Expected Structure of Presentation

- **Background** : What is the problem this paper is trying to solve?
- **Related Work or Why others fail**: What is wrong with (then) existing solutions?
- **Overview**: High-level idea of the proposed solution.
- **Technical Details**: Some details about the proposed solution.
- **Evaluation**: What does the paper evaluate and why?
- **Remarks**: What you think of the paper? Pros and Cons.
- **Conclusion**: What do you think of the problem and proposed solution?

Keep in Mind.

- Not all papers need to have this structure.
- Some papers need not have related work or why others fail part (esp. Attack papers)
- Reuse existing material -- from author's or conference website.
- Add your spin to the presentation.
- Your insights from the paper are important!!

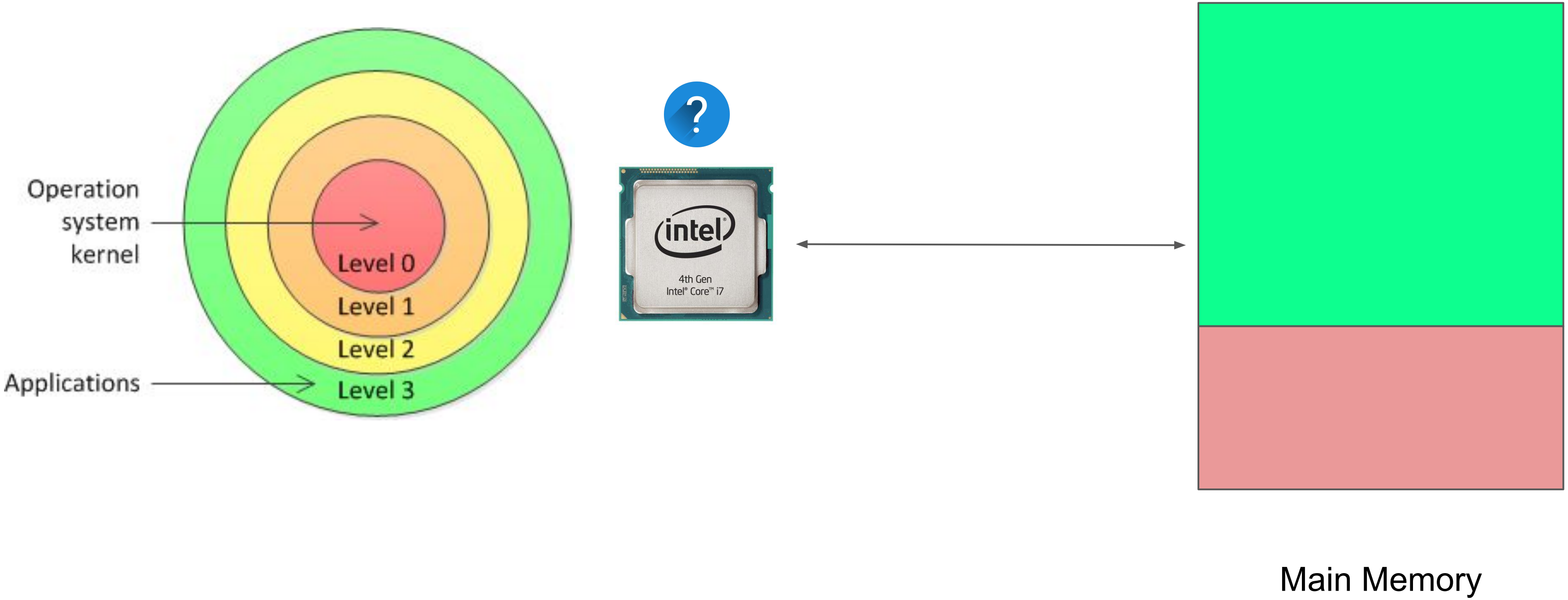
Boomerang: Exploiting the Semantic Gap in Trusted Execution Environments

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

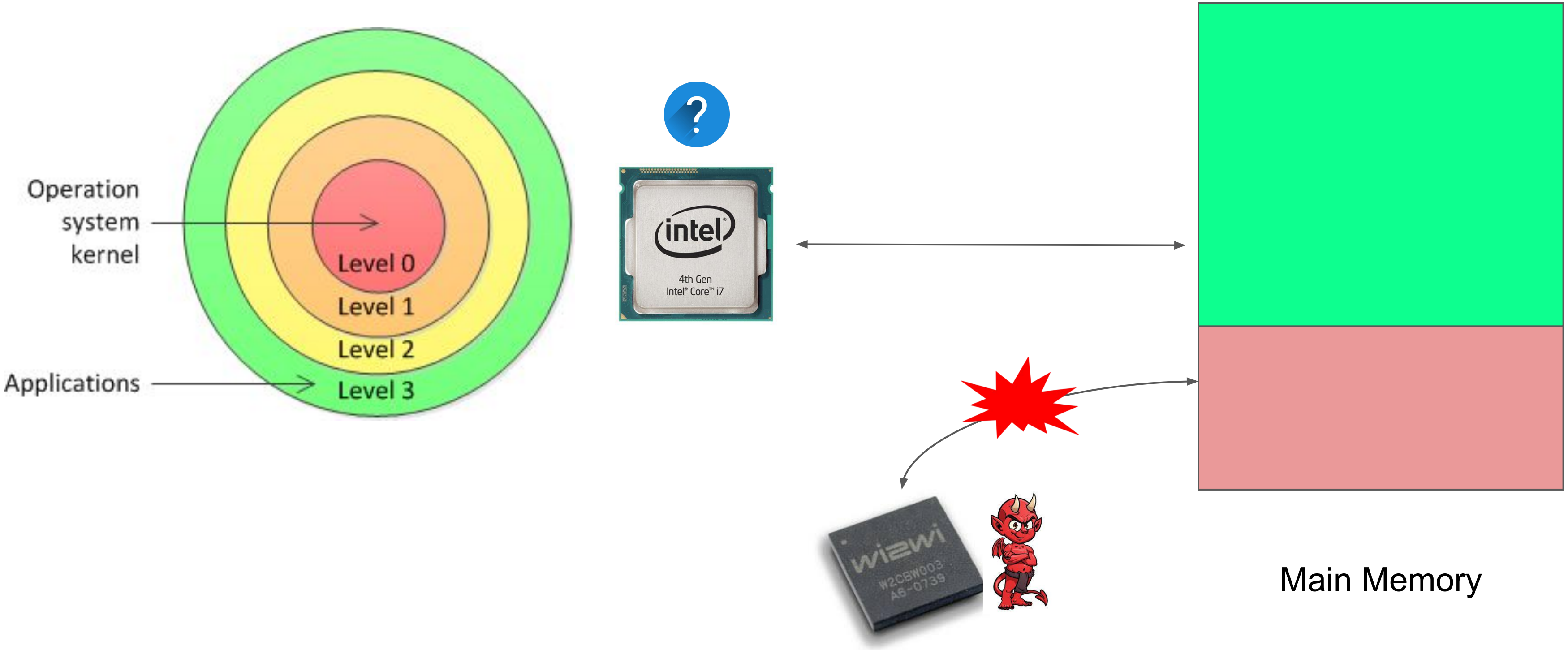
UCSB

Sandia National Laboratories

x86 Privilege levels



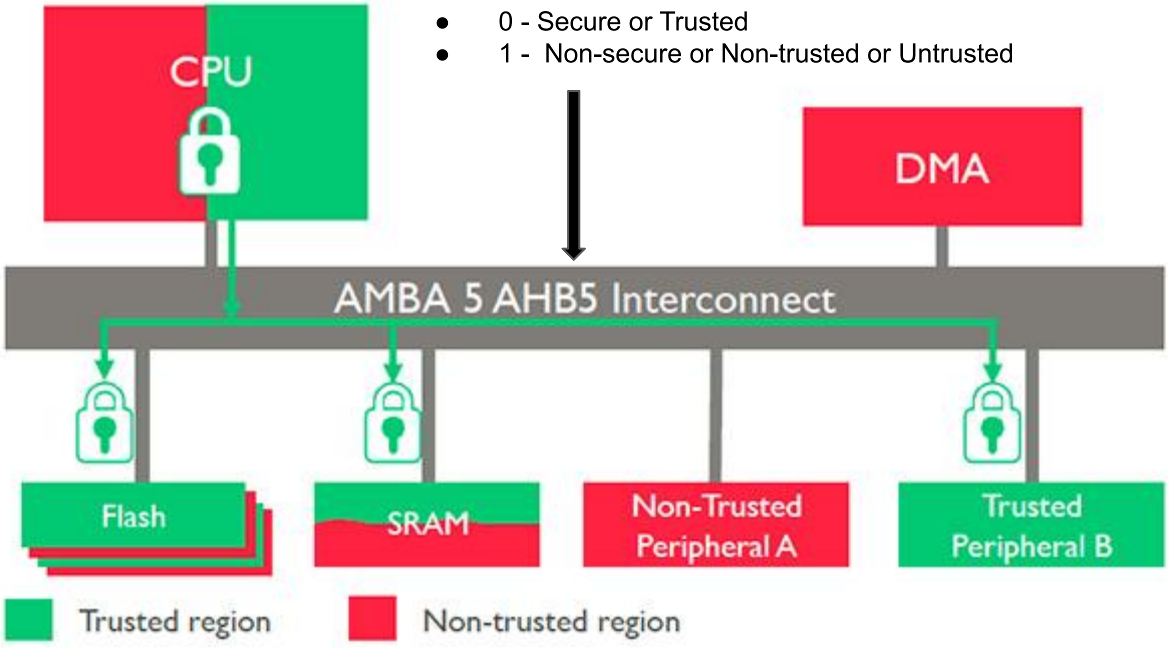
x86 Privilege levels



ARM TrustZone

NS Bit

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



Picture reused from arm.com

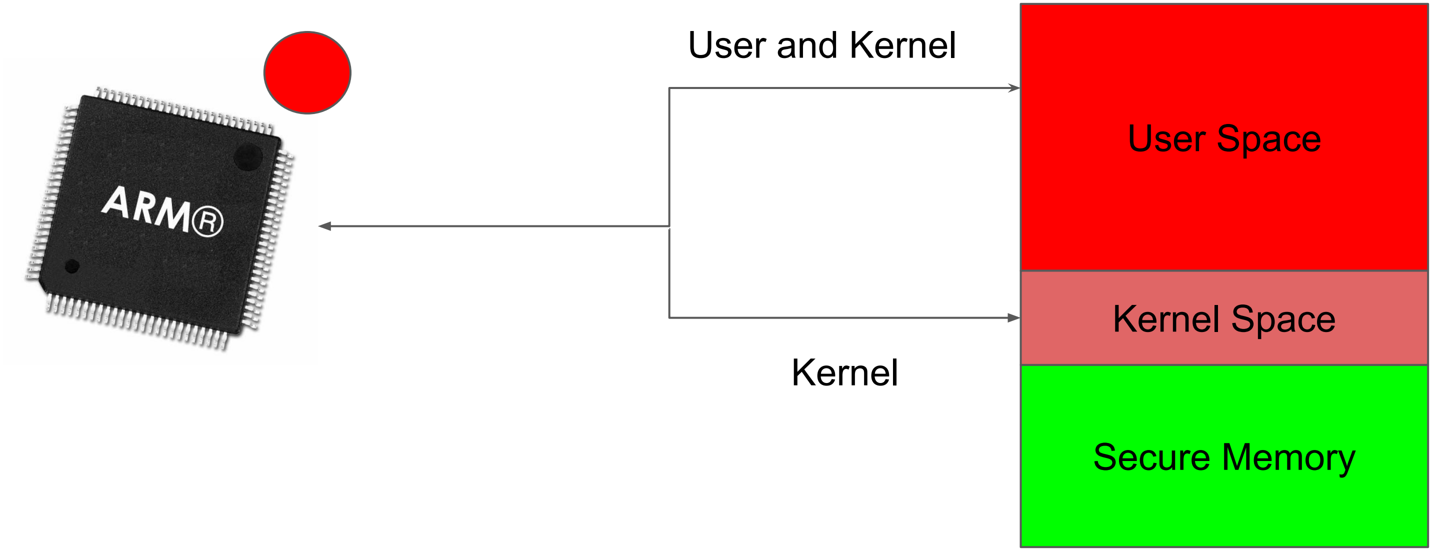
Trusted Execution Environment (TEE)

- Hardware-isolated execution environments (e.g., ARM TrustZone)

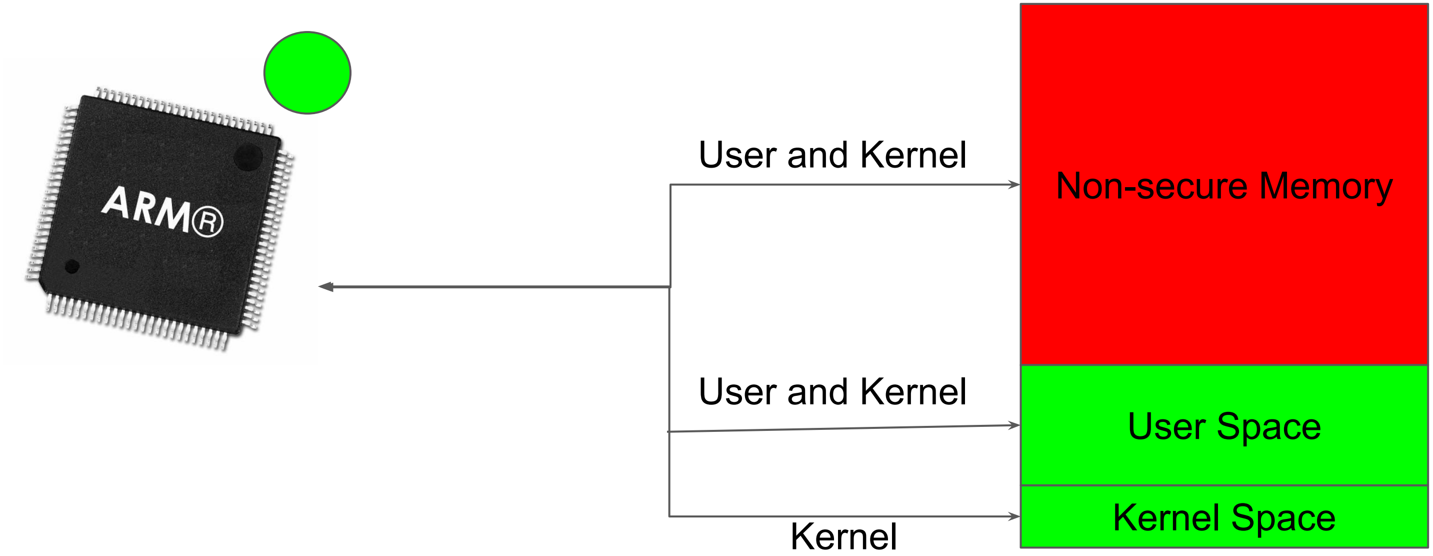
    - Non-secure world
        - Untrusted OS and untrusted applications (UAs) (e.g., Android and apps)

    - Secure world
        - Higher privilege, can access *everything*
        - Trusted OS and trusted applications (TAs).

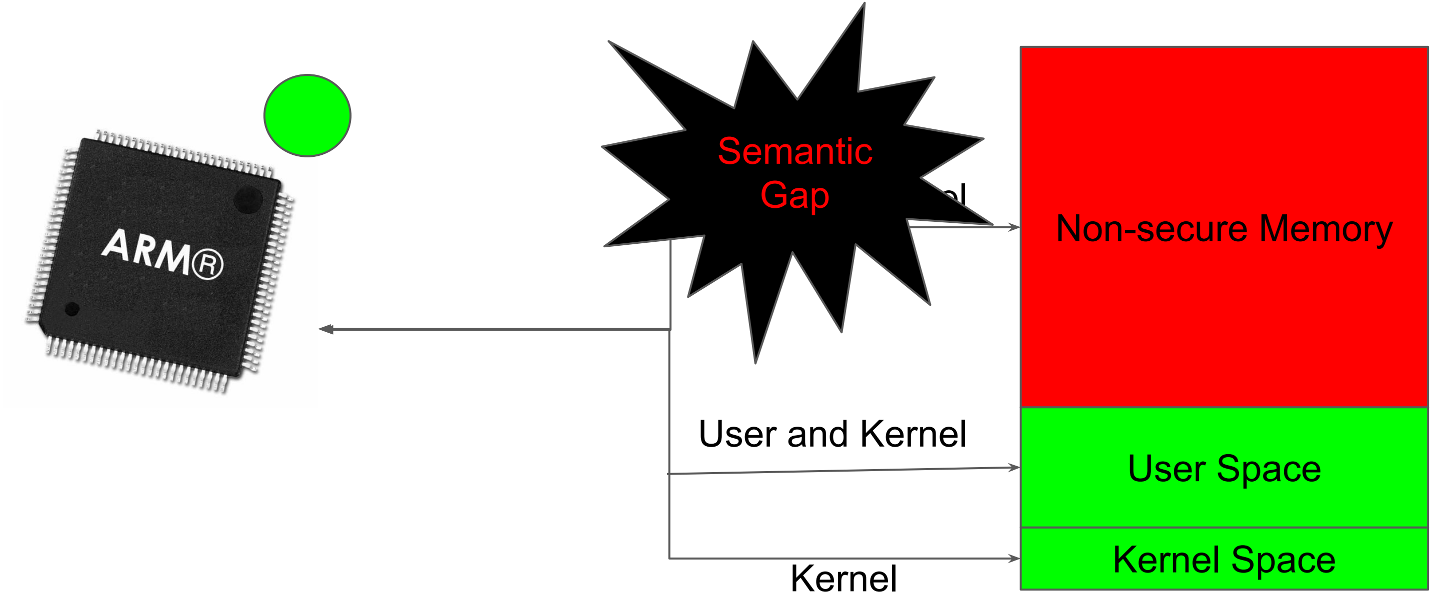
Normal World running Untrusted OS (e.g., Android)



Secure World running Trusted OS (e.g., QSEE)



Secure World running Trusted OS (e.g., QSEE)



Expectation

+

TrustZone®

System Security by ARM

=

Reality

+

TrustZone®
System Security by ARM

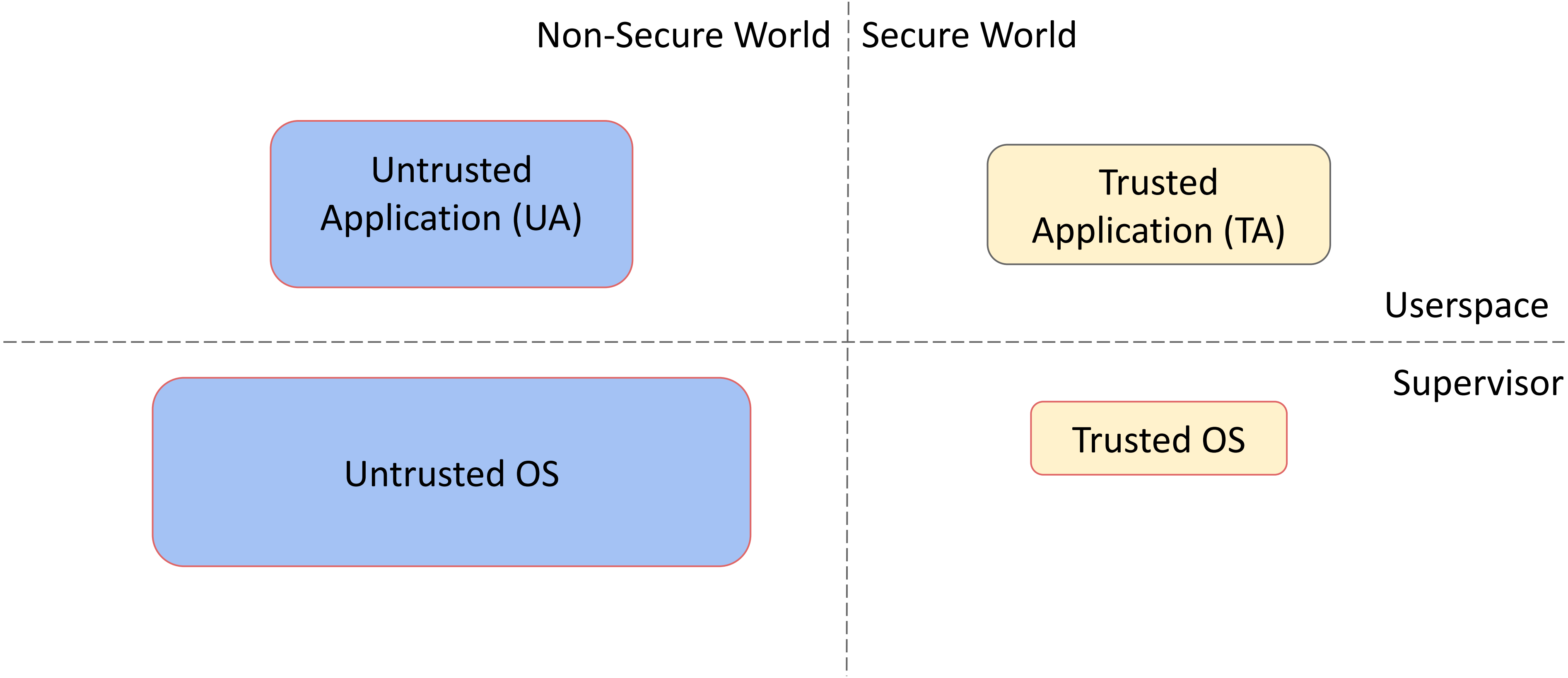
=

Untrusted OS ↔ Trusted OS

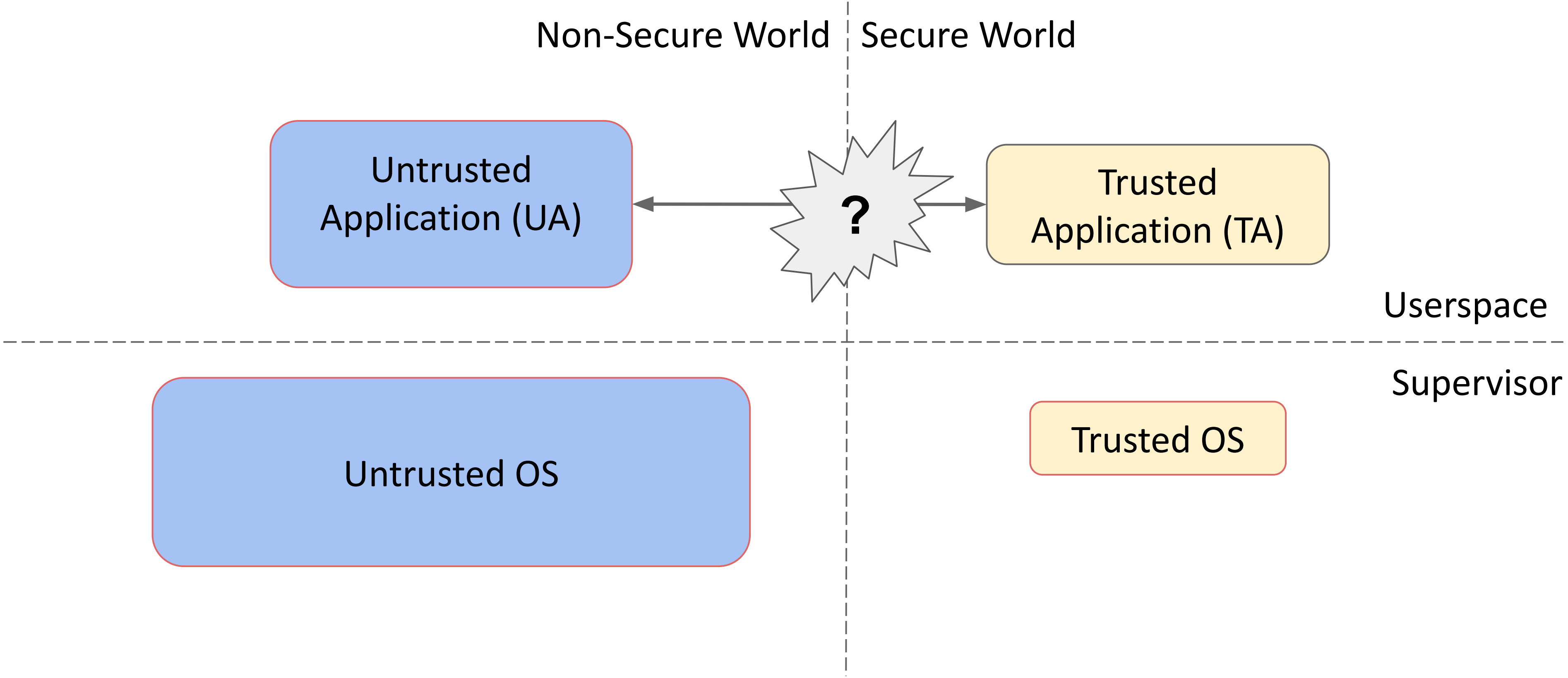
- Untrusted applications (UAs) request trusted applications (TAs) to perform privileged tasks.

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

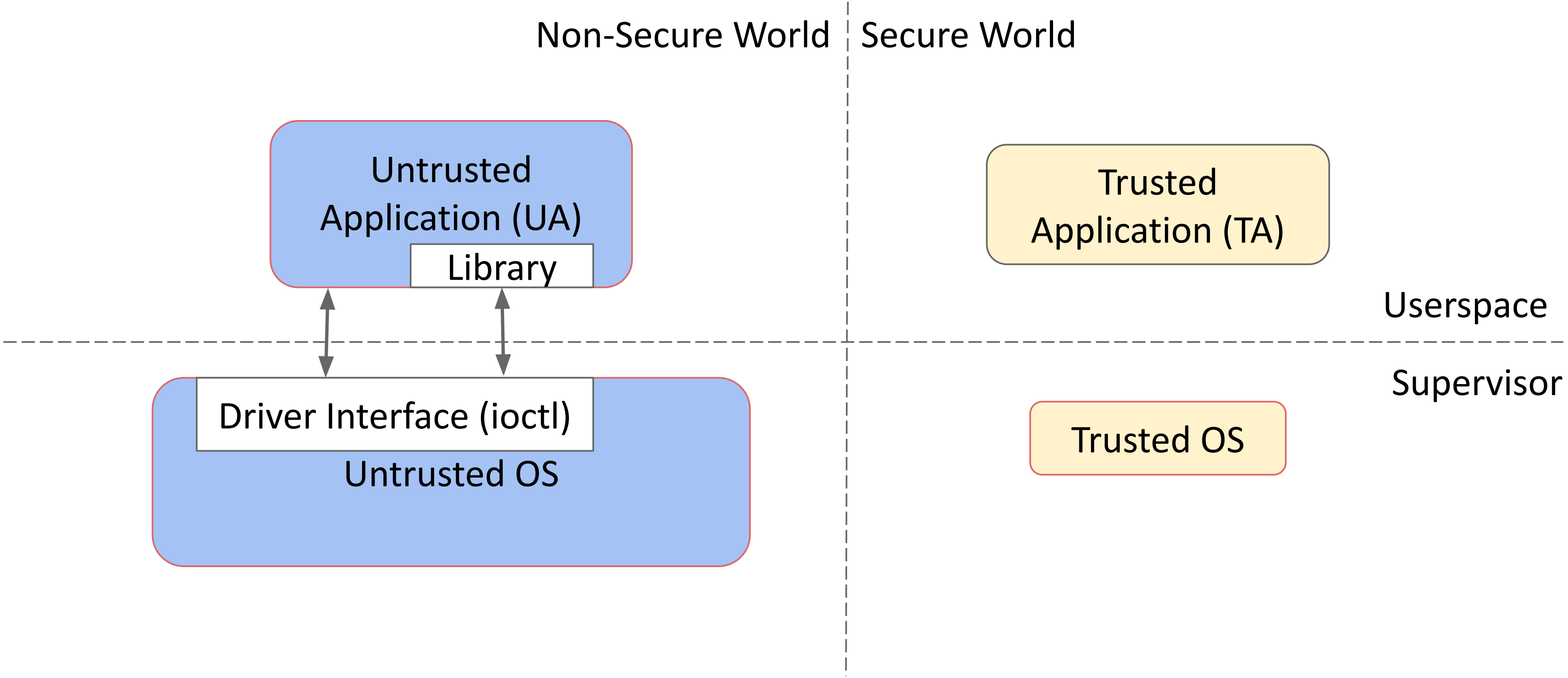
Untrusted OS ↔ Trusted OS



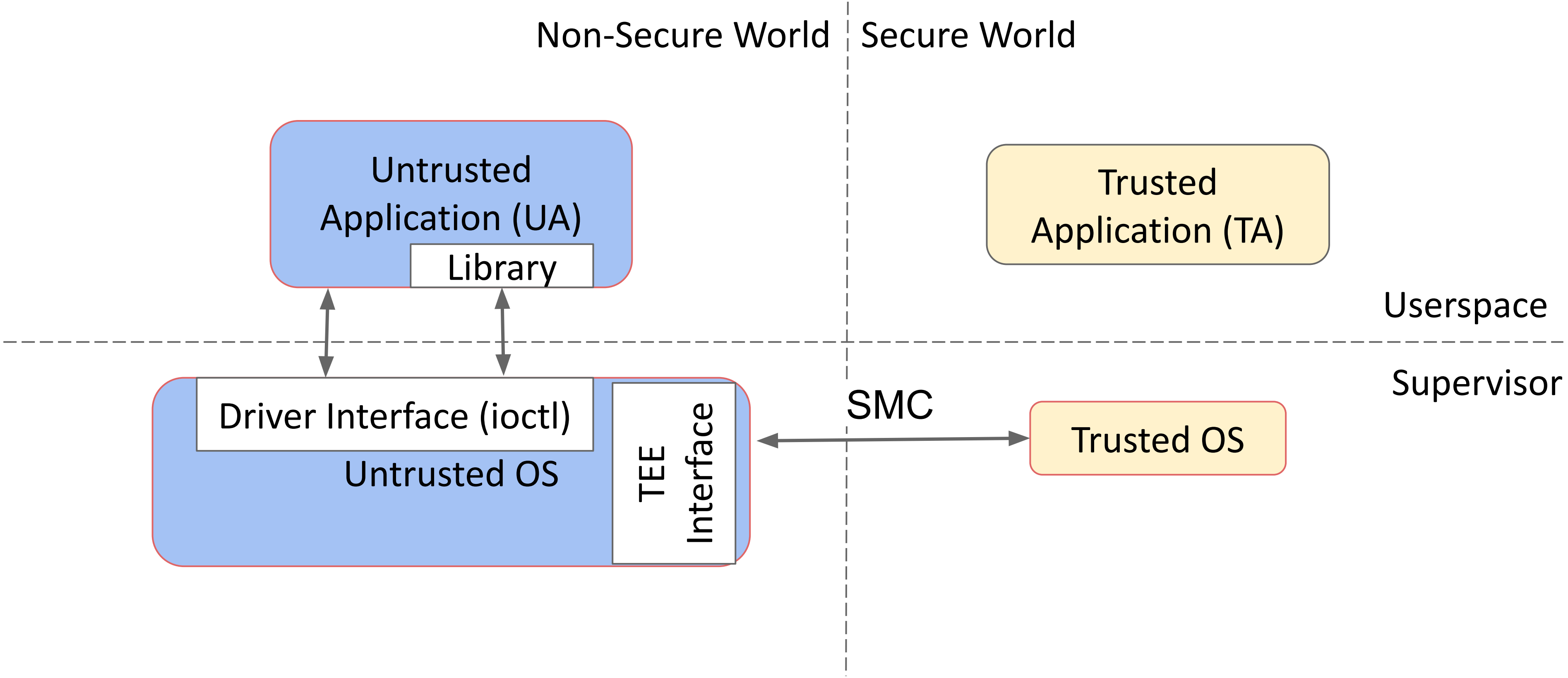
Untrusted OS ↔ Trusted OS



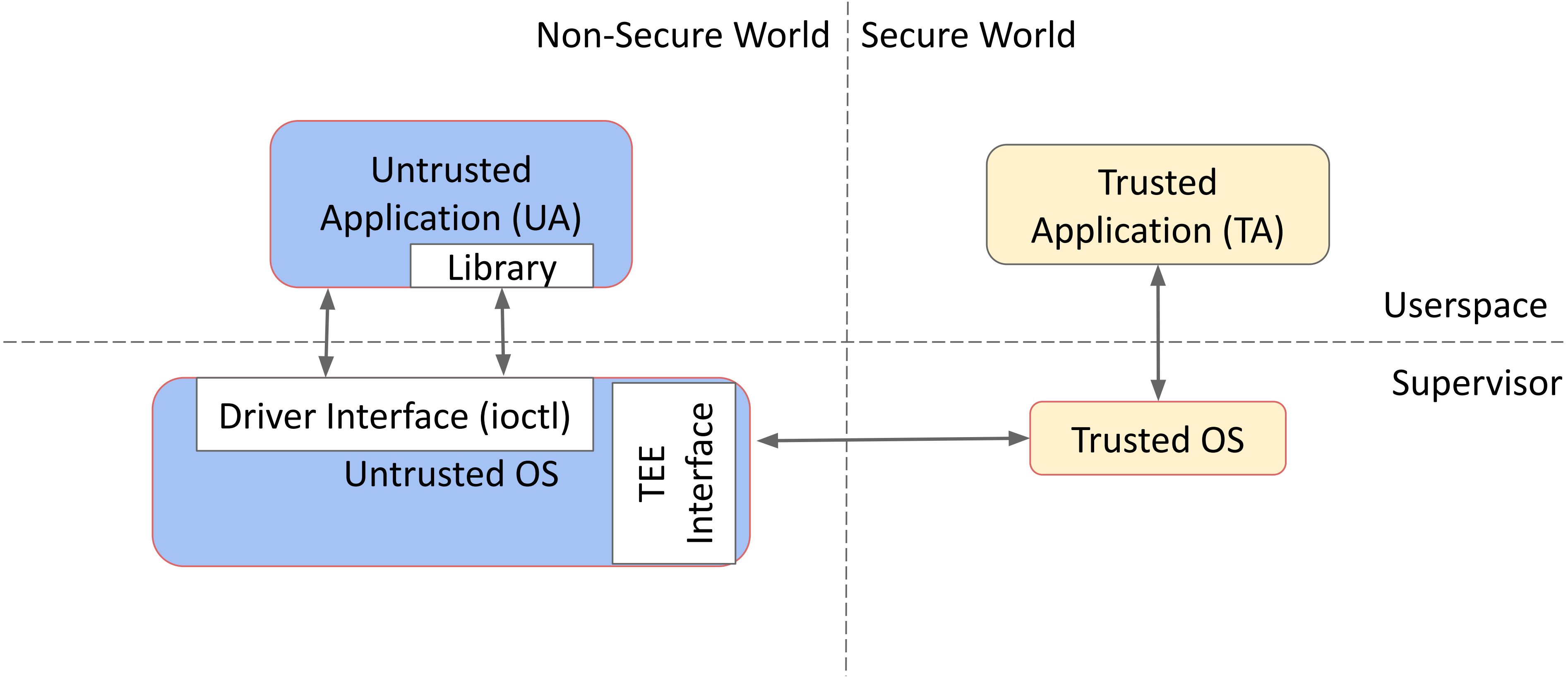
Untrusted OS ↔ Trusted OS



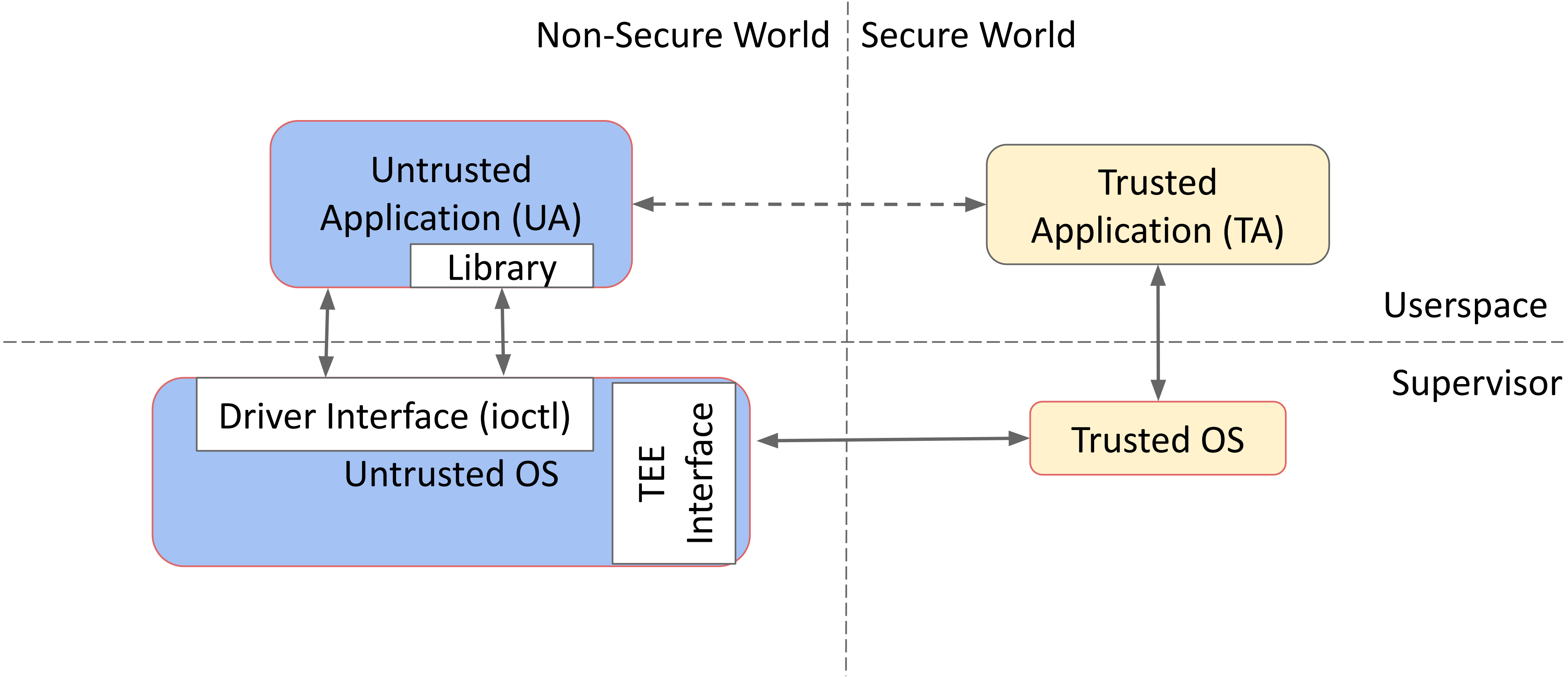
Untrusted OS ↔ Trusted OS



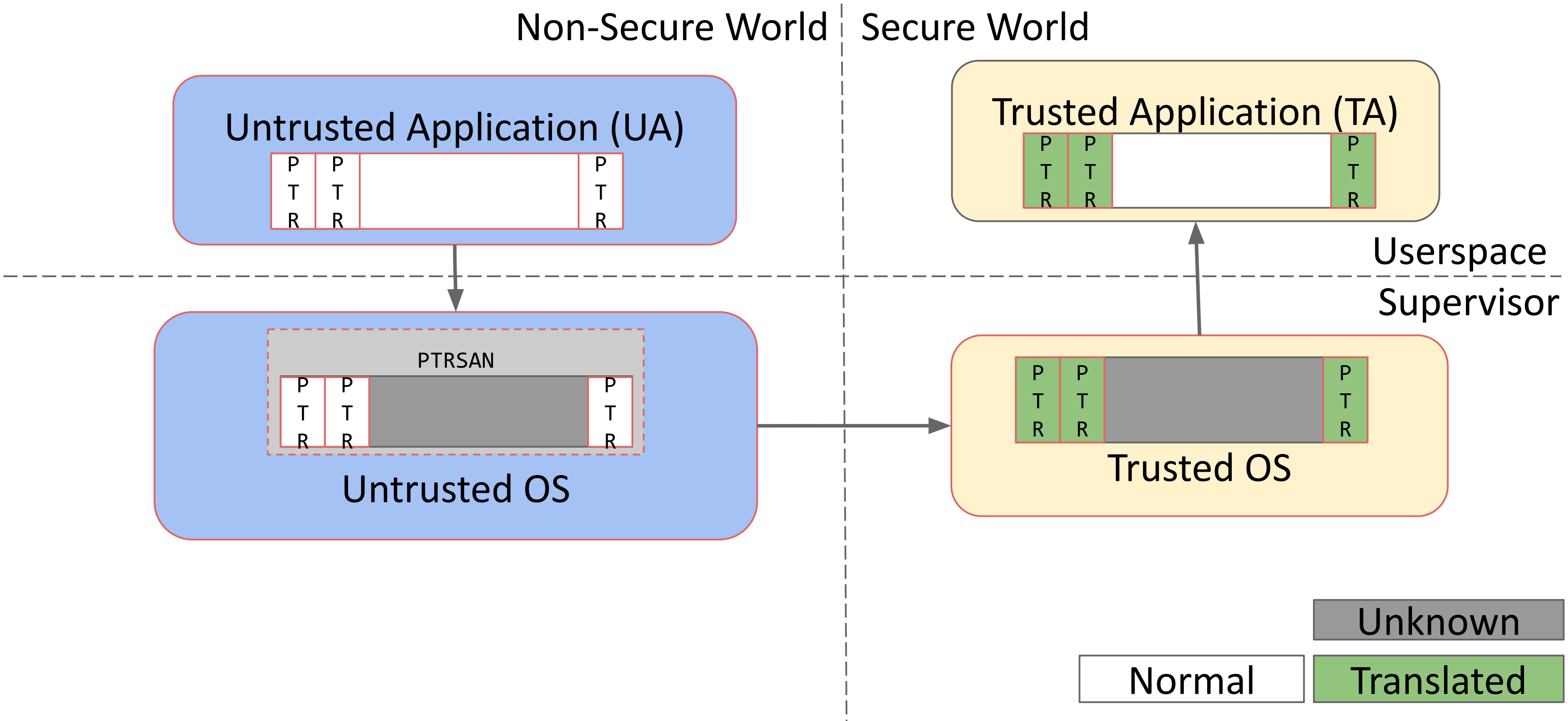
Untrusted OS ↔ Trusted OS



Untrusted OS ↔ Trusted OS



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
Normal
Translated

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.

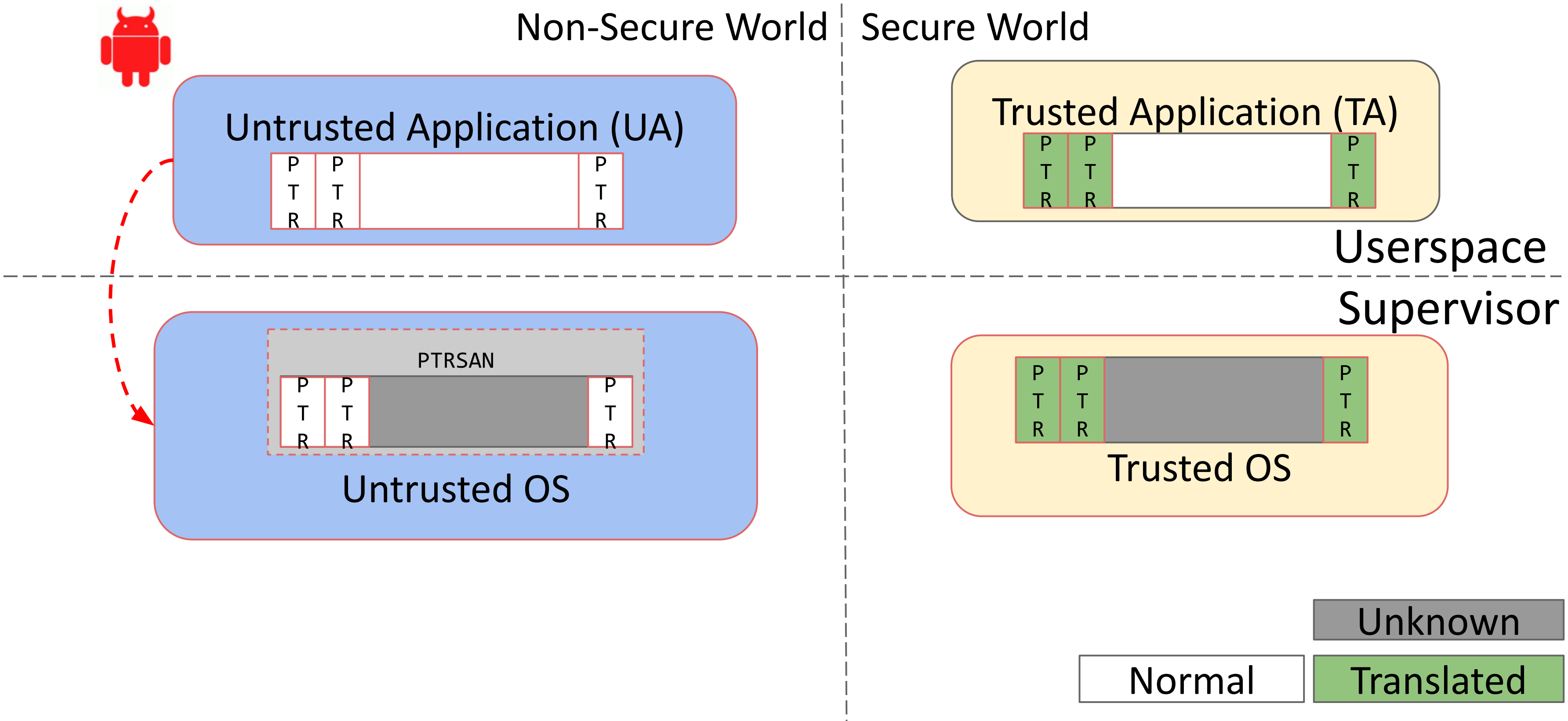
Handling untrusted pointers in trusted OS

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

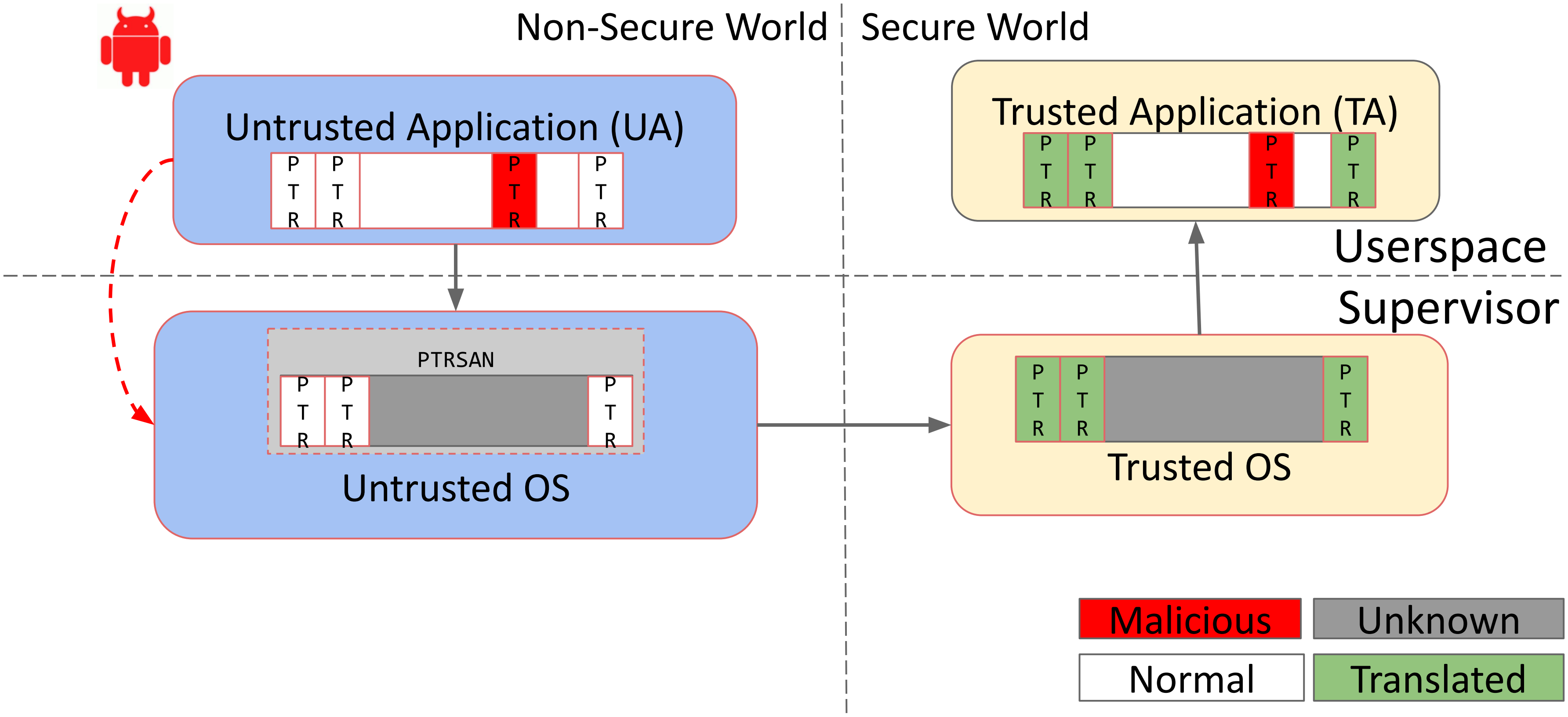
Semantic Gap

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

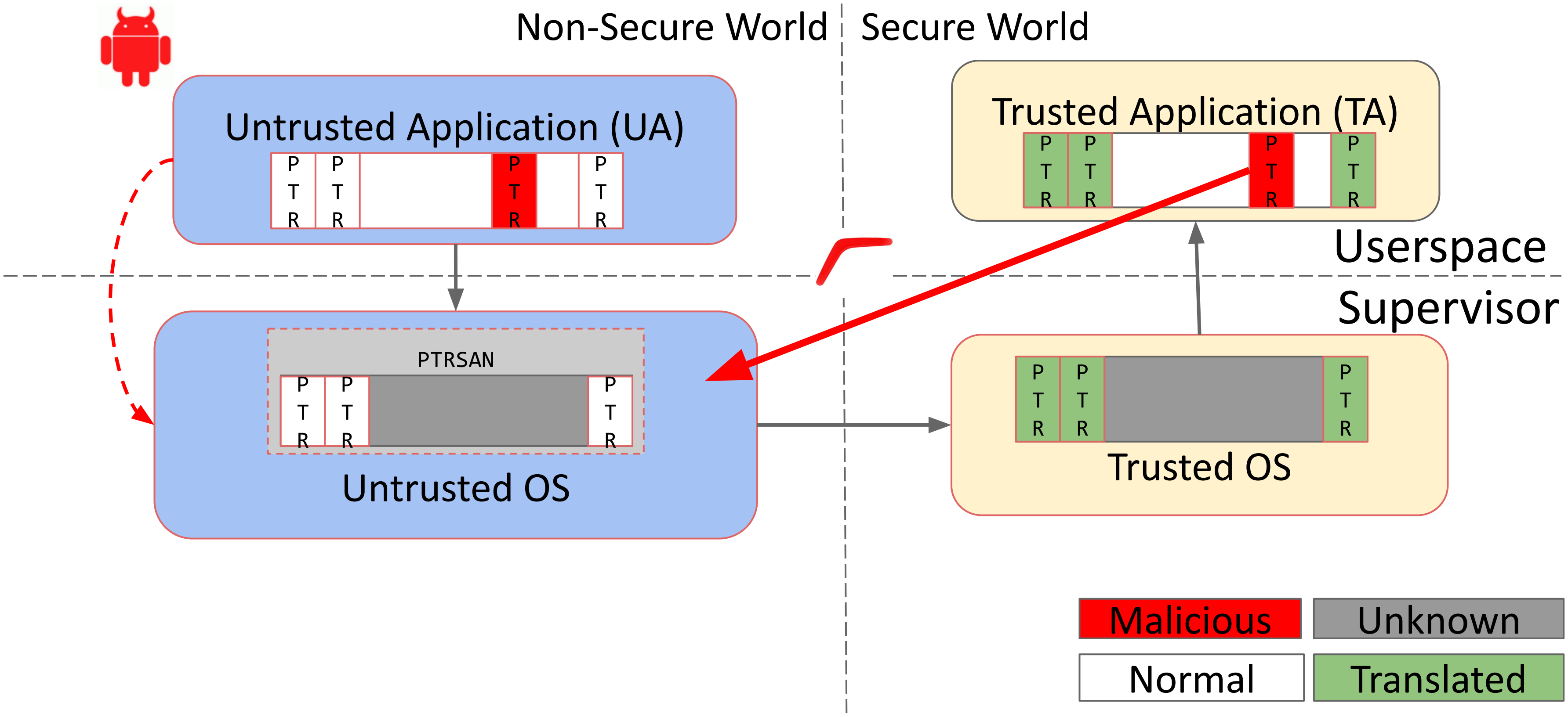
Bypassing Sanitization



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

*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 ($$$)

How to prevent Boomerang attacks?

Root Cause

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

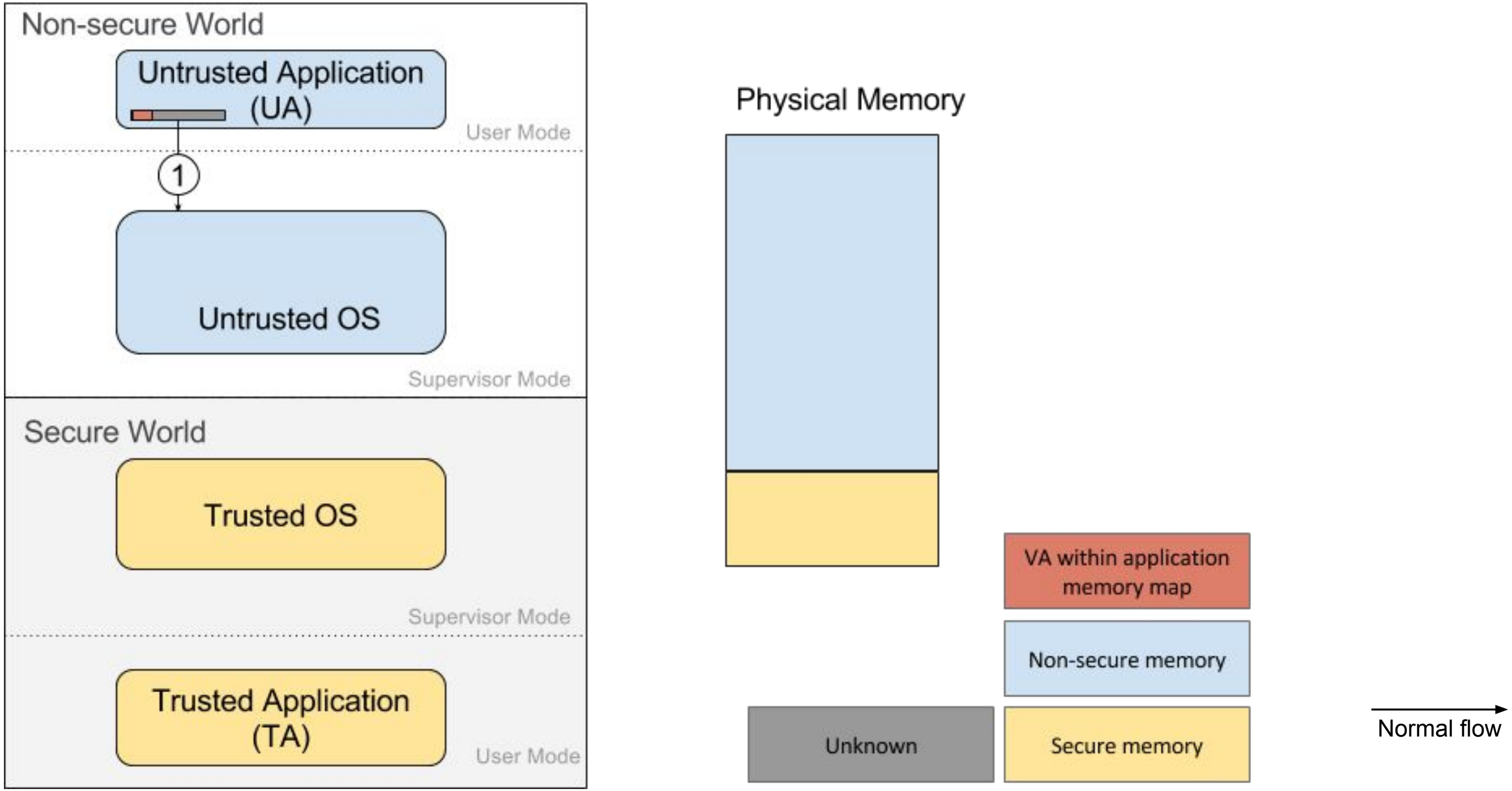
- Should have a mechanism for the TA (or TEE) to verify or bridge the semantic gap.

Cooperative Semantic Reconstruction (CSR)

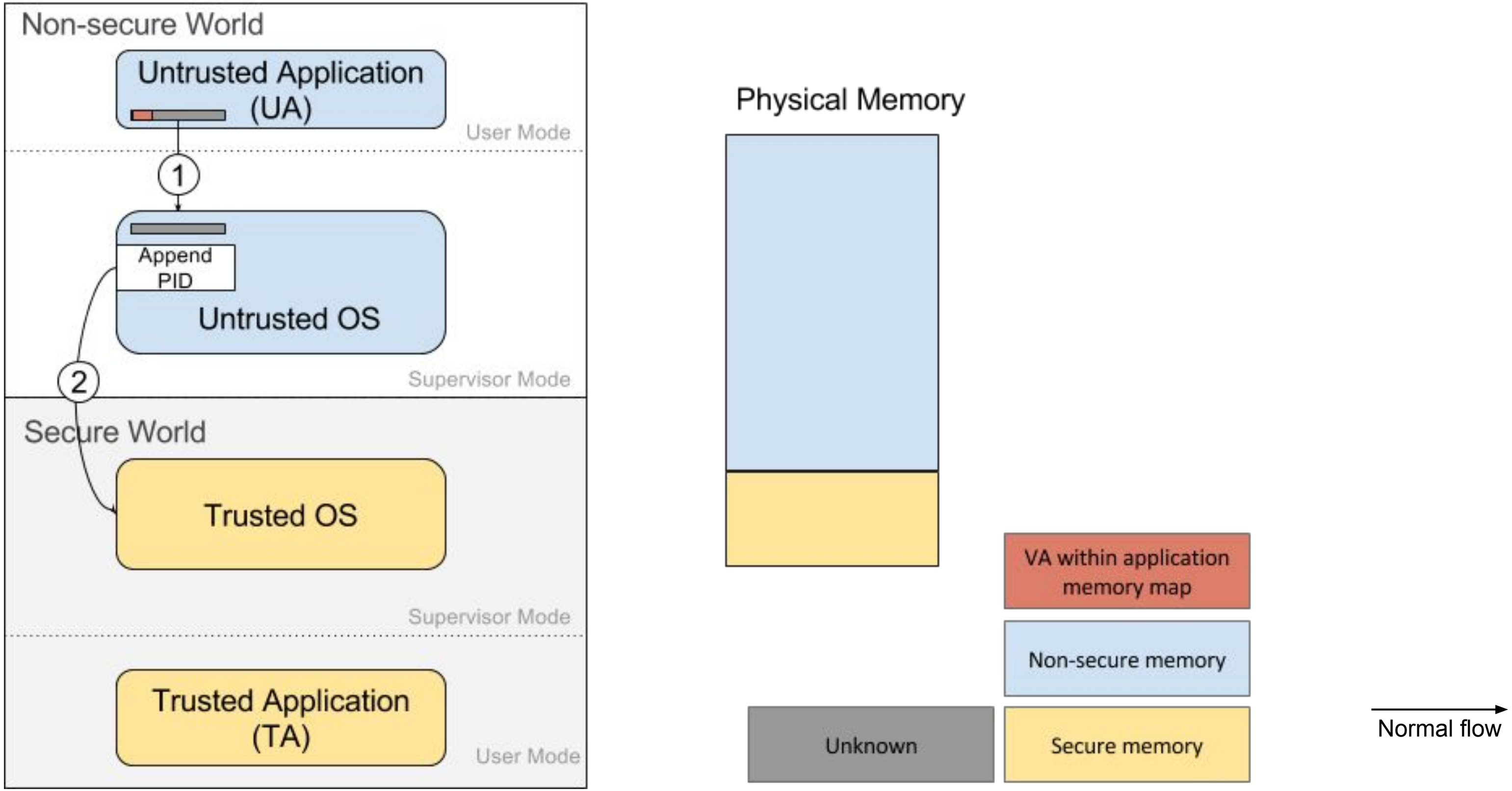
- Novel Defense proposed by us.

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

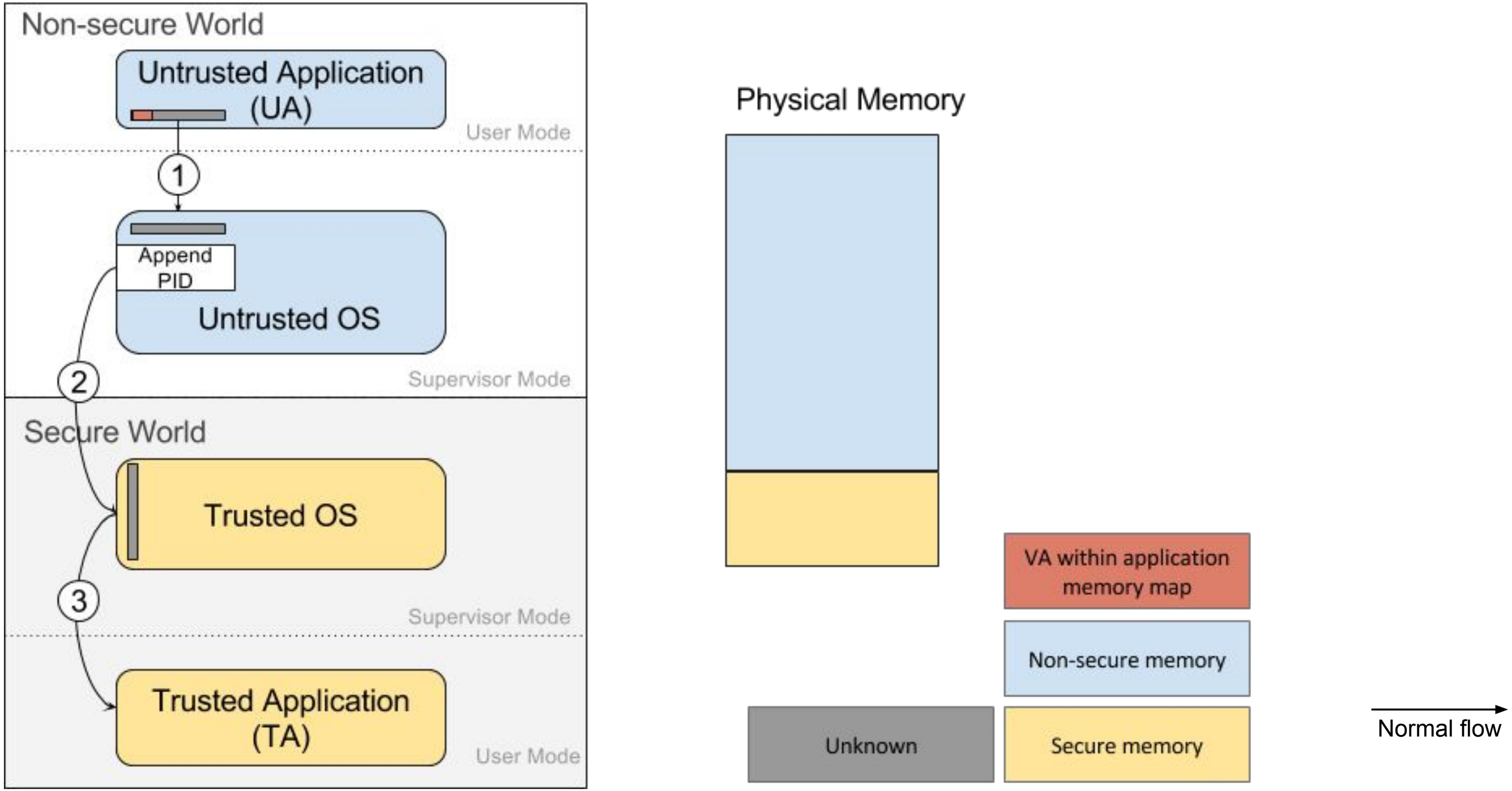
Cooperative Semantic Reconstruction (CSR)



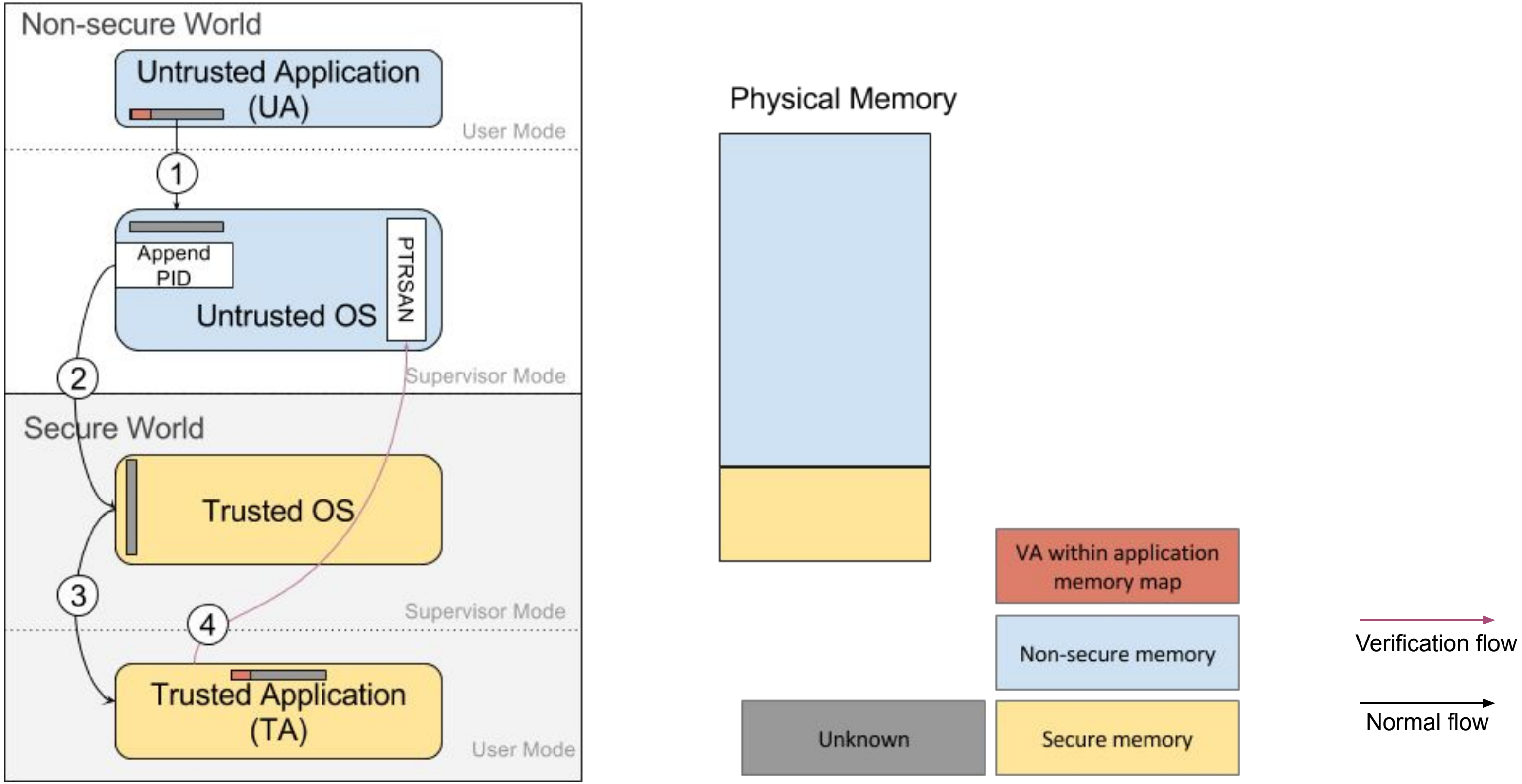
Cooperative Semantic Reconstruction (CSR)



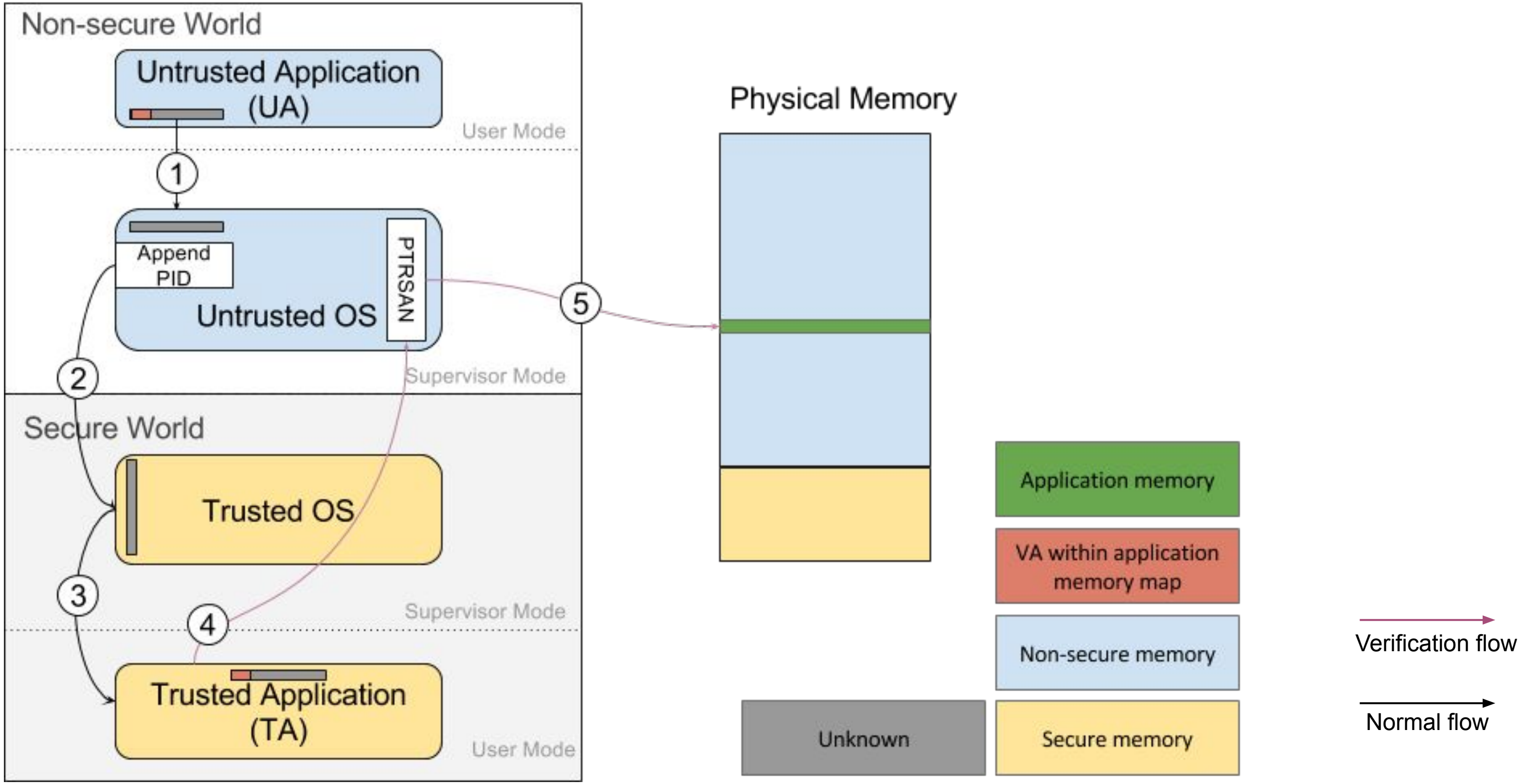
Cooperative Semantic Reconstruction (CSR)



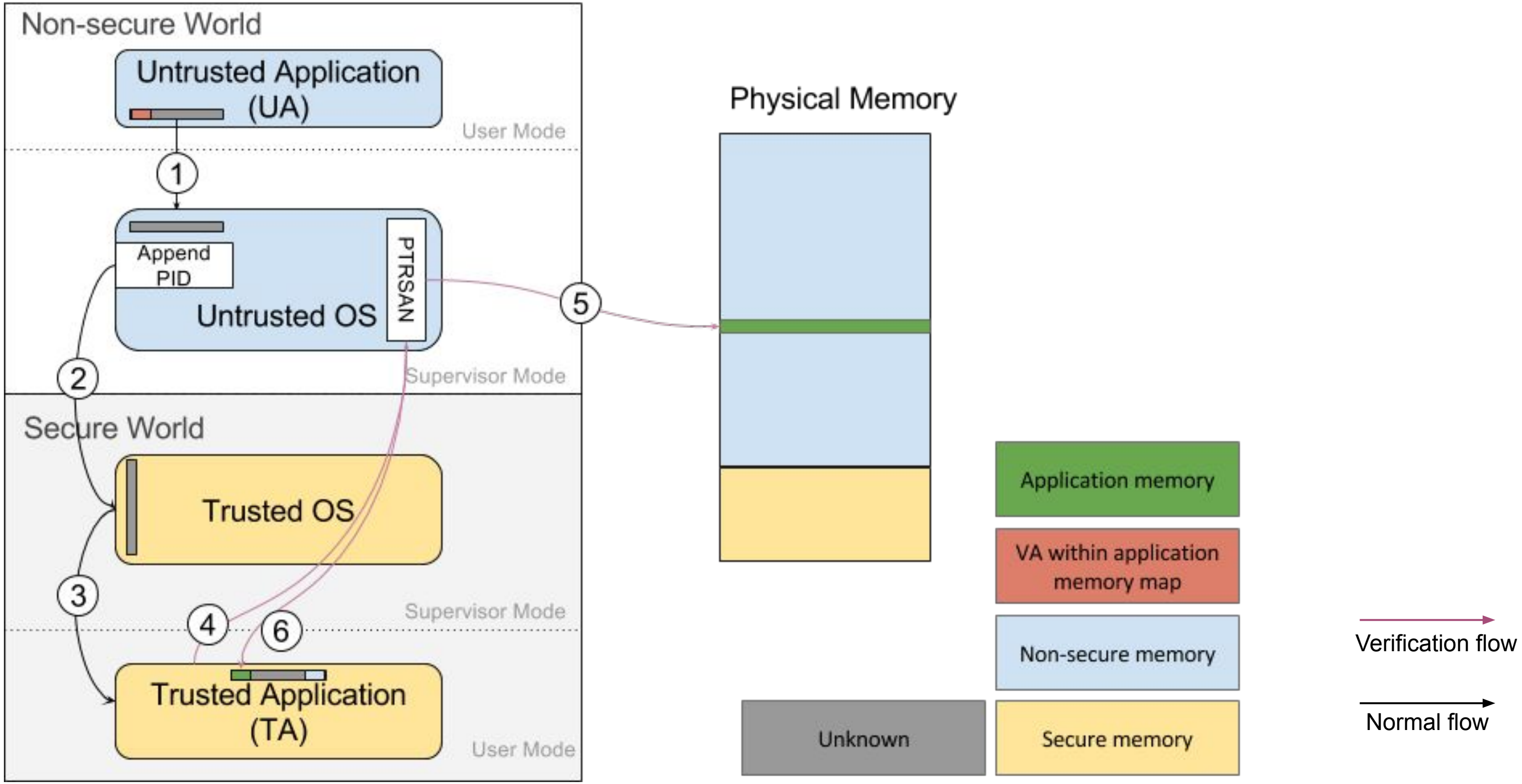
Cooperative Semantic Reconstruction (CSR)



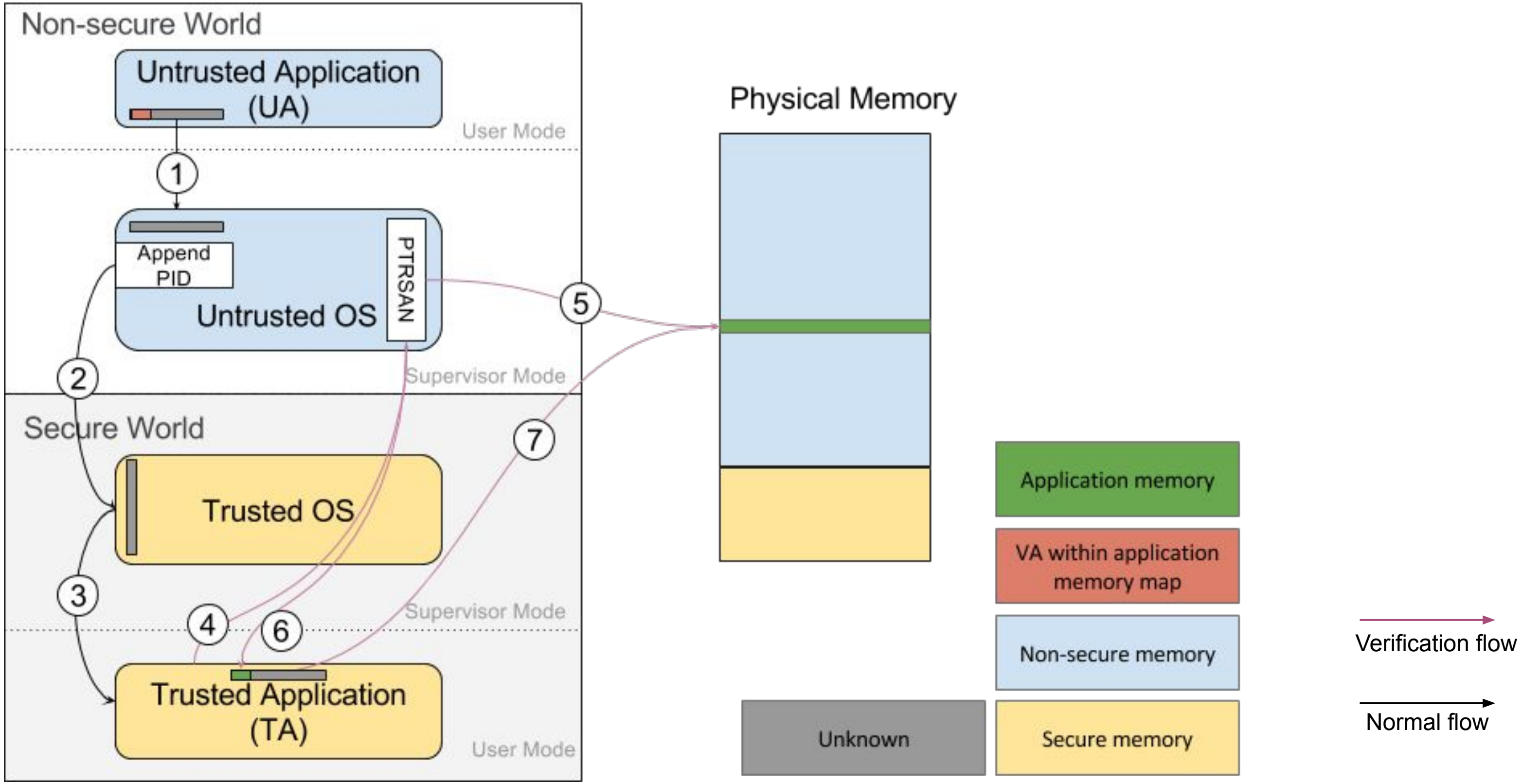
Cooperative Semantic Reconstruction (CSR)



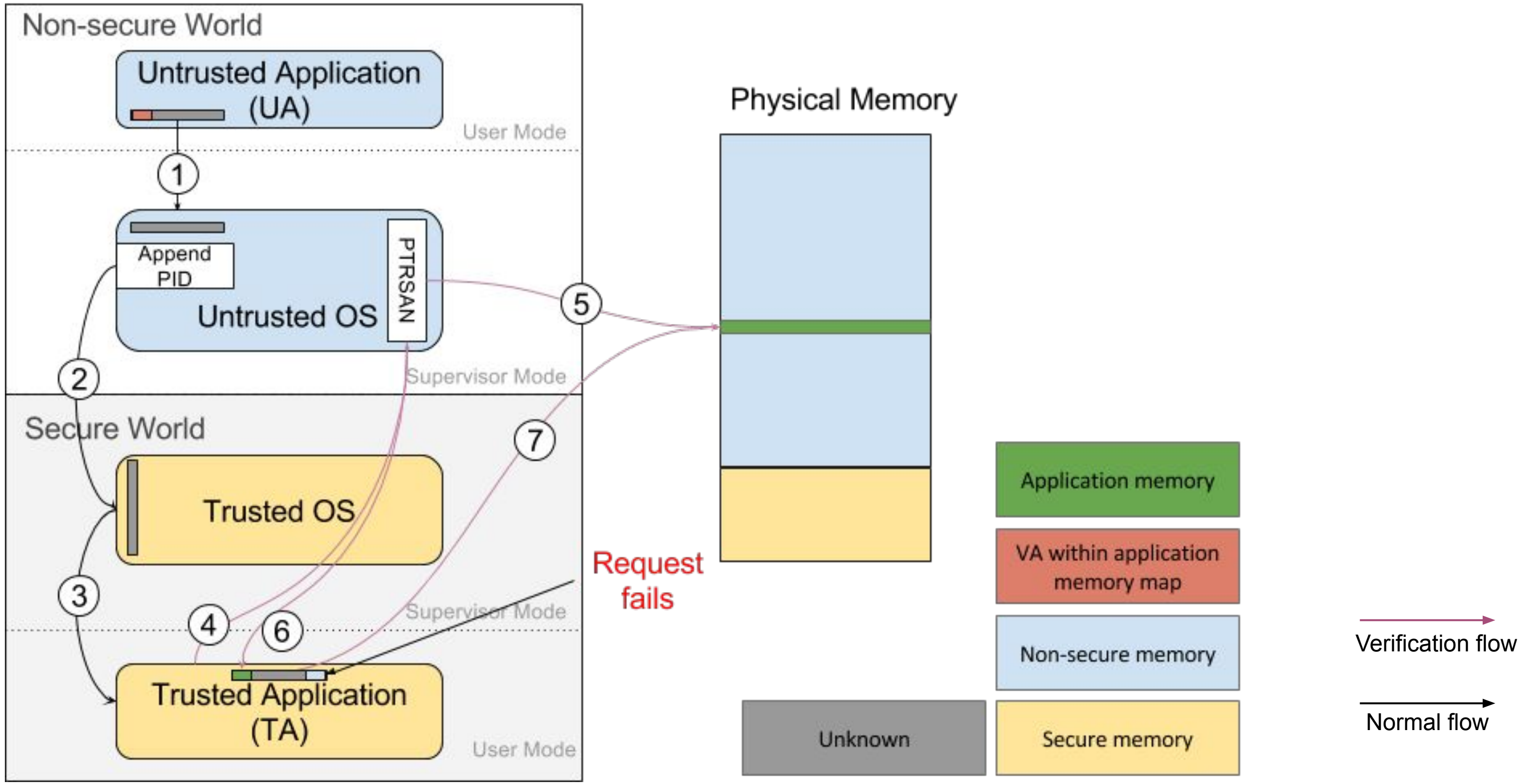
Cooperative Semantic Reconstruction (CSR)



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

- Microbenchmarks

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.

Remarks

- Neat trick -- not sure if this is a design flaw or an implementation flaw?
  - If PTRSAN is implemented correctly -- we can fix this! Right?
  - We could have these kind of flaws in any multiprocessor environments -- embedded systems.

- What happens in multi-core processor when the application changes its mapping AFTER translation?

- Require OS modifications!

Conclusion

- ✓ Boomerang: New class of bugs
- ✓ Good work.
- ✓ The problem can be generalized to other multiprocessor environments as well.

?

Backup

Automatic detection of vulnerable TAs

- Recover CFG of the TA
- Paths from the entry point to potential sinks
- Output the trace of Basic Block addresses
- Implemented using angr

Cooperative Semantic Reconstruction (CSR)

- Untrusted OS sends application id (e.g., pid) along with the request to Trusted OS.
- Raw pointers with application virtual address (VA) are passed directly to Trusted OS.
- TA or TEE consult untrusted OS to get the physical address corresponding to the VA of the pointer using application id (i.e., pid).