

ASSURE

Runtime Tracing

Meni Orenbach

NVIDIA

ASSURED Webinar

Online | 20th June 2023

www.project-assured.eu

Trusted Computing Base of Edge Devices

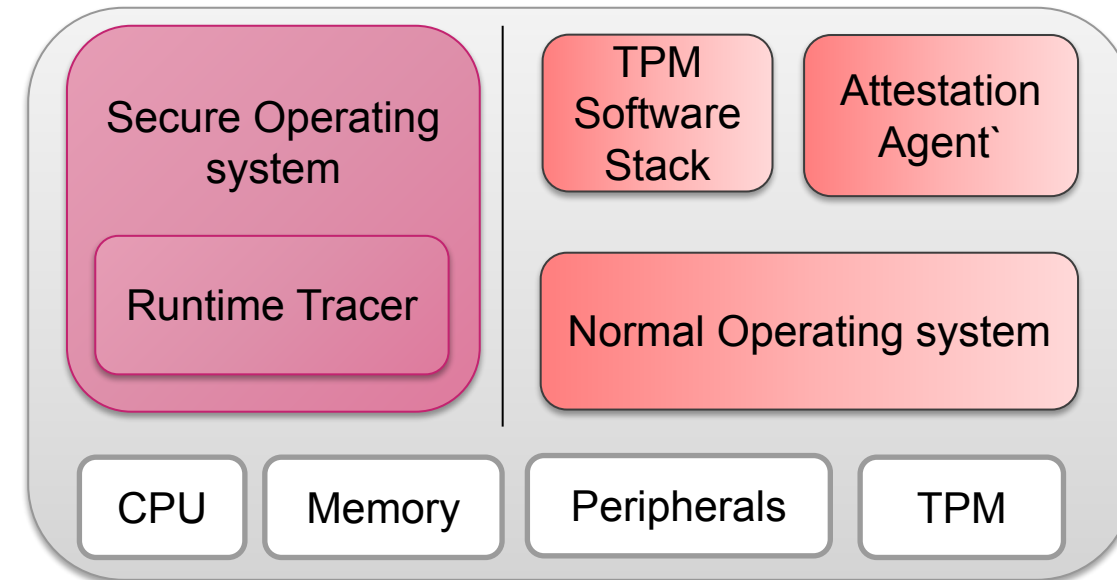
- Edge devices runs user services
- Unfortunately, services may contain vulnerabilities that can be exploited
 - E.g., buffer overflows
- Our vision in ASSURED
 - Detection rather than prevention
- How to maintain secure detection after an exploit?
 - Run security critical services in a Trusted Execution Environment (TEE)
 - Tracer prototype uses ARM TrustZone
 - Maintain a minimal TCB
 - Hardware
 - Firmware
 - Tracer & TEE Operating System
- Note, for wide adoption and ease-of-use we also support isolation via traditional OS MMU configuration

ASSURE

Trusted
Computing Base

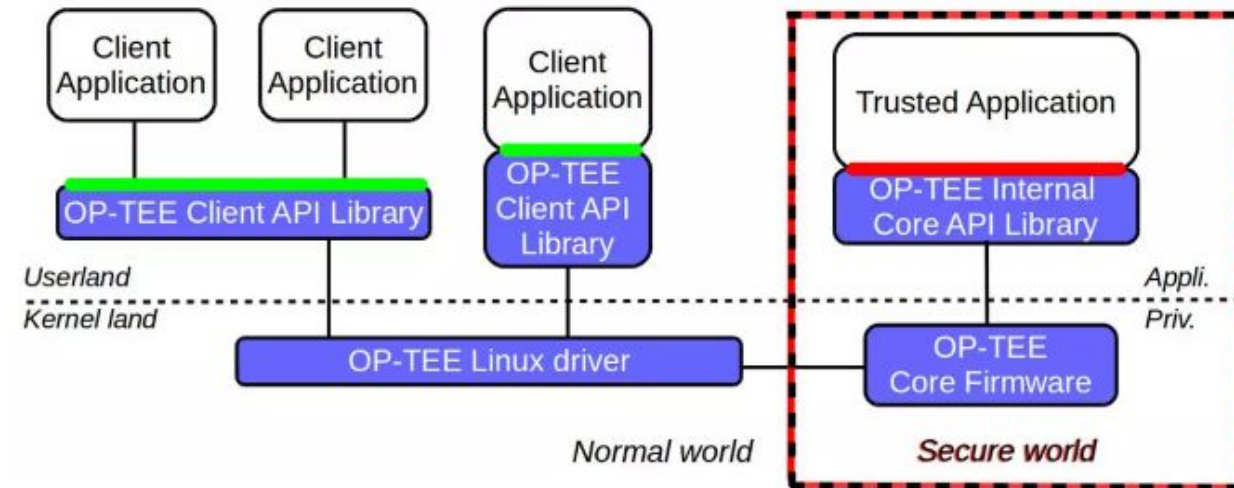
Untrusted

Hardware



TrustZone & OP-TEE Overview

- OP-TEE
 - Open-source TEE for ARM TrustZone
 - Support for many platforms¹
- Client opens session towards trusted application (TA)
 - TA identifies client and return session handle
- Client invokes TA commands
 - TA checks command ID and parses the parameters
 - TA executes the command
 - TA returns the output, status
- Client closes the TA session

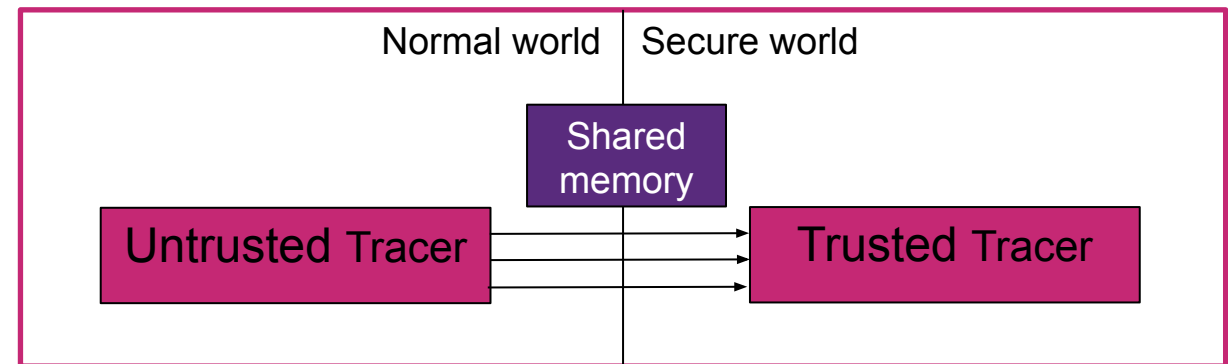


source: Build secure key management services in OP-TEE

[1] <https://optee.readthedocs.io/en/latest/general/platforms.html>

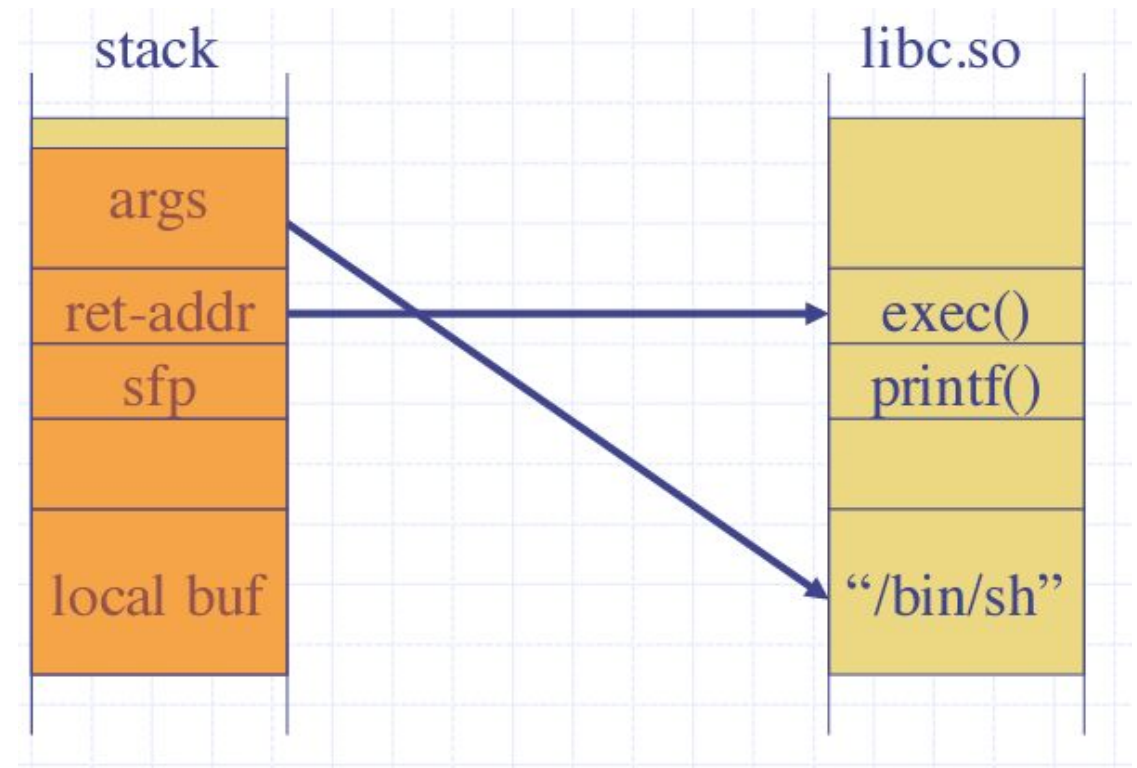
Tracer Operation in OP-TEE

- The tracer is partitioned to run in and out of the TEE: secure part and insecure part
- Untrusted tracer
 - Runs as regular process
 - Communicates with untrusted services
 - Attestation agent
 - User services
- Trusted tracer
 - Runs as a pseudo-TA: part of TEE OS
 - Performs isolated security critical operations
- Tracer maintains TEE security best-practices
 - Isolated memory, copy-to/from to avoid TOCTTOU attacks
 - Isolated OS services
 - Tracer secure-sensitive output is signed



Control Flow Hijacking Attacks


- Overflow attacks
 - Override function pointer/return address
 - Execution context hijacked to attacker-controlled flow
- Basis for different attacks
 - Return into libc
 - ROP, JOP, COOP



Data Oriented Programming Attacks

- Data-only attacks can also affect execution context
 - Change branches to taken/not-taken
 - Change number of loop iterations
- Control Flow Tracing can detect such attacks!

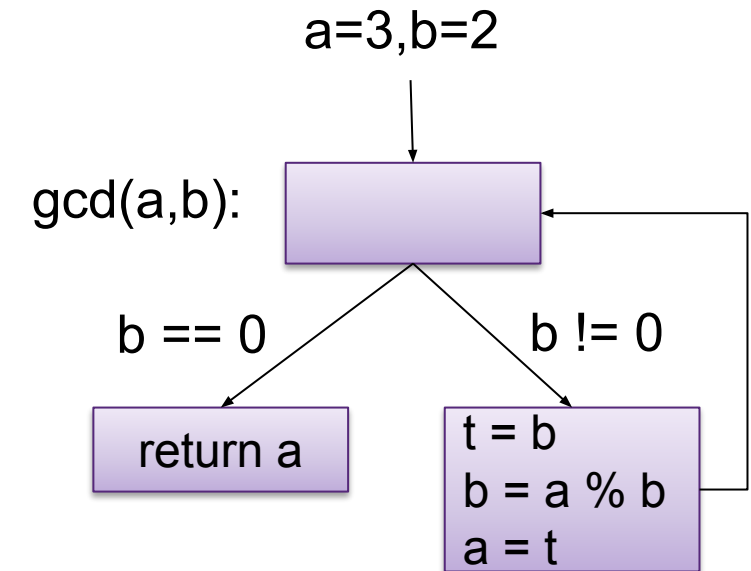
```
int number = 5;  
  
if (number > 0) {  
    // code  
}  
  
else {  
    // code  
}  
  
// code after if...else
```



Control Flow Tracing

- Programs execute in normal world
- Tracing granularity
 - Basic blocks
- Output: Ordered list of basic blocks

```
{ "0xffffffff3f9825c" : "no debug symbol" },  
{ "0xffffffff3f9827c" : "no debug symbol" },  
{ "0xffffffff3f983b4" : "no debug symbol" },  
{ "0xffffffff3f98448" : "no debug symbol" },  
{ "0xffffffff3f98488" : "no debug symbol" },  
{ "0xffffffff3f983a0" : "no debug symbol" },  
{ "0xffffffff3f9822c" : "no debug symbol" },  
{ "0xffffffff3f98248" : "no debug symbol" },  
{ "0xffffffff3f9825c" : "no debug symbol" },
```



List of control flows:

- `gcd_entry [a=3, b=2]`
- `gcd_b!=0 [a=3, b=2]`
- `gcd_entry [a=2, b=1]`
- `gcd_b!=0 [a=2, b=1]`
- `gcd_entry [a=1, b=0]`
- `gcd_b==0 [a=1, b=0]`

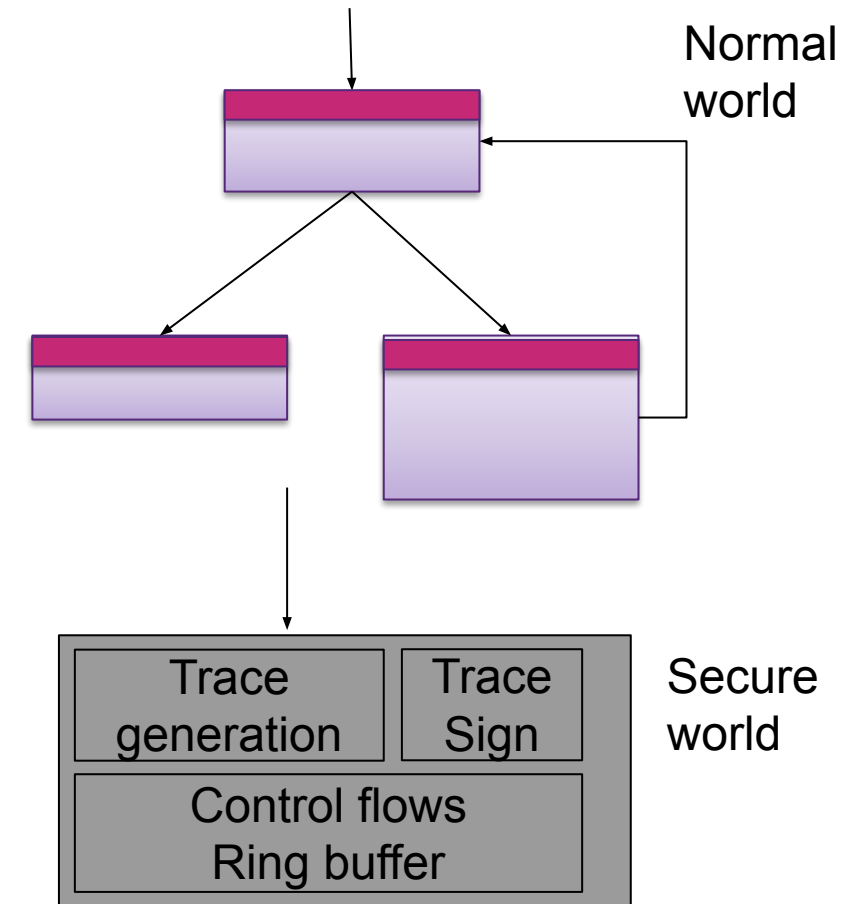
Control Flow Tracing Approaches

- Software: static vs dynamic rewriting
 - Static is usually most efficient than dynamic but
 - Limits interoperability
 - Requires manual modifications
- Hardware: hardware-assisted vs dedicated hardware
 - More efficient than software-only approaches
 - Limits usability and adoption
 - Dedicated hardware further limits usability and adoption

ASSURED Software Control Flow Tracing



- Prologue in each basic block (BBL)
 - Based on DynamoRIO framework
 - Log BBL address into a thread-local storage buffer
 - Once the buffer is filled
 - Send the context to the Trusted Tracer
- Attestation agent query latest traces
 - Trusted Tracer sends signed traces



<https://dynamorio.org/>

- Private key embedded in the tracer, available only to the secure world
- Tracer generates traces and signs a hash of them together with a per-request nonce
- Tracer can also delegate trust to the TPM
 - Verify traces signatures and sign the traces with TPM key
 - Enables adding support to using the TPM's attestation key additionally to the tracer private key towards remote attesting the traces authenticity.

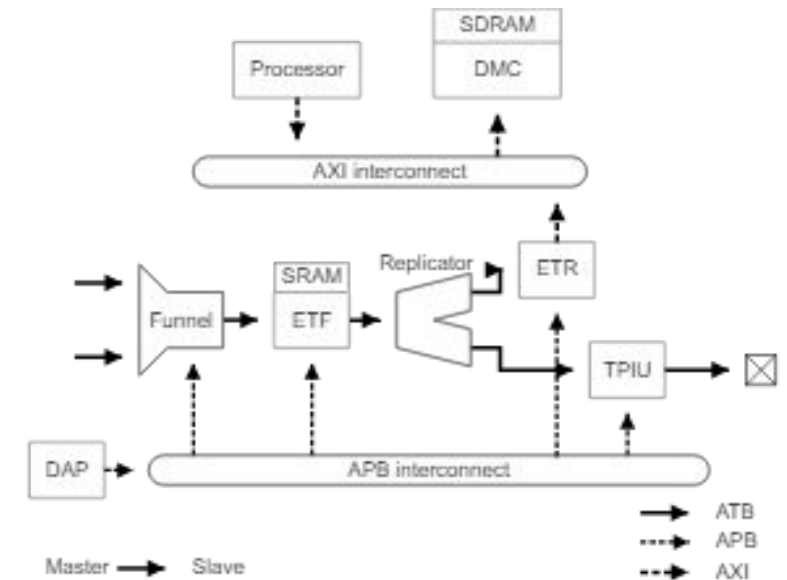
ASSURED Hardware-assisted Control Flow Tracing



Uses ARM Coresight ETM

Programming model

- Mode (FIFO)
- Status (Err, Empty, FtEmpty, TMCReady, Full)
- Control (TraceCaptEn)
- RAM Write Pointer & RAM Size
- RAM Read Pointer
- RAM Read Data



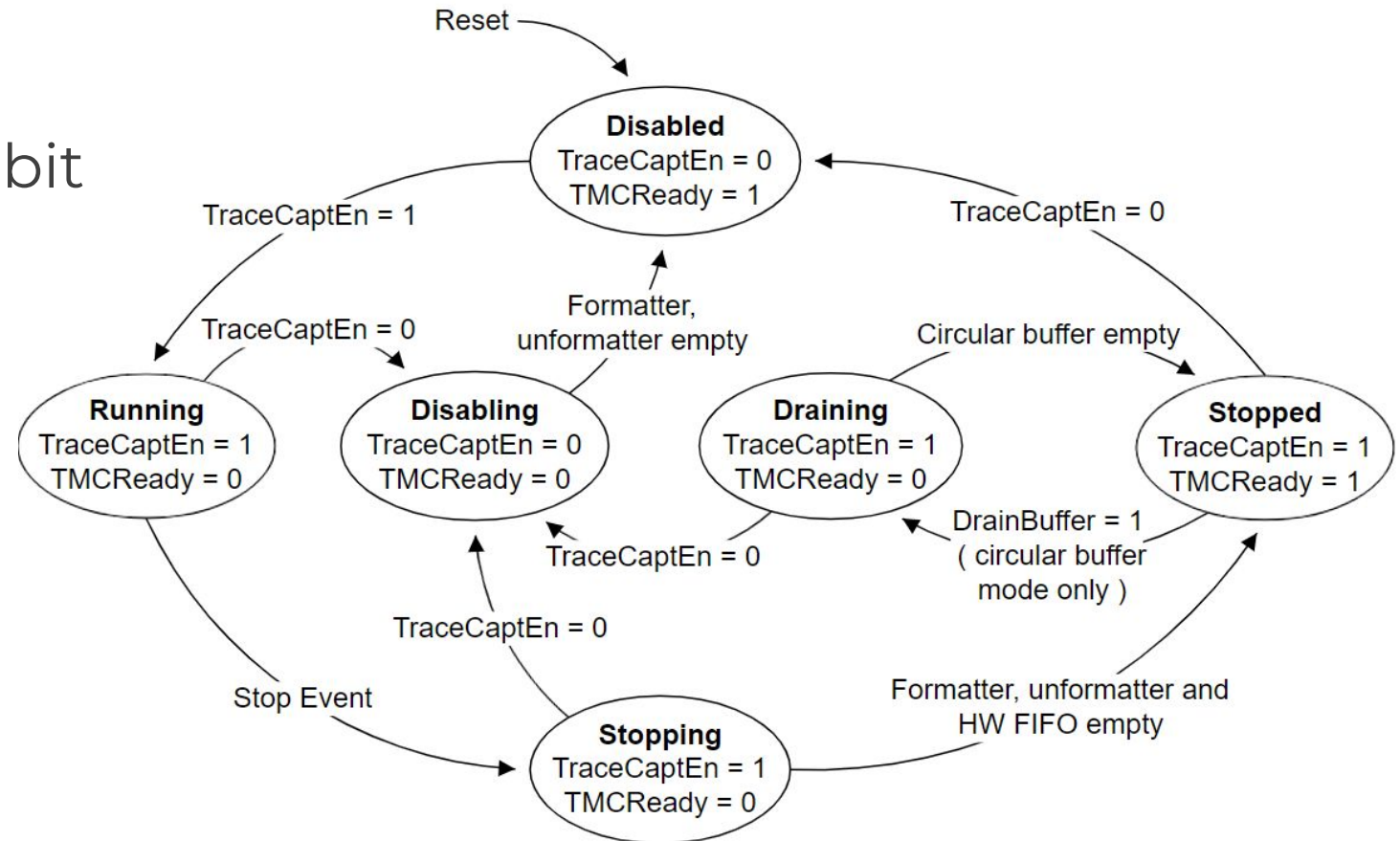
source: ARM Coresight Architecture

Writes to registers control tracing state machine

Trace Memory Controller state machine

Controlled by

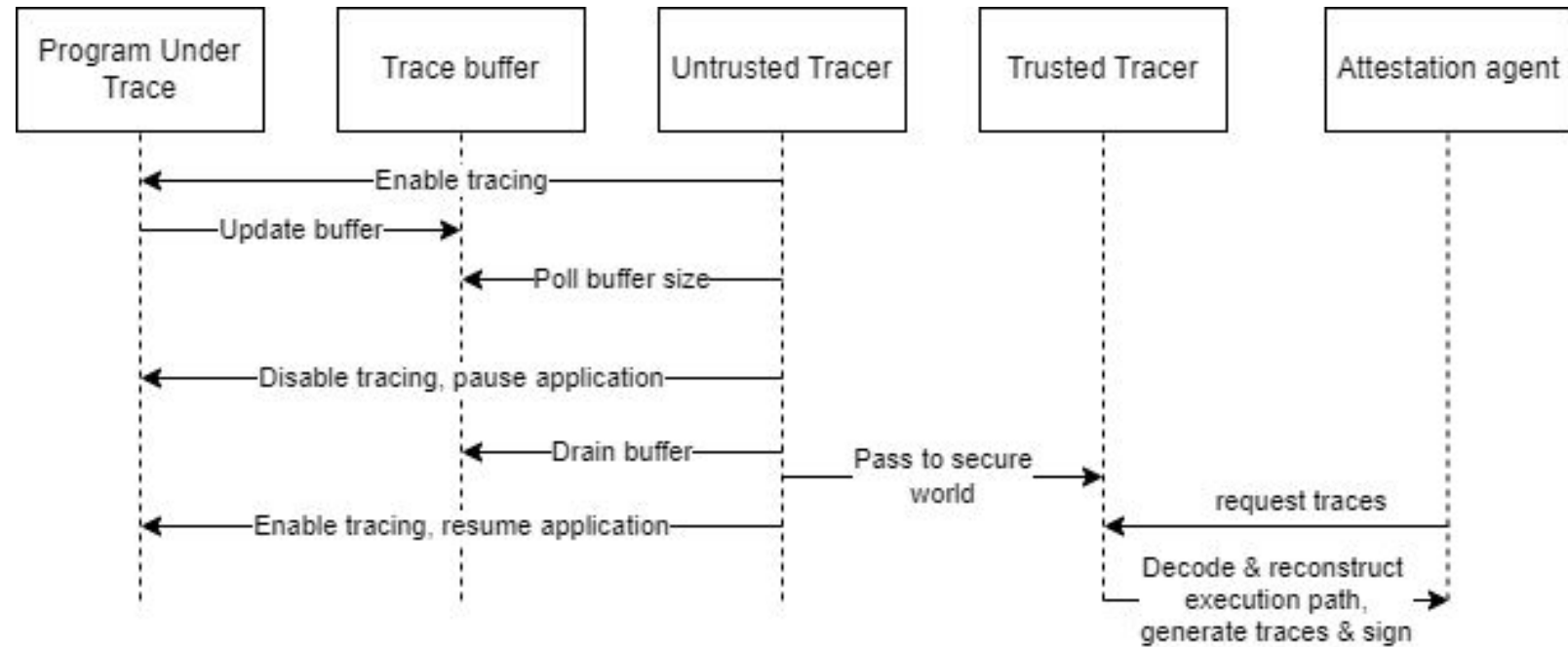
- CTL register
 - Trace capture enable bit
- STS register
 - TMCReady bit



source: ARM Coresight Architecture

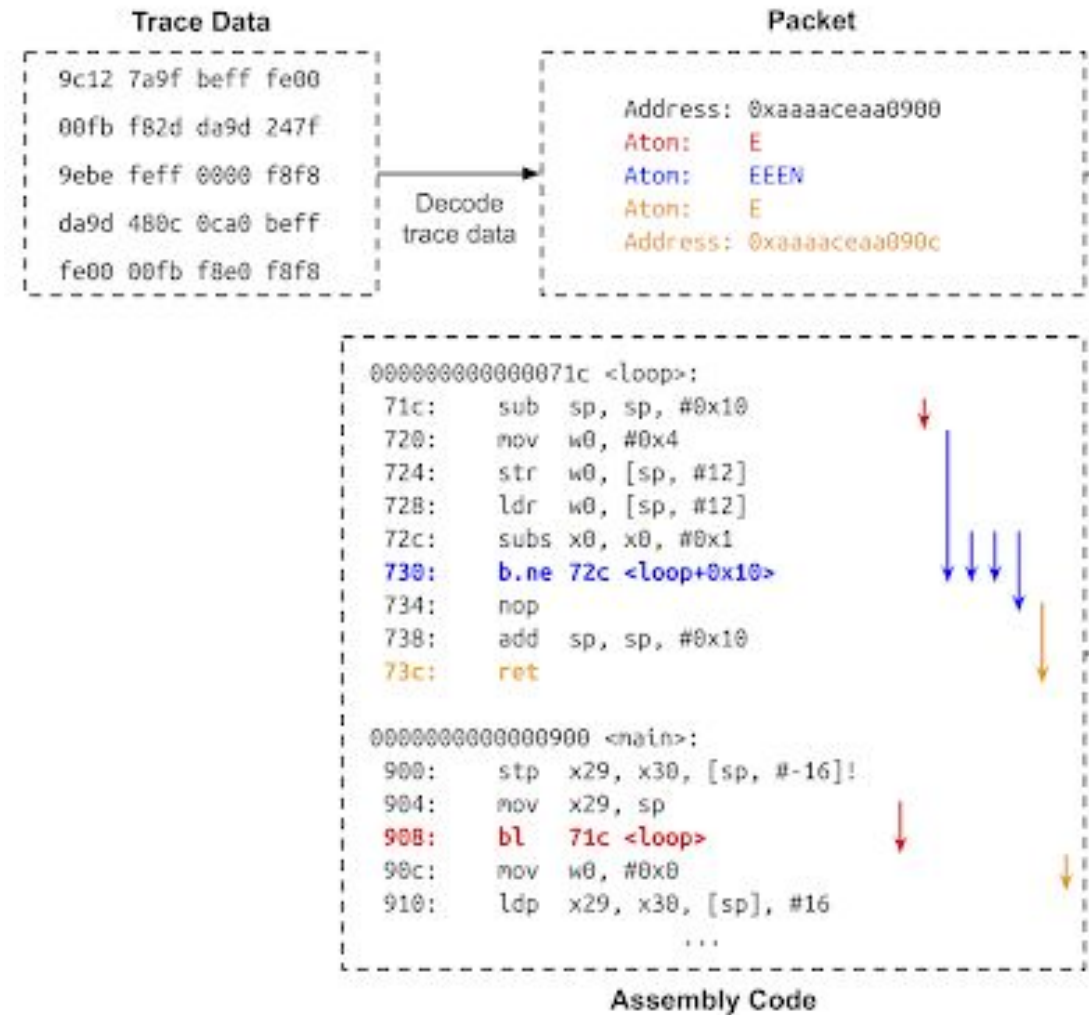
Control flow tracing

- Stored to physically contiguous buffer in kernel space
- Filters
 - Address ranges
 - PID



Coresight decode analysis flow

- Coresight traces are encoded
- Decoding done with ptm2human¹
- Recover complete execution path with program disassembly² & code analysis

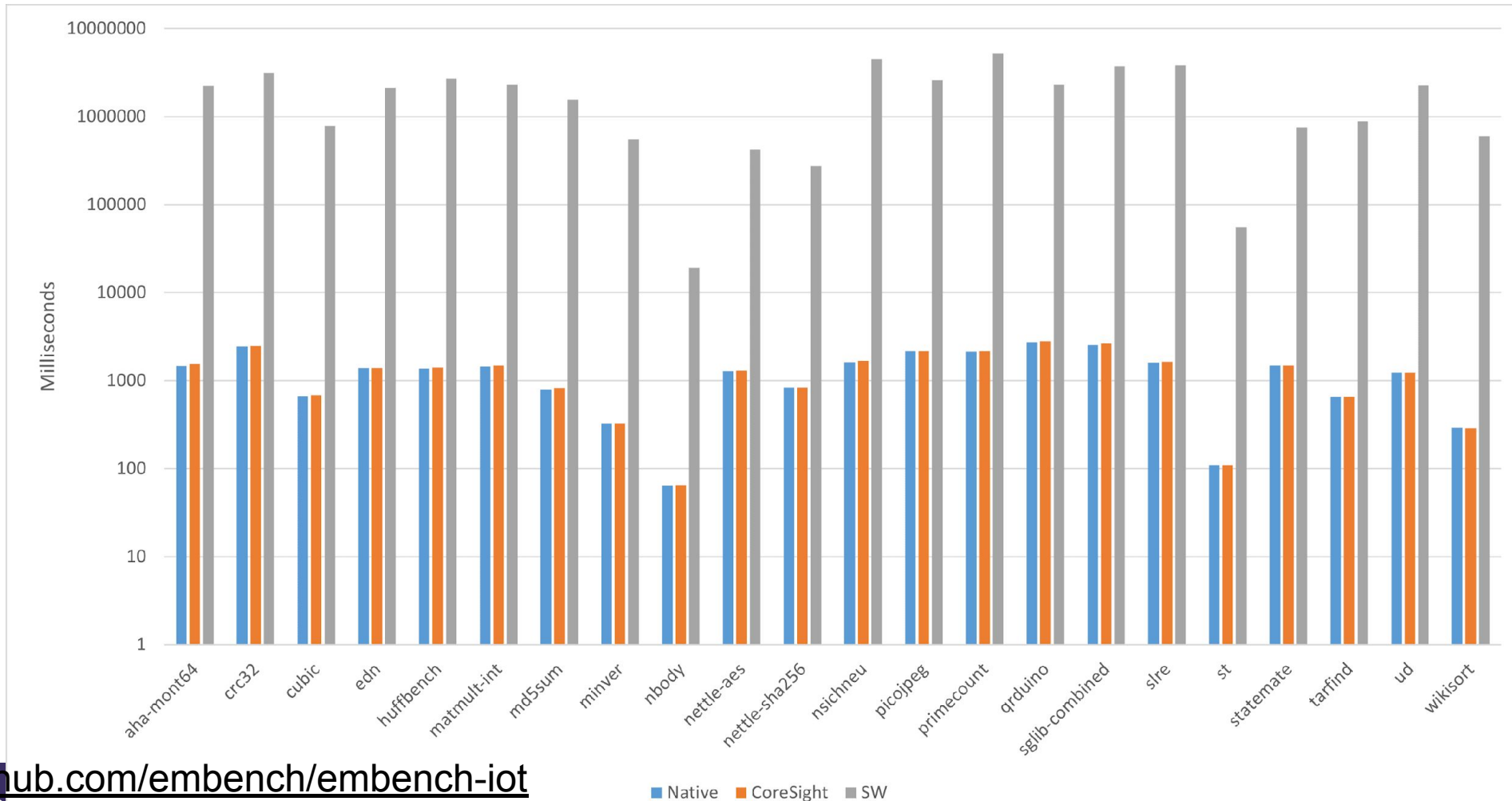


[1] <https://github.com/hwangcc23/ptm2human>

[2] <https://github.com/capstone-engine/capstone>

Comparing Tracing Approaches

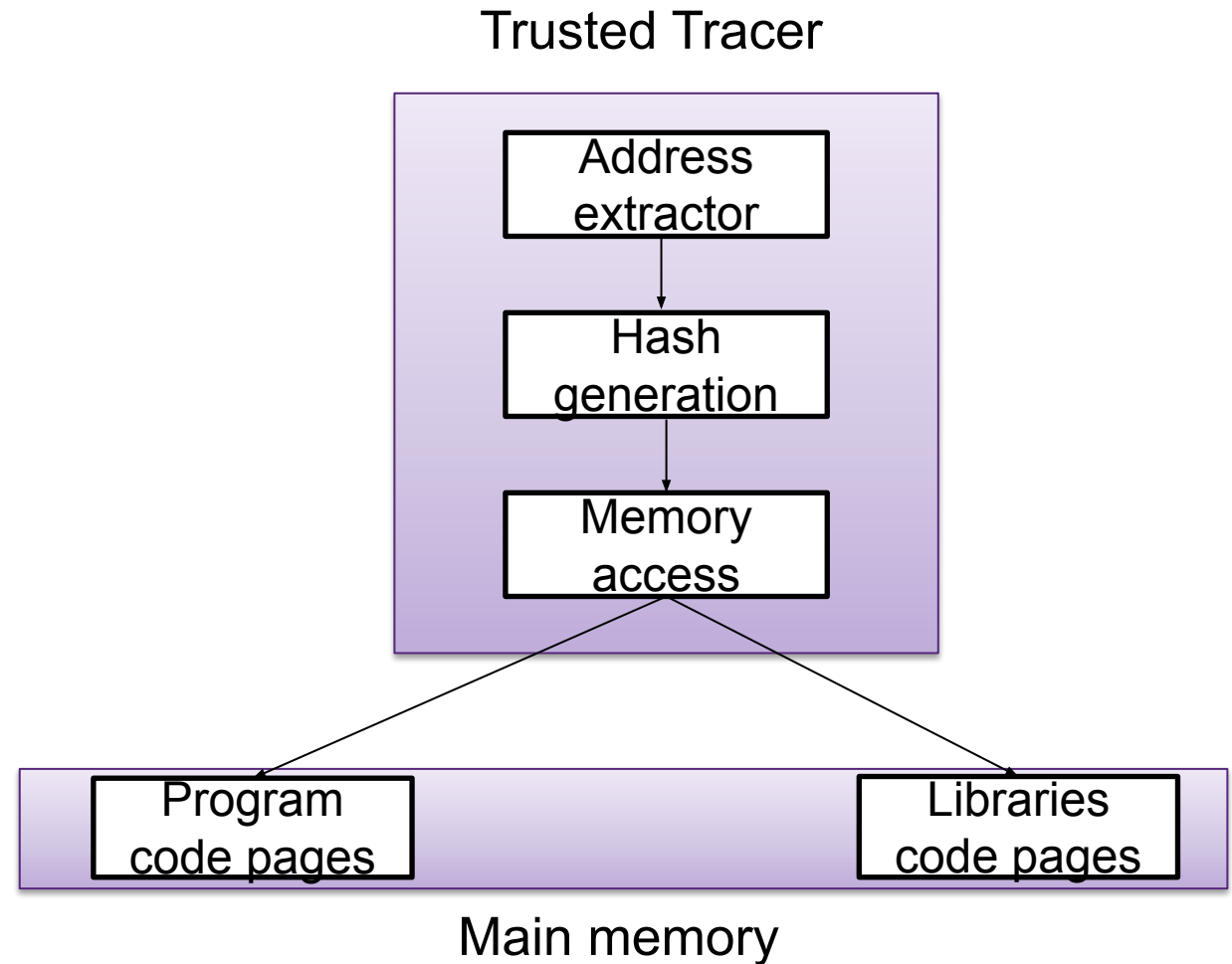
- Measured with Embench-IoT benchmark suite



- Linux Integrity Measurement Architecture (IMA)
 - Chain of trust – measure files before accessed/executed
 - Store measurements in kernel list
 - Extend measurements into TPM
 - Attest all measurements to third party
- Key idea: extend this approach into runtime integrity tracing
 - Measure in-memory processes/libraries
 - Mitigate memory-only attacks

Runtime configuration tracing

- Programs execute in normal world
- Upon attestation agent request
 - Request: process identifier, invalid bytes
- Trace in-memory representation of programs/libraries
 - Based on memory forensics techniques analyzing OS-related data structures
- Challenges
 - How to access memory?
 - How to handle unmapped pages?
 - How to get reference attestation value?

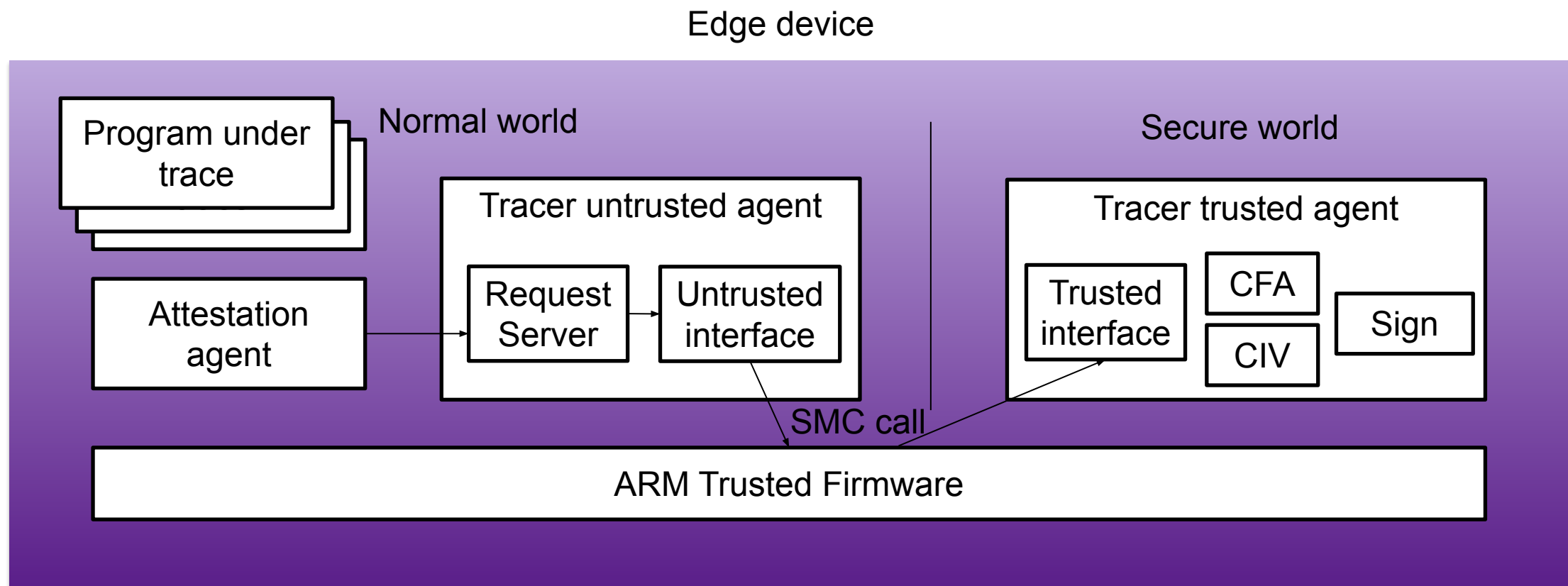


Runtime configuration tracing

- How to access the main memory?
 - With OP-TEE, the tracer runs as part of the OS and manipulates the MMU to map normal world pages for read access
 - Without OP-TEE, the tracer utilize a kernel module ¹
- Kernel symbols' virtual addresses embedded in the secure world, thereby enabling
 - Page table discovery and physical-virtual translations
 - Accessing specific memory regions to infer semantic information to detect code pages for libraries/processes
- How to handle unmapped pages?
 - The tracer pins traced process pages in memory
- How to get reference attestation value?
 - The tracer computes all offsets that may be legitimately changed by the dynamic loader and ignore them when computing the overall hash
 - The tracer computes a reference golden hash based on ELF parsing

[1] <https://github.com/NateBrune/fmem>

Putting it all together



Demo Time



ASSURE

Thank you



PROJECT-ASSURED.EU



@Project_Assured



ASSURED project is funded by the EU's Horizon2020 programme under Grant Agreement number 952697