Journal of Computer Security 17 (2009) 191–210
DOI 10.3233/JCS-2009-0337
IOS Press

Configuring storage-area networks using mandatory security

Benjamin Aziz [a,*], Simon N. Foley [b], John Herbert [b] and Garret Swart [c]
[a] *e-Science Centre, Science and Technology Facilities Council, Didcot OX11 0QX, UK*
E-mail: b.aziz@rl.ac.uk
[b] *Department of Computer Science, University College Cork, Cork, Ireland*
E-mails: {s.foley, herbert}@cs.ucc.ie
[c] *Oracle Corporation, Redwood Shores, CA 94065, USA*
E-mail: garret.swart@oracle.com

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 SAN services that incorporates the notion of risk as a measure of the robustness of the SAN's configuration and that formally defines a vulnerability common in systems with mandatory security, i.e. cascaded threats. Our abstract SAN model is flexible enough to reflect the data requirements, tractable for the administrator, and can be implemented as part of an automatic configuration system. The implementation is given as part of a prototype written in OPL.

Keywords: Storage-area networks (SAN), formal validation methods, security, configuration analysis

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 storage services, called a *datasets*, to its clients, typically file systems or database servers. Each dataset is constructed from physical disks but 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 these properties may change over the life of the dataset. These properties may include current capacity, availability 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 could be in excess of what is provided by a physical device, however, it is up

[*] Corresponding author. Tel.: +44 1235 778840; Fax: +44 1235 445945.

to the administrators to configure the SAN with enough resources so that datasets meeting the requirements can be provisioned.

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. One approach to data privacy is to encrypt the data before the application writes it to the dataset. This provides very good privacy but does not help with integrity or traffic analysis attacks. Another, more typical approach yielding all three security properties, is to secure the devices used to provide the SAN service. 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 security threat that we are concerned with in this paper is an attack by an *authorized* user of the SAN. It is more likely to take place in a large enterprise SAN environment that is shared by a wide range of organizations, or in a service provider that is providing storage services to many different (and possibly competing) 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 configuration of the SAN's internal security mechanisms only to determine which datasets can be accessed, and which ones cannot.

Using a single SAN system for storing multiple datasets, accessed by many different authorized clients, means that the implementation of the SAN must 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 devices making up 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 carrying out an attack, an authorized user attempts to 'copy' data from one dataset to another, in violation of the data-separation/security policy. The attack has consequence for both confidentiality (leaking of dataset) and integrity (unauthorized modification of dataset). The attack may be intentional, where the user deliberately attempts to violate the data-separation policy, or unintentional, where the software that the user executes contains malicious code that attempts to violate the policy unknown to the user. This data-separation security policy is not unlike the requirements for a Chinese Wall security policy [5], and the model that we develop in this paper also provides a novel risk-based interpretation for Chinese Walls.

In this paper we do not consider how underlying security services for SANs might be implemented [13], rather, we take a modeling approach and provide an abstract definition of what is meant by (data-separation) 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 configuration is chosen so as to be within an acceptable level of risk as specified by the security administrator.

The rest of the paper is structured as follows. In Section 2, we define a simple model of SANs. In Section 3, we define the notion of a secure SAN; essentially a SAN equipped with generalized form of mandatory security. In Section 4, we define a configuration analysis for our secure SANs that aims at minimising the risk level and in Section 5, we discuss the issue of cascaded vulnerabilities in secure SANs. In Section 6, we define our notion of an optimal configuration and in Section 7, we review an implementation of the configuration analysis in OPL and finally, in Section 8, we conclude the paper and discuss directions for future work.

2. Storage-area networks

According to the model of SANs defined in [1,19], a SAN is composed from the following elements, as illustrated in Fig. 1, where the terminal connectors express many–one and many–many relations among the different elements of the model. These elements are described in the following paragraphs.

2.1. Disks

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

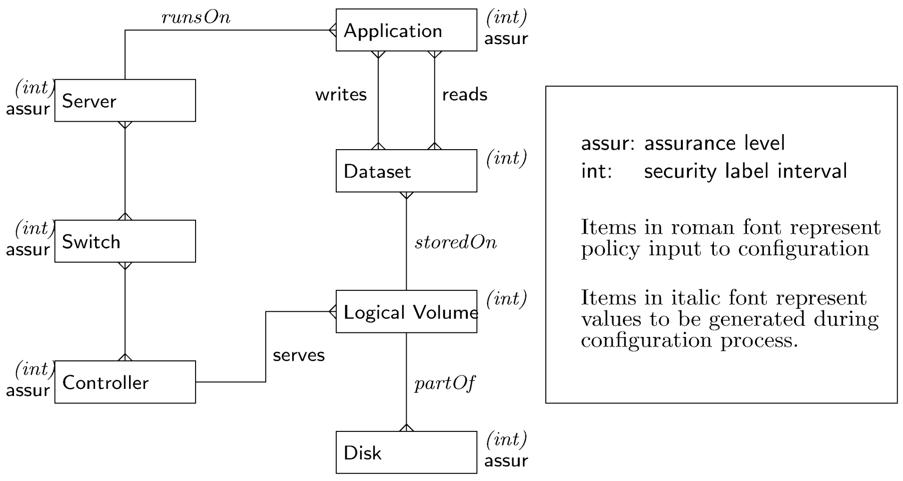


Fig. 1. The structure of a SAN.

2.2. *Disk controllers*

Usually, disks are arranged into arrays which are connected to one or more controllers. Each controller has a direct access to its attached arrays 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 $\mathit{CNTR} = \{ctr, ctr', \ldots\}$.

2.3. *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 [16]. 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 controllers to start serving the logical volumes it was serving.

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

$$\mathit{partOf} : \mathit{DISK} \rightarrow \mathit{LV},$$

$$\mathit{serves} : \mathit{CNTR} \rightarrow \wp(\mathit{LV}).$$

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

2.4. *Datasets*[1]

In our simplified model, 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 then 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 $\mathit{DATASET} = \{data, data', \ldots\}$.

The following function relates datasets to logical volumes:

$$\mathit{storedOn} : \mathit{DATASET} \rightarrow \mathit{LV}$$

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

[1] In SANS, these are more specifically called Logical Units or LUNs.

2.5. 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 Read and Write capabilities.

At any one time, an application runs on a particular *application server*, which 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*.

2.6. Switches

SAN switches, much like switches in a Local Area Network (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 must be directly connected to the SAN via a network controller specific to the SAN fabric being used.

Assuming that $DEVICE = CNTR \cup SWITCH \cup SERVER$ 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, $dev_1, \ldots, dev_n$. It is also possible to refer to the transitive closure of *connects* as $connects^*$.

3. A mandatory security model

In this section we propose a basic mandatory security model that will be used later to characterize security in SANs. The model builds on and extends the label-based model of Multilevel Security with partially trusted subjects [2,4,7]. It has been shown that a wide variety of application security policies can be encoded in terms of label-based policies [10].

3.1. Label-based security policies

A *label-based security policy* is a lattice of security labels (classifications), *SC*, with partial ordering, $\leqslant$, and the least-upper and greatest-lower bound operators, $\sqcup, \sqcap$, respectively. An example is the multilevel security policy with labels, $SC = \{unclassified, secret, topsecret\}$. In this paper, we shall adopt the lattice resulting from the powerset of the set of all organisation names ordered by subset inclusion with the empty set and the full set being the bottom and top elements, respectively. More formally, assume that $ORG = \{org_1, org_2, \ldots\}$ is the set of organisations, $\wp(ORG)$ is the powerset of *ORG*, then our security lattice is defined in the standard manner as:

$$(\wp(ORG), \subseteq, \cap, \cup, \emptyset, ORG).$$

Hence, $\forall x, y \in SC$: $x \leqslant y \iff x \subseteq y$. Intuitively, this means that label y is more important than label x if the set of organisations associated to y is a superset of that associated to x. This association could have different semantics depending on the type of entities x and y are being used to label. In what follows, we are interested mainly in *read/write* semantics.

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 $x \leqslant y$, 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 $int(e) = [int_\bot(e), int_\top(e)]$.

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 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 $int_\bot(A) \leqslant int_\top(B)$ holds. Other possible properties such as the integrity property can also be modeled using, for example, the Biba integrity axioms (where the flow of information is dictated by $int_\top(B) \leqslant int_\bot(A)$). In this paper, we shall only deal with the Bell–La Padula axioms.

Example 1. A SAN is to be configured to manage *IBM*, *HP* and *Exxon* information. The powerset lattice resulting from the set $\{IBM, HP, Exxon\}$ is shown in Fig. 2. According to our labeling policy, an application server that is being used to securely process IBM and Exxon data will be assigned the security interval $[\{\}, \{IBM, Exxon\}]$ and a dataset belonging to IBM will have the security interval $[\{IBM\}, \{IBM\}]$.

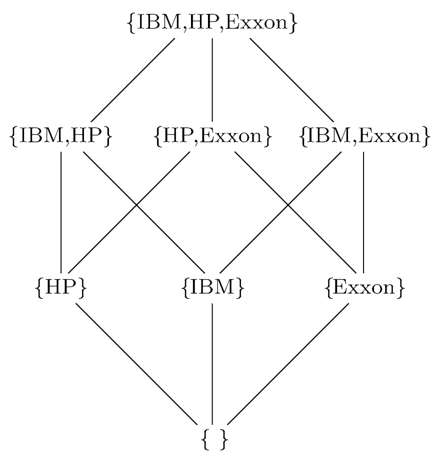


Fig. 2. The powerset lattice of $\{IBM, HP, Exxon\}$.

In a MAC model subjects do not necessarily correspond to users. For example, a subject might be a process that is created and/or owned by a user. In our model, a user is *cleared* to an interval and may create/own a subject with an interval that is a subset of the user's clearance. Thus, a user that is cleared to interval $[\{Exxon\}, \{IBM, Exxon\}]$ might create processes A and B with classification intervals $[\{Exxon\}, \{Exxon\}]$ and $[\{Exxon, IBM\}, \{IBM, Exxon\}]$, to process Exxon information and aggregates of Exxon and IBM information (and process A may send data to B, but not vice-versa). Thus, while the user is permitted to read IBM data, the user is not permitted to write IBM-only information. A wide variety of MAC policies can be enforced with these simple policies and the reader is referred to [10] for further examples.

3.2. Assurance levels

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, $\leqslant$. 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. Traditionally, assurance ratings are given as a total ordering, for example, $A1 > B3 > B2 > B1 > \cdots$ is the assurance ratings from [20]. In this paper, following [11], we generalize this to a lattice structure as it allows for incomparable assurance levels. 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.

3.3. Risk functions

The relationship between assurance levels and security intervals can provide an indication of how much an entity 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, a 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 encode the relationship between assurance levels and security intervals. Given an entity, e, with assurance level, a, and a security interval, $[x, y]$, then the risk that the entity can be compromised is defined as $risk([x, y], a, e) \in \mathbb{N}$ along with the standard "+" operator to aggregate risk values and the "<" operator to compare them.

It would be interesting in the future to develop algorithms that would compute the risk level of an entity, given its assurance level and security interval. An early approach that adopted the notion of *risk* was introduced in the "Red Book" [20] and the "Yellow Book" [6], which also discussed the issue of what constitutes acceptable risk levels when evaluating the security of computing systems.

Note that our choice of natural numbers as a quantification of risk is arbitrary, however this choice is simple and useful. Other choices (other algebras) could have been equally valid, however, the exploration of these is outside the scope of the paper.

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

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

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

$$risk\big([\{\}, \{IBM, HP\}], lo, disk\big) = risk\big([\{\}, \{IBM, HP, Exxon\}], lo, disk\big) = 40.$$

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

$$risk\big([\{\}, \{IBM, Exxon\}], lo, disk\big) = risk\big([\{\}, \{HP, Exxon\}], lo, disk\big) = 10.$$

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

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

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 [9, 15,17] which can be thought of as defining a ((un)acceptable aggregation) binary risk relation.

4. Configuring secure SANs

A *secure SAN* is a SAN extended with the mandatory security model defined in the previous section. 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.

More precisely, before the configuration process can commence we require:

- The security policy, that is, the lattice of security classes, *SC*, and the risk function, *risk*.
- The application requirements, that is, the security 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.

4.1. 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 since 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 compute the interval of an application that will access those datasets through streams, using the lattice *meet*, $\sqcap$, and *join*, $\sqcup$, operations:

$$\mathit{int}(\mathit{app}) = [(\sqcap\ \mathit{set}_\bot), (\sqcup\ \mathit{set}_\top)],$$

where,

$$\begin{aligned}\mathit{set}_\bot &= \{\mathit{int}_\bot(\mathit{data}) \mid (\mathit{app}, \mathit{op}, \mathit{data}) \in \mathit{STREAM} \wedge \mathit{op} = W\},\\ \mathit{set}_\top &= \{\mathit{int}_\top(\mathit{data}') \mid (\mathit{app}, \mathit{op}, \mathit{data}') \in \mathit{STREAM} \wedge \mathit{op} = R\}.\end{aligned}$$

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

$$\mathit{int}(\mathit{lv}) = [(\sqcap\ \mathit{set}_\bot), (\sqcup\ \mathit{set}_\top)],$$

where,

$$\begin{aligned}\mathit{set}_\bot &= \{\mathit{int}_\bot(\mathit{data}) \mid \mathit{data} \in \mathit{DATASET} \wedge \mathit{storedOn}(\mathit{data}) = \mathit{lv}\},\\ \mathit{set}_\top &= \{\mathit{int}_\top(\mathit{data}) \mid \mathit{data} \in \mathit{DATASET} \wedge \mathit{storedOn}(\mathit{data}) = \mathit{lv}\}.\end{aligned}$$

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

$$\mathit{int}(\mathit{ctr}) = [(\sqcap\ \mathit{set}_\bot), (\sqcup\ \mathit{set}_\top)],$$

where,

$$\begin{aligned}\mathit{set}_\bot &= \{\mathit{int}_\bot(\mathit{lv}) \mid \mathit{lv} \in \mathit{LV} \wedge \mathit{lv} \in \mathit{serves}(\mathit{ctr})\},\\ \mathit{set}_\top &= \{\mathit{int}_\top(\mathit{lv}) \mid \mathit{lv} \in \mathit{LV} \wedge \mathit{lv} \in \mathit{serves}(\mathit{ctr})\}.\end{aligned}$$

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

$$\mathit{int}(\mathit{disk}) = \mathit{int}(\mathit{partOf}(\mathit{disk})).$$

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:

$$\mathit{int}(\mathit{srv}) = [(\sqcap\ \mathit{set}_\bot), (\sqcup\ \mathit{set}_\top)],$$

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:

$$int(swt) = [(\sqcap\, set_\perp), (\sqcup\, set_\top)],$$

where,

$$\begin{aligned} set_\perp = \mu\, swt'.&\\ &\big(\{int_\perp(srv) \mid srv \in SERVER \wedge srv \in connects(swt)\}\\ &\cup \{int_\perp(ctr) \mid srv \in CNTR \wedge ctr \in connects(swt)\}\\ &\cup \{int_\perp(swt') \mid swt' \in SWITCH \wedge swt' \in connects(swt)\}\big),\\ set_\top = \mu\, swt'.&\\ &\big(\{int_\top(srv) \mid srv \in SERVER \wedge srv \in connects(swt)\}\\ &\cup \{int_\top(ctr) \mid srv \in CNTR \wedge ctr \in connects(swt)\}\\ &\cup \{int_\top(swt') \mid swt' \in SWITCH \wedge swt' \in connects(swt)\}\big). \end{aligned}$$

The usage of the least-fixed point operator, μ, is required since the definition of *connects* is recursive.

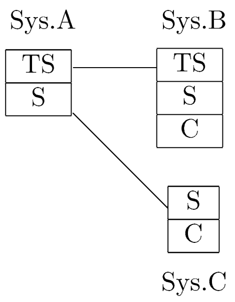
Example 3. Given the following dataset intervals:

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

and an application, *app*, using the following streams

$$(app, \{R\}, data_1),$$
$$(app, \{R, W\}, data_2),$$
$$(app, \{W\}, data_3),$$

then if *app* is classified with the interval, $[\{foo\}, \{IBM, Exxon, foo\}]$, it can handle the above data using the streams indicated.



Assurance Levels	
System	Level
Sys.A	B2
Sys.B	B3
Sys.C	B1

Fig. 3. The cascade vulnerability problem.

5. Cascade vulnerabilities

Cascade vulnerabilities were first identified in [20] in networks of systems with multilevel security classifications and assurance levels. In such systems, the problem arises in scenarios similar to that shown in Fig. 3.

Given the ordering on security classifications, $C \leqslant S \leqslant \mathit{TS}$, and ordering, $B1 \leqslant B2 \leqslant B3$, on levels of assurance, then this system exhibits a case of the cascade vulnerability problem arising from the fact that an attacker classified at security level, C, and possessing the capabilities to break assurance level, $B2$ (and below), can attack and break $Sys.A$ and $Sys.C$. As a result, the attacker can go through the following scenario: first, it downgrades information labeled *TS* on $Sys.A$ to level S. Second, it communicates this information to $Sys.C$. Finally, it downgrades the information to level C, which is the level of the attacker itself, i.e. it can read the information perfectly legally now. This scenario reflects a vulnerability, which undermines the assurance requirement stating that for information to be down-classified from *TS* to C, an attacker needs to be able to have capability at level $B3$ or higher.

Cascade vulnerability attacks are possible in any network model with security and assurance classifications. In the following sections, we present an analysis of the vulnerability in our model of secure SANs.

5.1. Introductory definitions

First, we need to clarify what we mean by a network.

Definition 5.1 (*Network*). Assuming a set of nodes, $\mathcal{N} = \{n_1, n_2, \ldots\}$, a set of edges, $\mathcal{E} = \{(n, n', [x, x]), \ldots\}$, where only data classified at security level, $[x, x]$[2] (or lower), is allowed to flow from n to n', then a network is defined as the following quadruple:

$$(\mathcal{N}, \mathcal{E}, \mathit{int}, \mathit{assur}),$$

[2]We follow the convention adopted in Section 4.1 that data have point security intervals.

where *int* : $\mathcal{N} \rightarrow$ (*SC* × *SC*) and *assur* : $\mathcal{N} \rightarrow A$ are the usual mappings from nodes to their security intervals and to their assurance levels, respectively.

A network then contains nodes, edges and security and assurance classification mappings. In a certain sense, a network is a directed graph with the extra information of assurance and security level mappings. Based on this definition of a network, we define a chain as follows.

Definition 5.2 (*Chain*). Given a network, $(\mathcal{N}, \mathcal{E}, \mathit{int}, \mathit{assur})$, then a chain is defined as any list of nodes, $[n_1, \ldots, n_m]$, which satisfies the property:

$$\forall n_i, n_{i+1} \in [n_1, \ldots, n_m], \exists [x, x] \in SC \times SC\colon (n_i, n_{i+1}, [x, x]) \in \mathcal{E}.$$

Hence, in Fig. 3, we have that $[\mathit{Sys}.B, \mathit{Sys}.A, \mathit{Sys}.D, \mathit{Sys}.C]$ is a chain, and so is $[\mathit{Sys}.D, \mathit{Sys}.C]$. Furthermore, a bad chain is a chain that can be compromised by some attacker.

Definition 5.3 (*Bad chain*). Given an attacker, I, with a capability to break a maximum assurance level, $a_I \in A$, then we say that a chain, $[n_1, \ldots, n_m]$, is a *bad chain* with respect to I, if the following is true:

$$\forall n \in [n_1, \ldots, n_m]\colon \mathit{assur}(n) \leqslant a_I.$$

This implies that every element in the chain can be broken by I. Using these definitions, we can now define a cascade vulnerability formally as follows.

Definition 5.4 (*Cascade vulnerability*). We say that an attacker, I, with security level, $\mathit{int}(I) \in SC \times SC$, and capability, $a_I \in A$, poses a *cascaded threat* using a bad chain, $[n_1, \ldots, n_m]$, to data classified at level $[x, x] \in \mathit{SC} \times \mathit{SC}$ on some node, $n \notin [n_1, \ldots, n_m]$, if the following condition holds true:

$$\exists n_i, n_j \in [n_1, \ldots, n_m]\colon (n, n_i, [x, x]) \in \mathcal{E} \wedge \mathit{int}_{\top}(I) \geqslant \mathit{int}_{\perp}(n_j).$$

The condition in Definition 5.4 highlights the role of two important nodes in a bad chain: the first, n_i, which is linked at security level, $[x, x]$, with the victim node, n, acts as an indirect leakage point since data classified at $[x, x]$ can legally pass from n to n_i. The second node, n_j, acts as a direct point of leakage of data to the attacker since the latter can legally read any data n_j is allowed to write. This is true because the attacker has a read-level of $\mathit{int}_{\top}(I)$, which is more than or equal to the write-level, $\mathit{int}_{\perp}(n_j)$. In other words, a cascade vulnerability arises whenever an attacker becomes capable of manipulating a bad chain to obtain sensitive data from a node, which has some sort of connection to the bad chain, and then downgrade and read the sensitive data.

In the following sections, we give concrete examples of the cascade vulnerability problem in the cases of networks of logical volumes and networks of application servers.

5.2. Networks of logical volumes

Using the model of SANs as defined in Section 2, we define a network of logical volumes as follows.

Definition 5.5 (*Network of logical volumes*). Given some definition of the function, $\mathit{storedOn}: \mathit{DATASET} \rightarrow \mathit{LV}$, and a set of streams, STREAM, then we can define a network of logical volumes as follows:

$$\begin{aligned}
&\bigl(\mathcal{N} = \{lv_1, \ldots, lv_m\},\\
&\mathcal{E} = \bigl\{(lv_i, lv_j, [x, x]) \mid lv_i, lv_j \in \{lv_1, \ldots, lv_m\}\\
&\qquad\qquad \wedge \exists(app, R, data_i), (app, W, data_j) \in \mathit{STREAM}\\
&\qquad\qquad \wedge \mathit{storedOn}(data_i) = lv_i \wedge \mathit{storedOn}(data_j) = lv_j\\
&\qquad\qquad \wedge [x, x] = \mathit{int}(data_i)\bigr\},\\
&\mathit{int}: \{lv_1, \ldots, lv_m\} \rightarrow (\mathit{SC} \times \mathit{SC}),\\
&\mathit{assur}: \{lv_1, \ldots, lv_m\} \rightarrow A\bigr).
\end{aligned}$$

Hence, the flow of information from lv_i to lv_j is modeled as the presence of an application, *app*, which reads from a dataset, $data_i$, stored on lv_i and then writes to another dataset, $data_j$, stored on lv_j. Note that since it is $data_i$ which is *flowing* from lv_i to lv_j, then $[x, x]$ is taken as the interval of $data_i$.

By adopting this definition of a network of logical volumes, we can reason about the presence of a cascade threat towards a logical volume, $lv \in \mathcal{N}$, at data interval, $[x, x]$, from an attacker, I, capable of breaking a bad chain of logical volumes, $[lv_1, \ldots, lv_m]$, as follows:

$$\begin{aligned}
&\mathit{cascadeLv}(lv, [x, x], I)\\
&\quad \stackrel{\text{def}}{=} \exists lv_i, lv_j \in [lv_1, \ldots, lv_m]\colon\ (lv, lv_i, [x, x]) \in \mathcal{E} \wedge \mathit{int}_{\top}(I) \geqslant \mathit{int}_{\bot}(lv_j).
\end{aligned}$$

This definition of cascade vulnerability in a network of logical volumes is a straightforward adaptation of Definition (5.4).

5.3. Networks of application servers

The other network that we identify within our model of secure SANs is the network of application servers.

Definition 5.6 (*Network of application servers*). Given some definition of the function, $\mathit{runsOn}: \mathit{APP} \rightarrow \mathit{SERVER}$, and a set of streams, STREAM, then we define a network of application servers by the following quadruple:

$$\begin{aligned}
&\big(\mathcal{N} = \{srv_1, \dots, srv_m\},\\
&\mathcal{E} = \big\{(srv_i, srv_j, [x,x]) \mid srv_i, srv_j \in \{srv_1, \dots, srv_m\}\\
&\qquad\qquad\qquad\qquad \wedge \exists (app_i, W, data), (app_j, R, data) \in STREAM\\
&\qquad\qquad\qquad\qquad \wedge runsOn(app_i) = srv_i \wedge runsOn(app_j) = srv_j\\
&\qquad\qquad\qquad\qquad \wedge [x,x] = int(data)\big\},\\
&int : \{srv_1, \dots, srv_m\} \to (SC \times SC),\\
&assur : \{srv_1, \dots, srv_m\} \to A\big).
\end{aligned}$$

In this definition, information flows from srv_i to srv_j if there exists a dataset, *data*, which acts as a shared space to which an application, app_i, running on srv_i writes and from which another application, app_j, running on srv_j reads from. The security interval, $[x, x]$, is taken directly as the interval of *ata*. Note that this manipulation of streams is quite different from that of the previous section. However, the definition of the cascade vulnerability is quite similar: a cascaded threat towards an application server, *srv*, is possible at security level, $[x, x]$, given some intruder, I, and a bad chain of application servers, $[srv_1, \dots, srv_m]$, if the following predicate holds true:

$$\begin{aligned}
&cascadeSrv(srv, [x,x], I)\\
&\quad \stackrel{\text{def}}{=} \exists srv_i, srv_j \in [srv_1, \dots, srv_m]:\\
&\quad (srv, srv_i, [x,x]) \in \mathcal{E} \wedge int_{\top}(I) \geqslant int_{\bot}(srv_j).
\end{aligned}$$

6. Optimal configurations

After defining the security intervals, $int(e)$, of every SAN entity, e, as in Section 4.1, and given that entities have fixed assurance levels, $assur(e)$, then the problem of finding optimal configurations consists in searching for definitions of one or more of the functions, *partOf*, *serves*, *runsOn*, *storedOn* and *connects*, such that the total risk level is at a minimum value:

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

Individual customers may insist on a Service Level Agreement (SLA) that limits the risk that their data is compromised. For example, a customer may provide a security interval int_{SLA} and a limit ν_{SLA} and require that:

$$\sum_{\forall e \in ENTITIES:\ int(e) \cap int_{SLA} \neq \{\}} risk(int(e), assur(e), e) \leqslant \nu_{SLA}. \tag{2}$$

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

Furthermore, the customer may require that no instances exist of the two cascade vulnerabilities as defined in Section 5 given some attacker model I:

$$\forall lv \in LV, x \in SC\colon \neg cascadeLv(lv, [x, x], I), \tag{3}$$

$$\forall srv \in SERVER, x \in SC\colon \neg cascadeSrv(srv, [x, x], I). \tag{4}$$

Example 4. In a particular incomplete configuration of a secure SAN, assume that the aggregate risk levels of applications, logical volumes, controllers, servers and switches are as follows:

$$\sum_{\forall app \in APP} risk(int(app), assur(app), app) = 70,$$

$$\sum_{\forall lv \in LV} risk(int(lv), assur(lv), lv) = 50,$$

$$\sum_{\forall ctr \in CNTR} risk(int(ctr), assur(ctr), ctr) = 45,$$

$$\sum_{\forall srv \in SERVER} risk(int(srv), assur(srv), srv) = 45,$$

$$\sum_{\forall swt \in SWITCH} risk(int(swt), assur(swt), swt) = 30.$$

Then, we would like to find out the optimal definition of the *partOf* function (i.e. the assignment distribution of physical disks to logical volumes), for each of the following:

(a) $\sum_{\forall e \in ENTITIES} risk(int(e), assur(e), e) = 300$,
(b) $\nu_{SLA} = 280$.

To solve case (a), we need to find out the maximum risk level disks may have:

$$\sum_{\forall disk \in DISK} risk(int(disk), assur(disk), disk)$$
$$= 300 - (70 + 50 + 45 + 45 + 30) = 60.$$

From the definition of disk security intervals, we obtain:

$$\sum_{\forall disk \in DISK} risk\big(int(partOf(disk)), assur(disk), disk\big) = 60.$$

Assuming we already have definitions of *serves*, *storedOn*, *runsOn* and *connects*, then we can solve the above equation to find out the most suitable definition(s) of *partOf* function and thereby complete the configuration of the SAN.

In the case of (b), the procedure is similar, except that now we have:

$$\sum_{\forall disk \in DISK} risk(int(disk), assur(disk), disk)$$
$$= 280 - (70 + 50 + 45 + 45 + 30) = 40.$$

7. 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 [21] for the implementation language because of its built-in logic and search capabilities.

The 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 formulae 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 [12].
- 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.

As a tool for exploring the space of SAN risk models, OPL worked well for us. As part of this research we went through several iterations of the risk model, expressing each iteration in OPL. This process was straightforward and helped uncover holes in our thinking. By using a logic language the semantic distance between the model and the implementation was small enough that the implementation actually acted as a useful means of expression, not just a hurdle to be jumped. The discipline of implementing and running the model also provided a check, keeping the model simpler than it might have been.

However the OPL implementation that we used was designed as a programming environment not as a deployment tool. Embedding it into a larger n-tier system would be difficult. In addition there are inputs where the search procedure we used could require exponential time, this is hard to avoid when looking for an exact solution, because the optimization problem itself is NP-complete by reduction to bin packing. To address both issues, a custom solver is the best answer, both for efficiency, encapsulation and packaging. Once the model is fixed, a custom solver allows for more efficient evaluation of the constraints and the objective functions, and for the use of approximation techniques, such as simulated annealing or genetic algorithms, that have been tuned to the model, Maintaining such a solver increases the cost of changing the model, but when looked at as part of the total cost of maintaining a complex SAN management product, it is not the highest order term.

8. Conclusion and future work

In this paper, we presented a model of mandatory security for SANs. The primary contribution in this paper was the development of a framework that can be used to determine a secure data-separation configuration of a SAN. The model incorporated an evaluation of the total risk incurred when data stored on a SAN are compromised. We also outlined an implementation of the configuration process 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 [3,7,9,10,14,15,17,20]. Therefore, we conjecture that the results in this paper can be usefully applied to other 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 [20]; the risk function is used to guide the generation of a secure configuration within an acceptable degree/measure of risk. The data-separation security policy is not unlike the requirements for a Chinese Wall security policy [5], and the model proposed in this paper provides a novel risk-based interpretation for Chinese Walls that can be used in a non-SAN situation. Given the duality between Chinese wall (aggregation) polices and separation of duty policies, we argue that the proposed model can also be used to capture dynamic separation of duty policies that are risk-based interpretations of those described in [8]. This is a topic for future work.

One limitation of our configuration technique is the possibility that it might be hard to apply it to newer shared storage models like those provided by Amazon's S3 (aws.amazon.com/s3). Primarily because a dataset on S3 would typically be spread out over the entire data center with portions stored on every single logical volume. This reduces their cost of administration and increases their parallelism but at the cost of not being able to bound the security foot print of the dataset.

Future work would explore the definition of risk functions and how they behave towards changes in assurance levels and security intervals. Also, we plan to investigate more deeply other security properties, such as integrity and availability, as well as other models of security, such as the continuous Usage Control ($\mathrm{UCON}_{\mathrm{ABC}}$) model [18]. Other future work includes extending the OPL implementation to be able to capture instances of the cascade vulnerability problem.

References

[1] B. Aziz, S.N. Foley, J. Herbert and G. Swart, Configuring storage area networks for mandatory security, in: *Proceedings of the 18th IFIP Annual Conference on Data and Applications Security*, C. Farkas and P. Samarati, eds, Kluwer, Sitges, Catalonia, July 2004, pp. 357–370.

[2] D.E. Bell, Concerning modelling of computer security, in: *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA, April 1988, pp. 8–13.

[3] D.E. Bell and L.J. La Padula, Secure computer systems: Unified exposition and multics interpretation, Technical report ESD-TR-75-306, Mitre Corporation, Bedford, MA, USA, July 1975.

[4] M. Branstad et al., Trusted mach design issues, in: *Proceedings of the 3rd AIAA/ASIS/DODCI Aerospace Computer Security Conference*, IEEE Press, Orlando, FL, December 1987.

[5] D.F.C. Brewer and M.J. Nash, The Chinese wall security policy, in: *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA, May 1989, pp. 206–214,

[6] DoD, Computer security requirements guidance for applying the department of defence trusted computer system evaluation criteria in specific environments, Technical report CSC-STD-003-85, DoD Computer Security Center, 1985 (Yellow Book).

[7] S.N. Foley, A model for secure information flow, in: *Proceedings of the* 1989 *IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA, May 1989, pp. 248–258.

[8] S.N. Foley, Secure information flow using security groups, in: *Proceedings of the 3rd IEEE Computer Security Foundations Workshop*, IEEE Computer Society Press, Franconia, NH, June 1990, pp. 62–72.

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

[10] S.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*, ACM Press, Zurich, April 1997, pp. 125–134.

[11] S.N. Foley, Conduit cascades and secure synchronization: in *Proceedings of the ACM New Security Paradigms Workshop*, ACM Press, Cork, September 2000, pp. 141–150.

[12] E. Freuder, Eliminating interchangeable values in constraint satisfaction problems, in: *Proceedings of the 9th National Conference on Artificial Intelligence*, Vol. 1, MIT Press, Anaheim, CA, July 1991, pp. 227–233.

[13] Y. Kim, M. Narasimha, F. Maino and G. Tsudik, Secure group services for storage area networks, in: *Proceedings of the First International IEEE Security in Storage Workshop*, IEEE Computer Society, Greenbelt, MD, December 2002, pp. 80–93.

[14] T.M.P. Lee, Using mandatory integrity to enforce 'commercial' security, in: *Proceedings of the Symposium on Security and Privacy*, IEEE Press, Oakland, CA, 1988, pp. 140–146.

[15] C. Meadows, Extending the brewer-nash model to a multilevel context, in: *Proceedings of the IEEE Symposium on Security and Privacy*, IEEE Computer Society Press, Oakland, CA, May 1990, pp. 95–102.

[16] 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)*, ACM Press, Chicago, June 1988, pp. 109–116.

[17] R.S. Sandhu, Lattice-based access control models, *Computer* **26**(11) (1993), 9–19.

[18] R. Sandhu and J. Park, The UCON$_{ABC}$ usage control model, *ACM Transactions on Information and System Security* **7**(1) (2004), 128–174.

[19] G. Swart, Storage management by constraint satisfaction. in: *Proceedings of the Workshop on Immediate Applications of Constraint Programming*, Kinsale, Cork, Ireland, September 2003.

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

[21] P. Van Hentenryck, *The OPL Optimization Programming Language*, MIT Press, Cambridge, MA, January 1999.