Processor Vulnerability Detection with the Aid of Assertions: RISC-V Case Study

Mohammad Reza Heidari Iman[1], Sallar Ahmadi-Pour[2], Rolf Drechsler[2,3], and Tara Ghasempouri[1]
[1]Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
[2]Institute of Computer Science, University of Bremen, Bremen, Germany
[3]Cyber-Physical Systems, DFKI GmbH, Bremen, Germany
{mohammadreza.heidari, tara.ghasempouri}@taltech.ee
{sallar, drechsler}@uni-bremen.de

***Abstract*—As RISC-V processors become more widely distributed, security issues arise. To this end, security verification techniques for processors should be incorporated into the design process to ensure the processor specifications, security requirements, and its actual implementation are consistent. Among the verification techniques, assertion-based verification has emerged as one of the most promising techniques. Although assertions are widely used for functional verification, there is limited effort in applying assertions for security verification. Thus, in this work, a novel security assertion-based technique is introduced for verifying the invulnerability of the processors against Trojan attacks. The experiments show that the proposed method can automatically generate security assertions in a very short amount of time and detect all the inserted Hardware Trojans in the processor, thereby accurately verifying the security of the processor.**

***Keywords*—Automatic Security Verification, RISC-V Security Verification, Security Assertion Mining, RISC-V Processors, Data Mining**

I. Introduction

Processors are ubiquitous in our daily lives and are embedded in nearly every electronic device [1]. In the world of processor design, RISC-V is regarded as one of the most promising technologies. RISC-V's Instruction Set Architecture (ISA) is increasingly adopted in open-source and commercial processor designs due to its flexibility in supporting extensions and customization, as well as its open instruction set [2]. While the openness of the RISC-V ISA is advantageous for fostering a large community to examine and extend the ISA continually, it also poses a disadvantage, as attackers can more easily target the architecture [3].

In this context, an innovative method is required to verify the alignment between the specifications and security requirements of RISC-V, the corresponding security-related ISA, and the practical implementation of the RISC-V extension.

This work was supported by the Estonian Research Council grants PSG837 and the German Federal Ministry of Education and Research (BMBF) within the project ECXL under contract no. 01IW22002, the project SASPIT under contract no. 16KIS1852K and VE-HEP under grant no. 16KIS1342.

979-8-3315-1766-3/24/$31.00 ©2024 IEEE

Numerous mature works on functional verification in the design process exist in the literature [4–12], and some of them can be applied to security verification [6, 13, 14]. The fundamental distinction between functional and security verification lies in their objectives: the former addresses functional problems in alignment with functional specifications, while the latter identifies security issues by considering security requirements and threat models [15].

Among functional verification techniques, semi-formal methods like assertion-based verification (ABV) [16] have garnered interest due to their scalability. In the realm of ABV, assertions represent logical formulas that describe the design's behavior. They enhance both the design's observability and error localization capabilities [17, 18].

There is limited literature on utilizing assertions for the security verification of processors [19–25]. In [20], security assertions for processor vulnerabilities are manually crafted to validate security requirements against the processor design. However, this manual approach demands significant time and expertise, incurring high costs. The method presented in [21] translates security assertion sets from one design to another, but it relies on manual assertion definition and is only evaluated for the OR1200 processor. The other study presented in [23, 24] proposes a method for manually identifying security-critical properties for use in the security verification of the OR1200 processor. Furthermore, [25] introduces a method named SPECS, which serves as a lightweight mechanism to protect software from security-critical bugs in the OR1200 processor. The work in [22] introduces a tool named Isadora for generating security assertions based on information flow tracking techniques, which is more suitable for only tracking the information flows of the processors and hardware designs. The work in [19] formally verifies information flow security for ARM processor kernel and user modes. Nevertheless, its formal nature introduces scalability issues, making it less applicable to large designs. Moreover, the work presented in [6] introduces an assertion miner called HARM designed explicitly for the functional verification of hardware designs. However, it purports the ability to identify Hardware Trojans (HTs) within these designs.

All in all, the majority of current approaches neither explicitly focus on the security verification of RISCV processors nor include an automated method to generate security assertions to check the consistency between processor's security requirements and the actual implementation [26].

Thus, to address this gap, the proposed method first collects the most important security properties of RISC-V processors by studying RISC-V specifications [27, 28] and processor security requirements in the literature [20, 21, 23–25] (Section IV-A); second, proposes a systematic approach to extract ISA characteristics of each security property (Section IV-B); third, introduces an automatic assertion miner which takes these ISA characteristics as well as simulation trace of the processor as inputs to automatically extract a set of security assertions (Section IV-C).

To the best of the author's knowledge, this is the first automated security-based assertion miner that can extract security assertions with the guidelines of ISA characteristics, directly from the simulation trace of the processor. This extracted security assertion set is employed in the security verification process to uncover vulnerabilities, such as Hardware Trojans, embedded within the design. Therefore, the contributions of this paper are listed as follows:

- A systematic method for translating security properties of the processor to the instruction set architecture. The idea is that the security properties that describe the general behavior of a 32-bit processor (*e.g.*, read correctly from memory) will be translated to the proposer 32-bit instruction set;
- An automatic security-based assertion miner to generate security assertions with a high Trojan detection rate;
- The proposed security-based assertion miner can automatically analyze the ISA characteristics (32-bit instruction set) and the simulation traces of the processor to generate security assertions in a short execution time;
- The proposed method is expandable to any processor family and is not limited to RISC-V.

The paper is organized as follows: The preliminaries are presented in Section II. The related background and concepts are described in Section III and the proposed method is elaborated in Section IV. Section V presents the experimental results and finally, Section VI concludes the paper.

II. Preliminaries

In this section, we briefly explain the definitions used in this paper.

Definition 1: A ***simulation trace*** consists of the values of the variables of hardware designs that have been stored as records of data for different time instants (clock cycles) during the execution of the designs [6].

Definition 2: A ***security property*** in this study is defined as a critical security aspect of the processor that neglecting consideration of it can lead to security vulnerabilities in the processor. These security properties are usually presented in the specification of the processor [27, 28].

Definition 3: A ***security assertion*** is a logical formula that must hold true during the execution of the design [8]. The general structure of a security assertion in Property Specification Language (PSL) is like $always(antecedent \rightarrow consequent)$, which implies that the consequent will hold whenever the antecedent occurs [29]. In this study, security assertions check the consistency between the defined behaviors in the *security properties* (Definition 2) and the actual implementation when it faces a vulnerability and a Trojan attack.

Definition 4: ***Temporal pattern next[$\mathcal{N}$]***: *Next[$\mathcal{N}$]* temporal pattern in PSL is in the form of: $always(antecedent \rightarrow next[\mathcal{N}]\ consequent)$ [29]. This temporal pattern means that when antecedent occurs, after $\mathcal{N}$ time instant (clock cycle), consequent will occur [29]. $\mathcal{N}$ is an integer value and $\mathcal{N} > 0$.

Definition 5: ***Frequent itemsets*** refer to a set of variables in simulation trace that occur with a frequency, indicating significant relations/associations between the variables.

Definition 6: An ***Association Rule (AR)*** is defined as an implication of the form $\mathcal{X} \rightarrow \mathcal{Y}$ where $\mathcal{X}, \mathcal{Y} \subseteq \mathcal{I}$, with $\mathcal{X} \cap \mathcal{Y} = \emptyset$, and $\mathcal{I}$ is a set of items [30–32]. $\mathcal{X}$ and $\mathcal{Y}$ are called *frequent itemsets*.

Definition 7: ***Support*** is a metric in association rule mining that indicates how frequently an itemset appears in the dataset [32, 33]. This value is between 0 and 1. For the rule $\mathcal{X} \rightarrow \mathcal{Y}$, the value of support is calculated with the following formula [32]:

$$Supp(\mathcal{X} \rightarrow \mathcal{Y}) = P(\mathcal{X} \cup \mathcal{Y}) \tag{1}$$

In (1), $P(\mathcal{X} \cup \mathcal{Y})$ is the probability where $\mathcal{X} \cup \mathcal{Y}$ indicates that a record contains both $\mathcal{X}$ and $\mathcal{Y}$, that is the union of itemsets $\mathcal{X}$ and $\mathcal{Y}$.

Definition 8: The **min_supp** value is the threshold and a minimum value for *support* to decide whether an itemset is frequent (*i.e.*, occurs frequently in the simulation trace) or not [32]. If the frequency of the itemset is more than this threshold, the itemset is considered a frequent itemset [32]. A higher value of min_supp leads to generating commonly occurring (general) ARs, while a lower value of min_supp leads to generating rarely occurring ARs (corner cases) [32, 34].

Definition 9: ***Confidence*** is an indication of how often the rule has been found to be true [35]. For the rule $\mathcal{X} \rightarrow \mathcal{Y}$, this value is calculated with the following formula [32, 35]:

$$Conf(\mathcal{X} \rightarrow \mathcal{Y}) = P(\mathcal{Y}|\mathcal{X}) \tag{2}$$

It evaluates the degree of certainty of the detected association rule. This is taken to be the conditional probability $P(\mathcal{Y}|\mathcal{X})$, that is the probability that a record containing $\mathcal{X}$ also contains $\mathcal{Y}$. This value is between 0 and 1.

Definition 10: The **min_conf** is the minimum value for *confidence* [32]. The higher value of min_conf leads to fewer but more accurate and valid association rules [32, 34].

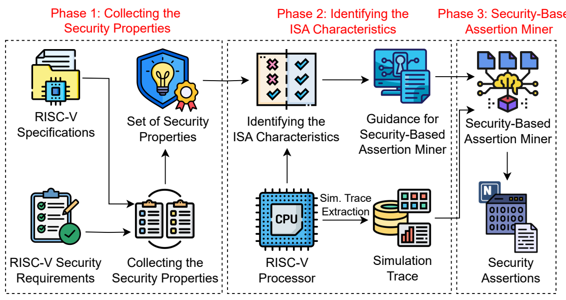

Phase 1: Collecting the Security Properties
Phase 2: Identifying the ISA Characteristics
Phase 3: Security-Based Assertion Miner
RISC-V Specifications
Set of Security Properties
Identifying the ISA Characteristics
Guidance for Security-Based Assertion Miner
Security-Based Assertion Miner
RISC-V Security Requirements
Collecting the Security Properties
RISC-V Processor
Sim. Trace Extraction
Simulation Trace
Security Assertions

Figure 1: Overview of the proposed method

III. Background

In this section, we briefly explain the related concepts and background used in this paper.

A. RISC-V Instruction Set Architecture

RISC-V provides a flexible instruction set for various general and application-specific scenarios. In this work, we utilize the RV32I base instruction set, but the proposed method is independent of the specific ISA or utilized instruction set extensions. The RV32I base instruction set defines a 32 bit architecture around 32 general-purpose registers `x0` to `x32` (with `x0` being constant 0). More information on the RISC-V instruction set and its various extensions can be found in Volume 1 [27], while further details on the privileged architecture, especially CSRS, can be found in Volume 2 [28] of the RISC-V Specification.

B. Threat model: MicroRV32 Platform

Amongst the many available open-source implementations, we chose the MicroRV32 platform [36], implemented in the open-source Hardware Description Language SpinalHDL. Through the modern SpinalHDL language, it is possible to prototype modifications in the data path quickly, while keeping control over the generated Verilog or VHDL description. The platform is synthesizable for FPGAs and ASICs, while providing a lightweight and robust microarchitecture. MicroRV32 features a configurable multi-cycle processor compliant to RV32IMC, meaning it is capable of the aforementioned RISC-V 32-bit base instruction set (I), the multiply/divide extension (M) and the compressed instruction extension (C). Within the platform, a set of peripherals enables the interaction with the outside environment, similar to other microcontroller units.

In this work, our threat model environment consists of a processor based on MicroRV32 embedded in a SoC, with various peripherals and a memory hierarchy.

C. Attack Model

To evaluate the proposed method, we have implemented Hardware Trojans (HTs) based on the following details. The attacker targets the RISC-V RTL code, aiming to disrupt normal operations of IPs and cause damages to the IP design house, *e.g.*, financial losses for any reason. Specifically, the attacker intends to add three Hardware Trojans to the processor: two of them alter the control unit's functionality, while one focuses on the memory and illegal access to it. The attacker possesses knowledge of the design modules and implementation. HTs typically consist of a trigger and a payload [37]. The trigger is the condition activating the Trojan, while the payload executes the malicious function [37]. Triggers can be of different types like Always-On, Conditional, or Time-Based, with payloads causing diverse corruptions like Data and Control Flow Manipulation, DoS, etc [37]. Our HTs use conditional triggers, activating under specific rare conditions, and their payload manipulates program data and control flow. Trojan 1 triggers a specific input combination in the control flow. Its payload alters the execution flow and causes incorrect computation in specific part of control unit. Trojan 2 activates through a specific sequence of control signals, initiating illegal memory access with a payload involving unauthorized memory access. Trojan 3 is triggered by an improbable combination of input conditions, leading to the alteration of the opcode signal and manipulation of update registers. Its payload executes incorrect instructions, disrupting the normal program flow, potentially compromising system integrity, and enabling the attacker to control the instruction sequence. In this work, the HTs have been implemented so that they will be activated in very rare conditions, making their detection difficult.

IV. Methodology

Fig. 1 provides an overview of our proposed method, which is structured in three key phases. These phases are 1- Collecting the Security Properties, 2- Identifying the ISA Characteristics, and 3- Security-Based Assertion Miner.

In the first phase, leveraging RISC-V specifications and already introduced security properties from literature, a set of RISC-V security properties (Definition 2) is collected. In the second phase, the security properties of the processor are translated to the instruction set architecture. In the third phase, operating with the simulation trace generated from the RISC-V design and the guidance report, the security-based assertion miner automatically mines a set of security assertions (Definition 3). Further details for each phase are elaborated in the subsequent subsections.

A. Collecting the Security Properties

After reviewing the RISC-V specifications [27, 28] and its security requirements from literature [20, 21, 23–25], a set of security properties (Definition 2) have been collected for the purpose of this work. These collected security properties are outlined in Table I. However, it is noteworthy that the proposed method can seamlessly extend to encompass other security properties. This versatility enables the method to cater not only to the collected set in our case study benchmark (RISC-V) but also to the diverse security properties of other processors.

In this study, we have collected security properties from the most important categories pertinent to processor security, namely Memory Access, Control Flow, and Updating

Table I: Collected Security Properties

Security Properties	**Security Property Type**
1: Calculation of memory address and memory data is correct.	Memory Access
2: Jumps update the program counter correctly.	Control Flow
3: Jumps update the link register correctly.	Update Register
4: Addition with register value and immediate value results in correct result in the correct target register.	Update Register
5: Load immediate value into the upper 20 bits in the correct target register.	Update Register
6: Adds the immediate as upper 20 bits to the program counter and puts it into the correct target register.	Update Register

Table II: Description of the Security Properties

Security Property description
1. The first security property describes the correctness and relation between the current load or store instruction and the resulting memory address and data on the memory interface. For example, if an HT corrupts the address maliciously to redirect the data, the mismatch between the intended address and the actual address should become visible.
2. The second security property ensures, that the jump instructions redirect the control flow correctly, *i.e.*, the change in program counter reflects the provided register and immediate values accordingly. In this case, an HT could redirect the control flow to malicious code before returning to the original application's code.
3. As jump instructions are used to call functions and return to the code calling a function, the storage of this return address can similarly be tampered with. Thus, the third security property ensures, that the return address saved on jump instructions is correct.
4. As control flow and memory access instructions prepare addresses and values through arithmetic and logic operations, their correctness is vital for security. To avoid information leakage through HT, the fourth security property assures that the `ADDI` (add immediate) instruction saves the correct result only into the correct target register. As an example, an HT could enable writing to the target register and a temporary register, in order to leak the information in a different subroutine.
5. The fifth security property ensures the correctness for the `LUI` (load upper immediate) instruction, that the accordingly extended immediate value is stored into the correct target register.
6. The sixth security property addresses the `AUIPC` (add upper immediate and program counter) instruction. The security property ensures the correct arithmetic and storage in the correct target register. As this instruction is utilized to prepare target addresses, an HT can potentially redirect the control flow to a different part of the code.

Registers. In papers [20, 21, 23–25] which represent the latest studies on processor security properties, these categories have been reported as critical areas requiring scrutiny in the security verification process of processors as depicted in Table I. Consequently, they have been employed in this paper. Further details and descriptions of these security properties are provided in Table II.

These security properties serve as the inputs for the next phase, *i.e.*, Identifying the ISA Characteristics.

B. Identifying the ISA Characteristics

In order to identify which parts of the ISA specification contribute to a security property, we will utilize an example that covers security properties 2 and 3. Consider the RISC-V instruction `JAL rd, offset`, *Jump And Link*. This instruction manipulates the control flow of the application, jumping to a new location in the program by setting the *Program Counter* (PC) to an `offset` encoded as an immediate value (*i.e.*, `PC = offset`). As a second part, the instruction will store the value of program counter of the instruction after the `JAL` instruction (`rd = PC + 4`). We can consider possible violation of security if, for example, a HW Trojan introduces are change of the offset under specific system conditions. This could lead to a possible attack in which an unwanted execution of different code occurs. Therefore, for the `JAL` instruction, we can denote two possible high-level properties:

(a) Jumps update the PC correctly according to the `offset` passed in the instruction immediate.
(b) Jumps update the register `rd` with the correct PC (*i.e.*, `rd = PC+4`).

Consequently, the security-based assertion miner would need to find potential security assertions covering the signals mentioned in the high-level properties. In a more general sense, these steps can be abstracted as follows:

1) Identifying instructions affected by threat model (*e.g.*, control flow integrity affects the instructions for branch, jump, and address manipulation).
2) Identifying each part of functional behavior contributing to instructions (*e.g.*, jump instruction changes PC and a register based on PC value).
3) Formulating high-level property reflecting parts of the functional behavior (*e.g.*, resulting addresses for store or load operations have to be consistent with the initial instruction and register values).
4) Utilizing the high-level property to identify the affected signals in the processor and its interfaces. This includes the relation between instruction bits and the observable behavior on results and interfaces.

In this phase, alongside the guidance report created based on the identified ISA characteristics, a simulation trace is generated from the RISC-V design. These inputs are utilized by the security-based assertion miner in the next phase.

C. Security-Based Assertion Miner

In this phase, we elaborate on the details of the proposed security-based assertion miner. As illustrated in Fig. 2, the proposed miner is comprised of four primary steps: 1) Signal Identification and Pre-processing of Simulation Trace, 2) Association Rule Mining, 3) Time Notation, and 4) Assertion Conversion.

During the first step, the exhaustive simulation trace of the RISC-V undergoes pre-processing to prepare the data. Subsequently, in the Association Rule Mining step, we apply our algorithm to the pre-processed simulation trace to mine all association rules (Definition 6) derived from the simulation trace. Afterward, in the third step, *i.e.,* Time Notation, the extracted association rules are integrated with the concept of time to generate appropriate time-integrated rules (temporal association rules) in the form of $next[\mathcal{N}]$ pattern (Definition 4). Consequently, the Assertion Conversion step transforms the rules from the previous step into assertions.

Algorithm 1 presents the detailed process of the first three steps of the security-based assertion miner. In this algorithm, $\mathcal{ST}$ denotes the simulation trace and $\mathcal{ST}'$ is the pre-processed simulation trace, while f and t represent the simulation trace's outputs and input values.

In the following subsections, we discuss each step of the security-based assertion miner in more detail.

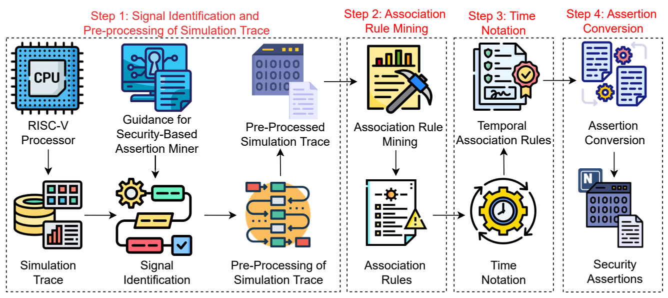

Step 1: Signal Identification and Pre-processing of Simulation Trace
Step 2: Association Rule Mining
Step 3: Time Notation
Step 4: Assertion Conversion
CPU
RISC-V Processor
Guidance for Security-Based Assertion Miner
Pre-Processed Simulation Trace
Association Rule Mining
Temporal Association Rules
Assertion Conversion
Simulation Trace
Signal Identification
Pre-Processing of Simulation Trace
Association Rules
Time Notation
N
Security Assertions

Figure 2: Proposed Security-Based Assertion Miner

Algorithm 1: Security-Based Assertion Miner

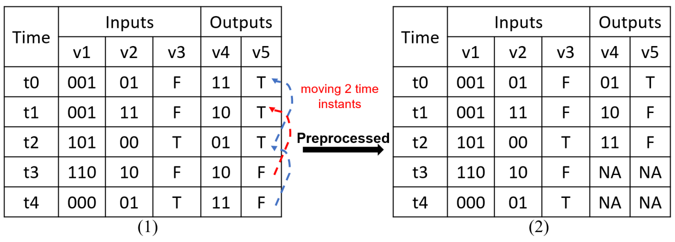
```
1  Input: $\mathcal{N}$, $\mathcal{ST}$, $min\_supp$
2  Output: $next[\mathcal{N}] = antecedent \rightarrow next[\mathcal{N}]consequent$
   /* Initialization                                   */
3  $\mathcal{R} = antecedent \rightarrow consequent$
4  $L_1 = \{frequent\ 1-itemsets \in \mathcal{ST}'\}$
5  K = 2
   /* Signal Identification & Pre-processing          */
6  $\mathcal{ST}' \leftarrow$ prune_nonrelevant_security_signals($\mathcal{ST}$)
7  forall $f \in \mathcal{ST}'$ do
8      $\mathcal{ST}' =$ MoveUp($f(\mathcal{N})$)
   /* Association Rule Mining                          */
9  while $L_{k-1}\ != \emptyset$ do
10     $C_k =$ generate_candidate_itemsets($L_{k-1}$)
11     $L_k =$ prune_infrequent_itemsets($C_k, min\_supp$)
12     $k = k+1$
13 foreach $frequent\ itemset\ L_i \in L$ do
14     foreach $subset\ S\ of\ L_i$ do
15         if $(S\ != \emptyset)\ \&\&\ (S\ !=\ L_i)$ then
16             $confidence = support(L_i)\ /\ support(S)$
17             if $confidence >= min\_conf$ then
18                 $\mathcal{R} \leftarrow$ association_rule($S => L_i$)
19 return $\mathcal{R}$
   /* Time Notation                                    */
20 if ($\mathcal{R}.antecedent == (t \in \mathcal{ST}')$ and ($\mathcal{R}.consequent == (f \in \mathcal{ST}')$) then
21     $next[\mathcal{N}] \leftarrow$ label($\mathcal{R}$)
22 else
23     Discard($\mathcal{R}$)
```

1) *Signal Identification and Pre-processing of Simulation Trace*

Lines 6 to 8 of the Algorithm 1 are related to the Signal Identification and Pre-processing step of the security-based assertion miner. In this step, at first, the RISC-V simulation trace and guidance report generated from the second phase of the method (Section IV-B) are processed for signal identification. In signal identification, the assertion miner prunes all the signals of the simulation trace that are not relevant to the identified security properties. Once signals associated with the specified security properties are identified, the simulation trace undergoes pre-processing.

To pre-process the simulation trace, all the identified output of the simulation trace is moved $\mathcal{N}$ records above its original position (line 8 of the Algorithm 1). However, the identified inputs of the simulation trace remain as they are. Traditional association rule mining algorithms (*e.g.*, Apriori [32]) cannot typically mine the rules in the form of *next[$\mathcal{N}$]*. Because of this reason, and also since the corresponding output of input variables in a sequential hardware design may occur in the simulation trace $\mathcal{N}$ time instants later, this pre-processing needs to be performed. This ensures the correct alignment of outputs with their corresponding inputs, allowing for accurate temporal analysis and also mining patterns for different $\mathcal{N}$ time instants (clock cycles).

Fig. 3 illustrates an example of pre-processing for *next[2]* clock cycles. The simulation trace in Fig. 3.1 is pre-processed by moving the output parts 2 time instants above their original positions, resulting in the modified simulation trace shown in Fig. 3.2. The figure uses T to represent the true value, and F to show the false value. The last two records in Fig. 3.2 are marked as not available (NA) due to the absence of data after time instant t4 to be moved in front of these two records.


moving 2 time instants
Preprocessed
(1)
(2)

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	11	T
t1	001	11	F	10	T
t2	101	00	T	01	T
t3	110	10	F	10	F
t4	000	01	T	11	F

Time	Inputs			Outputs	
	v1	v2	v3	v4	v5
t0	001	01	F	01	T
t1	001	11	F	10	F
t2	101	00	T	11	F
t3	110	10	F	NA	NA
t4	000	01	T	NA	NA

Figure 3: (1) Simulation trace (2) Pre-processed simulation trace

2) *Association Rule Mining*

The resulting pre-processed simulation trace is subsequently fed into lines 9 to 19 of Algorithm 1 to mine association rules. Applying these lines of the algorithm to the pre-processed simulation trace provides us with a set of association rules in the form of $antecedent \rightarrow consequent$.

In lines 9 to 12 of Algorithm 1, frequent itemsets (Definition 5) of various sizes (1-itemsets, 2-itemsets, etc.) are generated iteratively until the list of the frequent itemsets is empty. Specifically, the algorithm mines frequent itemsets whose support values (Definition 7) exceed the min_supp value (Definition 8), while pruning the others. In line 10 of the algorithm, C_k is the candidate itemsets of size k that are generated by combining frequent (k-1)-itemsets and L_k in line 11 of the algorithm is the set of frequent k-itemsets. In this algorithm, 1-itemsets consist of individual variables of simulation trace, 2-itemsets are pairs of variables, etc.

After mining the frequent itemsets and adding them to the L_k list, in lines 13 to 19 of the algorithm, the association rules are extracted from the list of frequent itemsets. Due to space limits, we refer interested readers to [32] and [38] for more information on the details of mining frequent itemsets and association rules.

In Algorithm 1, increasing the min_supp value results in fewer assertions that describe more general design behavior, while decreasing the min_supp value leads to assertions covering rare design behavior (corner cases). These corner cases are important as attackers can consider them for performing any corruption in the design. Similarly, raising the min_conf value produces fewer but more valid assertions. Valid assertions refer to assertions that will not be violated during the simulation with different attack scenarios. The utilization of these values in the security-based assertion miner facilitates an effective vulnerability detection process.

Table III: Comparison of the proposed method with HARM

Assertion Miner	Total number of Security Assertions	#Security Assertions Detecting Trojans	Ratio of Security Assertions Detecting Trojans to the Total Number of Assertions (%)	Execution Time
Proposed Method	4036	256	6.3	5min30sec
HARM [6]	16073	397	2.4	74min31sec

At this point, with the completion of the association rule mining, these rules serve as the fundamental components of the Time Notation step.

3) Time Notation

In the previous step, the method provides us a set of rules in the general form of $antecedent \rightarrow consequent$. In this step, the method integrates the concept of time into the association rules generated in the association rule mining step, leading to a set of temporal association rules in the form of $antecedent \rightarrow next[\mathcal{N}]consequent$. Lines 20 to 23 in Algorithm 1 describe the details of the Time Notation step. If the antecedent value matches an input in the simulation trace, and the consequent value has already been moved to another record in the simulation trace, the rule is labeled as a *next* temporal association rule. Otherwise, other mined rules are discarded.

4) Assertion Conversion

In this step, the mined temporal association rules are transformed into temporal security assertions (Definition 3) using the labels assigned in the Time Notation step. Thereby, the output of the Time Notation step for temporal association rules labeled as *next[$\mathcal{N}$]* is transformed into the PSL format "*always(antecedent $\rightarrow$ next[$\mathcal{N}$]consequent)*".

V. Experimental Results

For our experimental evaluation, we utilized the security-based assertion miner to generate security assertions in the form of SystemVerilog. We utilized the processor embedded in the open-source MicroRV32 platform [36]. For generating the simulation trace, we executed a software application on the processor that is centered around checksum calculations for embedded systems. It features different types of control flow, loops, arithmetic operations, and interaction with the memory as well as the available peripherals. Hence, we generated a simulation trace with 10000 records that activates various parts of the microarchitecture to provide a diverse data set for security assertion mining. The generated assertions are then evaluated by including Hardware Trojans in the processor's microarchitecture together with the generated security properties, to verify them. Notably, the values for the minimum support (Definition 8) and minimum confidence (Definition 10) have been set to 0.01 and 1, respectively. Moreover, $\mathcal{N}$ has been set to 2 for the $next[\mathcal{N}]$ pattern, but it can be adjusted to other values.

Table IV exhibits the detailed experimental results about the assertions that are associated with any of the security properties that we presented in Section IV-A. The proposed security-based assertion miner generated a total of 4036 security assertions. It should be noted that while the total number of generated assertions is 4036, some of them overlap so that they contain the signals of the design that are related to several security properties. According to the experimental results in Table IV, 870 and 1490 security assertions are related to security properties 1 and 2, which means that these numbers of assertions can cover the behaviors that have been described for these two security properties. The results indicate that for security properties 3 and 4, 1522 and 2026 security assertions have been mined, respectively. This figure for both the security properties 5 and 6 is 1898 assertions.

Table IV: Detailed Experimental Results on Six Different Security Properties

#Security Assertions	#Generated Security Assertions for each Security Property					
	Security Property1	Security Property2	Security Property3	Security Property4	Security Property5	Security Property6
4036	870	1490	1522	2026	1898	1898

Table V: Experimental Results on Detected Trojans

Trojans	#Security Assertions Detecting Trojans	Trojan Detection
Trojan 1	16	✓
Trojan 2	64	✓
Trojan 3	176	✓

✓: Trojan has been detected.

Table V presents the results of Trojan detection. The column '#Security Assertions Detecting Trojans' presents the number of security assertions that could detect any of the Trojans. For Trojan 1, 16 assertions detected it and 64 and 176 security assertions detected Trojans 2 and 3, respectively.

Table III presents a comparative analysis between our proposed security-based assertion miner and HARM [6]. HARM is explicitly presented as a tool that can be employed in the context of security verification for Trojan detection, making it a relevant tool for our comparison. While HARM generated 16073 assertions, extending the security verification process time, our method produced a more compact and accurate set of 4036 assertions. Among all the mined assertions by HARM, a total of 397 assertions could detect the Trojans, while we accomplished Trojan detection with 256 assertions. The ratio of Trojan detection by security assertions shows our method's superior effectiveness, with 6.3% compared to HARM's 2.4%. Considering the fact that there are only three Trojans, and as mentioned in Subsection III-C, the probability of their activation is very rare, 6.3% shows promise. These findings underscore the efficiency of our method in producing a smaller yet more potent and accurate set of security assertions. Moreover, our proposed miner demonstrated a significantly shorter execution time, completing the assertion mining in about 5 minutes, compared to HARM's duration of over an hour.

VI. Conclusion

In this paper, we introduced a method to generate security assertions for a RISC-V processor to detect Hardware Trojans. Our experiments show that these assertions effectively capture the intended security properties and detect injected Hardware Trojans within the design.

References

[1] M. R. H. Iman and P. Yaghmaie, "A software control flow checking technique in multi-core processors," *Int. J. Embed. Syst.*, vol. 13, pp. 136–147, 2020.

[2] A. Harris, T. Verma, S. Wei, L. Biernacki, A. Kisil, M. T. Aga, V. Bertacco, B. Kasikci, M. Tiwari, and T. Austin, "Morpheus ii: A risc-v security extension for protecting vulnerable software and hardware," in *HOST*, 2021, pp. 226–238.

[3] A. L. D. Antón, J. Müller, M. R. Fadiheh, D. Stoffel, and W. Kunz, "Fault attacks on access control in processors: Threat, formal analysis and microarchitectural mitigation," *IEEE Access*, vol. 11, pp. 52 695–52 711, 2023.

[4] R. Hariharan, T. Ghasempouri, B. Niazmand, and J. Raik, "From rtl liveness assertions to cost-effective hardware checkers," in *2018 Conference on DCIS*, 2018, pp. 1–6.

[5] S. Vasudevan, D. Sheridan, S. Patel, D. Tcheng, B. Tuohy, and D. Johnson, "Goldmine: automatic assertion generation using data mining and static analysis," in *Proc. of the DATE*, 2010, pp. 626–629.

[6] S. Germiniani and G. Pravadelli, "HARM: A hint-based assertion miner," *IEEE TCAD*, vol. 41, no. 11, pp. 4277–4288, 2022.

[7] J. Malburg, T. Flenker, and G. Fey, "Property mining using dynamic dependency graphs," in *ASP-DAC*, Jan 2017, pp. 244–250.

[8] A. Danese, N. D. Riva, and G. Pravadelli, "A-Team: Automatic template-based assertion miner," in *54th DAC conf.*, 2017, pp. 1–6.

[9] M. R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "A methodology for automated mining of compact and accurate assertion sets," in *NorCAS*, 2021, pp. 1–7.

[10] S. Germiniani and G. Pravadelli, "Exploiting clustering and decision-tree algorithms to mine ltl assertions containing non-boolean expressions," in *VLSI-SoC*, 2022.

[11] M. R. Heidari Iman, J. Raik, M. Jenihhin, G. Jervan, and T. Ghasempouri, "An automated method for mining high-quality assertion sets," *Microprocessors and Microsystems*, vol. 97, p. 104773, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141933123000194

[12] M. R. Heidari Iman, J. Raik, G. Jervan, and T. Ghasempouri, "IMMizer: An innovative cost-effective method for minimizing assertion sets," in *2022 25th Euromicro Conference on Digital System Design (DSD)*, 2022, pp. 671–678.

[13] H. Witharana, A. Jayasena, A. Whigham, and P. Mishra, "Automated generation of security assertions for rtl models," *J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 1, jan 2023.

[14] F. Erata, S. Deng, F. Zaghloul, W. Xiong, O. Demir, and J. Szefer, "Survey of approaches and techniques for security verification of computer systems," *J. Emerg. Technol. Comput. Syst.*, vol. 19, no. 1, jan 2023.

[15] T. Ghasempouri, J. Raik, C. Reinbrecht, S. Hamdioui, and M. Taouil, "Survey on architectural attacks: A unified classification and attack model," *ACM Comput. Surv.*, vol. 56, no. 2, sep 2023.

[16] T. Ghasempouri, J. Malburg, A. Danese, G. Pravadelli, G. Fey, and J. Raik, "Engineering of an effective automatic dynamic assertion mining platform," in *2019 IFIP/IEEE 27th VLSI-SoC*, 2019, pp. 111–116.

[17] M. Eslami, T. Ghasempouri, and S. Pagliarini, "Reusing verification assertions as security checkers for hardware trojan detection," in *2022 23rd ISQED Symposium*, 2022, pp. 1–6.

[18] H. Witharana, Y. Lyu, S. Charles, and P. Mishra, "A survey on assertion-based hardware verification," *ACM Comput. Surv.*, vol. 54, no. 11s, pp. 1–33, 2022.

[19] N. Dong, R. Guanciale, M. Dam, and A. Lööw, "Formal verification of correctness and information flow security for an in-order pipelined processor," in *FMCAD*, 2023, pp. 247–256.

[20] C. S. Chuah, C. Appold, and T. Leinmueller, "Formal verification of security properties on risc-v processors," in *21st MEMOCODE*, 2023, pp. 159–168.

[21] R. Zhang and C. Sturton, "Transys: Leveraging common security properties across hardware designs," in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 1713–1727.

[22] C. Deutschbein, A. Meza, F. Restuccia, R. Kastner, and C. Sturton, "Isadora: Automated information flow property generation for hardware designs," in *Proc. of the 5th Workshop on Attacks and Solutions in Hardware Security*, ser. ASHES '21. Association for Computing Machinery, 2021, p. 5–15.

[23] R. Zhang, N. Stanley, C. Griggs, A. Chi, and C. Sturton, "Identifying security critical properties for the dynamic verification of a processor," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, p. 541–554, apr 2017.

[24] B. Kumar, A. K. Jaiswal, V. S. Vineesh, and R. Shinde, "Analyzing hardware security properties of processors through model checking," in *2020 33rd Int. Conf. on VLSID*, 2020, pp. 107–112.

[25] M. Hicks, C. Sturton, S. T. King, and J. M. Smith, "Specs: A lightweight runtime mechanism for protecting software from security-critical processor bugs," in *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 517–529.

[26] F. Farahmandi, M. S. Rahman, S. R. Rajendran, and M. Tehranipoor, *CAD for Hardware/Software Security Verification*. Cham: Springer International Publishing, 2023, pp. 187–210.

[27] A. Waterman, K. Asanovic *et al.*, "The risc-v instruction set manual volume i: Unprivileged isa," *Document Version*, vol. 20191213, 2019.

[28] ——, "The risc-v instruction set manual, volume ii: Privileged architecture," *RISC-V Foundation*, 2019.

[29] "Standard for property specification language (PSL), ieee standard," p. 1–184, 2012.

[30] C. Antunes and A. L. Oliveira, "Temporal data mining: an overview," 2001.

[31] S. Bilqisth and K. Mustofa, "Determination of temporal association rules pattern using apriori algorithm," *IJCCS Journal*, vol. 14, p. 159, 04 2020.

[32] J. Han, M. Kamber, and J. Pei, "6 - mining frequent patterns, associations, and correlations: Basic concepts and methods," ser. The Morgan Kaufmann Series in Data Management Systems, 2012, pp. 243–278.

[33] M. Shahin, M. R. Heidari Iman, M. Kaushik, R. Sharma, T. Ghasempouri, and D. Draheim, "Exploring factors in a crossroad dataset using cluster-based association rule mining," *Procedia Computer Science*, vol. 201, pp. 231–238, 2022, the 13th International Conference on Ambient Systems, Networks and Technologies (ANT) / The 5th International Conference on Emerging Data and Industry 4.0 (EDI40).

[34] A. Roberts, M. R. Heidari Iman, M. Bellone, T. Ghasempouri, J. Raik, O. Maennel, M. Hamad, and S. Steinhorst, "ADAssure: Debugging methodology for autonomous driving control algorithms," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.

[35] M. R. H. Iman, P. Chikul, G. Jervan, H. Bahsi, and T. Ghasempouri, "Anomalous file system activity detection through temporal association rule mining," in *9th ICISSP,*, 2023, pp. 733–740.

[36] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "The microrv32 framework: An accessible and configurable open source risc-v cross-level platform for education and research," *Journal of Systems Architecture*, vol. 133, p. 102757, 2022.

[37] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *ISCAS*, 2015, pp. 2021–2024.

[38] M. R. Heidari Iman, G. Jervan, and T. Ghasempouri, "ARTmine: Automatic association rule mining with temporal behavior for hardware verification," in *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2024, pp. 1–6.