Security Coverage Metrics for Information Flow at the System Level

Ece Nur Demirhan Coşkun* Sallar Ahmadi-Pour[U] Muhammad Hassan*[U] Rolf Drechsler*[U]
*Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[U]Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
ece.coskun@dfki.de {drechsler, hassan, sallar}@uni-bremen.de

***Abstract*—In this paper, we introduce a novel set of security coverage metrics for information flow at the system level. The proposed security coverage metrics play a crucial role in assessing the qualification and quantification of various security properties, in addressing specific threat models, such as availability, and in identifying potential security vulnerabilities associated with information flow. To implement these metrics, we present SiMiT, a tool that leverages *Virtual Prototypes* (VP), and Static and Dynamic *Information Flow Tracking* (IFT) methodologies. We demonstrate the applicability of the proposed security coverage metrics through SiMiT on an open-source RISC-V VP architecture with its peripherals. By assessing the security properties using these metrics, we pave the way for a security-aware *Completeness Driven Development* (CDD) concept and the development of secure *System-on-Chip* (SoC) designs.**

I. Introduction

Modern *System-on-Chip* (SoC) designs are ubiquitous in *Internet of Things* (IoT) devices, combining software (SW), and digital hardware (HW) with microcontrollers and microprocessors, and various *Intellectual Properties* (IP). These diverse components seamlessly integrate to provide feature-rich functionality in IoT devices for mission-critical applications. Many of these critical applications, such as modern car *Electronic Control Unit* (ECU) systems, require real-time responses.

One bug in any of these critical systems could be catastrophic through incidents, such as delayed rescue operations or accidents [1]. Moreover, the presence of real-time requirements in HW designs makes them susceptible to Denial of Service (DoS) attacks [1]. These attacks render the IPs unavailable on demand, potentially causing the entire system to malfunction. An example of such vulnerability was found in [2], where an attacker could attempt to shut down the network established by *Roadside Units* (RSUs), disrupting communication between vehicles and/or RSUs. As a result, vehicles could not receive the road status information in time. In [3], JellyFish attack was introduced, where an attacker could disorder, delay, or periodically drop packets it was supposed to forward. These attack scenarios emphasize the importance of security validation techniques in preventing security vulnerabilities.

Information Flow Tracking (IFT) is one of the security validation techniques that has been shown as a powerful technique to help mitigate security vulnerabilities that violate information flow policies and non-interference properties such as confidentiality, integrity, and availability [4], [5]. However, the effectiveness of this technique relies on the accurate security property definition by the SoC designer, ensuring the detection of vulnerabilities aligned with the threat model. Therefore, the security properties should be assessed qualitatively and quantitatively as part of the security validation. This assessment can be done by defining security coverage metrics. The security coverage metrics assist verification engineers in gaining a better intuition to assess weaknesses, understand vulnerabilities, and derive appropriate security properties that enhance the security of SoC designs.

Consequently, it is crucial to proactively integrate security considerations very early in the SoC design phase [5], [6]. This approach helps to decrease in verification time and effort significantly, ensuring compliance with very tight *Time-to-Market (TTM)* budget. Therefore, *Virtual Prototypes* (VPs) are being increasingly adopted by the semiconductor industry [4], [5]. The VPs are the abstract SW models of the HW, implemented in SystemC [7] with its *Transaction Level Modeling* (TLM) [8] extension. They serve as a golden reference for early SW and HW development [5]. The VPs are used at the *Electronic System Level* (ESL) as the starting point for early design and verification in *Completeness-Driven Development* (CDD) concept [9]. The CDD is an approach used in the design phase to ensure completeness, i.e. verifying the complete behavior of the design at each level of abstraction. The CDD concept makes progress to subsequent abstraction levels only after achieving completeness by verifying the entire behavior at each level of abstraction.

The current CDD concept solely focuses on functional correctness. However, integrating the security aspect of the design process is of utmost importance. Building a security-aware CDD concept is crucial in addressing security vulnerabilities. It highlights the importance of questioning the security properties by applying security coverage metrics to mitigate potential threats. Although several security properties for IFT have been defined [4], [5] to validate the VP-based models based on the security requirements, there is still a lack of security coverage metrics to check the security properties at the system level.

Our Contributions: In this paper, we propose a novel set of security coverage metrics for Information Flow at the system level. Thereby, we assess the quality and quantity of various

This work was supported in part by the German Federal Ministry of Education and Research (BMBF) within the project ECXL under contract no. 01IW22002, the project PaSVer under contract no. 16ME0855, the project SASPIT under contract no. 16KIS1852K, and the project Scale4Edge under contract no. 16ME0127.

security properties. Then, we introduce SiMiT, which is a **S**tat**i**c and Dyna**mi**c IF**T** tool, to implement these security coverage metrics at the system-level. By assessing the security properties, it becomes possible to ensure that IFT techniques meet the necessary criteria for obtaining the desired security sign-off, a crucial milestone in the verification process. These novel security coverage metrics are defined based on information flow, such as **direct/indirect signal connectivity**, **partial/full path activation**, **information flow rate**.

The major contributions of this paper are summarized as follows:

- We define a novel set of security coverage metrics for system-level Information Flow to present valuable insights into how well these security coverage metrics can address the target threat models.
- We introduce SiMiT, a tool that leverages Static and Dynamic IFT techniques, to implement security coverage metrics at the system level.
- We assess the availability security properties with proposed security coverage metrics.
- We demonstrate the applicability of these security coverage metrics using an open source RISC-V VP [10].

Paper Structure: Section II discusses the related work. In Section III, we describe the target threat model and the motivating example. Then, in Section IV, we introduce the novel security coverage metrics. Afterwards, in Section V, we explain the IFT methodologies of SiMiT and the implementation of the security coverage metrics in SiMiT. Finally, Section VI presents the experimental results, and Section VII concludes the paper.

II. Related Work

In this section, we provide some related works, and then we discuss how SiMiT is related to and distinguished from others. Although several security coverage metrics have been introduced to evaluate vulnerabilities under various threat models [11], [12], such as Trojans, IP piracy, reverse engineering, side channels, and counterfeiting, these studies do not specifically address the metrics that we proposed in SiMiT.

Some security metrics were presented for *Quantitative Information Flow* (QIF) that serve as a measure of the information leakage magnitude. Researchers have translated QIF to the HW, with a specific emphasis on cryptographic cores and Trojans [13]–[15]. However, extending the application of QIF beyond these specific threat models presents a challenge that requires further research.

Yet, other security metrics on Vulnerability analysis were introduced in [16]. They specifically designed to test structures, placing particular emphasis on cryptographic cores, Trojans, and access control. This analysis involves performing a gate-level path traversal and calculating the distance, measured by the number of gates, from the asset to control or observation points.

In another work [17], the definition of attributes and security coverage metrics for the hyperflow graph was undertaken for several threat models, including confidentiality, integrity, and availability. While they presented useful security coverage metrics by applying noninterference, our approach provides security coverage metrics at the system level instead of RTL or Gate level that enables early bug detection. Additionally, we extend the assessments to include RISC-V VPs. To the best of our knowledge, no existing work has investigated security coverage metrics for VP-based IFT against availability security properties.

III. Background and Motivation

In this section, we describe the target threat model, specifically focusing on availability. After that, we discuss a motivating example to illustrate how security coverage metrics are defined.

A. Threat Model

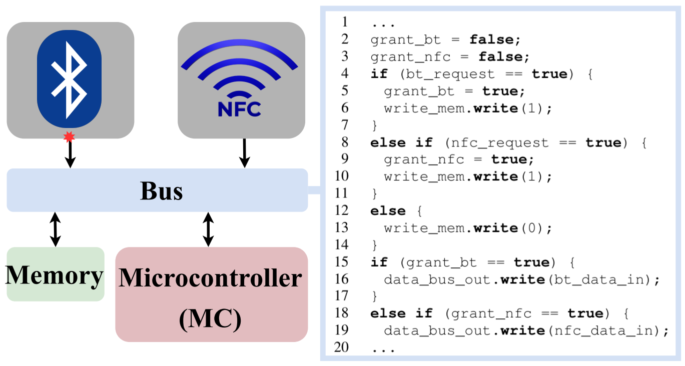
In SiMiT, we consider a threat model based on the *Confidentiality, Integrity, Availability, Authentication* (CIAA) principles [5], in particular **availability**. Availability problems arise when an IP uses some shared resources to the point that they are unavailable to other IPs [5].

B. Motivating Example

We provide here a motivating example that is used to illustrate the representation of the security coverage metrics throughout this paper. The motivating example is a simplified *Keyless Entry System* (KES) that utilizes *Near Field Communication* (NFC) and *Bluetooth* (BT) IPs for smart lock/unlock in KES via application-based authentication [18].

The system consists of a *Bus* that facilitates communication between an NFC module, a BT module, a *Microcontroller* (MC), and a *Memory* as shown in Fig. 1. The NFC enables short-range wireless communication for authentication to give an access to the authorized entities between the KES and NFC-enabled smartphones. The BT is also used for authentication to give an access to the authorized entities. The MC acts as the control unit and, responsible for processing data, managing access control policies, and executing keyless entry functionalities. It coordinates the integration of the NFC, BT, and *Memory* modules. The *Memory* stores authorized credentials, and critical data required for the operation of the KES. It holds configuration settings, access logs for the communication and authentication.

Now consider a scenario where the access control policy in the *Bus* is misconfigured, giving higher priority to BT over NFC



```
...
grant_bt = false;
grant_nfc = false;
if (bt_request == true) {
  grant_bt = true;
  write_mem.write(1);
}
else if (nfc_request == true) {
  grant_nfc = true;
  write_mem.write(1);
}
else {
  write_mem.write(0);
}
if (grant_bt == true) {
  data_bus_out.write(bt_data_in);
}
else if (grant_nfc == true) {
  data_bus_out.write(nfc_data_in);
...
```

Fig. 1: The SystemC design of a simplified *Keyless Entry System*, and code excerpt from the Bus IP implementing priority encoded access policy.

to the shared *Memory*. Thereby, the system utilizes the BT for authentication, while NFC is given lower priority seen in Fig. 1. The first priority is determined in *Line 4* of the code excerpt of the Bus IP: If the BT request signal of BT is true, the signal *grant_bt* is set and data is transferred from BT to *Memory*. The second priority is shown in *Line 8*. When *nfc_request* signal for the NFC is true, *grant_nfc* is set to allow access to *Memory*.

Most of the smart devices' peripherals (e.g. BT, NFC) are restricted to establish only one connection at a time. This can be interpreted as a security vulnerability, leading to eavesdropping, interception, or relay attacks from a longer range [18].

Due to a misconfiguration of priorities, attackers can potentially exploit these vulnerabilities to block the availability of NFC module on time, leading to the possibility of skipping requests from the NFC. However, these modules should have an equal opportunity to have access to the shared *Memory*. Such indirect information flow across another IP is difficult to detect, particularly without an automated analysis using IFT. Therefore, security coverage metrics are essential here to provide more information about Information Flows across a system to ensure that the security properties are correctly defined.

IV. Security Coverage Metrics

In this section, we introduce a novel set of security coverage metrics for Information Flow at the system level. Essentially, the proposed security coverage metrics to assess various security properties while addressing the threat model, availability, mentioned in Section III-A. In the following subsections, we explain how each metric provides details on what information/insights it can offer.

A. Direct Signal Connectivity

The *Direct Signal Connectivity* (DSC) metric determines whether a signal A and a signal B are directly, i.e. explicitly connected. The *Explicit Information Flow* (EIF) results from two modules that are directly communicating. For example, an EIF could occur, if the BT module and the NFC module were directly exchanging data.

B. Indirect Signal Connectivity

The *Indirect Signal Connectivity* (ISC) metric determines whether a signal A and a signal B are indirectly connected. These connections can be identified through *Implicit Information Flow* (IIF), which is more subtle compared to the EIF, and could result in the unavailability of the IPs through certain behaviors. The IIF might occur between two modules, where one belongs to the trusted zone and the other to the untrusted zone, sharing memory. An example of such an IIF can be observed between the BT module in an untrusted zone and the NFC module in a trusted zone, as both are allowed to access the memory via the Bus.

C. Partial Path Activation

Establishing a definite flow between signals goes beyond mere connectivity; it requires path activation as a crucial confirmation. Introducing the *Partial Path Activation* (PPA) metric helps illustrate this concept. The PPA metric serves as a quantitative measure to assess the extent to which paths between a signal A and a signal B are activated in a given simulation for a given time interval $[t_1, t_2]$. Activation of a path occurs when information originating from signal A successfully propagates to signal B. By employing this metric, we can effectively demonstrate that although a path exists between signal A and signal B in the system, an exact flow cannot be guaranteed.

D. Full Path Activation

The *Full Path Activation* (FPA) quantifies the total path activation for the whole execution time. For example, if there are 5 different paths in total from signal A to B, we look into how many of them are activated throughout the entire execution time.

E. Information Flow Rate

The *Information Flow Rate* (IFR) metric quantifies the occurrence of an information flow from a signal A to a signal B within a given time frame. In other words, by quantifying the IFR metric, we can have an insight into how many times the signal A reaches the signal B during the time interval.

V. SiMiT - Static and Dynamic IFT Tool

In this section, we first provide a high-level overview of SiMiT, depicted in Fig. 2. Then, we show how the security coverage metrics are implemented in SiMiT. The SiMiT is a tool that leverages Static and Dynamic IFT techniques that enables a comprehensive understanding of information flow across a system.

A. The Static and Dynamic IFT techniques of SiMiT

The various IFT techniques used in SiMiT are explained as follows:

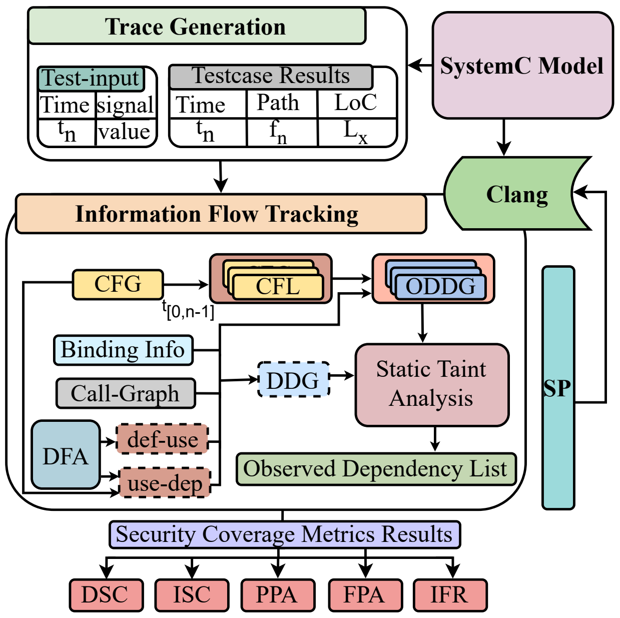


Fig. 2: The workflow of SiMiT (SP: Security Properties, LoC: Line of Codes, L_x: Line number, f_n: File name, CFL: Control Flow List, t_n: Simulation time)

*a) **Trace Generation**:* We save the information flow of the SystemC model at a given time t_n, along with the current file name f_n and the line number L_x within the module where it is located. In trace generation, we get comprehensive testsuits, but the quality of the testcases is out of the scope in this paper. Through the examination of the testcase results, we can infer the interconnection of signals during each instance.

*b) **Security property definition** (**SP**):* We target the availability security properties. It is defined such that various IPs are required to be available in a timely manner [19]. Thus, we define SPs as follows:

$$SP = \big\{(SI, SO) | SI \in \{\text{in}_1 = HS, ..\}, SO \in \{\text{out}_1 = AA, ..\}\big\} \quad (1)$$

The SP in Eq. (1) ensures that each SP has inputs with the *High Security* (HS) tag and outputs that must be *Always Available* (AA) when needed.

*c) **Binding Information** (**BI**) & **Call Graph** (**CG**):* The BI is necessary for the modules' connectivity. This information is crucial for understanding how data moves through the system and for constructing CG. The method constructs the CG once at the beginning. The CG is used to coordinate the analysis so that the information is propagated to the correct function inside the VP.

*d) **Control Flow Graph** (**CFG**) & **Control Flow List** (**CFL**):* The CFG extracted from the *Abstract Syntax Tree* (AST) of the VP using Clang to understand the relationship between various statements (data flow and control flow) of the design. We create the CFL for each time step of the testbench results. To create it, we take the CFG and add the traversed nodes of the CFG to the CFL. The traversal always starts from the root node of the CFG. When it is at a node that has a single child, it moves to it. When the current node is of a conditional type and has multiple children, the traversal moves to the child that was executed. To find out whether a child node was executed, we create a log file during the testbench execution, where we log the visited nodes with file names and line numbers. SiMiT uses that file to traverse to the executed child by comparing the file name and line number of the children nodes with the log file.

*e) **Data Flow Analysis** (**DFA**):* The DFA is used to create the definition-use (def-use) and the use-to-dependence (use-dep) pairs, for a given VP by performing intra-function analysis. The def-use pairs for a VP effectively addresses the question of "for each defined variable, which uses may potentially utilize its values?". The use-to-dependence represents the dependence for variables in the conditional statements of CFG blocks, where definitions in the possible succesors to the conditional statements are stored.

*f) **Data Dependency Graph** (**DDG**) & **Observed Data Dependency Graph** (**ODDG**):* The DDG is used to understand the relationship between the variables in a design, including signals, ports, and global and local variables of all modules. It is created from the CFG, def-use pairs, use-to-dependence pairs, function calls, and BI. The ODDG for each time step is created similarly, but by using the corresponding CFLs of the time steps, instead of the CFG.

*g) **Static Taint Analysis** (**STA**):* The STA is performed to generate the *Observed Dependency List* (ODL). It starts with a tainted source and adds variables while performing a *Depth First Search* (DFS) based on dependence data from each ODDG.

B. Illustration of Security Coverage Metrics in SiMiT

The following section explains how security coverage metrics are implemented in SiMiT, and shows the results from the motivating example.

*a) **DSC**:* To evaluate the DSC metric, from an HS-tagged signal A to an AA tagged signal B (see Eq. (1)), we identify the EIFs. To determine whether a variable is affected by secure inputs (HS tag), a DDG uses forward tracing from the corresponding secure input node to a *Secure Output* (SO) node (AA tag). The HS tag is assigned to all nodes in this trace that are related to the *Secure Input* (SI) and are added to the *Sensitive List of Secure Inputs* (SLSI). Furthermore, because the output variables may receive their final values via the intermediate variables, a backward tracing on the DDG is also performed to extract the variables of assignment statements that are explicitly related to the outputs with the AA tag. These nodes are added to the *Sensitive List of Secure Outputs* (SLSO). The CFG of VP is then analyzed to find all sensitive control signals that influence the occurrence of updates on variables with AA tags. Each control flow of CFG condition node type (e.g., if-else, switch-case, while, etc.) is visited, and its control variables are retrieved. If the intersection of the condition node's extracted control variables and the SLSI is not empty, additional analysis is performed on the condition node's child nodes, which are not conditional node types. This analysis identifies assignment statements whose left-hand side variables are in the SO list, in the case of EIFs. From the motivating example, one of the SP is defined as follows:

$$SP = (\{\texttt{bt\_enable\_in} = HS\}, \{\texttt{grant\_nfc} = AA\}) \quad (2)$$

The SP in Eq. (2) ensures that the signal `grant_nfc` sent by the NFC module to the *Bus* module must not be dependent on the primary input `bt_enable_in` of the BT module seen in Fig. 3. According to the assessment no direct flow is found in between, and $DSC = FALSE$.

*b) **ISC**:* The ISC metric distinguishes itself from the DSC metric in terms of handling paths from an HS-tagged signal A to an AA-tagged signal B that involve conditional edges. In such cases, one more step in addition to DSC is to look at the SLSO that gives the IIFs. According to the assessment with same *SP* in Eq. (2), we found that there exist indirect flow via controlling variable `bt_request` (n_{11}) (seen in Fig. 3), and $ISC = TRUE$.

*c) **PPA**:* The PPA implementation starts with gathering the number of paths, n_P, from the DDG where information originating from signal A reaches signal B. In the next step, for each time step of the testbench results, the ODDGs are used to find the number of partially activated paths, n_{PPA}, where information from signal A has reached signal B.

$$PPA(HS, AA, t1, t2) = \frac{n_{PPA}(HS, AA, t1, t2)}{n_P(HS, AA)} \quad (3)$$

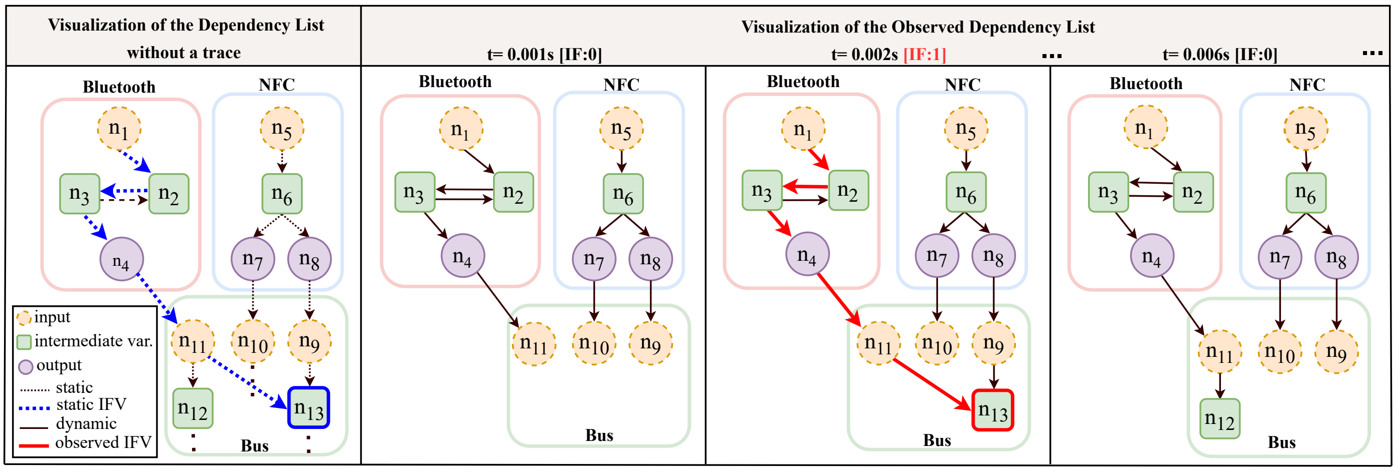


Fig. 3: The a part of the results of security coverage metrics from the motivating example, n_1: `bt_enable_in`, n_2: `bt_tag`, n_3: `bt_proximity_threshold`, n_4: `bt_pair_out`, n_5: `nfc_enable_in`, n_6: `nfc_proximity_threshold`, n_7: `nfc_tag_out`, n_8: `nfc_request_out` (NFC), n_9: `nfc_request` (Bus), n_{10}: `nfc_tag_id`, n_{11}: `bt_request`, n_{12}: `grant_bt`, n_{13}: `grant_nfc`, IFV: Information Flow Violation.

For the motivating example, from `bt_enable_in` to `grant_nfc` in Eq. (3), we find $n_P = 1$, $n_{PPA} = 1$, and $PPA = 1$ for the first 10 ms.

*d) **FPA**:* The FPA metric implementation distinguishes itself from the PPA by quantifying the total path activation for the whole execution time of the testbench. It is defined as:

$$FPA(HS, AA) = \frac{n_{FPA}(HS, AA)}{n_P(HS, AA)} \quad (4)$$

For the motivating example, from `bt_enable_in` to `grant_nfc` in Eq. (4), we find $n_P = 1$, $n_{FPA} = 1$, and $FPA = 1$.

*e) **IFR**:* To implement the IFR metric, we attach the tainted source information (HS tag), and perform STA for each tainted source to create the ODL. This list is scrutinized to calculate the percentage of IFR as follows:

$$IFR(HS, AA, s_1, s_2) = \frac{NF(HS, AA, s_1, s_2)}{1 + s_2 - s_1} \cdot 100 \quad (5)$$

The IFR metric in Eq. (5), is computed over a specified time interval, from sample index s_1 to s_2. The '1' is added since both s_1 and s_2 are inclusive. In this calculation, NF represents the number of flows from the HS tag (source) signal to the AA tag (destination) signal.

From the motivating example, the IFR metric results seen in Fig. 3 show one of the situations where `bt_enable_in` reaches `grant_nfc` at 2ms. In total we found that it happens 2 times in 10 ms, for $s_1 = 1$, $s_2 = 100$, and results in IFR = 2%.

VI. Experimental Results

We have demonstrated the applicability of the proposed security coverage metrics using an open source RISC-V VP [10] to demonstrate the applicability of the metrics. The RISC-V VP is based on the SystemC hardware modeling language and utilizes the TLM 2.0 modeling style. We evaluate the methods in three steps. First, we present a case study for an *Electronic Control Unit* (ECU) of a car engine immobilizer. Then, we show the effectiveness of our methods in detecting availability security problems. All the experiments were carried out on a PC equipped with 24 GB RAM and an Intel Core i7-8565U CPU running at 1.8 GHz.

Car Engine Immobilizer

In this experiment, we focus on a case study involving an ECU of a car engine immobilizer equipped with SoC based on a RISC-V *Central Processing Unit* (CPU), an *Universal Asynchronous Receiver/Transmitter* (UART), an *Controller Area Network* (CAN) controller, a *Platform Level Interrupt Controller* (PLIC), and a Memory centered around a central Bus shown in Fig. 4. The other peripherals are abstracted away for brevity.

The CAN plays a crucial role in managing low-level communication between the vehicle's internal systems, such as sensors, actuators, and the central processing unit CPU. On the other hand, the UART peripheral enables debugging capabilities and represents the attack surface a potential attacker can abuse to interact with the system. In both cases, the system interaction is modeled as random incoming received packets that are handled through buffers, that trigger configured interrupts upon a configured buffer limit. Both modules interact with the CPU through the PLIC, based on external interrupts. For the identification of the CAN and UART, the PLIC utilizes unique identifications, in our case study id = 2 for CAN and id = 8 for the UART.

In such a system, a potential security vulnerability may arise when the UART is (accidentally) given a higher priority

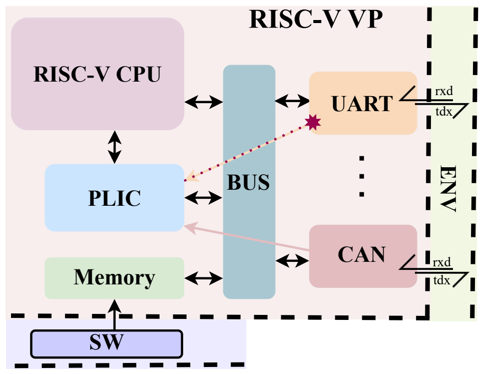


Fig. 4: The RISC-V VP model of a Car Engine Immobilizer

TABLE I: Security Coverage Metrics Results

No.	AA-tagged SOs	DSC	ISC	PPA	FPA	IFR (%)
SP_1	`interrupt_can`	FALSE	TRUE	1	1	$3\cdot10^{-4}$
SP_2	`hart_config`	FALSE	TRUE	1	1	$7.79\cdot10^{-3}$
SP_3	`c.m.f.m`	FALSE	TRUE	1	1	$7.78\cdot10^{-3}$
SP_4	`c.m.f.e`	FALSE	TRUE	1	1	$7.38\cdot10^{-3}$
SP_5	`target_harts`	FALSE	TRUE	1	1	$7.79\cdot10^{-3}$

`c.m.f.e`: `csrs.mcause.fields.exception_code`
`c.m.f.m`: `csrs.mip.fields.meip`

configuration than the CAN peripheral. If the UART generates an interrupt more frequently and reaches a certain threshold, the PLIC and thus the CPU will prioritize its processing, leading to delays or even the complete disregard of incoming CAN messages. Consequently, this configuration flaw prevents effective communication with the CPU and peripherals with crucial tasks, such as adaptive cruise control or collision avoidance.

The software executed on the processor initially configures the devices (i.e. the device interrupt configurations), registers interrupt handler callbacks for the CAN (id = 2) and UART (id = 8), and sets their respective priorities in the PLIC. The latter part contains the flaw, as the configured priorities are set such that the UART is prioritized over the CAN. In a normal use-case, such a flaw would not be visible due to the unused access to the ECUs debugging interface. Regarding availability, the CAN interrupt handling signals must be independent of unrelated signals, such as those generated by excessive tasks from the UART, which may cause the threshold for sending interrupt signals to be exceeded. Accordingly, we have defined multiple *SP*s with AA-tagged SO signals for the same HS-tagged input signal `plic_uart`, but chose five of these *SP*s that have IIF to present in Table I. They check whether the value of the SO signals are dependent on `plic_uart` via the other controlling variables. This means that the signal `plic_uart`, which is supposed to be isolated from the AA-tagged SO signals may affect the availability of CAN messages. We created trace instances for 15 ms and observed 196313 samples. As an example from Table I, SP_1 failed in the static analysis, where the variable `interrupt_can` was found to be dependent (implicitly) through 6 paths to `plic_uart` with controlling variables; `plic` and `min_id`. Then, we observed using dynamic analysis that 6 of these 6 paths from `plic_uart` to `interrupt_can` were activated. Additionally, `interrupt_can` was reached by `plic_uart` in 6 of the observed samples.

VII. Conclusion

In this paper, we have introduced a novel set of security coverage metrics designed for Information Flow at the system level. We present SiMiT, a tool that leverages scalable Static and Dynamic IFT techniques to implement these metrics directly on SystemC VP models with the TLM 2.0 extension. SiMiT identifies static paths and, for each time step in the trace results, observes instances to assess security properties and identify violated information flows. The applicability of these assessments has been demonstrated through the analysis of a complex RISC-V VP model.

The accuracy of the results, particularly concerning false positives, depends on the precision of static analysis and the reliability of input cases from dynamic analysis. Our study primarily focuses on digital systems, suggesting future research directions to extend the scope to analog-mixed signal systems and explore broader security aspects, such as confidentiality and integrity for a more comprehensive understanding.

References

[1] F. Sakiz and S. Sen, “A survey of attacks and detection mechanisms on intelligent transportation systems: VANETs and IoV,” *Ad Hoc Networks*, vol. 61, pp. 33–50, Jun. 2017.

[2] I. A. Sumra, I. Ahmad, H. Hasbullah, and J.-l. bin Ab Manan, “Classes of attacks in VANET,” in *2011 Saudi International Electronics, Communications and Photonics Conference (SIECPC)*, Apr. 2011, pp. 1–5.

[3] I. Aad, J.-P. Hubaux, and E. W. Knightly, “Impact of Denial of Service Attacks on Ad Hoc Networks,” *IEEE/ACM Transactions on Networking*, vol. 16, no. 4, pp. 791–802, Aug. 2008.

[4] E. N. Demirhan Coşkun, M. Hassan, and R. Drechsler, “Security Validation of VP-based Heterogeneous Systems: A Completeness-driven Perspective,” in *Methoden und Beschreibungssprachen Zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, Mar. 2023.

[5] E. N. Demirhan Coşkun, M. Hassan, M. Goli, and R. Drechsler, “VAST: Validation of VP-based Heterogeneous Systems against Availability Security Properties using Static Information Flow Tracking,” in *2023 24th International Symposium on Quality Electronic Design (ISQED)*, Apr. 2023, pp. 1–8.

[6] M. Hassan, D. Große, and R. Drechsler, *Enhanced Virtual Prototyping for Heterogeneous Systems*. Springer, 2023.

[7] “IEEE Standard for Standard SystemC Language Reference Manual,” *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)*, pp. 1–638, Jan. 2012.

[8] F. Ghenassia, Ed., *Transaction Level Modeling with SystemC: TLM Concepts and Applications for Embedded Systems*. Boston, MA: Springer US, 2005.

[9] R. Drechsler, M. Diepenbeck, D. Große, U. Kühne, H. M. Le, J. Seiter, M. Soeken, and R. Wille, “Completeness-Driven Development,” in *Graph Transformations*, ser. Lecture Notes in Computer Science, H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, Eds. Berlin, Heidelberg: Springer, 2012, pp. 38–50.

[10] V. Herdt, D. Große, H. M. Le, and R. Drechsler, “Extensible and Configurable RISC-V Based Virtual Prototype,” in *2018 Forum on Specification & Design Languages (FDL)*, Sep. 2018, pp. 5–16.

[11] K. Xiao, A. Nahiyan, and M. Tehranipoor, “Security Rule Checking in IC Design,” *Computer*, vol. 49, no. 8, pp. 54–61, Aug. 2016.

[12] M. Rostami, F. Koushanfar, and R. Karri, “A Primer on Hardware Security: Models, Methods, and Metrics,” *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, Aug. 2014.

[13] B. Mao, W. Hu, A. Althoff, J. Matai, Y. Tai, D. Mu, T. Sherwood, and R. Kastner, “Quantitative Analysis of Timing Channel Security in Cryptographic Hardware Design,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 9, pp. 1719–1732, Sep. 2018.

[14] X. Guo, R. G. Dutta, J. He, M. M. Tehranipoor, and Y. Jin, “QIF-Verilog: Quantitative Information-Flow based Hardware Description Languages for Pre-Silicon Security Assessment,” in *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, May 2019, pp. 91–100.

[15] L. M. Reimann, L. Hanel, D. Sisejkovic, F. Merchant, and R. Leupers, “QFlow: Quantitative Information Flow for Security-Aware Hardware Design in Verilog,” in *2021 IEEE 39th International Conference on Computer Design (ICCD)*, Oct. 2021, pp. 603–607.

[16] G. K. Contreras, A. Nahiyan, S. Bhunia, D. Forte, and M. Tehranipoor, “Security vulnerability analysis of design-for-test exploits for asset protection in SoCs,” in *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2017, pp. 617–622.

[17] A. Meza and R. Kastner, “Information Flow Coverage Metrics for Hardware Security Verification,” Apr. 2023.

[18] K. Lounis and M. Zulkernine, “Bluetooth Low Energy Makes “Just Works” Not Work,” in *2019 3rd Cyber Security in Networking Conference (CSNet)*, Oct. 2019, pp. 99–106.

[19] E. Jonsson, “Towards an integrated conceptual model of security and dependability,” in *First International Conference on Availability, Reliability and Security (ARES’06)*, Apr. 2006, pp. 646–653.