Proceedings

6th Layered Assurance Workshop

LAW 2012

Proceedings

6th Layered Assurance Workshop

Orlando, Florida, USA
3–4 December 2012

Sponsored by
Applied Computer Security Associates

Editorial production by Christoph Schuba

Table of Contents

Technical Program

Message from the Workshop Chair

The mission of LAW continues: to stimulate pre-competitive research into the foundations of layered and compositional assurance, to attract researchers to the hard problems still challenging practitioners seeking to achieve improved product and system assurance more affordably, and to provide a forum for the sharing of the problems and promising results among practitioners and researchers. We are once again fortunate to have a program with a wide range of applications and research, and will once again have the opportunity to engage in a relaxed but lively discussion among researchers and practitioners to explore the application of research results and to bring a clearer understanding of the work yet to be done.

I am pleased to welcome the participants in the 6th Layered Assurance Workshop. The LAW organizers strive each year to better serve you and to ensure that your investment in attending is worthwhile. To that end we invite leading figures in the diverse fields related to the research and practice of assurance to give LAW's keynote and invited talks. We bring the activities of relevant advanced research projects to your attention, and we provide you an opportunity to share your work as contributed papers and works-in-progress with other participants, with ample opportunities for interchange and to make contacts that you can continue to utilize throughout the coming years. The program you will enjoy this year would not have been possible without the energetic contributions of Peter Neumann and Gabriela Ciocarlie, and I hope you will join me in thanking them.

Best wishes for an enjoyable and productive Layered Assurance Workshop.

Rance J. DeLong, LAW 2012 Workshop Chair

Message from the Program Chair

It is my great pleasure to welcome you to the 6th Layered Assurance Workshop, hosted with ACSAC in Orlando Florida. A total of 8 research and 3 work-in-progress papers were accepted this year. I would like to thank all authors for their high-quality submissions and I am looking forward to very engaging discussions around the submitted topics.

It was a pleasure and privilege to work with the LAW Program Committee (PC) members and I thank them for their support, which in many cases has been continuous since the creation of LAW. Each paper received at least two reviews and two papers were shepherded in the process. Christoph Schuba, the LAW Proceedings Chair, has done a tremendous job collecting all the camera ready papers and putting them together as the first official proceedings for LAW. Thank you, Christoph!

Beyond the technical program in these proceedings, the LAW 2012 has even more to offer: exceptional invited talks and panels. I cannot thank enough Rance DeLong, the LAW Chair, and Peter G. Neumann, the Law Panel Chair, for bringing together the very best contributors and supporters of assured systems. We are going to have a blast. Last but not least, I would like to extend my thanks to Robert H'obbes' Zakon for all his help with the workshop. It takes 2 seconds to get an answer back from Robert when in need.

I hope you enjoy the workshop!

Gabriela F. Ciocarlie, LAW 2012 Program Chair

Organizing Committee

Rance J. DeLong, Workshop Chair
Gabriela F. Ciocarlie, Program Chair
Peter G. Neumann, Panel Chair
Christoph Schuba, Proceedings Chair

Program Committee

Gabriela F. Ciocarlie, SRI International (Program Chair)
Sean Barnum, The MITRE Corporation
Rance J. DeLong, Santa Clara University
Nick Mansourov, KDM Analytics
Peter G. Neumann, SRI International
Olin Sibert, Oxford Systems
Gordon Uchenick, Coverity, Inc.

Using Architecture to Reason About Information Security

Stephen Chong
Harvard University
Cambridge, MA, USA
chong@seas.harvard.edu

Ron van der Meyden
The University of New South Wales
Sydney, New South Wales, Australia
meyden@cse.unsw.edu.au

ABSTRACT

We demonstrate, by a number of examples, that information-flow security properties can be proved at a level of abstraction that describes only the causal structure of a system and local properties of trusted components. We specify these architectural descriptions of systems using a generalization of intransitive noninterference policies that admit the ability to filter information passed between communicating domains. We also show that in a concrete setting where the causal structure is enforced by access control, a static check of the access control setting plus local verification of the trusted components is sufficient to prove that a generalized intransitive noninterference policy is satisfied.

1. INTRODUCTION

System architectures are high-level designs that describe the overall structure of a system in terms of its components and their interactions. Proposals for architectural modeling languages (e.g., AADL and Acme [Garlan et al., 2000]) vary with respect to their level of detail and contents, but at the most abstract level, architectures specify the *causal structure of a system*.

The MILS (Multiple Independent Levels of Security and Safety) initiative [Alves-Foss et al., 2006; Vanfleet et al., 2005; Boettcher et al., 2008] of the US Air Force proposes to use architecture as a key part of the assurance case for high-assurance systems. The details of the MILS vision are still under development but, as articulated by Boettcher et al. [2008], it encompasses a 2-level design process, consisting of a policy level and a resource sharing level.

At the policy level, the system is described by an architecture in the form of a graph, in which vertices correspond to components and the edges specify permitted communication between components. In this respect, the architecture is like an *intransitive noninterference* security policy [Haigh and Young, 1987; Rushby, 1992; van der Meyden, 2007]. At the policy level, one might also specify which components are trusted, and the local policies that these components are trusted to enforce. According to the MILS vision, building a system according to the architecture, by composing components that satisfy their local policies, should result in *global* security and safety properties for the system.

At the resource sharing level, MILS envisages the use of a range of infrastructural mechanisms to ensure that the architectural information-flow policy is enforced despite components sharing resources such as processors, file systems, and network links. These mechanisms might include physical isolation, separation kernels, periods processing, cryptography and separating network infrastructure. It is intended that this infrastructure will be developed to a high level of assurance, so that a systems assurance case can be obtained by the composition of the assurance cases for trusted components and systems infrastructure. It is hoped this will enable a COTS-like market for infrastructural mechanisms and trusted components.

The key contribution of this paper is to demonstrate, through several examples, that it is in fact possible, as envisaged in the MILS literature, to derive interesting information security properties compositionally from a high-level specification of trusted components and their architectural structure. We focus on compositional reasoning about information-flow security properties.

We present a framework that allows the specification of a system architecture with local constraints on some system components. To give a precise meaning to the architectural structure, we extend the semantics for intransitive noninterference developed by van der Meyden [2007]. An architectural interpretation of this semantics has previously been given [van der Meyden, 2009]. In order to express constraints on trusted components, we extend architectures by labeling edges between components with functions that further restrict the information permitted to flow along edges. One of the contributions of the paper is to give a formal semantics to the enriched architectures that include these new types of edges.

We demonstrate the use of the framework through examples motivated by systems with interesting security requirements. These include multi-level secure databases, the Starlight Interactive Link [Anderson et al., 1996], and a simple electronic election system.

In each example, we identify an architectural structure and a mathematically precise set of local constraints on the trusted components. We then show that information-flow properties expressed in a logic of knowledge arise as a consequence of the interaction of the local constraints and the architectural structure. Our results show that for *any* system that is compliant with the architecture, if the trusted components satisfy their local constraints then the system satisfies the global information-flow properties.

The information security properties presented in the examples provide information-theoretic and application-specific guarantees. Thus, there are no covert channels that can violate the information security properties, and the negation of each information security property would constitute an application-specific attack.

Very few examples have been presented to date to formally justify the MILS approach to high-assurance secure systems development. One example is developed in Greve et al. [2003], but with respect to a more concrete model (based on a separation kernel formal security policy that deals with access control on memory segments) than the abstract, "noninterference" style semantics we consider. Our policy level model is more abstract, and allows greater flexibility for implementations. However, we also consider a more concrete model, systems with structured state subject to "reference monitor conditions" Rushby [1992]. We show that in this setting, to prove that a system complies with one of our extended architectures, it suffices to check a simple condition on the access control setting and to prove local properties of the trusted components.

By developing an abstract semantics for architectures and specifications of trusted components, and by developing additional examples, our work significantly advances the case that global information-flow security properties can be derived from a high-level systems architecture and local constraints on trusted components within this architecture, in the style of reasoning envisaged by Boettcher et al. [2008].

The structure of the paper is as follows. In Section 2 we review architectures and their semantics. In Section 3, we introduce the epistemic logic we use to express information security properties. In Section 4 we extend architectures with *filter functions* that allow fine-grained specification of what information flows between components. The extended architectures enable the proof of additional information security properties. We use examples throughout, to illustrate and motivate. The concrete model based on access control is developed in Section 5, and we discuss related work in Section 6.

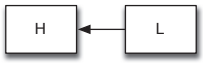
2. ARCHITECTURES AND SEMANTICS

Architectures give a policy level description of the structure of a system. We begin with a simple notion of architecture, following van der Meyden [2009]. A richer notion will be introduced later.

An architecture is a pair $\mathcal{A} = (D, \rightarrowtail)$, where D is a set of security domains, and the binary relation $\rightarrowtail \subseteq D \times D$ is an *information-flow policy*. The relation $\rightarrowtail$ is reflexive but not necessarily transitive. Intuitively, information is allowed to flow from domain u to domain v only if $u \rightarrowtail v$. The relation is reflexive as it is assumed that information flow within a domain cannot be prevented, so is always allowed.

2.1 Example: $\mathcal{HL}$ architecture

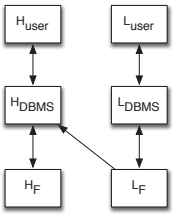
The architecture $\mathcal{HL} = (\{H, L\}, \{(L, L),\ (H, H),\ (L, H)\})$ consists of two security domains H and L, and the information-flow policy indicates that information is allowed to flow from L to H, in addition to the reflexive information flows. We can depict $\mathcal{HL}$ graphically, indicating security domains with rectangles, and the information-flow policy with arrows. We omit arrows for reflexive information flows.



2.2 Example: Hinke-Schaefer

A variety of architectures have been proposed for multi-level secure database management systems (MLS/DBMS) [Thuraisingham, 2005]. In the Hinke-Schaefer architecture [Hinke and Schaefer, 1975], several (untrusted) single-level DBMSs are composed together in a trusted operating system. Each user interacts with a single-level DBMS. The operating system enforces access control between the single-level DBMSs, allowing more restrictive DBMSs to read the storage files of less restrictive DBMSs, but not vice versa.

The following diagram shows architecture $\mathcal{HS}$, which represents the Hinke-Schaefer architecture for two security levels at the MILS policy level.



Domains H_{user} and L_{user} represent users of a high-security and low-security DBMS respectively; they interact with the single-level DBMSs H_{DBMS} and L_{DBMS} respectively. The single-level DBMSs store their data in database files denoted H_F and L_F. Note that information is allowed to flow to H_{DBMS} from both H_F and L_F, as the high-security DBMS is allowed to read the storage files of both the high-security and low-security DBMSs.

2.3 Machine model

To specify what it means for an implementation to satisfy an architecture, we must first define what an implementation is. We use the *state-observed* machine model [Rushby, 1992], which defines deterministic state-based machines.

A machine is a tuple $M = \langle S, s_0, A, D, \texttt{step}, \texttt{obs}, \texttt{dom}\rangle$ where S is a set of states, $s_0 \in S$ is the *initial state*, A is a set of *actions*, D is a set of domains, $\texttt{step} : S \times A \to S$ is a deterministic transition relation, $\texttt{dom} : A \to D$ associates a domain with each action, and observation function $\texttt{obs}$:

$D \times S \to O$ describes for each state what observations can be made by each domain, for some set of observations O.

We assume that it is possible to execute any action in any state: the function $\texttt{step}$ is total. Given sequence of actions $\alpha \in A^*$, we write $s \cdot \alpha$ for the state reached by performing each action in turn, starting in state s. We define $s \cdot \alpha$ inductively defined using the transition function $\texttt{step}$, by

$$s \cdot \epsilon = s$$
$$s \cdot \alpha a = \texttt{step}(s \cdot \alpha, a) \ .$$

(Here ϵ denotes the empty sequence.) For notational convenience, we write $\texttt{obs}_u$ for the function $\texttt{obs}(u, \cdot)$, and $\texttt{obs}_u(\alpha)$ for $\texttt{obs}_u(s_0 \cdot \alpha)$, where $\alpha \in A^*$.

Given a sequence $\alpha \in A^*$, the *view* of a group of domains G of α is the sequence of the group's observations and the actions that belong to members of the group. Intuitively, G's view is the history of its observations and the actions it has performed. The function $\texttt{view}_G$ defines the view of domain G. We first define the observation of group G at state s by $\texttt{obs}_G(s) = \langle \texttt{obs}_u(s) \rangle_{u \in G}$, i.e. the tuple of observations of individual $u \in G$. The view function is then defined inductively by

$$\texttt{view}_G(\epsilon) = \texttt{obs}_G(s_0)$$
$$\texttt{view}_G(\alpha a) = \begin{cases} \texttt{view}_G(\alpha)\, a\, \texttt{obs}_u(\alpha a) & \text{if } \texttt{dom}(a) \in G \\ \texttt{view}_G(\alpha) \circ \texttt{obs}_G(\alpha a) & \text{otherwise .} \end{cases}$$

To capture that the semantics is asynchronous and insensitive to stuttering of observations, the definition uses the absorptive concatenation operator $\circ$: for set X, sequence $\alpha \in X^*$, and element $x \in X$, $\alpha \circ x = \alpha$ if x is equal to the last element of α, and $\alpha \circ x = \alpha x$ otherwise. When $G = \{u\}$ is a singleton, then we write $\texttt{view}_u$ for $\texttt{view}_G$.

Finally, for any sequence of actions $\alpha \in A^*$, we write $\alpha \restriction G$ for the subsequence of α of actions whose domain is in the set G.

2.4 Semantics

A machine satisfies an architecture if in all possible executions of the machine, information flow is in accordance with the architecture's information-flow policy. We formalize this using an approach proposed by van der Meyden [2007], which involves the use of a concrete operational model to define an upper bound on the information that a domain is permitted to learn.

The operational model is captured using a function $\texttt{ta}_u$, which maps a sequence of actions $\alpha \in A^*$ to a representation of the maximal information that domain u is permitted to have after α, according to the policy $\rightarrowtail$. (Term "ta" is derived from *transmission* of information about *actions*; the definition corrects problems identified by van der Meyden [2007] with earlier "intransitive purge" based semantics [Rushby, 1992].)

An action of v should convey information to u only if $v \rightarrowtail u$. Moreover, the information conveyed should be no more than the information that v is permitted to have. Given machine $M = \langle S, s_0, A, D, \texttt{step}, \texttt{obs}, \texttt{dom} \rangle$, function $\texttt{ta}_u$ is defined inductively by $\texttt{ta}_u(\epsilon) = \epsilon$, and, for $\alpha \in A^*$ and $a \in A$,

$$\texttt{ta}_u(\alpha a) = \begin{cases} \texttt{ta}_u(\alpha) & \text{if } \texttt{dom}(a) \not\rightarrowtail u \\ (\texttt{ta}_u(\alpha), \texttt{ta}_{\texttt{dom}(a)}(\alpha), a) & \text{otherwise .} \end{cases}$$

Note that if information is not allowed to flow from $\texttt{dom}(a)$ to u, then $\texttt{ta}_u(\alpha a) = \texttt{ta}_u(\alpha)$, i.e., the maximal information permitted to u does not change. If information is allowed to flow from $\texttt{dom}(a)$ to u, then the information conveyed is at most the information that domain $\texttt{dom}(a)$ is permitted to have ($\texttt{ta}_{\texttt{dom}(a)}(\alpha)$), and the action a that was performed. Thus, in this case we add the tuple $(\texttt{ta}_{\texttt{dom}(a)}(\alpha), a)$ to the maximal information $\texttt{ta}_u(\alpha)$ that u was permitted to have before the action a was performed.

A machine is *TA-compliant* with an architecture if it has an appropriate set of domains, and for each domain u, what u observes in state $s_0 \cdot \alpha$ is determined by $\texttt{ta}_u(\alpha)$. That is, $\texttt{ta}_u$ describes the maximal information that u may learn: if in two runs α and α' the maximal information that u may learn is identical ($\texttt{ta}_u(\alpha) = \texttt{ta}_u(\alpha')$), then u's observations in each run must be identical ($\texttt{obs}_u(\alpha) = \texttt{obs}_u(\alpha')$).

Definition 1 (TA-compliance). *A system M is TA-compliant with architecture $(D, \rightarrowtail)$ if it has domains D and for all $u \in D$ and all sequences $\alpha, \alpha' \in A^*$ such that $\texttt{ta}_u(\alpha) = \texttt{ta}_u(\alpha')$, we have $\texttt{obs}_u(\alpha) = \texttt{obs}_u(\alpha')$.*

3. SECURITY PROPERTIES

We use a (fairly standard) propositional epistemic logic [Fagin et al., 1995b] to express information security properties. The syntax is defined as follows:

$$\phi, \psi ::= \top \mid p \mid \neg\phi \mid \phi \wedge \psi \mid K_G\phi \mid D_G\phi$$
$$G \text{ ranges over groups of domains.}$$

In case $G = \{u\}$ is a singleton, we write simply $K_u\phi$ for $K_G\phi$.

Formulas $\top$, p, $\neg\phi$, and $\phi \wedge \psi$ are standard from propositional logic: $\top$ is always satisfied, and p is a propositional constant. Epistemic formula $K_G\phi$ says that the group of domains G, considered as a single domain, knows ϕ, and $D_G\phi$ says that ϕ is *distributed knowledge* to the group of domains G, i.e., ϕ could be deduced if the information of all domains in G were combined.

Formulas are interpreted using a possible worlds semantics, where a world is a sequence of actions $\alpha \in A^*$. A *proposition* is a set $X \subseteq A^*$. We say proposition X is *non-trivial* if $X \neq \emptyset$ and $X \neq A^*$. An *interpretation function* π is a function from propositional constants to propositions.

We define the semantics of the logic using satisfaction relation $M, \pi, \alpha \vDash \phi$, which intuitively means that formula ϕ is true given interpretation function π, and machine M that has executed sequence $\alpha \in A^*$. Figure 1 defines relation $M, \pi, \alpha \vDash \phi$. We write $M, \pi \vDash \phi$ if for all $\alpha \in A^*$ we have $M, \pi, \alpha \vDash \phi$. We say that ϕ is *valid* if $M, \pi \vDash \phi$ for all systems M and interpretations π.

$$
\begin{array}{ll}
M,\pi,\alpha \vDash \top & \\
M,\pi,\alpha \vDash p & \text{iff } \alpha \in \pi(p) \\
M,\pi,\alpha \vDash \neg\phi & \text{iff } M,\pi,\alpha \nvDash \phi \\
M,\pi,\alpha \vDash \phi \wedge \psi & \text{iff } M,\pi,\alpha \vDash \phi \text{ and } M,\pi,\alpha \vDash \psi \\
M,\pi,\alpha \vDash K_G\phi & \text{iff } M,\pi,\alpha' \vDash \phi \text{ for all} \\
 & \qquad \alpha' \in A^* \text{ s.t. } \alpha \approx_G \alpha' \\
M,\pi,\alpha \vDash D_G\phi & \text{iff } M,\pi,\alpha' \vDash \phi \text{ for all} \\
 & \qquad \alpha' \in A^* \text{ s.t. } \alpha \approx^D_G \alpha'
\end{array}
$$

Figure 1: $M,\pi,\alpha \vDash \phi$

To interpret epistemic formulas $K_G\phi$, we use an indistinguishability relation for each group of domains G that describes what sequences of actions G considers possible given its view of the actual sequence of actions. Two sequences of actions $\alpha \in A^*$ and $\alpha' \in A^*$ are indistinguishable to group of domains G, written $\alpha \approx_G \alpha'$, if G's view of the two sequences are identical: $\alpha \approx_G \alpha' \iff \texttt{view}_G(\alpha) = \texttt{view}_G(\alpha')$. We interpret distributed knowledge $D_G\phi$ using a different indistinguishability relation $\approx^D_G$, defined as the intersection of $\approx_u$ for $u \in G$.

Distributed knowledge (Fagin et al. [1995a]) is the notion most commonly used in the literature on epistemic logic for the knowledge that a group would have if they pooled their local information, but group knowledge proves to have a stronger relationship to a type of architectural abstraction that we consider below. The two notions are related by the following result.

LEMMA 1. *For $u \in G$, the formulas $K_u\phi \Rightarrow D_G\phi$ and $D_G\phi \Rightarrow K_G\phi$ are valid.*

The converse relationship $K_G\phi \Rightarrow D_G\phi$ is not valid. For example, consider a system M with exactly two domains u, v, which both make observation $\perp$ at all states. Let $G = \{u, v\}$. Consider the proposition p with $\pi(p)$ consisting of all sequences in which there is an action of domain u that precedes any action of domain v. Let $\alpha = a_u a_v$ and $\alpha' = a_v a_u$ where a_u is an action of u and a_v is an action of v. Then $M,\pi,\alpha \models K_G p$, since $\texttt{view}_G(\alpha) = \texttt{view}_G(\beta)$ implies that $\beta = \alpha$. However, we have $\texttt{view}_u(\alpha) = \perp a_u \perp = \texttt{view}_u(\alpha')$, and similarly for domain v, so $\alpha \approx^D_G \alpha'$. Since $M,\pi,\alpha' \nvDash p$, we obtain that $M,\pi,\alpha \nvDash D_G p$.[1]

The value of using the less common notion $K_G\phi$ of group knowledge rather than distributed knowledge $D_G\phi$ is that it captures the way that knowledge properties are preserved under a particular type of architectural abstraction. Given a system $M = \langle S, s_0, A, D_1, \texttt{step}, \texttt{obs}, \texttt{dom}\rangle$ and a surjective mapping $r : D_1 \to D_2$, let $r(M) = \langle S, s_0, A, D_2, \texttt{step}, \texttt{obs}', \texttt{dom}'\rangle$ be the system that identical to M, except that it has domains D_2, and the functions $\texttt{dom}'$ and $\texttt{obs}'$ are defined by $\texttt{dom}' = r \circ \texttt{dom}$, and, for $u \in D_2$, $\texttt{obs}'_u(s) = \texttt{obs}_G(s)$, where $G = r^{-1}(u)$. Intuitively, each domain u in $r(M)$ corresponds to the group of domains $r^{-1}(u)$ in M, with every action of a domain in $r^{-1}(u)$ treated as an action of u.

[1]We remark that the example relies upon the assumption of asynchrony: it can be shown that in synchronous systems we have $K_G\phi \equiv D_G\phi$.

THEOREM 1. *Let $r : D_1 \to D_2$ be surjective and let M be a system with domains D_1. Then for all $u \in D_2$, interpretations π and sequences of actions α of M, we have $r(M),\pi,\alpha \models K_u p$ iff $M,\pi,\alpha \models K_G p$, where $G = r^{-1}(u)$.*

A proposition in a system M is G-action-local if the actions of domains in the group G suffice to decide the proposition. Formally, for a group G, a set $X \subseteq A^*$ is a *G-action-local proposition* if for all $\alpha, \alpha' \in A^*$ if $\alpha \restriction G = \alpha' \restriction G$, then $\alpha \in X \iff \alpha' \in X$. We write "$u$-action-local" when $G = \{u\}$. Note that if $G \subseteq G'$ and X is G-action local then X is G'-action local.

We can also reason about how architectural abstraction affects G-action-local propositions.

LEMMA 2. *Let $r : D_1 \to D_2$ be surjective and let M be a system with domains D_1. Then X is a G-action-local proposition in $r(M)$ iff X is a $r^{-1}(G)$-action-local proposition in M.*

3.1 Example: $\mathcal{HL}$ information security

The logic allows us to state information security properties about machines, in terms of the knowledge of domains. The architecture can provide sufficient structure to prove that a given information security property holds in all machines that comply with the architecture.

For example, using the $\mathcal{HL}$ architecture, we are able to show that in any execution of any machine that complies with $\mathcal{HL}$, the domain L does not know any non-trivial H-action-local proposition.

THEOREM 2. *If M is TA-compliant with $\mathcal{HL}$ and $\pi(p)$ is a non-trivial H-action-local proposition then $M,\pi \vDash \neg K_L p$.*

3.2 Example: Hinke-Schaefer

In the Hinke-Schaefer database architecture $\mathcal{HS}$, none of the domains L_{user}, L_{DBMS}, or L_F know anything about the domains H_{user}, H_{DBMS} or H_F. This is true even if we consider the distributed knowledge of L_{user}, L_{DBMS}, and L_F.

THEOREM 3. *Let system M be TA-compliant with $\mathcal{HS}$, and let $G = \{L_{user}, L_{DBMS}, L_F\}$. If $\pi(p)$ is a non-trivial $\{H_{user}, H_{DBMS}, H_F\}$-action-local proposition, then $M,\pi \vDash \neg K_G p$.*

The proof of Theorem 3 follows easily from Theorems 1 and 2, and because the $\mathcal{HL}$ architecture is an abstraction of the $\mathcal{HS}$ architecture (with $r^{-1}(H) = \{H_{user}, H_{DBMS}, H_F\}$ and $r^{-1}(L) = \{L_{user}, L_{DBMS}, L_F\}$).

Since $K_u p \Rightarrow K_G p$ is valid for $u \in G$, it follows that also $M,\pi \vDash \neg K_u p$ for $u \in \{L_{user}, L_{DBMS}, L_F\}$. In particular, L_{user} does not have any information about the High side of the system.

4. EXTENDED ARCHITECTURE

The architectures used so far impose coarse, global constraints on the causal structure of systems. If $u \rightarrowtail v$ then TA-compliance permits domain u to send to domain v any and all data it has. However, in many systems, key security properties depend on the fact that trusted components allow only certain information to flow from one domain to another. Finer specification of information flows in the architecture allow us to prove stronger information security properties.

In this section we extend the notion of architecture by introducing *filter functions* to allow fine-grained specification of what information flows between domains. We define semantics for these extended architectures, and present examples where the extended architectures allow us to prove strong information security properties.

4.1 Filter functions

An extended architecture is a pair $\mathcal{A} = (D, \rightarrowtail)$, where D is a set of security domains, and $\rightarrowtail \subseteq D \times D \times (\mathcal{L} \cup \{\top\})$, where $\mathcal{L}$ is a set of function names. We write $u \overset{f}{\rightarrowtail} v$ when $(u, v, f) \in \rightarrowtail$, write $u \rightarrowtail v$ as shorthand for $\exists f.\ (u, v, f) \in \rightarrowtail$, and $u \not\rightarrowtail v$ as shorthand for $\neg\exists f.\ (u, v, f) \in \rightarrowtail$.

Intuitively, $u \overset{f}{\rightarrowtail} v$ represents that information flow from u to v is permitted, but may be subject to constraints. In case $f = \top$, there are no constraints on information flow from u to v: any information that may be possessed by u is permitted to be passed to v when u acts, just as in the definition of TA-compliance. If $f \in \mathcal{L}$ then information is allowed to flow from domain u to domain v, but it needs to be *filtered* through the function denoted by f: only information output by this function may be transmitted from u to v. If $u \not\rightarrowtail v$ then no direct flow of information from u to v is permitted.

In some cases, it may be possible for the operating system or network infrastructure to enforce a given filter function. However, in general, a filter function is a local constraint on a trusted component of the system. That is, if $u \overset{f}{\rightarrowtail} v$ for $f \neq \top$, then component u is trusted to enforce that information sent to v is filtered appropriately.

We require that extended architectures have the following properties:

1. For all $u, v \in D$, there exists at most one $f \in \mathcal{L} \cup \{\top\}$ such that $u \overset{f}{\rightarrowtail} v$.
2. The relation $\rightarrowtail$ is reflexive in that for all $u \in D$ we have $(u, u, \top) \in \rightarrowtail$.

The first condition requires that all permitted flows of information from u to v are represented using a single labeled edge. Intuitively, any policy with multiple such edges can always be transformed into one satisfying this condition, by combining the pieces of information flowing across these edges into a tuple that flows across a single edge. The second condition is motivated from the fact, already noted above, that information flow from a domain to itself cannot be prevented.

Extended architectures do not define the interpretations of the function names $\mathcal{L}$. If $\mathcal{A} = (D, \rightarrowtail)$ is an extended architecture, an *interpretation* for $\mathcal{A}$ is a tuple $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$, where A is a set of actions, $\mathtt{dom} : A \to D$ assigns these actions to domains of $\mathcal{A}$, and $\mathtt{I}$ is a function mapping each $f \in \mathcal{L}$ to a function with domain A^* (and arbitrary codomain). We call the pair $(\mathcal{A}, \mathcal{I})$ an *interpreted extended architecture*, or simply an *interpreted architecture*.

Intuitively, if $u \overset{f}{\rightarrowtail} v$ and $\alpha \in A^*$ and $a \in A$ is an action with $\mathtt{dom}(a) = u$, then $\mathtt{I}(f)(\alpha a)$ is the information that is permitted to flow from u to v when the action a is performed after occurrence of the sequence of actions $\alpha \in A^*$.

Given extended architecture $(D, \rightarrowtail)$ and an architectural interpretation $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$, we define a function $\mathtt{fta}_u$ with domain A^* that, like $\mathtt{ta}_u$, captures that maximal information that domain u is permitted to have after a sequence of actions has been executed. The definition is recursive with a function $\mathtt{T}_{u,v}$ for $u, v \in D$, mapping a sequence $\alpha \in A^*$ and an action $a \in A$ with $\mathtt{dom}(a) = u$ to

$$\mathtt{T}_{v,u}(\alpha, a) = \begin{cases} \epsilon & \text{if } v \not\rightarrowtail u \\ (\mathtt{fta}_v(\alpha), a) & \text{if } v \overset{\top}{\rightarrowtail} u \\ \mathtt{I}(f)(\alpha a) & \text{if } v \overset{f}{\rightarrowtail} u. \end{cases}$$

Intuitively, $\mathtt{T}_{v,u}(\alpha, a)$ represents the new information permitted to be known by u when action a is performed after sequence α.

The function $\mathtt{fta}_u$ is defined by $\mathtt{fta}_u(\epsilon) = \epsilon$, and, for $\alpha \in A^*$ and $a \in A$, $\mathtt{fta}_u(\alpha a) = \mathtt{fta}_u(\alpha) \,\hat{}\, \mathtt{T}_{\mathtt{dom}(a),u}(\alpha, a)$ where $\hat{}$ is the operation of appending an element to the end of a sequence. Some important technical points concerning the append operation are that for any sequence σ, we define $\sigma \,\hat{}\, \epsilon = \sigma$ (i.e., appending the empty sequence ϵ has no effect), and if δ happens to be a nonempty sequence, then $\sigma \,\hat{}\, \delta$ is the sequence that extends the sequence σ by the single additional element δ. For example if δ is the sequence ab, then $\sigma \,\hat{}\, \delta$ has final element equal to sequence ab rather than b.

Unfolding the definition, we obtain

$$\mathtt{fta}_u(\alpha a) = \begin{cases} \mathtt{fta}_u(\alpha) & \text{if } \mathtt{dom}(a) \not\rightarrowtail u \\ \mathtt{fta}_u(\alpha) \,\hat{}\, (\mathtt{fta}_{\mathtt{dom}(a)}(\alpha), a) & \text{if } \mathtt{dom}(a) \overset{\top}{\rightarrowtail} u \\ \mathtt{fta}_u(\alpha) \,\hat{}\, \mathtt{I}(f)(\alpha a) & \text{if } \mathtt{dom}(a) \overset{f}{\rightarrowtail} u. \end{cases}$$

The first two clauses resemble the definition of $\mathtt{ta}_u$; the third adds to this that the information flowing along an edge labeled by a function name f is filtered by the interpretation $\mathtt{I}(f)$. Note that if $\mathtt{I}(f)(\alpha a) = \epsilon$, where $\mathtt{dom}(a) \overset{f}{\rightarrowtail} u$, then $\mathtt{fta}_u(\alpha a) = \mathtt{fta}_u(\alpha)$. That is, filter function $\mathtt{I}(f)$ can specify that no information should flow under certain conditions. Note also that $\mathtt{fta}_u$ and $\mathtt{T}_{v,u}$ have implicit parameters, viz., an information flow policy $\rightarrowtail$ and an architectural interpretation $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$. When we need to make parameters explicit, we write expressions such as $\mathtt{fta}_u^{(\rightarrowtail, \mathcal{I})}$, or $\mathtt{fta}_u^{\rightarrowtail}$.

The function $\mathtt{fta}_u$ is used analogously to $\mathtt{ta}_u$ to define the maximal information that a domain is permitted to observe for a given sequence of actions. However, $\mathtt{fta}_u$ is a more precise bound than $\mathtt{ta}_u$, as it uses filter functions to bound the information sent between domains.

It is reasonable to assume that information sent from u to v is information that u is permitted to have. We say that a function h with domain A^* is $\mathtt{fta}_u$-compatible when for all sequences $\alpha, \beta \in A^*$, $\mathtt{fta}_u(\alpha) = \mathtt{fta}_u(\beta)$ implies that for all $a \in A$ with $\mathtt{dom}(a) = u$ we have $h(\alpha a) = h(\beta a)$. We say that the interpretation $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$ is *compatible* with $\mathcal{A} = (D, \rightarrowtail)$ if for all $u \in D$ and edges $u \overset{f}{\rightarrowtail} v$ with $f \in \mathcal{L}$, the function $\mathtt{I}(f)$ is $\mathtt{fta}_u$-compatible. In what follows, we require that interpretations be compatible with their architectures.

A machine complies with an interpreted extended architecture if it has appropriate domains and actions, and for each domain u, what u observes in state $s_0 \cdot \alpha$ is determined by $\mathtt{fta}_u(\alpha)$. We call such a machine *FTA-compliant*. ("FTA" is derived from *filtered transmission* of information about *actions*.)

Definition 2 (FTA-compliant). *A machine $M = \langle S, s_0, A, D, \mathtt{step}, \mathtt{obs}, \mathtt{dom}\rangle$ is FTA-compliant with an interpreted architecture $(\mathcal{A}, \mathcal{I})$, with $\mathcal{A} = (D', \rightarrowtail)$ and $\mathcal{I} = (A', \mathtt{dom}', \mathtt{I})$, if $A = A'$, $D = D'$, $\mathtt{dom} = \mathtt{dom}'$ and for all agents $u \in D$ and all $\alpha, \alpha' \in A^*$, if $\mathtt{fta}_u(\alpha) = \mathtt{fta}_u(\alpha')$ then $\mathtt{obs}_u(\alpha) = \mathtt{obs}_u(\alpha')$.*

For an interpreted architecture $\mathcal{AI}$, with actions A and domains D, if π is an interpretation with $\pi(p) \subseteq A^*$, we write $\mathcal{AI}, \pi, \alpha \models \phi$ if $M, \pi, \alpha \models \phi$ for all systems M that are FTA-compliant with $\mathcal{AI}$. Similarly, we write $\mathcal{AI}, \pi \models \phi$ if $\mathcal{AI}, \pi, \alpha \models \phi$ for all $\alpha \in A^*$.

Separating extended architectures from their interpretations ensures that extended architectures can be completely represented by graphical diagrams with labeled edges. It also allows us to deal with examples where an extended architecture can be implemented in a variety of ways, and weak constraints on the set of actions and the set of filter functions suffice to enforce the security properties of interest. We will present a number of examples of this in what follows. To capture the constraints on the architectural interpretations at the semantic level, we use the notion of an *architectural specification*, which is a pair $(\mathcal{A}, \mathcal{C})$ where $\mathcal{A}$ is an extended architecture and $\mathcal{C}$ is a set of architectural interpretations for $\mathcal{A}$. (We will not attempt in this paper to develop any syntactic notation for architectural specifications.)

Definition 3. *A machine M is FTA-compliant with an architectural specification $(\mathcal{A}, \mathcal{C})$ if there exists an interpretation $\mathcal{I} \in \mathcal{C}$ such that M is FTA-compliant with the interpreted architecture $(\mathcal{A}, \mathcal{I})$.*

When drawing extended architectures, we annotate arrows between domains with the filter function names. For arrows drawn without a label, and elided reflexive arrows, the implied label is $\top$.

The following theorem shows that FTA-compliance generalizes TA-compliance. Thus, we are free to interpret a given architecture as an extended architecture.

Theorem 4. *Let $\mathcal{A}_1 = (D, \rightarrowtail_1)$ be an architecture, and let $\mathcal{A}_2 = (D, \rightarrowtail_2)$ be the extended architecture such that $(u, v, f) \in \rightarrowtail_2$ if and only if $f = \top$ and $(u, v) \in \rightarrowtail_1$. Let M be a machine with domains D, actions A and domain function $\mathtt{dom}$. Let $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$ be any interpretation for $\mathcal{A}_2$ with this set of actions and domain function. Then M is TA-compliant with $\mathcal{A}_1$ if and only if M is FTA-compliant with $(\mathcal{A}_2, \mathcal{I})$.*

FTA-compliance requires that if $\mathtt{fta}_u(\alpha) = \mathtt{fta}_u(\alpha')$ then the observations of u in state $s_0 \cdot \alpha$ and in state $s_0 \cdot \alpha'$ are equal. The following lemma shows that in fact FTA-compliance implies that if $\mathtt{fta}_u(\alpha) = \mathtt{fta}_u(\alpha')$ then we have that $\mathtt{view}_u(\alpha) = \mathtt{view}_u(\alpha')$.

We require a technical assumption for this result: say that an interpreted architecture $(\mathcal{A}, \mathcal{I})$ is *non-conflating* if for all $u, v \in D$, if $u \overset{f}{\rightarrowtail} v$ with $f \neq \top$ then for all actions $a, b \in A$ with $\mathtt{dom}(a) = u$ and $\mathtt{dom}(b) = v$, and for all $\alpha, \beta \in A^*$ we have $\mathtt{I}(f)(\alpha a) \neq (\mathtt{fta}_v(\beta), b)$. Recall that by reflexivity, $u \overset{f}{\rightarrowtail} v$ with $f \neq \top$ implies $u \neq v$. Intuitively, the condition states that, in the context of the definition of $\mathtt{fta}_v$, it is always possible for domain v to distinguish the type of information $\mathtt{I}(f)(\alpha a)$ transmitted to it from the type of information $(\mathtt{fta}_v(\beta), b)$ transmitted by v to itself. That is, v can distinguish between the effects of its own actions on $\mathtt{fta}_v$ and the effects of other domains' actions. Since, intuitively, v should be aware of its own actions, it is reasonable to expect that this is generally satisfied.

Lemma 3. *If M is FTA-compliant with non-conflating interpreted architecture $(\mathcal{A}, \mathcal{I})$, with $\mathcal{A} = (D, \rightarrowtail)$ and $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$, then for all agents $u \in D$ and all $\alpha, \alpha' \in A^*$ such that $\mathtt{fta}_u(\alpha) = \mathtt{fta}_u(\alpha')$ we have $\mathtt{view}_u(\alpha) = \mathtt{view}_u(\alpha')$*

We note that this result does not hold for conflating interpreted architectures. For example, consider an interpreted architecture in which $\mathtt{dom}(a) \overset{f}{\rightarrowtail} \mathtt{dom}(b)$ and $\mathtt{I}(f)(a) = (\epsilon, b)$ and a system in which $\mathtt{obs}_u(s) = \bot$ for all $u \in D$ and states s. This system is necessarily FTA-compliant with the architecture, but we have $\mathtt{fta}_{\mathtt{dom}(b)}(a) = (\epsilon, b) = \mathtt{fta}_{\mathtt{dom}(b)}(b)$ but $\mathtt{view}_{\mathtt{dom}(b)}(a) = \bot$ and $\mathtt{view}_{\mathtt{dom}(b)}(b) = \bot b \bot$.

However, it is always possible to convert an interpretation $\mathtt{I}$ into another equivalent non-conflating interpretation $\mathtt{I}'$, by defining $\mathtt{I}'(f)(\alpha) = (\mathtt{I}(f)(\alpha), x)$ for some fixed value x that is not in A. Note that $\mathtt{I}$ and $\mathtt{I}'$ are equivalent in the sense that $\mathtt{I}(f)(\alpha) = \mathtt{I}(g)(\beta)$ if and only if $\mathtt{I}'(f)(\alpha) = \mathtt{I}'(g)(\beta)$, so the "information content" of $\mathtt{I}$ and $\mathtt{I}'$ are the same. We therefore assume in what follows that interpreted architectures are non-conflating.

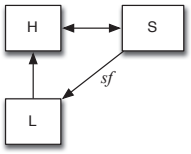
It may be onerous to prove $\mathtt{fta}_u$-compatibility directly for each function $\mathtt{I}(f)$ such that $u \overset{f}{\rightarrowtail} v$. However, it suffices to show that the function is $\mathtt{obs}_u$-compatible (that is, for all $\alpha, \beta \in A^*$, if $\mathtt{obs}_u(\alpha) = \mathtt{obs}_u(\beta)$ then that for all $a \in A$ with $\mathtt{dom}(a) = u$ we have $\mathtt{I}(f)(\alpha a) = \mathtt{I}(f)(\beta a)$), since FTA-compliance insists that if $\mathtt{fta}_u(\alpha) = \mathtt{fta}_u(\beta)$ then $\mathtt{obs}_u(\alpha) = \mathtt{obs}_u(\beta)$.

4.2 Example: Starlight Interactive Link

The Starlight Interactive Link [Anderson et al., 1996] provides interactive access from a high-security network to a low-security network. This allows a user on the high-security network to have windows open on her screen at differing security levels, while ensuring no high-security information goes to the low-security network.

Starlight has both hardware and software components. The hardware device is connected to both the high-security and low-security networks, and has a keyboard and mouse attached. There is a switch that can toggle between the high-security and low-security networks; input from the mouse and keyboard are sent to the network currently selected by the switch. Starlight allows data from the low-security network to be transferred to the high-security network, but not vice versa. The Starlight software components include proxy window clients and servers to allow the windowing environment to work in the presence of the Starlight hardware.

The following diagram shows extended architecture $\mathcal{SL}$, an architecture for the Starlight Interactive Link.



Domain H represents the high-security network (including the user's computer); domain L represents the low-security network; domain S represents the Starlight Interactive Link (including input devices), which routes keyboard and mouse events to either the high-security or low-security network. Note that there is no edge from domain L to domain S, as no information is sent directly from the low-security network to the Starlight Interactive Link. Instead, data from the low-security network (such as updates to the contents of a window) are sent to the high-security network, and thence to software components of the Starlight Interactive Link.

The edge labeled *sf* restricts what information is allowed to flow from the Starlight Interactive Link to low-security network L. We present an architectural specification $\mathcal{C}_{\mathcal{SL}}$ based on $\mathcal{SL}$ that expresses a constraint on interpretations of *sf*. An interpretation $(A, \mathtt{dom}, \mathtt{I})$ is included in $\mathcal{C}_{\mathcal{SL}}$ if the following conditions hold. Let $A_S = \{a \in A \mid \mathtt{dom}(a) = S\}$ be the set of actions belonging to domain S. We assume that there is a distinguished action $t \in A_S$ that toggles which network is receiving the input events. For all i, let $\alpha_i \in A^*$ be a sequence of actions that does not contain t. Intuitively, L is permitted to know about the occurrence of any t action, and the occurrence of any other action in A_S (e.g., keyboard or mouse input) that happens while the low-security network is selected (i.e., after an odd number of toggle actions). We capture this by the following assumption on interpretation $\mathtt{I}$:

$$\mathtt{I}(sf)(\alpha a) = \begin{cases} a & \text{if } a = t \text{ or} \\ & \quad \#_t(\alpha) \text{ is odd and } a \in A_S \\ \epsilon & \text{otherwise} \end{cases}$$

where $\#_t(\alpha)$ is the number of occurrences of t in α. For example, with the α_i consisting of only H and L actions, and s an S action, we have $\mathtt{I}(sf)(\alpha_0\, s) = \epsilon$, $\mathtt{I}(sf)(\alpha_0\, s\, t) = t$, $\mathtt{I}(sf)(\alpha_0\, s\, t\, \alpha_1\, s) = s$, and $\mathtt{I}(sf)(\alpha_0\, s\, t\, \alpha_1\, s\, t\, \alpha_2\, s) = \epsilon$, etc. It is straightforward to check for such an interpretation that $\mathtt{I}(sf)$ is $\mathtt{fta}_S$-compatible.

The component S, corresponding to the Starlight Interactive Link, is a trusted component, and the *sf* filter is a local constraint on the component. To verify that a system satisfies specification $(\mathcal{SL}, \mathcal{C}_{\mathcal{SL}})$, we would need to verify that S appropriately filters information sent to L, and that all other communication in the system complies with the architecture, to wit, that H cannot communicate directly with L, and L cannot communicate directly with S.

If a system does satisfy specification $(\mathcal{SL}, \mathcal{C}_{\mathcal{SL}})$, then we can show that domain L never knows any H-action-local proposition. Indeed, we can show something stronger, that L never knows any proposition about H actions and S actions that occur while the high-network is selected.

To fully capture this intuition requires a little care, since a proposition such as "H did action a between the first and second t actions" should not be known to L, but refers both to something that should be hidden from L and to something that L is permitted to observe (the t actions). We handle this by introducing a formal way to express that anything that L knows about the hidden content and its relation to t in fact depends just on the t events, (and so is permitted knowledge).

First, to represent the forbidden information, we define consider propositions that depend only on H actions, t actions and S actions while the system is toggled to the high security network. Let the *canonical form* of a sequence $\alpha \in A^*$ be the (unique) representation $\alpha = \alpha_0 t \alpha_1 t \alpha_2 \ldots t \alpha_n$ where each α_i contains no t actions. Define the function $togH : A^* \to A^*$ so that if $\alpha = \alpha_0 t \alpha_1 t \alpha_2 \ldots t \alpha_n$ is the canonical form then

$$togH(\alpha) = (\alpha_0 \restriction HS)\, t\, (\alpha_1 \restriction H)\, t\, (\alpha_2 \restriction HS)\, t \ldots t\, (\alpha_n \restriction G_n)$$

where G_n is HS if n is even and $G_n = H$ if n is odd. (Here we abbreviate the set $\{H\}$ to L and $\{H, S\}$ to HS.) Intuitively, $togH(\alpha)$ is the subsequence of α consisting of events that L is not permitted to know about, plus any t events. Formally, a proposition $X \subseteq A^*$ is *toggle-H dependent* if for all $\alpha, \beta \in A^*$, if $togH(\alpha) = togH(\beta)$, then $\alpha \in X$ iff $\beta \in X$.

To capture information that can be deduced from just the toggle events in a sequence (e.g. whether the number of t events is prime), we add to the logic a new domain T that sees only the T events. Formally, this domain corresponds to the view function defined by $\mathtt{view}_T(\alpha) = \#_t(\alpha)$, and is associated with the knowledge operator K_T with semantics derived in the usual way from this view function.

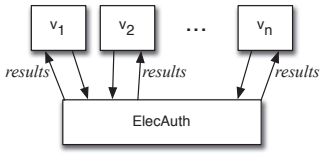
We can now formally capture that the only toggle-H dependent propositions that L knows are those that can be derived just from the t events that L sees, in the following result.

THEOREM 5. *Suppose system M is FTA-compliant with architectural specification $(\mathcal{SL}, \mathcal{C}_{\mathcal{SL}})$. If $\pi(p)$ is a toggle-H dependent proposition, then $M, \pi \vDash K_L p \Rightarrow K_T p$.*

Note that we would not be able show this security property in an architecture without filter functions, since the domain S can communicate with both H and L. It is the filter function that allows us to show that S's communication with L reveals nothing about H actions, and only limited information about S actions.

4.3 Example: Electronic election

The following diagram shows architecture $\mathcal{EE}$, an electronic election for n voters, coordinated by an election authority *ElecAuth*.



In many elections, the behavior of individual voters is confidential information: it should not be known how voters voted. (Elections have several information-flow security requirements. For example, the final result should be correctly computed from the votes—an integrity requirement. We focus here on the confidentiality requirement for voters.)

By specifying an additional local constraint on the election authority, we can show that this architecture enforces anonymity on the identity of the voters, thus satisfying the confidentiality requirement about the behavior of voters. The election authority's compliance with this local constraint might be assured by means of a careful verification of its implementation, or carefully designed cryptographic protocols (which may remove some or all of the trust required to be placed in the election authority).

We first assume that voters are homogenous in that they have the same set of actions. Let $(\mathcal{EE}, \mathcal{I})$ be an interpreted architecture where $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$. We say that architecture interpretation $\mathcal{I}$ is *voter homogenous* if for any voter v, the set of possible actions $A_v = \{a \in A \mid \mathtt{dom}(a) = v\}$ equals $\{a^v \mid a \in A_V\}$, where A_V is the set of action types available to voters. Intuitively, a^v represent the action of type a when performed by voter v.

We define a voter permutation P as a permutation over the set of voters V. Since all voters have the same set of actions in a voter-homogenous interpretation, we can apply a permutation P to sequence $\alpha \in A^*$, written $P(\alpha)$, as follows:

$$\begin{aligned} P(\epsilon) &= \epsilon \\ P(\alpha a) &= P(\alpha)a && \text{if } \mathtt{dom}(a) \notin V \\ P(\alpha a^v) &= P(\alpha)a^{P(v)} && \text{if } a \in A_V \ . \end{aligned}$$

We apply permutation P to proposition $X \subseteq A^*$ by applying P to each sequence $\alpha \in X$, i.e.,

$$P(X) = \{P(\alpha) \mid \alpha \in X\} \ .$$

Plainly, if X is a u-action-local proposition, for $u \in V$, then $P(X)$ is a $P(u)$-action-local proposition.

Using voter permutations, we can now state the local constraint that election results do not depend on (and thus do not reveal) the identity of any voter.

A1. Election results have the following *identity-oblivious* property: Given voter-homogenous interpreted architecture $(\mathcal{EE}, \mathcal{I})$ where $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$, for all voter permutations P, sequences $\alpha \in A^*$, and actions $a \in A$ with $\mathtt{dom}(a) = \mathit{ElecAuth}$, we have $\mathtt{I}(\mathit{results})(\alpha a) = \mathtt{I}(\mathit{results})(P(\alpha)a)$.

There are several possible interpretations of *results* that satisfy this constraint, such as a function that returns each candidate and her vote tally, or a function that returns the total number of votes submitted. However, a function that returns a ballot and the identity of the voter that submitted it doesn't satisfy constraint A1.

We define architecture specification $\mathcal{C}_{\mathcal{EE}}$ such that $\mathcal{I} \in \mathcal{C}_{\mathcal{EE}}$ if and only if $\mathcal{I}$ is a voter-homogenous interpretation of $\mathcal{EE}$ that satisfies constraint A1.

Given the local constraint A1, we can show that if voter v believes that some proposition X may be satisfied, then v also believes that $P(X)$ may be satisfied, for voter permutation P. For example, if Alice considers it possible that Bob voted for Obama and Charlie voted for McCain, then Alice considers it possible that Charlie voted for Obama and Bob voted for McCain.

THEOREM 6. *Let system M be FTA-compliant with architectural specification $(\mathcal{EE}, \mathcal{C}_{\mathcal{EE}})$. Let v be a voter. For all voter permutations P such that $P(v) = v$, if $\pi(q) = P(\pi(p))$, then $M, \pi, \alpha \vDash \neg K_v \neg p \Rightarrow \neg K_v \neg q$ for all $\alpha \in A^*$.*

Note that Theorem 6 does not imply that voter v learns nothing about other voters. For example, if the election results reveal that all voters voted for Obama, then voter v knows how every other voter voted. Also, if there are only 2 voters, v and w, then the results may reveal to v exactly how w voted. However, Theorem 6 provides anonymity: given the results, voter v cannot distinguish the behavior of other voters.

5. IMPLEMENTING ARCHITECTURES USING ACCESS CONTROL

One of the mechanisms that might be used to enforce compliance with an information flow architecture is access control restrictions on the ability of domains to read and write objects. This idea was already implicit in the Bell and La Padula [1976] approach of enforcing that high level information should not flow to low level domains through a "no read up" and "no write down" access control policy. The idea was given a more semantically well-founded expression by Rushby [1992], who established a formal relation between access control systems and a theory of information flow based on intransitive noninterference policies. Rushby's "reference monitor conditions" give semantics to the notion of reading and writing, which was absent in the work of Bell and La

Padula. Rushby's formulation was sharpened and shown to be closely related to TA-security by van der Meyden [2007].

In this section, we present a generalization of van der Meyden's formulation of access control, and show how enforcement of an access control policy together with local verification of trusted components can be used to assure that a system is compliant with an extended architecture.

We first recall some definitions and results from van der Meyden [2007]. The system model we have used to this point does not require the states of a system to be equipped with any internal structure. In practice, systems typically will be constructed as an assembly of components. To capture this, Rushby [1992] introduced the notion of a *system with structured state*, which is a system M (with states S and domains D) together with

1. a set $\mathtt{Obj}$ of *objects*,
2. a set V of *values*, and functions
3. $\mathtt{contents} : S \times \mathtt{Obj} \to V$, with $\mathtt{contents}(s, n)$ interpreted as the value of object n in state s, and
4. $\mathtt{observe}, \mathtt{alter} : D \to \mathcal{P}(\mathtt{Obj})$, with $\mathtt{observe}(u)$ and $\mathtt{alter}(u)$ interpreted as the set of objects that domain u can observe (or read) and alter (or write), respectively.

For brevity, we write $s(x)$ for $\mathtt{contents}(s, x)$. We call the pair $(\mathtt{observe}, \mathtt{alter})$ the *access control* table of the machine. For each domain u, we define an equivalence relation of "observable content equivalence" on states $s, t \in S$ by $s \approx_u^{oc} t$ if $s(x) = t(x)$ for all $x \in \mathtt{observe}(u)$.

Rushby introduced *reference monitor conditions* on such machines in order to capture formally the intuitions associated with the pair $(\mathtt{observe}, \mathtt{alter})$ being an access control table that restricts the ability of the actions to "read" and "write" the objects $\mathtt{Obj}$. van der Meyden [2007] sharpened these conditions to the following (the difference is in RM2):

RM1. If $s \approx_u^{oc} t$ then $\mathtt{obs}_u(s) = \mathtt{obs}_u(t)$.

RM2. For all actions $a \in A$, states $s, t \in S$ and objects $x \in \mathtt{alter}(dom(a))$, if $s \approx_{\mathtt{dom}(a)}^{oc} t$ and $s(x) = t(x)$ then $(s \cdot a)(x) = (t \cdot a)(x)$.

RM3. If $x \notin \mathtt{alter}(dom(a))$ then $s(x) = (s \cdot a)(x)$

Intuitively, RM1 states that a domain's observation depends only on the values of the objects that it can observe (or read). RM2 states that if action a is performed in a domain u that is permitted to alter an object x, then the new value of the object after the action depends only on its old value and the values of objects that domain u is permitted to observe. The final conditions RM3 says that if action a is performed in a domain that is not permitted to alter (or write) an object x, then the value of x does not change.

We note that the terminology "reference monitor conditions" points to the fact that these conditions can be enforced by a reference monitor that mediates all attempts to perform an action, simply by denying requests by a domain u to read an object not in $\mathtt{observe}(u)$ or write to an object not in $\mathtt{alter}(u)$.

In addition to the reference monitor assumptions, Rushby considers a condition stating that if there is an object that may be altered by domain u and observed by domain v, then the information flow policy should permit flow of information from u to v. (Obviously, the object x provides a channel for information to flow from u to v.)

AOI. If $\mathtt{alter}(u) \cap \mathtt{observe}(v) \neq \emptyset$ then $u \rightarrowtail v$.

van der Meyden [2007] shows the following, strengthening a result of Rushby [1992].

THEOREM 7. *If M is a system with structured state satisfying RM1-RM3 and AOI with respect to $\rightarrowtail$ then M is TA-secure with respect to $\rightarrowtail$.*

We now develop a generalization of this result to extended architectures. As a first step, note that in extended architectures, the situation where the information flow policy potentially permits flow of information from domain u to domain v corresponds to the existence of an edge $u \overset{f}{\rightarrowtail} v$ for some label f (possibly $\top$). This motivates the following variant of condition AOI:

AOI′. If $\mathtt{alter}(u) \cap \mathtt{observe}(v) \neq \emptyset$ then $u \overset{f}{\rightarrowtail} v$ for some f.

Next, we develop a set of conditions that check that information flow constraints of the form $u \overset{f}{\rightarrowtail} v$ with $f \neq \top$ have been correctly implemented in a system. Let $\mathcal{I} = (A, \mathtt{dom}, \mathtt{I})$ be an interpretation of architecture $\mathcal{A} = (D, \rightarrowtail)$. Consider the following constraints in a system M with actions A, domains D and domain function $\mathtt{dom}$:

I1. If $\mathtt{dom}(a) \overset{f}{\rightarrowtail} u$ for $f \neq \top$ and $\mathtt{I}(f)(\alpha, a) = \epsilon$ and $x \in \mathtt{observe}(u) \cap \mathtt{alter}(\mathtt{dom}(a))$ then $(s_0 \cdot \alpha a)(x) = (s_0 \cdot \alpha)(x)$.

I2. If $\mathtt{dom}(a) \overset{f}{\rightarrowtail} u$ with $f \neq \top$ and $\mathtt{dom}(b) \overset{g}{\rightarrowtail} u$ with $f \neq \top$ and $\mathtt{I}(f)(\alpha, a) = \mathtt{I}(g)(\beta, b) \neq \epsilon$ and $x \in \mathtt{observe}(u) \cap (\mathtt{alter}(\mathtt{dom}(a)) \cup \mathtt{alter}(\mathtt{dom}(b)))$ and $(s_0 \cdot \alpha)(x) = (s_0 \cdot \beta)(x)$ then $(s_0 \cdot \alpha a)(x) = (s_0 \cdot \beta b)(x)$.

We note that verification of these constraints requires consideration only of domains that are trusted, in the sense that they have outgoing edges not labelled $\top$, and the objects that such domains are permitted to alter. Thus verification of these constraints can be localized to the trusted domains. The following result states that such local verification, together with enforcement of an access control policy consistent with the information flow policy via a mechanism satisfying the reference monitor constraints, suffices to assure that an information flow policy has been satisfied:

THEOREM 8. *Let $\mathcal{AI}$ be a strongly non-conflating interpreted architecture. Suppose that M is a system with structured state satisfying RM1-RM3, AOI′ and I1-I2. Then M is FTA-compliant with $\mathcal{AI}$.*

6. RELATED WORK

Other work has sought to formally describe system architecture (e.g., AADL and Acme [Garlan et al., 2000]), and reason about the properties of systems conforming to a given architecture. There are many software engineering concerns that can be reasoned about in architectural design, such as maintainability, and reliability. This work focuses on reasoning about the information security of systems, and, as such, our architectures specify local constraints on information that may be communicated between components. The local constraints allow the inference of global information security properties. This work is complementary to work on other aspects of system design.

Relatively little theoretical work takes an architectural perspective on information security. One interesting line of work [Hansson et al., 2008] that takes a similar perspective to ours is conducted in the context the architectural modeling framework AADL. This work is based on the Bell La Padula model [Bell and La Padula, 1976]. In a similar spirit are works on Model Driven Security, which extend UML with security modeling notations. In Basin et al. [2006], the focus is on a UML extension for role-based access control policies and model transformations to implementation infrastructures such as Enterprise Java Beans or .NET. Jürjens [2005] extends UML with a focus on reasoning about secrecy in distributed applications employing security protocols. None of these approaches use the application-specific abstract non-interference-style semantics that underpins our contribution, nor do they target the type of reasoning envisaged in the MILS community for development of high-assurance systems built on infrastructure such as separation kernels.

A number of approaches have been developed that allow security of a composed system to be derived from security of its components. These include use of a stronger definitions of security such as restrictiveness [McCullough, 1990] or bisimulation-based nondeducibility on compositions [Focardi and Gorrieri, 1994]. McLean [1996] proposes a framework for specifying and reasoning about system composition, and the preservation of possiblistic security properties. In this work, we are not concerned with showing that security properties that hold of components also hold of a composite system. Instead we are concerned with proving global security properties, and identifying local constraints that components must satisfy. The literature on preservation of information flow security under composition has also largely limited itself to the simple policy $L \rightarrowtail H$.

Downgrading has historically been one of the motivations for generalizing the notion of noninterference to intransitive policies. Roscoe and Goldsmith [1999] argued against the ipurge-based semantics for noninterference on the grounds that the meaning it gives to the downgrader policy $H \rightarrowtail D \rightarrowtail L$ is too permissive. According to this semantics, any action by the downgrader D "opens the floodgates," in the sense that it allows all information about the High security domain H to flow to the low security domain L. Roscoe and Goldsmith proposed to deal with this issue by making the semantics of intransitive noninterference significantly more restrictive, in effect reverting to the purge-based definition of Goguen and Meseguer [1982, 1984]. Our approach to downgrading, using a filter function, provides an alternative approach that enables explicit specification of the information permitted to be released by the downgrader.

More recent work on downgrading has concentrated on downgrading in the setting of language-based security. Sabelfeld and Sands [2005] briefly survey recent work on downgrading in language-based settings, and propose several dimensions of downgrading, and prudent principles for downgrading. They regard intransitive noninterference as specifying *where* (in the security levels) downgrading may occur. Since we interpret security levels as system components, our architectures specify where in the system downgrading may occur. The filter functions that we propose in this work specify *what* information can be downgraded and *when* this may occur. Thus, our work combines the *what*, *where*, and *when* dimensions of downgrading. Recent work also considers multiple dimensions of downgrading, including Barthe et al. [2008], Banerjee et al. [2008], Mantel and Reinhard [2007] and Askarov and Sabelfeld [2007b].

Recent work [Askarov and Sabelfeld, 2007a; Banerjee et al., 2008] considers "knowledge-based" approaches to downgrading in language-based settings. However, they do not reason about security properties as general and application specific as used in this paper. O'Neill [2006] uses epistemic logic to specify many information security properties, but does not directly consider downgrading.

References

AADL. Architecture analysis and design language (AADL). SAE Standard AS5506/A, Jan. 2009.

J. Alves-Foss, W. Harrison, P. Oman, and C. Taylor. The MILS architecture for high-assurance embedded systems. *International Journal of Embedded Systems*, 2(3/4):239–47, Feb. 2006.

M. Anderson, C. North, J. Griffin, R. Milner, J. Yesberg, and K. Yiu. Starlight: Interactive link. *Annual Computer Security Applications Conference*, pages 55–63, 1996.

A. Askarov and A. Sabelfeld. Gradual release: Unifying declassification, encryption and key release policies. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 207–221. IEEE Computer Society, 2007a.

A. Askarov and A. Sabelfeld. Localized delimited release: combining the what and where dimensions of information release. In *Proceedings of the 2007 Workshop on Programming Languages and Analysis for Security*, pages 53–60. ACM Press, 2007b.

A. Banerjee, D. A. Naumann, and S. Rosenberg. Expressive declassification policies and modular static enforcement. In *Proceedings of the IEEE Symposium on Security and Privacy*. IEEE Computer Society, May 2008.

G. Barthe, S. Cavadini, and T. Rezk. Tractable enforcement of declassification policies. In *Proceedings of the 21st*

IEEE Computer Security Foundations Symposium. IEEE Computer Society, June 2008.

D. Basin, J. Doser, and T. Lodderstedt. Model driven security: From uml models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology*, 15(1):39–91, 2006.

D. Bell and L. La Padula. Secure computer system: unified exposition and multics interpretation. Technical Report ESD-TR-75-306, Mitre Corporation, Bedford, M.A., March 1976.

C. Boettcher, R. DeLong, J. Rushby, and W. Sifre. The MILS component integration approach to secure information sharing. In *Proceedings of the 27th IEEE/AIAA Digital Avionics Systems Conference*, pages 1.C.2–1–1.C.2–14, Oct. 2008.

R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning About Knowledge*. MIT-Press, 1995a.

R. Fagin, J. Y. Halpern, Y. Moses, and M. Y. Vardi. *Reasoning about Knowledge*. MIT Press, Cambridge, MA, 1995b.

R. Focardi and R. Gorrieri. A classification of security properties for process algebras. *Journal of Computer Security*, 3(1):5–33, 1994.

D. Garlan, R. T. Monroe, and D. Wile. Acme: Architectural description of component-based systems. In G. T. Leavens and M. Sitaraman, editors, *Foundations of Component-Based Systems*, pages 47–68. Cambridge University Press, 2000.

J. A. Goguen and J. Meseguer. Security policies and security models. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 11–20. IEEE Computer Society, Apr. 1982.

J. A. Goguen and J. Meseguer. Unwinding and inference control. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 75–86. IEEE Computer Society, Apr. 1984.

D. Greve, M. Wilding, and W. M. Vanfleet. A separation kernel formal security policy. In *Proceedings of the Fourth International Workshop on the ACL2 Prover and Its Applications*, July 2003.

J. T. Haigh and W. D. Young. Extending the noninterference version of MLS for SAT. *IEEE Transactions on Software Engineering*, 13(2):141–150, 1987.

J. Hansson, P. H. Feiler, and J. Morley. Building secure systems using model-based engineering and architectural models. *CrossTalk: The Journal of Defense Software Engineering*, 21(9), Sept. 2008.

T. H. Hinke and M. Schaefer. Secure data management system. Technical Report RADC-TR-75-266, System Development Corporation, Nov. 1975.

J. Jürjens. *Secure Systems Development with UML*. Springer, 2005.

H. Mantel and A. Reinhard. Controlling the what and where of declassification in language-based security. In R. D. Nicola, editor, *Proceedings of the 16th European Symposium on Programming*, volume 4421 of *Lecture Notes in Computer Science*, pages 141–156. Springer, 2007.

D. McCullough. A hookup theorem for multilevel security. *IEEE Transactions on Software Engineering*, 16(6):563–568, 1990.

J. McLean. A general theory of composition for a class of "possibilistic" properties. *IEEE Transactions on Software Engineering*, 22(1):53–67, 1996.

K. R. O'Neill. *Security and Anonymity in Interactive Systems*. PhD thesis, Cornell University, Aug. 2006.

A. W. Roscoe and M. H. Goldsmith. What is intransitive noninterference? In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*. IEEE Computer Society, 1999.

J. Rushby. Noninterference, transitivity, and channel-control security policies. Technical Report CSL-92-02, SRI International, Dec 1992.

A. Sabelfeld and D. Sands. Dimensions and principles of declassification. In *Proceedings of the 18th IEEE Computer Security Foundations Workshop*, pages 255–269. IEEE Computer Society, June 2005.

B. M. Thuraisingham. *Database and Applications Security: Integrating Information Security and Data Management*. CRC Press, 2005.

R. van der Meyden. What, indeed, is intransitive noninterference? In *Proceedings of the 12th European Symposium On Research In Computer Security*, volume 4734 of *Lecture Notes in Computer Science*, pages 235–250. Springer, Sept. 2007.

R. van der Meyden. Architectural refinement and notions of intransitive noninterference. In *Proceedings of the 1st International Symposium on Engineering Secure Software and Systems*, volume 5429 of *Lecture Notes in Computer Science*, pages 60–74. Springer, Feb. 2009.

W. Vanfleet, R. Beckworth, B. Calloni, J. Luke, C. Taylor, and G. Uchenick. MILS:architecture for high assurance embedded computing. *Crosstalk: The Journal of Defence Engineering*, pages 12–16, Aug. 2005.

Lessons Learned While Building A High Assurance Smart Card Operating System

P.A. Karger (deceased), S.K. McIntosh,
E.R. Palmer, D.C. Toll
IBM T.J. Watson Research Center
1101 Kitchawan Road,Yorktown Heights, NY 10598
+1-914-945-3000
skranjac, erpalmer, toll@us.ibm.com

Samuel Weber
National Science Foundation
4201 Wilson Blvd., Suite 1175
Arlington, VA 22230
+1-703-292-7096
sweber@nsf.gov

This paper has been previously published as Karger, McIntosh, Palmer, Toll, and Weber, 2011. Lessons Learned Building the Caernarvon High-Assurance Operating System, *IEEE Security & Privacy*, (Jan/Feb 2011).

Dynamic Cascade Vulnerability Checks in Real-World Networks

Rachel Craddock, Adrian Waller, Noel Butler, Sarah Pennington
Thales UK Research and Technology
Worton Drive, Reading, UK.
{Rachel.Craddock, Adrian.Waller, Noel.Butler, Sarah.Pennington}@uk.thalesgroup.com

David Llewellyn-Jones, Madjid Merabti, Qi Shi, Bob Askwith
School of Computing and Mathematical Sciences
Liverpool John Moores University, Liverpool, UK
{D.Llewellyn-Jones, M.Merabti, Q.Shi, B.Askwith}@ljmu.ac.uk

ABSTRACT
When networked systems that have been accredited to a certain level of security interact there is the potential for data compromise to occur with greater ease than for a single system accredited to the same level. Detecting such occurrences is known as the Cascade Vulnerability Problem. The problem is well known, and many algorithms have been proposed to detect and correct it. However, these algorithms tend to be centralised and require complete knowledge of the entire network to succeed, making them largely unsuitable for use within real networks, especially those dynamically created between multiple agencies. In this paper we apply the dynamic cascade vulnerability detection algorithm to a real Multi-Level Secure network and the SODA policy negotiation architecture. We show for the first time that the dynamic algorithm is practical and effective at preventing cascade vulnerabilities in a real Multi-Level Secure network.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General – *Security and protection (e.g., firewalls)*

General Terms
Algorithms, Management, Design, Experimentation, Security, Theory, Verification.

Keywords
Network security, Multi-Level Secure networks, Systems-of-Systems, Cascade Vulnerability Problem.

1. INTRODUCTION
The Cascade Vulnerability Problem (CVP) is a well-known security vulnerability that can occur in Multi-Level Secure (MLS) systems that interact across a network. The problem arises due to the way such secure systems are assured based on the security level of the data held on them.

In the most common formulation of the problem, data is assigned a security level from a well-ordered set of accreditations (*e.g.* TS, S, C, N, U). When used correctly data can be upgraded freely but cannot be downgraded. However an MLS device may store data of more than one level. Should the device become compromised in some way, an attacker may therefore be able to improperly reduce the security level of the data stored on the device to one of its lower levels. For each device a risk is associated with this event occurring (the greater the possible downgrade the more serious the potential damage caused, and therefore the greater the risk). To mitigate this risk, devices storing more than one security level must be assured to a certain level. A device with higher assurance is considered harder to compromise, and hence we can assign a difficulty which should be inversely proportional to the risk for each device.

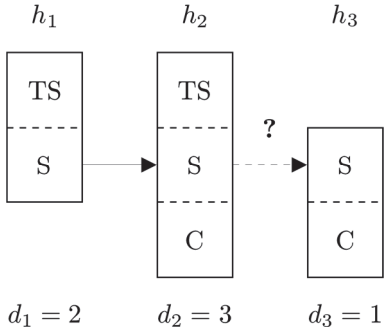


Figure 1. A simple cascade vulnerability.

This configuration works successfully for devices in isolation, but can fail when we connect devices together across a network, even when the network itself is secure. The reason for this is that the difficulty for an attacker to compromise multiple devices in order to lower the classification of some data can often be less than that required to lower the classification on a single device. The accreditation procedure – which generally considers only single devices – will therefore fail to provide adequate protection in a network setting.

Table 1. Security Difficulties Index Matrix.

		Max Data Sensitivity						
		U	N	C	S	TS	1C	MC
Min Clearance	U	0	1	2	3	4	5	6
	N	0	0	1	2	4	5	6
	C	0	0	0	1	3	4	5
	S	0	0	0	0	2	3	4
	TS	0	0	0	0	0	2	3
	1C	0	0	0	0	0	0	1
	MC	0	0	0	0	0	0	0

To illustrate this, consider the example shown in Figure 1. According to the Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments ("Yellow Book") [1, 2], the difficulty of compromising a machine assured to store TS and S level data such as h_1 in Figure 1 must be at least of difficulty of 2 (see Table 1). For a machine assured to store S and C level data such as h_3, the difficulty is 1. The difficulty of compromising both machines in Figure 1 is therefore the maximum of these, which is a difficulty of 2 (since it's assumed that any attacker sufficiently skilled to compromise a machine with difficulty 2 can do the same for a machine of difficulty 1). These difficulty levels are closely associated with risk, since the difficulty level must be commensurate with the level of risk. The risk requirements for defence systems are generally set in practice using a well-defined mandatory (and often classified) process, which ends up with discrete numerical security levels being defined (similar to the Common Criteria [3]). For devices evaluated to a given security level, both the difficulty and risk can be considered as related via this level as shown in the table.

So, with reference to Figure 1, an attacker could reduce the level of data from TS to C by downgrading it from TS to S on h_1, transmitting it to h_3 via h_2 and then downgrading it to C by compromising h_3, all of which can be achieved with difficulty 2. However, a single TS-S-C machine such as h_2 on its own must be accredited to difficulty level 3. Hence it is easier to compromise the networked system then would be required by the TCSEC criteria for a single isolated machine.

We note that this can be seen as an oversimplification of the problem. For example, different levels of compromise of the same machine can have wildly different consequences. Moreover, in practice risk levels are more likely to conform to a probability distribution than a discrete integer ordering, and may depend on a number of factors. Refinements that incorporate such ideas into risk classification systems have been discussed in the literature [4]. However in this paper we focus on the existing categorisation approaches, which reflect the reality of how systems are currently classified. These rather rigid approaches have other drawbacks too, such as the difficulty of assigning a coherent scale across organisations. Some of these issues we we will return to later (*e.g.* in Section 3.3).

Even for the simplified approaches in predominant use today, the CVP remains a problem. Various solutions for the CVP have been proposed, including methods to avoid, detect and correct problems that might result in such vulnerabilities within a network. Detection methods include the Fitch-Hoffman algorithm and the Horton algorithm [5], while correction methods include Fotakis and Gritzalis's combinatorial approach [6], Gritzalis and Spinellis's simulated annealing approach [7] and the soft constraint-based approach of Bistarelli *et al.* [8].

A notable characteristic of all of these approaches is that they are centralised and require complete knowledge of the network. In other words, the algorithms must be applied to the network as a whole, meaning the structure and nature of the network must be known in full at the point where the algorithm is applied. In many real network situations, such strong centralised omniscience may be neither desirable nor even possible.

In response to this, we (the authors) have developed a distributed algorithm that can detect cascade vulnerabilities in a distributed manner, with detection occurring at the point of connection of a new link in the network [9]. This algorithm has the benefit that no centralised control is required, and also that vulnerabilities can be detected using only local information at the point of connection, allowing new links that will cause a cascade vulnerability to be safely refused in an efficient way.

Although in our previous work we demonstrated the mathematical correctness of the algorithm, the equally important implementation requirement has not yet been considered. In this paper we therefore discuss the process of implementing the technique in a real world network using IP-based hardware encryption devices. We also present results to demonstrate its applicability in such real networks.

To our knowledge no previous cascade vulnerability algorithms have been integrated into any real network systems. This work therefore represents an important and novel advance in the development of realistic solutions for preventing cascade vulnerabilities from appearing in networks. Moreover, the implementation process highlighted a number of further considerations that have not until now appeared in the literature.

The remainder of this paper is structured as follows. In the next section we provide a general overview of our distributed CVP detection algorithm. In Section 3 we discuss our implementation using the novel SODA system used to implement our algorithm, as well as some discussion about the practical issues involved in implementation. We provide simulation results in Section 4 and conclude with a discussion about future work in Section 5.

2. Detection Overview

In this section we present a brief overview of the CVP detection algorithm. Since our intention is to focus on the practical application of the process we only present a high-level overview here; for the full details, please see the authors' publication [9].

Most CVP detection methods are centralised, requiring the algorithm to be applied to a model of the network of systems that captures its global characteristics. Consequently, a practical detection method would require some centralised system to have up-to-date locally-stored topology information about every system on the network. In practice, this may not be realistic, and so here we present our alternative – distributed – approach that weakens this requirement.

Our approach requires each system on the network to maintain a limited amount of data that captures the nature of the links entering and leaving the system. We call these the *input* and *output tables* for the system. Each table is an $n \times n$ matrix of difficulty values (sometimes presented in terms of *risks*), where n is the number of possible classification levels the network supports. We then define three processes to manage the data stored in these tables: an *initialisation procedure*, an *update procedure* and a *decision procedure*.

2.1 Initialisation Procedure

This simple step sets the initial values that should be stored in the tables after a system is initialised, and before it makes any connections with other systems on the MLS network.

Each cell in the input table represents the relationship between the level of the system storing the table, and a level of some system connected to it (either directly or indirectly through other systems) on an incoming link. The initial entries are set up according to the standard risk matrix used across the network (*e.g.* such as that shown in Table 1). However, where a certain downgrade or upgrade is impossible (*e.g.* because data of a certain level is never stored on a system) a special value indicating that the situation

cannot arise is entered into the table (represented here by a dash). For system h_2 in Figure 1 then, the input table would be as shown in Table 2. The output table is initialised similarly.

2.1.1.1 Update Procedure

When a node makes an outgoing connection to another node, the two connecting systems must first exchange input and output tables, as shown in Figure 2.

Each system then performs a merge on their tables as follows. Suppose d represents the difficulty of compromising the node h_1 making the outgoing connection (as taken from Table 1), and s is the sensitivity level of the proposed new link. We consider the input table for the system being connected to, shown as h_2 in Figure 2.

Each entry in the input table has a minimum clearance level (row) and a maximum sensitivity level (column) at location x, y. For each of the systems h connecting in (*i.e.* where there is an input link from h into h_2), we have a copy of their input table. For each such link, if $s > y$ we calculate m, the maximum value out of the two values d and the entry at position x, s in the input table for h. If $s \leq y$ then we simply set m equal to the value at position x, s in the input table of h. This gives us a value m for each of the incoming links. Taking the minimum of these values gives the new entry to store in the input table for system h_2.

The update procedure for the output table is the same, but with input tables/links replaced with output tables/links respectively.

Any changes to the tables that result must be back-propagated. That is, if any system is connected to h_2, then h_2 must send the changes to them so they can perform the same update process. Similarly, any changes must be propagated by h_1 to systems that it connects to. These messages need only be propagated as far as the calculations continue to result in changes to the tables: if the table does not change, no further propagation is needed.

This process ensures the tables throughout the network are kept up-to-date. Since the maximum and minimum operations are associative, in situations where multiple updates are occurring in the network, the system will eventually converge to the same state, independent of the order the updates are processed in.

2.2 Decision Procedure

The first step in the update process described in the previous section is for connecting systems to exchange input/output tables. Once these tables have been exchanged, but prior to the connection being confirmed (and therefore also before invoking the update process), each system performs a simple check using the two tables.

Suppose we assume the situation shown in Figure 2 again, with a link being created at level s. For each possible pair of sensitivity levels x and y, we calculate the maximum of the entry at position x, s in the input table and the entry at position s, y in the output table (a total of n^2 comparisons). If any of these generates a result less than the corresponding value at x, y in the difficulty matrix (Table 1) then this tells us a CVP *would* be triggered. Otherwise, the link is safe. In effect, this calculates the minimum difficulty of performing a downgrade using the existing links and the new link within the network. In order to avoid a CVP we require this to be lower than the pre-specified risk requirements, as discussed in the previous section.

Table 2. Example input table.

		Max Data Sensitivity						
		U	N	C	S	TS	1C	MC
Min Clearance	U	-	-	-	-	-	-	-
	N	-	-	-	-	-	-	-
	C	-	-	0	1	3	-	-
	S	-	-	0	0	2	-	-
	TS	-	-	0	0	0	-	-
	1C	-	-	-	-	-	-	-
	MC	-	-	-	-	-	-	-

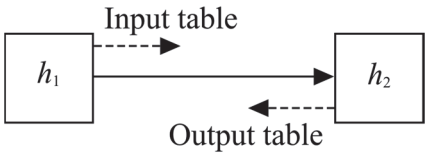


Figure 2. Output and Input table exchange.

Depending on the outcome of this calculation, either no CVP is detected – in which case the link can be confirmed and the update process can commence – or the link is refused and no further action is needed.

By performing this simple check, and managing the input and output tables using the initialisation and update processes, situations that would result in a CVP can be avoided. The check is both necessary and sufficient, ensuring links need never be refused unnecessarily.

3. Practical Issues

3.1 Secure On-Demand Architecture (SODA)

In order to make use of our proposed CVP detection algorithm in practice, we have incorporated it into our SODA architecture. SODA was originally developed as part of the EC SUPHICE project [10], but has since been extended. It is a novel architecture that allows dynamic, policy-based connections to be made semi-automatically between high assurance networks using hardware crypto devices to encrypt channels. SODA provides a rules-based approach to connection authorisation, with human administrators being able to define rules in an easily understood format, and being provided with the flexibility to change rules quickly as required.

We have extended SODA by including CVP checks into the security authorisation process. The implementation in SODA is in fact independent of any specific CVP detection algorithm, but has been intended for the algorithm presented above. SODA is described in the following text for authorising connections between networks via IP crypto devices. However, it is applicable for authorising any kind of connection between systems.

The basic process used in SODA (without CVP checks) to set up a secure connection between two networks is as shown in Table 3.

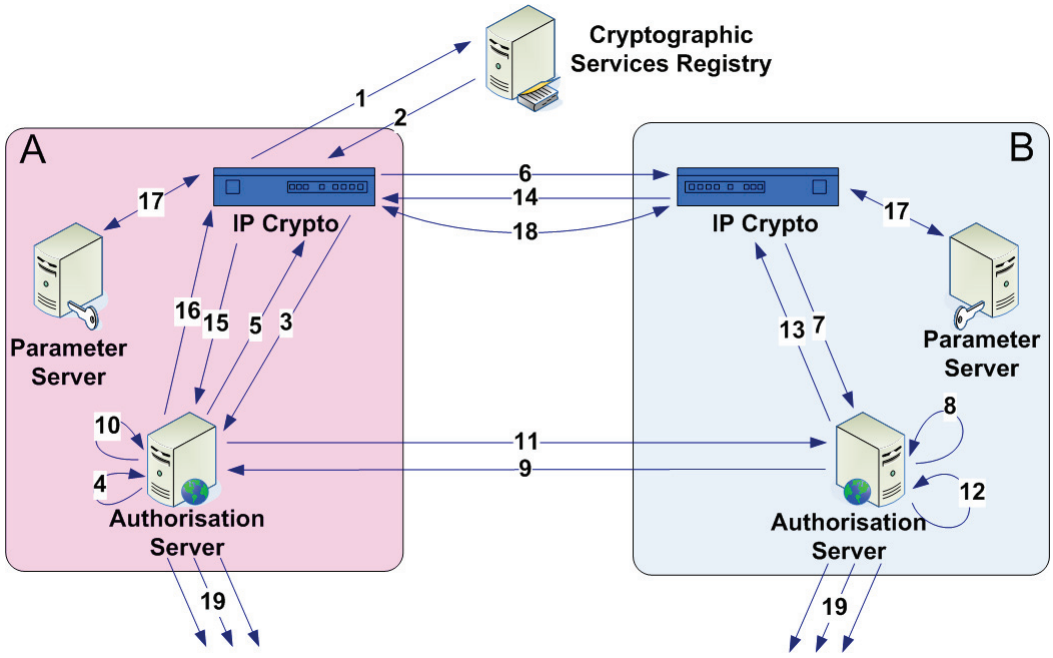


Figure 3 – SODA architecture.

Table 3 – SODA interactions.

#	Details
1	Search for details of available parties which meet certain criteria.
2	List of available parties, with security attribute information.
3	Request authorisation to connect.
4	Decision made about A connecting to B only.
5	Decision result, with any conditions.
6	If decision #5 is a 'yes', connection request sent to B, with conditions and address of A's authorization server.
7	Request authorisation to connect.
8	Decision made about B connecting to A only.
9	Connection request sent from B to A, including information required to determine a CVP.
10	Decision to accept/reject made based on CVP algorithm.
11	If decision #10 is 'yes', then information required to determine a CVP is sent to B's Authorisation Server. If 'no', then a reject message is sent.
12	Decision to accept/reject made based on CVP algorithm.
13	Decision provided to IP Crypto B.
14	Decision provided between the two parties.
15	If a 'yes', check with local Authorisation Server for permission to get connection parameters.
16	Decision from #10 provided to the crypto
17	Parameters obtained from the Parameter Servers
18	Secure connection made.
19	Notify other networks of the new connection (CVP algorithm 'Update' procedure).

When requested by a network (*A*), a Cryptographic Services Registry is used to provide details of available networks which meet certain criteria. Once a particular network (*B*) has been found, *A* can then determine whether it is possible to securely connect to *B*. The Registry provides appropriate attribute information about *B*, which *A* can then check against the policy rules contained within its local Authorisation Server. The Authorisation Server uses an embedded rules engine to process such an authorisation request, producing an 'accept', 'reject' or 'refer' decision (*i.e.* refer to a human operator for a decision). If *A*'s local Authorisation Server returns an accept decision, then a connection request is sent to *B*, containing *A*'s attributes. *B* then performs a similar process, and checks *A*'s attributes with its own Authorisation Server. If *B*'s Authorisation Server returns an accept decision, *i.e.* both networks agree to set up a secure connection, then each network's Parameter Server provides appropriate algorithms and certificates to its IP crypto and the connection is established.

This process does not take into account the effect of any networks which are already connected to either *A* or *B*. One of the security implications of these additional networks can be checked using CVP detection. The Authorisation Servers in each domain provide a mechanism for the exchange of information required for CVP detection, along with a processing location for the CVP detection module. In the SODA implementation, potential CVPs are checked after each network's Authorisation Server has returned an accept decision. The information required to detect a CVP is exchanged between the two networks, and each determines if it is safe to connect to the other network and its associated networks. The overall process of determining if the two networks can securely connect, followed by checking for a CVP, is shown in Figure 3, with the information exchanges between the different components presented in Table 3.

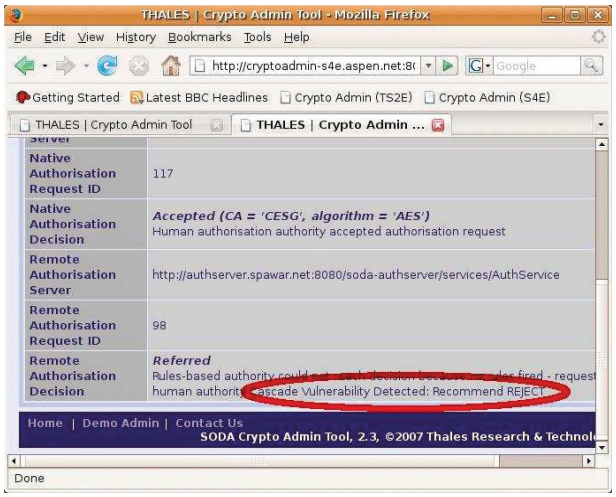


Figure 4 – SODA interface on CVP detection.

The SODA Authorisation Servers are currently implemented as web services running on a web server. A CVP detection module is integrated into the servers, implemented using Java web servlets with the Apache Tomcat web server. This module establishes connections between Authorisation Servers for the transmission of the required information (interactions 9 and 11 in Figure 3). Once the information has been exchanged, the servers determine whether or not the new connection will result in a CVP or not. Note that this exchange of information is relayed between the two IP Crypto devices, rather than requiring a new direct connection as such (omitted from Figure 3 for clarity).

In our implementation, the result of the CVP algorithm (steps 10 and 12 in Figure 3) is combined with the result of the existing policy negotiation and relayed to the operator via the SODA interface. In the case where a CVP is detected, the Authorisation Server provides a 'refer' response to the operator, indicating that the connection is unsafe. Since the decision procedure is symmetric, the result will be shown identically at both ends of the link. Figure 4 shows the result as it is displayed to the operator in the event that a CVP is detected. By referring the decision to the operator, this allows the CVP detection to be overruled and the connection made "at risk" in exceptional circumstances. A policy could be written to automatically reject connections if a CVP is detected, but the current approach provides more flexibility.

3.2 Concurrent Updates

At the moment, our implementation does not consider situations in which a node addition or removal is attempted before an in-progress update procedure has finished. Such situations will lead to multiple updates being propagated throughout the network at the same time. The order in which these are processed does not affect the final state reached, due to the associative nature of the maximum and minimum operators in the update equations. However, care is required to manage the update process, as further changes occurring before current updates have finished may result in the detection algorithm producing incorrect results. If the in-progress change is a node addition, then this could result in a CVP being added to the network, and if the in-progress change is a node removal, then this could result in the rejection of a new link which could have been allowed. We consider two main approaches to this problem.

3.2.1 Allow Concurrent Updates

Firstly, continue to allow updates to propagate concurrently (*i.e.* the current implementation). In this case, requests to connect a node which are rejected or approved can be revisited after an update has been received shortly afterwards. As mentioned above, the Input and Output tables will, eventually, correct themselves, allowing an incorrect decision to be corrected. For a short time an unsafe connection may be in place, but the probability of an attacker being able to exploit such a CVP in this time period will generally be very low. In addition, if the network is fairly static, then such overlapping updates are likely to be a rare occurrence in any case.

3.2.2 Prevent Concurrent Updates

If a network is highly dynamic, and/or the risk associated with unsafe connections is very high, a second approach of preventing concurrent updates may be preferable. One way of achieving this is described as follows.

3.2.2.1 The Update Manager

An Update Manager is used to 'lock' updates to prevent concurrent updates. If a connection is being added or removed from the network, the affected node first initiates an update procedure by asking the Update Manager for permission to perform an update. If no updates are occurring, then permission is granted. The CVP check and update process commences and the node which initiated the process informs the Update Manager when the process has completed. If an update was already in progress, the Update Manager denies the request and adds the node to the update queue. In either case, once the update process has finished, the Update Manager grants permission to the next node in the queue, or if the queue is empty, to the next update request that it receives. The security implications of a single Update Manager are to be discussed in 3.2.2.6.

3.2.2.2 Determining that an update process has completed

The question arises regarding how a node can know that an update process it initiated has completed. To achieve this, the solution exploits the recursive nature of the update process. In its simplest terms, when a node updates its tables, the update process includes passing on the updates to its up- or down-stream nodes (depending on whether this is an Input or Output table update). When the update reaches a node whose tables do not change, that update path is considered to have completed and a completion notification sent back along the path. As each node in the path receives a completion notification from all the nodes it sends an update to, the process eventually propagates the completion notifications back to the initiating node, which then informs the Update Manager.

In order to ensure the recursive update completes correctly, each node needs to maintain records of the updates it has received and sent, and of the completion messages it has received. By doing this, it can ensure that all updates it has sent have completed, and that when it has received all completion messages for a particular update it knows where to send its own completion message. For each update received by a given node, the records maintained by that node need to include the following:

- Global update identification number – This number is globally unique and is generated at the initiating node.
- Local update identification number – This number is locally unique to the specific node, and distinguishes

between different update messages received for a particular global update identity.

- Identity of the node which sent it this local update.
- Identity of the nodes to which this local update is sent (and hence the number of such nodes).

The update unique identification number is included in completion messages so that a node knows which updates have completed and which have not.

Authentication between systems and the centralised server is needed in order to avoid malicious nodes spoofing this identification number. If concurrent update messages can no longer be trusted, the approach effectively reduces to the situation described in Section 3.2.1. However, it's worth bearing in mind that for deployments of systems such as those described here, all nodes will tend to belong to a single – or a set of trusted – organisations. There is an inherent assumption about trust between nodes which would not be the case in a less restrictive environment, which for example means we can assume nodes aren't malicious unless already compromised.

Even in a trusted network environment authentication is crucial to avoid external attack. However, the issue of how authentication can be achieved remains part of our future work.

3.2.2.3 Dealing with unexpected disconnections

Official disconnections will ask the Update Manager for permission before proceeding, but unexpected disconnections (node or link) can also occur (*e.g.* due to technical failure). If a link is lost, the nodes at either end of that link detect this and inform the Update Manager. If no update is currently taking place, the network is locked while an update for the lost link takes place. If an update is in progress, then the Update Manager will maintain the network 'lock' once that update has finished, so that the nodes at either end of the lost link can then initiate the required new updates for the lost link. The network will be unlocked once these updates have completed.

If the link which has been lost is on a path which is currently being updated, then the node which sent an update message via that link will not receive a completion message from the node at the other end of the link. In such situations, the update algorithm acts as follows:

Consider the case where an update message was sent from node *A* to node *B*. The link between these two nodes is then lost, so node *B* cannot send a completion message to node *A*. Both nodes *A* and *B* inform the Update Manager that they have lost a link. Node *A* treats the link loss as if the path between *A* and *B* had not existed and thus it had not sent an update message to node *B* (this is a valid thing to do as that path now terminates at node *A*). It updates its records accordingly. Node *B* informs the Update Manager of the status of its current update process, either in-progress or complete. If the update is in-progress, the Update Manager will maintain the network 'lock' while it waits for a completion message from node *B*. Once the Update Manager receives a completion message from node *B* and receives a completion message from the node which initiated the update, it then triggers the lost link updates from nodes *A* and *B*. Only when these updates have completed is the network lock released.

If a node is lost, the multiple links connected to this node will have been lost. Nodes which have lost a link as a consequence of the node loss inform the Update manager of the link loss. The update algorithm then proceeds as described for the link loss situation. It does not matter that the gap between the remaining nodes is longer than a single link and the algorithm will still work in cases where two or more consecutive nodes on a path have been lost.

3.2.2.4 Discovering that a link has been lost

If an unexpected disconnection has occurred, a node will discover that link has disappeared when it does not receive a response to a request. Networks can be set up to include periodic connectivity checks between nodes. When a node discovers that a link has disappeared, it can contact the Update Manager as described above. However, the question remains as to how long a node should wait for a response, as network traffic load and the link latency can cause delays in response. In addition, some responses will not be immediate due to processing required in order to produce the response. The actual length of time to wait will be scenario dependent and is a known networking issue irrespective of the application described. It is considered out of scope here.

3.2.2.5 Dealing with simultaneous updates

Due to there being multiple paths through the network, it is possible for a node to receive several different updates simultaneously during the same update process. In such a situation, it must store all of these and process them sequentially. The order in which the updates are processed does not matter, due to the associative nature of the maximum and minimum operators in the update equations. Propagation of the updates from a node can wait until all the pending updates have been processed on that node. The node updates its matrices for each of the received updates, and only propagates these changes after the last matrix update has finished. This minimises the number of update messages the node has to send, and therefore reduces the overall duration of the update process.

3.2.2.6 Future work

The above process has not been implemented and a number of practical issues remain to be resolved. The main problem is that it assumes that all nodes have common access to an Update Manager. This may be possible through an out-of-band channel for example, separate to the main network connections being controlled by the CVP algorithm. As the information required to be sent to the Update Manager is simply a 'lock' request or release, the confidentiality requirements are low for this, as are any bandwidth requirements. However, careful consideration as to the integrity and availability of this channel would be needed to prevent a possible denial-of-service attack. A more likely alternative is for the Update Manager to be communicated with via the existing network(s). The problem with this approach is that as networks are connected and disconnected (perhaps unexpectedly) multiple Update Managers will need to be combined into one, or vice versa. To prevent a single point of failure, multiple, but coordinated, Update Managers may be needed for redundancy reasons in any case. This ought to be achievable, but the details of this are left as future work.

Currently our implementation does not address the issue of concurrent updates, making it difficult to predict the overhead of introducing a centralised update manager. However, our proposed design requires only minimal (effectively just a flag and identifier) information to be passed between nodes and to the update manager, so we expect the overhead to be relatively low.

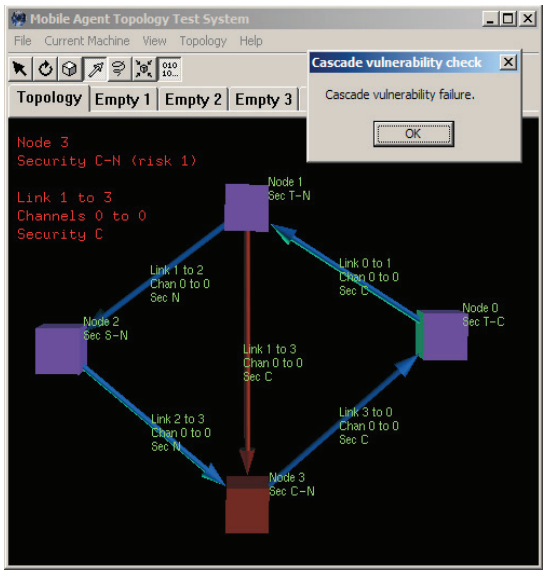


Figure 5. MATTS refuses cascading links.

3.3 Other Practical Issues

The standard description of the CVP assumes a common interpretation of the security policy, a common set of security levels *etc.*, and is mostly appropriate within one organization or nation, albeit with different security levels. However, a major application area is for managing the CVP in a coalition network involving multiple organizations and/or nations. In this case, the security policy to be applied, definition of levels etc. is likely to vary between the partners. One approach could be to create a common set of security levels, however, the policy to apply is unlikely to be the same amongst partners. It may be possible to adapt our algorithm to cope with multiple policies, and we are considering this as future work.

Another problem is that each node in the network will have access to information about the network as a whole based on the contents of its Input and Output tables. How much information can be inferred in this way, and how significant an issue this could be, is not at all obvious at present, and requires further investigation. This problem does not exist for centralized approaches, which assume a highly trusted central point that has visibility of the entire network. However, these approaches would have a similar, and perhaps worse, problem in a coalition setting, as the central point may not be trusted to maintain information about the entire network.

4. Simulation Results

While the implementation of our approach on the SODA architecture demonstrates its practical feasibility, to understand the performance characteristics of the technique requires a larger network testbed than we were able to deploy in practice. Consequently we also ran the process over a number of simulated test scenarios in order to generate performance results.

This was undertaken using the MATTS topology test system [11]. The software allows networks of components to be built up graphically for the purposes of testing secure component composition results. The CVP check is performed whenever an attempt is made to create a new link between components. If the check fails, the link is refused as shown in Figure 5. Whilst this allows us to demonstrate the accuracy and applicability of the method, we were also able to run timing tests to establish the effect of increasing the size of network in terms of both nodes and links. To provide a comparison, we also implemented the technique presented by Horton et al.[5] for the testing of a complete network. It is important to consider the differences between the two techniques: the Horton algorithm tests a complete network, whereas our proposed technique tests only the viability of adding a new link to the network. To provide a fair comparison, we therefore build up a random network using our technique to guarantee at each step that no vulnerability is introduced. What we are left with is a cascade-free network. Having done this, the resulting network is then re-tested using the Horton algorithm.

Our initial tests involved producing 1000 random graphs comprised of between 30 and 50 nodes, and between 100 and 500 links. Half of the graphs were built using our dynamic method to ensure a cascade-free graph. The remaining half were built similarly, but with one link *failing* our dynamic cascade vulnerability check. These graphs consequently contained at least one cascading path. The dynamic test and the Horton test results concurred in all cases, demonstrating the accuracy of our dynamic method in producing the same results as the Horton algorithm. In all of our experiments we applied the Security Risk function determined by the values as shown in Table 1, taken from the US Department of Defense's "Yellow Book" [2].

We then ran a number of further tests, generating cascade-free and vulnerable graphs using the same method as described above, to compare the time taken to test the graphs using the two methods. Various graphs with sizes varying from 20 nodes and 100 links to 50 nodes and 500 links were tested. The results of these timings, taking averaged results of five different randomly generated graphs, can be seen in Figure 6 and Figure 7. All tests were performed on an Intel XScale 80321 ARM powered device clocked at 600 MHz. The exact timing results are shown in Table 4 and Table 5.

As can be seen from the results represented in Figure 6, fixing the number of links and increasing the nodes in the network does not increase the checking time for our algorithm. Indeed we find that checking time decreases slightly as the number of nodes increases. In fact, this is not particularly surprising, since the algorithm works by following links that emanate from each new link added. Increasing the number of nodes has the consequence of decreasing the link density, thereby decreasing the proportion of paths that need checking.

This compares favourably to the Horton algorithm, which requires three nested iterations through every node. The algorithm is therefore cubic in the number of nodes, which is reflected in the increase shown in Figure 6. It is also worth noting the considerable increase in speed demonstrated by our algorithm. Although we note that the implementations of both of the algorithms were created with accuracy rather than optimisation in mind, the difference between the two was far greater than we had anticipated.

Fixing the number of nodes in the network and varying the number of links shows a different pattern. With reference to Figure 7 we can see that for our algorithm the analysis time increases linearly (or possibly slightly worse than linearly) as the number of links increases. Considering the two sets of results for increasing nodes and increasing links, we consider the result that analysis time increases linearly with size of graph to be plausible. For the Horton algorithm increasing the number of links also

increases analysis time, but this appears better than linear, dropping off as the proportion of links to nodes decreases. Once again, however, it is worth noting the considerable difference in analysis time required between the two algorithms. This is especially stark when we consider the different ways in which the algorithms work. For our dynamic algorithm, the analysis process is applied every time a link is added to the network. For the graphs generated for the results therefore, our algorithm would have been applied multiple times (500 times in the case of a graph with 500 links), whereas the Horton algorithm is applied only once when the graph has been completely generated.

In normal use, we would expect our algorithm to be applied only once whenever a new network link is created, such as when a new device is added to the network. It is useful therefore to consider the time required to apply the algorithm to an existing network when adding just a single link. To this end, we generated a number of relatively large cascade-free networks consisting of 1000 nodes and increasing links. The processing time required to test for a CVP as a new link is added was then measured. For each size of network we generated 10 distinct instances and added 500 new links to each. The resulting averaged time for adding a single link to the network is shown in Figure 8 and Table 6.

As we can see from these results the algorithm produced fast times, increasing linearly with the size of the network. Based on the calculated complexity of the algorithm this is not unexpected. In any real world scenario, this analysis time is negligible and likely to be too small to be relevant, especially as the update mechanism of the algorithm potentially occurs distributed across multiple machines. However, because of this distribution, the time required to coordinate the algorithm across the network is likely to be far more important, and this has not been taken into account in the results. The impact of such considerations is heavily dependent on the structure of the network.

5. Conclusion

In this paper we presented the application of our distributed CVP detection algorithm to the existing SODA on-demand policy-based network security architecture. To the best of our knowledge the process has never been applied in a real network before, and represents novel work that demonstrates the practicality of the approach for real-world systems. In addition, we also presented new simulation results to show the efficiency of the process and justify its scalability and applicability for larger networks.

The results are promising, both in terms of the implementation and the scalability of the results. Our algorithm was found to generate the correct results far more efficiently than the equivalent Horton algorithm.

However, a number of areas remain to be resolved for the refinement of the process, and which represent future work. In previous sections we discussed the need to tackle multiple updates simultaneously, or alternatively to prevent checks occurring mid-update while the input and output tables are in an inconsistent state, potentially generating an incorrect result. While we presented a number of potential solutions, we have yet to implement and test these using the SODA system. In addition, some further practical issues relating to the reconciliation of distinct policies when attempting to join multiple MLS systems together were also discussed.

These represent interesting areas for future work, and which we hope to pursue through the further extension of the SODA implementation described here.

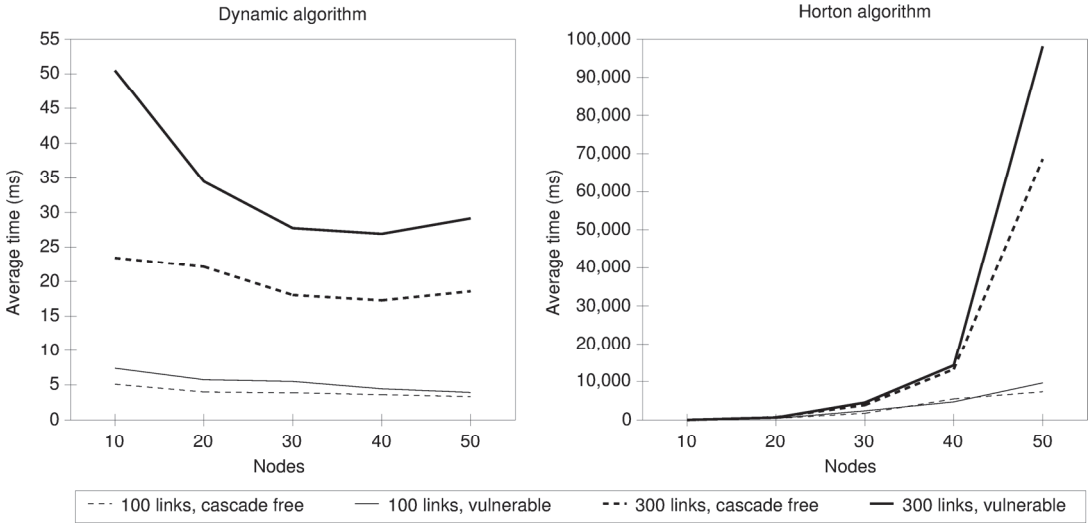


Figure 6. Analysis time as the number of network nodes changes.

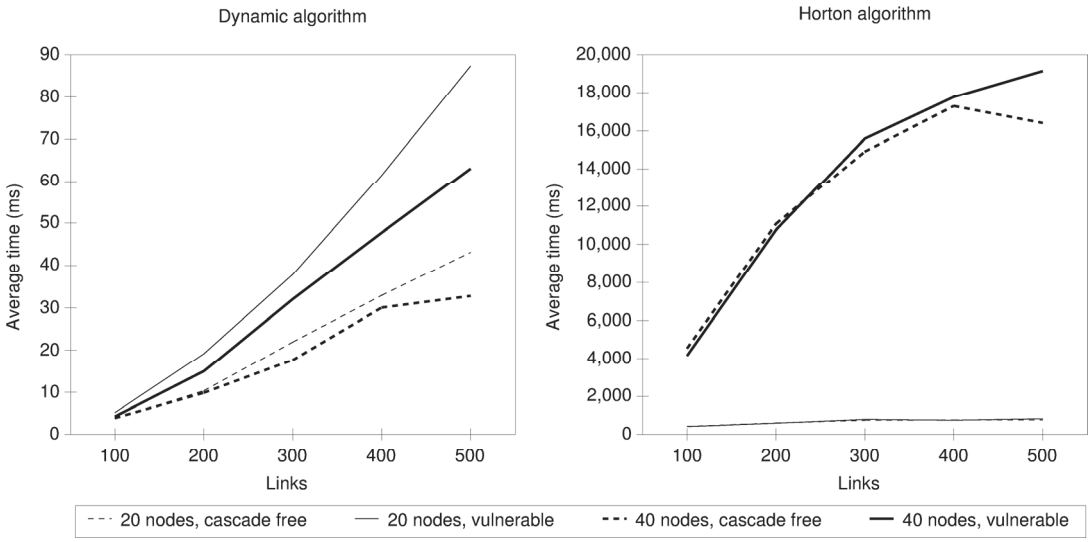


Figure 7. Analysis time as the number of network links changes.

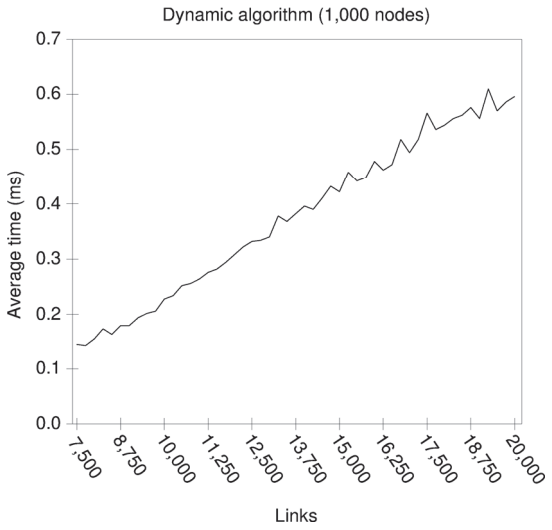


Figure 8. Analysis time for adding a new link to a network.

Table 4. Average analysis time (ms) as the number of network nodes changes.

	Dynamic algorithm				Horton algorithm			
	100 links		300 links		100 links		300 links	
Nodes	Safe	Vulnerable	Safe	Vulnerable	Safe	Vulnerable	Safe	Vulnerable
	5	7	23	50	23	23	28	32
	4	6	22	34	424	424	713	648
	4	6	18	28	1751	2375	3872	4540
	4	4	17	27	5456	4707	13160	14188
	3	4	19	29	7320	9629	68660	98158

Table 5. Average analysis time (ms) as the number of network links changes.

	Dynamic algorithm				Horton algorithm			
	20 nodes		40 nodes		20 nodes		40 nodes	
Links	Safe	Vulnerable	Safe	Vulnerable	Safe	Vulnerable	Safe	Vulnerable
	4	5	4	4	422	410	4551	4163
	10	19	10	15	602	592	11069	10751
	22	38	18	32	745	795	14914	15592
	33	62	30	48	768	749	17296	17774
	43	87	33	63	768	826	16408	19143

Table 6. Average analysis time for adding a new link to a network of 1000 nodes.

Links in network	Time (ms)
8000	0.156
10000	0.228
12000	0.308
14000	0.396
16000	0.478
18000	0.544
20000	0.596

REFERENCES

[1] Guidance for Applying the DoD Trusted Computer System Evaluation Criteria in Specific Environments (CSC-STD-003-85: “Yellow Book”). Department of Defense, 1985.

[2] Technical Rational Behind CSC-STD-003-85: Computer Security Requirements -- Guidance for Applying the Department of Defense TCSEC in Specific Environments (CSC-STD-004-85: “Yellow Book II”). Department of Defense, 1985.

[3] Common Criteria Development Board, Common Criteria for Information Technology Security Evaluation, Version 3.1. 2012.

[4] T. M. P. Lee, “Statistical models of trust: TCBs vs. people,” in IEEE Symposium on Security and Privacy, 1989, pp. 10–19.

[5] J. D. Horton et al., “The cascade vulnerability problem,” in Journal of Computer Security, 1993, vol. 2, no. 4.

[6] D. Fotakis and S. Gritzalis, “Efficient heuristic algorithms for correcting the Cascade Vulnerability Problem for interconnected networks,” Computer Communications, vol. 29, no. 11, 2006.

[7] S. Gritzalis and D. Spinellis, “The cascade vulnerability problem: the detection problem and a simulated annealing approach for its correction,” Microprocessors and Microsystems, vol. 21, no. 10, 1998.

[8] S. Bistarelli, S. N. Foley, and B. O. Sullivan, “A soft constraint-based approach to the cascade vulnerability problem,” Journal of Computer Security, vol. 13, no. 5, pp. 699–720, 2005.

[9] D. Llewellyn-Jones, M. Merabti, Q. Shi, B. Askwith, A. Waller, and R. Craddock, “Efficient Avoidance of the Cascade Problem for Dynamic Systems-of-Systems,” Under review.

[10] Thales e-Security Ltd, SUPHICE: on-demand secure communications provision. Preparatory Action for Security Research, 2006.

[11] D. Llewellyn-Jones, M. Merabti, Q. Shi, and B. Askwith, “An Extensible Framework for Practical Secure Component Composition in a Ubiquitous Computing Environment,” in 2004 International Symposium on Information Technology (ITCC 2004), 2004.

Towards Formal Evaluation of a High-Assurance Guard

Mark R. Heckman Roger R. Schell Edwards E. Reed

Aesec Global Services, Inc.
Palo Alto, California

{mark.heckman, roger.schell, ed.reed}@aesec.com

ABSTRACT

A transfer guard built on a high-assurance multilevel secure (MLS) trusted computing base (TCB) must be a trusted subject with the capability to perform downgrades not otherwise permitted by the MLS security policy. Formal evaluations of MLS systems containing trusted subjects are complicated when the trusted subjects are evaluated as part of a monolithic TCB. While well-developed techniques of "TCB subsets" and "TCB partitions" for composing MLS systems exist, approaches for applying these techniques for reasoning about a guard downgrade policy are not as well-developed. If the trusted downgrade process must be evaluated as part of the TCB, an inability to adequately reason about the downgrade policy would make it difficult to reason about the policy implemented by the system as a whole. And if trusted subjects cannot be evaluated separately and composed with the underlying TCB, that could lead to to expensive and repetitive certification and recertification of the system when downgrade policies change. A necessary pre-requisite for feasible evaluation of high-assurance transfer guard systems is to be able to separately evaluate the assurance of the underlying system and the implementation of the downgrade policy. This paper suggests an approach to extending the well-developed technique of "balanced assurance" to the formal evaluation of high-assurance transfer guards that could permit the downgrade function to be evaluated separately from the underlying TCB and then composed with it into an overall system.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection– *access controls, information flow*

General Terms

Design, Security, Verification

Keywords

Evaluation, GEMSOS, High-assurance, Multilevel security, TCB Subsets, TCB Partitions, Transfer Guard, Virtualization

1. INTRODUCTION

A transfer device is a type of "Cross Domain Service" (CDS) that permits the movement of data from one domain to another [1]. The information to be transferred is contained in the first domain, but is authorized for the second. The purpose of the transfer device is to ensure that the authorized information, and only the authorized information, is transferred to the second domain.

Examples of data transfer functions include the following:

- Anti-virus scans on information transferred from a low domain to a high domain, to protect the integrity of the high domain.
- "Dirty word" (specific content) searches on data being transferred from a high domain to a low domain, to moderate the leakage of sensitive data.
- Creating "sanitized" data releasable to a low domain out of sensitive data in a high domain. For example, summary U.S. census data may be released soon after a census, but the raw data cannot legally be released to the public domain for 72 years [5].

(Note that the latter two types of policies often require human review before information can be downgraded to the low domain, due to the inability of a computer algorithm in many cases to reliably examine data and determine that it contains no sensitive information.) We refer in this paper to transfer devices that implement any of these types of policies as transfer "guards".

Guards have generally been built on low-assurance, commodity technology. For example, a transfer device listed in a recent Unified Cross Domain (CD) Management Office (UCDMO) Baseline List, a list of CDSs that are available for deployment by U.S. Government agencies [6], runs on "commodity commercial off-the-shelf servers running Red Hat Enterprise Linux 5 with a Strict SELinux policy [7]", and SELinux has also been proposed as a suitable base for guards by others [8][10]. The U.S. National Security Agency (NSA), however, has said that "Security-enhanced Linux is only intended to demonstrate mandatory controls in a modern operating system like Linux and thus is very unlikely by itself to meet any interesting definition of secure system" [11].

The Aesec Virtual Guard (AVG) architecture is a design for a transfer guard built on a high-assurance TCB and intended to address weaknesses in current guard technology [12]. The AVG architecture incorporates the commercial, off-the-shelf (COTS) Gemini Secure Operating System (GEMSOS) TCB [14]. The NSA has previously evaluated the GEMSOS security kernel and product Ratings Maintenance Phase (RAMP) plan at the highest level, Class A1, of the NSA's "Trusted Computer System Evaluation Criteria" (TCSEC, also known as the "Orange Book") [16], as part of the evaluation of the Gemini Trusted Network Processor (GTNP) [15].

A goal of the AVG is to make it evaluable at the EAL7 level of the "Common Criteria for Information Technology Security Evaluation" [17] with a suitably strict protection profile that gives assurance equivalent to TCSEC Class A1. This goal requires the ability to formally reason about the security enforced by the system. Reasoning about an entire system as a monolithic entity, however, can be quite difficult. To make this effort feasible, the system must be decomposable into smaller parts that can be evaluated separately and then composed together into an overall evaluated system.

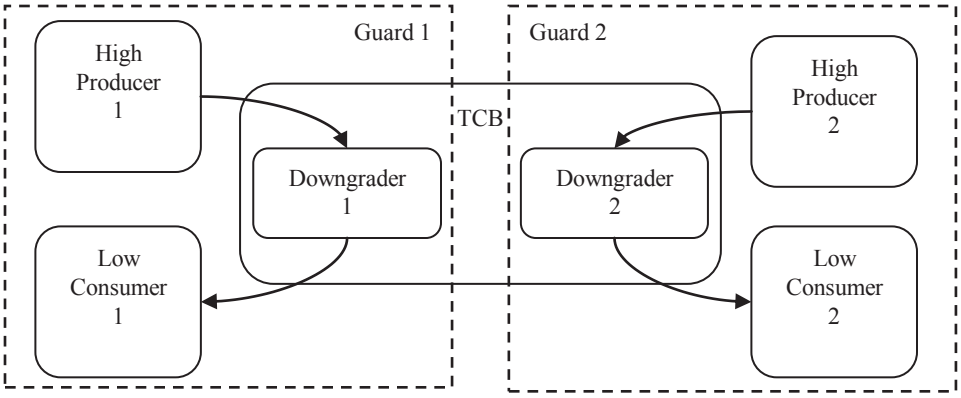


Figure 1 – Multiple, Independent Guards on one TCB

Ideally, we'd like to be able to separately certify (and accredit) transfer guards in the context of the overall system, with the ability to add or modify guards but requiring evaluation only of the new guards and without the need to reevaluate the entire system.

It is an intrinsic property of high-to-low transfer that the multilevel security (MLS)-enforcing TCB alone cannot provide assurance of the transfer security. A guard process must be a trusted subject that is, by definition, "trusted" with the capability to perform downgrades not otherwise permitted by the security policy. The use of trusted subjects complicates the evaluation of transfer guard systems. For example, while the Bell-LaPadula access control model, frequently used for evaluating the properties of MLS TCBs, includes the notion of a trusted subject as "those subjects not constrained by the *-property" [20] (the *-property says that a subject at a given security level must not write to any object at a lower security level), the model cannot address the information transfer function of the trusted subject itself, saying only "Outside the model, a subject, to be designated 'trusted,' must be shown not to consummate the undesirable transfer of high level information that *-property constraints prevent untrusted subjects from making."

Well-developed techniques exist for composing TCB systems where the systems share a homogeneous policy (e.g., "TCB Partitions" [19]) or where one system's policy is a refinement of the policy implemented by the other ("TCB Subsets" [18]). The application of these techniques, however, is complicated when trusted subjects are part of the component systems. For example, the technique of TCB Subsets, described in the "Trusted Database" interpretation (TDI) of the TCSEC [18], generally requires that each TCB subset must include all of the subjects that are trusted with respect to its technical policies. If this restriction holds, we would be unable to evaluate the transfer guard function separately from the TCB.

An additional problem arises when attempting to compose a collection of interconnected components when each has a different, heterogeneous security property, a problem sometimes called the "unconstrained composition" problem [2]. The downgrade function implemented by a transfer guard appears to enforce a quite different sort of policy than the access control policy enforced by the underlying TCB. And the downgrade function itself may consist of a series of multiple, different functions.

In this paper, we explore how the proven composition techniques of Trusted Subjects and TCB Partitions can be used to support compositional evaluation of a transfer guard implemented as a trusted subject, or subjects, running on a high-assurance MLS TCB. Our focus is on how these techniques can be used to evaluate the infrastructure that surrounds the downgrade function, but not necessarily how to evaluate the correctness of unconstrained composition in the downgrade function itself. First, we present an abstract architecture for a transfer guard and describe the AVG architecture and its use of trusted subjects. We then describe how different compositional techniques can be used to support an incremental evaluation of a high-assurance transfer guard system built using the AVG architecture.

2. ABSTRACT ARCHITECTURE

Figure 1 is an abstract depiction of a transfer guard system. One or more independent downgraders are connected to high "producers" that produce the information to be downgraded, and to low "consumers" that consume the downgraded data. The combination of high producer, downgrader, and low consumer constitutes a "guard". The underlying TCB on which the downgraders run isolates the guards from one another. Given a TCB of sufficiently high assurance, each guard may have a different downgrade range.

The key formal argument questions that need to be made about a guard system at this level of abstraction are

I-1. Can the producer, downgrader, and consumer of each guard be composed into a guard

I-2. Can the guards be shown to be isolated from one another

I-3. Can each downgrader be evaluated separately from the underlying TCB

Figure 2 depicts how a downgrader may consist of one or more "stages", each of which implements a different downgrade policy on the data. The stages are organized into a pipeline. One stage processes the data and then passes the data to the next stage. The last stage releases the data. Key formal arguments that need to be made at this level of abstraction are

II-1. Can the code in each stage be shown to enforce the downgrade policy that it implements?

II-2. Can the stages be composed into a single guard downgrade policy?

II-3. Can the isolation and pipeline ordering properties be proven?

Formal arguments I-1 through I-3, and II-3 are "infrastructure" arguments and can be shown using techniques of TCB Partitions and TCB Subsets. For II-1 and II-2, however, which deal with policies specific to the downgrade function of the guard, other techniques must be used.

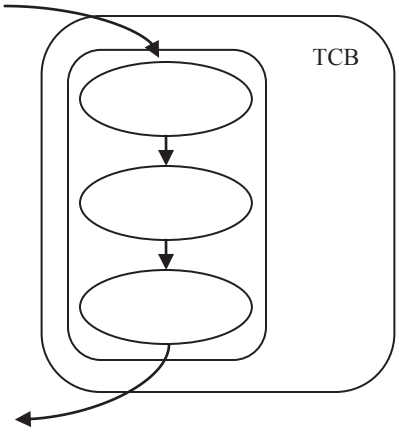


Figure 2 – Pipelined guard stages

Before discussing how the evaluation may proceed, we first describe an implementation of the abstract guard architecture on a high-assurance TCB. Several features of the implementation and the underlying TCB are fundamental to our composition arguments.

3. THE AVG ARCHITECTURE

We describe here an implementation of the abstract guard architecture described in section 2. The AVG architecture is a high-assurance transfer guard system based on the Class A1 GEMSOS TCB. We originally presented this description of the AVG in an earlier paper [12], but repeat it here for the convenience of the reader.

The AVG architecture is a design for a high-assurance transfer guard system implemented through GEMSOS-provided process isolation, GEMSOS-supported multilevel subjects, and assured pipelines created using GEMSOS mandatory access class labels. Network clients – the producers and consumers – communicate with the AVG through standard protocols such as Network File System (NFS). The processes that implement the communications protocols (equivalent to Unix "daemons") are single-level processes outside the TCB. The AVG architecture is depicted in figure 3.

3.1 Assured Pipelines

An assured pipeline limits communication within a sequence of processes so that each process in the pipeline can only receive information from the previous process and send information to the next process [4]. Processes outside the pipeline cannot interfere with data in the pipeline. Assured pipelines have been a feature in other guard designs as well as the AVG, although on low-assurance operating systems [8][10].

Assured pipelines in the AVG are implemented using integrity categories, which are part of the integrity component of the mandatory security labels assigned to every subject and object managed by the GEMSOS TCB [14]. Each process in a guard downgrader shares a unique integrity category, called the *guard identifier*. The guard identifier protects the downgrader from outside processes because the lattice-based MAC policy enforced by GEMSOS requires that the write label of subjects contain an integrity category in order to be able to modify objects that have that same integrity category. Only the processes in a downgrader, however, have the guard identifier integrity category assigned to that guard.

Additional integrity categories are used to keep the pipelines of the downgrader ordered and separate. For example, in figure 3, the Output Queue Manager is trusted within a very specific integrity range so that it can read from the Low Message Buffer, which has a secrecy level of "Low" and integrity categories "ic1" and "ic2", and write to the Low Message Queue, which has a secrecy level of "Low" and integrity categories: "ic1", "ic2", and "ic3". This is an example of an assured pipeline. Integrity category "ic1" is the guard identifier, while "ic2" and "ic3" are used to implement the

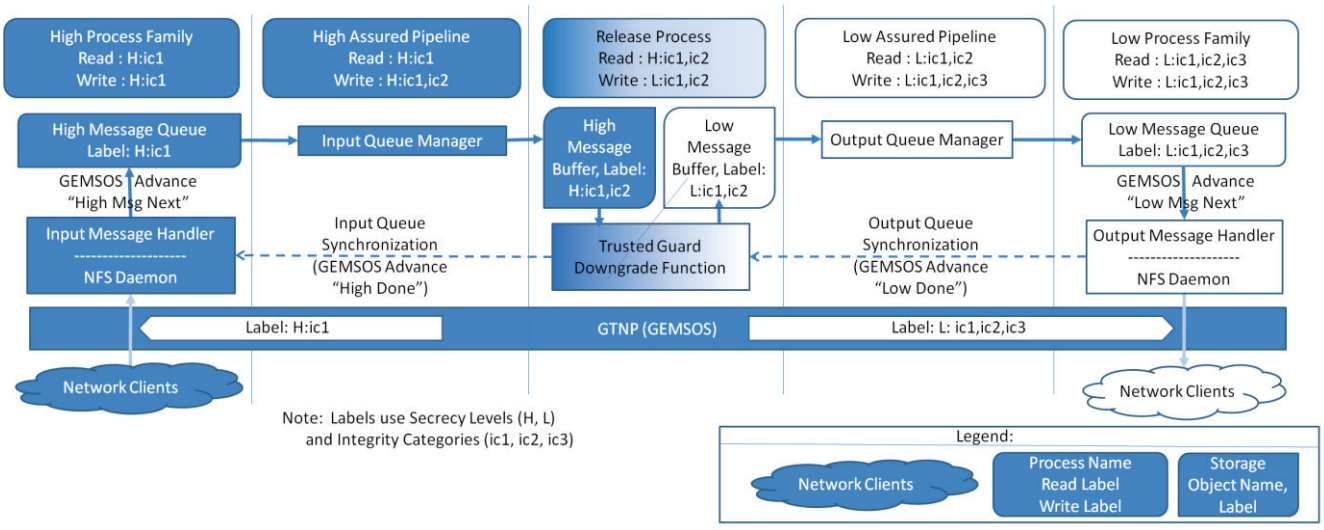


Figure 3 - Aesec Virtual Guard Architecture

assured pipeline. Other processes in the downgrader lack the "ic3" integrity category, so they cannot bypass the Release Process and write directly to the Low Message Queue. There is a second assured pipeline on the input side of the guard.

The use of guard identifier integrity categories and assured pipelines means that there can be multiple guard downgraders with different ranges on the same host system. Even trusted processes that are part of different guards cannot interfere with one another.

3.2 Trusted Subjects

The AVG architecture uses the multilevel, trusted subject feature of GEMSOS in two ways:

1. The process that performs the downgrade is trusted with respect to secrecy by the high source domain to maintain secrecy in the transfer to the low destination. For example, a sanitizer is "trusted" to remove sensitive information before putting data into the low destination domain. The range of trust of the downgrade process is explicitly limited by the GEMSOS-enforced labels assigned when the system is configured.
2. Assured pipeline processes are trusted with respect to integrity to create higher integrity results. In strict Biba integrity [3], low-integrity data cannot flow to a higher-integrity domain. Assured pipeline trusted processes, however, read from a lower-integrity domain and write to a higher-integrity domain (viz., a domain with an additional integrity category), which protects the pipeline against bypass.

3.3 AVG Design

The AVG implements a downgrader in three phases: input, release, and output.

3.3.1 Guard Input

In figure 3, a network client on the High network requests sanitized transfer of a message through the guard. In our initial prototype, this is implemented by copying a file containing the message from its local file system to a directory on the AVG system mounted by the client using the Network File System (NFS) protocol.

The directory (and the files it contains) has a secrecy level of "High", but also an integrity category "ic1". The "ic1" integrity category is the guard identifier that is unique to this guard, and every subject and object in the guard has the same guard identifier.

The actual transfer of the file from the client is accomplished by the single-level Input Message Handler process (which effectively serves as the NFS daemon for the High network). The Input Message Handler has the same access class ("High" secrecy and integrity category "ic1") as the High Message Queue so it can write to the Queue.

The Input Message Handler processes the file, writes the message into the High Message Queue, and signals the Input Queue Manager that a message is in the High Message Queue by incrementing an eventcount – a type of secure synchronization object implemented in GEMSOS [13]. The operation to increment an eventcount is called "advance". The eventcount has the same access class as the High Message Queue, so it can be read by the Input Queue Manager. Upon initialization, the Input Queue Manager reads the current eventcount value associated with the High Message Queue and then blocks while it waits for the eventcount to be incremented. When the Input Message Handler increments the eventcount, the Input Queue Manager wakes up so it can process a message in the queue. Another eventcount is used to signal the Input Message Handler when space is available in the High Message Queue. This type of synchronization using eventcounts is performed for each of the queues and buffers in the downgrader. We will omit further details about buffer synchronization in order to simplify the description of the downgrader.

The Input Queue Manager process transfers each message from the High Message Queue to a High Message Buffer that it shares with the Release Process. The Input Queue Manager is an assured pipeline process that is trusted within a very specific integrity range so that it can read from the High Message Queue, which has a secrecy level of "High" and integrity category "ic1", and write to the High Message Buffer, which has a secrecy level of "High" and integrity categories: "ic1" and "ic2. The additional category "ic2" means that the High Message Buffer has higher integrity.

3.3.2 Guard Release

The Release Process implements the downgrade function (a single stage function, in this example) and is trusted within a secrecy range "High" to "Low". Note that the "High" and "Low" labels in this example are not necessarily system high and system low, but are the configured high and low secrecy levels of the range to which the downgrader has been constrained. The installation and configuration of the guard creates this limited downgrade range, and the underlying TCB enforces it with high assurance, regardless of any attempts to exceed that range that might occur in the downgrade function.

The Release Process reads an input message from the High Message Buffer, performs downgrade processing on the message, and writes a downgraded message to the Low Message Buffer, which has secrecy level "Low" and the same two integrity categories ("ic1" and "ic2") as the High Message Buffer. From this point on, all processing in the guard is done at the "Low" secrecy level.

3.3.3 Guard Output

The Output Queue Manager is an assured pipeline that can read from the Low Message Buffer, which has a secrecy level of "Low" and integrity categories: "ic1" and "ic2", and write to the Low Message Queue, which has a secrecy level of "Low" and integrity categories: "ic1", "ic2", and "ic3".

The Output Queue Manager copies messages from the Low Message Buffer and puts them into the Low Message Queue. The Output Message Handler copies each message from the Low Message Queue into a file in a Low directory in the file system. Clients on the Low network can retrieve the "Low" files containing sanitized data by mounting the Low directory using the NFS protocol.

3.4 Multiple Pipeline Stages

An important advantage of the assured pipeline feature of the AVG architecture is the ability to easily add additional stages. The figure shows a single-stage downgrader, but additional assured pipelines can be created using additional integrity categories ("ic4", etc.). Each of these pipelines can be used to connect to an additional stage.

3.5 Performance

We implemented a prototype of the AVG and ran it on a single 550MHz Pentium III processor system. The downgrade function consisted of a single-stage "dirty word search". The system processed approximately 2500 4KB messages/second. This processing rate was consistent, up to the test maximum of 3 simultaneous guard instances.

4. FORMAL ARGUMENTS

We divide formal arguments into two parts: "infrastructure" and "downgrade policy". Infrastructure arguments are concerned with, for example, the isolation properties enforced by the underlying TCB. Downgrade policy arguments are concerned with the correctness of the unconstrained composition of the downgrade functions.

4.1 Infrastructure Arguments

Infrastructure arguments can be divided into four parts:

1. That the downgrader can be evaluated separately from the underlying TCB
2. That the composition of the producer, downgrader, and consumer implement a complete guard policy
3. That the guards are isolated from one another
4. That the isolation and pipeline ordering properties required by the downgrade policy components are correct

4.1.1 Process Isolation and Pipeline Ordering

The GEMSOS TCB enforces process separation, in general. The use of guard identifier integrity categories in the AVG ensures that all components of a downgrader are isolated and protected, even from trusted subjects in other downgraders.

The AVG uses TCB integrity categories to implement non-bypassable assured pipelines. The ordering provided by the assured pipelines in the AVG, however, is not complete: processes farther along the pipeline have higher integrity (viz., more integrity categories) and so can write data into earlier stages of the pipeline. There is no functional reason for them to do so, however, and there is no security risk from the point of view of information flow, because high-level data cannot bypass the downgrader.

4.1.2 Composition of Guard System

A guard consists of a producer, downgrader, and consumer, which communicate through a network. Each downgrader, whose isolation from other downgraders is enforced by the underlying TCB, is implemented in the AVG as a virtual machine.

The Trusted Network Interpretation (TNI) of the TCSEC proposes a virtual machine as an example of a network component [19]. A downgrader running on a virtual machine implemented using guard identifier integrity categories on a high-assurance TCB meets the TNI requirements for a NTCB partition in that there is a clearly distinguished TCB with a definitive protection domain boundary. Additional guards, which are kept separate by their unique guard identifiers and the enforcement of the underlying system, are also NTCP partitions. The evaluation of the entire system is a composition of the partitions. Consistent with this TNI approach the GTNP Final Evaluation Report prepared by NSA explicitly notes GEMSOS's ability to support "a virtual machine on top of the Virtual Machine Monitor provided by the GTNP" [16]. The AVG leverages this feature.

Having established that each of the downgraders running independently of one another on the TCB are network components, it is a relatively simple matter to compose a downgrader together with its producer and consumer, which are also network components, using the TNI's technique of "TCB Partitions" [19].

4.1.3 Separation of Downgrader from the TCB

This is a critical argument for the AVG in that, if no such argument can be made, it would be impossible to add or modify guards without the need to reevaluate the entire system.

In a transfer guard system, there are two, policy-enforcing entities: the downgrade policy implemented by the transfer guard and the access control policy enforced by the underlying TCB that protects and limits the transfer guard. The TCB definition includes the notion of trusted subjects while the guard downgrader refines the definition of trusted subject to sharply limit what those trusted subjects are allowed to do.

A formal argument about the distinctness between the guard and the underlying TCB can be made using the technique of TCB Subsets, applied in the "Trusted Database" interpretation (TDI) of the TCSEC [18]. Although the TDI says that each TCB subset must include all of the subjects that are trusted with respect to its technical policies, the TDI in section TC-6.4 also provides an example of a special case, where "a previously evaluated TCB or TCB subset is modified to accommodate the policy-enforcing elements of a new application layer" and "the alteration takes the form of a less primitive subset which is implemented, at least in part, with trusted subjects." In this example, the TDI says "the local analysis [of the more-primitive subset on which the trusted subject runs, i.e., the TCB] represents a reasonable candidate for analysis that need not be redone." The justification in the TDI is two-fold:

1. The policy of the more primitive subset (the TCB, in this case) includes the strict enforcement of a mandatory access control policy that allows trusted subjects to execute in the less-primitive subset domains that compose the downgrader. This condition is met by the multilevel subject feature of GEMSOS.
2. Analysis of the security properties of the TCB subset is unaffected by changes to the downgrader.

The TDI is careful to point out that "an assessment of the impact of [the less-primitive subset] on the behavior of [the TCB] cannot be made strictly by an examination of the trusted subjects and the definition of [the TCB's] interface. A global assessment … is required [18]." The important point, however, is that the downgrader can be changed without requiring re-evaluation of the underlying TCB.

Previous work on secure databases, moreover, has introduced the concept of "balanced assurance", concluding "We do not believe that it is necessary to require all of the Class Al assurance techniques for the extended TCB, because it is constrained by the underlying general-purpose TCB . . . because the risk associated with its incorrect operation is correspondingly less [9]." Similarly, a downgrader, although trusted, is constrained by the underlying TCB, so it does not necessarily require all of the Class A1 assurance techniques.

The AVG is an instance of the example in the TDI, where the downgrader is a set of trusted subjects running on the previously

evaluated TCB. The process separation and access control enforced by the underlying TCB (through use of the guard identifier integrity category) isolates the guard subsystem from the rest of the system. All guard processing is constrained by the guard identifier and the administratively configured downgrade range. In particular, only information in objects that are created with the guard identifier can be downgraded. On a properly configured system, the guard subjects can't write information outside of the downgrader because the guard identifier prevents them, and nothing outside the downgrader has the guard identifier so nothing outside the downgrader can write data to it. The guard identifier is an example of a "tamperproof attribute" that implements "virtual partitioning", a form of isolated protection domain [25].

4.2 Downgrade policy Arguments

The downgrade policy arguments are concerned with the proving that the downgrader correctly implements the downgrade policies. Formal techniques for downgrade policy arguments are not well developed, however, and although our current work did not extend to this area, we briefly describe here some published techniques for reasoning about the correctness of a downgrade stage and for composing downgrade stages into an overall downgrade policy, and explain how the isolation provided by the mandatory security enforcement of the underlying TCB satisfies pre-requisites for applying these techniques.

4.2.1 Single Stages

An early attempt at formal verification of a transfer guard was NASA's Restricted Access Processor (RAP). The RAP was intended to prevent messages containing classified data from reaching a system with uncleared users and to prevent messages sent by uncleared users from "affecting" classified data – i.e., simultaneously enforcing secrecy and integrity policies. The verification effort included a model designed specifically for the RAP and a formal top-level specification (FTLS). A formal proof was performed to show the correspondence between the model and the FTLS, and an informal argument showed the correspondence between the FTLS and the code [21].

A more recent example of a formal technique for reasoning about the implementation of a downgrade policy is the Guardol programming language. Guardol is specifically intended to aide in the creation and formal verification of transfer guards [22]. An unstated caveat of Guardol is that the properties of programs that are proven using Guardol cannot be enforced unless the integrity of the code and data is protected, as it would be in a high-assurance TCB. This is an example of how infrastructure arguments are not only separable from downgrade policy arguments, but are also supportive of them.

Class A1 requirements, including strict configuration management and special safeguards, along with GEMSOS classification labels, including integrity labels, can be used to protect code and data from tampering by other subjects in the TCB.

4.2.2 Composition of Stages

The composition of the different stages in a downgrader must be shown to correctly implement the overall guard downgrade policy. An example of a technique that can be used for this purpose is the composition principle of Abadi and Lamport [23][24]. A key requirement of this composition method is to partition "agents" (the entities that perform state changes) into environment agents and system agents. When composing two systems, the agents of one system are part of the environment of the other. The process separation and classification labels of the underlying TCB can be used to validate that the agents are completely identified and distinct, satisfying the prerequisites of the composition principle.

5. CONCLUSION

In this paper we have sketched out some of the difficulties involved with formally evaluating a high-assurance transfer guard and how a combination of techniques can be used to evaluate a transfer guard system in parts that can later be composed. There are two major advantages to this approach:

1. The compositional evaluation means that components can be evaluated separately, so if a component later changes, the entire evaluation does not have to be redone.
2. The compositional techniques for which there is a great deal of confidence can be separated from less-well-developed techniques that must be used for other components, thereby increasing confidence in the correctness of the overall evaluation.

6. REFERENCES

[1] Bailey, M. 2008. "The Unified Cross Domain Management Office: Bridging Security Domains and Cultures," *in CROSSTALK: The Journal of Defense Software Engineering*, July 2008. Available: http://www.crosstalkonline.org/storage/issue-archives/2008/200807/200807-Bailey.pdf

[2] Bell, D. 2005. "Looking back at the Bell-La Padula model". In *Proc. 21st Annual Computer Security Applications Conference* (ACSAC 05), Pages 337-351. Available: http://www.acsac.org/2005/papers/Bell.pdf

[3] Biba, K.J., "Integrity Considerations for Secure Computer Systems," MITRE Technical Report MTR-3153, April 1977.

[4] Boebert, W. and Kain, R. 1985. "A practical alternative to hierarchical integrity properties," in *Proceedings of the 8th DoD/NBS Computer Security Conference*, 1985, pp. 18-27.

[5] *The "72 Year Rule"*, United States Census Bureau, http://www.census.gov/history/www/genealogy/decennial_census_records/the_72_year_rule_1.html

[6] *UCDMO Cross Domain Baseline List: As of 27 January 2012*, Available: http://www.owlcti.com/pdfs/certifications/UCDMO_v.3.5.0_Baseline_Inventory.pdf

[7] *Raytheon High-Speed Guard*, http://www.raytheon.com/capabilities/rtnwcm/groups/iis/documents/content/rtn_iis_highspeedguard_ds.pdf

[8] Fletcher, B., Roberts, C., and Risser, K. 2007. "The design and implementation of a guard installation and administration framework", 2007 SELinux Symposium and Developer Summit, 26 January 2007. Available: http://selinuxsymposium.org/2007/papers/10-GIAF.pdf

[9] Lunt, T.F., Schell, R.R., Shockley, W.R., Heckman M., and Warren, D. "A near-term design for the SeaView multilevel database system". In Proc. 1988 Symposium on Security and Privacy. 1988. Available: http://www.cs.washington.edu/research/projects/poirot3/Oakland/sp/PAPERS/00044435.PDF

[10] MacMillan, K., Shimko, S., Sellers, C., Mayer, F., and Wilson, A. 2006. *Lessons learned developing cross-domain solutions on SELinux*, Tresys Technology, LLC. March 2 2006, unpublished white paper. Available: http://www.tresys.com/pdf/Lessons-Learned-in-CDS.pdf

[11] *SELinux Frequently Asked Questions (FAQ),* http://www.nsa.gov/research/selinux/faqs.shtml#I13

[12] Heckman, M.R., Schell, R R., Reed, E.E. 2012. "A high-assurance virtual guard architecture," to be published in *Proc. IEEE Military Communications Conference (MILCOM) 2012*.

[13] Reed, D. P. and Kanodia, R. K. 1979. "Synchronization with eventcounts and sequencers", *Communications of the ACM*, February 1979, Volume 22, No. 2, pp. 115-123

[14] Schell, R. R., Tao, T. F., and Heckman, M. R. 1985. "Designing the GEMSOS security kernel for security and performance," in *Proceedings of the 8th DoD/NBS Computer Security Conference*, 1985, pp. 108-119.

[15] *Final Evaluation Report, Gemini Computers, Incorporated, Gemini Trusted Network Processor, Version 1.01*, National Computer Security Center, 1995. Available: http://www.aesec.com/eval/NCSC-FER-94-008.pdf

[16] *Department of Defense Trusted Computer System Evaluation Criteria*, 5200.28-STD, United States National Computer Security Center, December 1985. Available: http://csrc.nist.gov/publications/history/dod85.pdf

[17] *Common Criteria for Information Technology Security Evaluation*, Version 3.1, CCMB-2009-07-001, July 2009

[18] *Trusted Database Management System Interpretation of the TCSEC (TDI),* April 1991, NCSC-TG-021. Available: https://www.fas.org/irp/nsa/rainbow/tg021.htm

[19] *Trusted Network Interpretation of the Trusted Computer System Evaluation Criteria*, DoD 5200.28–STD, 31 July 1987, NCSC–TG–005. Available: http://csrc.nist.gov/publications/secpubs/rainbow/tg005.txt

[20] Bell, D. E. and LaPadula, L. J. 1976. "Secure Computer System: Unified Exposition and Multics Interpretation," Technical Report ESD-TR-75-306, MITRE Corp., March 1976. Available: http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA023588

[21] Proctor, N., The Restricted Access Processor: an example of formal verification. *Proc. 1985 IEEE Symposium on Security and Privacy*, p. 49-53. Available: http://www.cs.washington.edu/research/projects/poirot3/Oakland/sp/PAPERS/00044558.PDF

[22] Hardin, D., Slind, K., and Whalen, M. 2011. "Introduction to the Guardol programming language and verification system," presented at the 5th Annual Layered Assurance Workshop (LAW 2011), Orlando, FL, USA. December, 2011. Available: http://fm.csl.sri.com/LAW/2011/law2011-paper-hardin.pdf

[23] Abadi, M. and Lamport, L. 1993. Composing specifications. *ACM Trans. Program. Lang. Syst*. 15, 1 (January 1993), 73-132. DOI= http://doi.acm.org/10.1145/151646.151649

[24] Heckman, M.R. and Levitt, K.N. 1998. "Applying the composition principle to verify a hierarchy of security servers." *Proc. of 31st Hawaii International Conference on System Sciences*, Vol. 3, p.338-347. Available: http://seclab.cs.ucdavis.edu/papers/pdfs/mh-kl-98.pdf

[25] Shockley, W.R. and Schell, R.R. 1987. TCB subsets for incremental evaluation". *Proc. AIAA/ASIS/IEEE 3rd Aerospace Computer Security Conference*, 1987, pp 131-139. Available: http://www.acsac.org/secshelf/papers/tcbsubsets.pdf

Composing Cross-Domain Solutions

Ashish Gehani Gabriela F. Ciocarlie

SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025, USA
{ashish.gehani,gabriela.ciocarlie}@sri.com

ABSTRACT

High-assurance systems with multiple security levels use data filters to facilitate the safe flow of information from higher to lower classification levels. Since filters play a critical security role, they must be formally verified. However, a wide range of sanitization strategies has been developed to address the wide variety of data content and contexts that arise in practice. As the diversity of the content and context increases, the complexity of monolithic filters grows rapidly, making them decreasingly tractable for formal verification. The MILS philosophy argues for the decomposition of any functional unit that is too large to be formally verified. Inspired by MILS, we argue that (i) data sanitization should be decomposed, (ii) each filter should handle a specific type of content, and (iii) the sanitization should provide a streaming differential privacy guarantee. Together, these will allow formal assurances for the data sanitization in the system.

Categories and Subject Descriptors

K.4.1 [**COMPUTERS AND SOCIETY**]: General—*Privacy*; D.4.6 [**OPERATING SYSTEMS**]: Security and Protection—*Information Flow Controls*

General Terms

Design, Security

Keywords

composition, cross domain solution, downgrade, filter, sanitize, MILS, streaming differential privacy, certification

1. INTRODUCTION

Despite being an integral component of the U.S. Defense Department's global information grid (GIG), current Cross-Domain Solutions (CDS) for assured information sharing (AIS) are known to have significant limitations, particularly for net-centric operations [15, 5]. Furthermore, CDS deployment times must be reduced from current levels (of months to years) to satisfy stringent requirements, such as U.S. Navy training event setup times (of one month for U.S.-coalition events, one week for U.S.-only events, or one day for Department of Defense-only events) [27].

This agile CDS vision can be achieved by decomposing the problem into subproblems that are more tractable, and then integrating the component solutions. The building blocks for achieving this include formal specifications for generic downgrading engines, formal languages for data sanitization rules [12, 7], filters for specific data types, and attribute-based access control [18]. Using pre-certified commercial off-the-shelf (COTS) CDS facilitates rapid deployment in the field. However, the availability of COTS devices depends on their timely evaluation. To this end, we examine an approach for complex data sanitization operations in decomposed filters that aims to speed up CDS evaluation time.

Section 2 describes the settings in which the need for data downgrading has arisen. Section 3 explains why the increasing complexity of downgrading operations poses a challenge to certifying the available solutions, the motivation for a paradigm shift in how downgrading is performed, and the limitations of current data sanitization technology in the new framework. Section 4 describes an architecture for decomposing downgrading to facilitate certification, and composing the assurance provided. We conclude in Section 5 that despite the challenges, it should be possible to deploy data sanitization technologies for the increasingly complex data and contexts of high-assurance systems within the target timeframes.

2. DATA DOWNGRADING

We examine the use of data downgrading in two settings: high-assurance systems and privacy-preserving data publication. We distinguish between a *filter*, *guard*, and *downgrader*, with the first performing a reduction in data fidelity, the second verifying it, and the third effecting the combination.

High-assurance CDSs are instrumental when information sharing across security domains is necessary *and* the repercussions of security breaches are very significant – for example, when sharing seemingly benign information discloses a latent vulnerability that can compromise a system. If the impact is virtually irreversible, even infrequent occurrences are not acceptable. The most conservative approach is to block all channels of information flow between components with differing security classifications. However, this prevents

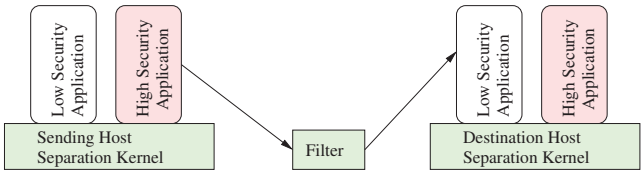


Figure 1: A *filter* intercedes on channels between hosts to sanitize data flowing from a higher security level to a lower level.

legitimate high-to-low information sharing and data transfers from lower-security sources to higher-security destinations. In the latter case, unidirectional communication links with data diodes can facilitate data flow, but this provides no protection against the introduction of malware.

If a system's security-critical components are decomposed into modules that can each be completely verified and between which no unauthorized interaction can occur, then the integrated system's security can be assured. The *separation kernel* introduced by Rushby [23] provides such functionality by isolating partitions running with multiple independent levels of security on a single host and controlling the information flow between partitions, as illustrated in Figure 1. Moreover, dowgraders need to cope with multiple types of data [22], requiring transformation and sanitization mechanisms to allow the information flow.

3. CERTIFICATION CHALLENGE

Since downgraders operate at the boundary between data of different sensitivity levels, they are security-critical components of a computing infrastructure. Assurance of the security of a system is therefore dependent on being able to verify the correctness of downgrader operations.

Early downgrading focused on specific data types and predefined contexts. For example, the U.S. Department of Defense's Global Positioning System (GPS) used to downgrade the location information available to civilians (and therefore also adversaries) by adding a pseudorandom error to part of the signal [16]. While this approach simplified the process of verifying that the downgrading operation conformed to its specification, the need for more complex downgrading policies became apparent with the development of *differential GPS*. (The errors added during downgrading could be calculated in real time by a receiver at a known location that then retransmitted the error stream to consumers who could use it to determine their own location with greater accuracy.)

As the range of data content sensitivities, environments from which it originates, communities with which it needs to be shared, and kinds of operations being performed on it continue to increase, so has the complexity of the rules used to downgrade data flowing between different security classifications. The statistical guarantees provided by privacy-preserving data publication algorithms depend on the soundness of the data sanitization infrastructure.

Certification of a downgrader requires detailed requirements and specifications, proofs of correctness, a structured design process, detailed documentation, and the development of test cases with sufficient coverage. Thus, as the downgrader becomes increasingly complicated, the cost of formal assurance and the time that elapses before it can be deployed in the field also grow.

4. DECOMPOSING SANITIZATION

A few approaches for handling data in multi-level secure systems have relied on separate instances of an application running at each security level [13, 26]. In contrast, we address the problem of downgrading data that has components with multiple classification levels, as occurs during complex joint training missions [4]. We advocate an approach that leverages the nature of the data being downgraded and the available trusted computing infrastructure to decompose the downgrading functionality to the point that each module can economically be formally specified and have its operational behavior verified. This is illustrated in Figure 2.

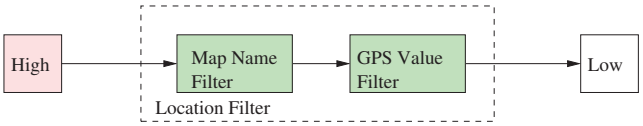


Figure 2: Data sanitization functionality can be decomposed until each sanitization primitive can be certified. For example, sanitization of location information can be split into one operation that redacts blacklisted names from a map while a second perturbs the numeric values in GPS values.

Architectural Context

Since we use an architecture-based solution to make complex data sanitization practical, we first describe the MILS [4] context in which it is framed.

The MILS philosophy advocates a top-down approach to securing a system, mirroring the architectural pattern employed by the *multiple independent levels of security* initiative [2]. The essence of the MILS architectural pattern is to first describe the required security policy in terms of a *policy architecture* (typically, with a diagram in which boxes encapsulate processing functions and arrows depict information flow). The difficulty of making the assurance case is then examined, assuming a direct mapping between the architecture and a physical realization of it. If there is difficulty in proving the assurance case, then the policy architecture is further decomposed until a point is reached where the assurance case can be proved.

The MILS approach separates the desired security properties from the resource-sharing problem. Commodity implementation of individual modules is facilitated through the development of *protection profiles* that articulate the properties that the resources must possess. Downgraders form one class of such functionality. A significant reason that current downgraders are limited to simple operations, such as redacting patterns contained in a database of blacklisted terms or perturbing numeric values using fixed rules [24], is to ensure that they are consistent with the MILS philosophy of individual modules implementing well-defined security properties.

Filter Configuration

Downgraders operate on structured data that has a defined data model [3]. The information is transferred in fixed format messages with multiple data fields and the allowable range for each specified in the associated metadata. The information is intended to support interoperability between machines. Consequently, any piece of data that is a candidate for sanitization is either of a type for which a filter exists or has an associated data model that can be used to decompose the data into constituent objects. The same analysis can be applied recursively to the constituent objects, allowing a suitable set of filters to be selected. Together, the set can be used to sanitize the target data object.

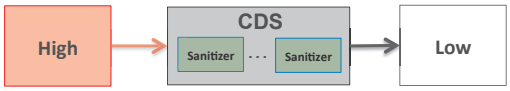


Figure 3: *Intra-CDS* composition deploys multiple filters within a single downgrader.

Assuming the set of filters needed has been determined, they can be deployed in multiple possible configurations. The sanitization guarantees must compose in all cases. In the first configuration, a single downgrader may utilize multiple filters internally to handle different data types, as illustrated in Figure 3. Traditional monolithic downgraders utilize this approach, parsing the incoming data and applying one or more sanitization algorithms to subsets of the information.

Figure 4: Data flows sequentially through consecutive filters when *serial CDS* composition is employed.

In future environments, where the CDS is decomposed into modular pieces that can be verified, two new configurations arise. In one configuration that we consider, multiple downgraders are employed in serial order, with computational operations performed between, as illustrated in Figure 4. One downgrader is used when the input of the intermediate computation must be sanitized, while a second downgrader is used to sanitize the output. The final configuration manifests when data flows from a higher classification level to a lower classification through multiple paths that do not intersect. In this case, a separate downgrader is needed along each path, as illustrated in Figure 5.

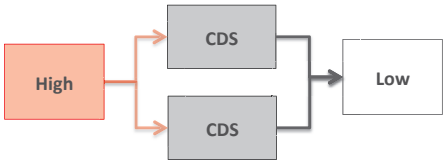


Figure 5: Data can also flow simultaneously through multiple filters when *parallel CDS* composition is utilized.

Sanitization Algorithms

Data sanitization algorithms have been extensively studied in the context of publishing privacy-sensitive data. In this setting, individuals contribute their records to a trusted publisher that processes the collected information. The results are stored in a database that can be queried [14]. To preserve the privacy of individual record owners, a downgrader sanitizes information derived from such databases.

Early work proposed perturbing the query inputs and outputs, and restricting the number of queries [1]. Since then, a range of strategies has been utilized [6], including *generalization* that coarsens, abstracts, or collects multiple data in equivalence classes, *suppression* that removes records from the sanitized output, *swapping* that interchanges attributes from different records, *randomization* that adds noise to perturb the data, and *multi-views* that provide sanitization through diverse perspectives.

The approaches may be domain agnostic and focus on *quasi-identifiers*, data fields that are potentially sensitive. Examples of this include *k-anonymity* [25], which aims to ensure that the output contains at least k records with the same quasi-identifiers, *l-diversity* [19], which ensures that auxiliary fields contain at least l different values, and ϵ*-privacy* [20]. Alternatively, the approaches may be customized to specific data types, such as network addresses [28], data formats, such as audit logs [17], or specific application domains, as is the case for each scheme that guarantees *differential privacy* [8].

Historically, the database that is being sanitized is assumed to be static. In many contexts this is a reasonable assumption since all the relevant information is collected first, before sanitization and querying is performed. In the case of a CDS, the data may never have been observed previously. Traditional offline algorithms must be replaced by online ones that perform data sanitization by operating on a stream of information as its arrives. Recent research on *streaming differential privacy* [9] provides a framework for designing sanitization algorithms appropriate for a CDS. Of particular utility is the fact that the guarantees in this framework compose. This provides a sound basis for designing modular filters that can be used in MILS architectures while ensuring that their combination provides formal sanitization guarantees.

Threat Model

The framework in which streaming differential privacy can be defined for a CDS assumes that each filter operates on a stream of data objects. Each incoming data object is processed in a single step, during which the object is examined, the internal state of the filter is emitted, and an output may be emitted. This is illustrated in Figure 6. The filter can also produce output at the end of the stream.

We conjecture that the *continual observation* [11] threat model used in streaming differential privacy can be adapted to allow an adversary to monitor all the output produced by a filter, as illustrated in Figure 7. In the MILS setting, the output of the filter flows to a lower security classification level where an adversary may have increased access.

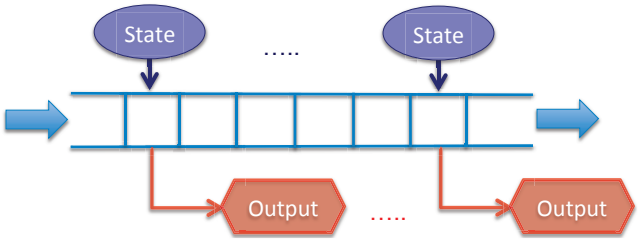


Figure 6: A CDS filter processes each data object in a separate step.

Pan privacy [11] guarantees that sensitive information that has been streamed through the filter will not be leaked even if the internal state is compromised, as illustrated in Figure 8. Pan privacy can be considered in distinct settings. In the first, the leakage of the internal state may be *announced* (when responding to a subpoena, for example). Alternatively, the compromise may be *unannounced*, in which case an adversary can access the internal state without any forewarning.

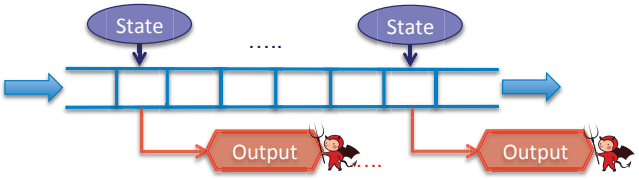


Figure 7: The output is assumed to be under *continual observation* by an adversary.

It is worth noting that increasing the power of the adversary decreases the possible guarantees that can be provided. In the MILS context, pan privacy is not necessary since the platform on which the filter operates would provide the necessary assurance that the internal state is not available to the adversary. However, since a filter that does not leak sensitive information after an intrusion may be of utility in a broader context, we consider the pan privacy case as well.

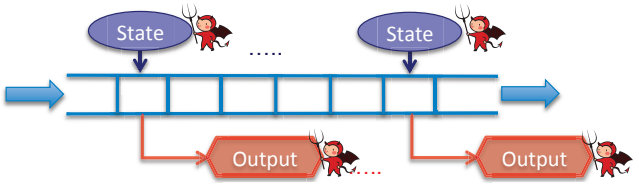


Figure 8: *Pan privacy* assumes that the adversary can gain access to the internal state of the filter.

Characterizing Leakage

To sanitize data that is being released to a lower security classification level, it is necessary to characterize the leakage of information. This is particularly challenging when the leakage itself may depend on auxiliary information that is available externally, but never observed by the downgrader. Intuitively, the amount of information that is being leaked corresponds to the quantity of data that must be removed from an unsanitized stream of objects S to yield a sanitized output stream S'.

Figure 9: *User-level X-adjacent* streams are identical if the elements in X are eliminated.

The notion of leakage can be formalized analogously to the way it is defined for streaming differential privacy. Instead of formulating leakage over query streams, we focus on streams of data objects. We consider streams S and S' to be *user-level X-adjacent* if the only difference between them is the presence of data objects $x \in X$. Figure 9 depicts two streams S and S' that are identical (as shown in red) after all occurrences of x (as shown in black) are eliminated. In this case, S and S' are X-adjacent. If only a single instance of each object $x \in X$ is replaced, then the two streams are considered to be *event-level X-adjacent*. Figure 10 depicts event-level X-adjacent streams that are identical (as shown in red) when a single element (shown in black) is eliminated.

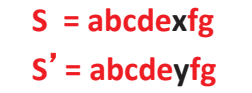


Figure 10: *Event-level X-adjacent* streams are identical if each element in X is eliminated just once.

Composable Sanitization

A sanitization algorithm A is said to be ϵ-*differentially private* against continual observation if for all pairs of X-adjacent streams S and S'

$$e^{-\epsilon} \leq \frac{Pr[A(S) = \sigma_1\sigma_2 \ldots \sigma_t]}{Pr[A(S') = \sigma_1\sigma_2 \ldots \sigma_t]} \leq e^{\epsilon}$$

where $\sigma_1\sigma_2 \ldots \sigma_t$ is the output generated by running algorithm A on the streams S and S' [10].

Consider an algorithm A that operates in the streaming differential privacy framework. A maps elements of the stream to $I \times \sigma$, where I is the set of internal states of the algorithm A, and σ is the set of possible output sequences. A is said to be ϵ-*differentially pan-private* [9] (against a single unannounced intrusion) if for all event- or user-level X-adjacent streams S and S', inputs $I' \subseteq I$, and outputs $\sigma' \subseteq \sigma$

$$e^{-\epsilon} \leq \frac{Pr[A(S) \in (I', \sigma')]}{Pr[A(S') \in (I', \sigma')]} \leq e^{\epsilon}.$$

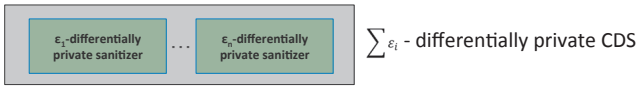


Figure 11: Differential privacy guarantees are additive over the collection of intra-CDS sanitizers utilized.

Algorithms that provide differential privacy guarantees can be composed while maintaining the assurance [10, 21]. This means that if an algorithm A_1 that provides an ϵ_1 differential privacy guarantee is combined with an algorithm A_2

Figure 12: Serial deployment of differentially private sanitizers results in an additive guarantee.

that provides an ϵ_2 differential privacy guarantee, the composition will provide an $\epsilon_1+\epsilon_2$ differential privacy guarantee.

If CDS downgraders implemented algorithms that provided analogous guarantees for streams of data (instead of streams of queries), it will be possible to deploy them in the intra-CDS configuration illustrated in Figure 11, the serial CDS configuration shown in Figure 4, or the parallel configuration depicted in Figure 5, while ensuring that the combination provides a formal sanitization guarantee.

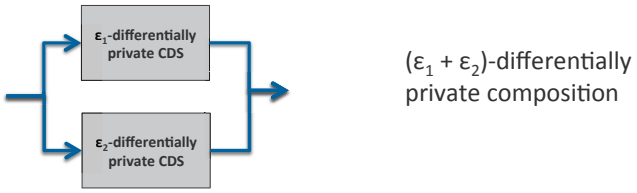


Figure 13: Parallel deployment of differentially private sanitizers results in an additive guarantee.

5. CONCLUSION

As the content and context of the data being transmitted in high-assurance systems continues to increase in complexity, the cost and time to certify cross-domain solutions is growing rapidly. We argue that downgrading functionality should be decomposed to the point where each filter provides a streaming differential privacy guarantee and its certification is economically viable. The resulting filters can be combined to provide equivalent functionality to that provided by monolithic downgraders. Of particular note is the fact that the streaming differential privacy guarantees of the constituent filters can compose.

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grant IIS-1116414. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

6. REFERENCES

[1] Nabil Adam and John Worthmann, Security-control methods for statistical databases: A comparative study, ACM Computing Surveys, Vol. 21(4), 1989.

[2] Jim Alves-Foss, W. Scott Harrison, Paul Oman, and Carol Taylor, The MILS architecture for high assurance embedded systems, International Journal of Embedded Systems, Vol. 2(3/4), 2006.

[3] Paul Beynon-Davies, Database systems, Macmillan, 2000.

[4] Carolyn Boettcher, Rance DeLong, John Rushby, and Wilmar Sifre, The MILS component integration approach to secure information sharing, 27th IEEE/AIAA Digital Avionics Systems Conference, 2008.

[5] Arthur Cebrowski and John Gartska, Net-centric warfare: Its origin and future, U.S. Naval Institute, 1998.

[6] Bee-Chung Chen, Daniel Kifer, Kristen LeFevre, and Ashwin Machanavajjhala, Privacy-preserving data publishing, Foundations and Trends in Databases, Vol. 2(1-2), 2009.

[7] Josiah Dodds, A development environment and static analyses for GUARDOL - a language for the specification of high assurance guards, Master's Thesis, Kansas State University, 2010.

[8] Cynthia Dwork, Differential privacy: A survey of results, Theory and Applications of Models of Computation, Lecture Notes in Computer Science, Vol. 4978, Springer-Verlag, 2008.

[9] Cynthia Dwork, Differential privacy in new settings, 21st ACM-SIAM Symposium on Discrete Algorithms, 2010.

[10] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy Rothblum, Differential privacy under continual observation, 42nd ACM Symposium on Theory of Computing, 2010.

[11] Cynthia Dwork, Moni Naor, Toniann Pitassi, Guy Rothblum, and Sergey Yekhanin, Pan-private streaming algorithms, 1st Symposium on Innovations in Computer Science, 2010.

[12] Boyd Fletcher, XML Data flow configuration file format specification, `http://iase.disa.mil/cds/helpful_tools/dfcf-specification_1_2_11.pdf`, 2008.

[13] Judith Froscher and Catherine Meadows, Achieving a trusted database management system using parallelism, Database Security II: Status and Prospects, North-Holland, 1989.

[14] Benjamin Fung, Ke Wang, Rui Chen, and Philip Yu, Privacy-preserving data publishing: A survey on recent developments, ACM Computing Surveys, 2010.

[15] GIG IA Architecture v1.1, `https://www.us.army.mil/suite/kc/13000401`

[16] Sameer Kumar and Kevin Moore, The evolution of Global Positioning System technology, Journal of Science Education and Technology, Vol. 11(1), 2002.

[17] Adam Lee, Parisa Tabriz, and Nikita Borisov, A privacy-preserving interdomain audit framework, 5th ACM Workshop on Privacy in Electronic Society, 2006.

[18] Ninghui Li, John Mitchell, and William Winsborough, Design of a role-based trust-management framework, IEEE Symposium on Security and Privacy, 2002.

[19] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam, l-diversity: Privacy beyond k-anonymity, ACM Transactions on Knowledge Discovery from Data, Vol. 1(1), 2007.

[20] Ashwin Machanavajjhala, Johannes Gehrke, and Michaela Goetz, Data publishing against realistic adversaries, Very Large Databases, Vol. 2(1), 2009.

[21] Darakshan Mir, S. Muthukrishnan, Aleksandar Nikolov, and Rebecca Wright, Pan-private algorithms: When memory does not help, 2010.

[22] Nancy Reed, Dave Bryson, James Garriss, Steve Gosnell, Brook Heaton, Gary Huber, David Jacobs, Mary Pulvermacher, Salim Semy, Chad Smith, and John Standard, Security guards for the future Web, MITRE Technical Report MTR 04W0000092, 2004.

[23] John Rushby, Design and verification of secure systems, ACM Symposium on Operating System Principles, Vol. 15, 1981.

[24] Richard Smith, Multilevel security, Handbook of Information Security, Wiley, 2006.

[25] Latanya Sweeney, k-anonymity: A model for protecting privacy, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, Vol. 10(5), 2002.

[26] Trusted Services Engine, Galois, `http://www.galois.com/files/TSE_Datasheet.pdf`

[27] U.S. Fleet Forces Command Briefing, Training - Cross Domain Information Sharing (T-CDIS) Summit, U.S. Joint Forces Command, Suffolk, VA, 28 October 2009.

[28] Jun Xu, Jinliang Fan, Mostafa Ammar, and Sue Moon, On the design and performance of prefix-preserving IP traffic trace anonymization, 1st ACM SIGCOM Workshop on Internet Measurement, 2001.

Information Assurance Certification with EDICT-IA™

Brian W. LaValley
WW Technology Group
(401) 348-8525
blavalley@wwtechgroup.com

Chris J. Walter
WW Technology Group
(410) 418-4353
cwalter@wwtechgroup.com

ABSTRACT

The required increase in information exchange and flow across diverse networks has created new information management security risks with higher levels of integration. The current information assurance certification/recertification is a costly, burdensome process that relies heavily on manual generation and review of documents to evaluate the information assurance capabilities of a system. These methods have large technical, programmatic and cost risks that are often hidden or unknown. These methods need to be evolved in order to support the current needs of the Air Force and other services.

In order to meet these challenges an approach was developed that provides for concrete representation and rigorous analysis of information assurance aspects of system architectures. The approach is supported by a suite of model-based design and analysis tools (EDICT-IA) that utilize open and standard modeling methods coupled with analysis techniques and certification evidence management.

The EDICT-IA tools provide model driven system design through system architecture modeling combined with analysis tools capable of evaluating information assurance properties. EDICT-IA supports early and incremental architecture evaluation and refinement throughout the system lifecycle. The EDICT-IA analysis tools can be applied to new system developments as well as to technology insertions and technology refreshes for existing systems.

These innovations advance in the state of the art for the design and evaluation of information assurance aspects of system architecture. These techniques and tools can be applied throughout the development process to yield benefits for risk reduction and cost and schedule reductions. The innovative modeling and analysis techniques provide system developers and certifiers with new capabilities to represent and evaluate the information assurance properties of system architectures.

Categories and Subject Descriptors

General Terms

Security, Model Based Design, Verification

Keywords

Information Assurance, Certification

1. INTRODUCTION

The increasing integration of sensitive information systems coupled with rapid technology development is generating systems of expanding size and complexity. These factors are rendering current design and certification methods untenable in terms of quality and schedule impact. Current manual inspection methods cannot keep up with the increases in complexity and requirements to rapidly field new systems.

The lack of domain specific and common representations for Information Assurance (IA) concerns places large burdens on developers and program managers:

- Each program must develop methods to express information assurance design and compliance with directives.
- Information Assurance aspects are not presented to certifiers until late in program development cycles.
- IA specification methods are often ad-hoc, with no support for systematic design and evaluation.

The current methods have many large technical, programmatic and cost risks that are often hidden or unknown.

1.1 Overview of Approach

An essential element to reducing the certification and accreditation costs of complex systems is to be able to break the certification problem into manageable pieces, which can be analyzed independently, and then provide a structured method for synthesis of the pieces into a system where certification confidence can be complied incrementally. The concept of separation of concerns is the backbone of the Multiple Independent Level of Security (MILS) approach for system security design. This concept establishes separation of security concerns such that it is possible to develop a hierarchical set of security services where each level of security service relies on the underlying layers to provide increasing levels of service. This concept is based on the early work by Rushby [1] [9] and is described by Alves-Foss [8]

Our work establishes a hierarchy of security services and properties that support the composition and analysis of system security properties that meet the certification requirements of IA systems. Our work, based on work by Van Fleet [3] [5], identifies a three tiered grouping of hierarchical system properties for information assurance as shown in Figure 1. The properties that we present here are semantic properties that are amenable to decomposition and application of formal property definition and analysis.

The first group, Application Properties, relates to and describes system attributes required for evaluation of for system security certification and accreditation. Five properties comprise this tier and are derived from the Information Technology Assurance Framework (IATF)[12] guidance on system design for

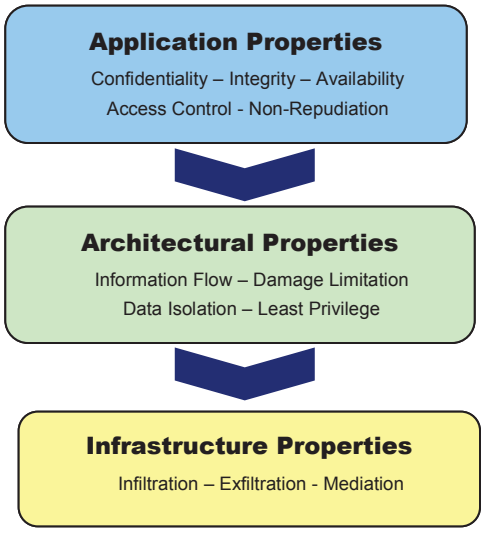


Figure 1 - Three Tier Grouping of IA Properties

information assurance. These properties also correspond to the Confidentiality, Integrity and Availability (CIA) categories used in many certification and accreditation (C&A) processes.

The Application Properties depend on the properties in the Architectural Properties. The four properties associated with this layer describe critical aspects of systems IA architecture that are necessary to provide the Application Properties.

The architecture tier in turn relies on the Infrastructure Properties. The Infrastructure Properties describe the IA capabilities of critical system infrastructure components such as separation kernels (SK) or inter-processor communication (IPC) mechanisms. There has been extensive work in using formal methods to prove the existence of these properties at the component level [2].

The following list provides a working definition of the properties at each tier in the hierarchy.

Application Properties

- *Confidentiality* - The ability of a system to prevent unauthorized disclosure of data (both stored and communicated).
- *Integrity* - The ability of a system to prevent unauthorized modification of data (both stored and communicated), detection and notification of unauthorized modification of data, and recording of all changes to data.
- *Availability* - The ability of a system to deliver timely, reliable access to data and information services for authorized users.
- *Access Control* - Ability of a system to limit access to networked resources (hardware and software) and data (stored and communicated) to authorized users.
- *Non-Repudiation* - The ability of a system to prove to a third party that an entity did participate in a communication.

Architecture Properties

- *Information Flow*- The set of communications flows in systems that are authorized to permit communications between system partitions. These are the only flows that are allowed to cross partition boundaries.
- *Least Privilege* - This property requires that in a particular abstraction layer of a computing environment every module (such as a process, a user or a program on the basis of the layer being considered) must be able to access only such information and resources that are necessary to its legitimate purpose.
- *Damage Isolation* - The consequences of an error or security breach are limited by specified partition boundaries and data protection mechanisms.
- *Data Isolation* - The architecture provides partitioning such that there are no data dependencies that cross partition boundaries.

Infrastructure Properties

- *Infiltration* – processing does not depend on or is affected by state in other partitions.
- *Exfiltration* – Processing cannot influence data/flows or the state of other objects not specified in the security policy.
- *Mediation* – The effect of the subject's execution depends only the resources with which it is allowed to interact.

This set of hierarchical information assurance properties is an essential building block for establishing cost effective high confidence system security certification and accreditation. The property definitions and dependency hierarchy support both top down system composition and bottom up establishment of critical system attributes for security that directly support system certification requirements.

2. Model Based Design Analysis

The approach using hierarchical information assurance property sets is supported by a model-driven design environment that gives IA system designers and certifiers significant new capabilities. By combining the property definitions with the work product items required to support system certification processes, a set of system abstractions has been developed that enable the system to be specified and analyzed at several levels of abstraction, depending on the development phase or the stakeholders concerns. An overview of these abstractions is shown in Figure 2.

The three tiered hierarchy of abstractions is supported through an interrelated set of models. The levels are: IA Requirements, IA Logical Architecture and IA System Architecture. The abstractions at each level contain information required to establish and compose the IA properties for the system under consideration. These three levels of abstraction build on one another in a similar fashion to the IA properties with the top level, IA Requirements, being the most abstract and the lowest level, IA System Architecture, being the most concrete. Each of the levels is linked to adjacent levels through mappings that provide traceability between the higher level entities and the lower level entities that compose it.

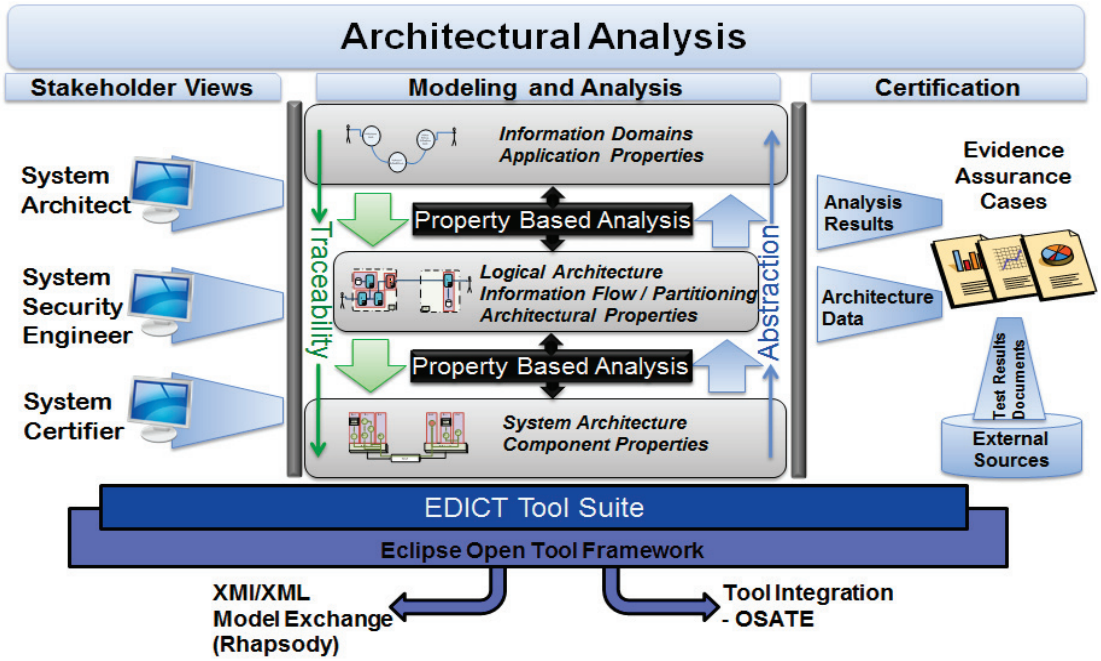


Figure 2 – Model Based IA

When utilized in a top down fashion the abstraction levels provide a structured environment to represent stepwise refinement and definition of systems IA requirements, design and implementation that can be used to support systems engineering processes. The system can be specified from a top down perspective with the properties at the higher level representing requirements to the next level down. The links between the abstraction layers provide full traceability from the highest level abstractions down to the lowest and enable auditable evaluation of the design for its ability to meet the IA system requirements.

Our approach provides a framework to incorporate system component attributes that are derived through third party testing or certification processes such as the Common Criteria. These component/subsystem level certification attributes, such as EAL rating, are entered into the property hierarchy at the appropriate level of abstraction when a component/subsystem is selected for use in the system architecture. The attributes are then matched with the requirements of the higher level abstractions that the component/subsystem supports. This upward flow of attribute values through the abstraction layers provides the basis for model based certification and high assurance.

The EDICT Tool Suite provides model-based engineering capabilities needed for the production of complex high confidence software intensive systems [6] [7] . With the EDICT tools, system developers, engineers, architects, and certifiers are able to establish models of system architecture and behavior, and evaluate the architectural mechanisms employed to ensure that dependability, safety and information assurance requirements are achieved. The results of the modeling and analysis techniques discussed in this paper are implemented in EDICT with Information Assurance tool extensions (EDICT-IA).

EDICT-IA supports early and incremental architecture evaluation and refinement throughout the system lifecycle. The tool capabilities can be applied to new system developments as well as to technology insertions and technology refreshes for existing systems. Figure 2 depicts an overview of these capabilities with modeling and analysis in the center, user role-specific views to the left and certification evidence management to the right hand side. The tools are implemented using the Eclipse open tool framework that provides for portability and common look and feel for the user interfaces.

Briefly summarizing, the capabilities address five major areas:

Architecture Modeling

- Creating / Importing architecture descriptions from standard languages (AADL, UML).

Architecture Augmentation for IA

- Adding IA specific model elements to define IA services.
- Partitioning – Encoding – Authentication - etc.

Information Domain Modeling

- Explicit specification of information separation and IA requirements.

Architectural Analysis for IA

- Analyze architecture structure and data flow for IA concerns.
  - Confidentiality – Integrity – Access Control

Analysis and Reporting for Certification

- Elements of JAFAN 6/3 currently supported.

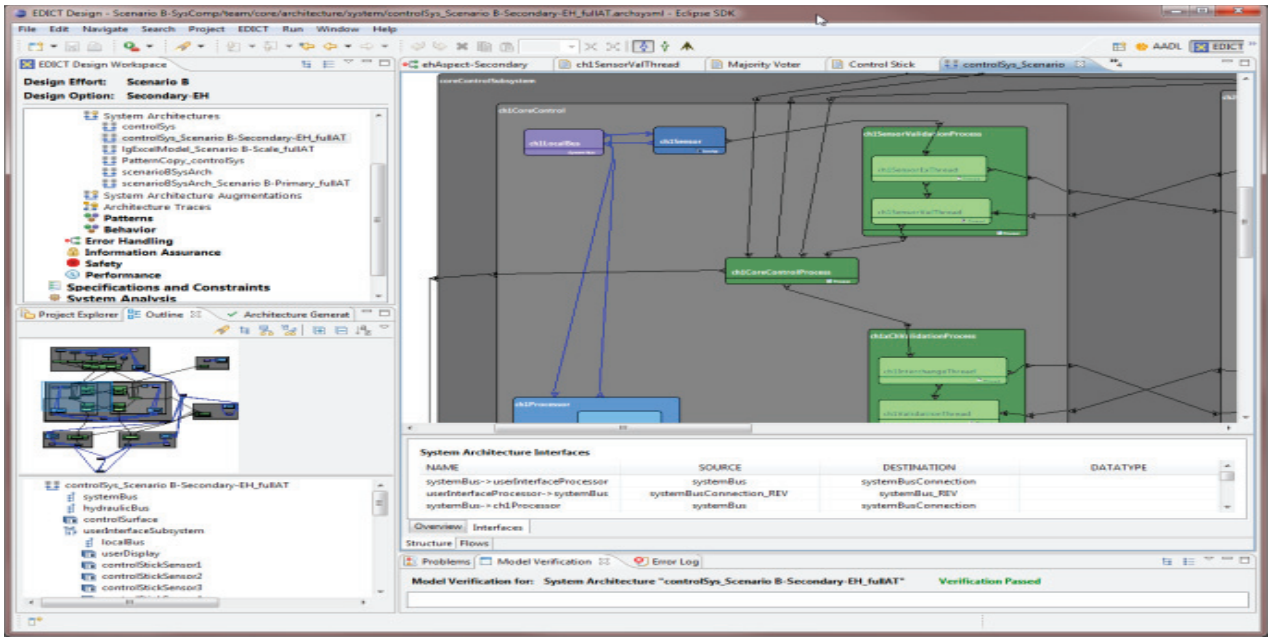


Figure 3 - The EDICT Tool Suite

These tool capabilities are integrated within a common graphical development environment that is shown in Figure 3. This figure captures a common arrangement of EDICT-IA views and editors with the system architecture editor in focus in the main editing area. On the top left hand side is the EDICT Design Workspace which provides for browsing and management of all of the models that are contained in the user's workspace. These are arranged in categories that make navigation to the desired content straight forward. This tool configuration also shows the Outline View on the bottom left which provides a capability for navigating and exploring large graphical depictions of architectures. Each of the models and analysis capabilities in the tool come with customized editors and views that enable modification and evaluation of model content and analysis results.

The rest of this paper describes the modeling and analysis techniques and the tool components that implement our vision for property based information assurance.

2.1 IA Requirements Modeling

The IA Requirements model defines the information assurance requirements by defining the top level security policies for the system under consideration and expressing the required application properties. This model establishes the required separation of information and processing through the specification of IA Domains and the allowed interactions between domains through cross domain interfaces. This model also encompasses the allowable flow of information with external systems and users.

The primary entity in this model is the *IA Domain*. An IA Domain is a set of system functions that are grouped based on their value and the security threats to the system. Each domain is then annotated with a set of properties and attributes that describe the IA characteristics associated with the domain. The domains are assigned a classification level and a required strength level (SML 1 -3) and evaluation assurance level (EAL 1 -7) for each of the five application properties. These specifications are C&A process agnostic to facilitate the use of these techniques with many C&A processes. Process specific front-ends for the domain tools can be developed that use nomenclature and model elements specific to the process. We have already completed this for JAFAN 6/3.

IA Domains have interfaces that define the allowed flow of data between domains. For instance, if a function in a domain requires the service or information from a function in a second domain, a cross domain interface is established in this model. These cross domain interfaces are important to the overall security analysis of the system because in many instances they represent the exchange of data between two differing levels of classification and therefore require the highest levels of scrutiny. Figure 4 contains a screen-shot of the EDICT editor that supports the specification of IA Requirements models.

Taken together, these specifications define a security policy for the system that establishes the required separation of data, the allowable data flows between domains and the level of IA

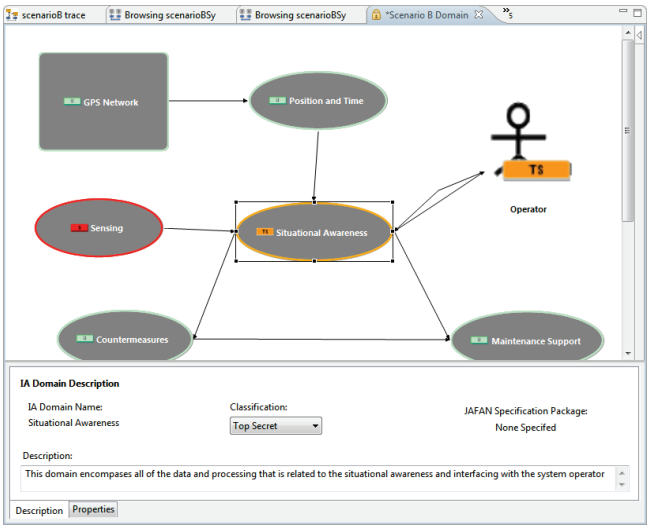


Figure 4 - IA Requirements Model Editor

services that must be supplied to provide the application properties in each domain.

2.2 IA Architectures

The purpose of the IA Architecture models is to capture the design of IA services and mechanisms that implement the security policy defined the IA Requirements model. The EDICT-IA tools allow for the definition of both logical and system architecture models. The logical architecture models represent an implementation independent model and the system architecture models provide a more detailed implementation specific view of the hardware and software architecture.

The *logical architecture model* is comprised of the following five entities that are essential for the analysis of the information assurance properties of a system:

1. *Information Processing* – Units of processing used to accomplish the functional requirements that are protected by the security services of the IA architecture.
2. *Information Storage* – Persisted information that is protected by the security services of the IA architecture.
3. *Information Flow* – The required flow of information between processing elements to meet the functional requirements.
4. *Partitioning* – The partitioning that is required in the system to meet the security requirements.
5. *Security Services* – Information processing elements that provide the security services required to meet application security properties. These elements include guards, encryptors, message routers and authentication services and devices.

The *system architecture model* presents a detailed view of the software and hardware components that are used to implement the system with emphasis on the components that provide the required attributes to support the security properties and services specified at the higher levels of abstraction. The purpose of this model is to define a specific system implementation which is intended to provide the services and properties defined in the IA requirements and logical architectures. This system architecture model can be linked directly to implementations at the code/hardware level to provide for eventual traceability to system implementations.

The model entities that are required at this level of abstraction are the same types that are commonly found in architecture description languages. These abstractions include items such as processors, networks, devices, processes, threads, software components and hierarchical systems/subsystem compositions. Our approach is to represent these items using a standard architecture description language and then annotate that language with the extensions required to support the security properties. EDICT contains two model translation capabilities that enable our approach to work with standardized modeling languages. We support the Architecture Analysis & Design Language (AADL) and UML/SysML models developed in the Rhapsody tool suite. The AADL is an open, standardized architecture modeling language with an extensible specification and implementation. The support for UML/SysML from the Rhapsody tool chain uses a subset of the diagrams to construct a coherent system architecture model for analysis.

2.2.1 Partitioning View

Analysis of either the logical or the system architecture models requires that the partitioning mechanisms that are used to achieve separation of concerns are explicitly represented [4]. There are three types of partitions that play a role in the analysis of information assurance properties for an architecture. They are:

Site – represents physical separation and may provide a level of physical security that supports the establishment of application level properties, such as Confidentiality, Integrity or Access Control for the elements that are allocated to the site.

Enclave – the enclave can be used to represent a traditional network partition enclave or the MILS definition of a group of partitions running at the same level of classification. Services may be allocated at this level that are common to the enclave and can be used to provide enclave boundary protections.

Partition – The partition is the fundamental element in the system that enforces separation between system components and regulates their interactions. This is essential to preventing unauthorized disclosure of information, damage limitation and integrity. Partitions can be used to represent physical partitioning such as whole machines or software implemented partitions such as MILS RTOS partitions.

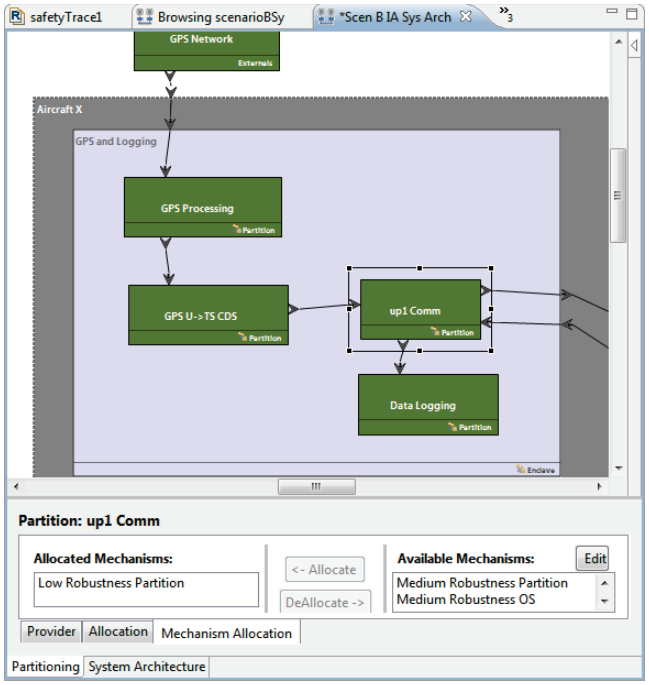


Figure 5 - Partition View

The EDICT-IA tool contains an editor that allows for the partition structure to be specified and viewed. Figure 5 is a screen shot of the editor that depicts a Site containing an Enclave which in turn contains a set of Partitions. The editor allows the user to assign IA Mechanisms to the structures to define their IA characteristics. These mechanisms are described in Section 2.2.4. The editor provides the ability to assign architecture components to the partitions to define their contents.

2.2.2 Connecting Partition Types

The three types of partitions that we have described provide protection for components inside the boundaries and control of access and information flow across the boundaries of the partitioning groups. Flow of information across the boundaries is controlled to various degrees and in order to properly evaluate the information assurance properties of the architecture we must be able to specify where information is allowed to flow through the

boundaries and any IA mechanisms that are applied to the information as it flows through a boundary.

An example is an enclave that uses an encrypting router to control network traffic. All information flows that pass through the router are encrypted when leaving the enclave and decrypted when entering the enclave. Another example is a partitioned RTOS that provides a port based communications service between partitions. We use a Communication Path model element that is composed of a source port, a destination port and a connection between the ports to model these types of communication mechanisms and the access they provide.. IA Mechanisms may be associated with the ports to define the information assurance services that are provided by the communications path.

The communication path connections are established by connection ports between partitioning groups. These connections are from port to port and establish an allowed communication path between partitioning groups that can be used to route communications between components that reside in the groups. Communication path connections can be seen in Figure 5 directed arcs between the partition structures.

2.2.3 Component Architecture View

In addition to the partition view of the architecture the EDICT-IA tool also provides a component view of the architecture. The component view is implemented with a graphical editor that can visualize hierarchical component based architecture specifications. The editor can display both logical architectures and architectures translated from AADL or UML/SysML.

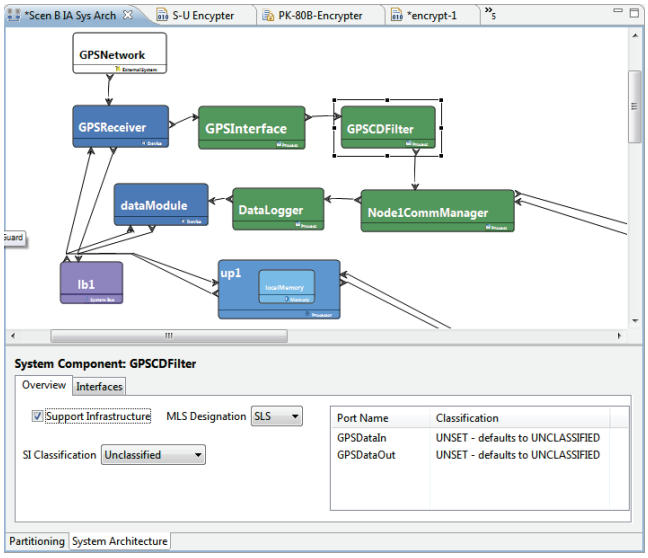


Figure 6 - Component Architecture View

The editor provides methods to view all of the components that compose the architecture, the interfaces they use to communicate and the data that is passed. Figure 6 is a screen shot of the editor showing a portion of a system architecture model that contains both hardware and software elements. The editor also supplies controls to augment the model with additional IA attributes and mechanisms to define the characteristics of the IA services supplied by the components.

2.2.4 IA Mechanisms

Current architecture modeling languages do not contain standardized language extensions to describe the properties and attributes that are critical to define the IA characteristics of the system or how and architecture meets a desired security policy. We have developed a method to augment the architecture models with additional model components that supply this information.

IA Mechanisms are used to represent system services that supply IA related functions for the architecture. They can be used to represent software or hardware functions and can be attached to the architecture by associating them with system components or ports on the components.

An IA Mechanism is a device, service, algorithm, transformation and/or augmentation that is applied in order to meet an IA requirement. The primary attributes of IA Mechanisms are their *strength levels*, which characterize the relative strength associated with the properties that the mechanism is to provide. Examples include encryptors, filters, data replicators, etc.

An *IA Mechanism Set* is a specification of an ordered application of one or more IA Mechanisms. IA Mechanism Sets represent models of security devices or high-level services (or well-defined sequences of such items) that may be reused throughout the system. Mechanism Sets allow for mechanisms to be grouped together to form higher level IA functions.

The EDICT-IA tools supply several categories of IA mechanisms that cover many types of IA services. Within each of these categories users define their own mechanisms to describe the services that are used. A library of mechanisms is stored in the tool that can then be used across many architectures. The tools currently support the following mechanism types:

- ***Encoding*** - Encoding mechanisms are able to encode and decode messages with encryption, hashing or error detection and correction codes. The type of coding that is employed will provide varying strengths in terms of integrity, error correction and confidentiality. There are three purposes that coding is used for:

  Integrity –To ensure the integrity of the data and support detection of unauthorized modification.

  Confidentiality – To prevent disclosure of the information that is coded. This includes all types of encryption.

  Non-Repudiation – To ensure that the sender and/or intended receiver of the encoded message can be authenticated. These fall under the category of identification encoding mechanisms such as message authentication codes or digital signatures.

- ***Cross Domain Solutions*** - Cross Domain Solution mechanisms are able to filter message contents and perform required checks or content modifications. The primary filtering capability that is of concern is the ability to filter messages from one level of classification to another level. Cross Domain Solution mechanisms take information at a classification level and downgrade or upgrade it to another level. The specifications for these mechanisms consist of an input classification and a resulting output classification after the message has passed through.

- ***Access Control*** – Access control mechanisms model Authentication, Authorization and Identification services. The mechanisms are able to model service for user I&A, peer I&A, and server authorization.

- ***Redundancy*** - Redundancy mechanisms produce or consume multiple redundant data flows and compare them for data integrity, or availability purposes. These types of mechanisms include pair wise checks along with voters. These mechanisms have been realized as Replicators

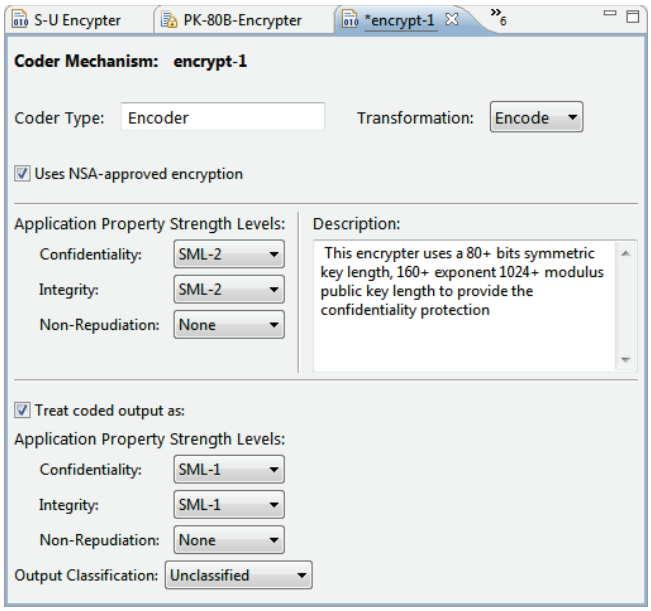


Figure 7 - Coding Mechanism Editor

(creating multiple flows) and Comparators (resolving multiple flows).

- ***Physical*** - Physical Group Security Mechanisms are used to describe the physical security elements in a system. These model elements can be used to define physical mechanisms such as combination locks, guards, manual back-up procedures and other related physical elements.
- ***Network Security*** - Network Security Mechanisms are used to describe the capabilities of network level devices that provide information assurance capabilities to all network traffic that is routed through them. These model elements are used to represent network routers and firewalls.
- ***Partitioning*** - Partitioning mechanisms are used to represent the attributes of the partitioning structures. These mechanisms can be used to represent physical partitions such as hardware machine boundaries or logical software implemented partitions such as in partitioned operating systems. These mechanisms can model partitions with varying degrees of robustness from commercial quality to formally verified and validated implementations.

Each of these mechanism types is supported by a graphical editor that enables the user to define mechanisms and their related information assurance attributes. An example is shown in Figure 7 that shows a screen shot of the coder mechanism editor. The ability to specify mechanisms and assignment to the components and communication ports in the architecture models gives users a very expressive environment in which to define system architectures and their IA services.

2.3 Traceability

The three layers of modeling (IA Requirements, IA Logical Architecture, and IA System Architecture) reflect a refinement of the system specification and design throughout the development process. These models need to be logically consistent between abstraction layers for the architectural analysis methods to be sound and complete.

A basis for establishing logical consistency is formal traceability between the layers of abstraction so that relations between the layers of abstraction can be specified and enforced. Missing or inconsistent elements of the trace can be detected automatically based on a set of traceability rules that ensure that the trace relations are complete and consistent.

The traceability rules for these models specify allowable relations between the models and required model content to ensure that all elements are traced. The rules define the allowable relationships between information domains at the IA requirements layer to a set of system components that implement the functions of the information domain. The rules also specify how cross-domain interfaces are traced to the component interfaces and architectural flows that carry the cross-domain information.

The trace relations are captured in architecture traceability models and persisted in the tool. Each traceability model is supplied with a graphical editor for model composition and a model verifier that enforces the traceability rules. Figure 8 shows the architecture traceability editor being used to trace an IA Domain to a set of architecture components. The view at the bottom of the figure is the results of the model verifier that produces errors and warnings when traceability rules are violated.

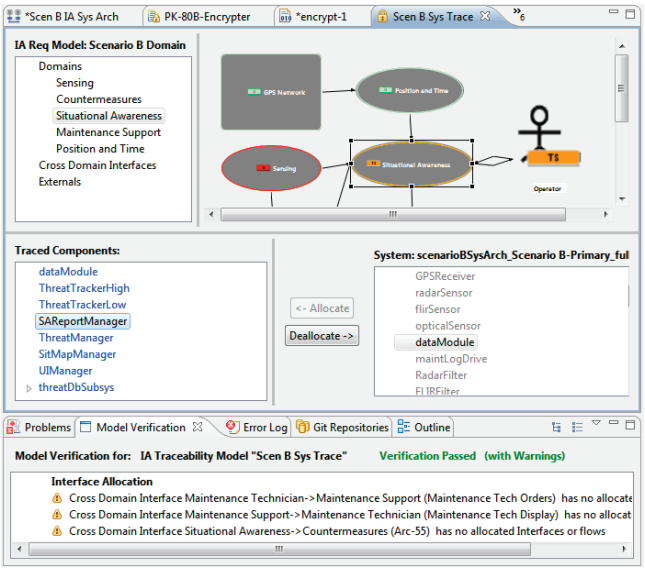


Figure 8 - Architecture Traceability

2.4 IA Architectural Analysis

The key innovation we developed is the architectural analysis methods for information assurance. These methods analyze the properties of system architectures to support the establishment of application level IA properties as outlined in Section 1.1.

We have developed a set of property mappings that specify the property dependencies between layers in the property hierarchy. An example of a mapping for the availability property is seen in Figure 9. In this case, the Availability property depends on the Information Flow, Damage Limitation and Data Isolation properties of the architecture level. The architecture properties in turn rely on the infrastructure properties of the components used to implement the architecture entities. For each application level property the architectural analysis conducted is tuned to the aspects of the architecture that affect the property under evaluation.

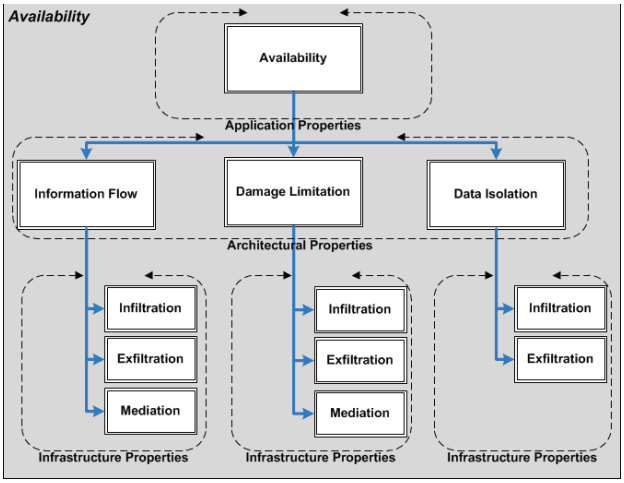


Figure 9 - Property Mappings

For each of the architectural properties we have established a set of analysis methods that evaluate the architecture and its ability to supply the required level of service for the application properties. The goal for analysis of each of these architecture properties is defined as:

1. ***Information Flow*** **-** Ensure that the information flows that are contained in the architecture meet application property requirements across the entire extent of the flow.

   Information flow analyses focus on the properties of information as it flows through an architecture. These analyzers are able to trace the flow of information and determine the IA properties that are present on the flow. The analyzers take into account the changing levels of IA requirements that may be present and the effects of the various IA partitioning and service mechanism that are applied to the data as it traverses the architecture. These analyses are particularly useful in determining if application properties associated with a piece of data are maintained as information is passed.

2. ***Data Isolation*** **-** Ensure the partitioning and data protection services that are specified in the architecture meet application requirements for isolation of data and processing.

   The analyses that support Data Isolation focus on the partitioning in the system and how it supports the isolation of data from unrelated processing or influence. The analyses evaluate many aspects of partitioning to ensure that both the required structure and strength of services are in place to meet application property needs.

3. ***Damage Limitation*** **-** Examine the partitioning structure and data access services to ensure that system requirements for damage limitation are not undermined.

   The analysis methods for this property concentrate on the ability of the architecture to prevent the unwanted changes to data that is stored and handled by the system. These analysis methods evaluate both partitioning aspects and information assurance services that are applied to the data.

4. ***Least Privilege*** **-** Ensure that access to architecture components for a given IA Domain matches the required access specified in the IA Requirements model and ensure that there are no additional data or influence path to the IA Domain components.

The EDICT-IA tool supplies several analyses for each of these architectural properties to ensure that all aspects of the design are covered. The analyzers provide valuable information with regard to partitioning enforcement between domains, flow of information between security domains, presence of multi-level secure components and the use of critical services such as encryption.

Each analysis method is supported by an analyzer component in the tool. The analyzers utilize the information from the requirements, architecture and traceability models to evaluate the architectural properties of the system and how well they support the required application properties. Each analyzer displays its results in a customized format that makes traversal and comprehension of the analyzer results straight forward. A sampling of the analyzer results are shown in Figure 10.

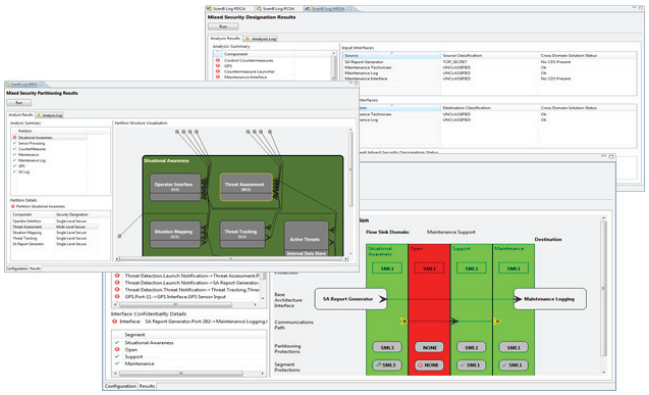


Figure 10 - Analyzer Results

Each analyzer is also equipped with the ability to generate reports in various formats (HTML, PDF, etc.) to facilitate the export of results from the EDICT tools.

The analyzers are built on an infrastructure that enables them to detect when results are no longer valid and to automatically re-run analyzers when any of the models they depend on are modified. This feature is powerful for round trip engineering where analysis results are used to modify architecture models and then analyzers are re-run to assess the impact of the changes.

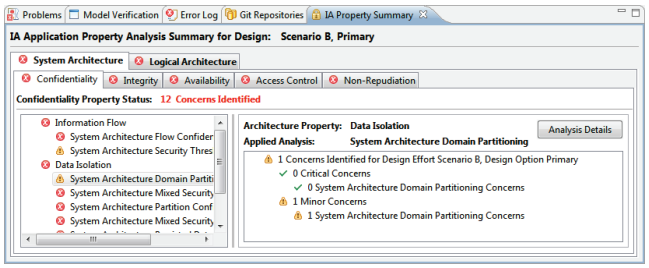


Figure 11 - IA Property Summary View

The analyzer infrastructure also provides for a common view that integrates the results of all 17 analyzers in a single view. This view is useful for quickly identifying problem areas within the architecture and addressing them. Figure 11 shows how the view organizes the results by IA property and provides links back to the analyzers so that result details can be evaluated.

3. Compliance and Certification

The EDICT-IA tools address the challenge of performing model based architectural analysis with the ability to generate evidence that aid in meeting the requirements that are part of a certification process. The results of architectural analysis are important for

communicating system architectures to independent reviewer and certification agents throughout the development and system maintenance phases.

Compliance with IA certification requirements (NIST 800-53, 8500.2, JAFAN 6/3, etc.)[13] [14] [15] [16] [17] requires the presentation of complex assurance arguments that rely on diverse sets of data to show compliance. Similar challenges are present in safety certification standards and processes. Safety cases are a method for presenting evidence to a certification authority that a system is safe to operate. The safety case presents a well-structured argument that is supported by the Goal Structuring Notation to present the elements of the argument (requirements, claims, evidence and context) and the relationships between these elements [10] [11] . It has been found that this type of strategy is also applicable to information assurance. We have developed a capability that uses assurance cases as a method for providing arguments that JAFAN requirements have been met and for supplying the evidence for certification.

The driving philosophy behind the assurance case is to provide a structure that demonstrates how IA claims are decomposed into sub-claims that are supported by evidence. Assurance cases are used to specify the assurance goals for the system under analysis and to construct a logical argument about how the system design, implementation, testing or operating procedures are used to supply the desired assurance goal. Each of these arguments is supported by solutions that are composed of evidence. The evidence supplies the data to support the solution and, in turn, support the argument that the goal has been achieved.

Safety cases have been accepted in many system certification contexts and have been used with success on safety critical systems. A logical extension is to expand this capability to information assurance certification. We believe that the application of assurance case structures to individual JAFAN security assurances and features is a stepping stone that can be used to move the certification community towards a property based approach to certification instead of a compliance based approach.

In order to have well-formed arguments and confidence that the desired system assurance properties are in place, the solution elements to each of the cases must contain evidence that the goals have been met. This is where the GSN and assurance case nomenclature excel at collecting and integrating diverse types of evidence to support the assurance arguments.

The JAFAN 6/3 certification capability allows for evidence in the solution elements be tied directly to data extracted from, or analysis performed by, the EDICT-IA modeling and analysis tools. The integration of model driven design data with the assurance case elements provides a direct path for establishing evidence, updating evidence and re-establishing confidence throughout the life cycle. In cases where evidence can be in the form of analysis results, violations that are detected in the analysis can be used as flags to indicate that claims are no longer true.

4. CONCLUSION

This paper presents novel work in developing a model-based approach that integrates views and tools for analyzing and certifying systems for information assurance attributes. The specific advancements on the state of the art that we have highlighted are:

- A structured, property based approach for evaluating system architectures for information assurance that is able to integrate information assurance properties at progressive levels of abstraction. Property composition is used both for top-down system specification and design activities and bottom up for certification evidence chains.
- An implementation independent representation of system information assurance architectures. The architecture contains the security services and properties that are required to meet system requirements and can be used early in system design cycles to reduce risk and gain confidence or support rapid assessment of system changes and security impacts once a system is fielded.
- A framework that supports traceability and decomposition from requirements down to detailed design models. This framework enables the management and consistency enforcement between the integrated models that compose the framework. The framework enables the integration of standardized modeling languages (AADL, UML) for architecture description to support integration with existing model based development approaches.
- A suite of tools, EDICT-IA, that implement the framework using open tool platforms and modeling languages. The tools provide for graphical visualization of architecture models, information assurance requirements, analysis results and certification data packages. The tools can also evaluate system architectures for Confidentiality, Integrity and Access Control properties. The analysis tools are supported by a change notification infrastructure that enables rapid evaluation of system modifications to support technology re-fresh and system upgrades
- Generation of certification evidence from model based analysis for JAFAN compliance. The JAFAN certification capability provide tools for organizing and presenting certification evidence from both model based data and analysis and external evidence sources such as reports and artifacts from other tools.

These advancements provide a significant advance in the state of the art for the design and evaluation of information assurance aspects of system architecture. These techniques and tools can be applied throughout the development process to yield benefits for reduction of risk, cost and schedule. The innovative modeling and analysis techniques provide system developers and certifiers with new capabilities to represent and evaluate the information assurance properties of system architectures.

5. REFERENCES

[1] J. Rushby. Design and verification of secure systems. In *Proceedings of the Eighth Symposium on Operating Systems Principles*, volume 15, December 1981.

[2] David Greve, Matthew Wilding, and W. Mark Vanfleet, "A Separation Kernel Formal Security Policy", *Fourth International Workshop on the ACL2 Prover and Its Applications (ACL2-2003)*, Boulder, CO, July 2003.

[3] W. Mark Vanfleet, et al., "Deeply Embedded High Assurance (Multiple Independent Levels of Security/Safety) MILS Architecture," http://www.omg.org/docs/security/02-11-02.pdf.

[4] John Rushby, "Partitioning for Safety and Security: Requirements, Mechanisms, and Assurance," NASA Contractor Report, CR-1999-209347, June 1999

[5] W. Mark Vanfleet, et al., "MILS:Architecture for High Assurance Embedded Computing,: CrossTalk, August 2005.

[6] EDICT, http://wwtechnology.com/EDICT/

[7] Walter, Chris; Brian LaValley, Peter Ellis; October 2012, "The EDICT Tool Platform For Model Based Architecture Modeling And Analysis, " in the 31st Digital Avionics Systems Conference, October 14-18, 2012

[8] J. Alves-Foss, W. Harrison, P. Oman, C. Taylor, "The MILS Architecture for High-Assurance Embedded Systems", International Journal of Embedded Systems, February 2005.

[9] J. Rushby, "Proof of Separability: A verification technique for a class of security kernels", *Proc. of Int. Symposium on Programming Lecture Notes in Computer Science*, vol. 137, pp. 352-367, 1982.

[10] Kelly, T. "Using Software Architecture Techniques to Support the Modular Certification of Safety Critical Systems", *11th Australian Workshop on Safety-Related programmable Systems*, Proc. SCS'06, 2006.

[11] Kelly, T. , Weaver, R. *The Goal Structuring Notation – A Safety Argument Notation,* University of York, http://www-users.cs.york.ac.uk/tpk/dsn2004.pdf.

[12] "Information Assurance Technical Framework", National Security Agency, Version 3.1, September 2002.

[13] Recommended Security Controls for Federal Information Systems and Organizations, NIST Special Publication 800-53 Revision 3, National Institute of Standards and Technology, August 2009.

[14] Joint Air Force – Army – Navy 6/3 Manual, October 2004.

[15] Joint Air Force – Army – Navy 6/3 Implementation Guide, September 2006.

[16] DoD Directive 8500.1, Information Assurance, October 2002.

[17] DoD Instruction 8500.2, Information Assurance Implementation, February 2003.

Secure Service Composition Adaptation Based on Simulated Annealing

Bo Zhou, David Llewellyn-Jones, Qi Shi,
Muhammad Asim, Madjid Merabti, David Lamb
School of Computing and Mathematical Sciences
Liverpool John Moores University
Byrom Street, Liverpool, L3 3AF, United Kingdom
{B.Zhou, D.Llewellyn-Jones, Q.Shi, M.Asim, M.Merabti, D.J.Lamb}@ljmu.ac.uk

ABSTRACT

Secure adaptation of service composition is crucial for service-oriented applications. An effective adaptation method must improve a composition's adherence to specified behaviour, performance and security guarantees at reasonable cost in terms of computing complexity, time consumption and so on. This demonstrates a general class of problems within the sphere of secure service composition adaptation to be NP-complete by reducing it to the 0/1 knapsack problem. The theory of simulated annealing is introduced and revised as a heuristic method for tackling the problem. As a prerequisite, we also propose a novel approach for service quantification in order to make sensible comparisons between services. The aim of this work is to optimise composition adaptation results quickly and accurately.

Keywords

Secure Composition Adaptation, Simulated Annealing, Heuristic Method, Knapsack Problem, NP-complete

1. INTRODUCTION

A service-oriented architecture (SOA) provides the platform for services from different providers to work together. SOAs offer new applications via composition of services; facilitated by standardised interoperations among services. An important issue that casts a shadow over SOA platforms and application development is that of security. Concerns around inconsistent security policies and configurations must be continually monitored and addressed through adaptation.

As part of the work undertaken for the Aniketos project [1], we carried out a study on existing techniques relevant to secure service composition and adaptation. The results reveal that while many efforts have been made in different areas to support SOAs, a practical solution is still missing for composition adaptation, which plays a crucial role in the secure service composition process [14].

In this paper we consider this issue from the angle of theoretical computer science and present a novel approach for secure service composition adaptation. While in some specific cases there are efficient solutions that can address the composition adaptation issue, the problem in general can be proven to be NP-complete (non-deterministic polynomial-time complete) [10] with the assistance of verification techniques. Therefore we demonstrate how a revised heuristics-based method using simulated annealing [13] can be applied to optimise the adaptation process. By abstracting and quantifying security properties of services, this method also allows direct comparison of services with user preferences. This work aims to optimise the composition adaptation result with improved efficiency.

The rest of the paper is organised as follows. The next section briefly introduces existing service composition and verification techniques, as well as the issue of composition adaptation. The principles of NP-complete problems and simulated annealing are explained in Section 3. Section 4 proves the composition adaptation issue to be NP-complete and presents a novel idea for quantifying services in respect to their security. These eventually contribute to a revised solution based on simulated annealing. The paper concludes with a brief review in Section 5.

2. VERIFICATION AND ADAPTATION

2.1 Standards and Modelling Languages

In this section, we briefly introduce existing techniques that help contribute to secure service composition functionality. SOA platforms provide a foundation for modelling, planning, searching for and composing services. They specify the architectures required, as well as providing tools and support for service composition standards. Enterprise servers such as Glassfish or Microsoft IIS do offer security parameterisations, but these are typically domain or platform-specific [8].

Therefore, in order to facilitate service composition across platforms, service modelling languages are used to describe a) the business requirements of a system and b) system resources. By expressing behaviour processes and system organisation in agreed formats, compositions can be validated against desired criteria and modified to suit required changes in operation.

A number of languages relevant to service composition already exit. Business Process Modeling Notation (BPMN) can be used to model relationships of composed services within a business model or process workflow [15]. Web Services Description Language (WSDL) is widely used for describing Web service interfaces [9]; subsequent standards augment the basic description to add semantic, behavioural, and to a limited extent, authentication and security data [5]. Other such property-based extensions, including Unified Services Description Language (USDL) [18], constitute stan-

dards that target trust and security, to bridge the previously-identified vendor divide. Security-domain-specific modelling languages are also used to specify security requirements and policies. For example, S×C [11] is able to support programmable negotiation and monitoring of services with regard to agreed contracts.

2.2 Verification and Monitoring

Once services are selected for composition through a SOA framework, verification is used to ensure selected services comply with the security policy at both design- and run-time. At design-time, formal analysis and proof-based techniques can establish that stated service behaviour is compliant. When services are operating, monitoring mechanisms are required to verify the composition's behaviour at run-time. If changes in service behaviour or operating context could affect the stated composition behaviour, countermeasures have to be taken, possibly including re-composition or adaptation.

There are already a variety of existing tools for performing analysis of individual or composed services. The Deploy [3] tools support formal analysis of security properties, focussing particularly on the design of systems satisfying strict formal specifications. The heuristic method developed in this paper does not obviate the need to apply formal methods for analysing services, but instead builds of them, while at the same time aiming to make certain elements (specifically adaptation based on determined properties) more efficient.

Similarly, AVANTSSAR [2] is a tool for reasoning on service compositions based on a formal language for specifying trust and security properties. PRRS [4] is another tool that supports run-time service monitoring and substitution. At present however these do not incorporate heuristic techniques such as those proposed here for dealing with service adaptation. We envisage this work could potentially be incorporated into tools such as these.

In our previous work [19] [20], we also developed a service composition verification platform that is able to highlight the potential security issues for composite service. The Mobile Agent Topology Test System (MATTS) has been designed primarily to allow modelling of system-of-systems scenarios and testing of secure service composition analysis techniques. It supports analysis of various security-associated risks for composite services. This analysis in MATTS is based on a special scripting language that allows the incorporation of both service properties and the interactions between services to be considered. For example, a MATTS script file can specify what kind of encryption algorithm a service should adopt, which particular service is allowed to make connections, or even which role is assigned to access the service. Each script file specifies a set of rules that determines whether the composition satisfies a particular security property. Once a script is loaded, the MATTS backend engine will measure the scenario against its rules. If the measurement result indicates a violation of a rule, the engine will notify the user. This process repeats every time when the scenario evolves and it only ends when order received to stop. It is worth noting that the verification is based on the assumption that the information provided by the user is correct. MATSS does not carry out any formal verification although it is possible to do it.

2.3 Issue with Composition Adaptation

Composition adaptation aims to mitigate issues identified during verification and monitoring. Available actions normally include rearranging, adding or removing components. Although adaptation options seem adequate, deciding the best action is complex. Because service behaviour may differ owing to differing implementations and their various security properties (*e.g.* the host system's status), a simple substitution of a service may result in dramatic changes to the security properties of the entire composition.

For example, assume a scenario where Service A uses Blowfish [17] as the encryption algorithm to communicate with other services in a composition. Later on more stringent trustworthiness requirements are introduced and a greater strength of encryption is mandated. A replacement Service B is found that satisfies the new trustworthiness requirement through support for an alternative set of stronger symmetric encryption algorithms. However, it transpires that the only common algorithm supported by Service B and the services it communicates with is 3DES [7] (considered to have similar strength to Blowfish). Thus its communication with other services cannot be satisfactorily secured because the security level offered by these encryption algorithms is still lower than that required by the new trustworthiness requirements. The entire composition is consequently considered to be vulnerable in terms of confidentiality, even though individually the services all fulfil the requirements.

Therefore the secure composition adaptation issue is actually an optimisation problem with constraints. Although a brute force approach to composition adaptation might simply cycle through and verify each of the possible solutions until a composition with adequate security characteristics is found, this is likely to be computationally impractical when the number of services is large. In fact, as will be further explained in the following sections, the adaptation issue can be shown to be NP-complete. In the next section we will give a further explanation about the problem of NP-completeness and introduce an heuristics-based method - simulated annealing - that can be used to augment existing secure composition verification techniques.

3. COMPUTATIONAL COMPLEXITY

3.1 The 0/1 Knapsack Problem

In the field of theoretical computer science, it has been demonstrated that some optimisation problems have no straightforward solutions that are computationally feasible, *i.e.* that no polynomial-time algorithms are known for solving them. These problems are defined as NP-complete if they can be solved in polynomial time [10].

A typical example of an NP-complete problem is the 0/1 knapsack problem [16]:

Problem 1. Given a set of items, each with a weight and a value, determine the items to be included in a combination so that the total weight is less than or equal to a given limit

(*e.g.* the capacity of a knapsack) and the total value is as large as possible.

The knapsack problem is one of the most intensively studied issues in optimisation. One obvious solution is to try out permutations of all the possibilities. The one with greatest value under the weight limit is ultimately the optimum answer. This solution is easy to implement but consumes considerable computing power in order to complete as the number of items increases. When the number of items is large, the amount of possible combinations becomes unmanageable. Its efficiency is $\mathrm{O}(n!)$ where n is the number of items to choose from.

The mathematical expression of the knapsack problem is as follows.

$$\max V = \sum_{i=1}^{n} v_i x_i, \tag{1}$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq W,$$

$$\text{where } n = \text{number of items};$$

$$v_i = \text{value of item } i;$$

$$w_i = \text{weight of item } i;$$

$$x_i \in \{0, 1\}.$$

Because there is no way to immediately tell if a particular combination is optimal, the problem thus cannot be either solved or verified in polynomial-time. However, if we change the question slightly to a decision problem as in the following statement, a particular combination can then be verified easily in linear time and the problem becomes NP-complete.

Problem 2. Given a set of items, each with a weight and a value, determine whether a particular combination of items has greater value than V without exceeding a weight limit W. The value V is the greatest value previously achieved with constraint W.

3.2 Simulated Annealing

Heuristics-based methods solve NP-complete problems based on experience or convergence, where exhaustive search for a solution is impractical. In contrast to the brute force approach, heuristic methods select compositions 'intelligently', attempting to find increasingly successful solutions until an adequate answer is found. Among these algorithms, simulated annealing is widely used to solve NP-complete problems in general. While heuristics cannot guarantee that the correct solution will be found in all cases, they can offer achievable, practical approaches for finding the correct solution in the vast majority of cases.

Simulated annealing is named to mimic the chemical process when atoms in metal become placed in relative order during the process of being overheated and gradually being cooled. Rather than finding the optimum solution, simulated annealing ends up with a very good solution. It balances efficiency with the accuracy of the result.

Now in the example of the knapsack problem, instead of exploring all the possibilities, simulated annealing starts with a random combination that satisfies the weight constraint, then searches for a close alternative that has greater value. Here the 'close alternative' means the new combination has minimal change to the original combination. If the new combination is better (while satisfying the weight constraint), then we replace the original one and repeat the process of close searching. Finally we will end up with a local optimum, which means the best solution we can find confined by the context of the original randomly chosen solution.

To limit the damage caused by the local optimum issue and ensure a broad solution space is explored, simulated annealing can also accept solutions that appear worse than the present solution. In this way, it's possible to increase the likelihood that a global optimum will be achieved.

To be specific, the process of solving the knapsack problem is described below [16]:

1. Choose any feasible combination randomly.
2. Randomly select an item from the original items set n.
3. If the item has not been collected, add the item to the combination. If the new combination does not satisfy the weight constraint, randomly remove an item from the combination until it becomes feasible.
4. While in step 3 if the item has already been collected, remove this item and pick up another item from the items set that forms a new feasible combination.
5. If the new combination satisfies equation 2, then the new combination will be accepted.

   $$e^{\frac{\Delta V}{T}} > R(0, 1) \tag{2}$$

   where ΔV represents the change of value, T is the simulated temperature value that confines the optimisation process, e is the mathematical constant and $R(0, 1)$ generates a random number between 0 and 1.
6. For the value function V, if ΔV is a positive, that means the new combination has greater value and equation 2 is always true; if ΔV is negative, than the new combination has lower value. However, it is still possible that the new combination will be accepted. In effect, with larger T, the chance of satisfying equation 2 is higher.
7. Repeat steps 2-6 several times, decreasing the simulated temperature value of T at each step until a lower threshold for the value of T is reached. T plays the role that confines the speed and scope of the optimisation process.
8. When T reaches the threshold value, the current solution will be the best one we can get.

Based on this principle, in the next section we will explore how to apply the simulated annealing approach on secure service composition adaptation and describe what is needed in order to do so. We first prove the NP-completeness of

service composition adaptation issue and propose a quantification method to allow direct comparison of service compositions.

4. SOLUTION BASED ON SIMULATED ANNEALING

4.1 Proof of NP-Completeness

To justify the application of simulated annealing to the secure service composition adaption process, a crucial step is to prove that the problem is indeed NP-complete. By definition [12], a decision problem D is NP-complete if

1. D is in NP, and
2. Every problem in NP is reducible to D in polynomial time.

For the first requirement, D can be shown to be in NP by demonstrating that a candidate solution to D can be verified in polynomial time; for the second requirement, we need to prove D is at least as hard as any known NP-complete problem.

To generalise the secure composition adaptation problem, we assume a composition C is composed from m services $C = \{S_1, S_2, \ldots, S_m\}$. For each service, there are n alternatives $S_1 \in \{S_{11}, S_{12}, \ldots, S_{1n}\}$; $S_2 \in \{S_{21}, S_{22}, \ldots, S_{2n}\}$ and so on. The alternative services normally all satisfy the requirements in terms of functionality, but have various characteristics and impacts in terms of security. The secure service composition adaption issue can be summarised as in the following definition.

Problem 3. Given a set of alternative services, what is the best combination of them in order to achieve the most secure service composition?

We mentioned in Section 2.2 that some verification techniques can verify whether a service composition satisfies a user's security policy. However, it is not enough to answer the above question as there is no way to tell immediately if a particular composition is the best combination unless we try out all of the possible permutations. Therefore we transform the issue slightly into a decision problem as stated below. Instead of finding the best adaption solution, our approach is to use the verification tool and a novel comparison method to strategically adapt the composition towards a better solution. In this way, the secure composition adaptation problem can be verified in polynomial time and the first requirement of NP-completeness is satisfied. More details about the comparison method can be found in the next section.

Problem 4. Given an alternative composition, is it securer than the most secure composition previously known?

Similar to the knapsack problem, in reality the collection of component services is limited by other constraints such as cost. In addition to this the user may also put extra requirements on service trustworthiness, running efficiency and many other QoS and security related factors. This is known as the 0/1 multidimensional knapsack problem [16]. To simplify the issue, we assume all other constraints are stated in the security policy and verified by the verification tool such as MATTS. In this paper we only look at the typical business decision where the user has to balance both security and cost, however this can be easily extended to the more general case. Therefore the problem can be further refined as follows.

Problem 5. Given an alternative composition, is it securer than the most secure one previously known, while remaining within the user's budget?

Now for the second requirement, if we can prove that the issue of secure service composition adaptation is harder than the knapsack problem, its NP-completeness will be proved. Again, assume there are m services in the composition C where $C = \{S_1, S_2, \ldots, S_m\}$. For each service there are n alternatives. We start with the simplest case where there is only one alternative for each service, and label these services S'_1, S'_2,...,S'_m respectively. If we can prove the simplest case is as hard as the knapsack problem, increase the alternative number n will only make it harder. Now assume each service is associated with a security evaluation value s_i and a price value p_i, and that their alternatives have value s'_i and p'_i. The mathematical expression for the composition optimisation issue then becomes

$$\max S = \sum_{i=1}^{m} (s'_i - s_i)x_i, \tag{3}$$

$$\text{subject to} \quad \sum_{i=1}^{m} (p'_i - p_i)x_i \leq P,$$

where m = number of services in C;
$s'_i - s_i$ = change of security if service i replaced;
$p'_i - p_i$ = change of price if service i replaced;
P = threshold value determined by user's budget;
$x_i \in \{0, 1\}$, decision virable with service i.

In equation 3, $x_i = 0$ means service S_i will not be changed; $x_i = 1$ means the service S_i will be replaced by S'_i. Comparing this equation with equation 1, the only difference is that 'change of security' and 'change of price' in equation 3 can be negative values, while in equation 1 the value and weight are always positive. This difference can be eliminated by adding a constant value at the both sides of the equation to insure the results are always positive. Therefore mathematically these two equations have the same complexity. Consequently the secure service composition adaptation problem is reducible to the 0/1 knapsack problem and its NP-completeness is proved.

It is worth noting that in equation 3 we assumed each service has a security evaluation value s_i and the composition's security value S is the sum of its component services' security values. This is also a simplified assumption and the real situation could be more subtle. However we believe the

correctness of equation 3 will not be affected as long as the relationship between the composition's security value and it component's value follows a linear or non-linear growth model. For example, replacing a component service with another that has higher Availability always likely to increase the Availability of the composition. In this paper we do not focus on any particular security properties. Instead, we try to work out a general solution that applicable to most of the cases, assuming the linear or non-linear growth model will be held. Therefore we will now go on to explain the meaning and calculation method for the security values s_i and composition security value S.

4.2 Quantifying Services

Security is a broad concept that includes many aspects such as confidentiality and privacy. One composition may be stronger than another in terms of confidentiality; while it is also possible that the very same composition has weaker protection for privacy. To tell if a new composition is better than another, we have to find a way to determine the value of S in equation 3.

The key to a quick and accurate analysis of service composition is to formalise the security property descriptions. A service description can cover various features about the service. For example, one feature may state 'Communication with others is secured', while the other one may state 'My estimated availability ratio is 90%'. Semantically the features can be described in very different ways. Although some policy languages have been proposed to formalise service descriptions [6], the content of the actual description is still very open. Due to this lack of confinement, the comparison of different services and their compositions can be very difficult. It is necessary to have a formalised description of services, both semantically and quantitatively. For example, in the above cases the text descriptions can be translated to statements like 'secured communication = true; (optional: encryption algorithm = 3DES)' and 'availability = 0.9'. The subjects at the left hand of the equations, *i.e.* 'secured communication', 'encryption algorithm' and 'availability' in this case, should be predefined, and their format/data type should have also been determined in advance. This will allow a platform to be built that can measure and compare different services and their compositions without human intervention.

To judge if a service composition is more secure than another, we propose the concept of *security impact factors* to standardise security properties such as those mentioned above. We choose five commonly used factors, namely Confidentiality, Integrity, Availability, Accountability and Non-Repudiation, to measure a service's security status. For example, 'secured communication = true' could contribute 0.5 to Confidentiality and 0.3 to Integrity; 'encryption algorithm = 3DES' could contribute another extra 0.1 to Confidentiality. A security property may have impact on more than one security factors such as Confidentiality and Integrity. The security impact factors are predetermined based on expertise, available as a special dictionary that translates from security property to security impact factor. Therefore the process of calculating security impact factors can be fairly fast. Moreover, in situations where new threats are identified or there is a breakthrough in the security protection techniques, the security impact factors can be updated and maintained easily with the dictionary. At the end of the evaluation process the services will be quantitatively estimated by these five key security factors, which are also the main concerns that are likely to be specified in a user's security policy at the business level.

Once the five security impact factors have been calculated we can give users the flexibility to adjust the weight of these five factors. This is important since in different scenarios the user's priorities may change. The weight could be any number between 0 and 1. Now assume a user sets the weights to 0.4, 0.2, 0.1, 0.1 and 0.2 respectively for the aforementioned five factors, the overall security value S for the service will be:

$$S = 0.4 \times C + 0.2 \times I + 0.1 \times V + 0.1 \times A + 0.2 \times R \quad (4)$$

where C represents the value of Confidentiality, I represents Integrity, V represents Availability, A represents Accountability and R represents Non-Repudiation.

4.3 Proposed Method

Building on the concepts established in the previous sections, we can now apply the principle of simulated annealing to secure service composition adaptation. To a certain extent, the resulting solution will reflect the user's preferences based on the weights he/she sets for the five security factors. Accordingly, the proposed solution has the following steps:

1. Choose a feasible list of services that can be replaced randomly.
2. Randomly select a service from the original composition.
3. If the service has not been included in the list for replacement, add the service to the list. Randomly choose alternative services from service pool for each service on the list and form a new composition. If the new composition does not satisfy the cost constraint, randomly remove a service from the replacement list until it becomes feasible.
4. While in step 3 if the service has already been selected, remove this service from the list and pick up another service from the original composition. Randomly choose alternative services to form a new feasible composition.
5. If the new composition satisfies equation 5, as well as passing the security verification test of a tool such as MATTS, than the new composition will be accepted:

$$e^{\frac{\Delta S}{T}} > R(0,1). \quad (5)$$

   where ΔS represents the change of security impact value calculated based on equation 4, T is the simulated temperature value that confines the optimisation process, e is the mathematical constant and $R(0,1)$ generates a random number between 0 and 1.
6. Let S be the security impact value. If ΔS is a positive, this means the new composition has a higher security score and equation 5 is always true; if ΔS is negative,

than the new composition has lower score. However, it is still possible that the new composition with negative ΔS can be accepted. In effect, with larger T, the chance of satisfying equation 5 is higher.

7. Repeat steps 2-6 several times decreasing T at each step until a lower threshold for the value of T is reached.

8. When T reaches the threshold value, the current solution will be the best one we can get.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we first demonstrated that the secure service composition adaptation issue is an NP-complete problem with the use of verification tool. A revised heuristics-based method, namely simulated annealing, was proposed to achieve a faster and more efficient adaptation process. In contrast to the intuitive brute force approach, this method can narrow down the choice of adaption with faster convergence. In order to achieve our goal, we also proposed a novel approach that abstracts and quantifies service compositions into five key security aspects: Confidentiality, Integrity, Availability, Accountability and Non-Repudiation. By prioritising these five aspects with different weights, a user can have influence on the resulting solution. We aim for the proposed method to serve as a starting point for achieving more effective secure adaptation of service compositions in the future.

It is worth noting that in this paper we prove the problem as NP-complete at the dimension of the number of service in a composition. Actually additional complexity arises from another dimension, most notably when the choice of alternative services increases. Further research effort is needed in order to improve the adaptation strategy by taking the number of alternative services into account.

Moreover, in this paper we only consider one constraint – that of cost – while assessing the new composition. This can be extended to include more constraints such as trustworthiness. As we explained, the problem will then become similar to the 0/1 multidimensional knapsack problem.

The justification for the use of simulated annealing presented here as a way of overcoming the inherent complexity of the problem is based on its similar application in other problem spaces. However, although we believe this offers a viable alternative to a brute force approach, the nature of the process means that the real benefits can only be properly understood through practical testing. This remains a task to be completed, and an important part of our future work will therefore be the practical validation of the technique in order to understand the speed and accuracy with which it converges on the optimal solution.

Finally, evaluating a composition that comprises a number of services is another research milestone. We made the somewhat simplified assumption in this paper that the security impact value of a composition is the sum of its component services' impact values. As explained earlier the actual case could be very different from one security property to another, depending also on how the component services are composed. Therefore it is unlikely that a single formula can establish a complex composition's properties. Determination of a composition's security properties is beyond the scope of this paper.

6. ACKNOWLEDGMENTS

The research leading to these results has received funding from the European Union Seventh Framework Programme (FP7/2007-2013) under grant no 257930 (Aniketos).

7. REFERENCES

[1] Aniketos project. *http://www.aniketos.eu/*.
[2] Avantssar website. *http://www.avantssar.eu*.
[3] Sap netweaver composition environment. *http://www.sap.com/platform/netweaver/*.
[4] Serenity project. *http://cordis.europa.eu/projects/rcn/78381_en.html*.
[5] I. R. Akkiraju and et al. *Web Service Semantics - WSDL-S*. 2005.
[6] I. Aktug and K. Naliuka. Conspec - a formal language for policy specification. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 197(1):45–58, February 2008.
[7] W. C. Barker. Recommendation for the triple data encryption algorithm (tdea) block cipher (version 1.1). *NIST Special Publication*, pages 800–67, May 2008.
[8] S. W. Chan. *Security Annotations and Authorization in GlassFish and the Java EE 5 SDK*. 2006.
[9] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. *Web Services Description Language (WSDL) 1.1*. 2001.
[10] S. Cook. The complexity of theorem proving procedures. In *Proceedings of the 3rd ACM symposium on Theory of computing*, pages 151–158, 1971.
[11] N. Dragoni and et al. Security-by-contract (sxc) for software and services of mobile systems. *At Your Service: Service-Oriented Computing from an EU Perspective*, pages 429–454, 2009.
[12] R. Karp. *50 Years of Integer Programming 1958-2008: Reducibility among combinatorial problems*. Springer, 2010.
[13] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science, New Series*, 220(4598):671–680, May 1983.
[14] P. H. Meland, J. B. Guerenabarrena, and D. Llewellyn-Jones. The challenges of secure and trustworthy service composition in the future internet. In *Proceedings of the 6th International Conference on System of Systems Engineering (SoSE 2011)*, pages 27–30, Albuquerque, New Mexico, USA, June 2011.
[15] OMG. *Business Process Model and Notation 2 Specification*. 2011.
[16] F. Qian and R. Ding. Simulated annealing for the 0/1 multidimensional knapsack problem. *Numerical Mathematics*, 16:320–327, 2007.
[17] B. Schneier. Description of a new variable-length key, 64-bit block cipher (blowfish). In *Proceedings of the 2nd International Workshop on Fast Software Encryption*, pages 191–204, Leuven, Belgium, December 1994. Springer-Verlag.
[18] W3C. *W3C Incubator: Unified Service Description Language*. 2005.

[19] B. Zhou, A. Arabo, O. Drew, D. Llewellyn-Jones, M. Merabti, Q. Shi, A. Waller, R. Craddock, G. Jones, and A. Yau. Data flow security analysis for system-of-systems in a public security incident. In *Proceedings of the 3rd Conference on Advances in Computer Security and Forensics (ACSF 2008)*, Liverpool, UK, July 2008.

[20] B. Zhou, O. Drew, A. Arabo, D. Llewellyn-Jones, K. Kifayat, M. Merabti, Q. Shi, R. Craddock, A. Waller, and G. Jones. System-of-systems boundary check in a public event scenario. In *Proceedings of the 5th International Conference on Systems of Systems Engineering*, Loughborough, UK, June 2010.

Atomizer: Fast, Scalable and Lightweight Heap Analyzer for Virtual Machines in a Cloud Environment

Salman Javaid
University of New Orleans
New Orleans, LA, USA
sjavaid@uno.edu

Aleksandar Zoranic
University of New Orleans
New Orleans, LA, USA
azoranic@uno.edu

Irfan Ahmed
University of New Orleans
New Orleans, LA, USA
irfan.ahmed@uno.edu

Golden G. Richard III
University of New Orleans
New Orleans, LA, USA
golden@cs.uno.edu

ABSTRACT

In recent years process heap-based attacks have increased significantly. These attacks exploit the system under attack via the heap, typically by using a heap spraying attack. A large number of malicious files and URLs offering dangerous contents are potentially encountered every day, both by client-side and server-side applications. Static and dynamic methods have been proposed to detect heap-based attacks in the literature, using various methodologies like NOZZLE. The main drawback with existing techniques is that they either consume too many resources or are complicated to implement. In this paper we propose Atomizer, which offloads process heap analysis for guest VMs to the privileged domain using Virtual Machine Introspection (VMI). Atomizer APIs can be used to implement various heap analyzing algorithms on processes running inside a VM. A simple heap-spray detection algorithm using Atomizer was implemented to determine the effectiveness of Atomizer. Use of Atomizer cannot be detected by in-guest VM malware, has minimal impact on the cloud server, is very effective in detecting heap spraying malwares, and is simple to deploy. Our architecture is particularly applicable to cloud environments where virtualization is used to host guest VMs.

Categories and Subject Descriptors

K.6.5 [**Security and Protection**]: Invasive software

General Terms

Security, Measurement, Experimentation

Keywords

Xen, Introspection, Cloud Computing

1. INTRODUCTION

Recent advances in operating system-based security mechanisms like Address Space Layout Randomization (ASLR) [22] and Data Execution Prevention (DEP) [3], as well as compiler-based security mechanisms like stack protection [13] have made it more difficult for malware developers to inject code to exploit computer systems. This is why more and more hackers have turned to heap-based techniques such as heap spray attacks [6] and just-in-time (JIT) spraying [11] to bypass these security mechanisms. These techniques rely heavily on the heap of a process to bypass protections and execute the exploit. This makes monitoring the activities that occur in the heap extremely important. Unfortunately, monitoring heap space is a resource intensive task and many heap monitoring tools do not examine the heap contents for performance reasons, instead relying on hooking or diverting system calls to detect malware in the heap.

In this paper we present Atomizer, which browses through the heaps of processes running inside VMs and looks for heap-based activities like heap spray. Atomizer runs on a privileged VM and uses VMI to access the heaps of the processes inside virtual machines running on a cloud server. Currently, only the heaps that are allocated by the operating system for the processes can be accessed by Atomizer. Application-generated heaps (e.g., those created by the Java Virtual Machine (JVM)) can also be examined by Atomizer, through available Atomizer APIs. Atomizer requires no changes to be made to virtual machine manager(VMM), VMs and the programs being monitored. Atomizer's architecture makes it difficult for any malware inside the VM to detect and disable it. Atomizer also monitors all the pages of the heap that have been swapped out of main memory. Atomizer is designed to be modular, so that new features can easily be added. The most popular type of heap-based attacks are heap sprays [19], that's why in this paper we are mainly focused on heap spray attacks, although Atomizer can be used for detecting any heap-based attacks.

Heap Spraying techniques involve instructing client side languages such as JavaScript to allocate large blocks of heap memory (50-200MB) containing malicious shellcode and NOP sleds that "slide" into the shellcode. The final piece of the puzzle relies on overwriting a function pointer to point to a random location within the large NOP sled heap object [2],

which typically requires a separate exploit. Large blocks of heap spray can easily be detected by simple scans, however, newer and less intrusive heap spray attacks that more accurately manipulate the layout of the application heap layout increase reliability and precision, without the need for large blocks of heap spray. Techniques like Feng Shui Heap Spray [23] defragments and makes holes in the heap object to insure that the function pointer is readily available and positioned properly for smashing with a heap overflow [15].

Techniques like JIT spraying [11] have been developed to bypass both ASLR and DEP. JIT spraying utilizes knowledge about a JIT compiler's architecture to spray the heap with executable code that can then be compiled by the JIT compiler. The JIT heap spray is constructed large enough to overwhelm and bypass ASLR. JIT spray uses return oriented programming (ROP) [20] gadgets to mark the heap pages as executable so that the contents of the pages can be executed. JIT spray uses a leaked pointer, which is essentially a random heap address, to jump to that location and follows the NOP sled down to the JIT shell code that is executed to exploit the system. Modern exploitation techniques, such JIT spraying, makes it even more important to look for malicious activities inside the heap of a process.

The rest of the paper is organized as follows: Section 2 summarizes related work. Section 3 presents Atomizer and details its architectural assumptions. Section 4 outlines the implementation and Section 5 the evaluation. Section 6 concludes the paper.

2. RELATED WORK

There has been a large volume of work published on heap spray detection. Most of the work focuses on JavaScript analysis and classifying web contents on the basis of dynamic and static features. This section covers the existing approaches and tools that are most related to Atomizer.

Cova *et al.* propose JSAND [12] that provides a framework to emulate JavaScript code and discover the key features that are most commonly found in malware. These features include code obfuscation, environment analysis techniques, and exploitation mechanisms. JSAND uses machine-learning techniques to establish the characteristics of normal JavaScript code and uses these characteristics to detect anomalous code [12]. JSAND is a finger printing technique that can be evaded by a clever malware developer. Techniques like time-based checks and exception handling described in [17] can be used to evade the emulation part of JSAND. Unlike Atomizer, JSAND relies heavily on emulation which can have many limitations [17].

Ratanaworabhan *et al.* propose NOZZLE [19] which performs static control flow analysis of parts of the heap, interpreting them as code segments to detect malicious content. NOZZLE accomplishes this by intercepting common heap allocating function calls and gathering information about the heap, as well as its content. This technique is browser-specific and could potentially be detected by the attacker. As described by NOZZLE, an attacker could time her heap sprays to avoid detection by this mechanism. Scanning heap objects via introspection requires no memory hooks and makes the entire guest operating system oblivious to heap analysis. NOZZLE also poses a 10% overhead, which has led to the development of an improved version of the programs called ZOZZLE [14].

LeMasters [18] demonstrates a simple Heap Inspector tool that visualizes heap sprayed NOP sleds by searching for byte patterns that resemble NOP instructions. It does so by injecting a DLL into the process that is being analyzed to create a gateway to the heap object. It further relies on the windows API to gather the heap data. All this combined not only significantly impacts the performance of the guest system, but also is detectable by malware.

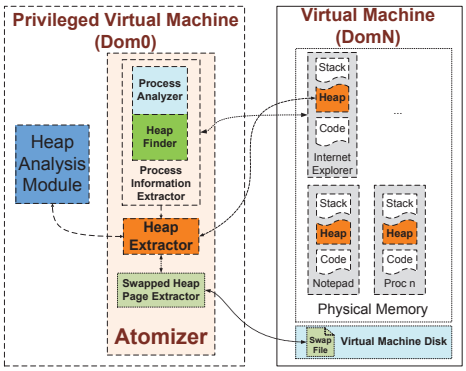


Figure 1: Atomizer Architecture

3. ATOMIZER ARCHITECTURE

The Atomizer infrastructure consists mainly of three components, Process Information Extractor, Heap Extractor and Swapped Heap Page Extractor as shown in Figure 1.

The first component, Process Information Extractor, is responsible for determining the basic attributes of the process we are monitoring. These basic attributes of the process include the internal operating system structures that provide us with valuable information about the location of the process in physical memory. The location of the heap is extracted in the second step called Heap Finder. There might be various heaps in the process and the location of all of them are passed on to the next component, Heap Extractor.

The Heap Extractor component receives the physical memory locations of the heaps from the first component of the Atomizer and extracts all the heap pages of the process. In a typical system some heap pages may be swapped out of physical memory for efficiency, so the Atomizer architecture also has the capability to access pages that are currently swapped out. When the Heap Extractor module encounters a swapped heap page, it uses the Swapped Heap Page Module to access those pages. The Swapped Heap Page Extractor component of the Atomizer uses information about the guest OS' swapping architecture to access swapped pages.

The Atomizer is designed to work in a cloud environment to monitor multiple VMs running in the environment using a single privileged system. The VMI interface provides access to the memory of all the VMs running on a single physical server. The Atomizer can browse through the heaps of all the processes running in these VMs with low performance impact, i.e., without using many of resources of the cloud

server. The Atomizer has a modular design which allows adding support for various operating systems. The Heap Analysis Module part is also designed in such a way that adding additional modules to enhance heap analysis is very straightforward. In particular, the Heap Extractor module provides an application programmable interface (API), allowing new heap analysis techniques to use the Heap Extractor API's to access heap memory. These API's can be used to detect any type of heap-based attacks.

4. IMPLEMENTATION

We developed a proof of concept prototype of Atomizer on XEN [9] that had Microsoft Windows XP (Service Pack 2) VMs running. We used the libVMI library [8] to introspect the heap of web browsers (Internet Explorer or Firefox) running on Windows XP virtual machines. We also used libguestfs [7] to access the page file of memory pages that have been swapped to the hard drive.

4.1 Process Information Extraction

Windows operating systems maintain information about executing processes in Process Environment Block (PEB) structures. In this part of the implementation, Atomizer looks for memory address of the PEB structure of the process. It goes through known memory addresses to locate the PEB structure using appropriate PEB signatures [21]. Once the location of the PEB structure is found the information regarding the process is extracted from it. That information includes the location of the heaps, number of heaps available and maximum number of heaps allowed by the OS. This information is then forwarded to the next component of the Atomizer.

Algorithm 1 Heap Memory Browsing using VAD tree

```
for i = 0x7FFD0000 to 0x7FFDF000 do
    if ((5 == i + 0xa4)&&(1 == i + 0xa8)) then
        PEB = i
        break;
    end if
end for
HEAPNUM := PEB + 0x88
HEAPADDRESS := PEB + 0x090
heapCounter := 0
while heapCounter < HEAPNUM do
    HEAPNODE := HEAPADDRESS +(4 * heapCounter)
    segmentCounter := 0
    while segmentCounter < 64 do
        HEAPSEGMENT := HEAPNODE + 0x58 + (4 * segmentCounter)
        HEAPENTRY := HEAPSEGMENT + 0x20
        while (HEAPENTRY + 0x005) ≠ 0 do
            HEAPSIZE := HEAPENTRY
            READ_MEMORY(HEAPENTRY, HEAPSIZE)
            HEAPENTRY := HEAPENTRY + (HEAPSIZE * 8)
        end while
        segmentCounter++
    end while
    heapCounter++
end while
```

4.2 Heap Extractor

The Heap Extractor component of Atomizer provides the main facility for browsing through the heap of the process using the virtual address descriptors (VAD) obtained via introspection. The memory manager in Windows maintains a set of VADs that describes the status of the process's address space [21]. To find the VAD tree of a process being monitored we use the process environment block structure (PEB) information received from Process Information Extractor. Using this information, we browse through all the heap nodes, heap segments, and heap entries in various heaps of a process. Algorithm 1 describes the algorithm used to browse through the entire heap. If a heap page entry is in the VAD tree and not in physical memory, than there is a possibility that the heap page has been swapped out of memory. As modern heap sprays do not require large amounts of NOP sleds, it is important that those heap pages are also examined. Details of how we access pages that are swapped out are provided in Section 4.3. Heap extractor can access the page memory both page by page and byte by byte. This facility has been provided to facilitate various algorithms that might perform better with either of these memory access methods. Our implementation uses the page by page access method to extract one page at a time from the heap and send it to the Heap Analysis Module (discussed in Section 4.4).

Algorithm 2 Simple NOP Sled Detection

```
NOPZ ← HASH-TABLE of NOPs/NOP-replacements
LIMIT ← 150
BUFFER ← Memory buffer from Heap size = SIZE
SKIP ← 1
index := 0
nops := 0
skipped := 0
while index < SIZE do
    if NOPZ[ BUFFER[index++] ] then
        nops++
    else if skipped < SKIP then
        skipped++
    else
        nops := 0
    end if
    if nops == LIMIT then
        NOP sled detected
    end if
end while
```

4.3 Swapped Heap Page Extractor

When the Atomizer needs to access a swapped out page, it follows a procedure similar to that of the guest OS. We describe this method as a two-step process where in the first step, the Atomizer retrieves the page file number and the page offset and in the second step, it uses this information to obtain the value of the virtual address from the page file. Our prototype is based on Windows XP (SP2), with Physical Address Extension (PAE) support enabled, for implementation. The PAE gives us four levels of virtual address translation where a (32-bit) virtual address (in PAE) contains the pointers to the page directory pointer table, page directory table, page table and page byte offset. Unlike the page directory pointer table, the page directory and page table both have 64 bit entries. PAE supports two page sizes i.e. 4K and 2M (also referred as large page). Depending on the page size (4K and 2M), the page file number and page offset are recorded into the page table or the page directory respectively. If the 7th bit of page directory entry is valid, it means that the entry is pointing to a large 2M page instead of a page table. In order to ensure that an entry in the page directory or page table contains the offset of a page in page file, its 0th (valid bit), 10th (prototype bit) and 11th (transition bit) bits must be zero. Moreover, the offset is present in the higher 32 bits (from 32 to 51 bits) of the 64-bit entry and the page number is present in the lower 32-bit of the entry. In the second step, Atomizer uses the libguestfs [7] library to access the page file of the guest VMs from Dom0 and reads the corresponding page using the page offset information. Atomizer then uses the page byte offset present in the virtual address to access the value of the virtual address.

4.4 Heap Analysis Module

The heap memory received from the heap extractor can be analyzed in the Heap Analysis Module. Various algorithms may be used here to determine whether the heap of the process contains malicious contents or not. Some techniques like STRIDE [10] and ECL-Polynop [16] use sequential analysis of network packets to detect NOP sleds in network traffic, rather than in the process heap itself. Using

the same model in our implementation, we implemented a simple Polymorphic NOP detection algorithm to detect the presence of NOP sleds in the heap using sequential analysis of the heap data. This implementation uses the Atomizer architecture to sequentially go through the heap memory and compares each byte of data with a hash table of NOPs and NOP replacements. The algorithm keeps tracks of sequences of NOPs/NOP replacements found and if the length of these sequences are beyond a certain limit it raises a flag. The sequence limit used in our experiments was 150 NOPs. This limit was selected because it gave no false positives in our experiments. This simple NOP sled detection algorithm is depicted in the Algorithm 2. This module demonstrates that our architecture provides a simple way of implementing these kinds of algorithms in the heap analysis module to detect NOP sleds. The presence of NOP sleds typically means that malicious content is present in the heap.

5. EVALUATION

5.1 Experimental Settings

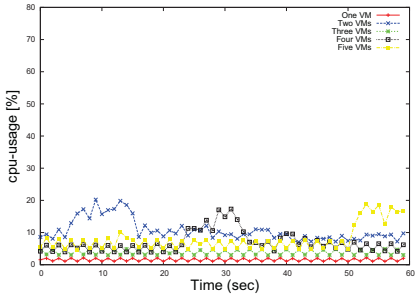
For this experiment, we built a simple cloud environment. This test bed featured a Quad Core i7 (2.67 GHz * 8) server with HyperThreading enabled and 18 GB of RAM. This server had a 64-bit privileged virtual machine (`Dom0`) running Fedora 16 (kernel 3.3.2-6) along with Xen 4.1.2 [9]. We instantiated five VM clones (`DomU: Dom1-Dom5`) in Xen from a single 32 bit Window XP (SP2) installation to make sure that all VMs are identical. We used the introspection library for VMI (libvmi-0.6) [8] and libguestfs [7] in all the experiments.

5.2 Malware Detection

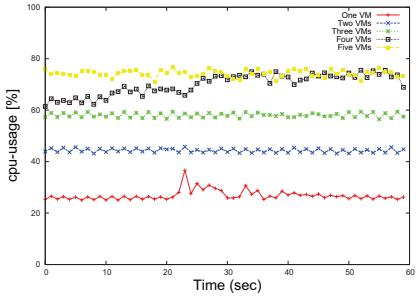
To test the effectiveness of Atomizer we tested it against various types of heap sprays commonly found on the web, as well as some custom made heap spray programs.

The first set of experiments involved the Skypher heap spray generator [5], an example of a simple heap spraying technique. Different variants of Skypher were tested on both Internet Explorer and Mozilla Firefox. Atomizer effectively detected the heap spray with no false positives. Atomizer was also tested against another known heap spray attack, Aurora [4]. This types of heap spray demonstrated some simple obfuscation techniques to hide the payload. The payload in Aurora was encrypted using the JavaScript libraries and decrypted at run time. The Atomizer detected the heap spray without any false positives.

To test the effectiveness of Atomizer against polymorphic NOP sled detection, in the second set of experiments, we created a small application in C that sprayed various polymorphic NOP sleds along with a dummy shell code in the heap. Most of the NOP sled obfuscation tools like ADMutate [1] are designed to evade Intrusion Detection Systems(IDS) by encrypting the packets which contain the heap spray code until it is executed. As we are monitoring the heap, the NOP sled cannot be encrypted in the heap making it the best place for detecting it. Polymorphic NOP sleds use NOPs and NOP replacements to simulate the behavior of random data in the heap. Our implementation uses a hash table of up to 118 known NOP replacements (which includes both one and two bytes NOP replacements) to detect the NOP sleds. This



(a) 1–5 Idle VMs running



(b) 1–5 VMs with light load & Atomizer

Figure 2: CPU Performance (CPU usage in *Dom*0)

hash table represents the most commonly used NOP replacements used by roots kits like ADMutate [1], which contains the biggest public list of NOP replacements [16]. NOP sleds that might include unknown NOP replacements may not be detected by our implementation but newly discovered NOP replacements may potentially be added seamlessly to our hash table. Our implementation successfully detected the polymorphic NOP sleds without any false positives.

The Atomizer was tested against a state of the art exploit, Heap Feng Shui [23]. Heap Feng Shui is a deterministic heap spray that reduces the amount of heap spraying required for the exploit. Heap Feng Shui uses the `HeapLib` (a JavaScript heap manipulation library) to defragment the heap so that it can align the heap nodes and consequently requires a smaller size of heap spray to execute the exploit. The Atomizer was able to detect this exploit which shows its effectiveness against state of the art exploits.

5.3 Experimental Performance Analysis

An experimental performance analysis of the Atomizer with respect to CPU resource utilization was done to determine the effect of heap spray detection on the cloud environment. The main purpose of these experiments was to determine the CPU resources used by Atomizer. For performance evaluation purposes in these experiments, the Atomizer was allowed to continue scanning the heap even after the heap spray block was found. For better performance the detector should be stopped when the first sign of malicious activity is raised.

In our experiments, the Atomizer was run on VMs with a simple workload (like a web browser running a YouTube video). This workload also represents the application being monitored by Atomizer during the experiment. The CPU usage was monitored while the systems ran. This was done to set a baseline that shows CPU usage in the normal usage of the VMs. Figure2(a) shows the CPU baseline for the VMs.

The Atomizer was run as a multi-process application to monitor all the virtual machines running on our server. The current multi-process implementation of Atomizer works around a lack of thread safety in the libVMI library, an issue that we are currently working to address. The Figure 2(b) shows the effect of Atomizer on the CPU. The Figure 2 shows the cumulative CPU usage of all the Virtual CPU's (VCPU) on the server (in our case we had eight VCPUs). This shows the overall effect of all the applications, including Atomizer and VMs on the CPU usage. Xentop [9] (the tool used by us to measure the CPU usage) adds up the percentage of all the VCPUs available to the system. The rise in the CPU usage is due to the processing needed by the light load running on the VMs plus the effect of Atomizer. The multi-process nature of the Atomizer also exerts more load on CPU. This limitation can be removed by implementing a multi-threaded version of Atomizer, a project that we have currently underway. The average CPU load increased incrementally, that shows that Atomizer has acceptable performance and is scalable.

6. CONCLUSION

In this paper we presented a novel and scalable method to analyze the heap memory of the processes running inside a set of VMs, via VM introspection. Atomizer can be easily extended by implementing new detection methods for any type of heap-based attacks. Expermental results show that Atomizer successfully detects various heap spray attacks and randomly generated polymorphic NOP sled samples with no false positives. Further work is required to improve the performance of our method, via a multi-threaded implementation of Atomizer.

Acknowledgment

This work was supported by the NSF grant, CNS #1016807.

7. REFERENCES

[1] Admutate. `http://www.pestpatrol.com/zks/pestinfo/a/admmutate.asp`.

[2] Advanced heap spraying technique. `http://www.blackhat.com/presentations/bh-europe-07/Sotirov/Presentation/bh-eu-07-sotirov-apr19.pdf`.

[3] Data execution prevention. `http://technet.microsoft.com/en-us/library/cc738483.aspx`.

[4] Heap spray exploit tutorial: Internet explorer use after free aurora vulnerability. `http://grey-corner.blogspot.com/2010/01/heap-spray-exploit-tutorial-internet.html`.

[5] Heap spray generator. `http://skypher.com/SkyLined/heap_spray/small_heap_spray_generator.html`.

[6] Internet explorer iframe src&name parameter bof remote compromise. `http://skypher.com/wiki/index.php/Www.edup.tudelft.nl/~bjwever/advisory_iframe.html.php`.

[7] Libguestfs. `http://libguestfs.org/`.

[8] Libvmi. `http://code.google.com/p/vmitools/`.

[9] XEN. `http://www.xen.org/`.

[10] P. Akritidis, E. P. Markatos, M. Polychronakis, and K. G. Anagnostakis. Stride: Polymorphic sled detection through instruction sequence analysis. In *20th IFIP International Information Security Conference (IFIP/SEC)*, Makuhari-Messe, Chiba, Japan, May 2005.

[11] D. Blazakis. Interpreter exploitation: Pointer inference and JIT spraying. In *BLACK HAT DC*, Arlington, VA, US, January 2010.

[12] M. Cova, C. Kruegel, and G. Vigna. Detection and analysis of drive-by-download attacks and malicious javascript code. In *International World Wide Web Conference*, Raleigh, North Carolina, US, April 2010.

[13] C. Cowan, P. Wagle, C. Pu, S. Beattie, and J. Walpole. Buffer overflows:attacks and defenses for the vulnerability of the decade. *Foundations of Intrusion Tolerant Systems*, pages 227–237, 2003.

[14] C. Curtsinger, B. Livshits, B. Zorn, and C. Seifert. ZOZZLE: Low-overhead mostly static javascript malware detection. In *USENIX Security Symposium*, San Francisco, California, US, August 2011.

[15] M. Daniel, J. Honoroff, and C. Miller. Engineering heap overflow exploitswith javascript. In *WOOT: Proceedings of the 2nd conference on USENIX Workshop on offensive technologies*, San Diego, California, US, 2008.

[16] Y. Gushin. http://www.ecl-labs.org/papers/ecl-poly.txt.

[17] F. Howard. Malware with your mocha? obfuscation and anti-emulation tricks in malicious javascript, September 2010. `http://www.sophos.com/security/technical-papers/malware_with_your_mocha.pdf`.

[18] A. LeMaster. Heap spray detection with heap inspector. In *Blackhat USA*, Las Vegas, Nevada, US, 2011.

[19] P. Ratanaworabhan, B. Livshits, and B. Zorn. NOZZLE: A defense against heap-spraying code injection attacks. In *USENIX Security Symposium*, San Francisco, California, US, August 2009.

[20] R. Roemer, E. Buchanan, H. Shacham, and S. Savage. Return-oriented programming: Systems, languages, and applications. *ACM Trans. Inf. Syst. Secur.*, 15(1):2:1–2:34, Mar. 2012.

[21] M. E. Russinovich, D. A. Solomon, and A. Ionescu. *Microsoft Windows internals*. Microsoft Press, fifth edition, 2009.

[22] H. Shacham, M. Page, B. Pfaff, E. Goh, N. Modadugu, and D. Boneh. On the effectiveness of address-space randomization. In *ACM Conference on Computer and Communications Security*, Washington, DC, US, October 2004.

[23] A. Sotirov. Heap fang shui in javascript. In *BLACK HAT Europe*, Amsterdam , Netherlands, March 2007.