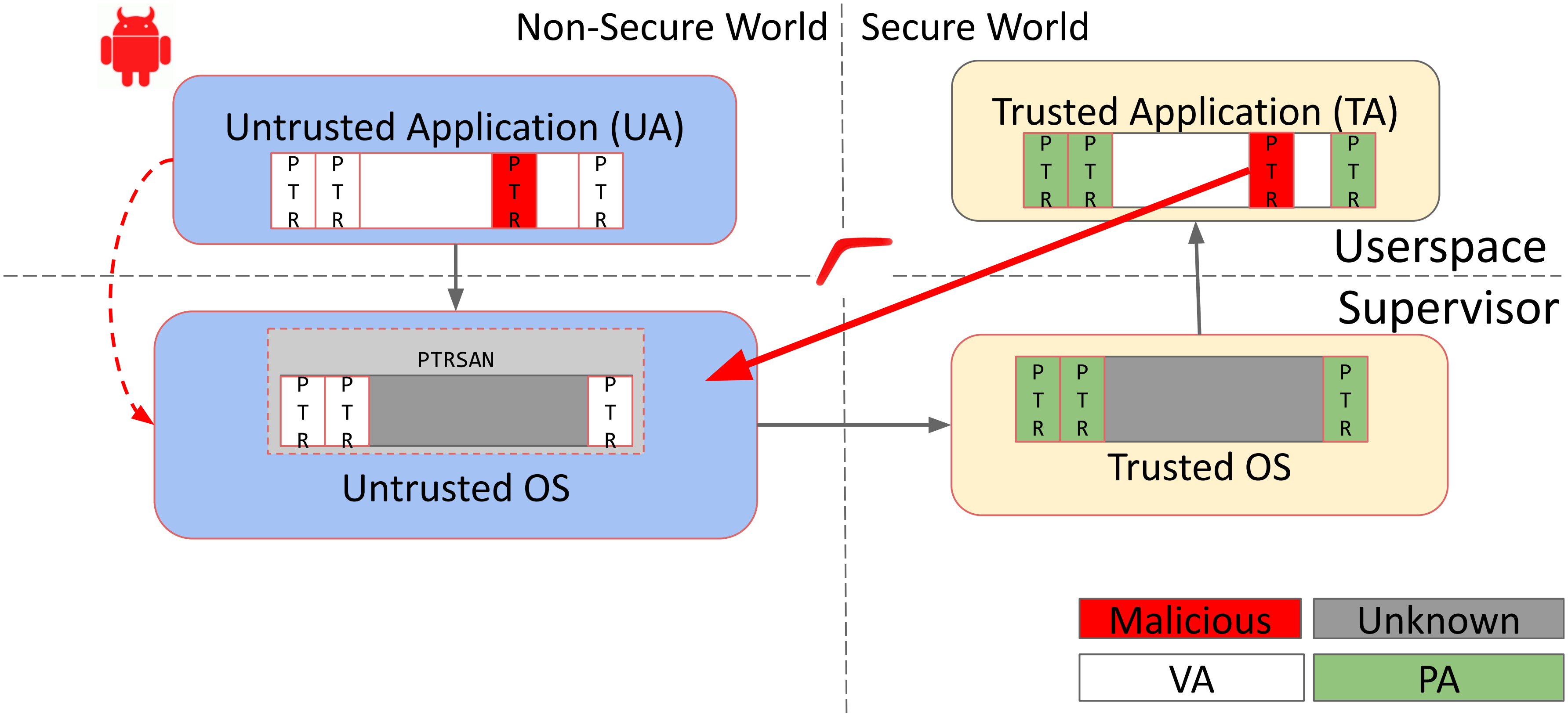
Bypassing Sanitization



Bypassing Sanitization



Boomerang flaw



Boomerang flaw

- Real world PTRSAN implementations are complex.

- Can we **bypass the validation** and make PTRSAN translate arbitrary physical address?

YES!!

- We can bypass PTRSAN *in all of the* popular TEE implementations.

TEE Name	Vendor	Impact	Bug Details
TrustedCore	Huawei	Arbitrary write	CVE-2016-8762
QSEE	Qualcomm	Arbitrary write	CVE-2016-5349
Trustonic	As used by Samsung	Arbitrary write	PZ-962*
Sierra TEE	Sierraware	Arbitrary write	No response from vendor
OP-TEE	Linaro	Write to other application's memory	Github issues 13, 14

*concurrently found by Google Project Zero (laginimaineb)

How to exploit Boomerang flaws?

Automatic detection of vulnerable TAs

- Goal: Find TAs which accepts pointers

- Static analysis of the TA binary:
  - Recover CFG of the TA
  - Paths from the entry point to potential sinks
  - Output the trace of Basic Block addresses

Results

TEE Name	Number of TAs	Vulnerable TAs
QSEE	3	3
TrustedCore	10	6

- ✓ **Arbitrary kernel memory read on Qualcomm phones.**
- ✓ **Kernel code execution on Huawei P8 and P9.**
- ✓ Demonstrated at GeekPwn.
- ✓ Geekpwn Grand Prize ($$$)

Impact

- Compromising untrusted OS == Rooting your device.
- Hundreds of millions of devices on the market today.
- Fixes yet to be released.
- Your device may be vulnerable!!!

seclab

Expectation

+

TrustZone®

System Security by ARM

=

Reality

+

TrustZone®
System Security by ARM

=

How to prevent Boomerang attacks?

Just fix PTRSAN? NO!!

This requires to understand the semantics of current and future TAs.

- Structure of the TA request?

- Which fields within the structure are pointers?

Root Cause

- **Semantic Gap**: Inability of the TA (or TEE) to verify whether the requested UA has access to the requested memory

- Should have a mechanism for the TA (or TEE) to verify or bridge the semantic gap.

Existing Defenses

- Page Table Introspection

- Dedicated Shared Memory Region (DSMR)

Page Table Introspection

- Implemented in NVIDIA Trusted Little Kernel.

- Untrusted OS **sends an id (e.g., pid) of the requested app (UA)** along with every request.

- **TA or TEE verify the access of all untrusted pointers** by referring to the requested **app page table**.

Page Table Introspection

Pros:

- Easy to implement.

Cons:

- Trusted OS depends on Untrusted OS
- Increases attack surface
- Page table walking could be dangerous

Dedicated Shared Memory Region (DSMR)

- Implemented in Open Platform -Trusted Execution Environment (OP-TEE).
- Dedicated memory region for communication between trusted and untrusted OS.
- UA should request access to the shared memory.
- TA or TEE verify that all untrusted pointers are within the dedicated memory region.

Dedicated Shared Memory Region (DSMR)

Pros:

- Simple
- Independence from Untrusted OS

Cons:

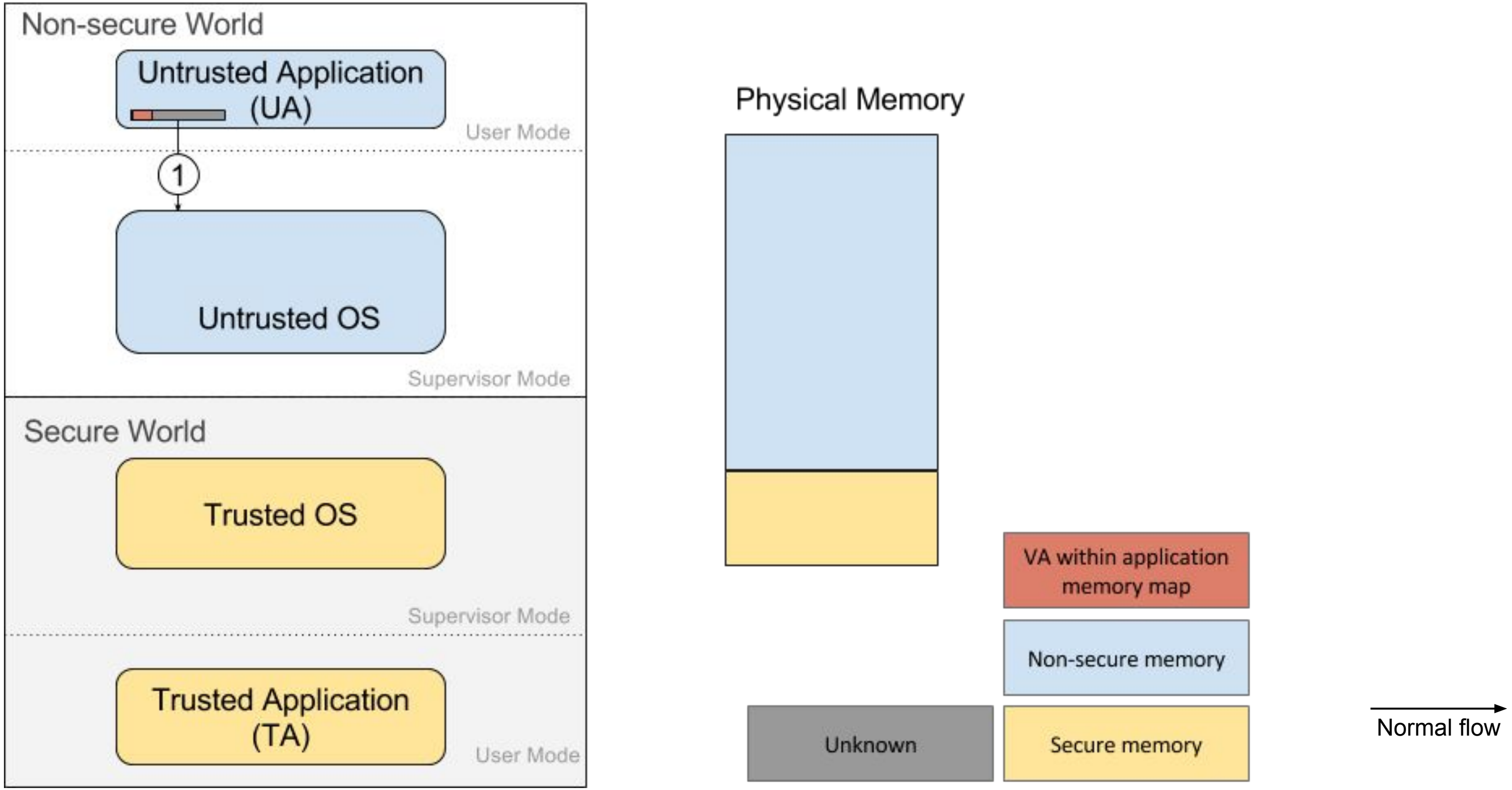
- **UA can interfere with other UAs via TAs (Partial Boomerang)**
- Additional copying to/from shared memory
- Allocation of shared memory could become bottleneck in case of multithreaded applications.
- Some applications (integrity monitoring) are hard to implemented using DSMR.

Cooperative Semantic Reconstruction (CSR)

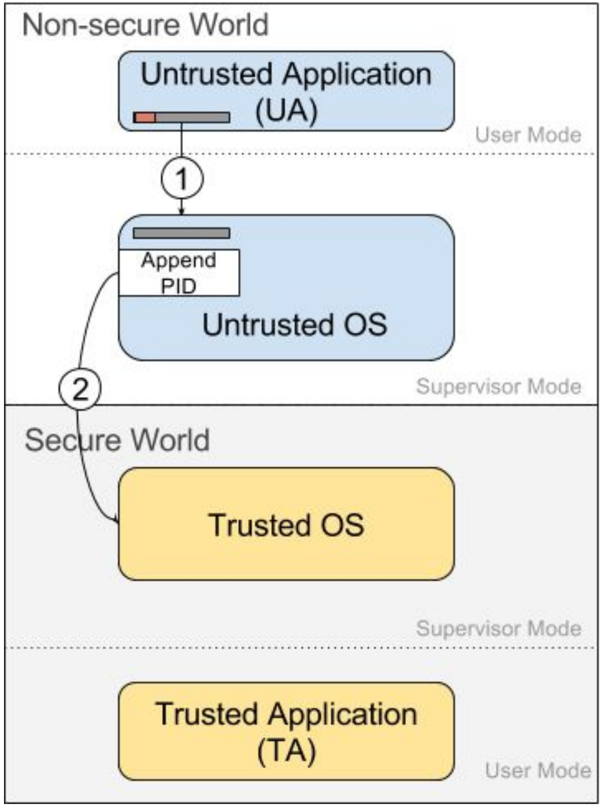
- Novel defense proposed by us.

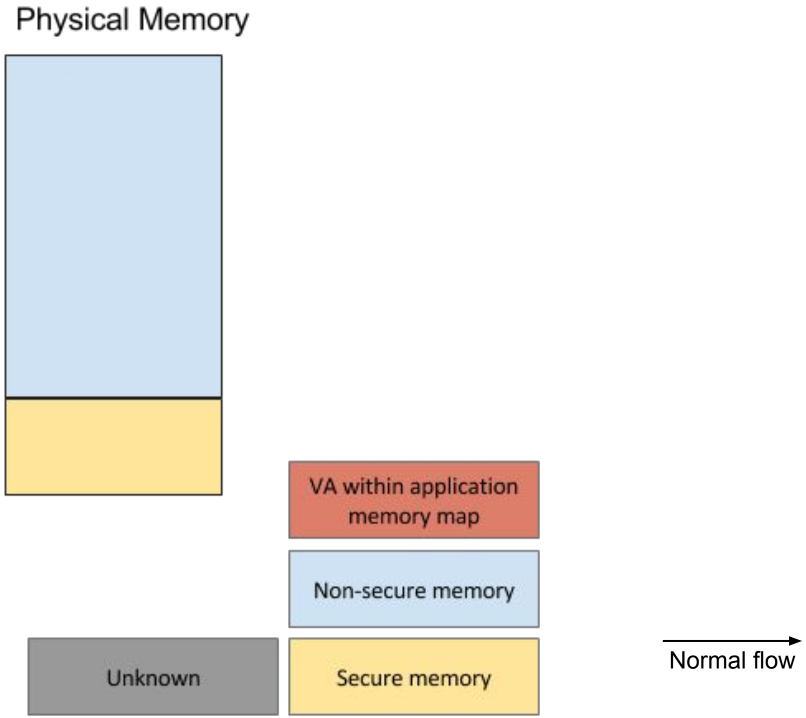
- Provides a channel for Trusted OS to query Untrusted OS for validation.

Cooperative Semantic Reconstruction (CSR)

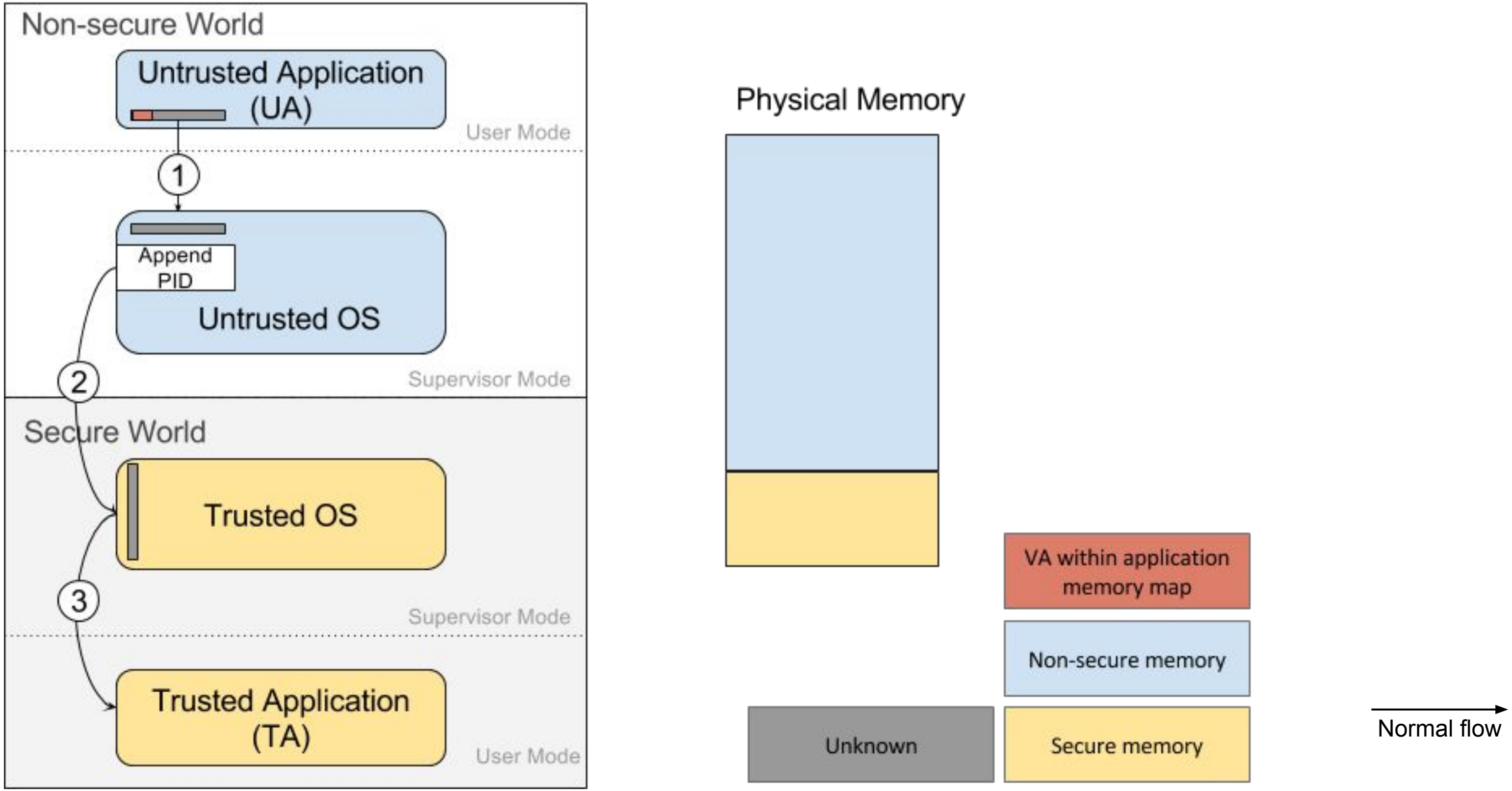


Cooperative Semantic Reconstruction (CSR)

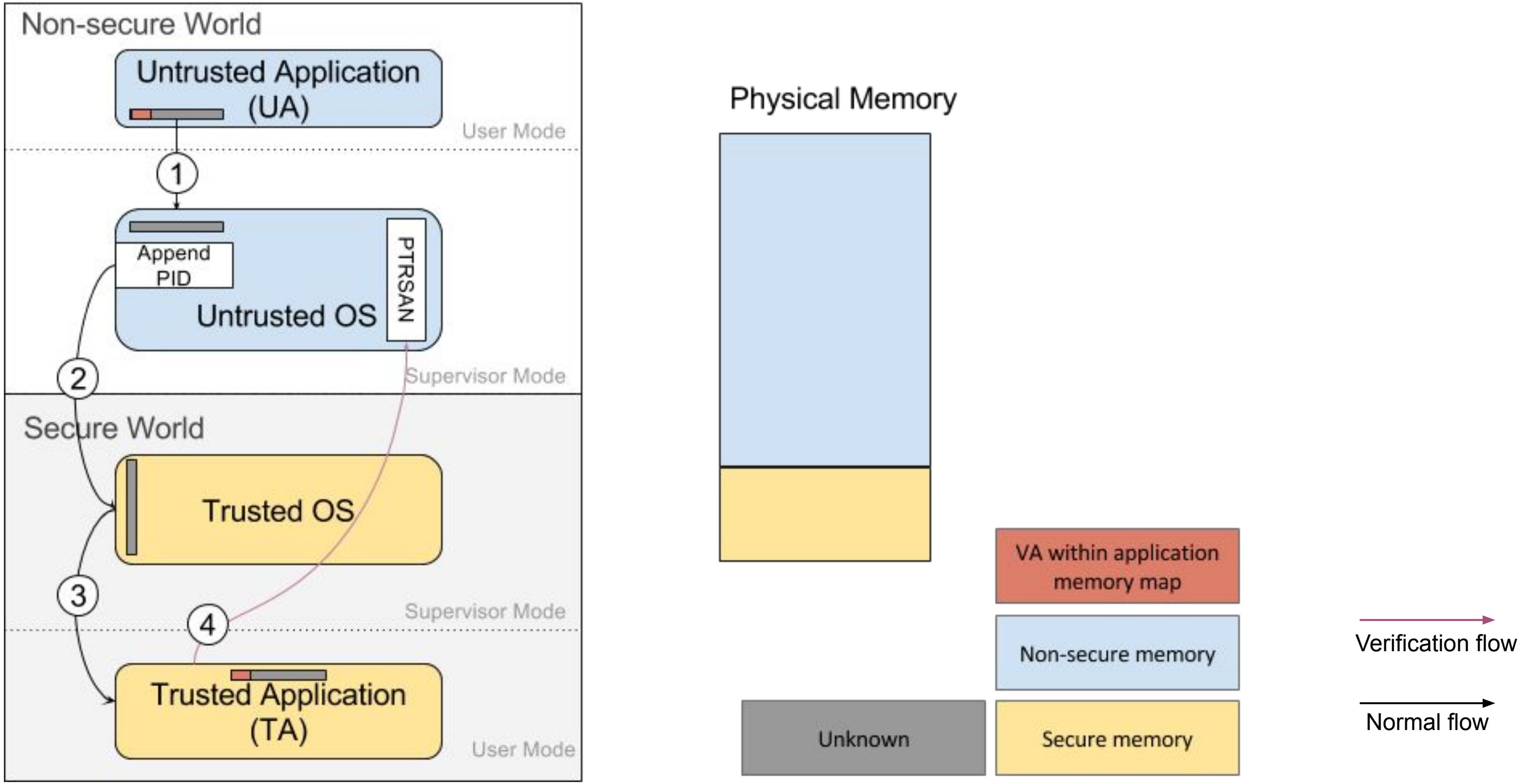




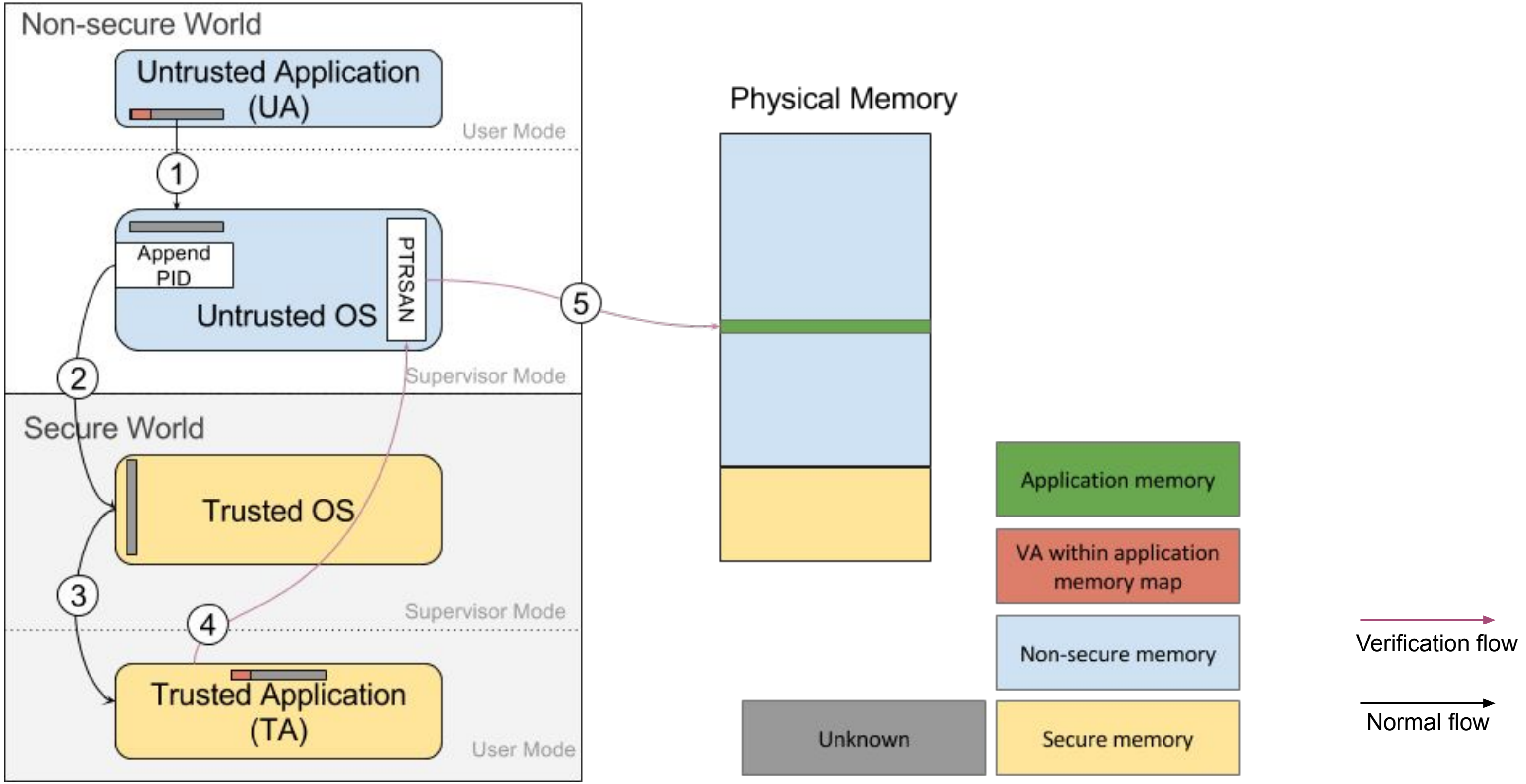
Cooperative Semantic Reconstruction (CSR)



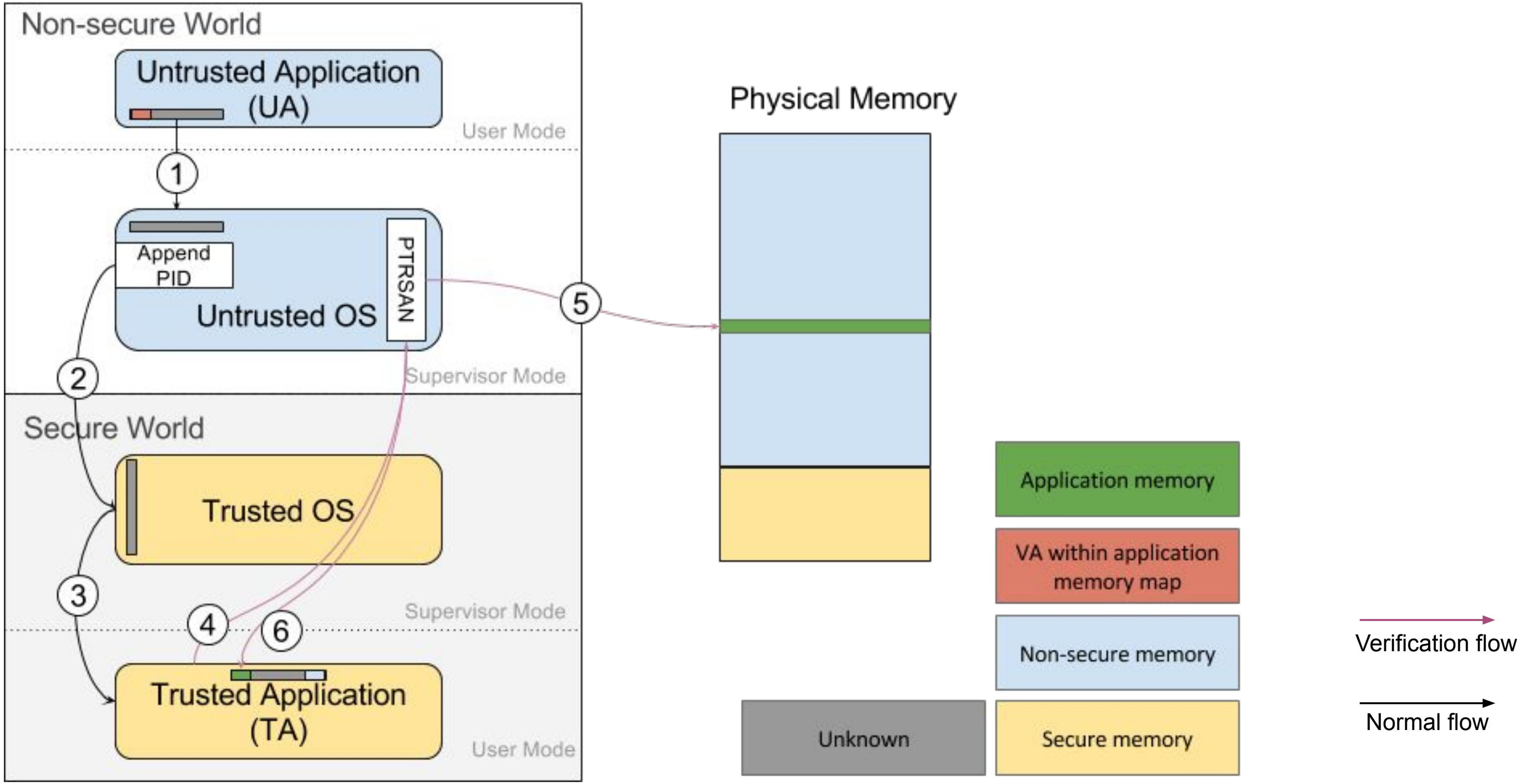
Cooperative Semantic Reconstruction (CSR)



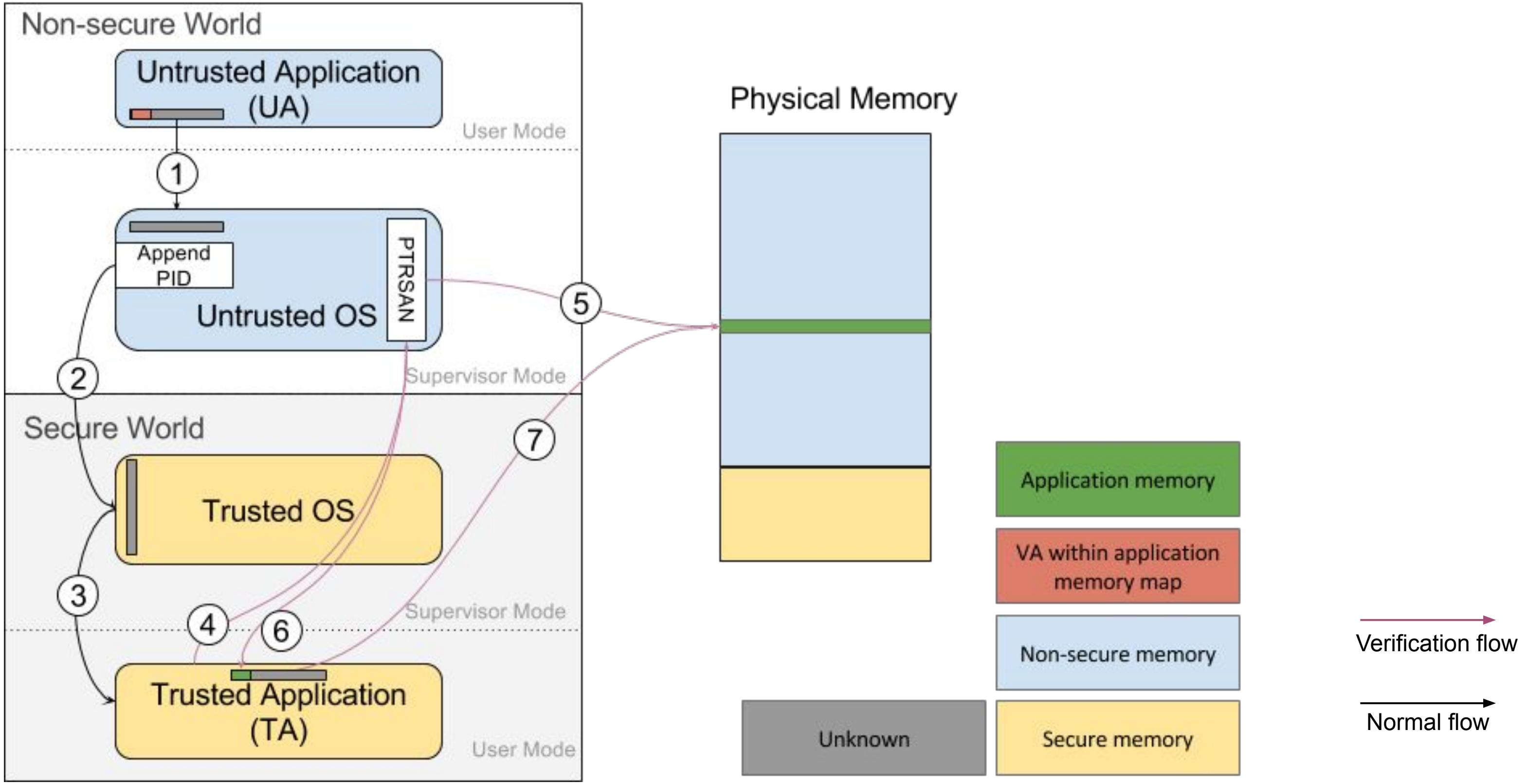
Cooperative Semantic Reconstruction (CSR)



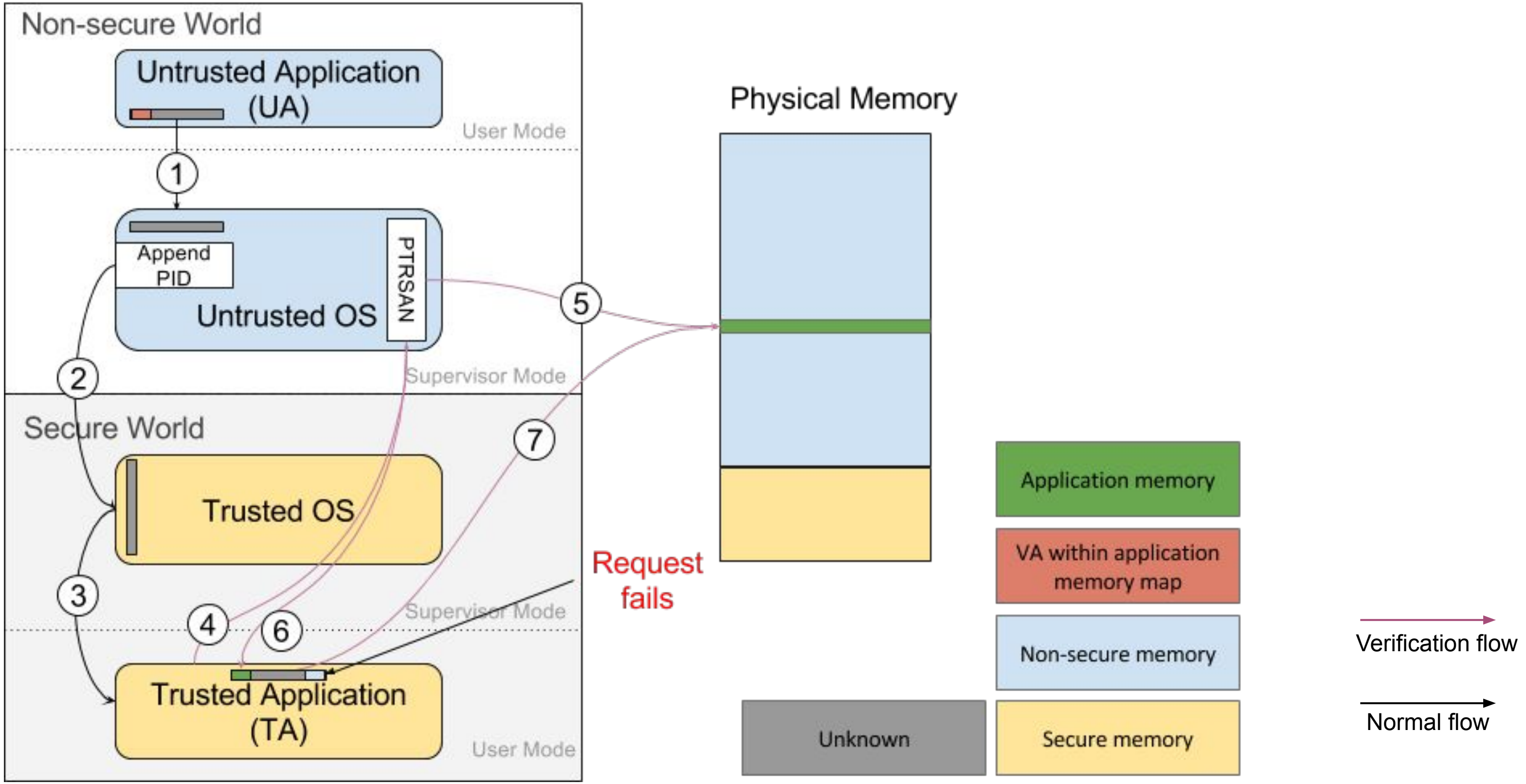
Cooperative Semantic Reconstruction (CSR)



Cooperative Semantic Reconstruction (CSR)



Cooperative Semantic Reconstruction (CSR)



Implementation

- Open Platform-Trusted Execution Environment (OP-TEE)
  - Easy to use
  - Helpful community
  - Has DSMR already implemented

- HiKey Development board (Lemaker Version)

Evaluation: CSR vs DSMR

- Microbenchmark: Time to validate single memory pointer/page.

Defense Name	Overhead Component	Overhead (μs)	Total Overhead (μs)
CSR	Untrusted OS verification	21.909	26.891
	Mapping in trusted OS	4.982	
DSMR	Shared memory allocation	13.795	21.777
	Shared memory release	7.982	

Evaluation: CSR vs DSMR

- XTEST
- Default OP-TEE Test suite.
- 63 Tests covering sanity, functionality, benchmarking and compliance.

Evaluation: CSR vs DSMR

Tests Category	Overhead (CSR - DSMR) averaged over 30 runs	
	Avg Time(%)	Avg Time (ms)
Basic Functionality	-0.58%	-7.168
Trusted-Untrusted Communication	4.45%	0.510
Crypto Operations	-1.72%	-901.548
Secure File Storage	0.03%	0.694
Average over All Categories	-0.0344%	-189.919 ms

CSR faster than DSMR **DSMR faster than CSR**

Evaluation: CSR vs DSMR

- DSMR is slow in practice:
  - Synchronized access for shared memory allocation.
  - Additional copying.

- CSR can be slow for simple requests.
  - Setup of tracking structures.

Conclusion

- ✓ Boomerang: New class of bugs
- ✓ Automated attack vector detection
- ✓ Novel, practical, and efficient solution against boomerang: Cooperative semantic reconstruction (CSR)
- ✓ Detection, exploits (?) , and defenses available at github