Moving Hardware from “Security through Obscurity” to “Secure by Design”

Professor Ryan Kastner
Dept. of Computer Science and Engineering
University of California, San Diego
kastner.ucsd.edu
Classic System Design

- Hardware is simple, unchanging, correct, and secure
- Software is OS and applications

Software

Processor
Classic View of System Security

- Processor
- Software
- Applications
- Programming Language
- Compiler/OS/Firmware
- Instruction Set
- Microarchitecture
- Functional Units
- Logic Gates
- Transistors
Modern View of System Security

- Boot
- VMM
- Apps
- OS
- RTOS
- Lib
- Secure Resources
- NoC
- Debug
- L2
- Mem
- Radio
- Crypto
- L1
- CPU
- I/O
- Programming Language
- Compiler/OS/Firmware
- Instruction Set
- Microarchitecture
- Functional Units
- Logic Gates
- Transistors
Modern View of System Security

Many Stakeholders:
With different goals and objectives

Distributed Authority:
Multiple OS, VM, VMM, Access Control

HW/SW Coupling:
Hardware Accelerators, SW/FW Managed Resources

Shared Resources:
IP Cores, Memories, Communication, I/O
Hardware Security Vulnerabilities

Design flaws

\textit{case} 1: ...
\textit{case} 2: ...
\vdots
\textit{case} n: ...

Malicious code

Untrusted IP

Timing channel

Power channel

EM radiation

- Crypto

\[\text{Boot} \quad \text{VMM} \quad \text{Apps} \]
\[\text{OS} \quad \text{RTOS} \quad \text{Lib} \]
\[\text{Secure Resources} \quad \text{NoC} \quad \text{Debug} \]
\[\text{L2} \quad \text{L1} \quad \text{Mem} \]
\[\text{CPU} \quad \text{CPU} \quad \text{I/O} \]
<table>
<thead>
<tr>
<th>Hardware Design</th>
<th># Transistors</th>
<th>Lines of Verilog</th>
<th>Similar SW: LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel 4004</td>
<td>2.3K</td>
<td>1.25K</td>
<td>Simple App: 10K</td>
</tr>
<tr>
<td>Centaur Media Unit</td>
<td>430K</td>
<td>570K</td>
<td>Space Shuttle: 400K</td>
</tr>
<tr>
<td>Intel Pentium 4</td>
<td>41M</td>
<td>1M</td>
<td>F22 Raptor: 1.7M</td>
</tr>
<tr>
<td>MIT Raw</td>
<td>100M</td>
<td>34K</td>
<td>Pacemaker: 80K</td>
</tr>
<tr>
<td>Oracle SPARC M7</td>
<td>10B</td>
<td>???</td>
<td></td>
</tr>
<tr>
<td>nVidia Pascal</td>
<td>15B</td>
<td>???</td>
<td></td>
</tr>
<tr>
<td>Xilinx Virtex Ultrascale</td>
<td>20B</td>
<td>???</td>
<td></td>
</tr>
</tbody>
</table>
Security is Expensive

- ~1 defect/error per 10 lines of code.
  - *The Art of Good Design*, Mike Keating, Synopsys

- RedHat Linux: Best Effort Safety (EAL 4+)
  - $30-$40 per LOC

- Integrity RTOS: Design for Formal Evaluation (EAL 6+)
  - $10,000 per LOC
  - More evaluation of process, not end artifact
Hardware Security Proof Techniques

Proof by Obfuscation

Proof by Handwaving

Proof by Exhaustion

Proof by Intimidation

"YOU WANT PROOF? I'LL GIVE YOU PROOF!"
Our Research

- Develop a *secure hardware design flow* that
  - Formally specifies *security properties*,
  - Identifies *security vulnerabilities*, and
  - Quantifies *security threats*.
- Focus on security properties related to *confidentiality, integrity, isolation, separation, and side channels*. 

Source: Intel & Tortuga Logic
Confidentiality

Hardware Block

System Resources:
- Radio
- Secure Resources
- Untrusted App
- Debug

Secret Data (Crypto Key)

Input

Output
Integrity

Hardware Block

Untrusted 3rd Party IP Core

Input

Output

System Resources:
- Radio
- Secure Resources
- Untrusted App
- Debug
Availability (Timing Channels)

Hardware Block

Untrusted 3rd Party IP Core

Input

Output

System Resources:
  Radio (DoS)
  Secure Resources
  Untrusted App
  Debug
Secure Resources

Mixed Trust Domains

Hardware Block

Untrusted 3rd Party IP Core

Trusted IP Core

Input

Output

System Resources:
- Radio (DoS)
- Secure Resources
- Untrusted App
- Debug

Mixed Trust Resources
Information flow analysis solves all of these problems
“One group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see” [Goguen & Meseguer’82].
Information Flow: Inverter

0/T → 0/U
1/T → 0/1/U

<table>
<thead>
<tr>
<th>a</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The output is marked as “untrusted” when at least one “untrusted” input can influence the output.
Does this low level tracking help?

Simple assumption that “bad inputs” always leads to “bad outputs” is overly conservative.
Safely Resetting the Counter

Simple assumption that “bad inputs” always leads to “bad outputs” is overly conservative
Formalizing GLIFT

“Original” Logic    GLIFT Analysis Logic

Automatically generate logic that tracks labels

Tracking logic is compositional

Captures timing channels, and real time constraints

Security constraints can be expressed as hardware assertions

[ASPLOS09, DAC10, TCAD11, TIFS12, …]
GLIFT Logic Generation Flow

Automatic synthesis from HDL

```verilog
reg [31:0] gen_reg [7:0];
wire [31:0] mux2greg0;
always @(posedge clk) begin
  g_reg[0] <= mux2greg0;
end
assign is_store = instrn[29] | instrn[22];

mux2x1_32b my_mux0(.in0(g_reg0), .in1(newval), .sel(p_sel0), .result(mux2greg0));

reg [31:0] gen_reg_shadow [7:0];
wire [31:0] mux2greg0_shadow;
g_reg_shadow[0] <= mux2greg0_shadow;
assign is_store_shadow = ( instrn_shadow[29] & Instrn_shadow[22] )
  | ( instrn_shadow[29] & (~ instrn_shadow[22]) & (~ instrn [22] ) )
  | ( (~ instrn_shadow[29] ) & instrn_shadow[22] & (~ instrn[29]) );

mux2x1_32b_shadow sh_my_mux0( .in0(g_reg0), .in0_t(g_reg0_shadow),
  .in1(newval), .in1_t(newval_shadow), .sel(p_sel0), .sel_t(p_sel0_shadow),
  .ot(mux2greg0_shadow) );
```
Hardware Security Design Flow

* Speaker has significant financial interest
Crypto Core

Does my key leak?

Key

Message

Control Inputs

Cipher text

Control outputs

Does my key leak?
Crypto Core in Verilog

Does my key leak?

module crypto ( clk, reset, load_i, decrypt_i,
          data_i, key_i, ready_o, data_o );
  input  [127:0] data_i;
  input  [127:0] key_i;
  output [127:0] data_o;
  input clk, reset, load_i, decrypt_i;
  output ready_o;

How do we express this and test it?

assert iflow ( key_i =/=> data_o );
assert iflow ( key_i =/=> ready_o );
Crypto Core

Does my key leak? YES

How severe is the problem?

Key

Message

Control Inputs

Cipher text

Control outputs
Quantitative Information Flow Tracking

Normalized entropy, average mutual information and average success of attack

RSA architectures (1 ~ 6)

Challenges + Opportunities: Joint Analysis

- Gate Level Information Flow Tracking (GLIFT)
  - Proving non-interference
  - Identifying possible flows

- Quantitative measure
  - Numerous statistical & information theoretic metrics
  - Precise measurement of information flow
  - Detecting harmful flows and security vulnerabilities

Key

Expansion

Substitution

Mix Columns

Add

Shift Rows

Round Key
Control Logic
Control Outputs
Cipher Text
Message
Key

Yes
31.6 bits
Challenges + Opportunities: Joint Analysis

GLIFT:
assert iflow(key $\not=\Rightarrow$ control); Fail

Mutual Information:
$mi(key, control) = 31.6;$

GLIFT:
assert iflow(secure_resources $\not=\Rightarrow$ io); Fail

Mutual Information:
$mi(secure_resources, io) = 0.1$

GLIFT:
assert iflow(apps $\not=\Rightarrow$
secure_resources); Pass

Mutual Information:
$mi(apps, secure_resources) = 0;$
Challenges + Opportunities: Measurement

- Methods for efficiently calculating security metrics
- Achieve a more accurate estimation of security metrics while collecting as few samples as possible.
  - Density estimation
  - Multivariate estimation
  - Hardware accelerated techniques
Challenges + Opportunities: Language

- Languages for specifying security properties
- A security specification language for describing the security properties about the hardware design
  - What variables are important to secure?
  - What locations are easily visible?
  - What is your risk tolerance?

![Diagram with Key Mem and AES interconnect]
Assertion: Key only flows through AES

```assert iflow (key !==> $all_outputs
ignoring aes.$all_outputs)```

- If assertion holds, key only flows to outputs through AES first
- Real design: 10M gates
Challenges + Opportunities: Language

Assertion: Key only flows through AES

\[
\text{assert iflow (key } \neq \Rightarrow \text{ all_outputs ignoring aes.$all\_outputs) }
\]

- If assertion holds, key only flows to outputs through AES first
- Real design: 10M gates
Challenges + Opportunities: Language

Assertion: Key only flows through AES

assert iflow (key =/= $all_outputs
    ignoring aes.$all_outputs)

- If assertion holds, key only flows to outputs through AES first
- Real design: 10M gates
Challenges + Opportunities: Faster Verification

- Simplify analysis logic
  - Add one sided errors
  - Incremental proofs

- Higher abstractions
  - Bits to bytes to words to ...
  - Gates to RTL to HLS to ...
Challenges + Opportunities: Real Applications

- **Tortuga Logic**
  - Working with top semiconductor companies
  - Tools available to license
  - Academic research to commercial tool

- **VeriDrone**
  - Formally verified hardware/software shims
  - NSF CPS
Secure hardware design flow

- Formally specify security properties,
- Identify security vulnerabilities, and
- Quantify security threats.

Focus on security properties related to confidentiality, integrity, isolation, separation, and side channels.

kastner.ucsd.edu
Questions?

Team

- UCSD: Wei Hu, Ali Irturk, Ryan Kastner, Jason Oberg, Jonathan Valamehr
- UCSB: Frederic T. Chong, Ben Hardekopf, Vineeth Kashyap, Xun Li, Bita Mazloom, Tim Sherwood, Hassan Wassel
- UT Austin: Mohit Tiwari

Publications