Static Information Flow Tracking (SIFT) Analysis for Hardware Design Verification

Christie Lincoln, Lisa Luo, Amir Uqdhah, Alvin Zhang
Advisors: Armaiti Ardeshiricham, Ryan Kastner

**Motivation**

Recently discovered attacks that exploit vulnerabilities in popular hardware allow private information to be leaked to public domains.

To prevent these attacks, hardware designers must eliminate these vulnerabilities via information flow tracking and security verification.

**Background and Problem**

Information Flow Tracking (IFT) can be used to identify what inputs affect which outputs. This is helpful in verification to determine if private inputs flow to public outputs, which indicates an information leak.

**Previous IFT Tools**

- **Gate Level IFT (GLIFT)**: Large overhead and lower level abstraction makes this tool less effective in verification.
- **Register Transfer Level IFT (RTLIFT)**: Attempts to solve the issues with GLIFT but only returns binary answers as to whether or not an input affects an output.

We want to develop a tool that provides more information, such as the source of the leakage and the path of the leaked data.

**Proposed Solution**

Trace the path of the inputs throughout the system to make it easier for designers to discover exactly when private information flows to public domain.

**Our Approach: Static IFT (SIFT)**

We analyze Verilog code by using a framework called Yosys, which creates an Abstract Syntax Tree (AST) of the design, and analyzing that resulting AST. We label sensitive input variables as tainted and as they interact with other variables, they taint output variables.

In the following example, we mark \( a \) as tainted. \( a \) taints \( x \), which taints \( y \), which taints \( z \).

We print out all the variables that the tainted inputs taint. Based on user input, we either print the tainted paths of each output or the tainted path of a specific output.

**Results**

After testing simple, singular modules simulating simple basic arithmetic methods, we tested more complex programs, which we gathered from Trust-Hub, a hardware security resource funded by the National Science Foundation (NSF). These programs contained current hardware encryption algorithms that had certain security design flaws.

Based on our tests, our tool, SIFT, is slower than RTLIFT but faster than GLIFT. However, SIFT provides more information than RTLIFT because SIFT provides the path of information leakage.

**Future Work**

Our tool is a step toward faster and easier debugging of hardware level code and serves to minimize security vulnerabilities in hardware designs. In the future, we want to be able to handle all Verilog syntax as our tool is unable to handle some syntax like concatenation operations. Our tool will be integrated as a part of a larger tool designed to automate hardware debugging.

**Acknowledgements**

We would like to thank Professor Christine Alvarado, Vignesh Gokul, and the Kastner Research Group (KRG) for their guidance and support.