
Thoshitha T. Gamage  Bruce M. McMillin  Thomas P. Roth
{ttgd5@mst.edu, ff@mst.edu, tprfh7@mst.edu}
Department of Computer Science
Intelligent Systems Center
Missouri University of Science and Technology, Rolla, MO 65409-0350

Abstract—This paper presents a general theory of event compensation as an information flow security enforcement mechanism for Cyber-Physical Systems (CPSs). The fundamental research problem, in a broader sense, being investigated is the fact that externally observable events in modern CPSs have the propensity to divulge sensitive settings to adversaries, resulting in a confidentiality violation. This is a less studied yet emerging concern in modern system security. A viable method to mitigate such violations is to use information flow security based enforcement mechanisms since access control based security models cannot impose restrictions on information propagation. A very few (if not none) models consider security at a system level rather than disjoint (cyber, physical and network) components; the disjoint nature of security analysis is not appropriate for systems with highly integrated physical and cyber infrastructures. In this respect, the compensation based security framework, proposed here, is foundational work that unifies cyber and physical aspects of security through shared semantics of information flow.

1 INTRODUCTION

Preserving the confidentiality of sensitive actions/events is a vital aspect of modern system security. By uncovering sensitive actions, adverse parties can identify the crucial components of a system and target specific attacks. Most modern systems are vastly complex in interconnective, multidisciplinary and composite nature, and rely on other similar systems for proper functionality. Numerous integration levels within these systems can be broadly categorized into two domains; a cyber infrastructure (computations, control algorithms, decision engines, databases, etc.) and a physical infrastructure (physical processes and components, links and connections, etc.). Commonly known as Cyber-Physical Systems (CPSs), such systems are put together to provide better resource utilization, control, fault tolerance and performance [1]. The prominent feature of these systems is that embedded computers and communication networks govern both physical manifestations and computations. This, in turn, affects how the two primary infrastructures interact with each other, and the outside world [2].

The proper functionality of CPSs has direct impact on the nations’ economic and social stability. In August 2003, an estimated 50 million people in the Midwest and Northeast portions of the United States and Ontario, Canada were affected by a power outage. In some parts of the United States, the outage lasted for 4 days [3]. It was estimated that the total cost in the United States ranged from $4 to $10 billion and in Canada, a loss in gross domestic
production of 0.7% for August. Although this event was triggered by a cascading failure in power lines loosely coupled with a cyber failure, rather than an intentional act of sabotage, it highlights and prompts the need for extensive research, methodologies, new tools and models of security to safeguard these critical infrastructures. Treating CPS security in a disjoint manner will prove fatal for future directions in system security. Although, for functional purposes, a CPS can be regarded as the intersection of properly built individual (cyber, physical, network) components. This unfortunately, is not the case in terms of security.

Within the context of system security policies and mechanisms, two main approaches can be identified: access control based methods and information flow based methods. However, a considerable amount of evidence found in literature suggest that access control might not be the future direction of system security [4–11]. From a security stand point, the problem in CPSs is that certain aspects of the physical portion is always observable. As a side effect, explicit information flow violations take place since externally observable physical manifestations can divulge sensitive system settings; adversaries to the system can potentially derive sensitive internal system settings by observing the external system changes (see [12] for a formal analysis). Such derived knowledge coupled with the semantic knowledge of the system can be used against the system in the form of integrity and availability attacks. This is, in fact, a direct consequence of confidentiality violation of sensitive action/event settings. Unfortunately, access control based security models cannot impose restrictions on information propagation. A viable alternative is to use flow based security models such as information flow security enforcement mechanisms.

Even within the information flow enforcement domain, only a handful (if not none) of work is done towards CPS security. The disjoint nature of analysis in existing models is not appropriate for systems with highly integrated components. Rather, CPS security needs to be considered at the system level [13]. Earlier developments in enforceable security properties were strictly based the safety property [14]. The safety property requirement is too strict for CPS security analysis and prevents Information Flow Security Properties (IFPs) from being enforced and monitored [6, 15, 16]. Further, static enforcement mechanisms are too imprecise [17]. However, unwinding theorem based security automata can be designed to capture possibilistic future execution sequences by not having to detect violations at present, but eventually sometime in future [10, 18, 19]. This allows IFPs to be enforced and monitored outside the Alpern-Schneider framework [20].

This paper presents an event compensation based generalized framework to enforce IFPs in CPSs. At the core of this concept is a coordinated action-correction pair called the “Compensating Couple”. The idea is to compensate the observable effect of a (potentially information flow violating) system event (action) by pairing it up with an appropriate reaction event(s). By executing a compensating couple in a timely (i.e. within a certain time period) and coordinated manner, the expected net observable change is either insignificant in terms of deducing sensitive information or equivalent to some other system characteristic(s). Thus, the objective is to obfuscate the observable effects of a system. This is not to be confused with obfuscating actual physical actions which no amount of compensation can reverse.

In this respect, the contributions of this work are:

- Introduce a class of system properties called P-compensate properties which are execution monitoring enforceable in cyber-physical systems. This extends the present understanding of enforceable security to the cyber-physical security domain
- Develop a semantic model to analyze confidentiality violation in cyber-physical systems. In particular, the ability to deduce sensitive system settings using inherently observable changes is precisely characterized
- Extend previous work on runtime enforceable policies by combining a predicate mechanism with the ability to inject events. This is used in formally developing an information flow based semantic enforce-
ment mechanism for cyber-physical systems
- Extend existing enforcement mechanisms beyond the safety property requirements by proposing an event compensation based security framework. This is fundamentally a unification of cyber and physical aspects of security through shared semantics of information flow.

The rest of the paper is organized as follows. Section 2 lists some of the recent work in enforcing information flow security policies. Section 3 introduces the proposed framework. The applicability of the proposed work on a CPS is presented in Section 4. Section 5 presents the correlation between the inherent external observability and the resulting deducibility in CPSs and Section 6 shows how compensation can protect Nondeducibility security in a CPS. Section 7 lists the conclusion of this work. Appendix A lists the notation and the nomenclature used in this paper. Finally, Appendix B is a formal description for external observation based security threats for a simple system.

2 RELATED WORK

Access control security models [21–24] are not sufficiently strong enough to prevent indirect security violations due to implicit and explicit information flows. Such security models can only impose spatial restrictions on information and resources but fail to prevent propagation [9]. Further, they cannot control how the data will be used after been read. Explicit information flow violations are a result of direct (and possibly intentional) information passing to an insecure domain. Implicit information flows are comparatively passive in nature where (potentially) unintentional flow violation are a result of the structure and/or the interactions within the system. A good example is a CPS with physically observable changes.

Consider a system with two security clearances: a high-level domain ($D_H$) and a low-level domain ($D_L$). In terms of information flow security, the ability to deduce sensitive $D_H$ information at a $D_L$ is a security violation. In other words, information flows between processes which are not supposed to communicate should be prevented [5, 25]. The sensitivity of information and the “communication” depend on how each process views (other processes of) the system. The term “deduce” is a generalization for different ways information flow security can be compromised. Fundamental information flow security properties introduced over the years such as Noninterference [26], Noninference [27] and Nondeducibility [28] attempt to characterize and capture these flows.

The “possibilistic” nature of information flow security properties is formalization of how an (potentially $D_L$) observer with sufficient knowledge of a target system can deduce confidential information. This is done by constructing (and refining) the set of all possible system behaviors consistent with his set of observations. [29] distinguished two dimensions of possibilistic security; such that the occurrence of a confidential event (i) does not increase possible observations and (ii) does not decrease possible observations at lower clearance levels [29, 30]. It is a violation if any $D_H$ activities increase or decrease the likelihood of observations.

Information flow property enforcement is two fold: static compile time enforcement and runtime enforcement. Recent work under the former method includes secure-type systems [8, 9, 31], mechanisms based on petri nets [32–34], process algebra and program logic [7, 35–42], computation slicing [43–45] and bisimulation [40, 46–48]. However, static information flow enforcement mechanisms tend to be too imprecise. Amidst the final outcome preserving the enforced security policy, a static enforcer can potentially reject a program based on a partial analysis. As an example [17], consider the sequential program, $V_H := V_L; V_L := V_H$ where $V_H \in D_H$ and $V_L \in D_L$ are two variables in high-level and low-level domains respectively. A static enforcement mechanism, such as a security-type system, will reject this program considering the second assignment to be an information flow from $D_H$ to $D_L$ though this is not a violation.

2.1 Runtime Enforcing IFPs

Conceptually, runtime enforcement is monitoring a system during its execution to identify if
it deviates from the set of pre-identified desired behaviors and taking appropriate measures. This is also known as Execution Monitoring (EM) enforceability. Execution monitors work by monitoring the computational steps of untrusted programs and intervening whenever execution is about to violate the security policy being enforced [49]. Each of the intended behaviors (or forbidden behaviors) of the system are in fact security invariants. These can be evaluated based on a certain security predicate; a predicate acts as a decision engine.

The earliest threshold on EM enforceable security policies was established in [14] by stating that only safety properties can be enforced using a monitoring mechanism. The monitor in this case, a büchi-like state automata called the security automata, enforces a certain security policy by terminating the target when a security violation is detected in an execution. The safety requirement, unfortunately, precludes flow-based security properties from being enforced as they are not safety properties [15, 50]. These properties are defined over sets of execution sets rather than execution sets [15]. Thus, the decision to terminate an execution can not be purely based on a detected violation on a single execution. This fundamental issue has lead to two alternative classes of resolutions;

1) Extend the capability of the security automata using additional structures or operations [10, 18, 29, 30, 49, 51, 52]
2) Develop monitors which can enforce non-safety properties [18, 53, 54]

### 3 The Framework

The proposed work of this article improves EM security automata [14] by combining it with an emulator [10] and the event insertion capability of the edit automata [18]. The system model considered in this work is property based, which consists of a collection of properties \(P_S\), a monitoring mechanism \(M_S\), an enforcement mechanism \(E_S\) and an execution mechanism \(X_S\). \(M_S\) checks each execution before or during the execution to make sure it adheres to the associated policy. Valid executions are passed onto \(X_S\) and allowed to run where violations are passed onto \(E_S\). This is shown in Figure 1.

Traditional \(E_S\)'s, such as the security automata in [14], take a conservative approach of enforcement where upon detecting a violation the execution is immediately terminated. For a CPS, unfortunately, the system has already failed at this point since information flow requirements have been violated. This work, on the other hand, considers a more optimistic approach.

![Fig. 1. A Generalized System Model Based on Properties](image)

**Definition 1. [Property Based System Model]**

A property based system model is 4-tuple \((P_S, M_S, E_S, X_S)\) consisting of a monitoring mechanism \((M_S)\), an enforcement mechanism \((E_S)\), an execution mechanism \((X_S)\) and the system as a set of execution sets \((P_S)\).

This framework employs event compensation as the particular enforcement scheme. Event compensation is only applicable to executions which are eligible for cleansing. Cleansing an execution allows it to extend beyond a violation point and is prevented from being discarded. Cleansing, in general, can refer to mechanisms which allow temporary but controlled lapse of property, roll back or injection of error correction actions. Edit automata [18], which can modify the behavior of an execution during runtime (with suppression and injection), is a good example of execution cleansing.

**Definition 2. [Execution Cleansing]**

Execution cleansing refers to mechanisms which make qualified system executions eligible for extension and prevent them from being discarded.

Only a certain class of qualified executions can be extended in this manner. Suppose there exists an execution \(\sigma\) with a distinctly identifiable violation point \(\sigma_j\), a valid prefix [...i]
and a projected postfix \([k\ldots]\). The optimistic assumption is that, in the absence of \(\sigma_i\), the execution maintains property \(\mathcal{P}\). This is similar in concept to the suppression operation introduced in [18]. Formally, this characteristic can be denoted as,

\[
\sigma = \sigma[\ldots i] \, \sigma_j \, \sigma[k\ldots]
\]

and,

\[
\sigma[\ldots i] \, \sigma[k\ldots] \in (\mathcal{P} \in \mathcal{P}^*_s)
\] (1)

Executions similar in form to (1) are eligible for cleansing under the proposed enforcement scheme. Mathematically, this allows correction action(s) to be injected immediately after \(\sigma_j\) and extend the execution. However, such injection needs to compensate for the effects of \(\sigma_j\) in order to maintain the desired property. By performing event compensation, \(\mathcal{E}_S\) restores the system back to an operational state. Thus, such an optimistic view of the system is also a liveness [20] feature.

### 3.1 \(\mathcal{P}\)–Compensate Property

This work quantifies an execution step as a finite sequence of controlled state transitions. By doing so, \(\mathcal{E}_S\) is empowered to inject more than one correction action, depending on the requirement and the specific property expected to maintain.

**Definition 3. [Compensation Sequence]** For some identifiable execution violation point \(\sigma_j \in \sigma\), a compensation sequence \(\varsigma \in \Sigma^*\) is defined as a finite sequence of states starting with \(\sigma_j\) which can compensate for \(\sigma_j\) for some property \(\mathcal{P}\).

\[
\sigma[\ldots i] \, \sigma[k\ldots] = \sigma[\ldots i] \, \varsigma \, \sigma[k\ldots]
\]

where, \(\varsigma \in \Sigma^*, \varsigma = \sigma[j\ldots]\)

Consequently, associated with \(\varsigma\) is a finite sequence of compensating actions \(\varphi = <\phi^c_j, \phi^c_{j+1}, \ldots, \phi^c_k> \in \Phi^*\) corresponding to each state transition in \(\varsigma\). Thus, a compensated execution takes the following form.

\[
\sigma[\ldots i] \xrightarrow{\phi^c_j} \sigma_j \xrightarrow{\phi^c_{j+1}} \sigma_{j+1} \ldots \xrightarrow{\phi^c_k} \sigma[k\ldots]
\]

where, \(\varsigma \in \Sigma^*, \varsigma = \sigma[j\ldots]\)

Once an eligible execution is identified, a compensation sequence is calculated to compensate for \(\sigma_j\). This is done by carefully calculating a \(\varphi\) which can lead the overall system back to a safe state with respect to the IFP \(\mathcal{P}\). The compensative feature of a system also indicates the ability to cleanse executions respect to some \(\mathcal{P}\).

[12] formally showed the potential of event compensation to preserve IFPs in a CPS. Even though there is a momentarily lapse in the corresponding feature, the compensated execution as a whole would still adhere to the desired property. The idea is that the system as a whole, not individual operations, need to satisfy the required property [55].

**Definition 4. [\(\mathcal{P}\)–Compensate Property]** A system is compensative with respect to a property \(\mathcal{P} \in \mathcal{P}_S\) if and only if, for some execution \(\sigma\) with an identifiable violation point at \(j\), there exists a compensation sequence \(\varsigma\).

\[
\exists \sigma \in \Sigma^*, \, \sigma \notin \mathcal{P} : \neg \rho(\sigma[\ldots j]), \quad \exists \varsigma \in \Sigma^* : \varsigma = \sigma[j\ldots], \, \sigma[\ldots i] \, \varsigma \, \sigma[k\ldots] \in \mathcal{P}
\] (2)

### 3.2 Compensating Couple [12]

In the most basic form, \(\varsigma\) consists of a single element and two associated actions, i.e., \(|\varphi| = 2\). This is formally defined as a compensating couple. With this, the security automata can be extended to a compensation automata as follows. The state space \(\mathcal{Q}\) is divided into two sets. \(\mathcal{W} \subseteq \mathcal{Q}\) is the set of stable(safe) states and \(\mathcal{U} \subseteq \mathcal{Q}\) is the set of vulnerable(unsafe) states.

**Definition 5 (Compensation Automata).** The compensation automata consists of 6-tuples \((\mathcal{Q}, \mathcal{Q}_0, I, \delta, \mathcal{W}, \mathcal{D})\) where,

- \(\mathcal{Q}\) is a set automatons states
- \(\mathcal{Q}_0\) is a set of initial states for the automaton \(\mathcal{Q}_0 \subseteq \mathcal{Q}\)
- \(I \subseteq \Phi\) is a set of input symbols of the form \((\phi^c_{i-1}, \phi^c_i) : (\phi^c_{i-1}, \phi^c_i) \in I)\)
- \(\delta\) is the a state transition function \(\delta : \mathcal{Q} \times I \rightarrow 2^\mathcal{Q}\) specified under a predicate \(\rho()\)
- \(\mathcal{W}\) is a set of final states \(\mathcal{W} \subseteq \mathcal{Q}\)
- \(\mathcal{D}\) is the set of security domains

With respect to IFP \(\mathcal{P}\), the effect of executing a compensating couple needs to be zero, i.e.,
\[ \phi_i^c - \phi_{i-1}^c = \langle \rangle. \] The second event of the pair, \( \phi_i^c \), needs to lead the state machine back to a stable state as well as compensate for the first event \( \phi_{i-1}^c \). The predicate \( \hat{\wp}() \) is used to make sure that each compensating transition adheres to the particular property \( \mathcal{P} \) in concern. This enables the compensation automata to maintain \( \mathcal{P} \)-compensating property during each “compensating” step of the execution. Thus,

\[
\forall j, \neg \hat{\wp}(\sigma[...i]\sigma_j[k...]) \Rightarrow \hat{\wp}(\sigma[...i] \xrightarrow{\phi} \sigma_k)
\]

However, as a side effect of the characteristic equation 3, \( \varphi \) steps through a sequence of unsafe states; certain intermediate states of every \( \varsigma \) can potentially violate property \( \mathcal{P} \). However, the argument is that, \( \varsigma \) is finite by definition and \( \varphi \) is executed in a timely and controlled manner to avoid external observations. Thus, effect of the inherent security vulnerability is temporary.

Each event in a compensating sequence needs to be from the secure domain, i.e., \( \forall \phi_i^c \in \varphi, \phi_i^c \supseteq \mathcal{D}_H \). The input symbol \((\phi_{i-1}^c, \phi_i^c)\) is the last state transition command and the next state transition command under the read head. Suppose that due to \( \phi_{i-1}^c \), the state machine has already transit to \( q_i \in \mathcal{Q} \). if \( q_i \in \mathcal{W} \), \( \phi_i^c \) is the reply part of an earlier compensating couple. On the other hand, if \( q_i \in \mathcal{U} \), \( \phi_i^c \) denotes request part of a new compensating couple. For a detailed technical explanation see [12].

### 4 APPLICATION: ADVANCED POWER GRID

This section examines the practical applicability of compensating events for the advanced electric power grid - the power grid improved for reliability using a Flexible AC Transmission System (FACTS) network.

FACTS devices are reconfigurable/reprogrammable power electronic devices which can change specific parameters of transmission lines such as line impedance, active and reactive power flow. Figure 2 is a 13-bus test feeder with a FACTS device installed. The advanced power grid, in fact, can be regarded as the composition of numerous similar blocks.

Each FACTS device has a set of transmission lines (and buses) under its control. In the case of a failure, these devices recalculate the overall power redistribution and change line parameters accordingly. The exact change applied on a particular line(s) is calculated using a distributed control algorithm by communicating with other FACTS devices in the network. This way, the overall power balance of the network is properly maintained.

In terms of security, it is important to maintain the confidentiality of each FACTS device’s setting. If an adversary can derive the overall state of the system, such knowledge can be used to identify the most critical and vulnerable links (transmission lines) of the network. This in fact is a serious cyber-physical threat as such cognition can be used against the power grid not only in the form of physical attacks, but also on the cyber domain by forcing erroneous FACTS settings [56]; the system state may divulge sensitive operational limitations along with the present status of both FACTS devices and transmission lines. Thus, it is important to prevent sensitive cyber domain information from flowing to the physical domain through cyber-physical interactions.

#### 4.1 Modeling the Advanced Power Grid as a DC Circuit

Analyzing the power grid as it is for any purpose is a computationally intensive task, mainly because of the complexity of the network and the number of variables to be considered. However, in the context of information...
flow security, parallels could be drawn between the former and a DC circuit model.

![Fig. 3. A Symbolic and a Simplified Representation of the Advanced Electric Power Grid](image)

The ability of a FACTS device to change transmission line parameters is similar to the capability of a series connected variable resistor in a DC circuit. Consumers (loads) are in parallel to the source and in comparison to the source voltage and load resistance, transmission line impedance and resistance is negligible.

With this information, the advanced electric power grid shown in Figure 3 is comparative to the DC circuit model shown in Figure 7. In practice, there is a limit to how much a FACTS controller can change the power flow of a transmission line. This limitation is considered in this analysis by restricting that the variable resistors in Figure 7 can only have a $\pm20\%$ change from the initial setting$^\ast$.

5. **Correlation between Observability and Deducibility**

Observability refers to an external observer’s ability to monitor/record/observe system changes. Deducibility is the amount of sensitive internal system information derived using observed data. The amount of sensitive information a single observer can deduce is different from what a set of collaborative observers can deduce.

$^\ast$. From this point onwards, consider the parallel reconfigurable units in Figure 4 as passive loads. As a result, the only reconfigurable units in Figure 5 are $R_A$ and $R_C$. Figure 7 is the equivalent model.

The strategic placement of observers is significant since readings from certain observers might turn out to be redundant. The relationship between the deducibility and the observability is an important aspect of CPS security analysis.

[57] evaluated the minimum number of observers required to fully derive all $\mathcal{D}_H$ commands in pure series and pure parallel connected networks. A similar evaluation on a network with series and parallel connections is presented as follows. Such a network is termed a **mix connection network** here onwards.

5.1 **Deducing a Mix Connection Network with Minimum Number of Observers**

**Lemma 1. [Minimum Number of Observers for Mix Connection Networks]** A mix connected network with $\eta$ number of reconfigurable units and $\kappa$ number of junctions can be fully deduced with a minimum of $\eta - \kappa$ number of observers.

![Fig. 4. The General Form of a Mix Connected Network of Reconfigurable/Reprogrammable Units](image)

The uniqueness of $\mathcal{D}_L$ projections[57] is an extended concept of Sutherland’s Nondeducibility[28] definition. The general form of a mix connection network and the basic building block is shown in Figure 4. The basic building block is used to extend the general form into specific ones for a series of experiments. As an example, Figure 5 shows a network of five configurable units, after inserting one basic building block.
Table 1 is the corresponding $\mathcal{D}_L$ observation matrix for the network in Figure 5. A $\mathcal{D}_L$ observation matrix lists changes in observations for each $\mathcal{D}_H$ action. Here, an increase in a variable resistor (or an observable change) is denoted by a $\uparrow$ and decrease by a $\downarrow$.

Each row in Table 1 is an execution where column 1 is the trace ($\mathcal{D}_H$ command) and columns 2-6 is the projection ($\mathcal{D}_L$ observable changes). Here is an example.

$$\sigma = \{ R_E \uparrow, I_A \downarrow, I_B \uparrow, I_C \downarrow, I_D \uparrow, I_E \downarrow \}$$
$$\zeta(\sigma) = \{ R_E \uparrow \}$$
$$\rho(\sigma, \mathcal{D}_L) = \{ I_A \downarrow, I_B \uparrow, I_C \downarrow, I_D \uparrow, I_E \downarrow \}$$

The objective of $\mathcal{D}_L$ observers is to uncover $\mathcal{D}_H$ commands (secret settings) purely based on $\mathcal{D}_L$ observations. A single observer has a limited view of the system and cannot decide $\mathcal{D}_H$ commands using his observations. For example, the second column of Table 1 corresponds to observations of observer $x$ in Figure 5. From his point of view, an increase in current flow possible due to any of $R_A \uparrow, R_B \uparrow, R_C \uparrow$ or $R_D \uparrow$. Thus, observer $x$ cannot distinguish the exact command.

By collaborating, a set of $\mathcal{D}_L$ observers can build unique event sequences corresponding to each $\mathcal{D}_H$ change. Note that there is a unique projection for each of the 10 possible traces in Table 1. This means that a sufficient number of synchronized observers who are able to read each of the current changes of the system can fully deduce this network.

The network shown in Figure 5 can be fully deduced by solving the KCL equations, $I_A = I_B + I_C$ and $I_C = I_D + I_E$, for two junctions $X$ and $Y$. These two equations could be solved with just three readings. In the first equation, knowing only two values is sufficient to deduce the other value. With one additional value the second equation is also solvable. What this means is, a minimum of three observers can fully deduce the network shown in Figure 5.

By repeating this experiment with different number of basic building blocks, it was possible to establish a relationship between the number of variable resistors and the number of junctions in the network. This is listed in Table 2. As a consequence, the claim in Lemma 1 can be easily proven using this observation and mathematical induction.
since it shows a violation of \( D_H \) command confidentiality due to \( D_L \) observations. In terms of IFPs, this is a violation of Nondeducibility security.

### 6 Nondeducibility–Compensate Property for CPSs

**Theorem 1.** [\( P \)-Compensate Property for CPSs] A system of reconfigurable units with a multiplicity of non-deducible combinations has the \( P \)-compensate property

Proof: Consider the definition of nondeducibility in equation 7 which states that a system with multiple equivalent \( D_L \) projections for every execution preserves nondeducibility security. In other words, assume a system where for every \( D_H \) input action (similar to the \( D_H \) changes listed in in Table (1)) there are two (or more) equivalent \( D_L \) projections. Such a system preserves the confidentiality of \( D_H \) actions since \( D_L \) users are unable to deduce \( D_H \) traces based on \( D_L \) projections.

The existence of multiple \( D_L \) projections means that an \( E_S \) with an appropriate security automata as in Definition (5) has multiple compensating sequence possibilities for each \( D_H \) system change. Thus, at any point \( j \) in an execution \( \sigma \) where \( M_S \) identifies a certain IFP violation, \( E_S \) can cleanse \( \sigma \) and preserve the \( P \)-compensate property. This can be achieved by using the appropriate predicate \( \phi(\cdot) \).

#### 6.1 Finding Compensating Couples

Section 5.1 only considers a single change in one of the reconfigurable units at a time. However, consider the following two executions.

\[
\sigma_1 = \{ R_A \uparrow, I_A \downarrow, I_B \downarrow, I_C \downarrow, I_D \downarrow, I_E \downarrow \} \\
\sigma_2 = \{ R_B \uparrow, I_A \downarrow, I_B \downarrow, I_C \downarrow, I_D \uparrow, I_E \uparrow \}
\]

Note that, \( I_C, I_D \) and \( I_E \) show opposite changes for two different commands. Technically, there is a potential that these two commands acting together may cancel out certain \( D_L \) observations. To demonstrate this point, an experiment was conducted by simultaneously changing \( R_A \) and \( R_C \) values in the abstract model in Figure 5.

Figure 6 shows the net observable change in different current readings for this experiment. The horizontal plane in Figure 6 defines all combinations of \( R_A \) and \( R_C \) settings while the vertical plane plots the corresponding current reading for each of these combinations1.

The points at which the vertical axis crosses the horizontal axis (crossing points) in each subfigure of Figure 6 represent the combination of \( R_A' \) and \( R_C' \) values where the corresponding current reading is equivalent to the initial steady state value2. This is a significant factor because at these values, there is no change in external observations. This is an direct instantiation of Theorem 1 to Nondeducibility–compensate property; any two executions which include crossing points for a particular current reading have equivalent \( D_L \) projections.

In relation to the framework in Section 3, the crossing points correspond to a particular compensating couple. More crossing points for a particular current reading means that there are several ways of selecting a compensating couple (an action-correction pair), while keeping the corresponding current reading unchanged.

However, only a limited number of \( D_H \) events have corresponding compensating couples. This is denoted by number of crossing points in Figure 6 and is a physical constraint of the system itself. Thus, not all \( D_H \) actions have corresponding corrections. This behavior of systems is formally characteristic in Corollary 6.1 below.

**Corollary 1.** [Partially \( P \)-Compensatable Mix Connection Networks with Range Limited Changes] Mix connected networks of reconfigurable units with range limited changes are partially \( P \)-compensatable.

Proof: In Figures 6(a) and 6(b), there are no crossing points for \( I_B \) except for the default value. Thus, within the \( \pm 20\% \) operational bound, there are no other compensating cou-

---

1. For the simplicity of the analysis, the variable resistors and loads are initialized to a base value of 10 \( \Omega \) (with a \( \pm 2 \) \( \Omega \) operational bound)

2. Experimental values showed that the changes in \( I_E \) are identical to that of \( I_D \) thus, Figure 6(d) represent both \( I_D \) and \( I_E \)
ple although it might be possible to find compensating couples outside this limited range. For all practical purposes, this is a physical constraint of the system and such constraints sometimes limit the possibility of producing sufficient multiplicity for compensation.

In contrast, consider Figure 6(c) with a steady state configuration of $[R_A = 10\Omega, R_C = 10\Omega, I_C = 2A]$. The multiple crossing points in this figure depict that, there exists other values of $R_A$ and $R_C$ which can still maintain the initial $I_C$ value. A similar analysis follows for Figure 6(d). Thus, the partial ability of the system to produce compensating couples due to the operational limitations characterizes partially $P$–compensatability.

Also, the contrapositive of Lemma (1) implies that any number of external observers below the minimum requirement can only partially deduce the system. From a system design perspective, this can be achieved by obfuscating $D_L$ observations using compensating actions. The following corollary describes this feature in mix connection networks.

**Corollary 2.** [Partially $P$–Compensatable Mix Connection Networks] Mix connected networks of reconfigurable units are partially $P$–compensatable.

**Proof:** A particular compensating couple can obfuscate or remove a certain $D_L$ observation. Such an obfuscation will yield in no observable change to some $D_L$ observer. This directly affects a collaborative set of observers’ ability to build unique event sequences. However, only a certain number of $D_H$ actions have matching compensating actions, and the ones with no compensating actions will be de-
ducible at the $D_L$. This violates some particular property $P$. For this reason, a mix connected network of reconfigurable units is partially $P$–compensatable.

### 6.2 Contribution of the Event Compensation on CPS Security

Having multiple possible compensating couples allow several possible substitutions for a single steady state change. For a state machine abstraction, this allows executions to be extended in multiple possible paths. For example, $\varphi = <R_A \uparrow, R_C \downarrow>$ or $\bar{\varphi} = <R_C \uparrow, R_A \downarrow>$, as in Figure 6(c), can be used to nullify changes in $I_C$. With this, the system becomes nondeterministic for $D_L$ observers; absence of $D_L$ observations does not reflect absence of $D_H$ changes.

Even detected changes could be due to one of the several possible $D_H$ action couples ($\varphi$ or $\bar{\varphi}$ above). On top of that is the possibility of physical layer failures and interrupts. This is an important feature since it removes the uniqueness of $D_L$ observations. As a consequence, such a system preserves Nondeducibility of confidential $D_H$ actions[12, 57].

Equally important is the ability to recover the system from a possible failure. Having multiple possible compensating couples empowers $D_H$ administrators when committing to a $D_H$ command. Administrators can calculate corrections for each $D_H$ command and use a compensating automata to enforce proper execution.

### 7 Conclusions

This work presented the practical application of the Event Compensation concept for CPSs. The possibility of having multiple possible compensating couples further strengthens systems’ ability to withstand information flow security based security violations. The central concept in applying event compensation is to shield certain $D_L$ changes using two (or more) $D_H$ actions so that deducing $D_H$ actions at the $D_L$ domain is prevented.

However in some instances, certain $D_L$ changes would be inevitable as there might not exist any way of shielding $D_L$ changes within the operational bounds of the system (Refer Figure 6(a) and 6(b)). The authors identify this as a physical limitation of the system which at present cannot be incorporated to the proposed framework. Yet, there exists other $D_L$ changes which could be shielded, which might provide sufficient protection at least in partially compensatable systems.

Even the ability to compensate one $D_L$ observation may still prove vital in preventing information flow from $D_H$ to $D_L$. Such compensation removes some aspects of the domain knowledge required by $D_L$ observers to identify and distinguish $D_H$ changes. Added is the fact that the traditional security policies and mechanisms have already proven to be inefficient in preserving and providing system security. Thus, the authors believe the contribution of this work is a step towards finding new ways of securing modern CPs from many forms of emerging vulnerabilities.

### References


[31] S. Zdancewic and A. C. Myers, “Secure information flow and CPS,” in ESOP ’01:


**APPENDIX A**

**NOMENCLATURE**

Let \( \Sigma^\infty \) be the universe of all executions and \( \Sigma^* \subset \Sigma^\infty \) denote the set of system executions. Each execution of this set \( \sigma = < \sigma_0, \sigma_1, \ldots > \in \Sigma^* \) is composed of a sequence of global states \( \sigma_i \in \sigma \). The subscript \( i \) denotes the \( i \)th element of the sequence. A particular global state \( \sigma_i \) is a snapshot of the system at some instance in time, and consists of a set of system variables \( \mathcal{V} \). The universe of system actions is the composition of input and output actions, i.e., \( \Phi^\infty = \Phi \cup \hat{\Phi} \). A subset \( \Phi \subset \Phi^\infty \) called input actions has the potential to influence one or more system variables \( v_i \in \mathcal{V} \) and trigger state transitions. Upon a change to \( \mathcal{V} \), the system assumes a new state. Thus, an execution \( \sigma \in \Sigma^* \) can be represented as,

\[
\sigma : \sigma_0 \xrightarrow{\phi_1} \sigma_1 \ldots \sigma_{i-1} \xrightarrow{\phi_i} \sigma_i \ldots
\]

The first action \( \phi_0 \) of each sequence is the null element which initializes and leads the execution to the initial state \( \sigma_0 \). Associated with each execution \( \sigma = < \sigma_0, \sigma_1, \ldots > \) is a sequence of actions called a trace \( \phi = < \phi_1, \phi_2 \ldots \in \Phi^* \). \( \Phi^* \) represents the universe of all traces. Elements of \( \Phi \) can not change \( \mathcal{V} \) but can present a view of the system (effectively a view of \( \mathcal{V} \)) to a particular security domain. Thus, a trace \( \phi \) is not (necessarily) exclusively input actions; if \( \phi_k \in \bar{\Phi} \implies \sigma_i \xrightarrow{\phi_k} \sigma_i \).

**Definition 6. [System Executions]** A system execution \( \sigma \in \Sigma^* \) is a sequence of global states of the system. A (concurrent) system is a collection of system executions (\( \Sigma^* \)).

A property \( \Gamma \subseteq \Sigma^* \) is a finite subset of executions. The safety property[20] definition below is one such example.

**Definition 7 (safety property).** A property \( \Gamma \) is a safety property if and only if,

\[
\forall \sigma \in \Sigma^* \nexists \Gamma \implies (\exists \beta \in \Sigma^* : \sigma[\ldots j] \beta \notin \Gamma)
\]

Safety properties are prefix-closed; no prefix of a valid execution can violate the property and any violation has a distinctly identifiable point. An invalid execution cannot be extended beyond the violation point and the violation is not undone[53]. \( \sigma[\ldots j] \) is the prefix of execution \( \sigma \) up to step \( j \).

A closely related concept is the security policy. Technically, a security policy \( \mathcal{P} \) is a generalized form a property, defined over a characteristic predicate \( \hat{\phi} \). The membership of each element in \( \mathcal{P} \) is determined by \( \hat{\phi} \).

**Definition 8 (security policy).** A set of executions \( \Pi \subseteq \Sigma^\infty \) satisfies a security policy \( \mathcal{P} \) if and only if,

\[
\mathcal{P}(\Pi) \implies \forall \sigma \in \Pi : \hat{\phi}(\sigma)
\]

In other words, a security policy is a property when each element of the property satisfies the characteristic predicate of the policy[53].

As an example, a security policy may require that each execution to complete the \( j^{th} \) step before \( i \) time units. In terms of safety properties, executions are only rejected after some finite
number of steps. Mathematically this is stated as follows.

\[
\forall \sigma \in \Sigma^\infty : \neg \phi(\sigma) \Rightarrow (\exists j : \neg \phi(\sigma[\ldots j])) \quad (4)
\]

**Definition 9 (liveness property).** A property \( \Gamma \) is a liveness property if and only if,

\[
\forall \sigma \in \Sigma^* : \exists \beta \in \Sigma^\infty : \sigma \beta \in \Gamma
\]

Any finite system execution, irrespective of whether it violates the corresponding property or not, can be extended by appending an infinite or finite sequence resulting in an execution satisfying the liveness property. Informally, liveness means “nothing irremediably bad can happen” during an execution [53].

**Definition 10 (Noninterference Property).** Noninterference defines that the outputs observable at \( D_L \) stays the same even after removing all \( D_H \) actions. Formally, this is stated as,

\[
\forall \sigma \in \Sigma^* : \zeta(\sigma) = \phi : \rho(\phi, D_L) = \rho(\pi(\phi, D_H), D_L)
\]

(5)

If the above equality holds, the corresponding system is considered noninterference secure. Added, if the value of a public system variable depends on a private system variable, then for that particular system, noninterference does not hold[58]. In general, noninterference means that a variation of \( D_H \) inputs does not cause a variation in \( D_L \) outputs[9]. This property is considered one of the conceptually hardest to implement on most real world systems, out of known information flow properties.

**Definition 11 (Noninference Property).** Noninference on the other hand is a more general form of noninterference because it can be directly applied to nondeterministic systems [15]. Here, a \( D_L \) cannot deduce that progress has been made in the \( D_H \) computation [59]. For each valid trace of the system, if the resulting trace of purging \( D_H \) events is still valid, then the system is considered noninference secure. Thus, noninference is closed under \( \pi \). Formally, this can be stated as follows.

\[
\forall \sigma \in \Sigma^* : \zeta(\sigma) = \phi : \zeta^{-1}(\pi(\phi, D_H)) \in \Sigma^*
\]

(6)

In general terms, noninference means that for every \( D_L \) there must be a possible trace which yields the same projection in which no \( D_H \) events occur [59]. Noninference is equivalent to noninterference for deterministic systems for which \( D_H \) outputs cannot be generated without \( D_H \) inputs [15]. However, inserting \( D_H \) inputs can influence \( D_L \) outputs.

**Definition 12 (Nondeducibility Property).** Nondeducibility is probably the most relaxed property, out of the three information flow properties introduced above. Basically, what nondeducibility states is that, for each \( D_L \) projection there are more than one possible (\( D_H \)) traces. Thus, for a nondeducibility secure system, \( \rho \) is a surjective relation at \( D_L \). Formally,

\[
\forall \phi \in \Phi^* : \pi(\phi, D_L) = \chi \implies \exists \psi \in \Phi^* : \psi \neq \phi : \pi(\psi, D_L) = \chi
\]

(7)

Consider the operation of the target system as a (possibility nondeterministic) state automata \( M = (D, Q, \Phi, \delta, Q_0) \). \( Q \) is the set of automata states, \( \Phi \) is a set of input actions/events, \( \delta : Q \times \Phi \to 2^Q \) is the state transition function and \( Q_0 \subseteq Q \) is the set of initial states of the automata. \( D \) is a set of security domains/clearances traditionally, \( D_H \) and \( D_L \).

**Definition 13 (System Automata).** The system automata \( M \) is 5-tuples \((D, Q, \Phi, \delta, Q_0)\) where,

- \( Q \) is a set automaton states
- \( Q_0 \) is a set of initial states for the automaton \( Q_0 \subseteq Q \)
- \( \Phi \) is a set of input symbols
- \( \delta \) is the a state transition function \( \delta : Q \times \Phi \to 2^Q \)
- \( D \) is the set of security domains

Only a subset of actions \( \Phi_{\text{obs}} \in \Phi^\infty \) (irrespective of input/output distinction) can be observed at any given security domain. Observations are mapped to security domains through specific functions. These functions act as information extractors from the state space. The trace function \( \zeta : \Sigma^* \to \Phi^* \) produces the action sequence \( \phi \in \Phi^* \) associated with the execution \( \sigma \in \Sigma^* \). Consequently, the trace contains both “confidential” and “nonconfidential” actions where only the latter is supposed to be observed by \( D_L \). The projection function \( \rho : \Phi^* \times D \to \Phi_{\text{obs}} \cap \Phi^* \times D \) is a function which takes a trace and a security domain as
input and produces a possible output sequence observable by the elements of \( D^\delta \). Lastly, the purge function \( \pi : \Phi^* \times D \rightarrow \Phi^* \Phi_D^* \) removes actions corresponding to a particular security domain from a given trace.

### Appendix B

#### External Observation based Security Threats

Relating information flow to a particular physical system requires a semantic understanding of its operations. Properly mapping system semantics into a security framework is a fundamental challenge in CPS security analysis. Further, the practical applicability and the effectiveness of any security model is very sensitive to semantic knowledge base of the system. In this respect, this section formally shows how external observation(s) can leak sensitive internal system settings.

![Fig. 7. A Simple DC Circuit with Two Variable Resistors and Three Loads](image)

The Kolmogorov’s Superposition Theorem states that any multivariate function can be represented as a superposition of one-dimensional functions\cite{[60]}. This technique is used in circuit analysis to find unknown voltage and/or current changes (or voltage/current values), of a circuit by systematically replacing independent voltage and current sources with short circuit and open circuit connections respectively.

Consider the simple DC circuit model shown in Figure 7. This circuit includes two variable resistors \( R_A, R_B \) and three loads \( L_A, L_B \) and \( L_C \). Assume the source voltage \( V \) is constant and \( R_A \) and \( R_B \) set values are sensitive. However, voltage and current readings of the circuit are externally measurable.

The overall system characteristics are described using Kirchhoff’s Current Law(KCL) and Kirchhoff’s Voltage Law(KVL) as follows. From KCL,

\[
I_S = I_A + I_B + I_C
\]

From KVL,

\[
V = (I_A + I_B + I_C) \cdot R_A + (I_C + I_B) \cdot R_B + I_B \cdot L_B
\]

Using (9), \( R_A \) and \( R_B \) can be described as follows.

\[
R_A = \alpha \cdot V - \beta \cdot R_B - \gamma \cdot L_B
\]

\[
R_B = \lambda \cdot V - \mu \cdot R_A - \nu \cdot L_B
\]

where,

\[
\alpha = \frac{1}{(I_A+I_B+I_C)} \quad \lambda = \frac{1}{(I_B+I_C)}
\]

\[
\beta = \frac{(I_B+I_C)}{(I_A+I_B+I_C)} \quad \mu = \frac{(I_A+I_B+I_C)}{(I_B+I_C)}
\]

\[
\gamma = \frac{I_B}{(I_A+I_B+I_C)} \quad \nu = \frac{I_B}{(I_B+I_C)}
\]

At some given time \( i \), the global state of this circuit is \( \sigma_i = \{I_A, I_B, I_C, R_A, R_B\} \). Associated with each of these system variables is a security domain. External observations are mapped to \( D_L \) that is, \( \{I_A, I_B, I_C\} \subseteq D_L \). Technically, \( \{R_A, R_B\} \supseteq D_H \) that is, in this model, the settings of the variable resistors are considered sensitive.

Consider in two subsequent global states, \( R_A \) and \( R_B \) change to different values, one at a time. Correspondingly,

\[
R_A' = \alpha' \cdot V - \beta' \cdot R_B - \gamma' \cdot L_B
\]

\[
R_B' = \lambda' \cdot V - \mu' \cdot R_A - \nu' \cdot L_B
\]

describe the new values for \( R_A \) and \( R_B \). Let’s define at the end of these changes the global state is \( \sigma_j = \{I_A', I_B', I_C', R_A', R_B'\} \). The aggregate internal system change \( a_j \supseteq D_H \) is represented as \( \sigma_i \xrightarrow{a_j} \sigma_j \).

The amount of change in \( R_A \) and \( R_B \), \( \delta_A = R_A - R_A' \) and \( \delta_B = R_B - R_B' \), is represented in equations (14) and (15).

From (10) - (12),

\[
\delta_A = (\alpha - \alpha') \cdot V - (\beta - \beta') \cdot R_B - (\gamma - \gamma') \cdot L_B
\]
From (11) - (13),
\[ \delta_B = (\lambda - \lambda') \cdot V - (\mu - \mu') \cdot R_A - (\nu - \nu') \cdot L_B \] (15)

Consider a case where \( \delta_A + \delta_B = 0 \). That is, \( R'_A \) and \( R'_B \) are selected in a way that the effective change of the two actions cancel out each other. Thus, the superposition of equations (14) and (15), given in (16), is similar to that of (9)*.

Taking (14) + (15),
\[ 0 = \Xi \cdot V - \bar{\mu} \cdot R_A - \bar{\beta} \cdot R_B - \Gamma \cdot L_B \] (16)

By solving (9) with (16), it is now possible to reveal the original settings of both \( R_A \) and \( R_B \). Thus, \( a_j \) results in an unintentional information flow and an information flow security violation by allowing \( D_L \) deduce \( D_H \) settings using \( D_L \) observations. Thus, the analysis shows how a set of coordinated external observations can be used to deduce sensitive internal system settings.

* Here, \( \Xi = (\alpha - \alpha' + \lambda - \lambda'), \bar{\mu} = (\mu - \mu'), \bar{\beta} = (\beta - \beta') \) and \( \Gamma = (\gamma - \gamma' + \nu - \nu') \).