Journal of Cryptographic Engineering
https://doi.org/10.1007/s13389-022-00306-w

REGULAR PAPER

Isadora: automated information-flow property generation for hardware security verification

Calvin Deutschbein[1] · Andres Meza[2] · Francesco Restuccia[2] · Ryan Kastner[2] · Cynthia Sturton[3]

Received: 19 June 2022 / Accepted: 21 October 2022

Abstract
Isadora is a specification mining tool for creating information-flow properties for hardware. Isadora combines hardware information-flow tracking and specification mining to produce properties that are suitable for the hardware security validation and support a better understanding of the hardware's security posture. Isadora is fully automated; the user provides only a hardware specification and a testbench—they do not need to supply a threat model or security requirements. Isadora is evaluated on a RISC-V processor, an SoC access control mechanism, and the OpenTitan hardware root of trust. Isadora generates security properties that align with Common Weakness Enumerations (CWEs) and with properties written manually by security experts.

Keywords Information-flow tracking · Specification mining · Hardware security validation

1 Introduction

Security validation is an important yet challenging part of the hardware design process. A strong validation provides assurance that the hardware is secure and trustworthy: it will not be vulnerable to attack once deployed, and it will reliably provide software and firmware with the advertised security features. A security validation engineer is tasked with defining the threat model, specifying the relevant security properties, detecting any violations of those properties, and assessing the consequences to system security.

Existing commercial design tools[1] Examples include Mentor Questa Secure Check, Cadence JasperGold Security Path Verification, and Tortuga Logic Radix.can verify security properties. But the validation is only as strong as the provided properties. *Defining properties is a crucial part of the security validation process that currently involves a significant manual undertaking.*

Isadora is an automated methodology that combines information-flow tracking with specification mining to create a hardware security specification. The specification can be used as a set of security properties suitable for use with existing security validation tools, and it can also be studied directly by the validation engineer to support their understanding of how information flows through the hardware and to better understand potential security weaknesses.

Hardware information-flow tracking (IFT) is a powerful security verification technique that monitors how information moves through the hardware [24]. Hardware IFT was introduced at the gate level [25,38] and has been optimized toward RTL [2,42] and HLS [33]. Existing IFT verification engines can be used to confirm whether a given information-flow property holds. However, it is up to the designer to specify the full set of desired flow behaviors.

Specification mining offers an automated alternative to manually writing properties. The technique works by analyzing traces of execution to find patterns of behavior. Specification mining can be applied to software [1] or hardware [19], and has recently been applied to system on a chip (SoC) designs [15,34]. Security specification mining focuses on finding patterns of behavior that are critical to security and has been shown valuable for the security analysis of processors [11,12,43]. However, many important vulnerabil-

[1] Examples include Mentor Questa Secure Check, Cadence JasperGold Security Path Verification, and Tortuga Logic Radix.

✉ Calvin Deutschbein
ckdeutschbein@willamette.edu

[1] Willamette University, University of North Carolina at Chapel Hill, Chapel Hill, USA

[2] University of California, San Diego, USA

[3] University of North Carolina at Chapel Hill, Chapel Hill, USA

Published online: 11 November 2022

ities relate to how information flows, properties that are not discoverable via trace analysis on traces of only functional values.

The key insight behind Isadora is to mine traces that consist of the functional values *and* their IFT labels. Mining upon this IFT-enhanced trace enables the generation of information-flow properties. The security labels encode information related to noninterference, which transforms the information-flow properties from the space of hyperproperties [6]—where trace-based mining does not apply—to the space of trace properties.

A naïve application of trace-based mining to an IFT-enhanced traces quickly runs into issues of complexity: the traces are large and overwhelm the miner. Additionally, the miner will discover properties over the security labels and original hardware signals that are meaningless and cannot be transformed back to the space of information-flow properties in the original design. To handle these issues we separate the process of identifying source–sink flow pairs in the design from the process of mining for the conditions that govern those flows. The first can be done by leveraging existing information-flow tracking tools and the second makes use of existing trace miners. The key to making the approach work is to synchronize the two parts using clock-cycle time.

The methodology we present here can inform an automated analysis of a hardware design by identifying flow relations between all design elements including conditional flows along with their conditions, multi-source flows, and multi-sink flows. The methodology requires no input from the designer beyond the design and testbench. For larger designs or for better performance, the designer may restrict the set of source and sink signals to find information flows within specific sub-spaces of the design.

To our knowledge, Isadora represents the first specification miner capable of extracting information-flow security properties from hardware designs. Isadora is a fully automated security specification miner for information-flow properties. Isadora uses information-flow tracking (IFT) technology from the simulation-based security verification engine, Radix-S by Tortuga Logic [28], and is implemented on top of Daikon [14], a popular invariant miner.

- Isadora characterizes the flow relations between all elements of a design.
- Isadora identifies important hardware security properties without guidance from the designer.
- Isadora can be used to find undesirable flows of information in the design.
- Isadora is applicable to SoCs and CPUs.

To measure our methodology and the usefulness of Isadora's mined specification, we evaluated Isadora over an access control module, a RISC-V CPU design, and the OpenTitan hardware root of trust. We evaluated the output of Isadora versus expected information-flow policies of the design and found information-flow specifications that, if followed, protect designs from known and potential future attack patterns.

2 Hardware security properties

Isadora generates different classes of information-flow properties. *Information-flow restrictions* indicate the lack of information flow between two hardware components. *Information-flow conditions* denote an information flow between two hardware components, but only when the hardware is in a certain state.

When considering properties, we can differentiate between properties of the design and the properties that are discovered from traces. In practice, a trace cannot explore all possible design states so the properties found by specification miners, including Isadora, may under-approximate design behavior. On the other hand, any information flow described by Isadora is a true flow of the given design. In this section, we discuss the classes of information-flow properties that Isadora can specify and provide a grammar for the properties. In Sect. 6, we explore how these properties may differ from true properties of the design.

2.1 Tracking information flow

Hardware IFT can precisely measure all digital information flows in the underlying hardware including implicit flows through hardware-specific timing channels [24]. Isadora uses register transfer level IFT [2] to track data flow between signals rather than considering individual bits, with 'signals' in this context referring to the Verilog notion of a register. Isadora may additionally be configured to consider Verilog wires, and this configuration was used on SoC designs. We use the term signal to refer to any design element, be it a register or wire in the Verilog sense.

Tracking proceeds as follows: for each signal $\mathtt{s}$ in the hardware, a new signal $\mathtt{s}^T$, which acts as the IFT label, is added along with the logic to track how the label propagates through the design. The resulting security model provides a logic that analyzes the security labels and system state, and updates those labels on a cycle-by-cycle basis. This is as documented in GLIFT [25] and RTL IFT [2]. Isadora uses Tortuga Logic Radix-S to generate a security model.

The security model is initialized by setting the label of a relevant *information source signal* (or multiple signals). All other tracking labels are initialized to zero. As the hardware executes, if information related to a source signal propagates to a second signal, the second signal's tracking label is

updated from to be nonzero, i.e., the security model indicates that there is a flow of information.

2.2 Information-flow restrictions

Using hardware IFT, we can express the property that information from signal $\mathtt{r}_1$ should never flow to signal $\mathtt{r}_2$: if $\mathtt{r}_1$ is the only signal whose tracking label $\mathtt{r}_1^T$ is initialized to nonzero, then for all possible executions, $\mathtt{r}_2$'s tracking label $\mathtt{r}_2^T$ should remain at zero:

$$(\forall \mathtt{r}_i,\ \mathtt{r}_i^T \neq 0 \leftrightarrow i = 1) \rightarrow \mathbb{G}\left(\mathtt{r}_2^T = 0\right).$$

This style of information-flow restriction can be useful for ensuring unprivileged users cannot influence sensitive state or for ensuring that sensitive information cannot leak to an unprotected memory space, unprotected I/O port, or some other publicly viewable hardware state. However, it cannot capture conditional properties, for example that register updates are allowed after reset.

2.3 Information-flow conditions

We can express the property that information from a signal $\mathtt{r}_1$ may flow to another signal $\mathtt{r}_2$ under some condition P: if $\mathtt{r}_1$ is the only signal whose tracking label $\mathtt{r}_1^T$ is initialized to nonzero then for all possible executions of the design, $\mathtt{r}_2$'s tracking label $\mathtt{r}_2^T$ will only become nonzero if some predicate P holds:

$$(\forall \mathtt{r}_i,\ \mathtt{r}_i^T \neq 0 \leftrightarrow i = 1) \rightarrow \mathbb{G}(\neg P \rightarrow (\mathtt{r}_2^T = 0 \rightarrow \mathbb{X}(\mathtt{r}_2^T = 0)))$$

This style of an information-flow condition can be used to express that register updates are allowable only during certain system states or that memory accesses requires successful access control checks.

2.4 Grammar of properties

We produce information-flow properties using a *restriction operator* `=/=>` that indicates the lack of information movement between two hardware signals. This allows us to succinctly express both information-flow restrictions and information-flow conditions. More formally, the operator `=/=>` indicates noninterference [18].

The grammar of Isadora properties is:

$$\begin{aligned}
\phi &\doteq \mathtt{r}_1 \texttt{=/=>}\ \mathtt{r}_2 \mid e \rightarrow \mathtt{r}_1\ \texttt{=/=>}\ \mathtt{r}_2 \\
e &\doteq b \wedge e \mid b \\
b &\doteq \mathtt{r} \in \{x, y, z\} \mid A\mathtt{r}_1 + B\mathtt{r}_2 + C = 0 \mid \mathtt{r}_1 \neq \mathtt{r}_2 \mid \mathtt{r} \\
&\quad = \mathrm{prev}(\mathtt{r}).
\end{aligned}$$

The property $\mathtt{r}_1$ `=/=>` $\mathtt{r}_2$ states that information flow from $\mathtt{r}_1$ to $\mathtt{r}_2$ is restricted. The property $e \rightarrow \mathtt{r}_1$ `=/=>` $\mathtt{r}_2$ states that $\neg e$ must hold for information to flow from $\mathtt{r}_1$ to $\mathtt{r}_2$. The symbol $\mathtt{r}$ is a signal in the design, $\mathtt{r} \in \{x, y, z\}$ means that $\mathtt{r}$ may take on any one of the values in a set of cardinality less than or equal to three, and prev($\mathtt{r}$) refers to the value of $\mathtt{r}$ in the previous clock cycle.

3 Methodology

Isadora analyzes a design in four phases: generating traces, identifying flows, mining for flow conditions, and postprocessing. An overview of the workflow is presented in Fig. 1.

Isadora uses an information-flow security model to add a security label to every hardware component and provide logic that calculates how to update the labels as the hardware executes. Isadora uses Tortuga Logic Radix-S to generate the security model. Radix-S performs a functional simulation using a user-supplied functional testbench. Radix-S uses its security model to compute an IFT-enhanced trace that specifies the value of every hardware signal along with its corresponding IFT label for every clock cycle during simulation. The labels act as additional metadata about how the hardware behaves with respect to propagating information. These IFT labels are crucial for Isadora to derive IFT security properties.

Next, Isadora studies the trace set to find every flow that occurred during the simulation of the design. This set of flows is complete: if a flow occurred between any two signals, it will be included. As the end of this phase, Isadora also produces the complete set of restrictions: pairs of signals between which no information flows.

Then, Isadora uses Daikon [14] to infer, for every flow, the predicates that specify the conditions under which the flow occurred.

The final phase removes redundant and irrelevant predicates from the set and logically combines the predicates with the information flows to produce the information-flow conditions. These, along with the information-flow restrictions from the second phase, form the information-flow specification produced by Isadora.

3.1 Generating traces with information-flow tracking

We use Tortuga Logic Radix-S to create the security model and generate the trace set. The simulation is driven by a functional testbench that is accompanied by the hardware designs. The testbenches are not modified in any manner. The testbench provides input values to the hardware to drive its functional execution. Additionally, we provide Radix-S

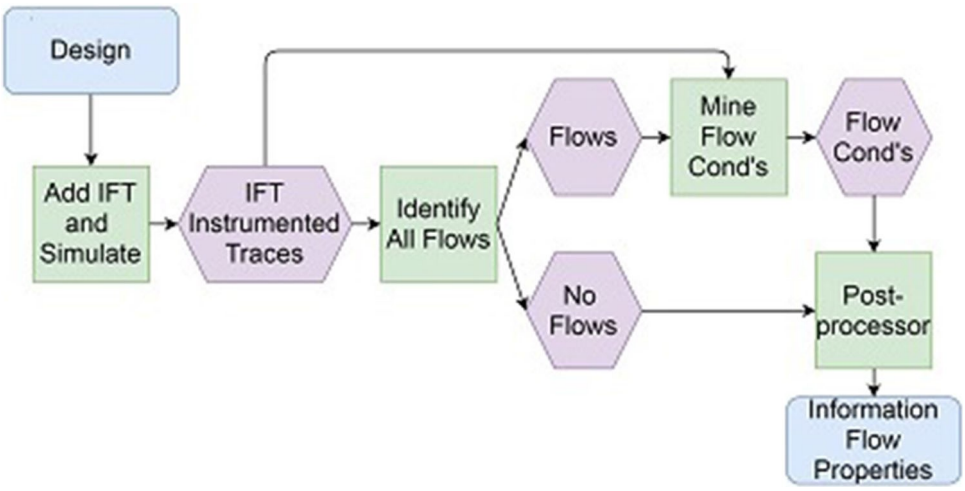

Fig. 1 An overview of the Isadora workflow

the src signal to track, i.e., the security label of src will be initially set (tainted).

Let $\tau_{\texttt{src}} = \langle \sigma_0, \sigma_1, \ldots, \sigma_n \rangle$ be the trace of a design instrumented to track how information flows from one signal, src, during execution of a testbench. The state σ_i of the design at time i is defined by a list of triples describing the current value of every design signal and corresponding tracking label in the instrumented design:

$$\sigma_i = \left[(\mathtt{s}_1, v_1, v_1^t), (\mathtt{s}_2, v_2, v_2^t), \ldots, (\mathtt{s}_m, v_m, v_m^t)\right]_i .$$

In order to distinguish the source of an information flow, each input signal must have a separate taint label. However, tracking multiple labels is expensive [25]. Therefore, Isadora takes a compositional approach. For each source signal, an IFT-enhanced trace is generated to track the flow of information from only a single input signal of the design, the src signal. This process may be applied to each signal in a design, or to a specified subset.

The end result is a set of traces for a hardware description D and testbench T: $\mathcal{T}_{\mathrm{DT}} = \{\tau_{\texttt{src}}, \tau_{\texttt{src}'}, \tau_{\texttt{src}''}, \ldots\}$. Each trace in this set describes how information can flow from a single input signal to the rest of the signals. Taken together, this set of traces describes how information flows during execution of the testbench T.

3.2 Identifying all flows

In the second phase, the set of traces is analyzed to identify:

(1) Every pair of signals between which a flow occurs, and
(2) The times within the trace at which each flow occurs.

Each trace $\tau_{\texttt{src}}$ is searched to find every state in which a tracking label goes from being set to 0 to being set to 1. In other words, every signal–values triple $(\mathtt{s}, v, v^t)$ that is of the form $(\mathtt{s}, v, 0)$ in state σ_{i-1} and $(\mathtt{s}, v, 1)$ in state σ_i is found and the time i is noted. This is stored as the tuple $(\texttt{src}, \mathtt{s}, \{i_0, i_1, \ldots\})$, which indicates that information from src reached signal s at all times $i \in \{i_0, i_1, \ldots\}$. We call this the *time-of-flow tuple*. There can be multiple times-of-flow for a given source signal to sink signal within a single trace because the tracking label may be reset to zero by events such as resets.

Once all traces have been analyzed, the time-of-flow tuples $(\texttt{src}, \mathtt{s}, \{i_0, i_1, \ldots\})$ are organized by time. For any given set of times $\{i_0, i_1, \ldots\}$ there may be multiple discovered flows. For all traces $\tau_{\texttt{src}}$ generated by a testbench, the timing of flows from source src can be compared to the timing of flows from a second source src′; the value i will refer to the same point in the testbench.

At the end of this phase, Isadora produces two outputs. The first is a list of the unique sets of times present within time-of-flow tuples and all the corresponding signal pairs for which flow is discovered at precisely the times in the set:

$$\begin{aligned} S_{\text{flows}} = [&\langle \{i_0, i_1, \ldots\} : \{(\texttt{src}_1, \mathtt{s}_1), (\texttt{src}_2, \mathtt{s}_2), \ldots\}\rangle; \\ &\langle \{i_0', i_1', \ldots\} : \{(\texttt{src}_1{}', \mathtt{s}_1{}'), (\texttt{src}_2{}', \mathtt{s}_2{}'), \ldots\}\rangle; \ldots]. \end{aligned}$$

The same src may flow to many sinks $\mathtt{s} \in \{\mathtt{s}_1, \mathtt{s}_2, \ldots\}$ at the same times $i \in \{i_0, i_1, \ldots\}$, and the same sink s may receive information from multiple sources $\texttt{src} \in \{\texttt{src}_1, \texttt{src}_2, \ldots\}$ at the same times $i \in \{i_0, i_1, \ldots\}$. The second output from this phase is a list of source–sink pairs between which information flow is restricted:

$$S_{\text{no-flow}} = \{(\texttt{src}, \mathtt{s}), (\texttt{src}', \mathtt{s}'), \ldots\}.$$

This set comprises the information-flow restrictions of the design and may be specified with the restriction operator as src =/=> s

3.3 Mining for flow conditions

In the third phase, Isadora finds information-flow conditions. For example, if every time `src` flows to `s`, the signal `r` has the value x, Isadora infers the information-flow condition:

$\neg(\texttt{r} = x) \rightarrow \texttt{src =/=> s}.$

Isadora uses the technique of dynamic invariant detection [14] on traces to infer behaviors using pre-defined patterns. In order to isolate the conditions for information flow between two signals, Isadora uses S_{flows} to find all the trace times i at which information flows from `src` to `s` during execution of the testbench. The corresponding trace is then decomposed to produce a set of trace slices that are two clock cycles in length, one for each time i. Consider time-of-flow tuple (`src`, `s`, $[i, j, k, \ldots]$), which as a notational convenience here uses distinct letters to denote time points rather than subscripts for clarity in the following expression. Given this tuple, Isadora will produce the trace slices $\langle\sigma_{i-1}, \sigma_i\rangle$, $\langle\sigma_{j-1}, \sigma_j\rangle$, $\langle\sigma_{k-1}, \sigma_k\rangle$. These trace slices include only the functional signals from the hardware; the IFT labels are removed. Invariant detection over these functional trace slices, or trace windows of length two, allows generates predicates specifying design state prior to and concurrent with an information flow. Predicates match one of the four patterns for expressions given by the grammar in Sect. 2.

3.4 Postprocessing

Finally, Isadora performs additional analysis to find invariants that may hold over the entire trace set by running the miner on the unsliced trace. Isadora eliminates any predicate found to be a trace-set invariant. A trivial example is the invariant $\texttt{clk} = \{0, 1\}$.

The final output from postprocessing is the information-flow restrictions. Isadora can express these as multi-source to multi-sink flows, where all source–sink pairs are similarly restricted. This produces comparatively few properties, which in practice were approximately as many as the number of unique source signals, and avoids redundant information. These information-flow conditions and the information-flow restrictions discovered in Phase 2 (Sect. 3.2) make up the set of information-flow properties produced by Isadora. Two examples of properties are shown in Appendix 1.

4 Implementation

Isadora uses Tortuga Logic Radix-S to generate the security model, the Questa Advanced Simulator to simulate the security model and functional specification and generate traces, and the Daikon [14] invariant miner to find flow conditions. A Python script manages the workflow and implements flow analysis and postprocessing.

4.1 Generating traces

An automated utility identifies every signal within a design and configures Tortuga Radix-S to build the security model separately for each of these signals, or accepts a set of input signals. We run Tortuga in exploration mode, which omits cone of influence analysis, and track flows to all design state using the `$all_outputs` variable. The resulting instrumented designs are simulated in QuestaSim over a testbench (see Evaluation, Sect. 5) to produce a trace of execution.

4.2 Identifying flows

Flow identification is implemented as a Python tool that reads in the traces generated by QuestaSim and produces the set of information-flow restrictions and the time-of-flow tuples. This phase combines the bit-level taint tracking by Radix-S into signal-level tracking. Each n-bit signal in the original design is then tracked by a 1-bit taint label, which will be set to 1 at the first point in the trace that any of the component n bits in the corresponding original design signal where tainted from the specified source.

4.3 Mining flow conditions

The mining phase is built on top of the Daikon invariant generation tool [14], which was developed for use with software programs. Daikon generates invariants over state variables for each point in a program. We built a Daikon front-end in Python (approximately 800 LoC) that converts the trace data to be Daikon readable, treating the state of the design at each clock cycle as a point in a program. The front-end also removes any unused or redundant signals and outputs relevant two-clock-cycle slices as described in Sect. 3.3.

4.4 Mining flow conditions

When identifying the flows in the design, Isadora must process a number source–sink pairs equal to $n * (n - 1)$ where n is the number of signals in the original design. This corresponds to n signals each acting as a potential source for all of the other $n - 1$ signals in the design. Of course the tool may be configured to consider only specified sources or specified sinks, and this technique is demonstrated in Sect. 5.5 for an SoC evaluation.

The resulting flow conditions describe the information-flow relationship between these pairs of signals. In the case of signals that must interact, and therefore have some flow relation, but should do so under only certain restricted circumstances, this mining technique can automatically and

precisely determine the circumstances under which the flow occurs in practice. Or, in the case of two signals that were not expected to have any flow relation, this technique can provide a precise description of the error state.

5 Evaluation

We assess the following questions to evaluate Isadora:

(1) Can Isadora independently mine security properties manually developed by hardware designers?
(2) Can Isadora automatically generate properties describing Common Weakness Enumerations (CWEs) [31] over a design?
(3) Does Isadora scale well for larger designs, such as CPUs or SoCs?

5.1 Designs

We assessed Isadora on three designs, the Access Control Wrapper (ACW) [36] the PicoRV32 RISC-V CPU, and an OpenTitan Hardware Root of Trust.

An ACW wraps an AXI controller and enforces on it a *local access control policy*, which is setup and maintained by a trusted entity. The ACW checks the validity of read and write requests issued by the wrapped AXI controller and rejects those that violate the local access control policy. We study a single-controller access control system. The input signals of the ACWs are dictated by the testbench, which initializes them with the access control policies and acts as the trusted entity. The design is shown in Fig. 2. The secure operation of the ACW and Aker-based access control systems has been verified through a property-based security validation process by the designers. We evaluate how Isadora's properties compare to the manually developed security specification.

PicoRV32 CPU is a CPU core that implements the RISC-V RV32IMC Instruction Set, an open standard instruction set architecture based on established reduced instruction set computer principles. We use the PicoRV32 CPU to evaluate how well Isadora automatically generates properties describing CWEs and to evaluate how well Isadora scales on a CPU design.

The OpenTitan SoC is an open source silicon hardware root of trust (RoT). Our analysis focuses on the one-time programmable memory (otp) module (u_otp_ctrl). The OTP holds important information like encryption keys and performs scrambling and other cryptographic operations when distributing keys across the OpenTitan SoC. For example, the OTP provides keys to the FLASH and SRAM that they use for encryption and authentication. We use the OpenTitan SoC to evaluate how well Isadora may be applied to large-scale designs using larger amounts of trace data.

5.2 Isadora runtime

There are two major parts of the Isadora framework. The first part generates the IFT-enhanced traces. The second part mines those traces using Daikon. Trace generation dominated the overall runtime, scaling at a pace slightly worse than linear with number of unique signals in a design. Trace generation is parallelizable and parallelization will be necessary on larger designs.

Trace generation involves several steps. A security model is generated separately for each source signal using Radix-S. Thus in order to analyze all hardware signals (as was done with the ACW and PicoRV32), Isadora generates a separate security model for each signal. Once the security model is generated, Isadora uses QuestaSim to simulate the security model and the functional (SystemVerilog) model using a provided functional testbench. The simulation generates a trace of hardware signals over time that includes both the signals' IFT labels and their functional values.

The design sizes are given in Table 1. For the ACW, trace generation took 9h33m. For the PicoRV32 CPU, trace generation took 8h35m. For the OpenTitan, trace generation took multiple days. The SoC evaluation mining time differs in a number of ways from the ACW and CPU times. Firstly, it was performed on a different, faster system. Secondly, it has many fewer sources and much greater trace length. Lastly, the initial unoptimized run over the SoC crashed on the 18th of 22 properties after a few hours, and the 2:50 time comes from removing all clock signals. Clock signals remained in consideration for the ACW and CPU.

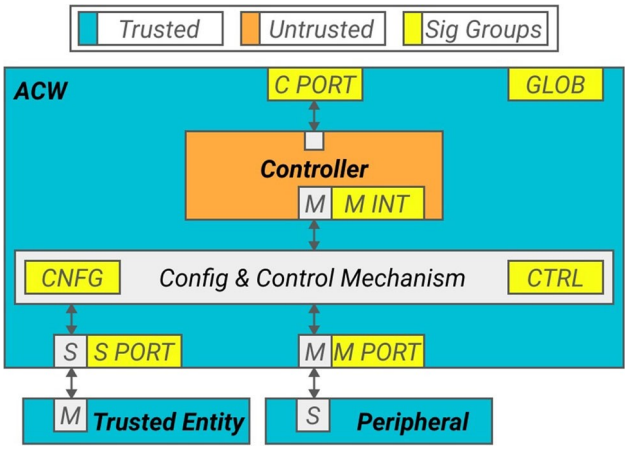

Fig. 2 Block diagram of the ACW design, with labeled signal groups

Table 1 Various size measures of the designs and trace evaluations

Design	Source signals	Total signals	LoC (verilog)	Trace cycles	Trace GBs	Daikon traces	Isadora props	Mining time (mins.)
Aker ACW	229	229	1940	598	0.7	252	303	29:51
PicoRV32 CPU	181	181	3140	1099	0.6	955	153	15:09
OpenTitan SoC	5	365	1.8 M	3.6 B	140	24117	22	2:50

5.2.1 Theoretical performance gains

When parallelizing all trace generation and all case mining, Isadora could theoretically evaluate the ACW in less than five minutes. Parallelizing the first phase requires a Radix-S and QuestaSim instance for each source signal, and each trace is generated in approximately 100 s. Further, the trace generation time is dominated by write-to-disk, and performance engineering techniques could likely reduce it significantly, such as by changing trace encoding or piping directly to later phases. Parallelizing the second phase requires only a Python instance for each source signal, and takes between 1 and 2 s per trace. Parallelizing the third phase requires a Daikon instance for each flow case, usually roughly the same number as unique sources, and takes between 10 and 30 s per flow case. Maximally parallelized, this gives a design-to-specification time of under four minutes for the Aker ACW and for similarly sized designs, including the PicoRV32 CPU.

Parallelization supports scaling up to SoC designs. Our evaluation suggests that mining time increases logarithmically with respect to trace length, and that by decomposing large designs into modules and submodules the mining stage can be applied to trace sets that stay under the threshold for which unique signals begin to dominate mining costs, usually around 2000-4000 signals, while recomposing system-wide properties in postprocessing. We also found that in the SoC design there were frequently signals across multiple modules that always held the same value, and in such a case many more signals may be considered before the internal inference engine within the miner is overwhelmed.

5.3 Designer-specified security properties

For the ACW, we compared Isadora's output against security assertions developed by the Aker [36] designers using the Common Weakness Enumerations (CWE) database [31] as a guide. These assertions, the CWEs described, and the results of Isadora on the ACW are shown in Table 2. For each assertion, Isadora mined either a property containing the assertion or found both a violation and the violating conditions for each assertion.

We reported the observed violations to the designers who determined that the design was secure and the error was in the manually developed specifications: an information-flow condition had been incorrectly specified as an information-flow restriction. Isadora found the correct information-flow condition.

Only 9 Isadora properties, out of 303 total Isadora properties generated, were required to cover the designer-provided assertions, including conditions specifying violations. The Isadora output properties may contain many source or sink signals that flow concurrently and their corresponding conditions, whereas the designers' assertions each considered a single source and sink. For example, on the ACW nine distinct read channel registers always flow to a corresponding read channel output wire at the same time, so Isadora outputs a single property for this design state. This state included the reset signal and a configuration signal both set to nonzero values, which were captured as flow conditions, demonstrating correct design implementation. This single Isadora property captured 18 low-level assertions related to multiple CWEs.

5.4 Automatic property generation

For the two designs with full trace sets, the ACW and PicoRV32 CPU, Isadora generates a specification describing all information flows and their conditions with hundreds of properties. To assess whether these properties are security properties, for each design we randomly selected 10 of the 303 or 153 total properties (using Python random.randint) and assessed their relevance to security.

We use CWEs as a metric to evaluate the security relevance of Isadora output properties. To do so, for each design, we first determine which CWEs apply to the design. For both the Aker ACW and PicoRV32 CPU, we used the Radix Coverage for Hardware Common Weakness Enumeration (CWE) Guide [28] to provide a list of CWEs that specifically apply to hardware. We considered each documented CWE for both designs. CWEs, while design agnostic, may refer to design features not present in the ACW or PicoRV32CPU or may not refer to information flows. High-level descriptions in multiple CWEs may correspond to the same low-level behavior for a design and we consider these CWEs together.

Information-flow CWEs for hardware describe source signals, sink signals, and possibly conditions. CWEs provide high-level descriptions, but Isadora targets an RTL definition. To apply these high-level descriptions to RTL, we first group signals for a design by inspecting Verilog files and,

Table 2 Isadora performance versus Aker assertions, on the ACW

Source signal	Sink signal	Predicate condition	Aker props	Isadora props	CWEs	Found?
M PORT	M INT	GLOB	19	2, 40	1258, 1266, 1270	✓
INT	M PORT		19	43, 53	1271, 1272 1280	
M PORT	M INT	C PORT	19	54, 204	1258, 1270	
M INT	M PORT		19	214	1272 1280	
S PORT	CNFG	–	4	2, 6	1269, 1272, 1280	✗

Table 3 The 14 CWEs considered for the ACW

CWE(s)	Description
1220	Read/write channel separation
1221-1259-1271	Correct initialization, reset, defaults
1258-1266-1270-1272	Access controls use operating modes
1274-1283	Anomaly registers log transactions
1280	Control checks precede access
1267-1269-1282	Configuration/user port separation

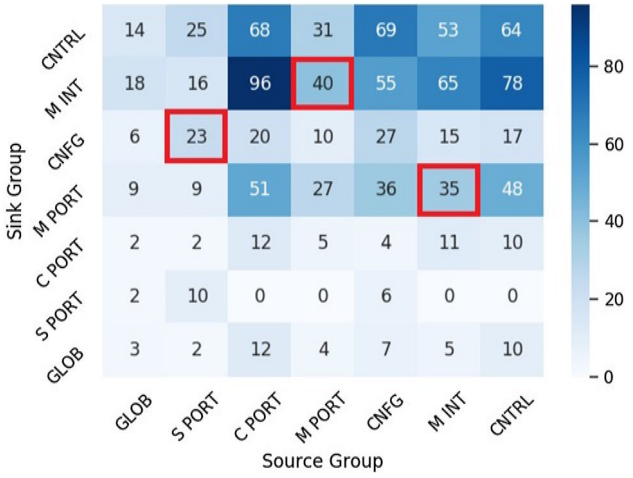

Fig. 3 Group-to-group conditional flow heatmap for the ACW

if available, designer notes. With the groups established, we label every property by which group-to-group flows they contain. We also determine which source–sink flows could be described in CWEs, which often correspond to, or even match exactly, a signal group. We use these groups to find CWE-relevant, low-level signals as sources, sinks, and conditions in an Isadora property. We also use these groups to characterize the relative frequency of conditional flows between different groups, which we present as heatmaps in the following subsections.

5.4.1 ACW information-flow conditions

Over the ACW we assess fourteen CWEs which we map to five plain-language descriptions of the design features, as shown in Table 3.

For the ACW signal groups, all signals were helpfully placed into groups by the designer and labeled within the design. The design contained seven distinct labeled groups:

- 'GLOB'—Global ports
- 'S PORT'—AXI secondary (S) interface ports of the ACW
- 'C PORT'—Connections to non-AXI ports of the controller
- 'M PORT'—AXI main (M) interface ports of the ACW
- 'CNFG'—Configuration signals
- 'M INT'—AXI M interface ports of the controller
- 'CTRL'—Control logic signals

GLOB signals are clock, reset, and interrupt lines. S PORT represents the signals that the trusted entity T uses to configure the ACW. C PORT represents the signals which are used to configure the controller C to generate traffic for testing. M PORT carries traffic between the peripheral P and the ACW's control mechanism. CNFG represents the design elements which manage and store the configuration of the ACW. M INT carries the traffic between the ACW's control mechanism and the controller. If it is legal according to the ACW's configuration, the control mechanism will send M INT traffic to M PORT and vice versa. CTRL represents the design elements of the aforementioned control mechanism.

First consider the heatmap view of the ACW in Fig. 3. In this view, all of the Aker properties fall into just 3 of the 49 categories; these are outlined in red. Further, all of the violations were found with S PORT to CNFG flows, while all satisfied assertions were flows between M INT and M PORT. Another interesting result visible in the heatmap is the infrequent flows into S PORT, which is used by the trusted entity to program the ACW. Most of the design features should not be able to reprogram the access control policy, so finding no flows along these cases provides a visual representation of secure design implementation with respect to these features.

For the ACW, all ten sampled properties encode CWE-defined behavior to prevent common weaknesses, as shown

Table 4 Sampled Isadora properties on the ACW

#	Description	1220	1221+	1258+	1274+	1280
3	Control check for first read request after reset	✓		✓		✓
10	Secure power-on		✓			
37	Anomalies and memory control set after reset	✓		✓	✓	✓
96	*T* via S PORT configures ACW	✓			✓	✓
106	Interrupts respect channel separation	✓				
154	Base address not visible to *P* during reset			✓		
163	Write transaction legality flows to *P*	✓				
227	Write channel anomaly register updates	✓			✓	
239	Write validity respects channel separation, reset	✓		✓		
252	Read validity respects channel separation, reset	✓		✓		

Table 5 The 18 CWEs considered for PicoRV32 CPU

CWE(s)	Description
276-1221-1271	Correct initialization, reset, defaults
440-1234-1280-1299	Memory accesses pass validity checks
1190	Memory isolated before reset
1191-1243-1244...	Debug signals do not interfere with
-1258-1295-1313	...any other signals
1245	Hardware state machine correctness
1252-1254-1264	Data and control separation

in Table 4. In this table, the columns labeled by a CWE number and a '+' refer to all the CWEs given in a row of Table 3. Eight out of the ten properties provide separation between read and write channels, which constitutes the main functionality of the ACW module. CWEs 1267, 1269, and 1282 describe information-flow restrictions, so they are not present within the samples drawn from the numbered information-flow restrictions, but we were able to verify they are included in Isadora's set of information-flow restrictions.

5.4.2 CPU information-flow conditions

Over the PicoRV32 CPU, we assess eighteen CWEs which we map to seven plain-language descriptions of the design features, as shown in Table 5.

The PicoRV32 CPU had no designer-specified signal groups so we used comments in the code, signal names, and code inspection to group all signals.

- 'out'—Output registers
- 'int'—Internal registers
- 'mem'—Memory interface
- 'ins'—Instruction registers
- 'dec'—Decoder
- 'dbg'—Debug signals and state
- 'msm'—Main state machine

The memory interface and the main state machine were indicated by comments in the code. The instruction registers, the decoder, and debug all appeared under one disproportionately large section described as the instruction decoder. Debug was grouped by name after manual analysis found registers in this region prefixed with 'dbg_', 'q_', or 'cached_' to interact with and only with one another. Instruction registers prefixed 'instr_' all operate similarly to each other and differently than the remaining decoder signals, which were placed in the main decoder group. Internal signals were the remaining unlabeled signals that appeared early within the design, such as program and cycle counters and interrupt signals, and the output registers were all signals declared as output registers.

First consider the heatmap view of the PicoRV32 CPU in Fig. 4. An interesting result visible in the heatmap is the flow isolation from debug signals to the rest of the design. Many exploits, both known and anticipated, target debug information leakage, and the heatmap shows this entire class of weakness is absent.

For the PicoRV32 CPU we find eight of ten sampled properties encode CWE-defined behavior to prevent common weaknesses. We present these results in Table 6. The columns labeled by a CWE number and a '+' refer to all the CWEs given in a row of Table 5. The remaining properties were single source or single sink properties representing a logical combination inside the decoder, and captured only functional correctness.

5.5 SoC evaluation

When extending Isadora to work on the larger OpenTitan design, we learned a number of lessons on scalability and on information flow in SoC designs. For performance, we can decompose SoC designs module-wise, and evaluate modules with their immediate submodules to generate properties across module hierarchies. Fortunately, Isadora experiences minimal time costs from considering much longer traces

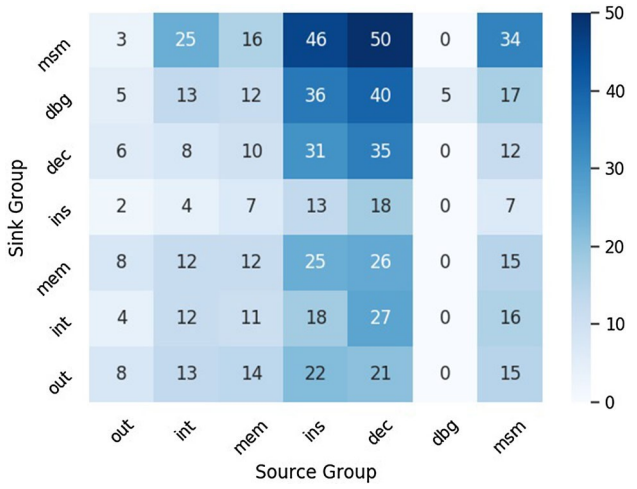

Fig. 4 Group-to-group conditional flow heatmap for PicoRV32 CPU

associated with full systems engaging in more advanced computations. In SoC designs with microarchitectural state, we may want to study many additional signals (Verilog wires) within a module, and by exploring designs module-wise we have the efficiency to do so. For SoC designs, it is especially helpful to specify fixed sources, either by signal or by a signal with a Verilog condition. SoC traces contain common patterns that can be considered separately when preparing trace for specification mining and removed to improve performance, such as clock ticks or other timing updates, to partition the design space. Omitting clock signals from consideration improved performance from routinely crashing the data miner after hours, to completing all data mining in under 3 min. Together, these techniques scale Isadora to SoC designs.

For the SoC, we generated full design traces for five preselected sources known to be of security interest. We selected two key registers:

(1) RndCnstKey—Random constant used as scrambling key
(2) UserKeys—Secret keys stored in OTP memory

and three seed registers used by the OTP's Key Derivation Interface to derive keys for various applications:

(1) sram_data_key_seed—For use in the SRAM Controller
(2) flash_data_key_seed—For use in the Flash Controller
(3) flash_addr_key_seed—For use in the Flash Controller

All sources were located in the u_otp_ctrl module, OpenTitan's one-time-programmable (OTP) memory controller, which served as the focus of our exploration. The flow sinks included signals in u_otp_ctrl as well as its first level submodules. We present the in-module signals and other modules in Table 7.

As a practical matter for evaluation, the size of an SoC made generating the hundreds of traces used for the ACW and CPU prohibitively costly for a single instance of an underoptimized research version of the tool. The resulting traces still contained over a hundred times more trace data than the CPU SoC sets over just these five sources. For each source, we only considered 'near' flows within the source module and its immediate submodules, with the understanding that information flow could then be tracked through the entire design using these intermediate signals.

Excluding flows to clock signals and flows at only time zero, there are 16 distinct flow cases on the SoC that are presented in Table 8. We present the number of clock cycles from the beginning of the trace at which the flow occurred, how many distinct times a flow occurred on the trace, the number of conditions discovered by Daikon, and the source and sink signals. For the SoC design, we did not proceed with the full postprocessing stage to avoid running Daikon over the entire SoC trace, so the number of Daikon conditions is presented as a proxy for the number of clauses of a postprocessed information-flow condition. These properties holistically provide interest insight into how information flows through the design.

Table 6 Sampled Isadora properties on PicoRV32 CPU

#	Description	276+	440+	1190	1191+	1245	1252+
1	No decoder leakage via debug				✓		
16	Instructions update state machine		✓			✓	
30	Decoder updates state machine		✓				
47	No state machine leakage via debug				✓		
52	SLT updates state machine					✓	
66	Handling of jump and load			✓	✓		✓
79	Loads update state machine					✓	
113	Decoder internal update						
130	Write validity respects reset					✓	
144	Decoder internal update						

Table 7 OpenTitan SoC module and register sinks, with brief descriptions

Sink module	Description
u_otp_ctrl_kdi	Key Derivation Interface (KDI) for deriving static/ephemeral keys for use in scrambling mechanisms
u_otp	Physical OTP memory
u_prim_edn_req	Interfaces entropy distribution network for reseeding and ephemeral key derivation
u_otp_ctrl_dai	Direct Access Interface (DAI), an access-control-enabled interface to OTP memory
Sink signals	**Description**
core_tl_o	TileLink-UL bus device interface
edn_o	Entropy request to the entropy distribution network for LFSR reseeding and ephemeral key derivation
intr_otp_error_o	Interrupt indicating an error has occurred in the OTP controller
intr_otp_operation_done_o	Interrupt indicating a direct access command or digest calculation operation has completed
otbn_otp_key_i	Key derivation requests from OTBN DMEM scrambling device
otp_alert_o	Signals carrying alerts from the OTP controller to analog sensor top (AST) module
otp_ast_pwr_seq_h_i	Power sequencing signals coming from AST (VCC domain)
otp_ext_voltage_h_io	External voltage for OTP
otp_obs_o	Observability signal from the OTP controller to analog sensor top (AST) module
prim_tl_o	TileLink-UL bus device interface
pwr_otp_i	Initialization request coming from power manager
rst_ni	Primary reset signal to OTP controller
scan_en_i	Enable signal for DFT-related scan chains

Table 8 OpenTitan SoC Properties. Times given are approximations for brevity. Starred (*) times diverged after the 7th co-occurrence. Double starred (**) sinks also included rst_ni, rst_edn_ni, intr_otp_operation_done_o, intr_otp_error_o

Index	Flow time(s)	No. of occurrences	No. of conditions	Conditional flow relation(s)
01	0	1	2295	127 total flows
02	$0 \wedge 1396538900$	2	2198	src[1-5] =?=> edn_o
03	$0 \wedge 1396940500 + 3313200n$	14	92712	src[1-5] =?=> prim_tl_o
04	$0 \wedge 1398647300 + 1706800n*$	52	111833	src[3-5] =?=> core_tl_o
05	$0 \wedge 1398647300 + 1706800n*$	52	111822	src[1] =?=> core_tl_o
06	$0 \wedge 1398647300 + 1706800n*$	52	111811	src[2] =?=> core_tl_o
07	$0 \wedge 1408285700$	2	2083	src[2] =?=> otbn_otp_key_i+**
08	$0 \wedge 1408787700 + 3313200n$	70	114379	src[1-5] =?=> otp_ast_pwr_seq_h_i
09	$0 \wedge 1409892100$	2	2084	src[1] =?=> otbn_otp_key_i+**
10	$0 \wedge 1412100900 + 3313200n$	69	113763	src[1-5] =?=> pwr_otp_i
11	$0 \wedge 1426056500 \wedge 1446538100$	3	2096	src[1-5] =?=> otp_obs_o
12	$0 \wedge 1459891300 + 9939600n$	12	9517	src[1-5] =?=> data_copy
13	$0 \wedge 1463706500$	2	2085	src[3-5] =?=> rst_ni
14	$0 \wedge 1479870900 + 3313200n$	5	2040	src[1-5] =?=> data_load
15	1396840100	1	2290	src[1-5] =?=> otp_alert_o
16	1636093300	1	2239	src[1-5] =?=> otp_ext_voltage_h_io
17	1636193700	1	2295	src[1-5] =?=> scan_en_i

For example, property 06 specifies a flow from the secret key stored in the OTP to the core_tl_o register, a TileLink-UL bus device interface first occurred at time zero, then approximately 1.4 billion picoseconds after the trace latter, then approximately regularly every 1.7 million picoseconds for total of 52 total occurrences. Isadora found 111811 predicate expressions that held at each of these 52 time points.

We find that generally for flows that occurred multiple times, they reoccurred some fixed number of cycles from the previous occurrence and appear roughly periodic. Of the 16 flow cases, 14 flow into a single signal. 13 include a flow at time zero, showing that there is an initial flow during system initialization before the design successfully completes reset. Ten are co-occurring flows from all sources to a single sink. Seven occur more than ten times. The three seed signals always flow concurrently to the same sink. The key signals do flow into the same signals but at different times. These 16 properties capture a total of 68 distinct source–sink pairs. Two signals, core_tl_o and rst_ni, are a sink for all five sources but do not receive flows at the same time, being reached at separate time points by the seed signals and each key signal.

The time zero only flows are distinct between key and seed signals, though the same within keys and within seeds, and contain many overlapping signals such as core_tl_i. At time zero there are 127 unique flows, suggesting that information flow is less restricted in the design prior to successfully completing reset.

We tracked flows to unique signals by name, yet many of these signals existed across multiple of the considered submodules, so they constitute multiple flows to corresponding signals in different modules. For example, core_tl_o only exists as a single register signal in the u_otp_ctrl module that served as the top module for our analysis, while the edn_o signal corresponded to a wire in both the u_otp_ctrl and its u_prim_edn_req module. In the full design, there are a total of 17 edn_o wires traversing a nested module structure for which this tracked flow is applicable.

The specificity of flow conditions was also strongly bimodal, with all flows either having between 2000 and 2300 Daikon conditions or between 90000 and 115000 Daikon conditions. Any time there were 5 or fewer flows was in the former group and all others were in the latter group.

As with the ACW and CPU, we generate a heatmap of the conditional flows, two of which are shown in Figs. 5 and 6. The first is a signal-to-module view. Rather than rely on developing families for signals from manual code inspection, we track flows across module boundaries. That is, we log each flow from the five designated sources within the top-level u_otp_ctrl module, and track how many flows reach each of its immediate submodules. The second is signal-to-signal view within a single module.

By viewing the signal-to-module heatmap, we can also see additional relationships between the signals. Flows overwhelming occur within the u_otp_ctrl top-level module, with an only a few irregular flows being directed to submodules. Some modules are reached much more often than

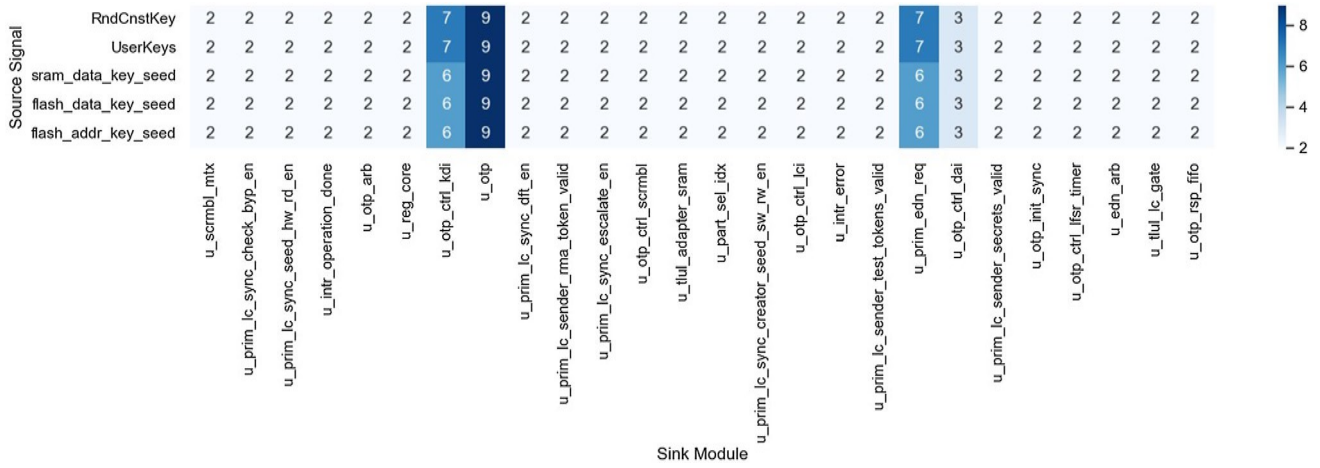

Fig. 5 Signal-to-module conditional flow heatmap for OpenTitan SoC

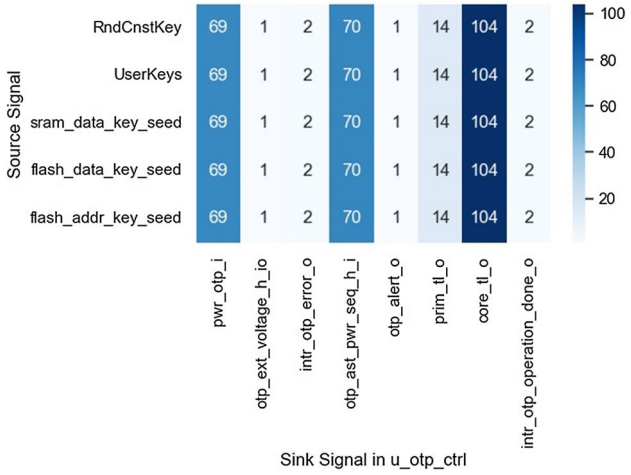

Fig. 6 Signal-to-signal conditional flow heatmap for an OpenTitan SoC module

others, especially u_otp, followed by u_otp_ctrl_kdi and u_prim_edn_req. Of the remaining modules, only u_otp_ctrl_dai receives more than 2 flows from any source, and all other submodules receive exactly two from each source. Additionally, we note that every immediate submodule is captured in this heatmap, so there are no submodules that receive no information flows. Given that many conditional flows occurred from all five sources simultaneously, it is not necessarily surprising to see the common case of all but two submodules being reached from each of the five sources the same number of times, with a single difference of one instance (6 vs 7) between seed and key signals for the u_otp_ctrl_kdi and u_prim_edn_req modules.

By viewing the signal-to-signal heatmap show that while flows from the various source to sinks do not necessarily occur at the same time, they occur very close to the same number of times, and exactly the same number of times within the u_otp_ctrl module, though at different points in time. This suggests that at least with u_otp_ctrl, whatever events are driving information flow from the designated source signals, the events have the same intramodule effect except in terms of timing. One interesting result with regard to timing perhaps is the few signals reached only at time zero by seed signals may be reached after reset is completed by the key signals, such as intr_otp_operation_done_o. Further, this signal-to-signal mapping also highlights a few signals to be considered especially carefully in security validation efforts, namely the core_tl_o, pwr_otp_i, and otp_ast_pwr_seq_h_i signals, which together account for approximately half of total times information flowed from any of the five sources.

6 Discussion

In this section, we discuss the some limitations and further work.

False positives may be introduced by insufficient trace coverage, by limitations of information-flow tracking, or by incorrectly classifying functional properties as security properties. Sampling output properties found a 10% false positivity rate with respect to misclassification. This rate is discussed in greater detail in Sect. 6.2.

With regard to false negatives, they fall into two cases: known and unknown. Isadora captured all known assertions for the ACW (Sect. 5.3). For ACW and CPU properties (Sect. 5.4), the sampled properties partially addressed all CWEs manually determined to be relevant, but no CWE was completely covered within the sampled properties. A manual inspection is needed to rule out the possibility of false negatives with respect to CWEs. Unknown false negatives can arise from limitations in trace coverage or in logical specificity.

6.1 Trace reliance

As a specification miner, Isadora relies on traces. The second stage relies on traces with sufficient case coverage to drive information flow through all channels present in the design. The third stage relies on traces to infer flow predicates. Over buggy hardware, these predicates may form a specification describing buggy behavior. Traces may not cover all cases that can be reached by a design or even occur during normal design operation.

Traces may not precisely describe some design features. For example, when considering property number 154 on the ACW, one of the sampled properties, Isadora found predicates that ARLEN_wire and AWLEN_wire are both set to be exactly 8 for any flow to occur. This property is shown in full in Appendix 2.

The AxLEN_wire signals set transaction burst size for reads and writes. For transactions in write channels, the ARLEN_wire value should be irrelevant, and this clause within the broader property constitutes a likely false positive.

The AWLEN_wire is a different case. In a properly configured write channel supporting transactions, this signal would necessarily be nonzero, and for wrapping bursts must be a power of two, but manual inspection of the code provides no indication the value must be precisely 8.

While Isadora is testbench reliant, testbench generation is an active area of research, and is more fully explored in related works such as Meng et al. [29], which studies concolic testing for RTL.

6.2 Functional properties

When using CWE-relevance as the metric, Isadora does include functional properties in its output, as shown in Table 6. Sampling output properties found a 10% false positivity rate with respect to misclassification for the sampled properties from both designs, with 0 of 10 properties found to be false positives over the ACW, and 2 of 10 properties found to be false positives over the PicoRV32 RISC-V CPU.

We attribute finding functional properties solely on CPU primarily to differences in design and testbench. The ACW studied was the target of validation efforts related to information flow, and the testbench we used was developed as part of those efforts. Further, as an access control module, by nature much of its functionality was relevant to secure access control.

With RISC-V, a minimal testbench was used that was intended only to run the design in an environment without access to the full RISC-V toolchain (such as our simulation environment for IFT-enhanced trace generation), and much of the design was devoted to behavior for which CWEs did not apply, such as logical updates during instruction decoding. One example of an Isadora property classified as functional is shown in Appendix 3.

6.3 Initialization flows

Manual examination of output properties suggests patterns of information flow during initialization, which is the first 4 cycles for the ACW, first 80 for RISC-V, and first cycle for the SoC, are highly dissimilar to later flows. During initialization, Isadora discovers flow conditions referencing signals with unknown states. Isadora also finds concurrent flows between elements for which no concurrent flows occur after reset. Because conditions are inferred from comingled trace slices from during and after initialization, the output properties may be insufficiently precise to capture secure behavior related to this boundary.

6.4 Scalability

Isadora encounters a few challenges when scaling to SoC designs. First, as a practical matter, we restricted our evaluation over the SoC to only 5 sources, while we used hundreds each for the ACW and CPU. Any time sources are being restricted, especially by specifying them manually, we most be cognizant that some flow between two signals could seem relevant while neither signal may independently seem relevant for any number of reasons. Future work on Isadora can address this challenge in part by supporting parallelization and by using less memory intensive storage for traces to improve performance.

Second, Isadora relied on using modules with the OpenTitan SoC to divide the large state space of the design. However, it is not necessarily the case that in every design modules are well suited to this task, or that a sophisticated adversary may not target the specific challenge of infer flow conditions across multiple module boundaries.

There are also ways to approach scalability as a research question rather than an engineering task. For example, while the Verilog notion of a module is helpful as a starting point, further work could address the task of how to partition the design to reduce mining and trace generation costs, such as by extracting control signals as is done in the Astarte miner [13]. Astarte tackled scaling up to CISC designs by using a two-stage mining processor over design signals (in contrast to the two-stage Isadora mining processor that considered design signals only in the second stage). Astarte examined candidate control bits, each individual bit within a known control register, to evaluate whether bits encoding behavior modifying state for the design. In the first stage, Astarte examined all registers and individual bits within registers (in practice the n-bit subfields did not appear to correspond to meaningful properties, but the functionality was maintained) to find families of bits which changed under the same circumstances

(that is, at the same time points). In the second stage, trace properties were generated for the subset of the trace points where bits were to either zero or to one. For many bits, the trace properties across these two subsets of the trace points were identical to the trace properties found across the trace as a whole, but for a few bits, the design had demonstrably different behavior.

For Isadora, we can use this control bit (or register or n-bit register subfield) oriented technique both with respect to individual control bits within the design signals but also with respect to individual tracking labels. While this may generate many possible candidate inputs to the invariant detection stage of our mining process, each run of this stage is individually inexpensive and does not require redoing the trace generation stage that overwhelmingly dominated our time and memory costs.

7 Related work

7.1 Properties of hardware designs

Automatic extraction of security-critical assertions from hardware designs enables assertion-based verification without first manually defining properties [23]. IODINE looks for possible instances of known design patterns, such as one-hot encoding or mutual exclusion between signals. [20]. More recent papers use data mining of simulation traces to extract more detailed assertions [5,21] or temporal properties [26]. Recent work has focused on mining temporal properties from execution traces [7–9,26]. A combination of static and dynamic analysis extracts word-level properties [27].

The first security properties developed for hardware designs were manually crafted [3,4,22]. SCIFinder semi-automatically generates security-critical properties for a RISC processor design [43] and Astarte generates security-critical properties for x86 [13]. Recent hackathons have revealed the types of properties needed to find exploitable bugs for RISC SoCs [10].

7.2 Mining specifications for software

The foundational work in specification mining comes from the software domain [1] in which execution traces are examined to infer temporal specifications as regular expressions. Subsequent work used both static and dynamic traces to filter candidate specifications [39]. More recent work has tackled imperfect execution traces [41], and the complexity of the search space [16,17,35]. Daikon, which produces invariants rather than temporal properties, learns properties that express desired semantics of a program [14].

In the software domain, a number of papers have developed security-specific specification mining tools. These tools use human specified rules [37], observe instances of deviant behavior [14,30,32], or identify instances of known bugs [40].

8 Conclusion

We implemented and presented Isadora, a specification miner for creating information-flow specifications of hardware designs. By combining information-flow tracking and specification mining, we are able to produce security relevant information-flow properties of a design without prior knowledge of security agreements or specifications.

We show that Isadora characterizes the flow relations between all elements of a design and identifies important information-flow security properties of an ACW and a CPU according to Common Weakness Enumerations and that it scales to accommodate full SoC designs.

Acknowledgements We thank our reviewers for their insightful comments and suggestions. This material is based upon work supported by the National Science Foundation under Grant Nos. CNS-1816637 and 1718586, by the Semiconductor Research Corporation, and by Intel. Any opinions, findings, conclusions, and recommendations expressed in this paper are solely those of the authors.

A Sample properties

In this section, we show examples of Isadora output.

A1 Case 154: ACW security property

```
case 154: 2_121_250_379_543
_src_ in {w_base_addr_wire, M_AXI_AWREADY_wire,
AW_CH_DIS,  w_max_outs_wire, AW_ILLEGAL_REQ,
w_num_trans_wire, AW_STATE,  AW_CH_EN}
=/=>
_snk_ in {M_AXI_WDATA}
unless
0 != _inv_ in {ADDR_LSB, ARESETN, M_AXI_ARBURST_wire,
M_AXI_ARCACHE_wire, M_AXI_ARLEN_wire, M_AXI_ARREADY,
M_AXI_ARSIZE_wire, M_AXI_AWBURST_wire,
M_AXI_AWCACHE_wire, M_AXI_AWLEN_wire, M_AXI_AWREADY,
M_AXI_AWSIZE_wire, M_AXI_BREADY, M_AXI_BREADY_wire,
M_AXI_WREADY, M_AXI_WREADY_wire, M_AXI_WSTRB_wire,
OPT_MEM_ADDR_BITS, S_AXI_CTRL_BREADY,
S_AXI_CTRL_RREADY, data_val_wire, r_burst_len_wire,
r_displ_wire, r_max_outs_wire, r_num_trans_wire,
r_phase_wire, w_burst_len_wire, w_displ_wire,
w_max_outs_wire, w_num_trans_wire, w_phase_wire}
```

Fig. 7 An example of an Isadora property, Case 154, over the Aker ACW

To consider the output properties of Isadora, Fig. 7 shows an example of Isadora output, Case 154 of the 303 output properties over the ACW module. This a case that was sampled during evaluation. Here the condition predicates shown are signal equality testing versus zero. Other predicates are captured within the workflow but not propagated to individual properties formatted for output.

A visible difference between an Isadora output property and the property grammar of Sect. 2 is that at output stage Isadora properties may specify multiple source signals, may consider multiple sink signals though do not do so in this case, and may contain multiple invariants as conditions.

Case 154 includes an example of a flow condition between internal and peripheral visible signals in addition to specifying other aspects of design behavior. This is similar to the example of write readiness from Sect. 2, but in Case 154, the flow is from the internal signal to the peripheral, though the power state predicate is identical. Of note, as in the case of write readiness, this flow occurs exclusively within the write channel, as denoted by the 'W' present in ready wire and the data register.

AWREADY_int =/=> WDATA unless (ARESETN $\neq 0$).

A2 Case 144: ACW functional property

One example of an Isadora property classified as functional, with truncated flow conditions, is presented in Fig. 8, and captures a logical update to an internal decoder signal. This additionally shows an example of a property over multiple sinks, a single source, and for which there are predicates capturing both equality and inequality to zero.

```
case 144: 128
_src_ in {instr_lw}
=/=>
_snk_ in {is_slti_blt_slt, is_sltiu_bltu_sltu}
unless
0 == _r_ in {alu_eq, alu_shl, alu_shr,  ... }
0 != _r_ in {alu_add_sub, alu_lts, alu_ltu, ... }
```

Fig. 8 An example of an Isadora property, Case 144, over RISC-V

References

1. Ammons, G., Bodík, R., and Larus, J.R.: Mining specifications. In 29th Symposium on Principles of Programming Languages (POPL). ACM, 4–16. (2002) https://doi.org/10.1145/503272.503275
2. Ardeshiricham, A., Hu, W., Marxen, J., Kastner, R.: Register transfer level information flow tracking for provably secure hardware design. In Design. Automation Test in Europe Conference Exhibition (DATE) **2017**, 1691–1696 (2017)
3. Bilzor, M., Huffmire, T., Irvine, C., Levin, T.: Security checkers: Detecting processor malicious inclusions at runtime. In International Symposium on Hardware-Oriented Security and Trust (HOST). IEEE, 34–39 (2011)
4. Brown, M.: Cross-validation processor specifications. Master's thesis, University of North Carolina at Chapel Hill (2017)
5. Chang, P.-H., Wang, L.C.: Automatic assertion extraction via sequential data mining of simulation traces. In 15th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 607–612 (2010)
6. Clarkson, M.R., Schneider, F.B.: Hyperproperties. J. Comput. Secur. **18**6, 1157–1210. (2010). http://dl.acm.org/citation.cfm?id=1891823.1891830
7. Danese, A., Ghasempouri, T., Pravadelli, G.: Automatic extraction of assertions from execution traces of behavioural models. In Design, Automation Test in Europe Conference Exhibition (DATE), 67–72 (2015)
8. Danese, A., Pravadelli, G., Zandonà, I.: Automatic generation of power state machines through dynamic mining of temporal assertions. In Design, Automation Test in Europe Conference Exhibition (DATE), 606–611 (2016)
9. Danese, A., Riva, N.D., Pravadelli, G.: A-TEAM: Automatic template-based assertion miner. In 54th Design Automation Conference (DAC). ACM/EDAC/IEEE, 1–6 (2017)
10. Dessouky, G., Gens, D., Haney, P., Persyn, G., Kanuparthi, A., Khattri, H., Fung, J.M., Sadeghi, A.-R., Rajendran, J.: Hardfails: Insights into software-exploitable hardware bugs. In 28th USENIX Security Symposium. USENIX Association, 213–230 (2019)
11. Deutschbein, C., Sturton, C.: Mining security critical linear temporal logic specifications for processors. In International Workshop on Microprocessor and SoC Test, Security, and Verification (MTV). IEEE (2018)
12. Deutschbein, C., Sturton, C.: Evaluating security specification mining for a cisc architecture. In 2020 IEEE International Symposium on Hardware Oriented Security and Trust (HOST), 164–175 (2020)
13. Deutschbein, C., Sturton, C.: Evaluating security specification mining for a CISC architecture. In Symposium on Hardware Oriented Security and Trust (HOST). IEEE (2020)
14. Ernst, M.D., Perkins, J.H., Guo, P.J., McCamant, S., Pacheco, C., Tschantz, M.S., Xiao, C.: The Daikon system for dynamic detection of likely invariants. Sci. Comput. Program. **69**(1–3), 35–45 (2007). https://doi.org/10.1016/j.scico.2007.01.015
15. Farzana, N., Rahman, F., Tehranipoor, M., Farahmandi, F.: Soc security verification using property checking. In 2019 IEEE International Test Conference (ITC), 1–10 (2019)
16. Gabel, M., Su, Z.: Javert: Fully automatic mining of general temporal properties from dynamic traces. In 16th International Symposium on Foundations of Software Engineering (FSE). ACM, 339–349. (2008a). https://doi.org/10.1145/1453101.1453150
17. Gabel, M., Su, Z.: Symbolic mining of temporal specifications. In 30th International Conference on Software Engineering (ICSE). ACM, 51–60. (2008b). https://doi.org/10.1145/1368088.1368096
18. Goguen, J.A., Meseguer, J.: Security policies and security models. In 1982 IEEE Symposium on Security and Privacy, 11–11 (1982)
19. Hangal, S., Chandra, N., Narayanan, S., Chakravorty, S.: IODINE: a tool to automatically infer dynamic invariants for hardware designs. In 42nd annual Design Automation Conference. ACM, 775–778 (2005a)
20. Hangal, S., Narayanan, S., Chandra, N., Chakravorty, S.: IODINE: A tool to automatically infer dynamic invariants for hardware designs. In 42nd Design Automation Conference (DAC). IEEE (2005b)
21. Hertz, S., Sheridan, D., Vasudevan, S.: Mining hardware assertions with guidance from static analysis. Trans Comput-Aided Design Integr Circuits Syst **32**(6), 952–965 (2013)

22. Hicks, M., Sturton, C., King, S.T., Smith, J.M.: SPECS: A lightweight runtime mechanism for protecting software from security-critical processor bugs. In Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM, 517–529 (2015) https://doi.org/10.1145/2694344.2694366
23. Hu, W., Althoff, A., Ardeshiricham, A., Kastner, R.: Towards property driven hardware security. In 2016 17th International Workshop on Microprocessor and SOC Test and Verification (MTV). IEEE, 51–56 (2016)
24. Hu, W., Ardeshiricham, A., Kastner, R.: Hardware information flow tracking. ACM Comput. Surv. **54**, 4 (2021)
25. Hu, W., Mu, D., Oberg, J., Mao, B., Tiwari, M., Sherwood, T., Kastner, R.: Gate-level information flow tracking for security lattices. ACM Trans. Des. Autom. Electron. Syst. **20**, 1 (2014). https://doi.org/10.1145/2676548
26. Li, W., Forin, A., Seshia, S.A.: Scalable specification mining for verification and diagnosis. In 47th Design Automation Conference (DAC). ACM, 755–760 (2010). https://doi.org/10.1145/1837274.1837466
27. Liu, L., Lin, C., Vasudevan, S.: Word level feature discovery to enhance quality of assertion mining. In International Conference on Computer-Aided Design (ICCAD). IEEE/ACM, 210–217 (2012)
28. Logic, T.: Radix Coverage for Hardware Common Weakness Enumeration (CWE) Guide
29. Meng, X., Kundu, S., Kanuparthi, A.K., Basu, K.: Rtl-contest: Concolic testing on rtl for detecting security vulnerabilities. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1 (2021)
30. Min, C., Kashyap, S., Lee, B., Song, C., Kim, T.: Cross-checking semantic correctness: The case of finding file system bugs. In 25th Symposium on Operating Systems Principles (SOSP). ACM, 361–377. (2015). https://doi.org/10.1145/2815400.2815422
31. MITRE. The Common Weakness Enumeration Official Webpage
32. Perkins, J.H., Kim, S., Larsen, S., Amarasinghe, S., Bachrach, J., Carbin, M., Pacheco, C., Sherwood, F., Sidiroglou, S., Sullivan, G., Wong, W.-F., Zibin, Y., Ernst, M.D., Rinard, M.: Automatically patching errors in deployed software. In 22nd Symposium on Operating Systems Principles (SOSP). ACM, 87–102 (2009). https://doi.org/10.1145/1629575.1629585
33. Pilato, C., Wu, K., Garg, S., Karri, R., Regazzoni, F.: Tainthls: High-level synthesis for dynamic information flow tracking. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **38**(5), 798–808 (2019)
34. Rawat, M., Muduli, S.K., Subramanyan, P.: Mining hyperproperties from behavioral traces. In 2020 IFIP/IEEE 28th International Conference on Very Large Scale Integration (VLSI-SOC), 88–93 (2020)
35. Reger, G., Barringer, H., Rydeheard, D.: A pattern-based approach to parametric specification mining. In 28th International Conference on Automated Software Engineering (ASE). IEEE/ACM, 658–663 (2013)
36. Restuccia, F., Meza, A., Kastner, R.: KER: A design and verification framework for safe and secure soc access control. CoRR arxiv:2106.13263 (2021)
37. Tan, L., Zhang, X., Ma, X., Xiong, W., Zhou, Y.: AutoISES: Automatically inferring security specifications and detecting violations. In 17th USENIX Security Symposium. USENIX Association, 379–394 (2008). http://dl.acm.org/citation.cfm?id=1496711.1496737
38. Tiwari, M., Wassel, H.M., Mazloom, B., Mysore, S., Chong, F.T., and Sherwood, T.: Complete information flow tracking from the gates up. In Proceedings of the 14th international conference on Architectural support for programming languages and operating systems, 109–120 (2009)
39. Weimer, W., Necula, G.C.: Mining temporal specifications for error detection. In 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). Springer-Verlag, 461–476 (2005). https://doi.org/10.1007/978-3-540-31980-1_30
40. Yamaguchi, F., Lindner, F., Rieck, K.: Vulnerability extrapolation: Assisted discovery of vulnerabilities using machine learning. In 5th USENIX Conference on Offensive Technologies (WOOT). USENIX Association, 13 (2011). http://dl.acm.org/citation.cfm?id=2028052.2028065
41. Yang, J., Evans, D., Bhardwaj, D., Bhat, T., Das, M.: Perracotta: mining temporal API rules from imperfect traces. In 28th International Conference on Software Engineering (ICSE). ACM, 282–291 (2006). https://doi.org/10.1145/1134285.1134325
42. Zhang, D., Wang, Y., Suh, G.E., Myers, A.C.: A hardware design language for timing-sensitive information-flow security. Acm Sigplan Notices **50**(4), 503–516 (2015)
43. Zhang, R., Stanley, N., Griggs, C., Chi, A., Sturton, C.: Identifying security critical properties for the dynamic verification of a processor. In Architectural Support for Prog. Lang. and Operating Sys. (ASPLOS). ACM (2017)