Design of Access Control Mechanisms in SoCs with Formal Integrity Guarantees

USENIX Security Symposium ’23
2023-08-10

Dino Mehmedagić, Mohammad Rahmani Fadiheh, Johannes Müller, Anna Lena Duque Antón, Dominik Stoffel, Wolfgang Kunz
Threat Model

> Increasing need for SoCs with diversified hardware
> Third-party IPs → trust issues 😞
> SoC Access Control Mechanism
  > Domains: High-security vs low-security
  > Access control ensures that communication between domains doesn’t endanger security
Security Target

> **Operation integrity:**

> Forbidden information flow: _low_ security domain $\rightarrow$ _high_ security domain outputs

> **UPEC:**

> *Exhaustive* verification of information flow restrictions at the RTL

> Interval Property Checking (IPC)

> 2-instance (miter) model
UPEC for Operation Integrity

---

UPEC-OI:
assume
\[ t_0 \quad \text{high\_micro\_state}_1 = \text{high\_micro\_state}_2; \]
\[ t_0 \ldots t_k \quad \text{primary\_inputs}_1 = \text{primary\_inputs}_2; \]
\[ t_0 \ldots t_k \quad \text{access\_ctrl\_configured}_1; \]
prove
\[ t_k \quad \text{secure\_outputs}_1 = \text{secure\_outputs}_2 \]

How long does \( k \) need to be?

Too long!
Decomposing the Proof

UPEC-OI:
assume
\( t_0 \) high_micro_state_1 = high_micro_state_2;
\( t_0 \ldots t_k \) primary_inputs_1 = primary_inputs_2;
\( t_0 \ldots t_k \) access_ctrl_configured_1;
prove
\( t_k \) secure_outputs_1 = secure_outputs_2
\( t_k \) high_soc_state_1 = high_soc_state_2;
primary_inputs_1 = primary_inputs_2

secure_outputs_1 = secure_outputs_2

M1

SoC 1

S1

S2

S3

Access Ctrl

M2

high_micro_state_1 = high_micro_state_2

high_soc_state_1 = high_soc_state_2

M1

SoC 2

S1

S2

S3

Access Ctrl

M2

secure_outputs_1 = secure_outputs_2?
SoC 1

high_micro_state_1 = high_micro_state_2

primary_inputs_1 = primary_inputs_2

secure_outputs_1 = secure_outputs_2

S1
S2
S3

access control configured 1

SoC 2

high_soc_state_1 = high_soc_state_2

T-ALERT

secure_outputs_1 = secure_outputs_2
UPEC-OI Verification Methodology

> Induction-based approach to completely verify operation integrity
  > Base proof: Find all P-alerts and verify OI for a bounded time window $k$
  > Step proof: Use IPC’s symbolic initial state to fast forward to any future time point in which a T-alert can occur

> Additional optimizations
  > Sound blackboxing
  > Spatial, temporal decomposition, T-alert trigger expansion...
Case Study: OpenTitan
UPEC-Driven Design of Access Control

> Add malicious IPs to model the threat
> Equip SoC with access control mechanism in the interconnect
> Refine the access control mechanism through a UPEC-OI-driven design flow
## Case Study: Results

### UPEC-Driven Design of Access Control

<table>
<thead>
<tr>
<th>Overall design process</th>
<th>3 person-months</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of verify-patch iterations</td>
<td>19</td>
</tr>
<tr>
<td>Average property check time</td>
<td>~5 minutes</td>
</tr>
<tr>
<td>Longest UPEC-OI check time</td>
<td>11 hours</td>
</tr>
<tr>
<td>Peak memory consumption</td>
<td>25 GB</td>
</tr>
<tr>
<td>Design size</td>
<td>14 million state bits</td>
</tr>
</tbody>
</table>
Conclusion

> Developed a methodology to formally verify operation integrity:

  > Property formulation
  > Proof decomposition
  > Scalability and usability optimizations

> Case study shows: UPEC-OI is feasible for realistic SoCs

Thank you!

Questions?

Contact me at:
dino.mehmedagic@edu.rptu.de