Journal of Computer Security 13 (2005) 699–720
IOS Press

A soft constraint-based approach to the cascade vulnerability problem

Stefano Bistarelli [a,b], Simon N. Foley [c] and Barry O'Sullivan [d]
[a] *Istituto di Informatica e Telematica, CNR, Pisa, Italy*
E-mail: stefano.bistarelli@iit.cnr.it
[b] *Dipartimento di Scienze, Universitá degli Studi "G. D'annunzio", Pescara, Italy*
E-mail: bista@sci.unich.it
[c] *Department of Computer Science, University College Cork, Ireland*
E-mail: s.foley@cs.ucc.ie
[d] *Cork Constraint Computation Centre, Department of Computer Science, University College Cork, Ireland*
E-mail: b.osullivan@cs.ucc.ie

The security of a network configuration is based not just on the security of its individual components and their direct interconnections, but also on the potential for systems to interoperate indirectly across network routes. Such interoperation has been shown to provide the potential for cascading paths that violate security, in a circuitous manner, across a network. In this paper we show how constraint satisfaction provides a natural approach to expressing the necessary constraints to ensure multilevel security across a network configuration. In particular, soft constraints are used to detect and eliminate the cascading network paths that compromise security. Taking this approach results in practical advancements over existing solutions to this problem. In particular, constraint satisfaction highlights the set of all cascading paths, which we can eliminate in polynomial time by breaking a minimal number of system links to ensure security.

1. Introduction

The composition of secure systems is not necessarily secure. A user may be able to gain unauthorised access to an object by taking a circuitous access route across individually secure but interoperating systems [16]. Determining security is based not just on the individual system authorisation mechanisms but also on how the systems are configured to interoperate. For example, if Alice is permitted to have access to Bob's files on the administration system, and Clare is permitted access Alice's files on the sales system, then is it safe to support file sharing between these systems? The extent of system interoperation must be limited if the administration security policy states that Clare is not permitted access to Bob's (administration) files.

The *cascade vulnerability problem* [29] is concerned with secure interoperation, and considers the *assurance risk* of composing multilevel secure systems that are evaluated to different levels of assurance according to the criteria specified in [29].

The transitivity of the multilevel security policy upheld across all secure systems ensures that their multilevel composition is secure; however, interoperability and data sharing between systems may increase the risk of compromise beyond that accepted by the assurance level. For example, it may be an acceptable risk to store only secret and top-secret data on a medium assurance system, and only classified and secret data on another medium assurance system; classified and top-secret data may be stored simultaneously only on "high" assurance systems. However, if these medium assurance systems interoperate at classification secret, then the acceptable risk of compromise is no longer adequate as there is an unacceptable cascading risk from top-secret across the network to classified.

Existing research has considered schemes for *detecting* these security vulnerabilities and for *eliminating* them by reconfiguring system interoperation. While the detection of the cascade vulnerability [14–16,18,23] can be easily achieved, their optimal elimination is NP-complete [16–18].

We present an approach to using constraints [13,21] for reasoning about secure interoperation. Constraint solving is an emerging software technology for modelling and solving large-scale optimisation problems [30]. Constraints have been successfully applied to a number of problems in computer security [2,3,6,7,24]. However, the cascade vulnerability problem, and secure interoperation in general, have not been studied.

The approach that we present in this paper represents a paradigm shift in the modelling, detection and elimination of the cascade vulnerability problem. We present a constraint model that provides a natural description of an arbitrary multilevel secure network. Any solution to the model represents a cascading path through the network, providing significantly more information on its vulnerabilities than the existing approaches, and providing a basis for eliminating the cascade vulnerability problem. Previous approaches [14,18] detect a single cascading path in polynomial time, but correcting the cascade in an optimal way is NP-complete. Using a constraint model, we can rely on a significant body of successful techniques from the field of constraint processing for finding the set of cascading paths which, once found, can be eliminated in polynomial time. These results are applicable to secure interoperation in general.

This paper combines and extends work originally published in [8,9]. The paper is organised as follows. Section 2 provides background on the cascade vulnerability problem and on soft constraints. Section 3 describes a soft constraint-based model for describing multilevel secure networks. Solutions to this constraint model represent the possible information flow paths through the network and Section 4 characterises the cascade vulnerability problem in terms of these solutions. An example is considered in Section 5 and Section 6 proposes a polynomial-time scheme for eliminating cascading paths from the network.

2. Background

2.1. The cascade vulnerability problem

Figure 1 gives an example of a multilevel security (MLS) network configuration with a cascade vulnerability problem [14].

The network is comprised of multilevel secure systems Sys.E, Sys.F, Sys.G and Sys.H storing classified (C), secret (S) and top-secret (T) information as depicted in Fig. 1. Each system is accredited according to levels of assurance C2<B1<B2<B3 from [28,29]. For example, Sys.F is used to simultaneously store classified, secret and top-secret information and, therefore, (according to [28,29]) must be evaluated at level B3 or higher, reflecting the high level of confidence that must be placed in the secure operation of the system. This is to counter the risk of an attacker compromising the system and copying top-secret information to classified. Sys.H, on the other hand, has been evaluated at the lowest level of assurance C2 and, therefore, may be used only to store single level data.

However, the security-level interoperability defined by the system connections in Fig. 1 results in a cascade vulnerability across the network. There is a risk that an attacker who has the ability to compromise security on B2 or lower assured systems can copy T to S on Sys.E, to S on Sys.H to S to C on Sys.G. This is contrary to the requirement that the level of assurance that T cannot be copied to C should be B3 or higher. This requirement is met by the individual systems but not as a result of their interoperation. A generalised form of the cascade vulnerability problem is defined as follows.

2.1.1. MLS

A multilevel secure system enforces a lattice-based security policy $\mathcal{L}$ of security levels that has ordering relation $\leqslant$. Given $x, y : \mathcal{L}$ then $x \leqslant y$ means that information may flow from level x to level y, for example, $\mathsf{C} \leqslant \mathsf{S} \leqslant \mathsf{T}$.

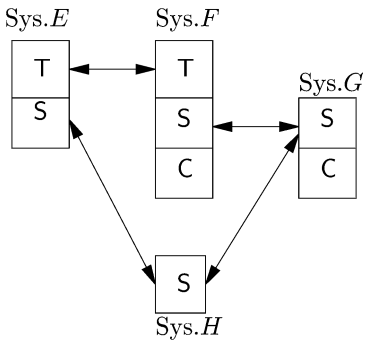

Fig. 1. Network configuration with a cascade vulnerability problem.

2.1.2. Assurance levels

Security criteria define a lattice, $\mathcal{A}$, of assurance levels with ordering $\leqslant$. Given $x, y : \mathcal{A}$, then $x \leqslant y$ means that a system evaluated at y is no less secure than a system evaluated at x, or alternatively, that an attacker that can compromise a system evaluated at y can compromise a system evaluated at x. Let $\mathcal{S}$ define the set of all possible systems. We define $accred : \mathcal{S} \rightarrow \mathcal{A}$ where $accred(s)$ gives the assurance level of system $s : \mathcal{S}$, and is taken to represent the minimum effort required by an attacker to compromise system s.

2.1.3. Acceptable risk

Security evaluation criteria also define an acceptable risk function $risk : \mathcal{L} \times \mathcal{L} \rightarrow \mathcal{A}$, such that given $l, l' : \mathcal{L}$ then $risk(l, l')$ defines the minimum acceptable risk of compromise from l to l'; it represents the minimum acceptable effort required to "compromise security" and copy/downgrade information from level l to level l'. Without any loss of generality we assume that there is no security enforcement at the lowest assurance level $\mathbf{0}$, and thus, if $l \leqslant l'$ then $risk(l, l') = \mathbf{0}$. For example, the function $risk$ encodes the assurance matrix for the "B" levels (from [28,29]) 1, 2 and 3, with: 0 representing no security enforcement, as $risk(\mathsf{C}, \mathsf{S}) = risk(\mathsf{C}, \mathsf{T}) = risk(\mathsf{S}, \mathsf{T}) = 0$, $risk(\mathsf{S}, \mathsf{C}) = 1$, $risk(\mathsf{T}, \mathsf{S}) = 2$, and $risk(\mathsf{T}, \mathsf{C}) = 3$, and so forth.

2.1.4. Evaluated systems

Individual systems must be assured higher than the minimum acceptable risk to compromise the data they store. If a system s can hold information at levels l and l' then $risk(l, l') \leqslant accred(s)$.

2.1.5. Network model

A system node in our network model is a pair, l_s, and represents the fact that system s can hold information at level l. Thus, a system is a collection of nodes that represent the data it holds. For example, in Fig. 1, Sys.E can store secret and top-secret information and is represented by nodes S_E and T_E. A network of systems is a weighted graph of these nodes according to how they are connected. An w-weighted arc from l_s to $l'_{s'}$ means that it requires at minimum w effort to directly copy information at level l held on system s to level l' on system s'.

2.1.6. Cascading risks

Arcs are used to represent direct flows within a system and interoperation links between systems. A flow $l \leqslant l'$ that is permitted on system s is represented as a (assurance) $\mathbf{0}$-weighted arc from l_s to l'_s; if a flow is not permitted between levels l and l' that are held on system s then it is represented as an arc weighted as $accred(s)$ from l_s to l'_s.

A link from system s to s' that connects l-level information is represented as a $\mathbf{0}$-weighted arc from l_s to $l_{s'}$—all other pairs l_s to $l_{s'}$ not related in this way are either represented as having no arc, or an arc with the maximum assurance value $\mathbf{1}$.

Given pairs l_s and $l'_{s'}$ we then define *effort*$(l_s, l'_{s'})$ as the minimum effort required to compromise the network and copy and/or downgrade level l information held on

system s to level l' information on system s'. As an example, in Fig. 1, the effort to "copy" top-secret information on system Sys.E to classified on system Sys.G is *effort*$(\mathsf{T}_E, \mathsf{C}_G) = B2$ via the path through Sys.H.

2.1.7. Cascade freedom

We require that for any pair of systems s and s' and levels l and l', then we have $risk(l, l') \leqslant$ *effort*$(l_s, l'_{s'})$. Given a path in the network from l_s to $l'_{s'}$, then its cascade weight is the maximum weight that directly connects any two nodes on the path. This reflects the minimum effort that will be required by an attacker to copy information from l_s to $l'_{s'}$ by using this path. *effort*$(l_s, l'_{s'})$ is the minimum of the cascade weights for all paths that connect l_s to $l'_{s'}$.

2.2. Soft constraints

Several formalisations of the concept of *soft constraints* are currently available. In the following, we refer to the one based on c-semirings [4,11], which can be shown to generalise and express many of the others [5].

A soft constraint may be seen as a constraint where each instantiation of its variables has an associated value from a partially ordered set that can be interpreted as a set of preference values. Combining constraints will then have to take into account such additional values, and thus the formalism has also to provide suitable operations for combination ($\times$) and comparison ($+$) of tuples of values and constraints. This is why this formalisation is based on the concept of c-semiring, which is just a set plus two operations.

2.2.1. Semirings

A semiring is a tuple $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: (1) $\mathcal{A}$ is a set and $\mathbf{0}, \mathbf{1} \in \mathcal{A}$; (2) $+$ is commutative, associative and $\mathbf{0}$ is its unit element; (3) $\times$ is associative, distributes over $+$, $\mathbf{1}$ is its unit element and $\mathbf{0}$ is its absorbing element. A c-semiring is a semiring $\langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ such that: $+$ is idempotent, $\mathbf{1}$ is its absorbing element and $\times$ is commutative. Let us consider the relation $\leqslant_S$ over $\mathcal{A}$ such that $a \leqslant_S b$ iff $a + b = b$. Then it is possible to prove that (see [11]): (1) $\leqslant_S$ is a partial order; (2) $+$ and $\times$ are monotone on $\leqslant_S$; (3) $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; (4) $\langle \mathcal{A}, \leqslant_S \rangle$ is a lattice and, for all $a, b \in \mathcal{A}$, $a + b = lub(a, b)$ (where lub is the *least upper bound*). Moreover, if $\times$ is idempotent, then: $+$ distributes over $\times$; $\langle \mathcal{A}, \leqslant_S \rangle$ is a distributive lattice and $\times$ its glb (*greatest lower bound*). Informally, the relation $\leqslant_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leqslant_S b$, we will say that *b is better than a*. In the following, when the semiring will be clear from the context, $a \leqslant_S b$ will be often indicated by $a \leqslant b$.

2.2.2. Constraint problems

Given a semiring $S = \langle \mathcal{A}, +, \times, \mathbf{0}, \mathbf{1} \rangle$ and an ordered set of variables V over a finite domain D, a *constraint* is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. By using this notation we define

$\mathcal{C} : \eta \to \mathcal{A}$ as the set of all possible constraints that can be built starting from S, D and V.

Note that in this *functional* formulation, each constraint is a function. Such a function involves all the variables in V, but it depends on the assignment of only a finite subset of them. So, for instance, a binary constraint $c_{x,y}$ over variables x and y, is a function $c_{x,y} : V \to D \to \mathcal{A}$, but it depends only on the assignment of variables $\{x, y\} \subseteq V$. We call this subset the *support* of the constraint. More formally, consider a constraint $c \in \mathcal{C}$. We define its support as $supp(c) = \{v \in V \mid \exists \eta, d_1, d_2 . c\eta[v := d_1] \neq c\eta[v := d_2]\}$, where

$$\eta[v := d]v' = \begin{cases} d & \text{if } v = v', \\ \eta v' & \text{otherwise.} \end{cases}$$

Note that $c\eta[v := d_1]$ means $c\eta'$ where η' is η modified with the assignment $v := d_1$ (that is the operator [ ] has precedence over application). Note also that $c\eta$ is the application of a constraint function $c : V \to D \to \mathcal{A}$ to a function $\eta : V \to D$; what we obtain, is a semiring value $c\eta = a$.

A *soft constraint satisfaction problem* (SCSP) is a pair $\langle C, con \rangle$ where $con \subseteq V$ and C is a set of constraints: con is the set of variables of interest for the constraint set C, which however may concern also variables not in con. Note that a classical CSP is a SCSP where the chosen c-semiring is: $S_{CSP} = \langle \{false, true\}, \vee, \wedge, false, true \rangle$. Fuzzy CSPs [26] (FCSP) can instead be modelled in the SCSP framework by choosing the c-semiring $S_{FCSP} = \langle [0, 1], max, min, 0, 1 \rangle$. Many other "soft" CSPs (Probabilistic, weighted, ...) can be modelled by using a suitable semiring structure ($S_{prob} = \langle [0, 1], max, \times, 0, 1 \rangle$, $S_{weight} = \langle \mathcal{R}, min, +, +\infty, 0 \rangle$, ...).

Figure 2 shows the graph representation of a fuzzy CSP. Variables and constraints are represented, respectively, by nodes and by undirected (unary for c_1 and c_3 and binary for c_2) arcs, and semiring values are written to the right of the corresponding tuples. The variables of interest (that is the set con) are represented with a double circle. Here we assume that the domain D of the variables contains only elements a and b and c.

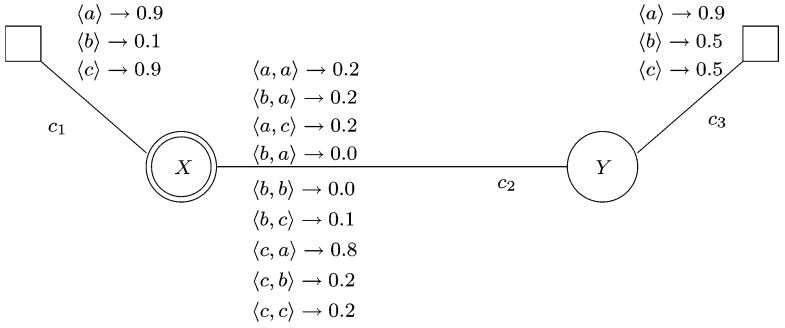

Fig. 2. A fuzzy CSP.

2.2.3. Combining and projecting soft constraints

Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ is defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$. In words, combining two constraints means building a new constraint whose support involves all the variables of the original ones, and which associates with each tuple of domain values for such variables a semiring element which is obtained by multiplying the elements associated by the original constraints to the appropriate sub-tuples. It is easy to verify that $supp(c_1 \otimes c_2) \subseteq supp(c_1) \cup supp(c_2)$.

Given a constraint $c \in \mathcal{C}$ and a variable $v \in V$, the *projection* of c over $V - \{v\}$, written $c \Downarrow_{(V-\{v\})}$ is the constraint c' s.t. $c'\eta = \sum_{d \in D} c\eta[v := d]$. Informally, projecting means eliminating some variables from the support. This is done by associating with each tuple over the remaining variables a semiring element which is the sum of the elements associated by the original constraint to all the extensions of this tuple over the eliminated variables. In short, combination is performed via the multiplicative operation of the semiring, and projection via the additive one.

2.2.4. Solutions

A *solution* of an SCSP $P = \langle C, con \rangle$ is the constraint $Sol(P) = (\bigotimes C) \Downarrow_{con}$. That is, we combine all constraints, and then project over the variables in con. In this way we get the constraint with support (not greater than) con which is "induced" by the entire SCSP. Note that when all the variables are of interest we do not need to perform any projection.

For example, the solution of the fuzzy CSP of Fig. 2 associates a semiring element with every domain value of variable x. Such an element is obtained by first combining all the constraints together. For instance, for the tuple $\langle a, a \rangle$ (that is, $x = y = a$), we have to compute the minimum between 0.9 (which is the value assigned to $x = a$ in constraint c_1), 0.8 (which is the value assigned to $\langle x = a, y = a \rangle$ in c_2) and 0.9 (which is the value for $y = a$ in c_3). Hence, the resulting value for this tuple is 0.8. We can do the same work for tuple $\langle a, b \rangle \rightarrow 0.2$, $\langle a, c \rangle \rightarrow 0.2$, $\langle b, a \rangle \rightarrow 0$, $\langle b, b \rangle \rightarrow 0$, $\langle b, c \rangle \rightarrow 0.1$, $\langle c, a \rangle \rightarrow 0.8$, $\langle c, b \rangle \rightarrow 0.2$ and $\langle c, c \rangle \rightarrow 0.2$. The obtained tuples are then projected over variable x, obtaining the solution $\langle a \rangle \rightarrow 0.8$, $\langle b \rangle \rightarrow 0.1$ and $\langle c \rangle \rightarrow 0.8$.

3. Modelling MLS networks

Consider a network $N = \{E, F, G, H, \ldots\}$ of a finite arbitrary number n of systems. This network is represented in our constraint model in terms of all possible paths (of length n and less) that connect the systems. The paths are modelled using $2 \times n$ *path* variables, where each path variable P_i^s and P_i^d can be instantiated to be one system of the network. A path through the network is represented by a specific instantiation of the variables $[P_1^s, P_1^d, P_2^s, P_2^d, \ldots, P_n^s, P_n^d]$. In particular, the instantiation of the pair of nodes P_i^s and P_i^d, for $i := 1 \ldots n$, represents the flow from the source P_i^s to the destination P_i^d within the system at the i-th position of the path.

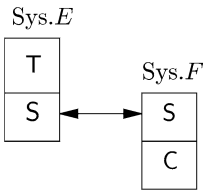

Fig. 3. A simple network.

Similarly, instantiation of P_i^d and P_{i+1}^s, for $i := 1 \ldots n-1$ represents the flow among the i-th and the $i+1$-th system in the specific instantiated path.

Consider for instance the network $N = \{E, F\}$ represented in Fig. 3 involving two systems, Sys.E and Sys.F, with Sys.E handling information at level top-secret (T) and secret (S), and Sys.F handling information at level secret (S) and confidential (C). We can capture this instance by using 4 path variables: $[P_1^s, P_1^d, P_2^s, P_2^d]$.

3.1. Path variable domains

The domain of each path variable defines the set of possible security levels available on each system. In particular, each *source* variable P_i^s contains domain elements marked with s, and each *destination* variable P_i^d contains domain elements marked with d.

The network in Fig. 3 has in our model 4 variables $[P_1^s, P_1^d, P_2^s, P_2^d]$ with $\text{dom}(P_i^s) = \{\mathsf{T}_E^s, \mathsf{S}_E^s, \mathsf{S}_F^s, \mathsf{C}_F^s\}$, with $i := 1, 2$, and $\text{dom}(P_i^d) = \{\mathsf{T}_E^d, \mathsf{S}_E^d, \mathsf{S}_F^d, \mathsf{C}_F^d\}$, for $i := 1, 2$. The strategy in our network model is that a network configuration will be represented as a series of constraints between the source and destination variables, representing all possible paths across the network.

In general, when the network contains $n > 2$ systems, we also need to be able to deal with shorter paths of length $k < n$. To do this, we need to extend the domain of each path variable, $P_i^?$ (where ? stands alternatively for source s and destination d), for any $i > 2$, with some artificial elements. More precisely, we extended the domain $\text{dom}(P_i^?)' = \text{dom}(P_i^?) \cup \{*_1^?, *_2^?, \ldots, *_{i-2}^?\}$. These $*$ elements are added to deal with paths shorter than n. This is necessary because solving an SCSP requires finding an assignment for each variable in the SCSP and we may want to represent paths shorter than the number of nodes in the network.

3.2. Modelling each system

The flows that are possible within a particular system occuring in position i of a network path are modelled as a constraint between the source and destination variables P_i^s and P_i^d. These flows reflect the accesses that are permitted by the system's MLS security mechanism. For example, secret information is permitted to flow to top-secret in Sys.E and, thus, there is a constraint $[P_1^s := \mathsf{S}_E^s; P_1^d := \mathsf{T}_E^d]$, which evaluates to 0, meaning unrestricted information flow. The flows that are possible within a system can be categorised in three ways.

- *Flow*$_{permitted}$ represents the information flows that are permitted by the policy in each node. For example, S may flow to T in Sys.E.
- *Flow*$_{risk}$ represents the information flows that are not permitted by the policy, but for which there is a risk of flow if the system became compromised. For example, the risk of a flow from T to S in system Sys.E is B2 (assurance level 2), corresponding to the level of assurance at which Sys.E has been evaluated.
- *Flow*$_{invalid}$: represents all the remaining flows that are not valid (that is, are impossible for the given system). For example, a flow from T to C is not possible on Sys.E since the system has not been configured to store information labelled as classified.

Consider an arbitrary system S_i that can occur on position i of a path through the network. Between each pair of variables P_i^s and P_i^d for each system S_i, we define a soft constraint, $c_{(P_i^s,P_i^d)}$, that gives a weight to each possible (permitted or risk) flow within that system. Various semirings could be used to represent the network and the associated policy. We use the following semiring in this paper, although our results are general and are not limited to this particular one:

$$S_{cascade} = \langle \mathbb{N}, min, max, +\infty, 0 \rangle.$$

Given this semiring, the constraint $c_{(P_i^s,P_i^d)}$ representing the flow inside a system S that occurs on position i of the path is defined as follows:

$$c_{(P_i^s,P_i^d)}(s,d) = \begin{cases} \mathit{accred}(S_i) & (s,d) \in \mathit{Flow}_{risk} \\ & \quad \text{(risk flows);} \\ 0 & (s,d) \in \mathit{Flow}_{permitted} \\ & \quad \text{(permitted flows);} \\ +\infty & \text{otherwise} \\ & \quad \text{(invalid flows).} \end{cases}$$

Since the domain of the variables $P_i^?$ (where ? stands for s and d) has been extended with the elements $\{*_1^?, *_2^?, \ldots, *_{i-2}^?\}$, we have also to consider these "artificial" values. In particular, we extend the definition of each constraint $c_{(P_i^s,P_i^d)}$ as follows:

$$c_{(P_i^s,P_i^d)}(s,d) = \begin{cases} 0 & (s,d) \in \{(*_1^s, *_1^d), \ldots, (*_{i-2}^s, *_{i-2}^d)\} \\ & \quad \text{(Artificial permitted flows);} \\ +\infty & \text{otherwise} \\ & \quad \text{(Artificial invalid flows).} \end{cases}$$

The constraints between P_1^s and P_1^d and between P_2^s and P_2^d in Fig. 4 depicts some of the system flow constraints that model the network configuration in Fig. 3. The use of the '$*$' elements and the representation of connections between systems within the network is considered in the next section.

Fig. 4. Flow constraints within Sys.E and Sys.F.

3.3. Modelling the network

Security level interconnections between systems result in two classes of flows between systems in a network.

- *Network*$_{permitted}$ represents information flows permitted by the connection policy between each system and represents direct synchronisation flows between systems. For example, the secret-level connection from Sys.E to Sys.F in Fig. 3 corresponds to a network permitted flow from S_E to S_F.
- *Network*$_{invalid}$ represents the absence of direct connection between the systems in the network configuration.

For each adjacent pair of systems at positions i and $i+1$ along a network path, we define a soft constraint, $c_{(P_i^d, P_{i+1}^s)}$, that defines the possible synchronisations between the systems. Note that these constraints are defined between the destination variable of the first system, P_i^d, and the source variable of the second system, P_{i+1}^s. The constraint $c_{(P_i^d, P_{i+1}^s)}$ representing the synchronisation flows between system at positions i and $i+1$ is defined as follows:

$$c_{(P_i^d, P_{i+1}^s)}(d,s) = \begin{cases} 0 & (d,s) \in \textit{Network}_{permitted} \\ & \text{(Policy permitted synchronisation);} \\ +\infty & \text{otherwise} \\ & \text{(invalid synchronisation).} \end{cases}$$

For example, the network configuration *Network*$_{permitted}$ for Fig. 3 is defined as follows:

$$\textit{Network}_{permitted} = \{(S_E^d, S_F^s), (S_F^d, S_E^s)\}$$

Note that the connection model does not consider assurance risks for connections; this can be achieved, if desired, by explicitly modelling the connections by their components (for example, a link encryption device) and corresponding assurance levels.

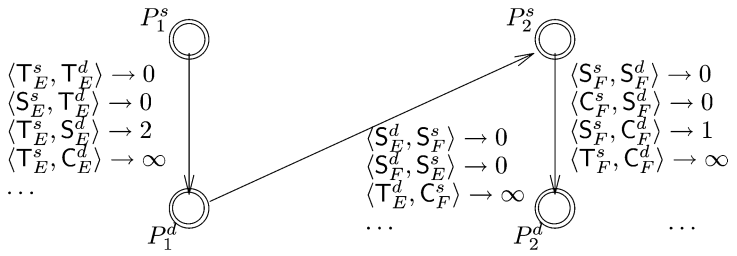

Fig. 5. Flow constraints on the connection from Sys.E to Sys.F.

When connecting systems at P_i^d and P_{i+1}^s it is also necessary to consider the constraints imposed by the artificial elements $*_i^?$. The definition of each constraint $c_{(P_i^d, P_{i+1}^s)}$ is extended as follows:

$$c_{(P_i^d, P_{i+1}^s)}(d,s) = \begin{cases} 0 & (d,s) \in \{(*_1^d, *_2^s), \ldots, (*_{i-3}^d, *_{i-2}^s)\} \\ & \quad \cup \{(\sharp, *_1^s) \text{ for all } \sharp \in D(P_i^d)\} \\ & \text{(Artificial permitted synchronisation);} \\ +\infty & \text{otherwise} \\ & \text{(Artificial invalid synchronisation).} \end{cases}$$

The extension of this constraint is slightly different to the previous system-level constraints. In particular, it enables us to model the connection between the last real domain element in the path and the first $*_1^s$-element.

Figure 5 depicts some of the system and network flow constraints that model the network configuration in Fig. 3. This constraint network represents a number of paths of length 4 (corresponding to $2 * n$, where n is the number of systems in the configuration). The path represented by the constraint solution $\eta = [P_1^s := \mathsf{T}_E^s; P_1^d := \mathsf{S}_E^d; P_2^s := \mathsf{S}_F^s; P_2^d := \mathsf{C}_F^d]$ has assurance/risk values: 2 (top-secret to secret compromise on Sys.E); 0 (secret level system connection); and 1 (secret to classified compromise on Sys.F), on the relating arcs, respectively.

For a given configuration, every path through the network must be modelled in this way. Figure 5 represents just one path and its subpaths, starting from Sys.E. Further constraints must be added to the model of the configuration to depict the other paths. For example, paths that start from Sys.F. Figure 6 depicts some of the constraints that represents all paths for the configuration. In more complex network configurations the $*$ elements allow the modelling of alternate paths between systems. For example, in the case of Fig. 1, paths (of length four) include $[E; F; G; H]$, $[E; H; G; *]$, and so forth.

In addition to ensuring that systems are configured in a valid way, we also need to ensure that no two pairs of path variables represent the same system. This ensures that our model does not capture cyclic paths. Therefore, we need to post an *alldifferent* [25] constraint amongst all the variables in the model. An *alldifferent* constraint ensures that all variables over which it is defined take on different values.

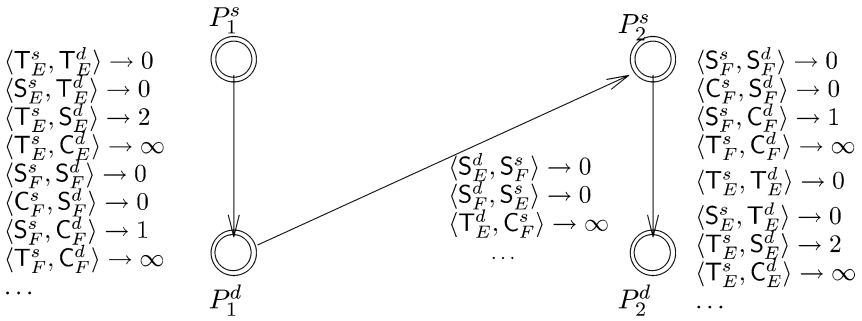

Fig. 6. Modelling multiple paths.

The solutions of the defined SCSP (referred to as the Effort-CSP, $\mathcal{E}$), that is all the solutions with a weight lower than $+\infty$, represents all of the possible paths through the system. The semiring level associated with each path (solution) gives a measure of the effort required to compromise the network using that specific path.

4. Detecting cascade vulnerabilities

To determine whether or not there exists a cascade vulnerability problem, we need to compare the effort required to compromise the network against the risk of compromising the system as a whole. Therefore, we introduce a set of *risk* constraints, $\mathcal{R} = \{r_{(P_1^s, P_i^d)} | i \in \{2, \ldots, n\}\}$. The weight of each instance of $r_{(P_1^s, P_i^d)}$ represents the risk associated with the path from P_1^s to P_i^d. The cost of each tuple in these constraints is defined as follows:

$$r_{(S_1^s, S_i^d)}(s, d) = \begin{cases} \mathbf{0} & \text{if } d = *_i^d; \\ risk(s, d) & \text{otherwise.} \end{cases}$$

The set of solutions of the SCSP $\mathcal{E}$ (that is the Effort-CSP defined above), each of whose associated semiring level is lower than $+\infty$, represents the set of paths through the network. The semiring level associated with each solution-path of $\mathcal{E}$ represents the minimum *effort* required to compromise the network, while the combination of the constraints in $\mathcal{R}$ (the Risk-CSP) gives the risk for all the paths. Therefore, a cascading path can be identified as any path η where the risk associated with the path exceeds the effort to compromise it, that is, where the following constraint is satisfied:

$$\bigotimes \mathcal{R}\eta > \bigotimes \mathcal{E}\eta$$

Therefore, by adding the above constraint to our constraint model, the existence of a solution to that model indicates that here exists a cascading path. Furthermore, the

set of solutions provides the set of cascading paths. This provides us with a basis upon which we can set about removing the cascade vulnerability problem from the network by eliminating all solutions of the model.

5. An example

In this section we encode the network example described in Fig. 1 within the proposed constraints model. Figure 7 depicts the structure of the constraint relationships in this model. We first present an example of how our model identifies a cascade-free path, and then present an example of detecting a cascading path.

For the purposes of the examples, the risk lattice is assumed to be as follows: $risk(\mathsf{C},\mathsf{S}) = risk(\mathsf{C},\mathsf{T}) = risk(\mathsf{S},\mathsf{T}) = 0$, $risk(\mathsf{S},\mathsf{C}) = 1$, $risk(\mathsf{T},\mathsf{S}) = 2$, $risk(\mathsf{T},\mathsf{C}) = 3$.

Figure 7 presents the structure of the constraint model for an example from [14]. Our model comprises 8 path variables, $P_1^s, P_1^d, P_2^s, P_2^d, P_3^s, P_3^d, P_4^s$, and P_4^d, and 3 risk variables, $r_{(P_1^s,P_2^d)}, r_{(P_1^s,P_3^d)}$ and $r_{(P_1^s,P_4^d)}$. The domain of each path variable, $D(P_i^?)$, is: $\{\mathsf{T}_E^?, \mathsf{S}_E^?, \mathsf{T}_F^?, \mathsf{S}_F^?, \mathsf{C}_F^?, \mathsf{S}_G^?, \mathsf{C}_G^?, \mathsf{S}_H^?\}$ (where ? stands alternatively for s and d) and $i := 1,\ldots,4$. Note that we also extend each domain using $*_i^?$ values as described above, but do not show this here for conciseness.

5.1. A cascade-free path

Consider the following path through the network:

$$\eta = [P_1^s := \mathsf{T}_E^s, P_1^d := \mathsf{T}_E^d, P_2^s := \mathsf{T}_F^s, P_2^d := \mathsf{S}_F^d,$$
$$P_3^s := \mathsf{S}_G^s, P_3^d := \mathsf{C}_G^d, P_4^s := *_1^s, P_4^d := *_1^d].$$

This scenario is illustrated in Fig. 8.

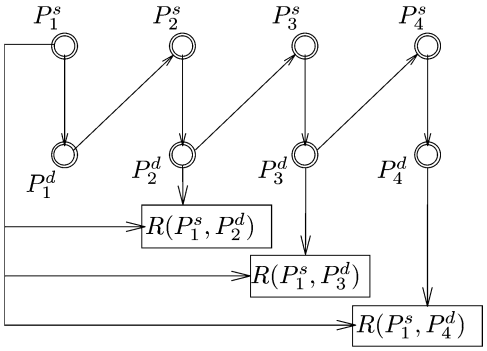

Fig. 7. The constraint model structure.

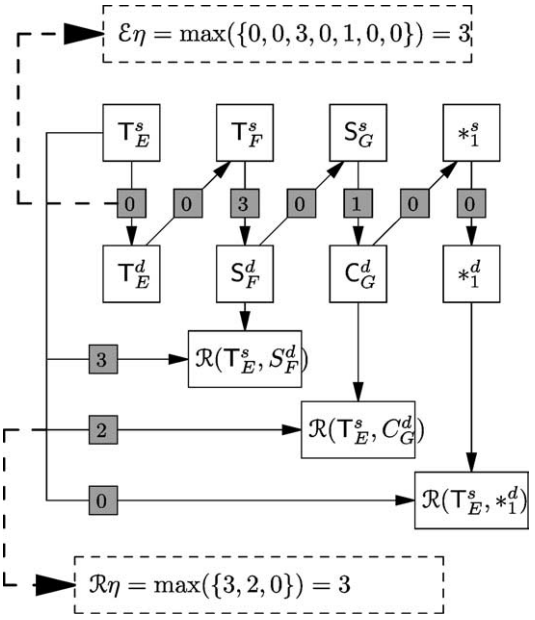

Fig. 8. A cascade-free path.

Evaluating the cascade detection constraint we get the following, proving that this path is cascade-free:

$$\bigotimes \mathcal{R}\eta > \bigotimes \mathcal{E}\eta \equiv 3 > 3 \equiv False.$$

5.2. A cascading path

Consider the following path through the network, depicted in Fig. 9:

$$\eta = [P^s_1 := \mathsf{T}^s_E, P^d_1 := \mathsf{S}^d_E, P^s_2 := \mathsf{S}^s_H, P^d_2 := \mathsf{S}^d_H,$$
$$P^s_3 := \mathsf{S}^s_G, P^d_3 := \mathsf{C}^d_G, P^s_4 := *^s_1, P^d_4 := *^d_1]$$

Evaluating the cascade detection constraint we get the following:

$$\bigotimes \mathcal{R}\eta > \bigotimes \mathcal{E}\eta \equiv 3 > 2 \equiv True$$

Therefore, this path exhibits a cascade vulnerability problem.

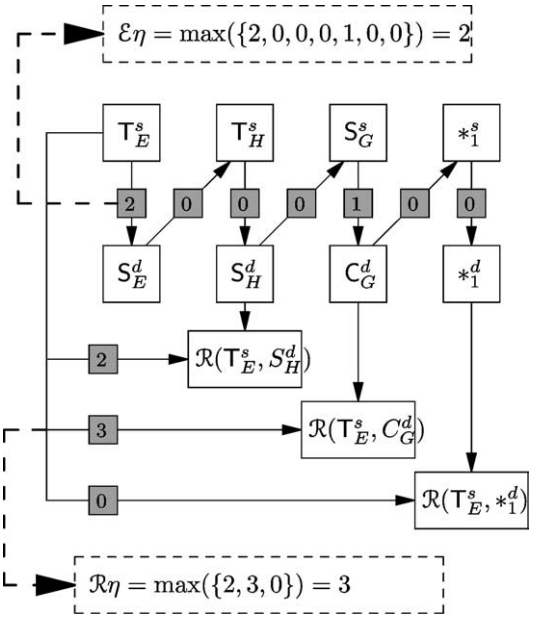

Fig. 9. A cascading path.

6. Eliminating cascade vulnerabilities

In order to eliminate the cascade vulnerability problem from an MLS network it is necessary to remove all of the cascading paths that run through it. However, in breaking links (connections) between systems, the services provided by the network are affected. Therefore, when eliminating the cascade vulnerability problem it is preferable to break as few links as possible.

The set of solutions to the constraint model presented earlier provides all of the cascading paths within the network. Therefore, in order to remove all cascade vulnerabilities from the network, we need to eliminate all solutions to the constraint model.

The problem of finding the *minimum number* of links to break in order to remove all of the cascading paths can be reformulated within our framework as the problem of finding the minimum number of unary constraints (each removing a link) that have to be added to the problem to make it unsatisfiable. This problem has been solved in [1] by solving the corresponding *minimum hitting-set* problem. Given a collection J of subsets of a finite set K, the minimum-hitting set of J is the smallest (cardinality) set $K' \subseteq K$ from J such that K' contains at least one element from each subset in J. Unfortunately, finding the *minimum hitting-set* is NP-complete.

Our approach to solving the problem takes a slightly different approach to [1]. Rather than finding a subset of the links with minimum cardinality, we focus on

finding a set of links that is *minimal* in the sense that no link from such a set can be re-introduced without resulting in a cascading path. That is, we are looking for a set of unary constraints to add to our constraint model (with each unary constraint removing a link) such that the resultant CSP is unsatisfiable, and such that no proper subset of this set would give rise to an unsatisfiable CSP.

Definition 1 (Minimal Set of Removed Links). A set of links that we remove is *minimal* if:

1. The resultant network is cascade free, and
2. No proper subset of this set gives rise to a network without the cascade vulnerability problem.

Central to our approach to eliminating cascading paths is the notion of a *cascading path generator*.

Definition 2 (Cascading Path Generator). A cascading path generator, η, is a path involving a sequence of assignments to path variables that is not a super-sequence of another cascading path. Note that we ignore the artificial values $*^{?}_{s/d}$.

The set of cascading path generators are representative of all cascading paths in the set of solutions to our constraint model. In fact, each cascading path is a cascading path generator itself or one of its extensions. If an extension of a cascading path generator introduces no more critical links (i.e., that are responsible for introducing additional cascading paths) the cascading effect on this path is removed when we remove the problem from the cascading path generator. If other critical links are added, the new links will also appear in another cascading path generator and will be dealt with when eliminating all cascading vulnerability problems due to that generator. Thus, to focus attention on the causes of the cascade vulnerabilities it is sufficient to limit any elimination strategy to the set of cascading path generators. These can be obtained from the set of solutions in polynomial-time.

Each cascading path generator has a very important property: namely, that it is sufficient to remove one link on the path in order to remove the cascading effect associated with it.

Theorem 1. *Given a cascading path generator, η, removing any of the links on that path removes the cascading effect along that path.*

Proof. This follows from the definition of cascading path generator. A cascading path generator represented by η does not, by definition, contain any sub-path that is also a cascading path. Therefore, we can safely remove *any* link in η and be sure the cascade along η (and any other cascading paths that η may generate) will be removed. □

Starting from the set of all of the cascading path generators our goal is to find a set of links to be removed (unary constraints to be added) in order to obtain a cascade free network (a CSP with no solutions). Moreover we want to find a *minimal* set of links, as discussed earlier.

Example 1. Consider multilevel secure systems E, F, G, H, I and J managing information at security levels e, f, g, h, i, j and k that are configured according to Fig. 10. Each system is evaluated at the lowest level of assurance 1, while the risk function specifies $risk(x, y) = 1$ for any level x and y, except for the cases of $risk(\mathsf{e}, \mathsf{h}) = 2$, $risk(\mathsf{e}, \mathsf{i}) = 2$ $risk(\mathsf{f}, \mathsf{i}) = 2$ $risk(\mathsf{g}, \mathsf{j}) = 2$. This configuration results in the following set of cascading path generators:

$$P_1 = \langle L_1, L_2 \rangle$$
$$P_2 = \langle L_2, L_3 \rangle$$
$$P_3 = \langle L_3, L_4 \rangle$$
$$P_4 = \langle L_4, L_5 \rangle.$$

Note that while path $P_5 = \langle L_1, L_2, L_3 \rangle$ is also a cascading path it is not a cascading path generator: P_5 is covered by cascading path generators $\langle L_1, L_2 \rangle$ and $\langle L_2, L_3 \rangle$; removal of any link from each of these paths will also ensure elimination of cascading path P_5.

The sets $\{L_1, L_3, L_5\}$, $\{L_2, L_3, L_5\}$ and $\{L_2, L_4\}$ are all minimal sets of links that we can remove to eliminate all cascade paths in the system. Notice that the cascading path $\{L_2, L_3, L_4\}$ is not minimal because by removing L_3 we can still obtain a cascade-free configuration.

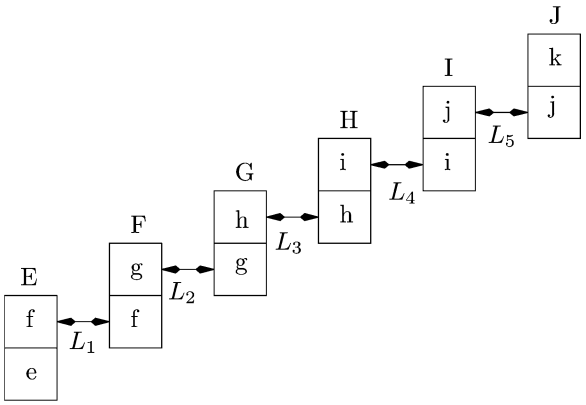

Fig. 10. Multiple cascading path generators.

Following the approach in [12,20] used to find minimal conflict sets, we describe an approach to find a minimal set of links to be removed from the network. The following algorithm is used to select an approximation of a minimal set of removed links (the algorithm is known in the literature as *greedy minimal hitting-set* [19,22,27]):

1. Maintain a counter for each link involved in the set of cascading path generators that need to be removed.
2. Remove the most common link (the link with the highest counter), thus removing all cascading path generators involving that link; in case of a tie a random one is selected.
3. Update the link counters built in Step 1 to reflect the effect of reducing the set of cascading path generators that we need to consider.
4. Continue removing links and updating the link counters until all cascading paths have been removed.

This is a polynomial-time procedure that gives an approximation of a minimal set of removed links. Starting from this approximation we must now verify that it is a *minimal* set.

This is done using the technique proposed in [12,20]. The strategy behind this technique is as follows. Given a set of links $\{l_1, l_2, \ldots, l_n\}$ generated by the procedure above, then l_n must be part of a minimal set of links, since it was required to eliminate all cascading paths. The process is then repeated, this time starting from l_n, and building the set $\{l_n, l_1, l_2, \ldots, l_k\}$ until all cascading paths have again been removed. Since adding a constraint corresponding to l_k leads to an unsatisfiable CSP, l_k must also be part of the minimal set too. At the next step we start by incrementally building the set $\{l_n, l_k, l_1, l_2, \ldots, l_m\}$, until the initial set we use for the current iteration eliminates all cascading paths. An example is presented below to illustrate the strategy.

Example 2. Consider again the system configuration depicted in Fig. 10. Using the greedy minimal hitting-set algorithm, defined above, we could[1] find the following approximation of the minimal set of removed links: $r_1 = \{L_3, L_2, L_5\}$. We now apply the iterative technique [12,20] to computing a minimal conflict as described above. Link L_5 leads to an inconsistent CSP and, therefore, L_5 must be in the minimal set. An alternative set of links to remove is built, starting from L_5 and following the same order as before. In removing link L_5 we remove only path P_4 and then we remove link L_3, removing paths P_2 and P_3. To remove all the paths we have to also remove link L_2, which removes the path P_1. L_2 is the last removed link, so it has to be in the minimal set with L_5.

Starting again, this time from set $\{L_5, L_2\}$, we remove paths P_1, P_2 and P_4. Cascading path generators remain in the network and thus we continue to select more

[1] Notice that more than one result can be obtained if ties are encountered as we consider the link counters.

Table 1
Computing minimal sets of links to break all cascading paths

Step	Removed links	Paths remaining	Partial minimal set of links to break
1.	$\{L_3\}$	$\{P_1, P_4\}$	$\{\}$
2.	$\{L_3, L_2\}$	$\{P_4\}$	$\{\}$
3.	$\{L_3, L_2, L_5\}$	$\{\}$	$\{L_5\}$
4.	$\{L_5\}$	$\{P_1, P_2, P_3\}$	$\{L_5\}$
5.	$\{L_5, L_3\}$	$\{P_1\}$	$\{L_5\}$
6.	$\{L_5, L_3, L_2\}$	$\{\}$	$\{L_5, L_2\}$
7.	$\{L_5, L_2\}$	$\{P_3\}$	$\{L_5, L_2\}$
8.	$\{L_5, L_2, L_3\}$	$\{\}$	$\{L_5, L_2, L_3\}$ (minimal)
1.	$\{L_3\}$	$\{P_1, P_4\}$	$\{\}$
2.	$\{L_3, L_2\}$	$\{P_4\}$	$\{\}$
3.	$\{L_3, L_2, L_4\}$	$\{\}$	$\{L_4\}$
4.	$\{L_4\}$	$\{P_1, P_2\}$	$\{L_4\}$
5.	$\{L_4, L_3\}$	$\{P_1\}$	$\{L_4\}$
6.	$\{L_4, L_3, L_2\}$	$\{\}$	$\{L_4, L_2\}$
7.	$\{L_4, L_2\}$	$\{\}$	$\{L_4, L_2\}$ (minimal)

links for removal. Selecting L_3 results in the removal of P_3 and, since the initial set that we used for the current iteration eliminates all cascading paths, we can say that the set $r_1 = \{L_3, L_2, L_5\}$ is a minimal set of links we can use.

In this case the greedy algorithm gave a minimal set of links. This is not always the case. Suppose the greedy algorithm first removes the link L_3 (that has a counter of 2), then removes link L_2 (that has a counter of 1) and then removes L_4 instead of L_5 (both have counter value 1). We obtain the set $r_2 = \{L_3, L_2, L_4\}$ as the approximation of the minimal set of removed links. Running the algorithm we first remove L_3 (removing paths P_2 and P_3), then L_2 (removing path P_1) and then link L_4 (removing the last path P_4). We now start from the L_4, but must also remove links L_3 and L_2 to remove all paths. Now, starting with the set $r_2^* = \{L_4, L_2\}$ we remove all the paths, and thus r_2^* is a minimal cascading path.

Table 1 depicts the runs of the algorithm for the two examples.

The following results regarding this procedure follow from [12,20].

Theorem 2 (Soundness). *The network we obtain from applying the procedure is cascade free.*

Proof. The procedure used to remove links is applied until no cascading path are present. Therefore, the network is cascade free. □

Theorem 3 (Minimality). *The number of links that are removed from the network is minimal.*

Proof. Notice that we can eliminate the cascading effect from a cascading path generator by removing any one of its links. The minimality of this set follows from [12,20]. □

7. Conclusion

In this paper we have presented a new approach to detecting and eliminating the cascade vulnerability problem in multilevel secure systems based on soft constraints. Soft constraints have been successfully applied to other problems in computer security. The Role-Based Access Control policy model described in [3] uses soft-constraints to define authorisation but does not consider the issue of secure/cascading authorisation. [6,24] considers how soft constraints might be used to specify noninterference-style security properties for systems. In [2] soft constraints are used to represent confidentiality and authentication properties of security protocols. A soft constraint based model is used in [10] to analyze the secure interoperation problem in [16]. These results, and the results in this paper, demonstrate the usefulness of constraints as a general purpose modelling technique for security.

The approach that we present in this paper represents a paradigm shift in the modelling, detection and elimination of the cascade vulnerability problem. In particular, our constraint model provides a natural and declarative description of an arbitrary multilevel secure system. Any solution to the model represents a cascading path, which provides significantly more information regarding the vulnerabilities in the network than the existing approaches. The set of solutions to the proposed constraint model provides a basis for removing the cascade vulnerability problem. Previous approaches [14,18] detect a single cascading path in polynomial time, but correcting the cascade in an optimal way is NP-complete. As described above, detecting all paths in the constraint model is NP-hard, however elimination of a minimal number of links is polynomial.

While constraint solving is NP-complete in general, this has not detracted from its uptake as a practical approach to solving many real-world problems [30]. Using a constraint model, we can rely on a significant body of successful techniques for finding the set of cascading paths, which once found, can be eliminated in polynomial time. These results are applicable to secure interoperation in general.

Acknowledgements

The work in this paper builds upon earlier work presented in two conference papers by the authors [8,9]. We are indebted to the anonymous referees whose comments helped us to improve the presentation considerably.

This work has received partial support from the Italian MIUR project "Constraint Based Verification of Reactive Systems" (COVER) and from Enterprise Ireland under their Basic Research Grant Scheme (Grant Numbers SC/02/289 and SC/2003/007) and their International Collaboration Programme (Grant Number IC/2003/88).

References

[1] J. Bailey and P.J. Stuckey, Discovery of minimal unsatisfiable subsets of constraints using hitting-set dualization, in: *Proc. Practical Aspects of Declarative Languages (PADL 2005)*, Volume 3350 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 174–186.

[2] G. Bella and S. Bistarelli, Soft constraint programming to analysing security protocols, *Theory and Practice of Logic Programming (TPLP), (Special Issue on Verification and Computational Logic)* **4**(5) (2004), 1–28.

[3] V.G. Bharadwaj and J.S. Baras, Towards automated negotiation of access control policies, in: *Proc. of IEEE Workshop Policies for Distributed Systems and Networks*, 2003, pp. 77–80.

[4] S. Bistarelli, *Semirings for Soft Constraint Solving and Programming*, Volume LNCS 2962, Springer, 2004.

[5] S. Bistarelli, H. Fargier, U. Montanari, F. Rossi, T. Schiex and G. Verfaillie, Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison, *CONSTRAINTS: An International Journal* **4**(3) (1999), 199–240.

[6] S. Bistarelli and S.N. Foley, Analysis of integrity policies using soft constraints, in: *Proc. of IEEE Workshop Policies for Distributed Systems and Networks*, 2003, pp. 77–80.

[7] S. Bistarelli and S.N. Foley, A constraint based framework for dependability goals: Integrity, in: *Proc. of 22nd International Conference on Computer Safety, Reliability and Security (SAFECOMP2003)*, Volume 2788 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 130–143.

[8] S. Bistarelli, S.N. Foley and B. O'Sullivan, Detecting and eliminating the cascade vulnerability problem from multi-level security networks using soft constraints, in: *Proceedings of AAAI/IAAI-2004 (16th Innovative Applications of AI Conference)*, AAAI Press, San Jose, 2004, pp. 808–813.

[9] S. Bistarelli, S.N. Foley and B. O'Sullivan, Modelling and detecting the cascade vulnerability problem using soft constraints, in: *Proceedings of ACM Symposium on Applied Computing (SAC-2004)*, ACM Press, 2004, pp. 383–390.

[10] S. Bistarelli, S.N. Foley and B. O'Sullivan, Reasoning about secure interoperation using soft constraints, in: *Proceedings of FAST-2004 Workshop on Formal Aspects of Security and Trust*, 2004.

[11] S. Bistarelli, U. Montanari and F. Rossi, Semiring-based Constraint Solving and Optimization, *JACM* **44**(2) (1997), 201–236.

[12] J.L.N. de Siqueira and J.-F. Puget, Explanation-based generalisation of failures, in: *Proc. 8th European Conference on Artificial Intelligence, ECAI 88*, Pitmann Publishing, 1988, pp. 339–344.

[13] R. Dechter, *Constraint Processing*, Morgan-Kaufmann, 2003.

[14] J.A. Fitch and L.J. Hoffman, A shortest path network security model, *Computers and Security* **12**(2) (1993), 169–189.

[15] S.N. Foley, Conduit cascades and secure synchronization, in: *ACM New Security Paradigms Workshop*, 2000.

[16] L. Gong and X. Qian, The complexity and composability of secure interoperation, in: *Proceedings of the Symposium on Security and Privacy*, Oakland, CA, IEEE Computer Society Press, 1994, pp. 190–200.

[17] S. Gritalis and D. Spinellis, The cascade vulnerability problem: The detection problem and a simulated annealing approach to its correction, *Microprocessors and Microsystems* **21**(10) (1998), 621–628.

[18] J.D. Horton, R. Harland, E. Ashby, R.H. Cooper, W.F. Hyslop, B.G. Nickerson, W.M. Stewart and O.K. Ward, The cascade vulnerability problem, *Journal of Computer Security* **2**(4) (1993), 279–290.

[19] D.S. Johnson, Approximation algorithms for combinatorial problems, *Journal of Computer and System Sciences* **9** (1974), 256–278.

[20] U. Junker, QUICKXPLAIN: Conflict detection for arbitrary constraint propagation algorithms, in: *IJCAI '01 Workshop on Modelling and Solving problems with constraints*, Seattle, WA, USA, 2001.

[21] V. Kumar, Algorithms for constraint-satisfaction problems: A survey, *AI Magazine* **13**(1) (1992), 32–44.

[22] L. Lovasz, On the ratio of optimal integral and fractional covers, *Discrete Mathematics* **13** (1975), 383–390.

[23] J.K Millen and M.W. Schwartz, The cascading problem for interconnected networks, in: *4th Aerospace Computer Security Applications Conference*, 1988, pp. 269–273.

[24] A. Di Pierro, C. Hankin and H. Wiklicky, On approximate non-interference, in: *Proc. of Workshop on Issues in the Theory of Security*, IFIP WG1.7, 2002.

[25] J.-C. Regin, A filtering algorithm for constraints of difference in csps, in: *Proceedings AAAI-94*, 1994, pp. 362–367.

[26] T. Schiex, Possibilistic constraint satisfaction problems, or "how to handle soft constraints"? in: *Proc. 8th Conf. of Uncertainty in AI*, 1992, pp. 269–275.

[27] P. Slavík, A tight analysis of the greedy algorithm for set cover, in: *Proc. of the 28th ACM Symposium on Theory of Computing (STOC)*, 1996, pp. 435–441.

[28] TCSEC, Computer security requirements – guidance for applying the department of defense trusted computer system evaluation criteria in specific environments, Technical Report CSC-STD-003-85, National Computer Security Center, Orange Book, 1985.

[29] TNI, Trusted computer system evaluation criteria: Trusted network interpretation, Technical report, National Computer Security Center, Red Book, 1987.

[30] M. Wallace, Practical applications of constraint programming, *Constraints* **1**(1–2) (1996) 139–168.