Cascade Vulnerability Problem Simulator Tool

Christian Servin and Martine Ceberio
Department of Computer Science
University of Texas at El Paso
El Paso, TX 79968-0518, USA
christians@miners.utep.edu

***Abstract*—In this paper, we present a tool called the *CVP Simulator Tool*, which enables users to model a computer network and its connectivity, and detects and eliminates any existing Cascade Vulnerability Problems (CVP). Users can add computers into a network, define what kind of information the computer can hold (e.g., Top Secret, Classified), establish connections between computers, assign a value to this connection, and simulate information flow between computers.**

The presented tool integrates the best features of two programming languages, object oriented and logic programming, to express solutions in terms of constraints using Constraint Programming (CP), which is a paradigm that has been successfully applied to large scale problems such as combinatorial search, optimization, and scheduling.

I. Introduction

In the discipline of computer security, the field of *trust management design* is dedicated to the design of trusted systems, in particular trusted networks. We use a common trusted mechanisim known as *multi-level security* (MLS), which allows simultaneous access to systems by users with different levels of security clearance in an interoperating network. For example, let us consider a simple scenario of a university composed of three departments: payroll, financial aid, and academic services. We know that the payroll department deals with sensitive information, such as social security numbers, birthdays, wages amounts, etc. The financial aid department may use information that the payroll owns. Similarly, the academic department communicates with the financial aid department. Vulnerability arises when an intruder takes advantage of the network connectivity and creates an inappropriate flow of information across the network, leading to the so-called Cascade Vulnerability Problem (CVP) [6].

Currently, there are vulnerability scanners (commercial and free software) that help network administrators detect vulnerabilities in a given network. Many banking and financial infrastructures, business companies, and university IT administrators use these tools to intuitively analyze and detect anomalies in their networks, e.g., statistics about how many people connect to a specific computer, intrusion detection systems, anti-virus/spyware detectors. Although there are many services and tools that detect vulnerabilities in a network, there are no tools that detect the CVP.

Over the last four years several approaches to solve this problem have been proposed; in particular those from Bistarelli et al. [1], [2], which model, detect, and properly eliminate the CVP in a network. These particular approaches express a solution of the problem using soft constraint programming. In [5] approach extended these approaches by incorporating real-world criteria in the connections between computers, such as the bandwidth, electricity, and cost of connections. Considering such features in CVP results in generating a constraint optimization problem.

We present the CVP Simulator Tool, which is a Java-based application that detects the existence of a cascade vulnerability problem in a network if it exists, identifies the elements that are generating this problem, and properly eliminates the minimum elements that cause this problem in the network. The objective of building this tool is to assess the work presented in the approaches mentioned above.

II. Background

A. A network and its representation

A network is a set of interconnected computers. A computer holds at least one information level that can interact with other computers. For example, let us consider a network of two computers $C1$ and $C2$. The computer $C1$ holds information at the *top secret* and *secret* level and computer $C2$ only owns information at the level *secret*. Computer $C1$ and $C2$ share information at the secret level like is as shown in Figure 1.

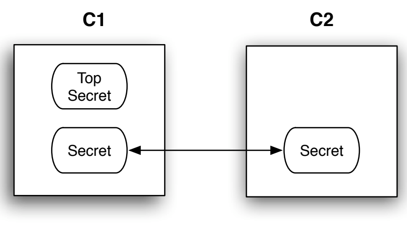


Fig. 1. A simple network composed of two computers, each computer holds various information levels.

B. The multi-level security (MLS) mechanism

A *Multi-level Security* (MLS) mechanism enables users with different levels of security clearance to simultaneously access computer systems. As a result, these security clearances prevent users from obtaining access to information for which they do not own authorization.

MLS permits users from a higher clearance level to access lower clearance information. For instance, a user with a clearance level *top secret* can access a *secret* level whereas in the same manner, a user with a *secret* level can access information that owns *classified* clearance level, but not vice-versa.

This MLS mechanism enforces a lattice-based security policy ℓ of security levels, which has ordering relation $\leq$. Given $x, y \in \ell$, $x \leq y$ means that information may flow from level x to y; e.g., C $\leq$ S $\leq$ T, where C is classified, S is secret, and T is top secret information.

C. The cascade vulnerability problem (CVP)

A CVP arises in approved-trusted networks. An approved network is a network such that every computer that belongs to it agrees to own a security assurance level. A CVP arises when an intruder takes advantage of the network connectivity to compromise information across a range of sensitivity levels, and the span of accessed levels exceeds the accreditation range of any computer. Let us illustrate a potential CVP in a MLS network in Figure 2.

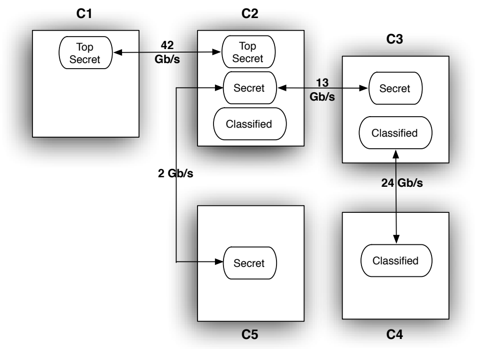


Fig. 2. A potential CVP. Information in higher clearance level (*top secret*) leaks to a lower clearance level (*classified*) through this MLS network.

D. Constraint Solving

Constraint Programming (CP) is a powerful programming paradigm that has been successfully applied in many applications such as optimization in operational research, logic circuits design in engineering, business applications, and protein folding in bioinformatics.

Constraints are relations between variables or unknowns with a corresponding range of values. We can illustrate this with an example: a student is driving late to school on the highway. There is a maximal speed limit of 60 miles per hour on the highway . The current time is 7:45 am and the class starts at 8:00 am. The person is 15 miles away from school, and he/she must arrive on time to class.

Every single sentence is either a fact or a constraint: $speed\_limit \leq 60$, $distance = 15$, and $remaining\_time = 15$. These constraints are defined by the relations between variables (speed limit, distance, current time, and class time). Likewise, every variable is included in a range of restricted values, e.g., the student must drive on the highway no more than 60 miles per hour.

The formulation of the problem is called a *constraint satisfaction problem (CSP).*

CSP. A Constraint Satisfaction Problem is a triple φ = *(V, D, C)*, where:

- $V = \{v_1, \ldots, v_n\}$ is a finite set of variable(s).
- $D = \{d_1, \ldots, d_n\}$ is a finite set of domain(s), where each domain is a set of values for the corresponding variable.
- $C = \{c_1, \ldots, c_n\}$ is a finite set of constraint(s) restricting the values that the variables $\{v_1, \ldots, v_n\}$ can simultaneously take.

1) Constraints in a network: The proposed approaches in [1], [2] introduced the notion of *risk (R)* and *effort (E)*. In order to determine the existence of such a cascading effect, we need to compare the effort required to compromise the network against the risk of compromising the system as a whole.

A cascading path can be identified as any path η where the risk associated with the path exceeds the effort to compromise it. In other words, it means that the following constraint is satisfied:

$$R\eta > E\eta$$

If is the case that this constraint is satisfied, which indeed the risk of compromising the system is greater than the effort required to compromised, means that there exists a cascading path. Then we do the following:

- extract the paths that are provoking this CVP using [1], [2],
- apply algorithms to eliminate the minimum number of elements that are cascading paths generators: these algorithms are the *Minimal Weighted Hitting Set* MWHS and the *Minimal Set Cover* MSC mentioned in [3], [4],
- returns the results.

III. CVP Simulator Tool

The objective of this tool is to detect the existence of a cascade vulnerability problem (if it exists) in a network, identify the connections that are generating the problem, and properly eliminate the minimum connections as possible that provoke this problem in the network. In order to achieve the best repair (i.e., cutting the minimal and least expensive connection), an optimization algorithm has been incorporated MWHS to eliminate the least expensive connections in the network.

This tool combines the best of two programing languages: Java and SWI-Prolog.

A. Object-oriented programming through Java

The tool takes advantage of Java's operating system portability, object oriented manipulation, and graphical libraries. We mainly take advantage of Java's portability because it enables us to run this tool on any computer as its slogan says "*Write once, run anywhere*"[1]. Also, the *message passing*

[1]Sun Microsystems: http://www.java.sun.com

between computers, which can be represented as objects in Java, enables us to take advantage of connectivity.

Finally we use Java to render the graphical interface to enable users to interact with this tool through a network simulation that displays the computers in the network and the weighted connections between them.

B. *Solving constraints with SWI-Prolog*

Constraint Solving is a generalization of Constraint Logic Programming (CLP), which extended Logic Programming, by embedding constraints (i.e., not longer predicates) into a logic program such as Prolog. Constraints bring important features such as logical variables and backtracking and other solving algorithms s.a. propagation. It is natural to use a constraint logic programming paradigm to describe the constraints and to solve our particular problem.

The constraint solver is a Prolog program that processes the constraints that are displayed in the interface, and returns a set of all possible solutions that satisfy the constraints.

C. *Features/Functionalities*

The main features and functionalities of the CVP simulator tool presents are:

- the creation of computer systems and networks,
- the creation of links between computers systems,
- the assignment of weights to the links,
- the detection of the existence of a CVP in a network,
- the extraction of the cascading path generators (η) from the network in case there is a CVP,
- the use of applied optimization algorithms,
- the identification of the least expensive connections to eliminate in a network.

IV. Architecture Design

The tool is an example of a *model-view-controller* (MVC) architectural pattern. The model is the representation of the algorithms used, e.g., algorithms that detect and eliminate the CVP. The viewer component are multiple GUIs that interact with the user, and the controller component handles the interaction between GUI and model. The MVC pattern is shown in Figure 3.

Figure 4 shows how the system components interact with each other.

V. The CVP Simulator Tool and its Components

- *Customize panel:* that enables a user to add a computer into the network and specify the type of sensitivity information it owns, e.g., *Top-Secret, Secret, Classified.*

- *Configuration panel:* enables the creations of connections between systems and assign a value (weight) to that connection.
- *Simulation panel:* shows the tracing of the algorithm in the network, and provides the possible solutions. The algorithm is selectable by the user (only two algorithms have been implemented in this first version).

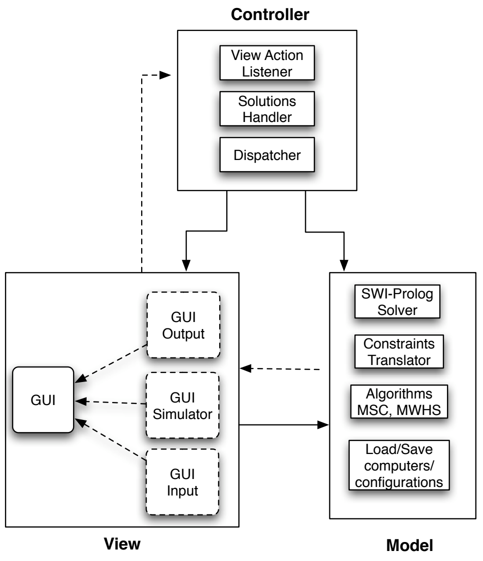


Fig. 3. The model-view-controller pattern for the CVP simulator tool

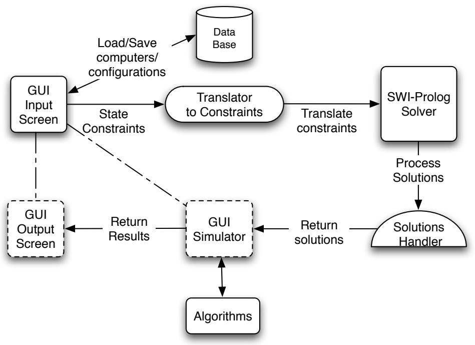
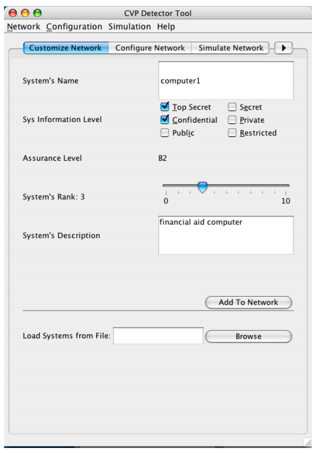


Fig. 4. The CVP simulator tool architecture design



System's Name: computer1

Sys Information Level: [✓ Top Secret] Secret [✓ Confidential] Private Public Restricted

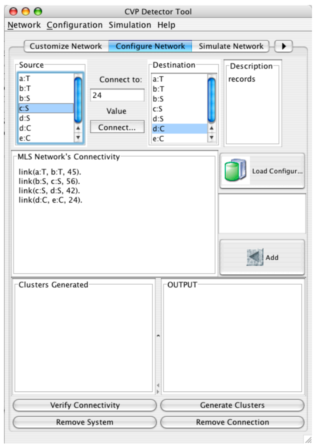
Assurance Level: B2

System's Rank: 3 (0 – 10)

System's Description: financial aid computer

Load Systems from File:

Fig. 5. Customize panel: enables adding a new computer to the network.



Source: a:T, b:T, b:S, c:S, d:S, d:C, e:C

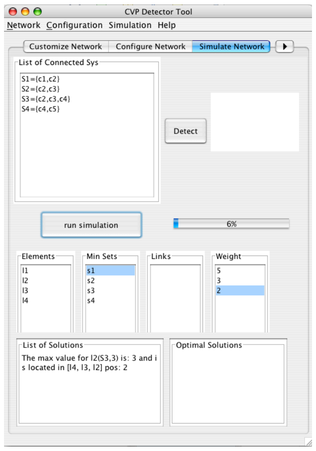
Connect to: 24

Destination: a:T, b:T, b:S, c:S, d:S, d:C, e:C

Description: records

```
link(a:T, b:T, 45).
link(b:S, c:S, 56).
link(c:S, d:S, 42).
link(d:C, e:C, 24).
```

Fig. 6. Configure panel: Allows to establish connections between computers.



```
S1={c1,c2}
S2={c2,c3}
S3={c2,c3,c4}
S4={c4,c5}
```

Elements	Min Sets	Links	Weight
l1	s1		5
l2	s2		3
l3	s3		2
l4	s4		

```
The max value for l2(S3,3) is: 3 and i
s located in [l4, l3, l2] pos: 2
```

Fig. 7. Simulate panel: simulates the specified algorithm to eliminate the CVP.

- *Constraint translator:* extracts the computer information that is added from the customize panel and the connections established in the configuration panel, and generates a SWI-Prolog file with the information extracted from these panels.
- *SWI-Prolog solver:* invokes SWI-Prolog and consults the file generated by the constraint translation component. This solver then processes the constraints to detect a CVP in the network. In the case that the detection constraints are satisfied (that indeed there exists a CVP in the network), the solver finds all the paths that are causing this cascading problem.
- *Solution handler:* opens a listener stream to the SWI-Prolog solver, and it stores all the output from the SWI-Prolog solver until it finishes to process all the constraints, then gives the results to the simulation panel.
- *Algorithm selection:* This component enables users to select the algorithm that will be applied in the elimination process of elements that generate the CVP. The chosen algorithm is used in the simulator and the results are displayed in the simulator panel. The algorithms in this first version of the application are implemented in Java, and their APIs are provided in case the user would like to incorporate another algorithm into the tool.

A. Constraint translator

This component is in charge of extracting the *facts* and *constraints* from a given network. Consider the network from Figure 1. The network is composed of 5 computer with different sensitiveness compartments:
C1, owns a *top-secret* compartment;
C2, owns a *top-secret, secret*, and *classified* compartments;
C3, owns a *secret* and *classified* compartments;
C4, owns a *classified* compartment;
C5, owns a *secret* component.
these facts are translated and represented in Prolog facts called **computer** as follows:

```
computer(c1, [ t ] ).
computer(c2, [ t, s, c ] ).
computer(c3, [ t, s ] ).
computer(c4, [ c ] ).
```

In the same manner, the facts that represent the links between computers:
C1 is connected to C2 through the top-secret compartment, and this connection has a value of 42 Gb/s,
C2 is connected to C3 through the secret compartment, and this connection has a value of 13 Gb/s, and to C5 with a 2 Gb/s value,
C3 is connected to C4 through the classified compartment, and this connection has a value of 24 Gb/s,
are also represented as follows:

```
link(c1 , [ [c2, t, 42] ]).
link(c2 , [ [c3, s, 13], [c5, s, 2] ]).
link(c3 , [ [c4, c, 24] ]).
```

In Prolog words: the rule **link** takes two objects, a *computer source*, i.e. $c2$, and an *list composed of lists* (an adjacent list) that represents all connections to computer source, i.e.,

$$[[c3, s, 13], [c5, s, 2]]$$

These facts are written in a Prolog file and then passed to the *Prolog solver* to solve the constraints.

B. Prolog solver

This solver is a Prolog file executed by SWI-Prolog that takes as an input the file generated in the previous section and returns a set of paths η that are the least expensive links in the network to eliminate in order to avoid a CVP.

The entire procedure of the Prolog solver is demonstrated in the *Simulation Panel*. In this panel step is shown step by step how the algorithm is working, and the resulting set is displayed in the *Result text area*.

VI. Conclusion and Future work

We presented a Java and Prolog-based application called CVP simulator tool that integrates two powerful programming languages paradigms: object-oriented programming (Java) and logic programming (SWI-Prolog). This tool enables users to simulate a computer network in an intuitive manner. In addition, this tool incorporates approaches proposed on [1], [2], and [5] to find a CVP (if it exists) in a network and to properly eliminate the elements that are generating this vulnerability.

The CVP simulator tool is highly maintainable, though there is room to incorporate additional optimization algorithms in solving the cascade vulnerability problem. We are currently working on a graphical representation of the network and its connectivity, and the proposed network according to the best optimized solution proposed by the solver. For more information about this tool please visit:

http://www.cs.utep.edu/students/christians/CVP/CVPTool.html

Acknowledgments

This work was supported by the Alliances for Graduate Education and the Professoriate (AGEP) grant HRD-0302788 and the Louis Strokes Alliance for Minority Participation (LSAMP) grant HRD-0217691 from the National Science Foundation (NSF).

The authors are thankful to Stefano Bistarelli and Eric Freudenthal for valuable suggestions in the development of this work.

References

[1] Stefano Bistarelli and Simon N. Foley. Detecting and eliminating the cascade vulnerability problem from multi-level security networks using soft constraints. In *Innovative Applications of Artificial Intelligence Conference*, pages 25–29. IAAI, July 2004.

[2] Stefano Bistarelli, B. O'Sullivan, and Simon N. Foley. Modelling and detecting the cascade vulnerabiliy problem using soft constraints. In *ACM Symposium on Applied Computing (SAC 2004)*, pages 14–17. ACM, March 2004.

[3] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*. Springer, third edition, 2006.

[4] Ronald L. Rivest and Charles E. Leiserson. *Introduction to Algorithms*. McGraw-Hill, Inc., New York, NY, USA, 1990.

[5] Christian Servin, Martine Ceberio, Eric Freudenthal, and Stefano Bistarelli. An optimization approach for the cascade vulnerability problem using soft constraints. In Marek Reformat and Michael R. Berthold, editors, *Proceedings of the 26th International Conference of the North American Fuzzy Information Processing Society NAFIPS'2007*, pages 372–377, San Diego, California, June 2007. IEEE, Press.

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