Towards Zero Trust: An Experience Report

Jason Lowdermilk and Simha Sethumadhavan
Primary security goal: move functionality into hardware
The reason we want to reduce the attack surface

- Millions of lines of code.
- Thousands of exploitable bugs.
- An attacker only needs one to break security.

Even air-gapped systems are vulnerable to supply-chain attacks!
The reason we want to reduce the attack surface

- Millions of lines of code.
- Thousands of exploitable bugs.
- An attacker only needs one to break security.

Fewer lines of code  ➔  More trust
To what extent can we reduce the attack surface under realistic engineering constraints?
Outline for rest of the talk

- Background on Satellite Ecosystem
- Our new architecture
- Key technical steps to increase trust
  - Migrating software to FPGA (using HLS)
  - Memory safety improvements
  - Avoiding hardware backdoors
- Reflection and Takeaways
What is a satellite front-end processor?
What is a satellite front-end processor?
What do Front-End Processors (FEPs) look like today?
What do Front-End Processors (FEPs) look like today?

- FEPs can be large!
- Redundant strings of equipment
- Rows of rack-mounted servers, network routers, cryptos, data guards
- Additional equipments for infrastructure: virtual network functions, test equipment, patch panels
Outline for rest of the talk

● Background on Satellite Ecosystem
● Our new architecture
● Key technical steps to increase trust
  ○ Migrating software to FPGA (using HLS)
  ○ Memory safety improvements
  ○ Avoiding hardware backdoors
● Reflection and Takeaways
Outline for rest of the talk

● Background on Satellite Ecosystem
● Our new architecture
● Key technical steps to increase trust
  ○ Migrating software to FPGA (using HLS)
  ○ Memory safety improvements
  ○ Avoiding hardware backdoors
● Reflection and Takeaways
High-level synthesis, two ways

1. Conversion of legacy code to Verilog

```plaintext
data_t decode_rs_ccsds_top(data_t din)
{
    int o = (b == 0) ? 1 : 0;
    data_t r = d[o][cnt];
    int p = 0;
}
```

2. Implementation of new code using Clash

```plaintext
module decode_rs_ccsds_top_decode_rs_ccsds_top (ap_clk,
ap_rst,
ap_start,
ap_done,

assign c$j2_case_alt = found ? c$j2_case_al
assign c$j2_case_alt_0 = c$j5_case_scrut ?
always @(*) begin
    case(c$ds_app_arg)
        3'b010 : result_12 = c$j3_case_alt;
```
Implementing forward-error correction

Step 1: refactoring at the “C” level

```c
int decode_rs_ccsds(data_t *data, int *eras_pos, int no_eras, int pad)
```

```c
data_t decode_rs_ccsds_top(data_t din)
```
Implementing forward-error correction

Step 1: refactoring at the “C” level

Step 2: conversion to Verilog (Vitis HLS)
 Implementing forward-error correction

Step 1: refactoring at the “C” level
Step 2: conversion to Verilog (Vitis HLS)
Step 3: byte streaming -> bit streaming
Implementing forward-error correction

Step 1: refactoring at the “C” level
Step 2: conversion to Verilog (Vitis HLS)
Step 3: byte streaming -> bit streaming
Step 4: round robin parallelism
Outline for rest of the talk

● Background on Satellite Ecosystem
● Our new architecture
● Key technical steps to increase trust
  ○ Migrating software to FPGA (using HLS)
  ○ Memory safety improvements
  ○ Avoiding hardware backdoors
● Reflection and Takeaways
Major benefit: Removal of memory safety issues!

Microsoft: 70% of exploits are related to memory safety

Our hardware-based architecture has complete memory safety, since there are no stack frames to corrupt, and no instruction pointer to overwrite.

Legacy code may require refactoring to adapt dynamic memory usage into static memory.
Outline for rest of the talk

● Background on Satellite Ecosystem
● Our new architecture
● Key technical steps to increase trust
  ○ Migrating software to FPGA (using HLS)
  ○ Memory safety improvements
  ○ Avoiding hardware backdoors
● Reflection and Takeaways
Trust in people and tools

- Developers themselves may be targeted
- EDA tools may become compromised
Trust in people and tools

- Developers themselves may be targeted
- EDA tools may become compromised
- Solution: Use commercially available tool to scan for backdoors [CCS 2013]

https://chipscan.us/cloud
https://dl.acm.org/doi/10.1145/2508859.2516654
Hardware may also have trust issues

- General-purpose processors may have “unfixable” flaws such as Spectre
- High likelihood of other undiscovered vulnerabilities

Avoid the use of general-purpose processors

- Use only the reconfigurable portions of the FPGA
- Reduced risk of targeted backdoors inserted by the foundry
Outline for rest of the talk

● Background on Satellite Ecosystem
● Our new architecture
● Key technical steps to increase trust
  ○ Migrating software to FPGA (using HLS)
  ○ Memory safety improvements
  ○ Avoiding hardware backdoors
● Reflection and Takeaways
Results, Reflection and Takeaways

Trust Benefits

- Debloating: 100x reduction with cloud architecture
- Complete memory safety
- Applicable to other systems

Performance Benefits

- 125 MBPS in cloud-based FPGA
  - 20-30 MBPS is typical for software FEPs
- Very low latency due to the limited use of memory

Productivity Benefits

- HLS provides software-like development flow
  - Fuzz testing
  - High productivity

Total effort was 18 person months
Reflection

- Hardware tools have improved significantly
  - Can convert legacy code!
  - Can program in high-level languages
  - Productivity gains through fuzz testing
  - Can check for backdoors!
  - Many of these tasks are not possible with full-fledged software

- Room for improvement in terms of how FPGAs are implemented on the cloud

- Security without trust is impossible, goal is to minimize trust
  - New assurance flow offers an alternative to formal verification and homomorphic encryption
  - Promising results
  - But this is one result and we need more results to understand generality and problems

- Thank you and questions!