Towards Completeness: Security Coverage Metrics for System Level Information Flow [*]

Ece Nur Demirhan Coşkun[a], Sallar Ahmadi-Pour[b], Muhammad Hassan[a,b], and Rolf Drechsler[a,b]
[a]Cyber-Physical Systems, DFKI GmbH, 28359 Bremen, Germany
[b]Institute of Computer Science, University of Bremen, 28359 Bremen, German

Abstract

The increasing use of complex, feature-rich systems necessitates robust security measures. A single vulnerability can trigger far-reaching and disastrous outcomes, such as rendering various *Intellectual Properties* (IPs) unavailable and causing system malfunction. It is crucial to integrate security policies early in the design phase and to define specific *Security Properties* (SPs) regarding threat models. To tackle various threat models and pinpoint potential violations, we assess the SPs using *Security Coverage Metrics* (SCMs). This paper provides an overview of SCMs targeting availability threats and related weaknesses for system level information flow. To implement the SCMs, we show SiMiT; a tool that leverages *Virtual Prototypes* (VP) and uses Static and Dynamic *Information Flow Tracking* (IFT) techniques. We demonstrate the applicability of the SCMs on an open-source RISC-V VP to show how these metrics advance the concept of security-aware *Completeness Driven Development* (CDD) and secure *System-on-Chip* (SOC) designs. Finally, we discuss the future direction of SCMs.

1 Introduction

Hardware designs with real-time requirements are increasingly vulnerable to security threats, such as *Denial of Service* (DoS) attacks [1]. These attacks pose a significant risk as they can render the *Intellectual Properties* (IPs) inoperative when they are most needed. This could potentially lead to the IPs failing to function properly, or even cause the entire system to malfunction [1]. Such vulnerabilities highlight the importance of secure design in modern *System-on-Chip* (SOC) architectures.
Information Flow Tracking (IFT) is an effective security validation technique, particularly for addressing vulnerabilities which violate information flow policies. These policies are fundamental for maintaining important aspects like confidentiality, integrity, and availability [2]. The effectiveness of IFT hinges on the precise definition of *Security Properties* (SPs). These definitions aim to identify vulnerabilities that align with the specific threat model being targeted. Therefore, it is vital to conduct both qualitative and quantitative analyses of these SPs. These analyses should include the use of *Security Coverage Metrics* (SCMs) as part of the security validation process [3]. These metrics are vital for verification engineers to identify coverage gaps, enabling them to effectively evaluate system weaknesses, pinpoint vulnerabilities, and understand their impacts. Such understanding is important for strengthening the security framework of SOC designs.
These security measures should be integrated from the beginning of the SOC design process, due to strict *Time-to-Market* (TTM) constraints [2]. This necessity has led to a growing preference for *Virtual Prototypes* (VPs) in the semiconductor industry [2, 3]. VPs are sophisticated software models of hardware, often developed using SystemC and its *Transaction Level Modeling* (TLM) formalism [4]. They serve as critical reference points for early stages of software and hardware development. VPs are utilized at the *Electronic System Level* (ESL) as the initial step for early design and verification within the concept of *Completeness-Driven Development* (CDD) [5]. The high level idea of CDD is to verify every aspect of the design at each abstraction level (e.g., System/RTL/Gate/Layout), and to ensure that each stage of development is completely checked in a measurable manner before proceeding to the next.
The existing CDD concept primarily emphasizes functional correctness, but does not consider security. Developing a security-aware CDD concept is key to identifying the security weaknesses. This involves the use of SCMs to analyze SPs and reduce possible risks. While various SPs for IFT have been established to assess VP-based models against security threats [2], there are still shortcomings in SCMs for evaluating these SPs at the system level.
In this paper, we provide an overview of SCMs as a crucial part of the security validation technique of VP-based systems within the context of CDD. Therefore, we focus on assessing various SPs by measuring them qualitatively and quantitatively. Then, we show SiMiT, a Static and Dynamic IFT tool, designed to implement these SCMs at the system level. By evaluating these SPs, we can ensure that IFT techniques satisfy the necessary criteria for achieving the desired security sign-off, an essential milestone in the verification process. Our experimental results on an open-

[*]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.

source RISC-V VP case study show the applicability of the SCMs effectively.

2 Preliminary

This section explains the targeted threat model, availability, and gives a motivating example to guide the explanation of the SCMs.

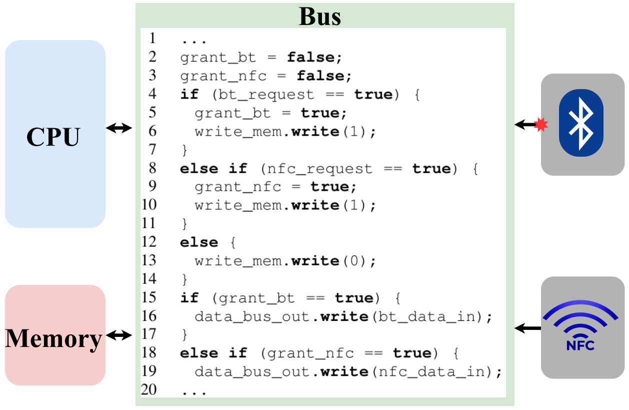
2.1 Threat Model

In this paper, the threat model is based on the principles of *Confidentiality*, *Integrity*, *Availability*, and *Authentication* (CIAA), with a specific focus on availability. Availability problems arise when an IP overuses shared resources, making them inaccessible to other IPs [2]. In essence, availability refers to the timely access of these IPs.

2.2 Motivating Example

This section describes a simplified SystemC model of *Keyless Entry System* (KES) to illustrate the usage of the SCMs. It consists of *Near Field Communication* (NFC) and *Bluetooth* (BT) modules to enable smart lock/unlock function through an authentication process. The model shown in Figure 1 includes a Bus which connects an NFC module, a BT module, a *Central Processing Unit* (CPU), and a Memory unit. The NFC supports short-range wireless communication for authentication to give access to the authorized entities between the KES and NFC-compatible smartphones. Similarly, the BT is utilized to authenticate and grant access to authorized entities. The CPU serves as the control unit and is responsible for processing data, managing access control policies, and handling keyless entry features as well as coordinating the integration of the NFC, the BT, and Memory modules. The Memory stores authorized credentials and vital operational data for the KES, including configuration settings and access logs related to communication and authentication.

In a scenario involving an improperly configured access control system within the Bus IP, shown in Figure 1; the BT is erroneously prioritized over the NFC for memory access. Here, the BT authentication takes priority (Line 4), while the NFC access is deprioritized to a secondary status (Line 8). This arrangement leads to the BT requests initiating data transfer to memory, whereas the NFC requests are only allowed access afterward. Misconfigured priorities can allow attackers to disrupt the NFC, missing critical



```
1   ...
2   grant_bt = false;
3   grant_nfc = false;
4   if (bt_request == true) {
5     grant_bt = true;
6     write_mem.write(1);
7   }
8   else if (nfc_request == true) {
9     grant_nfc = true;
10    write_mem.write(1);
11  }
12  else {
13    write_mem.write(0);
14  }
15  if (grant_bt == true) {
16    data_bus_out.write(bt_data_in);
17  }
18  else if (grant_nfc == true) {
19    data_bus_out.write(nfc_data_in);
20  ...
```

Figure 1 The SystemC model of a simplified KES

requests. These modules need equal memory access, but detecting such relation is challenging without automated IFT tools as well as effective SCMs.

3 Security Coverage Metrics

This section presents a novel set of SCMs for system level information flow. These metrics assess a range of SPs and specifically target the availability threat model.

The following subsections will delve into how each metric offers specific insights and detailed information.

Direct Signal Connectivity The *Direct Signal Connectivity* (DSC) metric determines direct/explicit flow between two signals. The *Explicit Information Flow* (EIF) can be observed when there is a direct communication between two modules. For instance, if the BT and the NFC from Figure 1 had a direct connection, the EIF could occur.

Indirect Signal Connectivity The *Indirect Signal Connectivity* (ISC) metric determines the implicit connection between two signals. This connection can be detected through *Implicit Information Flow* (IIF), which is subtler than EIF. The IIF can lead to the unavailability of IPs due to certain behaviors. It commonly occurs between two modules where one is in a trusted zone and the other is in an untrusted zone, sharing a common memory. An example of IIF is the interaction between the BT in the untrusted zone and the NFC in the trusted zone, both accessing memory via the Bus.

Partial Path Activation The concept of information flow involves more than just connectivity; it necessitates the actual activation of these connections. The *Partial Path Activation* (PPA) metric quantifies the extent to which a path is activated between two signals during a specified time interval $[t_1, t_2]$. A path is activated when data successfully flows from a signal to reach another signal. The PPA metric is crucial for demonstrating that the presence of a pathway between these signals does not always guarantee flow of information.

Full Path Activation The *Full Path Activation* (FPA) serves as a quantitative metric that quantifies the complete activation of paths over the entire execution. For instance, if there are five distinct paths from one signal to another, FPA observes how many of these paths remain active throughout the full execution time.

Information Flow Rate The *Information Flow Rate* (IFR) metric captures how frequently information flow occurs from one signal to another over a specific time frame. By calculating the IFR, we can understand the number of instances in which the signal successfully communicates with the other signal within this time frame.

4 SiMiT: A static+dynamic IFT tool

This section explains SiMiT, and the implementation of the SCMs, as shown in Figure 2.

4.1 Methodologies of SiMiT

SiMiT is a tool that leverages both Static and Dynamic IFT methods. It uses Clang to capture and accurately track information for expressions, statements, and other constructs in a program. It starts with the trace generation, saving the

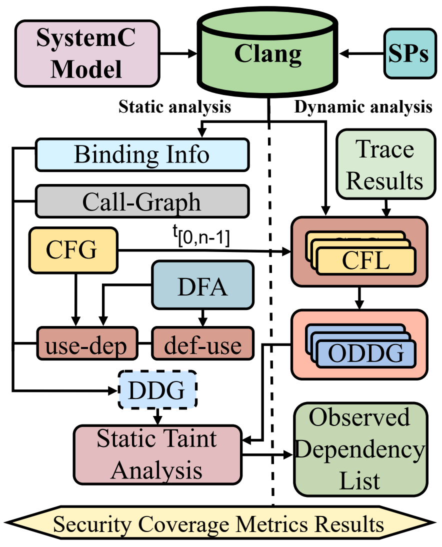


Figure 2 The workflow of SiMiT

information flow of a given SystemC model at a specific time frame, along with the current file names and their corresponding line numbers. Next step is to define availability SPs, which necessitates that various IPs must be available in a timely manner [6]. Therefore, we define SPs as follows:

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

Eq. (1) ensures that inputs with the *High Security* (HS) tag are handled in a manner that guarantees outputs are *Always Available* (AA) when needed.

After that, the *Binding Information* (BI) is extracted to establish module connectivity. This information is crucial for comprehending how data flows through the system and for constructing *Call-Graphs* (CG). The CG is utilized to coordinate the analysis, ensuring that information is accurately propagated to the respective function within the VP. Next, *Control Flow Graph* (CFG) is extracted by adding traversed nodes from the CFG to a *Control Flow List* (CFL), starting from the CFG's root node. The traversal process moves to the single child of a node when present. In cases where the current node is conditional and has multiple children, the traversal proceeds to the child node that was executed. To determine which child node was executed, we maintain a log file during the trace generation. This log records the visited nodes, including file names and line numbers. SiMiT then uses this log to navigate to the executed child node by matching the file names and line numbers of the child nodes with those in the log. Then, *Data Flow Analysis* (DFA) is used to construct definition-use (def-use) and use-to-dependence (use-dep) pairs for a given VP through intra-function analysis. Next, the *Data Dependency Graph* (DDG) is used to map the relationships among variables in a design, including signals, ports, and both global and local variables across modules [2]. It is generated using the CFG, def-use pairs, use-dep pairs, function calls, and BI. In contrast, the *Observed Data Dependency Graph* (ODDG) for each time step is formed using the relevant CFLs for those time steps, rather than the CFG. Then, *Static Taint Analysis* (STA) is conducted to generate the *Observed Dependency List* (ODL). This process begins with a tainted source and incrementally includes variables by executing a *Depth First Search* (DFS) that utilizes dependence data from each ODDG.

4.2 Implementation of SCMs in SiMiT

DSC: To evaluate the DSC metric for a transition from an HS-tag signal to an AA-tag signal (see Eq.(1)), we identify EIFs. To determine if a variable is influenced by *Secure Inputs* (SIs), we employ the DDG that uses forward tracing from the corresponding SI node to a *Secure Output* (SO) node. Nodes in this trace related to the SI are assigned the HS-tag and added to the *Sensitive List of Secure Inputs* (SLSI). Additionally, since output variables may derive their values from intermediate variables, we conduct backward tracing on the DDG to extract assignment statements explicitly linked to outputs with the AA-tag. These nodes are added to the *Sensitive List of Secure Outputs* (SLSO). Then, the CFG of VP is analyzed to identify all sensitive control signals influencing updates on variables with AA-tags. Each control flow node type (e.g., if-else, switch-case, while) is visited, and their control variables are retrieved. If the intersection of the condition node's extracted control variables and SLSI is not empty, an additional analysis is performed on the condition node's child nodes, excluding conditional node types. This analysis identifies assignment statements with left-hand side variables in the SO list in the case of EIFs. From the motivating example, one of the SP is defined as follows:

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

In Eq. (2), the SP ensures that the signal `grant_nfc` sent by the NFC module to the *Bus* module must not be dependent on `bt_enable_in` of the BT module seen in Figure 1. According to the assessment no direct flow is found in between, and $DSC = FALSE$.

ISC: This metric distinguishes itself from the DSC in terms of handling paths that connect an HS-tag signal to an AA-tag signal through conditional edges. In such cases, an additional step, beyond what DSC provides, involves examining the SLSO to determine the IIFs. According to the assessment with same *SP* in Eq. (2), there exist indirect flow via controlling variable (in Figure 1) `bt_request` (n_{11}), and $ISC = TRUE$.

PPA: The implementation of PPA begins by identifying the number of paths, denoted as n_P, within the DDG where information originating from a HS-tag signal eventually reaches AA-tag signal. In the subsequent step, for each time step in the trace results, we utilize the ODDGs to determine the count of partially activated paths, represented as n_{PPA}, where information from HS-tag signal successfully propagates to AA-tag signal.

$$PPA(HS,AA,t_1,t_2)=\frac{n_{PPA}(HS,AA,t_1,t_2)}{n_P(HS,AA)} \quad (3)$$

To illustrate, from `bt_enable_in` to `grant_nfc`, we found $n_p = 1$, $n_{PPA} = 1$, and $PPA = 1$, using Eq. (3) for the first 10 ms.

FPA: The implementation of the FPA metric sets it apart from the PPA metric by measuring the overall path activation throughout the entire execution. FPA is defined as:

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

Table 1 Security Coverage Metrics Results

No.	AA-tagged SOs	DSC	ISC	PPA	FPA	IFR (%)
SP_1	`interrupt_can`	FALSE	TRUE	1	1	$3{\cdot}10^{-4}$
SP_2	`hart_config`	FALSE	TRUE	1	1	$7.79{\cdot}10^{-3}$
SP_3	`target_harts`	FALSE	TRUE	1	1	$7.79{\cdot}10^{-3}$

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

IFR: To calculate the IFR metric, SiMiT starts by attaching the tainted source information, denoted by the HS-tag. Next, it performs STA for each tainted source, resulting in the creation of the ODL. The ODL is then carefully examined to determine the percentage of IFR, using the following formula,

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

computed over a specified time interval, ranging from sample index s_1 to s_2. The addition of '1' is necessary since both s_1 and s_2 are inclusive in this context. In this calculation, NF represents the number of flows from the HS-tag signal to the AA-tag signal. At the motivating example, `bt_enable_in` reaches `grant_nfc` at 2 ms. We found that it occurs 2 times in 10 ms, for $s_1 = 1$, $s_2 = 100$, and results in IFR $= 2\%$.

5 Experimental Results

We applied the SCMs to a RISC-V VP [7] based on SystemC with TLM 2.0 modeling. The case study is an *Electronic Control Unit* (ECU) for a car engine immobilizer, which includes a SOC based on a RISC-V *Central Processing Unit* (CPU), a *Universal Asynchronous Receiver/Transmitter* (UART), a *Controller Area Network* (CAN) controller, a *Platform Level Interrupt Controller* (PLIC), and memory centered around a Bus. While the CAN manages low-level communication between the vehicle's internal systems, the UART enables debugging capabilities, but poses a potential attack surface that attackers could exploit to interact with the system. Both modules interact with the CPU trough the PLIC processing incoming packets, which are received, buffered, and trigger configured interrupts upon reaching a specified buffer limit.

Misprioritizing the UART over the CAN can lead to CPU delays, hindering communication with peripherals using critical tasks. The software on the processor sets up device interrupts and registers handler callbacks for CAN (id = 2) and UART (id = 8). A flaw prioritizes UART over CAN, which usually does not impact normal use due to limited access to the ECU's debug interface. However, CAN interrupt handling must be distinct from UART tasks to avoid signal interference. We identified multiple SPs influenced by UART signals, highlighting a potential impact on CAN message availability. Our analysis over a 15 ms trace observed 196,313 samples to assess this effect. For example, from Table 1, in static analysis, SP_1 failed due to `interrupt_can`, which was implicitly dependent on `plic_uart` through six identified paths, influenced by the controlling variables `plic` and `min_id`.

6 Conclusion and Future Direction

We have discussed a set of SCMs for system level information flow, and presented SiMiT, a tool that leverages scalable Static and Dynamic IFT techniques to implement the SCMs on SystemC VP models. We showed the demonstration of the SCMs on an open-source RISC-V VP [7].

One potential research direction involves investigating the transferability of SPs across different abstraction levels (e.g., System/RTL/Gate/Layout). This requires appropriate completeness measures to ensure the overall behavior of the design is accurately assessed. In this regard, our study initially focuses on the system level. However, even if the security metrics of the design are satisfied at one abstraction level, transitioning to a lower-level abstraction may introduce new vulnerabilities. These are the "transformation-induced" changes, which can expose it to additional vulnerabilities. Consequently, SPs should be adapted for each subsequent abstraction level. This emergence necessitates defining additional properties to maintain the security goal, ensuring consistency with the overall objective. By following the CDD concept with a focus on security in the early design phase, future research can offer a more holistic and comprehensive view.

7 Literature

[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] 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.

[3] E. N. Demirhan Coşkun, S. Ahmadi-Pour, M. Hassan, and R. Drechsler, "Security Coverage Metrics for Information Flow at the System Level," in *29th Asia and South Pacific Design Automation Conference (ASP-DAC'24) (Accepted).*

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

[5] 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.

[6] 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.

[7] 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.