Dynamic GARUDA: Securing Programs with Hardware Monitors Using Higher Language Abstractions

Garett Cunningham, Gordon Stewart, David Juedes and Avinash Karanth
School of Electrical Engineering and Computer Science, Ohio University

This research was sponsored by NSF SaTC program CCF-1936794

Introduction

Low-level embedded systems are especially vulnerable to attacks that exploit flaws in either software or hardware to gain control of program behavior. Hardware monitors have shown promise toward attacking the issue by catching malicious instructions and enforcing some expert-defined policy at runtime. However, the efficiency of monitors comes at the cost of ease of implementation. To bridge the abstraction gap, we define a high-level language for writing dynamically reconfigurable security policies, named Dynamic GARUDA, alongside a Verilog compiler to support realizing policies as hardware monitors.

Prior Work & Motivation

Previously, the GARUDA language[1] attempted to close the abstraction gap by introducing a high-level language for simple policies with a compiler to synthesizable Verilog code. The simplicity plus the software-to-hardware compilation yielded efficient monitors that saw minimized overhead compared to other approaches.

The language is composed of a simple list of commands:

```
pol := PId            // Do nothing
     | PDrop          // Fail
     | PTest e p      // If e, then do p
     | PUpd e         // Update data with e
     | PChoice p1 p2  // Do p1 or p2
     | PConcat p1 p2  // Do p1 then p2
```

Policies consist of compositions of these commands. The resulting language is simple, but powerfully expressive in the policies it can encode. However, policies cannot be reconfigured once they are written. Experts may wish to change behavior on demand in response to runtime flags, such as in preventing data leakage or checking shadow stacks. This motivates our rework of GARUDA to support dynamically reconfiguring policies.

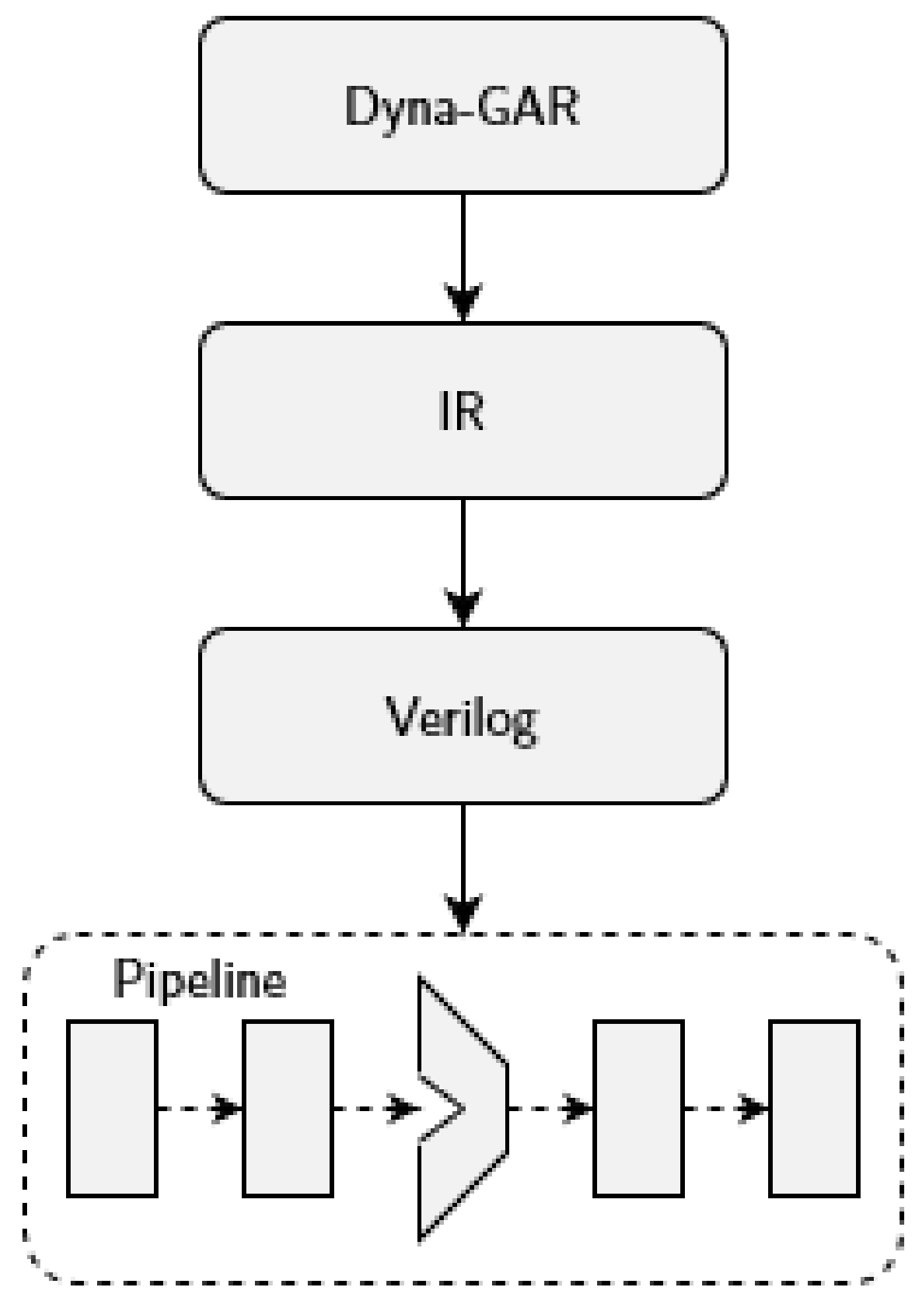
Dynamic GARUDA

Semantics:
We redefine GARUDA's semantics using inspiration from the bitstream processing language Ziria[2]:

```
stream := upd e     // Update data with e
        | done e    // Update and return e
        | ite e s1 s2  // Conditional branch
        | x <- s1; s2  // Stream staging
        | s1 >> s2     // Stream composition
        | loop s       // Stream looping
```

Stream staging (`x <- s1; s2`) best demonstrates how policies can dynamically change behavior. The command runs the policy `s1` until it returns a value to `x`. `s2` is then run with the value of `x` passed as additional input. Loops allow this behavior for the same stream, emulating an internal register that stores the value of `x`.

Synthesizing Hardware Monitors:
To produce low-overhead hardware monitors, we rework the original Verilog compiler to support our new semantics.



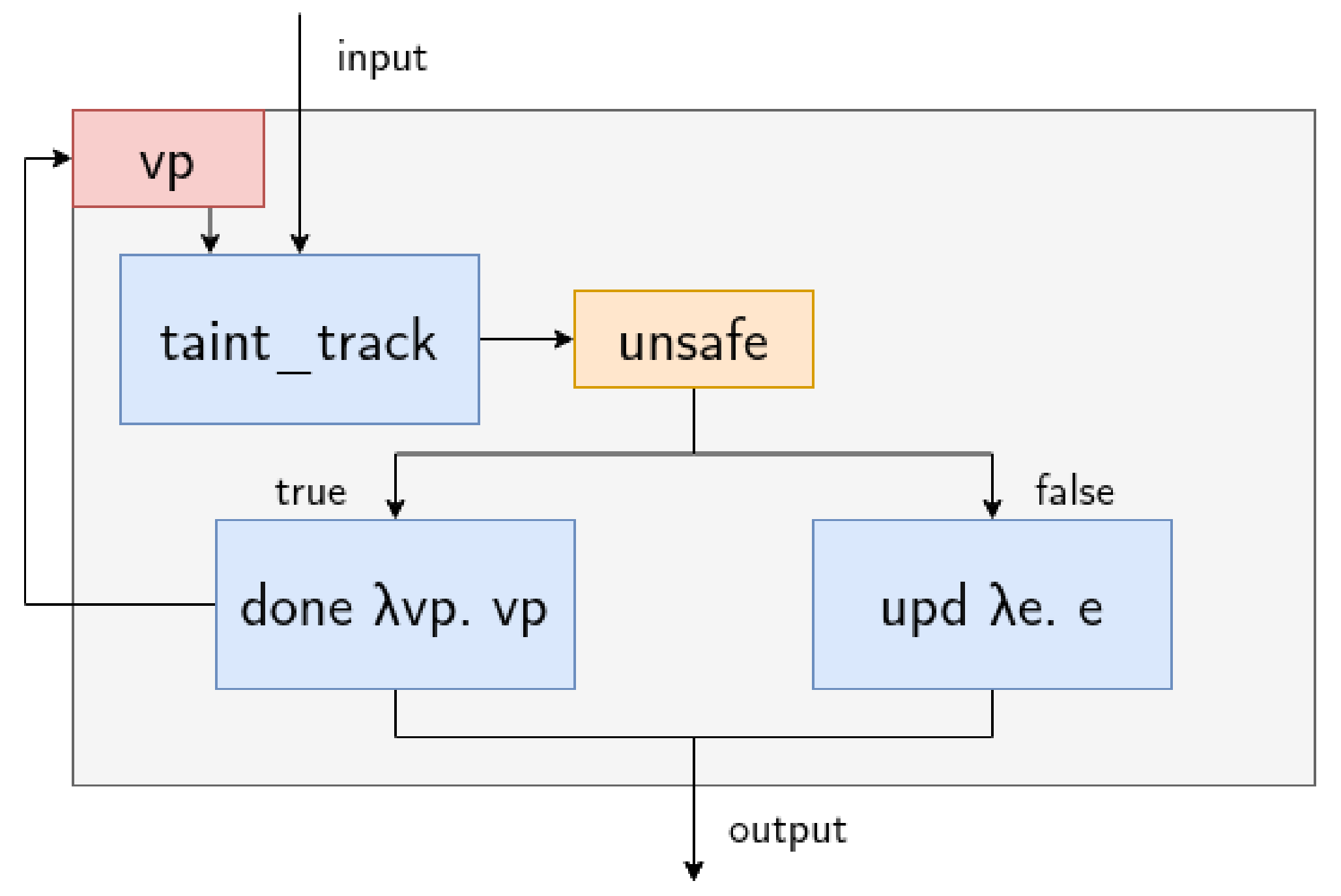
We modularize staged streams and introduce lines for control and data transfer. This enforces that policies only run when specified and have access to passed variables represented in the high-level code.

Applications

Dynamic GARUDA can express a much wider range of policies than prior work. For example, Speculative Taint Tracking[3] (STT) enacts security measures when pre-computing code. The following Dynamic GARUDA code snippet implements STT:

```
Definition stt :=
  loop (fun vp =>
    taint_track vp >>
    ite (unsafe)
      (done (fun _ => vp))
      (upd (fun e => e))).
```

`stt` controls the flow of pre-computed instructions. If the logic in `taint_track` shows that the instruction is not safe, then the computation stalls by fixing the *visibility point* `vp` and changing the instruction to a no-op. Else, instructions pass unchanged. As a hardware monitor, the policy is as follows:



Future Work

We are continuing to evaluate the performance and effectiveness of Dynamic GARUDA for security applications, including:

- Benchmarking policies to estimate overhead.
- Optimizing compilation to Verilog to minimize power and area consumption.
- Formally verifying our compiler toolchain in Coq and supporting formal verification of policies.

References
[1] S. Sefton et al. "GARUDA: Designing energy-efficient hardware monitors from high-level policies for secure information flow," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018
[2] G. Stewart et al. "Ziria: A DSL for wireless systems programming," *Proceedings of the 20th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2015
[3] J. Yu et al. "Speculative taint tracking (STT): A comprehensive protection for speculatively accessed data," *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019