StreamBox-TZ: Secure Stream Analytics at the Edge with TrustZone

Heejin Park¹, Shuang Zhai¹, Long Lu², and Felix Xiaozhu Lin¹

¹Purdue ECE, ²Northeastern University

http://xsel.rocks
Scenario: Edge Processing for IoT

- Large telemetry data streams come from IoT sensors
  - 140M samples/day (smart grid), 1-2 TB/day (oil production)
- Edge processing is emerging
  - Cleanse and summarize data
- ARM processor as edge device – compelling performance at low power
Security Threats on Edge Processing

Commodity OS (Linux / Windows)

Libraries (libc, libstdc++, libssl)

Analytics Engine (Flink, Esper, Spark Streaming, SensorBee)

Edge

- Wide Attack Surface
  - Large and complex software stack (engine, libraries, and OS)

- High-value target
  - Data aggregated from multiple sources

Common IoT weaknesses

- Lack of professional supervision
- Weak configuration
- Delayed security updates
**Threat Model**

Powerful adversaries – control entire edge OS
How to securely process large streams of IoT data at the edge?

**Approach:** Isolate data and its computation within TEE, shielding them from the remaining edge software stack.
Background: ARM TrustZone

- Security extension for TEE enforcement

- Resource partitioning
  - Normal world and secure world
  - Limited secure world memory

- Trusted IO
  - Peripheral is owned by secure world
  - Direct access to secure world
Challenges

Design constraints:  
1. Minimal TCB  
2. Limited memory in TEE

Challenge 1) What functionalities should be in TEE and behind what interfaces?

Challenge 2) How to execute stream analytics with high throughput and low delay?

Challenge 3) As untrusted components are in analytics, how to verify the outcome?
Why Are Existing Systems Inadequate?

Engine and its libs in TEE

- Large and complex engine
- Potentially vulnerable Libraries

Partitioning as-is

- Unsuitable for existing engines
- Mismatch TEE’s limited memory

StreamBox-TZ

- How is our design different?
  - Solution 1) Architecting a data plane for protection
  - Solution 2) Optimizing a data plane performance within a TEE
  - Solution 3) Verifying edge analytics execution
Technique 1: Architecting a Data Plane for Protection

Data plane (trusted)
- Enclose all the analytics data and its computation
- Keep low TCB (5K SLoC)

Control plane (untrusted)
- Contains control functions (e.g. scheduling)

Narrow, shared nothing interface
- Only 4 entry functions
  (init; clean; entry point; debug)
Technique 2: Optimization – Parallel Execution In a TEE

Popular Stream Operators

GroupByKey, Windowing, AvgPerKey, Distinct, SumByKey, AggregateByKey, SortByKey, TopKPerKey, CountByKey, CountByWindow, Filter, MedianByKey, ...

Trusted Primitives

Sort, Merge, Segment, SumCnt, TopK, Concat, Join, Count, Sum, Unique, FilterBand, Median, ...

- **Trusted Primitives**: low-level stream algorithms
  - Building stream operators
  - Single threaded, stateless and oblivious to sync.
  - Sharing cache-coherent memory space in TEE → simplify data sharing/avoid copy cost
Technique 2: Optimization – Memory Mngt in a TEE

- **Unbounded array**
  - Data container for trusted primitives & operator states
  - In-place growing backed by on-demand paging inside TEE

- **Space efficient** – append-only buffer in a contiguous memory region
- **No relocation overhead** – large, dedicated virtual memory regions
- **Lightweight** – 0.7 KSLoC (9x fewer than malloc, 20x fewer than jemalloc)
Technique 2: Optimization – Low Ingestion by Trusted IO

- Source-edge link – **via OS**
  - Copy- and crypto-overhead

- Source-edge link – **via trusted IO**
  - Direct ingestion into TEE, bypassing OS

Copy and decrypt data before processing

Direct data ingestion in cleartext
Technique 3: Verifying Edge Analytics Execution

- Tracking the origin of result (data lineage) is insufficient – lack of steam semantics

- **Audit record**: generated by data plane when invoked
  - Data flow among primitives with stream model-awareness (e.g. window, watermark)

- **Cloud verifier**: replays all ingestion records
  - **Correctness** – all ingested data is processed correctly
  - **Freshness** – the pipeline has low output delays
Implementation & Evaluation

- **Implementation**
  - Control plane (written by C++, 12.4K SLoC newly added)
    - Reuse and modify code from StreamBox [ATC 17]
  - Data plane (written by C, 5K SLoC)
    - Newly implement form scratch

- **6 benchmarks of processing sensor data streams**
  - Filter, Power, WinSum, TopK, Distinct, and join

- **4 different versions of StreamBox-TZ**

---

<table>
<thead>
<tr>
<th>SoC</th>
<th>HiSilicon Kirin 620, TDP 36W</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>8x ARM Cortex-A53@1.2 GHz</td>
</tr>
<tr>
<td>Mem</td>
<td>2GB LPDDR3@800 MHz</td>
</tr>
<tr>
<td>OS</td>
<td>Normal: Debian 8 (Linux 4.4)</td>
</tr>
<tr>
<td></td>
<td>Secure: OP-TEE 2.3</td>
</tr>
</tbody>
</table>
Security Overhead: what is the cost of isolation?
Insecure: Running in Normal World

Native performance of StreamBox-TZ running in normal world

Insecure

Data Plane

OS

Ctrl Plane

Native performance of StreamBox-TZ running in

Throughput (MEvents/s)

Throughput (MB/s)

Filter (10ms)

Power (600ms)
Modest Security Overhead from Isolation

Throughput loss due to isolation

- Modest security overhead (less than 25%)
What is the impact of Trusted IO?
Impact of w/o Trusted IO

- 40 ~ 50% throughput loss without trusted IO
  - Data copy-, crypto-overhead

Performance impact from direct data (trusted IO) ingesting into TEE
Conclusion

- **StreamBox-TZ**: secure stream analytics engine at the edge
  - Exploit ARM TrustZone as TEE
  - Data-intensive, parallel computations on minimal TCB

- Compelling performance with strong security
  - Architecting **data plane** for protection
  - **Optimizing** data plane performance within a TEE
  - **Verifying** edge analytics execution
Questions?