Towards a “Design for Secure System” EDA tool

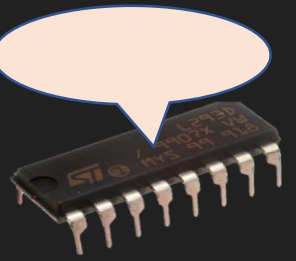
Prof. Avi Mendelson
CS department, Technion
avi.mendelson@Technion.ac.il

Agenda

- Our main current activities
  - RE related
- Why do we need an EDA tool for secure systems
- Possible implementation directions
- Discussion

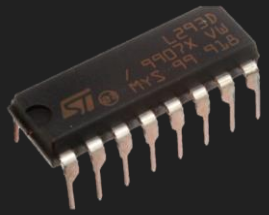
Current main research direction:

Subgraph Matching for Hardware Reverse Engineering

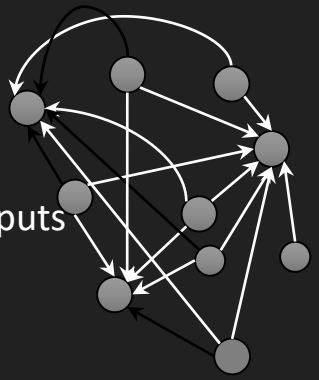


Technion Israel Institute of Technology

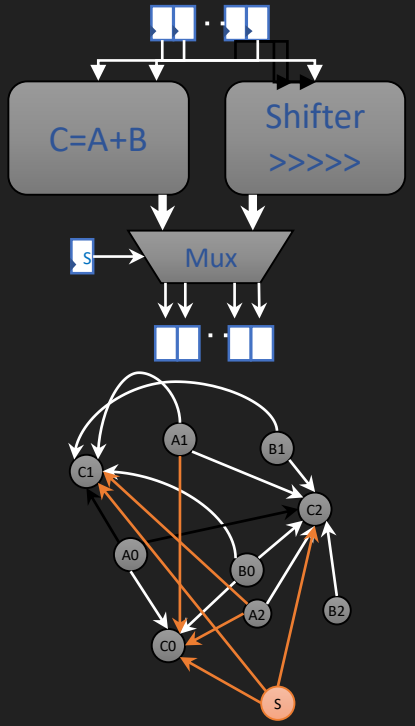
Hardware reverse engineering



Generate netlist graph of inputs-outputs



Apply ML techniques to identify subcircuits

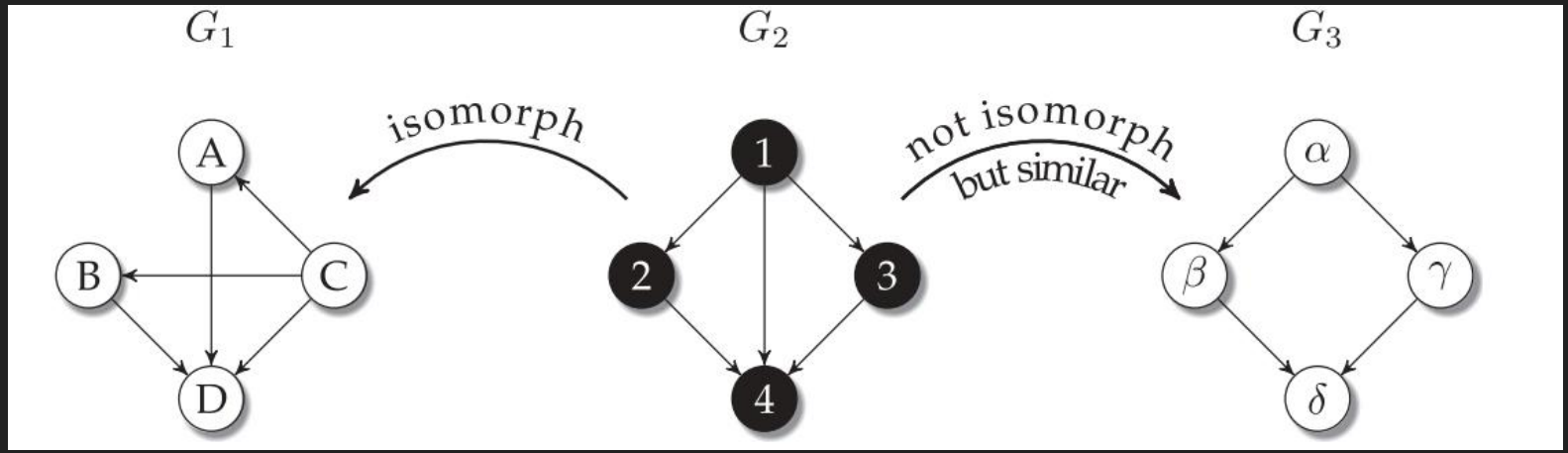


localization **segmentation**

cat dog duck

semantic segmentation

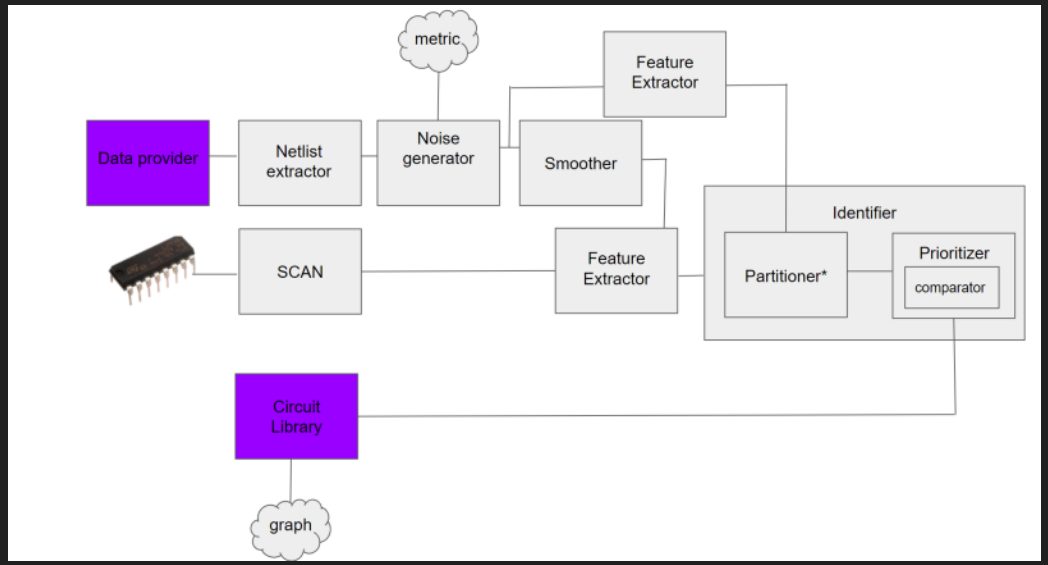
Graph isomorphism and graph similarities



Base on: Graph Similarity and its Applications to Hardware Security, Marc Fyrbiak, IEEE trans. computers, 2020

Graph Similarity based tool

- **Data provider:** VHDL, netlist of Scan based
- **Netlist extraction:** if the input is from HAL
- **Smoother -** convert a gate-level netlist to a flip-flop dependency graph (FFG).
- **SCAN reader:** An alternative to the HDL data provider.
- **Circuit Library:** sub-graphs with known issues
- **Identifier:** sub-circuit localization stage.
- **Partitioning: partition** (clustering) the circuit by points of interest.
- **Prioritizer:** Determines the order in which the sub-circuits are compared
- **Comparator:** find the suspected subgraphs



Due to the limited number of examples, we try to synthetically generate (sub-)graphs based on existing design -- WIP

Agenda

- Our main current activities
  - RE related
- Why do we need an EDA tool for secure systems
- Possible implementation directions
- Discussion

Why do we need an EDA based tool

- We strongly believe that it is near to impossible to take an existing design and make it secure
- Security needs to be considered throughout the entire development process.
- Unfortunately, we do not have enough tools to support it
- In SW we developed the notion of “Static analysis”; what is the equivalence in HW? E.g.,
  - Type checking
  - Memory boundaries
  - Uninitialized values
  - Can we check timing violations in static analysis?

What we can expect to achieve with such a tool?

- To assist the design process at ALL levels of maturity
- To analyze the CURRENT state of the design (statically) in order to indicate if
  - There are known security hazards; e.g., accessing restricted data is not protected
  - Under some conditions, the design may be exposed to security hazards; e.g., the secure signal needs to be raised at least 2 cycles before any unprivileged read

- If your design contains 3PIP (3rd party IP) → does the integration of this IP may lead to a security hazard

Agenda

- Our main current activities
  - RE related
- Why do we need an EDA tool for secure systems
- Possible implementation directions
- Discussion

A deeper look at Hardware based CWEs

- **A (access control) and D (Debug)**
  require a kind of Information flow or Taint analysis.
  - Recent papers suggest the use of AVL trees from each source (e.g., pin)
- **T (Timing)**
  usually use dynamic analysis. But static analysis can be used to calculate the conditions under which a threat may be caused.
- **M (Microcode)**
  can use the same technique as T, but needs to represent the microcode operation as well (including timing information).
- **F (Features) and O (Others)**
  most of them can be handled via a lint type of tool or a dynamic analysis
- **S (Side Channel)**
  Depending on the type of the side channel attack, we may use graph similarities and heat-based techniques to estimate the existence of security threat

A Access Control	D Debug	F Feature	M Microcode	O Other	S Side channel	T Timing related
1263	1301	1192	1271	440	1255	1279
1323	1323	1294	1331	1332		325
1243	1313	1272	1315	1332		1331
1314	1272		1342	1311		1315
1247	1258					821
1220	1191					1264
1220						
1318						
1283						
1037						
663						
821						
1264						

CWEs do not report Trojan Horses, although we may like to address this threat as well

Different design alternatives that we are currently considering

Data-flow – Information based

- Recent work (*) suggests the use of "AST" (abstract syntax trees) that represent Information flow trees
- It is similar to SW and HW tools are based on data-flow analysis + conditions that enable/block information flow

(*) Ahmad, Baleegh, et al. "Don't CWEAT It: Toward CWE Analysis Techniques in Early Stages of Hardware Design." *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*. Oct. 2022.

Formal verification

- Widely used in System design
- Suffer from "state explosion and simplifications of assumptions are needed to make it practical
- The quality of the results depends on the assumptions and simplifications you are making

GNN based

Two ways to use the graph representation

- In a supervised way – but we need many examples for that
- To use it as a NAS (Network Architecture Search) to guide a more effective way to perform formal verification

(GNN can also be used to locate locations that are sensitive to HTH insertion)

Preliminary thoughts:

- We believe that AST is too restricted and searching over the Netlist Graphs is a better choice
- We need to annotate the graph to include timing and control information
- Using time-analysis techniques, similar to WCET calculation can provide the timing related constrains
- We believe that using NAS based techniques is promising.

New Graph-based Static Analysis tools

Graph-based representation

- Multiple paths
- Contain metadata
- Can deal with complex situations.
- Allow dealing with Timing considerations
- Allow extracting sufficient and necessary conditions in an automatic way

(*)

AST-based representation

- Single path
- Hand-written queries
- Limited opportunities

Comments

Thanks

Backup

Formal presentation of a CWE

- Tortuga logic suggests the use of the following format (other tools suggests similar notations)
  assert iflow(signal or event =/=> condition of state);

- **For example assert iflow(secret_key =/=> insecure_mem);**

This rule states that the secret key should “not flow" or leak to an insecure memory where secret_key and insecure_mem are signals in the Verilog, SystemVerilog, or VHDL design.
The not flow operator makes this specification easy and compact.

Anohter Example
assert iflow (
{{Signals carrying confidential information}}
when ({{Privileged-mode bit is set}})
=/=>
{{Signals visible to unauthorized actor}}
);

Tortuga logic suggests a 5-step algorithm to check that CWE-related issues do not exist in a design

1. Identify CWE(s) relevant to the threat model.
2. State plain-language security requirement identified in the CWE(s).
3. List the assets (in the form of data or design signals), objectives (confidentiality, integrity, availability), and security boundaries of the design as they correspond to step 2.
4. Use the Radix security rule template for the corresponding CWE verification environments from Cadence®, Mentor® A Siemens Business, and Synopsys®.
5. Leverage the security verification