Hardware Security Challenges and Solutions

Mike Bartley
TVS, Founder and CEO
Agenda

- Some background on your speaker and testing safety related systems
- Threats and solutions
- Verifying those solutions
- Bare metal monitoring
Your speaker: Mike Bartley

- PhD in Mathematical Logic
- MSc in Software Engineering
- MBA

- Worked in software testing and hardware verification for over 25 years
  - Praxis, IPL, ST-Micro, Infineon, Panasonic, ARM, NXP, nVidia, ClearSpeed, Gnodal, DisplayLink, Dialog, ...
  - Worked in formal verification of both software and hardware

- Started TVS in 2008
  - Software testing and hardware verification products and services
  - Offices in India, UK, France and Germany
TVS - Global Leaders in Test and Verification

Continuous geographical expansion…
Threat growth – Just a Software Problem?

Number of breaches per threat action category over time

- Hacking
- Malware
- Social
- Physical
- Misuse
- Error

Source: Verizon
Perimeter Security vs. Layers of Security

Cisco predict that by 2020 50 billion chips will be connected to the Internet.

PS: TTTech predict that by 2020 50% of all chips will be safety-related.

PPS: Often chips will be both (e.g. connected cars 😊)
Threats

• Hackers will try to find an exploit allowing unrestricted access to assets or services.
  – Financial, Media, Device repurposing...

• A hacker will attempt to gain control of the system
  – If the behaviour of a smart meter can be altered then there is an obvious financial gain
  – The ability to modify the software controlling a driverless car would allow a hacker to target individuals or groups and to extort money from the manufacturer

• At its heart software running on a hardware platform embedding a CPU and peripherals
  – Attacks:
    • Modify the software running on that platform
    • Exploit design bugs – unintended functionality
    • Introduce a fault that can be exploited
Attack Vectors

• Access: an attacker may have direct physical access to the system or attempt to hack it remotely

• Local logical attacks
  – Exploitation of bugs (design mistakes)
  – Modification of the boot code. e.g. a simple re-flash
  – Exploitation of debug or test functionality

• Local physical attacks
  – Fault injection, e.g. power glitch
  – Side channel analysis to reveal secret keys
  – Reverse engineering

• Remote attacks, via the communication interface
  – Exploitation of bugs (design mistakes, protocol errors)
  – Forging messages or software updates
  – Injecting protocol faults
  – Spying on messages to learn about the system
  – Man in the middle

• The cost of a breach can be huge
Basic Hardening

• Locking down of debug and test functionality
• Digital authentication of all code
  – Signed boot code
  – Signed updates
• Protected Communication Interfaces
  – Signing & Encryption of all messages (SSL)
• Requires encryption and authentication keys
  – These keys must be protected
    • Immutable public keys
    • Immutable and non readable secret keys
• These objectives cannot be achieved with software alone
• Typically the hardware infrastructure must support:
  – Embedded cryptographic keys
  – Secure factory provisioning
  – Cryptographic engines
  – Random number generation
  – A non volatile version counter
  – Access control hardware
    • Memory space
    • Interfaces
  – A secure compute environment
  – May require: Fault and side channel resistance

• These elements are collectively known as a hardware root of trust – they underpin the security of the entire platform!
Secure systems demand a very high level of verification
  - Patching hardware is usually not an option
Traditional verification has focused on positive testing
Security verification must also cover negative testing
  - Ensuring that unintended functionality does not exist that could be exploited
  - Full code (HDL) coverage is important – ideally no untested state
  - A common technique for interfaces is fuzz testing
    • A security interface is abused by passing illegal data (opcode, address, etc.) and the response monitored for unexpected behaviour
    • Usually a combination of random and targeted tests are applied
• **Side Channel Analysis**
  – The computations performed on an SoC device cause its power consumption to be modulated
    • Often the consumption is data dependent
    • Thus information leaks!
  – If not properly defended against, an attacker who can trigger a cryptographic computation and monitor the supply current may be able to obtain the key
    • Simple Power Analysis (SPA)
    • Differential Power Analysis (DPA) – very powerful (Kocher et al.)
    • DEMA – Differential analysis using EM radiation
  – Data depended computation timing also leaks information
  – Attacks are non invasive – relatively low cost
  – Countermeasures try to mask and hide the signal (SNR)
  – Specialist verification techniques are needed
    • Up to now evaluation has been largely done on the end hardware
    • Current academic research is examining the use of power simulations
Advanced Attacks

• Fault Injection
  – An attacker may attempt to subvert the system operation by introducing a fault
    • Power Supply Glitch
  – A well timed fault may subvert the direction of a branch

• Physical reverse Engineering
  – An attacker may attempt to reverse engineer and de-layer an SoC device
    • For example to extract secret keys from internal non volatile memory
  – Invasive and very expensive
  – It requires specialist equipment and knowledge
  – The return must be worth the investment
  – Possible countermeasures include tamper detectors and shielding
In Summary

- Good security is difficult
- Good security is not free
  - But the cost of a breach is higher!
- Strong verification is a must
- Advanced protection requires advanced countermeasures
  - Signal masking and hiding algorithms
    - But also special circuit design techniques
  - Built in redundancy
  - Tamper detection
  - All requiring strong verification!
Secure Data over Packet Channel

- No direct association between signal name and secure / non-secure! There’s a control / temporal component also.

- Secure Formal does not help in analyzing strength of Cryptography blocks.

**Key**

**M**atch

**Next Candidate**

**Data Channels**

**Packet Encoder**

**FIFO**

**Packet Router**

**Crypt**

**Cached Candidate**

**Match Comparator**

**RED = secure**

**Green = non-secure**

**Blue = mixed**

**Key Memory**
• Identify secure signals [sources] we are concerned about (typically not many)

• [Auto] identify all the places it might be able to reach (typically hundreds or thousands – ALL the outputs, or top level signals of the block)

• Confirm that signal can ONLY reach the places it’s supposed to, and not anyplace where the bad guys could steal it.

• **No way to directly specify this in SVA!**
- Critical component is adversely affected
• Secret data is unintentionally leaked
Case Study – Top-25 Semi Company
Key Flowing Out Of Design

- 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 world results
  - State-of-the-art design with over 10 million gates
  - Actual required properties, impossible to visually inspect
• 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 world results
  – State-of-the-art design with over 10 million gates
  – Actual required properties, impossible to visually inspect
**Case Study – Top-25 Semi Company**

**Key Flowing Out Of Design**

- **Assertion:** Key only flows through AES
  - assert iflow (key $\neq$ $\Rightarrow$ $\forall$ all_outputs ignoring aes.$\forall$ all_outputs);
  - If assertion holds, key only flows to outputs through AES first

- **Real world results**
  - State-of-the-art design with over 10 million gates
  - Actual required properties, impossible to visually inspect
Demo: AES Key Leakage

Property:
assert iflow (key =/= data_o);

Result (demo):
Fails in 4 cycles
Key XOR Data flows to pins, security flaw
Demo: AES Key Leakage

Property:
assert iflow (key /=/> data_o);

Result:
Fails in 506 cycles
Encrypted data flows to pins
Flow is allowed, ready_o=1
Demo: AES Key Leakage
Demo: AES Key Leakage

Property:
assert iflow (key =/= data_o) || ready_o;

Result:
Assertion Holds
Demo: AES Key Leakage
Post-silicon Hardware Monitoring

A system consisting of blocks that observe and monitor activity as well as forwarding events using a self-contained message network can be used to create an over-arching security system.

The secure channel connects the UltraSoC domain with a service.
UltraSoC can detect attacks such as
- attempts to read from secure memory,
- attempts to write to blocks in unexpected or unauthorized ways,
- and accesses that may be probes or distributed denial of service (DDoS) attacks.
Summary

- Threats and solutions
- Verifying those solutions
- Bare metal monitoring
Questions

- Mike Bartley, CEO, TVS
  mike@testandverification.com

- Elchananan Rappaport, President, Gila Logic, Inc.
  elchanan@gilalogic.com

- Jason Oberg
  CEO, Tortuga Logic
  Obergjason@tortugalogic.com