A Formal Methodology for Procedural Security Assessment

Komminist Weldemariam and Adolfo Villafiorita
Center For Information Technology
Fondazione Bruno Kessler
Trento 38100, Italy
Email: (sisai,adolfo)@fbk.eu

***Abstract*—Formal analysis techniques can deliver important support during ICT-based innovation (or redesign) efforts in e-government services. This paper discusses a formal methodology for assessing the procedural security of an organization. We do so by explicitly reasoning on critical information flow named assets flows. With this it is possible to understand how critical assets are modified in unlawful manner, which can trigger security and privacy violations, thereby (automatically) detecting security weaknesses within an organization under evaluation.**

***Keywords*-procedures; security assessment; modeling and analysis; formal methods.**

I. INTRODUCTION

Currently, several organizations and enterprises across many countries are evaluating and introducing ICT-based solutions with the aim of improving and delivering quality (public) services. For instance, very recently the Italian Government launched a *certified email* (in Italian "Posta Certificat@") service for its citizens [1]. This service enables citizens to legally communicate with the public administration or institutions through a certified e-mail system, with the aim of achieving a paperless bureaucracy, thereby reducing time, energy and money waste for institutions and citizens. In this setting, a significant portion of asset that can contain information and data, much of which is sensitive (e.g., the certified email account), is managed and controlled by introducing organizational regulations and procedures in order to enhance the security and privacy of (non-) digital assets. Such sensitive assets can also be used in business exchanges among (in the above scenario, e.g., citizens with PA) inter-business collaborations and (virtual) organizations with a certain understanding on the different roles the participants play; and, at the same time by including assumptions on their correct and incorrect behaviors, and their rights, duties, and obligations in order to avoid misunderstanding and ambiguities in such business relationships. Not to mention, these assets and their interrelations can also contain inherent weaknesses or vulnerabilities [2], [3], [4] and which are of two types.

The first, although out of scope, is technical vulnerabilities (for which a number of techniques exist), a hardware or software weakness, or design deficiency, that leaves a system open to attack, thereby resulting in unacceptable risk of information compromise, information alteration, or service denial [5]. The second one is procedural vulnerabilities, weaknesses within an organization due to the lack of proper implementation of security policies related to managerial or procedural deficiency, resulting in compromising the security and privacy of the organization as well as individuals within the organization [3], [6]. However, techniques that can help to model and assess such vulnerabilities are absent or very unsatisfactory, and thus procedural security analysis.

This paper complements our previous work [7] by showing how formal techniques can be used for the modeling and analysis of procedures in an organization under evaluation. We do so by presenting a formal framework for representing organization system as assets-flows. The concepts of our framework (roles and actors, actions and processes, responsibilities and constraints) can allow (business or security) analysts to capture organization model in a way that is both intuitive and mathematically formal. The use of a formal technique can allow us to determine whether a given information stipulates certain (procedural) security properties —e.g., that the responsibilities assigned to roles are fulfilled and that the constraints are maintained. With this it is possible to understand how critical assets are modified in unlawful manner within an organization. Thus, we believe that, this is important for both developed and developing nations where the development and deployment of ICT-based solutions in several areas of security-critical e-government services or applications are in progress.

The next section briefly describes the background material for procedural security analysis. Its formal model is presented in Section III. Section IV discusses the mapping of such model into executable specifications. Finally, conclusion and future work are discussed in Section V.

II. PROCEDURAL SECURITY ANALYSIS

A typical approach for inherent information flow within an organization is access control (see, e.g., in [8]). That is the accesses of objects by subjects, restricted by specific access permissions —e.g., by assigning *read* and *write* permissions to some sensitive assets. Moreover, approaches such as based on formal techniques and methodologies have been used to model (system) processes [9], [10], [11], [12]. These works mainly concentrate on constructing business process

 ISBN: 978-1-61208-116-8

models with correct creation and termination of artifacts during their lifecycle, by providing some supports to perform automated analysis. Accordingly, some of these approaches hint integrations with existing formal methods' tools. However, they hardly concentrate on security analysis especially on analyzing the security of organizational or procedural weaknesses. As noted in [13], [14], risks and attacks not only depend upon the security levels the new systems offer, but also occur by circumventing on the procedures and controls regulating the way in which the systems are operated. For example, what happen if one can get a fake certified email by circumventing the procedures required for request and delivery of such services. Obviously, this could lead to maliciously communicate with the PA thereby accessing public services accordingly. Therefore, it is important to analyze the security of such procedures that grant accesses to sensitive data and services, and thus procedural security analysis.

To be able to conduct a security analysis, at least enough information must be present to deal with assets and global threats, i.e. at an abstract level, subjects and objects in the procedures and system must be identifiable. The starting point of the procedural security analysis methodology is an initial model, describing a coarse procedure or system process without security-related aspects. This model describe the procedure or procedures to be analyzed in a systematic way. Secondly, we extend this model with attack information, meaning that we generate an extended model from the model defined in the previous step. In the extended model, thus, not only assets are modified according to what the procedures define but they can also be transformed by the (random) execution of one or more threat actions. Thirdly, the encoding of the asset-flows in terms of executable specifications is performed using formal language. Fourth, we specify security properties for formal analysis. More specifically, we specify the (un-)desired (procedural) security properties —namely, the security goals that have to be satisfied (unsatisfied), are then encoded using mathematical formula, which in turn together with the model are given as input to the analysis tool. Thus, we perform security verification and assess the results. Security verification is the verification that the global security requirements are fulfilled with respect to the threat scenario. If the result of the security verification is that a particular security requirement is violated, there is a corresponding attack on the procedures and consequently on the system. Otherwise, the procedure is secure given the assumptions included in the model. This is obviously via the model checker, i.e., if a property is proved to be false, the analysis tool generates a counterexample which opens up further discussion.

III. A Formal Model of Procedural Security

Figure 1 shows a high-level representation of the information and the behavioral (i.e., the lifecycle) models of assets. The perspective shown in the figure offers three complementary views: workflow, assets class, and state machine diagram views. In the *workflow diagram* view, workflow activity sequences are defined. The *state machine* view describes the behavior of an asset in terms of a transition system in which transitions are enabled due to explicit execution of workflow activities. The activities in the workflow are transformation functions that influence the behaviors of the assets. A finite state transition diagram for each feature of an asset constitutes the global state machine for that asset.

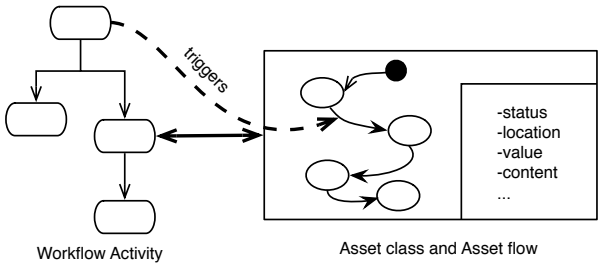

Figure 1. An asset-flow view of a business process model

For instance, Figure 2 shows a simple example of asset-flow model for asset instance **A** with three states $[s_1]$, $[s_2]$, and $[s_3]$. The corresponding finite state machine, therefore, will possibly have three sequential states each of which corresponds to **A**'s current features values.

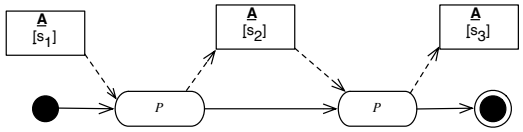

Figure 2. Example of a single instance Asset-flow model in three states.

A. Formalization of the Model Elements

We assume the following notations and their definitions:

- $\mathcal{T}_p$ be a set of primitive types, such as bounded integer and boolean;
- $\mathcal{C}$ be a set of asset classes (*names*);
- $\mathcal{A}$ be a set of attributes (*names*);
- $\mathcal{ID}_C$ be a set of identifiers that describes the identifiers for each asset class $C \in \mathcal{C}$;
- $\mathcal{S}$ is a set of assets states, where each $s \in \mathcal{S}$ is a sort of truth assignment over the variables values.

Note that all the above sets are finite, which is essential for the formal verification process such as by using model checkers.

A *type* $\mathcal{T}$ is an element of the primitive types $\mathcal{T}_p$ and the class identifiers C; namely, $\mathcal{T} = \mathcal{T}_p \cup \mathcal{C}$ (we assume that $\mathcal{T}_p$ and $\mathcal{C}$ are disjoint).

Definition *3.1:* An asset class signature is a triple $\langle C, A, \psi \rangle$ where $C \in \mathcal{C}$, $A \in \mathcal{A}$ is a a set of attributes for the asset class C, and $\psi : \mathcal{A} \rightarrow \mathcal{T}$ is a total function that maps each attribute of an asset into its corresponding type.

Without loss of generality, we assume a fixed interpretation domain associated to each $\mathcal{T}$ type. That is the *domain* of each type $t \in \mathcal{T}$, denoted $\mathcal{D}^t$, is defined in the following way: if $t \in \mathcal{T}_p$ is a primitive type, then the domain $\mathcal{D}^t$ is some known set of $values$ of type (e.g., integer or boolean); if $t \in \mathcal{C}$ an identifier type, then $\mathcal{D}^t$ defines existing instances of an asset class identifier for t (i.e., $\mathcal{D}^t = \mathcal{ID}_t$). We require all variables must have their corresponding *values* all along their life. For undefined location and unassigned content of an asset, we use an *undefined* and a *null* constant values respectively. The interpretation is that the location is not known and the content value is not either assigned yet or reset to contain null.

Definition *3.2:* An asset instance is a triple $\langle ID_C, C, \phi \rangle$, where $ID_C \in \mathcal{ID}$ is a class identifier and ϕ a partial function, given an instance of a class C, that assigns each variable $a \in A_C$ of type $t \in \mathcal{T}$ a value in D^t (i.e., $\phi(a) = D^t(\psi(a))$).

An asset can have multiple instances. We denote the set of asset instances by $\vec{\mathcal{O}}_{C,C\in\mathcal{C}}$ and $\vec{\mathcal{O}}$ for all instances over Σ. However, in this work, we mainly focus on how a single asset instance $I \in \vec{\mathcal{O}}_C$ can evolve from some initial state through other states. The set of *variable-value* pairs for I defines the *state* of a given asset. The *state* of an asset is, therefore, the current situation called *"snapshot"* of an asset instance I and its value is the truth assignment over the variables. An asset is *initial*, if all the variables are in their initial state and ϕ is undefined for some attributes and *final*, if all the variables values do not change anymore.

The information provided previously is the basis for the formalization, namely how we represent the assets structure and workflows to arrive at what we call executable models. The formalization allows the model to be more amenable to formal analysis, hence it shifts the focus to dynamic aspects of the assets.

1) Defining Workflow formally: As noted earlier, the initial values for the variables of an asset can be assigned at the time of the instance creation or otherwise assigned by an analyst. However, only due to the execution of a workflow activity over these variables can possibly change the initial configuration of the asset.

Roughly speaking, a workflow activity is described by input assets, preconditions, and effects of the activity over the assets (a similar interpretation can be found in [9], [11], [15]). The effect of a workflow activity is regarded as a change in state of the input assets. Not all assets change their states thought, since it is not always the case that an execution of a workflow activity enables state transition to all the input assets (e.g., reading the content of a password does not change its state).

For each executable workflow activity we specify which actors participate in the workflow with predefined privileges or responsibilities or both. These information not only allow to describe who does what during the execution of an activity, but, more importantly in the context of organizational security modeling and analysis, who manages what data and with what privileges. Such information are static, namely they are known before executing a workflow (e.g., as described in a legal document or contractual agreement between two entities) and are encoded in our model to describe a workflow scenario. We, therefore, use these information along with the activities to describe a workflow model as a deterministic finite state machine in which the states are constructed by a set of activities, and the transitions are described by the current state and a matching condition over the accessory information.

Formally, we define the workflow model as follows.

Definition *3.3 ($\mathcal{W}$):* A workflow model is a quadruple $\langle P, s_0, s_f, C, \Delta \rangle$ where

- P is a set of activities or processes (*names*);
- $s_0, s_f \in P$ are initial and final activities of the workflow respectively;
- C is guard expression over accessory information, and;
- $\Delta \subseteq P \times C \times P$ is a transition relation between a current activity and its successor activities in which a transition is labelled with a condition over accessory information.

The above definition is meant to express the fact that there exist a set of activities within a particular workflow, that describes a procedure under analysis, in which by knowing the current state of the workflow, and if a condition is met, it should be obvious to determine the next state of the workflow. We call an instance of a workflow model, a program counter *"pc"* that contains the value of the current state (i.e., the active activity) in the workflow. There is one program counter *"pc"* for each workflow model at run time. In actual business process or workflow specification, in fact, it is possible to have multiple activities that can run in synchronous or asynchronous mode. We focus on sequential execution of a workflow in this work.

2) Defining Asset-flows formally: The state of an asset is specified by the assignments of values to variables (or simply valuations), which allows to describe the evolution of an asset. The evolution is expressed by the sequence of states through which an asset undergoes during the execution of a process. Since the state of an asset is described by the valuations over its variables, therefore, it makes sense to encode the state of each variable as a finite state machine. The workflow instances, along with some matching conditions, define transitions for modeling the lifecycle of the assets. Thus, an asset-flow can easily be modeled using a transition system that facilitate formal analysis.

Definition *3.4:* An asset-flow model (AFM) is a 5-tuple $\langle AS, I, \mathcal{W}_\pi, C_\pi, \Delta_\pi \rangle$ where

 ISBN: 978-1-61208-116-8

- $AS \in \mathcal{S}$ is a finite set of assets' (instances) states;
- $I \subseteq AS$ is an initial states of the assets;
- $\mathcal{W}_\pi$ is a set of workflow instances;
- C_π is a set of conditions constructed over the attributes representing the matching construct as a guard, that specify the condition must meet for the state to be changed, along with the current activity;
- $\Delta_\pi \subseteq AS \times \mathcal{W}_\pi \times C_\pi \times AS$ is a transition relation between a current state of an asset and its successor states in which a transition is labelled with an activity and a condition.

A collection of individual AFM constitutes *assets-flow* models, and we represent it by $\mathcal{M}$. Therefore, $\mathcal{M}$ is regarded as the global configuration of the domain of interest, namely the procedures under analysis. The semantic of the global configuration $\mathcal{M}$ can be interpreted in the following way. Each $m \in \mathcal{M}$ is regarded as an abstract state machine, which has three major components: a workflow activity sequence (possibly maintained in a queue), a workflow activity dispatcher, and an activity processor. Workflow activities are added to the end of the activity queue. The activity dispatcher chooses, dequeues, and provides the next *"pc"* (i.e., an activity) to the activity processor. Each *"pc"* is then used as a *transformation* function that can possibly change the state of an asset by modifying or changing one or more variables values of the asset. One state machine per feature variable encodes the lifecycle of that state variable. A set of such state machines constitutes the global state machine for the corresponding asset instance. By defining a semantic for the state machines corresponding to each feature of an asset and linking it with $m \in \mathcal{M}$, therefore, we have implicitly defined how $\mathcal{M}$ behaves.

B. Model Extension

In order to analyze what are the possible attacks of a given (set of) procedures, we need to encode asset threats in the nominal model and generate the extended model for $\mathcal{M}$. Structurally, in fact, there is no difference between $\mathcal{M}$ and the extended model. However, the main difference lies on the assets state *set* and on the transitions specification. This means that, the extended model possibly will have more states than the other due to the execution of threat-actions that can change the state of an asset into an undesired one. On the transitions side, on the other hand, the definition did specify the fact that transitions are triggered only by nominal workflow activities. We need to incorporate in the extended model the fact that an asset could be in any possible states and that such states can also be changed by the execution of malicious processes.

However, it is pretty straightforward from the definition we gave and by extending the definition of the workflow model to include all the malicious processes that an adversary might execute. Thus, in extended model, assets are not only manipulated according to what should happen in the nominal case, but can also be transformed by the execution of one or more assets threat-actions.

IV. Encoding using Formal Language

Assets-flow models $\mathcal{M}$ can become executable specification to allow formal analysis through verification tools on their evolution, including their malicious evolution due to threat-actions. Our aim here is to represent the model $\mathcal{M}$ into executable specification using NuSMV input language [16]. As noted in [16], the NuSMV semantic is based on a state-based formalism in which the behavior is defined by Kripke transition systems. However, the definition we gave for $\mathcal{M}$ is an action-based formalism in which the behavior is defined by (a sort of) labelled transition systems. Thus, we need to rearrange the previous definition to align with the semantic of Kripke structure so that the encoding of NuSMV specifications can be tackled.

Definition *4.1:* Let APs are set of atomic propositions ranged over some boolean expressions on the valuations of the variables. An asset flow model (AFM) is a Kripke structure over a set of atomic propositions AP defined by a quadruple $\langle AS_K, I_K, \Delta_K, \mathcal{L}_K \rangle$ where

- AS_K is a finite set of assets (instances) states;
- $I_K \subseteq AS_K$ is set of initial states;
- $\Delta_K \subseteq AS_K \times AS_K$ is a transition relation between a current state of an asset and its successor states;
- $\mathcal{L}_K : AS_K \rightarrow 2^{AP}$ is the labeling function which returns the set of atomic propositions which hold in a state.

The encoding of $\mathcal{M}$ in the NuSMV input language can be treated as a problem of defining a mapping between the two structures, i.e., between the structure specifying the model $\mathcal{M}$ and the Kripke structure. More specifically, the following encoding rules are defined to map $\mathcal{M}$ into the NuSMV counterpart.

Rule 1: The workflow model is encoded in NuSMV as a special module, and each workflow activity $p_i \in P$ for $i = 1, \ldots, n$ representing the domain activities (i.e., processes) in $\mathcal{W}$ are encoded in the NuSMV input language as a scalar variable program counter ($\mathtt{pc}$) in which p_i are its symbolic values.

In order to determine the state transition of the program counter, we introduce some predicates (see Table I). They are mainly associated with the accessary information, such as actor-role and actor-activity assignments. The table also shows the corresponding state variables in NuSMV input language.

Rule 2: The accessary information are encoded in the NuSMV input language within the $Workflow$ module in the following way (see also Table I):

- For each actor-role assignment, we introduce a variable assign_a_r. assign_a_r is true iff the predicate AssignR(a,r) is true for an actor $a \in Actor$ and a role $r \in Role$;

Table I
ACCESSORY INFORMATION AS PREDICATES.

Predicate	Meaning	NuSMV variable
AssignR(a,r)	assignment of actor $a \in Actor$ to role $r \in Role$	assign_a_r
AssignA(a,p)	assignment of actor $a \in Actor$ to an activity $p \in P$	assign_a_p
r_Active_for_a	role $r \in Role$ is active for actor $a \in A$	activefor_a_r
ExecA(a,p)	actor $a \in A$ executes an activity $p \in P$	exec_a_p

- For each actor-process assignment, we introduce a variable assign_a_p. assign_a_p is true iff the predicate AssignA(a,r) is true for an actor $a \in Actor$ and an activity $p \in P$;
- For each role activation r_Active_for_a, we define a state variable Activefor_a_r;
- Similarly, we define a variable Exec_a_p for every actor performing an activity, i.e., iff ExecA(a,p) is true.

Rule 2 defines accessary information for the transition relation of `pc` state variable. Notice that activities can only be executed if the activity instance in question is assigned to an actor —i.e., ExecA(a,p) $\Rightarrow$ AssignA(a,p). Moreover, a group of actors can perform the same activity, as discussed in the previous chapter.

Rule 3: For each asset instance in $\vec{\mathcal{O}}$, a NuSMV module is defined:

```
MODULE ASSET_NAME  (...)
```

Rule 4: An asset with no content in $\mathcal{M}_1$ is mapped to a symbolic value *"null"* in NuSMV. Similarly, an asset whose current location is not known or unspecified in $\mathcal{M}_1$ is mapped to a symbolic value *"unspecified"* in NuSMV.

Rule 5: The $location$, representing all the possible places of an asset, is encoded in the NuSMV input language as scalar variables `loc` in which loc_i for $i = 1, \ldots, n$ and "$undefined''$ are its symbolic values. The $content$, representing all the contents of an asset at a particular point of time, is encoded in the NuSMV input language as `content` in which $content_i$ for $i = 1, \ldots, n$ and "$null''$ are its symbolic values. The $value$, representing all security risk values for an asset, is encoded in NuSMV input language as `value` in which $noValue$, low, $high$ and $critical$ are its symbolic values. Finally, each domain specific `property` of an asset in an asset-flow model is encoded as a boolean value in NuSMV.

Rule 3 states that a module is defined for each asset (instance) in $\mathcal{M}$. In Rule 5, whereas each feature of the asset is defined as a state variable within the asset module specification. An *unknown* `location` and a *null* `value` are both encoded by symbolic values as defined by Rule 4.

Rule 6: The transition specification for each state variable is encoded by the current value of the program counter and some boolean expressions over the current state of the asset.

The above rule (i.e., Rule 6) encodes the transition specifications. The transition from one asset state to the next is determined by the current value of the `pc` and some condition over the current state of the asset instance.

Since all the above rules are related to the encoding of $\mathcal{M}$, we need to provide additional rule for encoding the extended model. The model extension corresponds to proving an extension in the NuSMV model with one or more applicable attack-actions. That is, a specification of how the assets can be in undesired states. This can be done by associating threat-actions with variables defined inside the module per asset instance. Moreover, the `Workflow` module should also need to be extended in order to include the malicious process executions. In particular, the model extension can be done by using the following strategies:

- by defining a scalar state variable to encode all the possible malicious process within the `Workflow` module. Therefore, the program counter does not only have values from nominal workflow activities but also from possible set of malicious workflow activities;
- by defining a transition specification for each activation of a threat-action on asset instance under the corresponding asset module in NuSMV, where the malicious activity is in place for enabling the transition;
- by defining boolean variable to monitor the execution of the corresponding threat-action. This variable will be true iff when the corresponding threat-action takes place;
- by introducing a scalar value "garbage" for the content state variable related to the introduction of malicious asset and a boolean variable. This variable will be true iff a predicate associated with the action (e.g., MAsset(t)) is true by a threat-action, say t.

The above strategies facilitate the task of model extension, by adding a number of boolean appendage variables that are needed to capture the malicious asset flows and the execution of threat actions to form the extended model specification in NuSMV input language. In this way, therefore, the model extension is performed for each applicable threat-action against the normal flow of assets. At this point, we have the model representing the assets-flows, which is ready to supply for the analysis tool. Before the analysis, however, we need to describe the security properties we intend to check against the model using standard LTL/CTL logic. Once all the properties of interest are specified with respect to the analysis goals, the next activities are formal verification and analysis of the results. Security verification is the verification that the global security requirements are fulfilled with respect to the threat scenario. If the result of the

 ISBN: 978-1-61208-116-8

security verification is that a particular security requirement is violated, there is a corresponding attack on the procedures and consequently on the system. Otherwise, the procedure is secure given the assumptions included in the model. This is obviously via the model checker, i.e., if a property is proved to be false, the analysis tool generates a counterexample which opens up further discussion.

V. Conclusion and Future Work

We have described a framework where procedurally rich systems in model-based assessment drive the construction of models by extending the usage scenario of procedural security analysis. The presented approach is aimed at basic security aspects relevant for organizational-oriented processes, providing guidelines to analyst performing procedural security analysis based on explicit reasoning on assets-flows. The approach can be used to analyze and evaluate the impact of threats, and consequently to come out with a set of (security) procedural requirements. Thus, an organization or enterprise can apply the approach to assess their procedural security posture prior to introduce ICT-based solutions. Here, assurance is not implied by the trust in the model but follows from the formal analysis of the model. The analysis is based on a set of formal security requirements and provides formal proofs for use as countermeasures (i.e., evidence).

The work described here is still in progress, and we are currently completing the theoretical framework of the approach. We admit that this work clearly lacks a working case study, illustrating a proof of concept of the presented approach. Moreover, the implementation of the approach in terms of a tool is not discussed. However, we are currently defining a model-based verification approach using UML. More specifically, we reuse existing formal semantics for UML activity diagrams specifying workflow models that correspond to the asset-flows semantics discussed in this paper. The semantics will be translated to NuSMV model based on meta-model transformations. To translate a UML activity diagram model into the NuSMV counterpart, we use an intermediate model called activity hyper-edge model which abstracts the activity diagram, specifically according to the semantics of the asset-flow model. The definition of a set of generic library of attack models corresponding to threat-actions is part of the future work.

References

[1] Italian Ministry of Public Affairs and Innovation, "It: Launch of a certified email system to communicate with the public administration," http://www.innovazionepa.gov.it/lazione-del-ministro/iniziative-e-sperimentazioni/sperimentazione-pec/pec-primo-piano.aspx, October 2010.

[2] Federal Agency for Security in Information Technology, "IT Baseline Protection Manual," http://www.iwar.org.uk/comsec/resources/standards/germany/itbpm.pdf., October 2000. Last accessed on Feb. 2011.

[3] Common Criteria, "Common Criteria for Information Technology Security Evaluation," http://www.commoncriteriaportal.org/, 2007. Last accessed on Nov. 2010.

[4] S. E. Parkin, A. van Moorsel, and R. Coles, "An Information Security Ontology Incorporating Human-Behavioural Implications," in *SIN '09*. ACM, 2009, pp. 46–55.

[5] M. Bishop, *Computer Security Art and Science*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[6] B. S. Institution, "BS ISO/IEC 27002:2005 — Information Technology —Security Techniques —Code of Practice for Information Security Management," http://www.iso27001security.com/html/27002.html, 2005. Last accessed on Feb. 2010.

[7] K. Weldemariam and A. Villafiorita, "Formal Procedural Security Modeling and Analysis," in *Proceedings the International Conference on Risks and Security of Internet and Systems*, ser. CRiSiS '08. IEEE, 2008, pp. 249–254.

[8] Ravi S. Sandhu and Edward J. Coyne and Hal L. Feinstein and Charles E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, 1996.

[9] M. Koubarakis and D. Plexousakis, "A Formal Model for Business Process Modeling and Design," in *CAiSE*, ser. LNCS, Benkt Wangler and Lars Bergman, Ed. Springer, 2000, pp. 142–156.

[10] R. Eshuis, "Symbolic Model Checking of UML Activity Diagrams," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 1, pp. 1–38, 2006.

[11] K. Bhattacharya, C. E. Gerede, R. Hull, R. Liu, and J. Su, "Towards Formal Analysis of Artifact-Centric Business Process Models," in *Proceedings of the 5th international conference on Business process management*, ser. BPM '07, vol. 4714. Springer, 2007, pp. 288–304.

[12] A. Deutsch, R. Hull, F. Patrizi, and V. Vianu, "Automatic Verification of Data-Centric Business Processes," in *ICDT '09*. ACM, 2009, pp. 252–267.

[13] A. Xenakis and A. Macintosh, "Procedural Security Analysis of Electronic Voting," in *Proceedings of the 6th international conference on Electronic commerce*, ser. ICEC '04. ACM Press, 2004, pp. 541–546.

[14] K. Weldemariam, "Using Formal Methods for Building More Secure and Reliable e-voting Systems," Ph.D. dissertation, University of Trento, Via Sommarive 14, March 2010.

[15] C. E. Gerede and J. Su, "Specification and Verification of Artifact Behaviors in Business Process Models," in *ICSOC*, ser. LNCS, vol. 4749. Springer, 2007, pp. 181–192.

[16] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An Open Source Tool for Symbolic Model Checking," in *CAV'02*, ser. LNCS. Springer, January 2002, pp. 241–268.

 ISBN: 978-1-61208-116-8