Modelling and Detecting the Cascade Vulnerability Problem using Soft Constraints

Stefano Bistarelli
Dipartimento di Scienze
Università di Chieti-Pescara, Italy
Istituto di Informatica e Telematica, C.N.R.
Pisa, Italy
stefano.bistarelli@iit.cnr.it

Simon N. Foley and Barry O'Sullivan
Department of Computer Science
University College Cork
Ireland
{s.foley,b.osullivan}@cs.ucc.ie

ABSTRACT

Establishing network security is based not just on the security of its component systems but also on how they are configured to interoperate. In this paper we consider how soft constraints provide an approach to detecting the cascade vulnerability problem: whether system interoperation provides circuitous or cascading routes across the network that increase the risk of violation of multilevel security. Taking the constraints approach means that we are building on techniques that have proven success in solving large-scale problems from other domains.

Categories and Subject Descriptors

D.3.3 [**Programming Languages**]: Language Constructs and Feature—*Constraints*; K.6.5 [**Management of Computing and Information Systems**]: Security and Protection; D.4.6 [**Operating Systems**]: Security and Protection—*Information flows control*

General Terms

Security

Keywords

Constraints, Soft Constraints, Security, Multilevel Security

1. INTRODUCTION

In its most general case, determining the security (that is, the safety problem) of a system is undecidable [18]. This has led to the design of a wide range of decidable security mechanisms that are based on more restrictive forms of security, for example, [3, 6]. These mechanisms decide whether an access by a subject is authorised according to the rules set out in a security policy. A system is secure (upholds its security policy) if it is not possible for a subject to gain unauthorised access.

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, 15][1]. 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 inter-operation must be limited if the administration security policy states that Clare is not permitted access to Bob's (administration) files.

The *cascade vulnerability problem* [2, 21] is also concerned with secure inter-operation, and considers the *assurance risk* of composing multilevel secure systems that are evaluated to different levels of assurance according to the criteria [2]. 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 topsecret across the network to classified.

Existing research has considered schemes for *detecting* these security vulnerabilities and for *correcting* them by re-configuring system inter-operation. While detection of some security vulnerability [21, 19, 14, 16] can be easily achieved, their optimal correction is NP-complete [19, 17, 16] and simulated annealing and integer linear programming are suggested for possible practical approximations.

We are investigating the potential of using constraints [9,

[1]McCullough [20] demonstrates that a form of process composition does not necessarily preserve the noninterference (security) property. This paper takes the more abstract viewpoint and assumes that the underlying security properties of the systems are composable.

SAC '04 March 14–17, 2004, Nicosia, Cyprus

22] for modelling secure inter-operation. Constraint Solving is an emerging software technology for declarative description and effective solving of large problems. The constraint programming process consists of the generation of requirements (constraints) and solution of these requirements, by specialised constraint solvers. The advantages of expressing secure inter-operation as a constraint satisfaction problem is that there exists a wide body of existing research results on solving this problem for large systems of constraints in a fully mechanised manner [12, 24, 4]. Constraints have been used in many practical analysis tools, such as Concurrent Engineering and Computer Aided Verification [10, 11]. Thus, the results in this paper provide a direction for the development of practical configuration analysis tools for secure inter-operation.

In this paper we describe how the cascade vulnerability problem can be modelled using soft constraints [9], and consider their potential for detecting cascades. These results are applicable to secure inter-operation in general. The paper is organised as follows. Section 2 provides background on the cascade problem and soft constraints. Section 2.1 describes our general model of the cascade problem and this is modelled using soft constraints in Section 3. Section 4 codifies an example within the proposed soft constraints model.

2. BACKGROUND

2.1 The Cascade Problem

Figure 1 gives an example of a MLS network configuration with a cascade vulnerability problem [14]. The network is

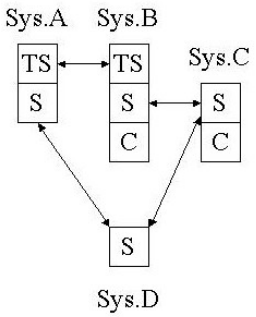

Figure 1: Network configuration with a potential cascade problem.

comprised of multilevel secure systems Sys.A, Sys.B, Sys.C and Sys.D storing classified (C), secret (S) and top-secret (T) information as depicted in Figure 1. Each system is accredited according to levels of assurance C2<B1<B2<B3 from [2, 1]. For example, Sys.B is used to simultaneously store C, S and T information and, therefore, (according to [2, 1]) 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 T information to C. Sys.D, 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 Figure 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.A, to S on Sys.D to S to C on Sys.C. This is contrary to the criteria 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 problem is defined as follows.

2.1.1 MLS

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

2.1.2 Assurance Levels

A security criteria defines a lattice[2] $\mathcal{A}$ of assurance levels with ordering $\leq$. Given $x, y : \mathcal{A}$, then $x \leq y$ means that a system evaluated at y is no less secure than a system evaluated at x, or alternatively, that an attacker than can compromise a system evaluated at y can compromise a system evaluated at x. Let $\mathcal{S}$ define the set of all system names. Define $accred : \mathcal{S} \rightarrow \mathcal{A}$ where $accred(s)$ gives the assurance level of system $s : S$, 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 $risk : L \times L \rightarrow \mathcal{A}$, such that given $l, l' : L$ 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 generalisation we assume that there is no security enforcement at the lowest assurance level $\mathbf{0}$, and thus, if $l \leq l'$ then $risk(l, l') = \mathbf{0}$. For example, function $risk$ encodes the assurance matrix (for B levels) from [2, 1] as $risk(C, S) = risk(C, T) = risk(S, T) = 0$, $risk(S, C) = 1$, $risk(T, S) = 2$, and $risk(T, C) = 3$, 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

$$risk(l, l') <= accred(s).$$

2.1.5 Network Model

A node is a pair (s, l) and represents the fact that system s can hold information at level l. A system is a collection of nodes that represent the data it holds. For example, in Figure 1, Sys.A is represented by nodes $(Sys.A, S)$ and $(Sys.A, T)$. A network of systems is a weighted graph of these nodes according to how they are connected. An w-weighted arc from (s, l) to (s', l') means that it requires 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

[2]This generalises the total ordering of assurance levels defined in [2, 1].

Arcs are used to represent direct flows within a system and interoperation links between systems. A flow $l \leq l'$ that is permitted on system s is represented as a (assurance) $\mathbf{0}$-weighted arc from (s,l) to (s,l'); 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 (s,l) to (s,l').

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

Given pairs (s,l) and (s,l') we then define

$$effort((s,l),(s',l'))$$

as the minimum effort required to compromise the network and copy/downgrade level l information held on system s to level l' information on system s'. As an example, in Figure 1, $effort((\text{Sys.A},T),(\text{Sys.C},C)) = B2$ via the path through Sys.D.

2.1.7 Cascade Freedom

We require that for any systems s,s' and levels l,l' then

$$risk(l,l') <= effort((s,l),(s',l'))$$

Given a path in the network from (s,l) to (s',l'), then its cascade weighting 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 (s,l) to (s',l') by using this path. $effort((s,l),(s',l'))$ is the minimum of the cascade weightings for all paths that connect (s,l) to (s',l').

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 [9], which can be shown to generalise and express many of the others [7].

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 $\leq_S$ over $\mathcal{A}$ such that $a \leq_S b$ iff $a + b = b$. Then it is possible to prove that (see [9]): 1. $\leq_S$ is a partial order; 2. $+$ and $\times$ are monotone on $\leq_S$; 3. $\mathbf{0}$ is its minimum and $\mathbf{1}$ its maximum; 4. $\langle \mathcal{A}, \leq_S \rangle$ is a complete 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 $\times$; $\langle \mathcal{A}, \leq_S \rangle$ is a complete distributive lattice and $\times$ its *glb* (*greatest lower bound*). Informally, the relation $\leq_S$ gives us a way to compare semiring values and constraints. In fact, when we have $a \leq_S b$, we will say that b *is better than* a. In the following, when the semiring will be clear from the context, $a \leq_S b$ will be often indicated by $a \leq b$.

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 $\mathcal{C} = \eta \rightarrow \mathcal{A}$ 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 $c_{x,y} : V \rightarrow D \rightarrow \mathcal{A}$, 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 $c : V \rightarrow D \rightarrow \mathcal{A}$ to a function $\eta : V \rightarrow D$; what we obtain, is a semiring value $c\eta = a$.

A *soft constraint satisfaction problem* 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] 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.

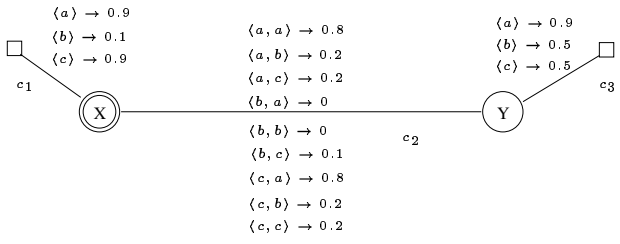

Figure 2: A fuzzy CSP.

2.2.3 Combining and projecting soft constraints.

Given the set $\mathcal{C}$, the combination function $\otimes : \mathcal{C} \times \mathcal{C} \to \mathcal{C}$ 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 Figure 2 associates a semiring element to 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 $\langle a, b\rangle \to 0.2$, $\langle a, c\rangle \to 0.2$, $\langle b, a\rangle \to 0$, $\langle b, b\rangle \to 0$, $\langle b, c\rangle \to 0.1$, $\langle c, a\rangle \to 0.8$, $\langle c, b\rangle \to 0.2$ and $\langle c, c\rangle \to 0.2$. The obtained tuples are then projected over variable x, obtaining the solution $\langle a\rangle \to 0.8$, $\langle b\rangle \to 0.1$ and $\langle c\rangle \to 0.8$.

3. MODELLING MLS NETWORKS

Consider a network $N = \{A, B, C, \ldots\}$ of a finite arbitrary number n of systems. In our constraint model, this network of n nodes is represented using $2 \times n$ *system-node variables*. Each system-node variable S_i^s and S_i^d, for $i := 1 \ldots n$ can be instantiated to be one system of the network. Each of the possible flows of information among the systems of the network are represented by a specific instantiation of the variables S_1^s,S_1^d,S_2^s,S_2^d,...,S_n^s,S_n^d. In particular, the instantiation of the pair of nodes S_i^s and S_i^d, for $i := 1 \ldots n$, represents the flow from the source S_i^s to the destination S_i^d inside the i-th System in the specific path. Similarly, instantiation of S_i^d and S_{i+1}^s, for $i := 1 \ldots n-1$ represents the flow among the the i-th and the $i+1$-th System in the specific instantiated path.

Consider for instance the network $N = \{A, B\}$ represented in Figure 3 involving two systems, A and B, with system A handling information at level Top-Secret (T) and Secret (S), and system B handling information at level Secret (S) and Confidential (C). We can capture this instance by using 4 system-node variables: S_1^s,S_1^d,S_2^s,S_2^d.

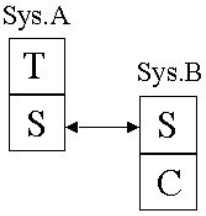

Figure 3: A simple network.

3.1 System-Node Variable Domains

The domain of each system-node variable contains pieces of information describing the possible security levels available on each system. In particular, each *source* variable S_i^s contains domain elements marked with s, and each *destination* variable S_i^d contains domain elements marked with d.

The network in Figure 3 has in our model 4 variables $S_1^s, S_1^d, S_2^s, S_2^d$ with domain $D(S_i^s) = \{T_A^s, S_A^s, S_B^s, C_B^s\}$, with $i := 1, 2$, and $D(S_i^d) = \{T_A^d, S_A^d, S_B^d, C_B^d\}$, with $i := 1, 2$.

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

3.2 Modelling each System

The constraint on each system defines three classes of system flows.

- $Flow_{permitted}$ represents the flows permitted by the policy in each node;
- $Flow_{risk}$: represents the flows that are not permitted by the policy, but for which there is a risk of flow if the system became compromised;
- $Flow_{invalid}$: represent all the remaining flows and are not valid.

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

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

Given this semiring, the constraint $c_{(S_i^s, S_i^d)}$ representing the

flow inside system i is defined as follows:

$$c_{(S_i^s,S_i^d)}(s,d) = \begin{cases} accred(S_i) & \text{when}(s,d) \in Flow_{risk} \\ & \text{(risk flows)} \\ 0 & \text{when}(s,d) \in Flow_{permitted} \\ & \text{(permitted flows)} \\ +\infty & \text{otherwise.} \\ & \text{(invalid flows)} \end{cases}$$

Recall that $accred(S_i)$ is the accreditation value of System i. For example, given the MLS policy ordering $C \leq S \leq T$, then we have

$$Flow_{permitted} = \{(T_A^s, T_A^d), (S_A^s, S_A^d), (S_A^s, T_A^d), (S_B^s, S_B^d), (C_B^s, C_B^d), (C_B^s, S_B^d)\}$$
$$Flow_{risk} = \{(T_A^s, S_A^d), (S_B^s, C_B^d)\}$$

and $Flow_{invalid}$ set contains all the remaining tuples.

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

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

3.3 Modelling the Network

Flow constraints between systems result in two classes of network flows.

- $Network_{permitted}$ represents flows permitted by the connection policy between each system and represents direct synchronisation flows between systems.
- $Network_{invalid}$: this represents the absence of direct connection between the systems.

Between each pair of systems, S_i and S_{i+1}, we define a soft constraint, $c_{(S_i^d,S_{i+1}^s)}$, that defines the possible synchronisation between systems i and $i+1$. Note that these constraints are defined between the destination system-node variable of the first system, S_i^d, and the source system-node variable of the second system, S_{i+1}^s. The constraint $c_{(S_i^d,S_{i+1}^s)}$ representing the synchronization flows between system i and $i+1$ can be defined as follows:

$$c_{(S_i^d,S_{i+1}^s)}(d,s) = \begin{cases} 0 & \text{when}(d,s) \in Network_{permitted} \\ & \text{(Policy permitted synchronization)} \\ +\infty & \text{otherwise.} \\ & \text{(invalid synchronization)} \end{cases}$$

For example, constraint $Network_{permitted}$ for Figure 3 is defined as follows (note that $Network_{impossible}$ contains the remaining tuples).

$$Network_{permitted} = \{(S_A^d, S_B^s), (S_B^d, S_A^s)\}$$

Note that the proposed 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.

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

$$c_{(S_i^d,S_{i+1}^s)}(d,s) = \begin{cases} 0 & \text{when}(d,s) \in \{(*_1^d, *_2^s), \ldots, (*_{i-3}^d, *_{i-2}^s)\} \\ & \quad \cup \{(\sharp, *_1^s) \text{s.t. } \sharp \in D(S_i^d)\} \\ & \text{(Artificial permitted synchronization)} \\ +\infty & \text{otherwise} \\ & \text{(Artificial invalid synchronization)} \end{cases}$$

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.

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

The solution of the defined Soft CSP (let us call this E), that is all the solution with level lower than $+\infty$, returns all the possible cascade path of the system. The level associated to each path gives a measure of the effort required to compromise the network for that specific path.

3.4 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 with the risk of compromising the system as a whole. Therefore, we introduce a set of *risk* constraints, $R = \{r_{(S_1^s,S_i^d)} | i \in \{2, \ldots, n\}\}$. The weight of each instance of $r_{(S_1^s,S_i^d)}$ represents the risk associated with the path from S_1^s to S_i^d. The cost of each tuple in these constraints is defined as follows:

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

The set of solutions of the soft CSP E (that is the cascade-CSP defined in Sections 3.1, 3.2, 3.3)whose associated semiring level is lower than $+\infty$ provides the set of possible paths of the network. Each solution-path of E gives the minimum *effort* required to compromise the network, while the combination of the constraints in 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, i.e. where the following constraint is satisfied:

$$\bigotimes R\eta > \bigotimes E\eta$$

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 from the model.

4. AN EXAMPLE

In this section we encode the network example described in Section 2.1 within the proposed constraints model. Figure 4 depicts the structure of the constraint relationships in this model. In Section 4.1 we present an example of how our model identifies a cascade-free path. In Section 4.2 we present an example of detecting a path with a cascade problem.

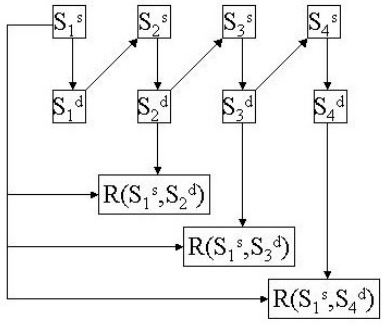

Figure 4: A constraint Model Structure.

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

Figure 4 presents the structure of the constraint model for an example from [14]. Our model comprises 8 system-node variables, $S_1^s, S_1^d, S_2^s, S_2^d, S_3^s, S_3^d, S_4^s$, and S_4^d, and 3 risk variables, $r_{(S_1^s,S_2^d)}, r_{(S_1^s,S_3^d)}$ and $r_{(S_1^s,S_4^d)}$. The domain of each system-node variable, $D(S_i^?)$, is: $\{T_A^?, S_A^?, T_B^?, S_B^?, C_B^?, S_C^?, C_C^?, S_D^?\}$ (where ? stands alternatively for s and d) and $i := 1, \ldots, 4$.

4.1 A Cascade-free Path

Consider the following path through the network:

$$\eta = [S_1^s := T_A^s, S_1^d := T_A^d, S_2^s := T_B^s, S_2^d := S_B^d, \\ S_3^s := S_C^s, S_3^d := C_C^d, S_4^s := *_1^s, S_4^d := *_1^d]$$

This scenario is illustrated in Figure 6. Evaluating the cascade detection constraint we get the following, and therefore, this path there is not a cascade.

$$\bigotimes R\eta > \bigotimes E\eta \equiv 3 > 3 \equiv False$$

4.2 A Cascading Path

Consider the following path through the network:

$$\eta = [S_1^s := T_A^s, S_1^d := S_A^d, S_2^s := S_D^s, S_2^d := S_D^d, \\ S_3^s := S_C^s, S_3^d := C_C^d, S_4^s := *_1^s, S_4^d := *_1^d]$$

This is depicted in Figure 6. Evaluating the cascade detection constraint we get the following, and therefore, this path does exhibit a cascade problem.

$$\bigotimes R\eta > \bigotimes E\eta \equiv 3 > 2 \equiv True$$

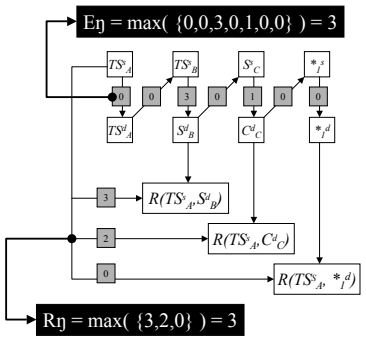

Figure 5: Cascade Free Path.

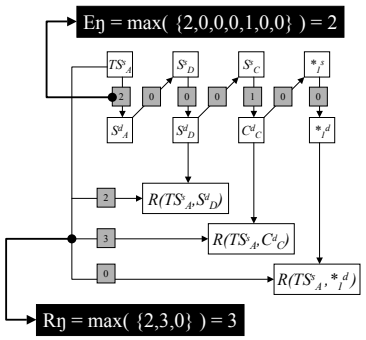

Figure 6: Cascading Path.

5. CONCLUSION

In this paper we have presented a new approach to detecting 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 [27] uses soft-constraints to define authorisation but does not consider issue of secure/cascading authorization . [8, 23] considers how soft constraints might be used to specify noninterference-style security properties for systems. In [5] soft constraints are used to represent confidentiality and authentication properties of security protocols. These results, and the results in this paper, demonstrate the usefulness of constraints as a general purpose modelling technique for security.

While constraint solving is NP-complete in general, this has not detracted from the uptake of constraint processing as a practical approach to solving many real-world problems [28], and should not be regarded as a fatal disadvantage – on the contrary, constraint solving is becoming the paradigm of choice in many large-scale optimisation problems. Many tractable classes of constraint satisfaction problem have been identified, and there are also a number of pow-

erful "global constraints" which have polynomial-time inference algorithms associated with them. For example, particular constraints which is of relevance here are the shortest path constraint [13] and the "all different" constraint [25].

The approach we present in this paper represents a paradigm shift in the modelling and detection of the cascade 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, 19] detect a single cascading path in polynomial time, but correcting the cascade in an optimal way is NP-complete. As discussed above, detecting all paths in the constraint model is NP-complete, however we conjecture that the correction of the cascade problem, in this model, will be polynomial; this will be a focus of our future work. We also plan to implement a software tool to apply the theoretical results of the paper.

6. ACKNOWLEDGMENTS

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 Number SC/02/289) and their International Collaboration Programme (Grant Number IC/2003/88).

7. REFERENCES

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

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

[3] P. Amman and R. Sandhu. The extended schematic protection model. *Journal of Computer Security*, 1(4), 1992.

[4] G. J. Badros, A. Borning, and P. J. Stuckey. The cassowary linear arithmetic constraint solving algorithm. *ACM Transactions on Computer Human Interaction*, 8(4):276–306, dec 2001.

[5] G. Bella and S. Bistarelli. Soft Constraints for Security Protocol Analysis: Confidentiality. In *Proc. of the 3rd International Symposium on Practical Aspects of Declarative Languages (PADL'01)*, LNCS 1990, pages 108–122. Springer-Verlag, 2001.

[6] E. Bertino et al. An authorization model and its formal semantics. In *Proceedings of the European Symposium on Research in Computer Security*, pages 127–142. Springer LNCS 1485, 1998.

[7] 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. Kluwer*, 4(3), 1999.

[8] S. Bistarelli and S. Foley. Analysis of integrity policies using soft constraints. In *Proceedings of IEEE Workshop Policies for Distributed Systems and Networks*, pages 77–80, June 2003.

[9] S. Bistarelli, U. Montanari, and F. Rossi. Semiring-based Constraint Solving and Optimization. *Journal of the ACM*, 44(2):201–236, Mar 1997.

[10] W. Chan, R. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In O. Grumberg, editor, *Computer Aided Verification, 9th International Conference, CAV'97 Proceedings*, volume 1254 of *Lecture Notes in Computer Science*, pages 316–327, Haifa, Israel, June 1997. Springer-Verlag.

[11] G. Delzanno and T. Bultan. Constraint-based verification of client-server protocols. In *Proceedings CP2001*, volume 2239 of *Lecture Notes in Computer Science*, pages 286–?? Springer-Verlag, 2001.

[12] M. Dincbas, P. V. Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. The constraint logic programming language chip. In *Proceedings of FGCS*, pages 693–702, 1988.

[13] T. Fahle, U. Junker, S. Karisch, N. Kohl, M. Sellmann, and B. Vaaben. Constraint programming based column generation for crew assignment. Technical Report TR-RI-99-212, University of Paderborn, 1999.

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

[15] S. 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*, pages 190–200, Oakland, CA, May 1994. IEEE Computer Society Press.

[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):621–628, 1998.

[18] M. Harrison, W. Ruzzo, and J. Ullman. Protection in operating systems. *Communications of the ACM*, 19:461–471, 1976.

[19] R. Horton et al. The cascade vulnerability problem. *Journal of Computer Security*, 2(4):279–290, 1993.

[20] D. McCullough. Noninterference and the composability of security properties. In *Proceedings 1988 IEEE Symposium on Security and Privacy*, pages 177–186. IEEE Computer Society Press, 1988.

[21] J. Millen and M. Schwartz. The cascading problem for interconnected networks. In *4th Aerospace Computer Security Applications Conference*, pages 269–273. IEEE CS Press, Dec. 1988.

[22] U. Montanari. Networks of constraints: Fundamental properties and applications to picture processing. *Information Science*, 7:95–132, 1974. Also Technical Report, Carnegie Mellon University, 1971.

[23] A. D. Pierro, C. Hankin, and H. Wiklicky. On approximate non-interference. In *editor, Proceedings of WITS'02 – Workshop on Issues in the Theory of Security*. IFIP WG1.7, 2002.

[24] J. Puget. A c++ implementation of clp. In

Proceedings of the 2nd Singapore International Conference on Intelligent Systems, 1994.

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

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

[27] J. B. V.G. Bharadwaj. Towards automated negotiation of access control policies. In *Proceedings of IEEE Workshop Policies for Distributed Systems and Networks*, pages 77–80, June 2003.

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