CONFIGURING STORAGE-AREA NETWORKS FOR MANDATORY SECURITY

Benjamin Aziz, Simon N. Foley, John Herbert, and Garret Swart

Abstract Storage-area networks are a popular and efficient way of building large storage systems both in an enterprise environment and for multi-domain storage service providers. In both environments the network and the storage has to be configured to ensure that the data is maintained securely and can be delivered efficiently. In this paper we describe a model of mandatory security for multi-domain storage services that is flexible enough to reflect the data requirements, tractable for the administrator, and implementable as part of an automatic configuration system. We describe the model abstractly, its implementation as part of a prototype SAN configuration system written in OPL, and illustrate its operation on a set of sample configurations.

1. INTRODUCTION

Storage-Area Networks (SANs) constitute an important element in modern IT infrastructures due to their efficient management of the underlying storage capabilities in environments shared by several servers, applications, users or even organisations. The purpose of any SAN service is to provide virtual disk services, called *datasets,* to its clients, typically file systems or database servers. Each dataset acts like a physical disk to the client applications. However, unlike a physical disk, which has a fixed set of properties that were set when the disk was designed, a dataset has a set of requirements that are specified when the dataset is created and may be changed over the life of the dataset. These properties may include current capacity, availability of the dataset service in the face of component failures, reliability of the data stored on the dataset in the face of media failures, and any real time performance requirements. These requirements may be in excess of what can be provided by any physical device, but it is up to the SAN to configure itself to meet the requirements.

Like any other multi-user system, a SAN also has security requirements. These can include data privacy; protecting data from unauthorized readers, data

integrity; protecting data from unauthorized updates, and additional privacy of the data traffic. In an enterprise configuration, this is often done by a combination of security kernels in the SAN devices themselves, as well as firewalls that restrict access of the SAN service to those client machines authorised to use the service. In such an environment, both the firewall and the SAN's own internal security system must be breached before unauthorized access can take place.

The kind of attack we are concerned with in this paper is an attack by an authorized user of the SAN. This is more likely to take place in a large enterprise SAN environment shared by a wide range of organizations, or in a service provider that is providing storage services to many different customers. In such an environment, the administrator cannot use the firewall to prevent the attack because it is initiated from a valid user of the SAN that must be allowed to use the service. The administrator must rely on the SAN's internal system only to determine which datasets can be accessed, and which ones cannot.

Using a single SAN system for storing multiple datasets that are to be accessed by many different authorized clients means that the implementation of the SAN's system has to be trusted to provide the correct semantics. This reliance on a single layer of software to provide data separation is a potential soft spot in the security of the system. One solution to this security problem, which we adopt in this paper, is to specify carefully the data separation requirements on the SAN. Once such requirements are specified, this will lead to a search for an acceptable SAN configuration that will maintain a low risk level associated with losing the privacy or integrity of the data being stored on the SAN.

In this paper, we propose a model of mandatory security for SANs. The primary contribution in this paper is the development of a framework that can be used to calculate the best multilevel secure configuration of a SAN. The 'best' configuration is based on tradeoffs that can be made on the basis of the potential risk of compromise of the components that are available to make up the SAN. Finding the best configuration is reduced to a constraint satisfaction problem within the framework.

We do not consider how underlying security services for SANs might be implemented [15], rather, we take a modelling approach and provide an abstract definition of what is meant by security in a SAN. We are concerned with threats from insiders, and in particular, the primary contribution in this paper is the development of a framework that can be used to determine a secure configuration of a SAN. In addition, the selection of a configuration is guided to be within some acceptable level of risk that is defined by the security administrator.

The remainder of the paper is structured as follows. § 2 reviews related work and a model of SANs is developed in §3. § 5 develops a model for a secure SAN configuration based on the security model described in §4. We outline an OPL §6 implementation of the algorithm used in searching for acceptable

SAN configurations. Finally, in §7, we conclude the work and give directions for future work.

2. RELATED WORK

In this paper we build on research from several different areas: security modelling, storage configuration, and constraint programming. Storage configuration has received much commercial attention in recent years and various systems for optimizing construction of SAN systems have been built and deployed. Industry working groups have been developing standards for interoperability in management and configuration and in describing storage requirements and device capabilities (see [5, 11, 13]). Automated configuration of storage systems has been explored by HP by [24,1], Simultaneous satisfaction of performance, capacity, cost and reliability requirements in RAID system configuration has been described by one of the authors [22].

We use techniques from constraint programming to state and to solve this configuration problem. Configuration problems have been attacked using constraint programming for several years [19, 7] and people have solved these problems both by using generic tools and developing specialized tools. We use OPL [12], because its simple logic based style makes it easy to experiment with different security models and then to speed up the solution using domain specific information. It is likely that using more specialized configuration based solvers will be more efficient at solving large instances of this type of problem.

The goal of this paper is to develop a model of a secure SAN that ensures that datasets are properly separated, allocated and managed across the SAN. There is a resemblance between this problem and the Chinese Wall security policy [4]: different datasets must be allocated across a SAN in such a way that their storage and access using the underlying SAN components does not give rise to conflicts. We use a label-based model [6,9] to represent this type of security in SANs. While conceptually simple, lattice/label-based models can be used to characterize mechanisms that support a wide range of mandatory security requirements, including multilevel security [2, 23, 9], Chinese Walls [8, 17, 21], separation of duty [9], well-formed transactions [9, 16], and Role Based Access Controls.

Existing research that considers the automatic configuration of systems to meet security goals includes [14, 18]. In [18], Millen adapts Meadows lattice-based Chinese Wall mechanism [17] to determine optimal configurations that can survive component failures. [14] considers the configuration of multilevel secure systems to support multilevel secure workflows.

3. STORAGE-AREA NETWORKS

The structure of a SAN is illustrated in Figure 1. This structure is described in the following paragraphs.

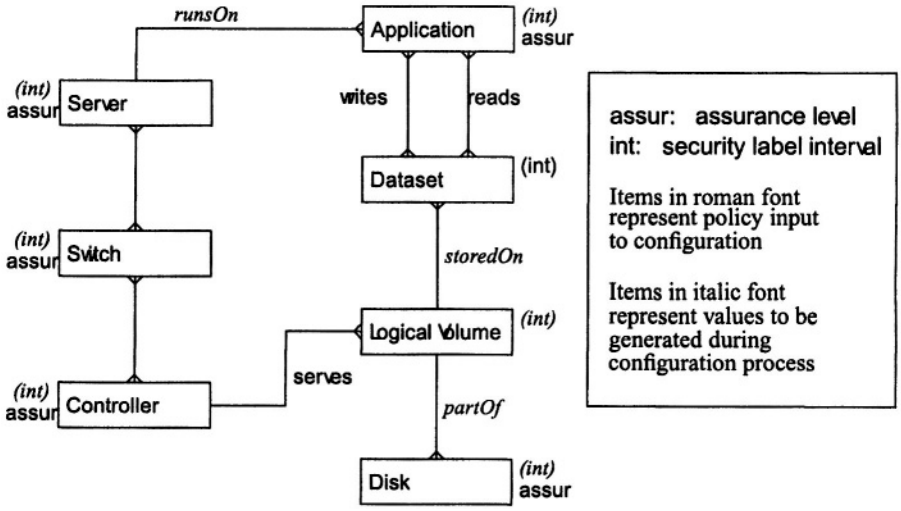

Figure 1. The Structure of a SAN.

Disks. These are the physical storage units that may include tapes, hard disks, optical and solid state devices. We write the set of all disks as $DISK = \{disk, disk', \ldots\}$.

Disks Controllers. In general, disks may either be a direct part of the SAN, or more typically, they are packaged as part of a storage array that is controlled with one or more controllers. Each controller has a direct access to the array and, in general, the controller will retain a few spare disks inside the array for use in the event of a failure. We refer to the set of disk controllers as $CONTROLLER = \{cntr, cntr', \ldots\}$.

Logical Volumes. A controller usually builds one or more large virtual disks, which we call logical volumes, out of the physical storage units that it is connected to using RAID techniques [20]. For availability reasons, logical volumes are not typically tied to a single controller and the failure of the primary controller will trigger its back-up to start and serve the logical volumes.

We define the set of logical volumes as $LV = \{lv, lv', \ldots\}$. Furthermore, we define the following two functions, which relate logical volumes to disks

and controllers, respectively:

$$partOf : DISK \rightarrow LV$$
$$serves : CONTROLLER \rightarrow \wp(LV)$$

Hence, $partOf(disk) = lv$ denotes that *disk* is part of the logical volume lv, and $serves(cntr) = \{lv_1, \ldots, lv_n\}$ denotes that logical volumes $lv_1, \ldots, lv_n$ are served by controller *cntr*.

Datasets. A dataset is a collection of related data files that are accessed by applications as a unit. Controllers implement the datasets by constructing RAID-based logical volumes with the appropriate properties out of a set of identical physical disks and partitioning them to datasets of the appropriate size. Examples of datasets include the set of web files comprising a website, the set of files holding a database and a user's email files. We write the set of datasets as $DATASET = \{data, data', \ldots\}$.

The following function relates datasets to logical volumes:

$$storedOn : DATASET \rightarrow LV$$

such that $storedOn(data) = lv$ signifies the fact that the dataset, *data,* is stored on the logical unit, lv.

Applications, Streams, and Servers. Applications, which constitute a set, $APP = \{app, app', \ldots\}$, are active entities that read and write information stored in one or more datasets. In effect, datasets are regarded as virtual disks that are accessed by applications using *streams.* A stream is regarded as a triple:

$$(app, op, data) \in STREAM$$

where an application, *app,* accesses some dataset, *data,* using operation, $op \subseteq \{R, W\}$, which is a subset of the *R*ead and *W*rite capabilities.

At any one time, an application runs on a particular *application server,* which in turn, it may be running more than one application. We write the set of application servers as $SERVER = \{srv, srv', \ldots\}$, and we define the following function, relating applications to their application servers:

$$runsOn : APP \rightarrow SERVER$$

such that $runsOn(app) = srv$ expresses the fact that application, *app,* is currently running on the application server, *srv.*

Switches. SAN switches, much like switches in a LAN, are used to connect the components of a SAN (i.e. its controllers, other switches and application servers) and to connect a SAN to a set of LANs. For a SAN application to

be able to access a SAN, the application must either be connected to the LAN which is connected to the SAN, or it must be directly connected to the SAN via a network controller specific to the SAN fabric being used.

Assuming that $DEVICE = CONTROLLER \cup SWITCH \cup SERVER$ is the set of devices of a SAN, ranged over by $dev, dev', \ldots$, then we can define the following function:

$$connects : SWITCH \rightarrow \wp(DEVICE)$$

such that $connects(swt) = \{dev_1, \ldots, dev_n\}$ denotes the fact that *swt* currently connects the devices, $dev_1, \ldots, dev_n$.

4. A LABEL-BASED SECURITY MODEL

In this section we propose a basic mandatory label-based security model that will be used in the next section to characterize security in SANs. The model is based on the Bell-La Padula model with partially trusted subjects [6, 9]. It has been shown that a wide variety of application security policies can be encoded in terms of label-based policies.

A *label-based security policy* is a lattice of security labels (classifications), *SC,* with partial ordering, $\leq$, and the least-upper and greatest-lower bound operators, $\sqcup, \sqcap$, respectively. An example is the multilevel security policy with labels, $SC = \{unclassified, secret, topsecret\}$.

Let *ENTITIES* represent the set of all components that can source and/or sink information (disks, controllers, applications, etc.). Every entity, e, is bound to an interval of the policy lattice, where $int(e) = (x, y) \in SC \times SC$, and $x \leq y$, is interpreted to mean that entity e may sink information at class y or lower and may source information at class x or higher. We also write $int(e) = [int_\perp(e), int_\top(e)]$.

If entity e is a subject (in the traditional sense) then $int(e) = [x, y]$ corresponds to a partially trusted subject that may view/read information at class y and lower (vmax) and may write/alter information at class x and higher (amin). Conventional objects may be interpreted within this model as entities that are bound to an interval $[x, x]$ with a single level. Informally, $int(e) = [x, y]$ means that an entity, e, can be trusted to properly manage multilevel information within the security interval, $[x, y]$.

Within this model, the definition of a secure system is simply a generalization of the Bell-La Padula axioms (the simple security condition and star property). A system is secure if for all entities, *A* and *B,* such that information can flow from *A* to *B* then $int_\perp(A) \leq int_\top(B)$ holds.

EXAMPLE 1 A SAN is to be configured to manage *IBM, HP* and *Exxon* information. A security policy is defined by the powerset lattice of these values with ordering relation $\subseteq$. An application server that is being used to securely

process IBM and Exxon data has a security interval $[\{\}, \{IBM, Exxon\}]$ and an IBM dataset has security interval $[\{IBM\}, \{IBM\}]$. △

An *assurance level* expresses the level of confidence in the capability of an entity to properly meet its security requirements. An assurance policy is a lattice of assurance levels, A, with partial ordering, $\leq$. The assurance of a device may depend on the complexity of the device's function, the amount of testing that has been applied to the device, the frequency of use, and the development methodology that has been used. For example, $A1 > B3 > B2 > B1 > ...$ is the lattice of assurance ratings from [23]. Every entity, e, has an associated assurance rating, $assur(e) \in A$. An off-the-shelf physical disk might have a low assurance rating; a multilevel secure application system that has been formally evaluated might have a high assurance rating, while a less formally developed embedded device with very limited functionality might also have a high assurance rating.

The relationship between assurance levels and security intervals can provide an indication of how much an entity with the given assurance level should be relied upon. A low-assurance off-the-shelf physical disk configured with security interval $[\{IBM\}, \{IBM\}]$ can be relied upon to manage single-level IBM data. However, the same disk should not be configured with an interval $[\{IBM\}, \{IBM, HP\}]$ as there is not sufficient assurance that it will reliably manage/separate the data between these competing organizations. On the other hand, an off-the-shelf disk configured with the interval $[\{\}, \{IBM, Exxon\}]$ is sufficient to manage/separate the non-competing IBM and Exxon data.

We use a *risk function* to quantify the relationship between assurance levels and security intervals. Given an assurance level, a, and a security interval, $[x, y]$, then the risk that the entity with assurance level a can be compromised is defined as $risk([x, y], a) \in \mathbb{N}$. Note that, for simplicity, we quantify risk by a natural number. The values for this function are specified as part of the requirements for the configuration by the security administrator.

EXAMPLE 2 Consider the security policy from the previous example and an assurance lattice, $lo \leq hi$. There is a low security risk to using an off-the-shelf disk for *single* level data and thus the risk is specified as:

$$\begin{aligned} risk([\{IBM\}, \{IBM\}], lo) &= 1 \\ risk([\{HP\}, \{HP\}], lo) &= 1 \\ risk([\{Exxon\}, \{Exxon\}], lo) &= 1 \end{aligned}$$

There is a high security risk when using the same disk to manage/store multi-level data from competing organizations:

$$\begin{aligned} risk([\{\}, \{IBM, HP\}], lo) &= 40 \\ risk([\{\}, \{IBM, HP, Exxon\}], lo) &= 40 \end{aligned}$$

However, there is less of a risk using the off-the-shelf disk to store multilevel data from non-competing organizations:

$$\begin{aligned} risk([\{\},\{IBM,Exxon\}],lo) &= 10 \\ risk([\{\},\{HP,Exxon\}],lo) &= 10 \end{aligned}$$

If we assume that a specialized high assurance disk will properly manage/partition data at different security classifications, then there is less risk when using this disk to manage data from competing organizations:

$$\begin{aligned} risk([\{\},\{IBM,HP\}],hi) &= 10 \\ risk([\{\},\{IBM,HP,Exxon\}],hi) &= 10 \end{aligned}$$

△

In providing a relationship between assurance levels and security intervals, the risk function provides a novel approach to characterizing aggregation problems. This contrasts with the lattice based strategies for Chinese Walls that are described in [8, 17, 21] which can be thought as defining an (acceptable aggregation or not) binary risk relation.

5. CONFIGURING SECURE SANs

A *secure SAN* is a SAN extended with the label-based security model. In the context of our security model, configuring a SAN means searching for a configuration of the SAN devices that meets the specified security policy, the applications' data requirements and any service level agreement (SLA) that may have been agreed with the customers of the data, and that has the least amount of risk possible.

Before the configuration process commences we require the following.

- The security policy, that is, the lattice of security classes, *SC,* and the risk function, *risk.*
- The application requirements, that is, the security point intervals of all datasets, *int*(*data*), and the set of streams that relate each application to the datasets that it reads and writes.
- A set of risk limitations in the form of a security class and a maximum risk threshold. These limitations correspond to a customer requirements for an upper bound on the risk in storing one a particular security point class.
- The device specifications, that is, the set of servers, controllers, switches and disks that the SAN is to be configured from. For each such device, e, we need its assurance level, $assur(e)$.

Solving the configuration problem will result in finding values for the *partOf, serves, storedOn, runsOn* and *connects* functions that define a particular instance of a SAN system.

5.1 Defining Security Intervals

A dataset, *data,* is initially assigned a point interval, $int(data) = [x, x]$, representing the sensitivity of that dataset. This is a reasonable assumption as datasets are passive entities that can only be manipulated and will never themselves manipulate other datasets. Using the point intervals of a set of datasets, it is possible to directly compute the interval of an application that will access those datasets by means of streams:

$int(app) = [(\sqcap\, set_\perp), (\sqcup\, set_\top)]$, where,
$set_\perp = \{int_\perp(data) \mid (app, (op, Attr), data) \in STREAM \wedge op \supseteq \{W\}\}$
$set_\top = \{int_\top(data') \mid (app, (op, Attr), data') \in STREAM \wedge op \supseteq \{R\}\}$

Now, for a particular setting of the *storedOn* function, we can define the security intervals of the logical volumes:

$int(lv) = [(\sqcap\, set_\perp), (\sqcup\, set_\top)]$, where,
$set_\perp = \{int_\perp(data) \mid data \in DATASET \wedge storedOn(data) = lv\}$
$set_\top = \{int_\top(data) \mid data \in DATASET \wedge storedOn(data) = lv\}$

From the security intervals of logical volumes and given a particular definition of the *serves* function, we can define the security intervals of controllers:

$int(ctr) = [(\sqcap\, set_\perp), (\sqcup\, set_\top)]$ where,
$set_\perp = \{int_\perp(lv) \mid lv \in LV \wedge lv \in serves(ctr)\}$
$set_\top = \{int_\top(lv) \mid lv \in LV \wedge lv \in serves(ctr)\}$

Similarly, security intervals of disks may be defined based on the security intervals of the logical volumes and a definition of the *partOf* function:

$int(disk) = int(partOf(disk))$

On the other hand, the security interval of an application server is defined based on the security intervals of its applications running and a definition of the *runsOn* function:

$int(srv) = [(\sqcap\, set_\perp), (\sqcup\, set_\top)]$ where,
$set_\perp = \{int_\perp(app) \mid app \in APP \wedge runsOn(app) = srv\}$
$set_\top = \{int_\top(app) \mid app \in APP \wedge runsOn(app) = srv\}$

Finally, intervals of switches are computed from intervals of the devices they connect (i.e. other switches, controllers, disks and application servers), given

a definition of the *connects* function:

$$
\begin{aligned}
&int(swt) = [(\sqcap\, set_{\perp}), (\sqcup\, set_{\top})] \text{ where,}\\
&set_{\perp} = \mu\, swt'.\\
&\quad (\{int_{\perp}(srv) \mid srv \in SERVER \wedge srv \in connects(swt)\} \cup\\
&\quad \{int_{\perp}(ctr) \mid srv \in CONTROLLER \wedge ctr \in connects(swt)\} \cup\\
&\quad \{int_{\perp}(swt') \mid swt' \in SWITCH \wedge swt' \in connects(swt)\})\\
&set_{\top} = \mu\, swt'.\\
&\quad (\{int_{\top}(srv) \mid srv \in SERVER \wedge srv \in connects(swt)\} \cup\\
&\quad \{int_{\top}(ctr) \mid srv \in CONTROLLER \wedge ctr \in connects(swt)\} \cup\\
&\quad \{int_{\top}(swt') \mid swt' \in SWITCH \wedge swt' \in connects(swt)\})
\end{aligned}
$$

The usage of the least-fixed point operator, μ, is required since the definition of *int*(*swt*) is recursive.

EXAMPLE 3 Given the following dataset intervals:

$$
\begin{aligned}
&int(data_1) = [\{IBM, Exxon, foo\}, \{IBM, Exxon, foo\}]\\
&int(data_2) = [\{IBM, foo\}, \{IBM, foo\}]\\
&int(data_3) = [\{foo\}, \{foo\}],
\end{aligned}
$$

and the following streams

$$
\begin{aligned}
&(app, (\{R\}, Attr_1), data_1)\\
&(app, (\{R, W\}, Attr_2), data_2)\\
&(app, (\{W\}, Attr_3), data_3),
\end{aligned}
$$

then applications classified with intervals $[\{foo\}, \{IBM, Exxon, foo\}]$ can handle the above data using the streams indicated. △

5.2 Optimal Configurations

After defining the security intervals, $int(e)$, of every SAN entity, e, as in the previous section, and given that entities have fixed assurance levels, $assur(e)$, then we can optimize he definitions of *partOf, serves, storedOn* and *connects.* This can be formalized by saying that we are looking for the choice of these functions that minimizes:

$$\sum_{\forall e \in ENTITIES} risk(int(e), assur(e)) \qquad (1)$$

Individual customers may insist on a Service Level Agreement (SLA) that limits the risk that their data is compromised. For example, a customer may pro-

vide a security interval int_{SLA} and a limit ν_{SLA} and require that

$$\sum_{\forall e \in ENTITIES:\ int(e) \cap int_{SLA} \neq \{\}} risk(int(e), assur(e)) <= \nu_{SLA} \quad (2)$$

The SLA assures the customer that the risk in the configuration for storing their data is low enough.

6. OPL IMPLEMENTATION

To test our understanding of this security and configuration model and to test its usefulness, we implemented the model and used it to generate the lowest risk configuration that meets the requirements. We decided to use OPL for the implementation language because of its built-in logic and search capabilities.

An OPL program consists of five pieces:

- Input Data Model: Describes all the data that must be supplied to define a particular instance of the problem to be solved. The input data is generally validated to make sure that the request is not obviously inconsistent.
- Variable Data Model: Describes the data that the program is to determine values for.
- Constraints: A set of relations that must hold between the variables and the input data. The number of these constraints can depend on the input provided. If all the constraints hold for a particular assignment to the variable data, that assignment is called a feasible solution.
- Objective function: A function that is maximized or minimized from among the feasible solutions. OPL reports new maxima or minima as they are determined during the search process, finally terminating when the search space of variable data has been exhausted.
- Search procedure: An optional plan for how to find the optimal solution. Typically this involves carefully choosing the order in which the variable data is examined and noticing when further changes will be ineffective.

The OPL input data consists of an instantiation of the Input Data Model. The output of the OPL program is a sequence of successively improving feasible solutions.

For this application, the Input Data Model is used to represent all needed input: the security policy, the application requirements, any SLA requirements, and the device specifications. We do validation of the input data to ensure that the security class forms a lattice and that the risk function is consistent with the lattice, and also to ensure that any static requirements, e.g. requirements on the applications themselves, are met.

The Variable Data Model is used to represent the interval for each device and the SAN configuration functions, that is, *storedOn, runsOn, serves, partOf* and *connects.* In the worst case, finding the optimal configuration means examining every combination of values in the Variable Data Model, so it is very important to make sure that there is a minimum of redundancy or over specification in the model.

The constraints fall into several categories:

- Device interval constraints implement the formulas defined in the previous section.
- Configuration consistency constraints make sure that the configuration meets the basic requirements, for example, that each logical volume is assigned enough disks to store the assigned datasets, that servers and controllers are all connected to switches, and switches to each other.
- Canonicalization constraints prune all but one equivalent configuration from the configuration space. This is important for reducing the search space as discussed in [10].
- SLA constraints ensure that the risk for a particular security class is limited to the agreed value.

The final piece of the OPL program is the search method. In this case it simply makes judicious choices about which part of the variable data space to explore first. The primary issue in the search is to make sure that the intervals are evaluated once the needed bits of the configuration have been generated. Quick elimination of infeasible or less optimal alternatives is the key to a fast running OPL application.

7. CONCLUSION AND FUTURE WORK

In this paper, we presented a model of mandatory security for SANs. The primary contribution in this paper is the development of a framework that can be used to calculate the lowest risk multilevel secure configuration of a SAN. The risk function is configured to reflect the probability of leakage and the cost of the consequences of the leakage The 'best' configuration is determined by searching among the space of all valid SAN configurations for the one with the minimum aggregate risk. We also outlined an implementation of the configuration search in OPL.

A label-based model is used to represent security in SANs. While conceptually simple, lattice/label-based models can be used to characterize mechanisms that support a wide range of mandatory security requirements [2, 23, 9, 8, 16, 17, 21]. Therefore, we conjecture that the results in this paper can be use-

fully applied to other more specific mandatory protection models such as Role Based Access Control; this is a topic of ongoing research.

The SAN security model extends the dual-label/partially trusted subject lattice model with the addition of a risk function. This function is used to encode the level of risk associated with storing/managing combinations of information on entities evaluated to certain degrees of assurance. This is more flexible than the conventional assurance/evaluation criteria approach [23]; the risk function is used to guide the generation of a secure configuration within an acceptable degree/measure of risk. However, like conventional evaluation criteria, there is the potential for cascading channels [23] within the space of secure configurations. We are currently exploring how constraint-based techniques for removing channel cascades [3] can be used to reduce the space of suitable configurations.

Acknowledgments

We are grateful for helpful feedback from the anonymous referees. This work is supported by the Boole Centre for Research in Informatics, University College Cork under the HEA-PRTLI scheme and from Science Foundation Ireland under Grant 00/PI.1/C075.

References

[1] Eric Anderson, Michael Hobbs, Kimberly Keeton, Susan Spence, Mustafa Uysal, and Alistair C. Veitch. Hippodrome: Running circles around storage administration. In Darrell D. E. Long, editor, *Proceedings of the FAST'02 Conference on File and Storage Technologies,* pages 175–188, Monterey, California, USA, January 2002. USENIX.

[2] D.E. Bell and L.J. La Padula. Secure computer systems: Unified exposition and multics interpretation. Technical Report ESD–TR–75–306, Mitre Corporation, July 1975.

[3] Stefano Bistarelli, Simon N. Foley, and Barry O'Sullivan. Modelling and detecting the cascade vulnerability problem using soft constraints. In *Proceedings of the ACM Symposium on Applied Computing,* Nicosia, Cyprus, March 2004. ACM Press.

[4] D.F.C. Brewer and M.J. Nash. The Chinese wall security policy. In *Proceedings of the* 1989 *IEEE Symposium on Security and Privacy,* pages 206–214, Oakland, California, USA, May 1989. IEEE Computer Society Press.

[5] Inc. Distributed Management Task Force. Web-based enterprise management (wbem) initiative. http://www.dmtf.org/standards/wbem.

[6] M. Branstad et al. Trusted mach design issues. In *Proceedings of the* 3rd *AIAA/ASIS/DODCI Aerospace Computer Security Conference,* Orlando, Florida, USA, December 1987. IEEE Press.

[7] Gerhard Fleischanderl, Gerhard E. Friedrich, Alois Haselbeck, Herwig Schreiner, and Markus Stumptner. Configuring large systems using generative constraint satisfaction. *IEEE Intelligent Systems,* 13(4):59–68, July 1998.

[8] Simon N. Foley. Aggregation and separation as noninterference properties. *Journal of Computer Security,* 1(2):159–188, 1992.

[9] Simon N. Foley. The specification and implementation of commercial security requirements including dynamic segregation of duties. In *Proceedings of the 4th ACM Conference on Computer and Communications Security,* pages 125–134, Zurich, Switzerland, April 1997. ACM Press.

[10] Eugene Freuder. Eliminating interchangeable values in constraint satisfaction problems. In *Proceedings of the 9th National Conference on Artificial Intelligence,* volume 1, pages 227–233, California, USA, July 1991. MIT Press.

[11] Garth A. Gibson, Jeffrey Scott Vitter, and John Wilkes. Strategic directions in storage i/o issues in large-scale computing. *ACM Computing Surveys,* 28(4):779–793, 1996.

[12] Pascal Van Hentenryck. *The OPL Optimization Programming Language.* MIT Press, Cambridge, Massachusetts, USA, January 1999.

[13] Tim Howes and Darrel Thomas. Gaining control of complexity: The standard for the data center. Technical report, DCML: Data Center Markup Language, Sussex, UK, 2003. http://www.dcml.org/pdf/DCML_tech_whitepaper.pdf.

[14] M.H. Kang et al. A strategy for an mls workflow management system. In *Proceedings of the 13th Annual IFIP WG11.3 Working Conference on Database Security,* 1999.

[15] Yongdae Kim, Maithili Narasimha, Fabio Maino, and Gene Tsudik. Secure group services for storage area networks. In *Proceedings of the First International IEEE Security in Storage Workshop,* pages 80–93, Greenbelt, Maryland, USA, December 2002. IEEE Computer Society.

[16] T.M.P. Lee. Using mandatory integrity to enforce 'commerical' security. In *Proceedings of the Symposium on Security and Privacy,* pages 140–146, 1988.

[17] Catherine Meadows. Extending the brewer-nash model to a multilevel context. In *Proceedings of the IEEE Symposium on Security and Privacy,* pages 95–102, Oakland, California, USA, May 1990. IEEE Computer Society Press.

[18] J.K. Millen. Local reconfiguration policies. In *Proceedings of the Symposium on Security and Privacy,* 1999.

[19] Sanjay Mittal and Felix Frayman. Towards a generic model of configuration. In N. S. Sridharan, editor, *Proceedings of 11th International Joint Conference on Artificial Intelligence,* pages 1395–1401, Detroit, Michigan, USA, August 1989.

[20] D.A. Patterson, G.A. Gibson, and R.H. Katz. A case for redundant arrays of inexpensive disks (raid). In *Proceedings of the* 1988 *ACM Conference on the Management of Data (SIGMOD),* pages 109–116, Chicago, USA, June 1988.

[21] Ravi S. Sandhu. Lattice-based access control models. *Computer,* 26(11):9–19, November 1993.

[22] Garret Swart. Storage management by constraint satisfaction. In *Proceedings of the Workshop on Immediate Applications of Constraint Programming,* Kinsale, Cork, Ireland, September 2003.

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

[24] John Wilkes, Richard A. Golding, Carl Staelin, and Tim Sullivan. The hp autoraid hierarchical storage system. In *Proceedings of the 15th ACM Symposium on Operating System Principles,* pages 96–108, Copper Mountain Resort, Colorado, USA, December 1995. ACM Press.