Detecting and Eliminating the Cascade Vulnerability Problem from Multi-level Security Networks using Soft Constraints[*]

Stefano Bistarelli
Dipartimento di Scienze
Università di Chieti-Pescara, Italy
bista@sci.unich.it
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

The security of a network configuration is based, not just on the security of its individual components and their direct interconnections, but it is also based on the potential for systems to inter-operate indirectly across network routes. Such inter-operation 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 programming 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 violate 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, upon which we can compute in polynomial time an optimal reconfiguration of the network and ensure security.

Introduction

The composition of secure systems is not necessarily secure. A user may be able to gain unauthorized access to an object by taking a circuitous access route across individually secure but inter-operating systems (Foley 2000; Gong & Qian 1994). Determining security is based not just on the individual system authorization mechanisms but also on how the systems are configured to inter-operate. 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* (TNI 1987) is 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 specified in (TNI 1987). The transitivity of the multilevel security policy upheld across all secure systems ensures that their multilevel composition is secure; however, inter-operability 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 inter-operate 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 inter-operation. While detection of some security vulnerability (Millen & Schwartz 1988; Horton & others 1993; Fitch & Hoffman 1993; Gong & Qian 1994) can be easily achieved, their optimal elimination is NP-complete (Horton & others 1993; Gritalis & Spinellis 1998; Gong & Qian 1994).

We present an approach to using constraints (Dechter 2003) for reasoning about secure inter-operation. Constraint solving is an emerging software technology for modeling and solving large-scale optimization problems (Wallace 1996). Constraints have been successfully applied to a number of problems in computer security (Bharadwaj & Baras 2003; Bistarelli & Foley 2003; Pierro, Hankin, & Wiklicky 2002; Bella & Bistarelli 2001). However, the cascade vulnerability problem, and secure inter-operation in general, have not been studied in the context of constraints.

The approach that we present in this paper represents a paradigm shift in the modeling, detection and elimination of the cascade 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 (Fitch & Hoffman 1993; Horton & others 1993) detect a single cascading

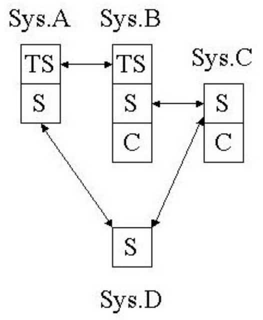
[*]This work has received partial support from the Italian MIUR project "Constraint Based Verification of Reactive Systems" (COVER), from Enterprise Ireland under their Basic Research Grant Scheme (Grant Number SC/02/289) and their International Collaboration Programme (Grant Number IC/2003/88) and the Boole Centre for Research in Informatics, UCC, under the HEA-PRTLI scheme.

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 with reasonable effort, which once found, can be eliminated in polynomial time. These results are applicable to secure inter-operation in general.

Background

The Cascade Problem

Figure 1 gives an example of a multi-level secure (MLS) network configuration with a cascade vulnerability problem (Fitch & Hoffman 1993).



Assurance Ratings	
System	Level
Sys.A	B2
Sys.B	B3
Sys.C	B1
Sys.D	C2

Figure 1: Network configuration with a potential cascade problem.

The network is 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 (TNI 1987; TCSEC 1985). For example, Sys.B is used to simultaneously store C, S and T information and, therefore, (according to (TNI 1987; TCSEC 1985)) 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 inter-operability 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 inter-operation. A generalized form of the cascade problem is defined as follows.

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, $C \leq S \leq T$.

Assurance Levels Security criteria define a lattice, $\mathcal{A}$, 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 that can compromise a system evaluated at y can compromise a system evaluated at x. Let $\mathcal{S}$ define the set of all system names. We 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.

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 generalization 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 from (TNI 1987; TCSEC 1985) as $risk(C, S) = risk(C, T) = risk(S, T) = 0$ [C2], $risk(S, C) = 1$ [B1], and so forth.

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 $risk(l, l') \leq accred(s)$.

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'.

Cascading Risks Arcs are used to represent direct flows within a system and inter-operation 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, $effort((Sys.A, T), (Sys.C, C)) = B2$ via the path through $Sys.D$.

Cascade Freedom We require that for any systems s,s' and levels l,l' then $risk(l, l') \leq effort((s, l), (s', l'))$. 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').

Soft Constraints

Several formalizations of the concept of *soft constraints* are currently available. In the following, we refer to the one based on c-semirings (Bistarelli, Montanari, & Rossi 1997) which can be shown to generalize and express many of the others (Bistarelli *et al.* 1999).

Soft constraints associate a qualitative or quantitative value either with the entire constraint or with each assignment of its variables. More precisely, they are based on a semiring structure $S = \langle A, +, \times, \mathbf{0}, \mathbf{1}\rangle$ and a set of variables V with domain D. In particular the semiring operation $\times$ is used to combine constraints together, and the $+$ operator for projection.

Technically, a *constraint* is a function which, given an assignment $\eta : V \rightarrow D$ of the variables, returns a value of the semiring. So $\mathcal{C} = \eta \rightarrow A$ is the set of all possible constraints that can be built starting from S, D and V (values in A are interpreted as level of preference or importance or cost). Using the levels, we can order constraints: to say that c_1 is better then c_2 we will write $c_1 \sqsupseteq c_2$.

When using soft constraints it is necessary to specify, via suitable combination operators, how the level of preference of a global solution is obtained from the preferences in the constraints. The combined weight of a set of constraints is computed using the operator $\otimes : \mathcal{C} \times \mathcal{C} \rightarrow \mathcal{C}$ defined as $(c_1 \otimes c_2)\eta = c_1\eta \times_S c_2\eta$. Moreover, 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. $c'\eta = \sum_{d\in D} c\eta[v := d]$.

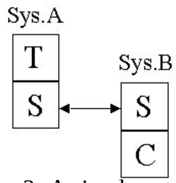


Figure 2: A simple network.

Modeling 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,$\ldots$,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 2 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.

System-Node Variable Domains

The domain of each system-node variable defines the set of 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 2 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 $D(S_i^?)' = D(S_i^?) \cup \{*_1^?, *_2^?, \ldots, *_{i-2}^?\}$. These $*$ elements are added to deal with paths shorter than n. This is necessary because solving 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.

Modeling each System

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

- $Flow_{permitted}$ represents the information flows permitted by the policy in each node;
- $Flow_{risk}$ 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;
- $Flow_{invalid}$ represents all the remaining flows which are not valid (i.e. that are impossible for the given network).

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 this paper, although our results are general and 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:

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

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

$$\begin{aligned} Flow_{permitted} &= \{(T_A^s, T_A^d), (S_A^s, S_A^d), (S_A^s, T_A^d), \\ &\quad (S_B^s, S_B^d), (C_B^s, C_B^d), (C_B^s, S_B^d)\} \\ Flow_{risk} &= \{(T_A^s, S_A^d), (S_B^s, C_B^d)\} \end{aligned}$$

and the $Flow_{invalid}$ 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 consider these "artificial" values. In particular, we extend the definition of each constraint $c_{(S_i^s, S_i^d)}$ as follows:

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

Modeling the Network

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

- $Network_{permitted}$ represents information flows permitted by the connection policy between each system and represents direct synchronization flows between systems;
- $Network_{invalid}$ 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 synchronizations 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:

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

For example, constraint $Network_{permitted}$ for Figure 2 is defined as follows.

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

Note that the model does not consider assurance risks for connections: this can be achieved, if desired, by explicitly modeling 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:

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

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* (Regin 1994) constraint amongst all the variables in 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, E, that is all the solution with level lower than $+\infty$, represents all of the possible paths through the system. The semiring level associated with each path gives a measure of the effort required to compromise the network using that specific path.

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 Effort-CSP defined above) 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 E represents 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.

An Example

In this section we encode the network example described in Figure 2 within the proposed constraints model. Figure 3 depicts the structure of the constraint relationships in this model. We first present an example of how our model identifies a cascade-free path, before we present an example of detecting a cascading path.

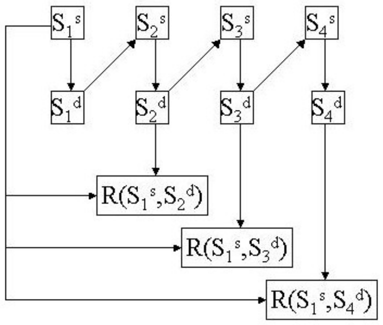


Figure 3: A constraint Model Structure.

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 3 presents the structure of the constraint model for an example from (Fitch & Hoffman 1993). 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 $i := 1, \ldots, 4$. Note that we also extend each domain using $*_i^?$ values as described above, but don't show this here for conciseness.

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 4.

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

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

A Cascading Path

Consider the following path through the network, depicted in Figure 5:

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

Evaluating the cascade detection constraint we get the following, characterizing a cascading path:

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

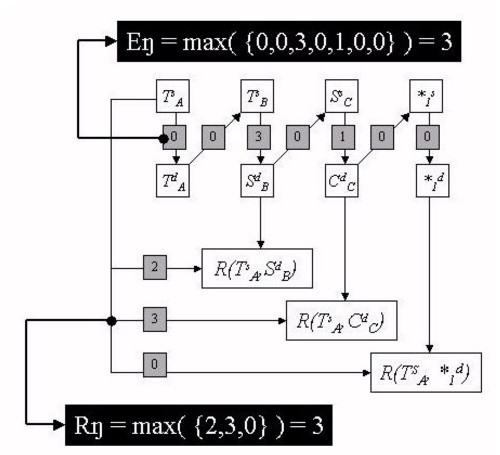


Figure 4: Cascade-free Path.

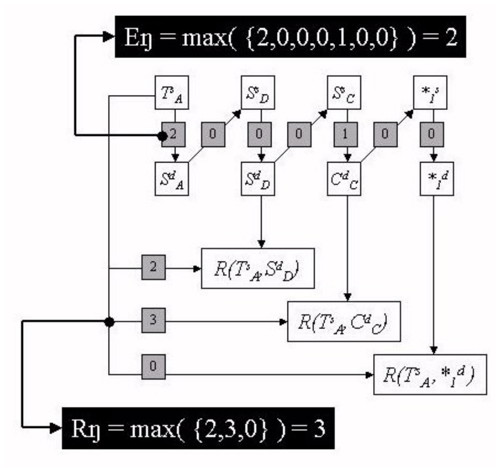


Figure 5: Cascading Path.

Eliminating Cascade Vulnerabilities

Clearly, in order to eliminate the cascade vulnerability problem from an MLS network we need to remove all cascading paths through it. However, by breaking links (connections) between systems, the services provided by the network are affected. Therefore, when eliminating the cascade vulnerability problem we require that the number of links to be broken is minimal.

The set of solutions to the constraint model presented earlier provides all the cascading paths in the network. Therefore, in order to remove all cascade vulnerabilities from the network, we need to eliminate all solutions to the constraint model. Central to our approach to eliminating cascading paths is the notion of a *minimal cascading path*.

Definition 1 (Minimal Cascading Path) *A minimal cascading path, η, is a path involving a sequence of assignments to system-node variables that is not a sub-sequence of another cascading path. Note that we ignore the artificial values $*_i^?$.*

Recall that the set of solutions to the constraint model not only contains all minimal cascading paths, but also all of their extensions. Thus, in order to focus attention on the causes of the cascade vulnerabilities we must limit ourselves to the set of minimal cascading paths. These can be obtained from the set of solutions in polynomial-time.

Each minimal cascading path 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 minimal cascading path η, removing any of the links on that path removes the cascading effect along that path.*

Therefore, given the set of minimal cascading paths, we simply apply the following simple steps in order to remove the minimal number of links to eliminate the cascade vulnerability problem:

1. we maintain a counter for each link involved in the set of minimal cascading paths we need to remove;
2. we remove the most common link (the link with the highest counter), thus removing all minimal cascading paths involving that link;
3. we update the link counters built in Step 1 to reflect the effect of reducing the set of minimal cascading paths that we need to consider;
4. we continue removing links and updating the link counters until all cascading paths have been removed.

This is a polynomial-time procedure, for which the following can be easily proven:

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

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

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. These results demonstrate the usefulness of constraints as a general purpose modeling technique for security.

The approach we present in this paper represents a paradigm shift in the modeling, detection and elimination of the cascade problem. In particular, our constraint model provides a natural and declarative description of an arbitrary multilevel secure network. 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 (Fitch & Hoffman 1993; Horton & others 1993) 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-complete, however their optimal elimination 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 (Wallace 1996). 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. In this paper we explored the cascade vulnerability problem in the context of inter-operating multi-level secure systems. We expect that it is straightforward to generalize these results to cascades within secure inter-operation in general (Foley 2000; Gong & Qian 1994).

References

Bella, G., and Bistarelli, S. 2001. Soft Constraints for Security Protocol Analysis: Confidentiality. In *Proc. of PADL'01*, LNCS 1990, 108–122.

Bharadwaj, V., and Baras, J. 2003. Towards automated negotiation of access control policies. In *Proc. of IEEE Workshop Policies for Distributed Systems and Networks*, 77–80.

Bistarelli, S., and Foley, S. 2003. Analysis of integrity policies using soft constraints. In *Proc. of IEEE Workshop Policies for Distributed Systems and Networks*, 77–80.

Bistarelli, S.; Fargier, H.; Montanari, U.; Rossi, F.; Schiex, T.; and Verfaillie, G. 1999. Semiring-based CSPs and Valued CSPs: Frameworks, properties, and comparison. *CONSTRAINTS: An international journal. Kluwer* 4(3).

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

Dechter, R. 2003. *Constraint Processing*. Morgan Kaufmann.

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

Foley, S. 2000. Conduit cascades and secure synchronization. In *ACM New Security Paradigms Workshop*.

Gong, L., and Qian, X. 1994. The complexity and composability of secure interoperation. In *Proceedings of the Symposium on Security and Privacy*, 190–200. Oakland, CA: IEEE Comp. Soc. Press.

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

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

Millen, J., and Schwartz, M. 1988. The cascading problem for interconnected networks. In *4th Aerospace Computer Security Applications Conference*, 269–273.

Pierro, A. D.; Hankin, C.; and Wiklicky, H. 2002. On approximate non-interference. In *Proc. of Workshop on Issues in the Theory of Security*. IFIP WG1.7.

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

TCSEC. 1985. 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.

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

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