Leveraging Tracing For Efficient Attestation

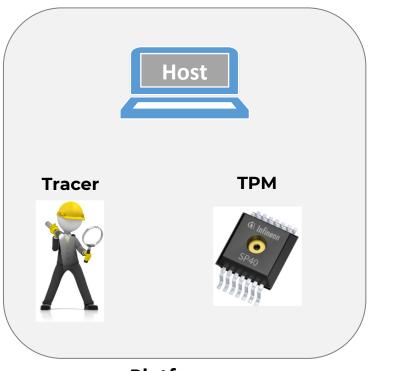
Stefanos Vasileiadis Security Engineer, Security Group

UBITECH Ltd

ASSURED WEBINAR

Webinar | 20 June 2023

Trusted Computing Base



Platform

Challenges

- Trusting the Tracer
 - OPTEE ARM Trustzone
- Semanting Trust between the Tracer and the TPM
 - ASSURED

Integrity of Tracer / Authentication of Traces

- The Tracer is responsible for continuously monitoring the processes executed in the device it belongs to, and collects information that is required in the context of the attestation schemes.
- This information can include control flow graphs used in Control-Flow Attestation (CFA), and hashes of configuration properties used in Configuration Integrity Verification (CIV).
- The Tracer is executed as a user space program and needs to be added to our Trusted Computing Base.
 - In order to prove the validity of the measurements that the TPM receives from the Tracer and uses in the context of the implemented attestation protocols, we employ a Pre-Installed Key
 - This key will be used to send signed traces to the Verifier, who is responsible for verifying their integrity.
 - Protocol securing the integrity of the reported traces. Protects against replay attacks and impersonation attacks, and ensures the integrity of the traces during correct protocol execution.

Traces Integrity && Authentication

- The TPM shares with the Tracer the Attestation Key's name.
- For every invocation of the Tracer the TPM creates a fresh policy session and shares the session nonce.
- The Tracer then calculates an authorization digest which includes the session nonce, the Attestation Key's name and the hashed traces.
- Tracer Signs the authorization digest with its private key.
- The TPM loads the Tracer's Public Key and executes TPM2_policySigned with the policy session that was created specifically for this challenge.

TPM Verifier Tracer sk_{TPM}, pk_{TPM} sk_T, pk_T, N (Traces) pk_T P = DefinePolicy(1)TPM2_CreatePrimary(P) $K_{h_{TPM}} \rightarrow (sk_{TPM}; pk_{TPM})$ K_{hTPM}, K_{TPM} TPM2_StartAuthSession $S_p \rightarrow \text{fresh } S$ $\xrightarrow{S_p}$ $H_N = H(N)$ $H_a = \text{ComputeAuthorizationDigest}(K_{TPM}, H_N)$ (2) $\sigma_{H_a} = \text{Sign} (H_a)_{sk_T}$ TPM2_LoadExternal(pk_T) $K_{h_T} \rightarrow pk_T$ K_{h_T}

TPM2_PolicySigned(Params, σ_{H_a}, K_{h_T})

 $\begin{array}{l} H_{a_{TPM}} = \text{ComputeAuthDigest(Params)} \textcircled{2} \\ S = H(S \mid\mid CC \mid\mid K_{h_{T}} \rightarrow name) \\ \Longleftrightarrow \text{VeritySignature}(H_{a_{TPM}}, \sigma_{H_{a}}, K_{h_{T}}) \\ S \rightarrow \text{CpHash} = \text{Params} \rightarrow \text{CpHash} \end{array}$

TPM2.Sign (H_N, K_{h_TPM}, S_p)

 $\begin{array}{l} \mathsf{CpRef} = \mathsf{ComputeCpHash} \textcircled{3} \\ \sigma_{TPM} = \mathsf{Sign} \left(H_N, K_{h_{TPM}} \right) \\ \Longleftrightarrow \quad CpRef == S \rightarrow CpHash \\ \land S == K_{h_{TPM}} \rightarrow Policy \end{array}$

 $\xrightarrow{\sigma_{TPM}}$

 $\sigma_{TPM}, K_{TPM}, N, \sigma_{Ha}$

 $\begin{array}{l} P_{ref} = {\sf DefinePolicy}(Pk_{t_{name}}) \textcircled{1} \\ A_{ref} = {\sf ComputeAuthorizationDigest}(K_{TPM}, H_N) \textcircled{2} \\ {\sf VerifySignature}(\sigma_{H_a}, A_{ref}, pk_T) \\ {\sf VerifySignature}(H(N), \sigma_{TPM}, K_{TPM} \rightarrow pk_{TPM}) \\ {\sf VerifyPolicy}(P_{ref}, K_{TPM}) \end{array}$

Tracing Techniques

eBPF

- Fully software based solution, leveraging kernel hooks.
- ASSURED Tracer
 - Fully software based solution.
- Intel PT
 - Pseudo hardware based <u>solution, requires</u> support from an intel CPU.
- Coresight Tracing

 Full hardware based

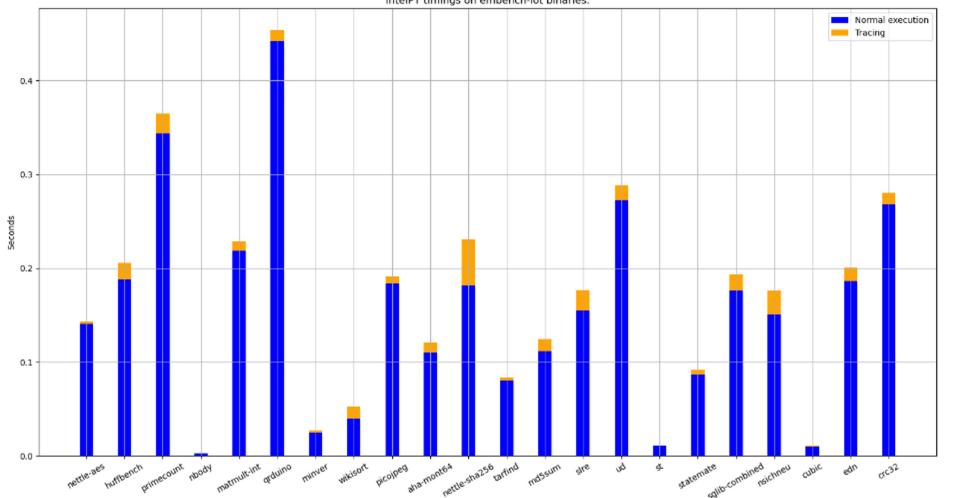
 solution.



ASSURE



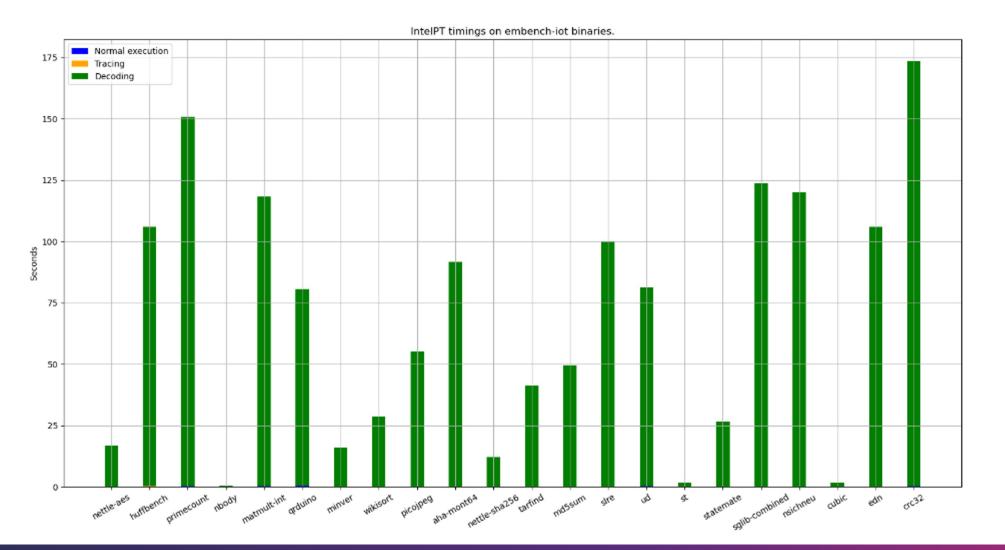
Intel PT Evaluation



IntelPT timings on embench-iot binaries.

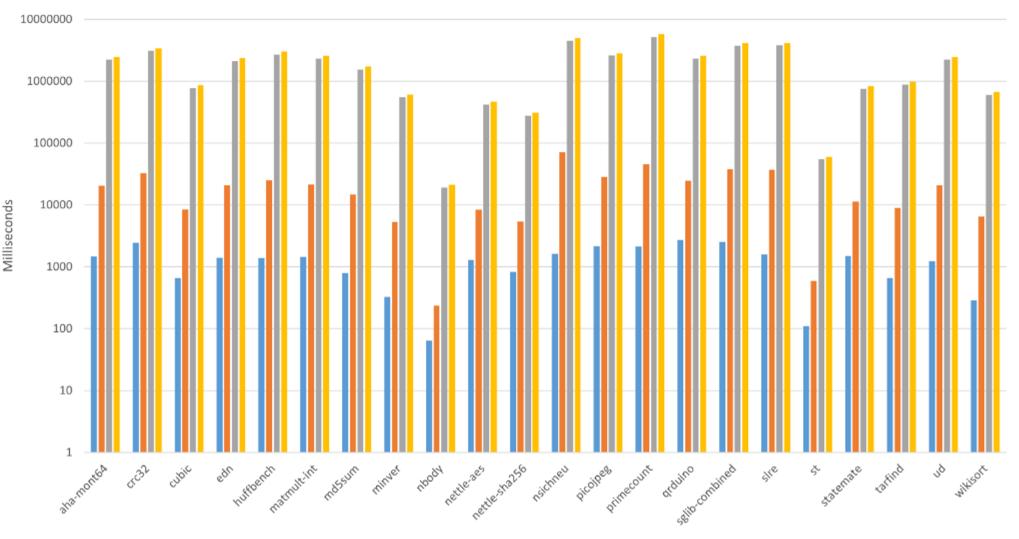
- Evaluated against a laptop that has an intel CPU and support pt tracing.
- Results Extracted for embench IOT applications.

Intel PT Evaluation



 Even though the pseudo hardware based solution adds just a small overhead on the execution of the application, it's really heavy to decode the traces.

ASSURED Tracer Evaluation



- When we are not batching or decoding the debug symbols we get really good results.
- There is no comparison between the coresight Tracer.
- Almost the same or even better in some cases with the pseudo hardware based solution (Inte PT).

Native SW_no_debug_batching SW SW_no_debug



THANKS







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