Storage-Based Intrusion Detection for Storage Area Networks (SANs)

Mohammad Banikazemi Dan Poff Bulent Abali
Thomas J. Watson Research Center
IBM Research
Yorktown Heights, NY 10598
{*mb, poff, abali*}*@us.ibm.com*

Abstract

Storage systems are the next frontier for providing protection against intrusion. Since storage systems see changes to persistent data, several types of intrusions can be detected by storage systems. Intrusion detection (ID) techniques can be deployed in various storage systems. In this paper, we study how intrusions can be detected at the block storage level and in SAN environments. We propose novel approaches for storage-based intrusion detection and discuss how features of state-of-the-art block storage systems can be used for intrusion detection and recovery of compromised data. In particular we present two prototype systems. First we present a real time intrusion detection system (IDS) which has been integrated within a storage management and virtualization system. In this system incoming requests for storage blocks are examined for signs of intrusions in real time. We then discuss how intrusion detection schemes can be deployed as an appliance loosely coupled with a SAN storage system. The major advantage of this approach is that it does not require any modification and enhancement to the storage system software. In this approach, we use the space and time efficient point-in-time copy operation provided by SAN storage devices. We also present performance results showing that the impact of ID on the overall storage system performance is negligible. Recovering data in compromised systems is also discussed.

1. Introduction

Intrusion detection systems (IDSs) are mainly either host-based or network-based. Host-based IDSs operate as a part of the host operating environment [2, 13, 16] and monitor local activities for signs of intrusion. Network-based IDSs monitor network traffic for signs of suspicious activities [3, 14]. Storage systems are the next frontier for providing protection against intrusion [15]. Storage systems see changes to persistent data and can therefore detect several types of intrusions, especially those persisting across boots. Storage systems are particularly suited for this purpose because they continue operating even after the host system is compromised. Furthermore, since they provide a narrow interface to the outside world (such as the SCSI command set), they are more difficult to be compromised themselves. Intrusion detection techniques can be deployed in a diverse group of storage systems. In this paper, we study how intrusions can be detected in environments with SAN block storage systems. These systems include a large number of commercial systems such as IBM ESS, SVC, and DS4000 series, EMC Symmetrix, and Hitachi TagmaStore.

We propose two approaches for storage-based intrusion detection and discuss how existing features of block storage systems can be used for intrusion detection and for recovery of compromised data. In particular we present two storage-based IDSs which we have built in our labs. First we present a real time intrusion detection system which has been integrated within the IBM SVC, a storage management and virtualization engine for Storage Area Networks (SANs). In this system file-based ID access rules are converted to block-based rules such that incoming requests for storage blocks are examined for any sign of intrusion. When there is a possible sign of intrusion, the content of the data block is inspected if necessary. Whenever an intrusion is detected, various actions such as informing the system administrators, rejecting suspicious IO requests, or delay-

Proceedings of the 22nd IEEE/13th NASA Goddard Conference on Mass Storage Systems and Technologies (MSST 2005)
0-7695-2318-8/05 $20.00 © 2005 IEEE

ing them. Furthermore, we show that the performance impact of the integration of the IDS into IBM SVC storage system is negligible.

We then discuss a second IDS which can be deployed as an intrusion detection appliance loosely coupled with the storage system. The major advantage of this approach over the first IDS is that it does not require any modification and enhancement to the storage system software. In this approach, we use the space and time efficient point-in-time copy operation provided by the IBM DS4000 series storage controllers, called FlashCopy. These operations are used to create copies of the logical volumes of interest. These copies are then mounted for inspection and necessary operations are performed on them to detect signs of intrusion. We use the last known good copy volumes to recover compromised data if an intrusion is detected. We also discuss how a minor enhancement to storage systems, for providing a list of modified blocks to the IDS, can improve the scalability of such a scheme.

The major contributions of this paper are as follows.

- Design and implementation issues for building an IDS as an integrated part of a block storage system are discussed.
- Performance results show that the performance impact of using such an IDS on the storage system is negligible.
- Another storage-based IDS is presented which uses features of modern SAN storage systems such as point-in-time copy to provide intrusion detection schemes without requiring any changes in storage devices.
- Recovering data in compromised systems is also discussed.

The rest of this paper is organized as follows. We briefly discuss Storage Area Networks in Section 2. The main components of rule base IDSs are discussed in section 3. Then we describe the environment in which we have developed our prototypes in Section 4. Section 5 presents our prototype implementation. The performance results are presented in Section 6. Related work is discussed in Section 7. Future work and our conclusions are presented in Section 8.

2. Storage Area Networks (SANs)

The networked storage market has been growing as a result of growing demand for storage capacity. Networked storage systems can be divided into two major groups: Network Attached Storage (NAS) and Storage Area Networks (SAN). NAS systems typically provide a file system interface while SANs provide block storage services. NAS systems are usually Ethernet-based while SAN systems mostly rely on the Fibre Channel interconnect. Introduction of iSCSI and Object Store Devices are blurring the line between SAN and NAS. In this paper we focus on SANs.

SAN systems are made of storage devices, specialized networks for connecting data storage devices to servers and the required management layer for setting up and maintaining connections between these servers and data storage devices. Fibre Channel interconnect is the primary interconnect used in SAN environments. Similar to direct attached disks, SAN systems provide a simple block level (fixed size) interface for storing and retrieving data [5]. Figure 1 shows a SAN environment with multiple storage devices.

Even though SANs have gained a wide acceptance, the problem of managing a heterogeneous storage system is still a major challenge [6]. Block virtualization approach is used to address some of the complexities involved in managing such systems by aggregating the storage into a common pool. Storage units are assigned to host systems (*i.e.*, servers) from this common pool. Furthermore, SAN storage virtualization and management systems may provide performance enhancing services such as caching, and other services such as copy services. In the rest of this paper we refer to the storage area network and the storage devices attached to it as the SAN system.

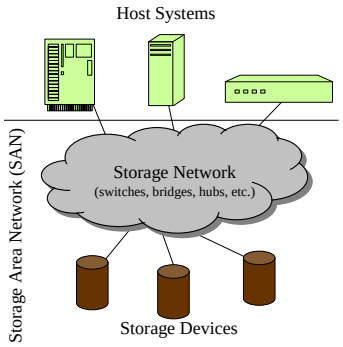

Figure 1. Storage Area Network (SAN).

3. Intrusion Detection in SAN Environments

Rule-based (policy-based) IDSs are one of the major kinds of IDSs. In this paper we focus on such IDSs. A

rule-based Intrusion Detection system has three major components: 1) access rules and mechanisms for specifying rules, 2) mechanisms for monitoring for rule violations, and 3) actions taken in response to rule violations. Storage-based IDSs have the same major components. In this section we discuss these components for IDSs in SAN environments.

3.1. Access Rules

Since files are what system users and administrators mostly deal with, it is desirable and perhaps required to define the IDS access rules with respect to files rather than storage data blocks. File-based access rules can be used to monitor accesses to the content of a file and/or the corresponding metadata, such as the file access permission fields. Access rules specify which files and which parts of their metadata are to be monitored and what types of access to these fields constitute a rule violation. Tripwire [11] is a file system integrity checker and a host-based IDS which uses such a scheme. A similar approach can be used for storage-based IDSs. In such systems, a configuration file containing access rules for files (or groups of files) is used. By the use of a selection mask, each entry in this file defines how a file can be accessed.

3.2. Detecting Intrusions

Even though access rules are defined with respect to files, block storage devices have no notion of files or how different blocks are associated with each other to create a file. Therefore, SAN-based IDSs should be able to bridge this gap. To this end, SAN-based IDSs can take two main approaches: 1) converting file-based access rules to storage block-based rules by understanding file system data structures or 2) using file system implementations to monitor/evaluate files at file system level.

When file-based rules are translated to storage block access rules, access to storage devices can be monitored in real time. We call this type of IDS a real time IDS. The translation can be performed by the block storage system itself if it incorporates file system smarts or by using a different system and feeding the translation to the block storage system. In order to have a real time IDS, the IDS needs to update block-based rules in a real time fashion whenever it is needed.

If real time intrusion detection (ID) is not a requirement, file system level ID techniques can be deployed. We call these systems delayed IDSs. Several features provided by modern SAN systems can be used to facilitate the ID task. In particular, copy services provided by block storage systems can be used to create a copy of volumes of interest for mounting file systems and monitoring and evaluating the system at file system level. In Section 5, we will discuss two implementations where each deploys one of these approaches.

3.3. Response to Intrusions

A set of potential methods for responding to intrusion has been discussed in [7, 15]. These methods include generating alerts, preventing requests from completing, and slowing storage requests while awaiting response from system administrators. These methods can be deployed by storage-based IDSs. In addition to these schemes, operations such as block level versioning [4], point-in-time copy and continuous copy can be employed whenever a possible violation is detected to preserve a copy of data while the situation is being analyzed. For delayed IDSs, it is important to provide mechanisms for recovering data by rewinding the whole disk to a point in time when no intrusions had occurred. Point-in-time copy operations can be used to keep a good copy of the volumes of interest available. These copies can then be used for recovering individual files or the whole volume.

In the next section we briefly discuss the characteristics of two typical SAN systems. Then, we show how IDSs discussed in this section can be built on these storage systems.

4. Implementation Environment

In this section, we present the description of two storage systems used in our prototypes: IBM TotalStorage SAN Volume Controller (SVC) and IBM TotalStorage DS4000 (DS4000).

IBM SVC is a storage management and virtualization system built on a cluster of Pentium-based servers [6]. SVC provides redundancy, modularity and scalability and manages SAN-attached block devices. A cluster-based approach is used in order to provide a better scalability and also higher availability. SVC virtualizes storage devices into a common pool and allocates storage to host systems from that common pool. Furthermore, deployment of SVC in enterprises enhances the manageability of enterprise storage systems by providing a single point of management for all of storage devices in the system.

SVC uses an *in-band* approach which means that all I/O traffic as well as the management and configuration requests is directed to it and are serviced by it. SVC is designed such that it only uses off-the-shelf component and can support different SAN fabrics by using appropriate adapters. SVC software runs almost entirely in user mode for better performance and avoiding kernel-mode/user-mode switching. This characteristic also makes the development and debugging of new features much easier making SVC the perfect platform for development and evaluation of new ideas and techniques.

SVC supports advanced functions such as data block caching, fast-write cache, copy services, and quality of service metering and reporting [6]. Figure 2 shows a pair of SVC nodes with different storage devices and host systems. In such an environment Fibre Channel zoning is used to zone host systems with the SVC on one hand and storage devices and SVC on the other hand. Fibre Channel zoning is performed such that host systems cannot see the storage devices directly. All requests from host systems is sent to SVC and SVC performs the operation and returns the response to the host. SVC caches data blocks for better performance.

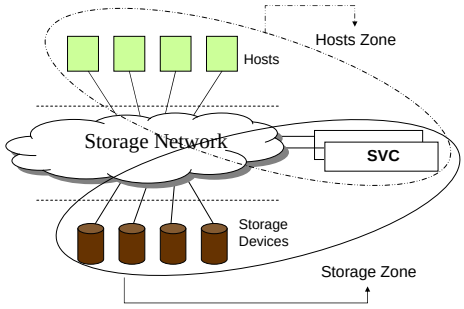

Figure 2. Using the SVC storage system.

The other storage system we used for developing one of our prototypes is an **IBM DS4000** [9] system. IBM DS4000 series is a family of mid-level storage controllers capable of supporting terabytes of SAN attached disks and hundreds of logical volumes (LUNs). In the rest of this paper we refer to any of the members of DS4000 series as a DS4000 system. DS4000 systems use dual redundant components for high availability. They also support multiple RAID levels for data protection. Similar to SVC, DS4000 systems use the in-line design and provide data block caching. The DS4000 management software is used for creating LUNs and mapping them to appropriate host systems.

DS4000 systems provide various copy services such as a space and time efficient point-in-time copy called FlashCopy. This operation provides a method for copying LUNs instantaneously because during the FlashCopy operation, content of the source LUN is not physically copied. Only control data structures for the newly created LUN are created such that accesses to its data blocks are referred to the corresponding physical data blocks in the source LUN. Only when a block in either source (original) or target (copy) LUN is modified is the relation between corresponding data blocks in two LUNs broken, affected blocks copied, and future accesses referred to different physical data blocks.

5. Prototype Implementations

In this section we discuss two prototype systems we have developed. We first present a real time intrusion detection system (IDS) built as an integrated part of IBM SVC. Then we present the second prototype which is a delayed intrusion detection (ID) appliance working with IBM DS4000 systems. It should be noted that the IDSs described here are research prototypes and do not exist as parts of any IBM product.

5.1. Real Time Storage-Based (RTSB) IDS

We have built a real time IDS which monitors accesses to storage devices at storage block level. This system is built as an integrated part of the IBM SVC storage system. As mentioned in Section 3, for performing ID techniques at storage block level, we need to convert the file-based access rules, to block-based rules. In our prototype, file-based rules are converted to block-based rules by an enhanced SVC. The storage system is provided with file-based access rules through the same secure channels used for accessing it for administrative purposes. SVC then converts these rules to block level rules. A schematic of RTSB is shown in Figure 3. In this figure we have omitted the storage network components (such as Fibre Channel switches) for simplicity. It can be seen that all incoming requests go through the ID component first for inspection.

Currently in our prototype, the block storage system identifies file system blocks associated with files in the *Linux ext2* file system by traversing the storage data blocks and interpreting the super block and i-node information. In particular, for any given file in a Tripwire-like access rule, corresponding file data blocks (and then storage data blocks) are identified. These blocks include both data and metadata blocks. Metadata blocks

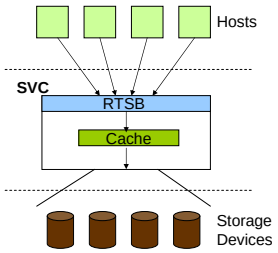

Figure 3. The RTSB IDS.

consist of not only the file metadata itself (that is the block containing the file i-node) but also blocks associated with the directories in the file path. For example, metadata blocks for the file */xyz/abc* consist of the i-node block for the file *abc* and also the data blocks for / and */xyz* directories. Then, the file access rule is converted to access rules for these blocks which specify what types of access to a block or parts of a block are permitted.

Once file-based rules are converted to block-based rules, incoming requests to the storage system are monitored. If there is any access rule associated with a given block, the access is further examined to verify if any of the access rules are violated. In the current implementation, for each LUN a bitmap is used to keep track of blocks with access rules. Whenever an access rule gets associated with a block the corresponding bit in the bitmap is set. This way, figuring out if an access needs further scrutiny can be performed efficiently. Obviously, more space efficient schemes such as using hash tables for keeping track of data blocks with access rules can be deployed. We have also implemented another scheme which is more space efficient than the bitmap approach. We will discuss this scheme in Section 6.

If a block is not shared by multiple files, and in cases where any write and/or read access to the block results in a violation of an access rule, appropriate action can be immediately initiated. Cases where a block is shared by more than one file, and/or only accesses to certain portions of the block causes a violation are more complicated. For read accesses, there is no way for the storage system to see which portion of the block is being used and therefore if there is a rule which prohibits read access to a block or parts of it, appropriate action should be taken as if a violation has occurred. This feature is used for implementing honey pot schemes for detecting intruders. For write accesses, IDS needs to examine the content of the block and compare it with the old content to determine which part of the block is being modified and if an access rule is being violated. In our implementation, whenever a write request to a block with restricted write access is received, the current content of the block is first read and then compared with the new data in order to determine which part of the block is being modified and if any rules are being violated.

The access rules are checked in a layer above the SVC cache layer. Therefore, in cases where a comparison between the current and new content is necessary, the old content may be residing in the storage cache. In these cases, the overhead of accessing the old content is very low. Storage blocks corresponding to files which are being monitored can be given higher priority for caching purposes. This feature is not implemented in the current prototype. In Section 6 we investigate the performance impact of the inspection of content of data blocks. In current implementation, any rule violations triggers an email to the system administrator. The violation is not only identified by the storage block but also file(s) and file-based rules which have resulted in the block level rule. In order to do this, each block level rule has a pointer to the list of file level rules from which it has been created.

5.2. File Level Storage-based (FLSB) IDS

The scheme presented in the previous section requires modification of the storage system software and firmware. Furthermore, it requires the conversion of file-based rules to block-based rules by the storage system. This requires that the storage system have the smarts to interpret the metadata of file systems residing on its LUNs. This becomes a more challenging requirement when a file system of interest is not supported by the native operating system/environment of the storage controller. In this section, we present a second storage-based IDS we have built which addresses these issues by operating at the file system level. The IDS described here can be trivially implemented using any storage controller supporting space/time efficient point-in-time copy operation without requiring any changes to the storage system.

A schematic of the FLSB system is shown in Figure 4. The storage used by upper host system(s) is being monitored for signs of intrusion. FLSB in this figure is made of the lower host system(s). The FLSB system can be made arbitrarily secure by locating it in a secure location and disconnecting it from the network. FLSB can be made of multiple host systems each possibly running different operating systems such that the file systems used by host systems which are being monitored can be easily monitored. FLSB can also be running on

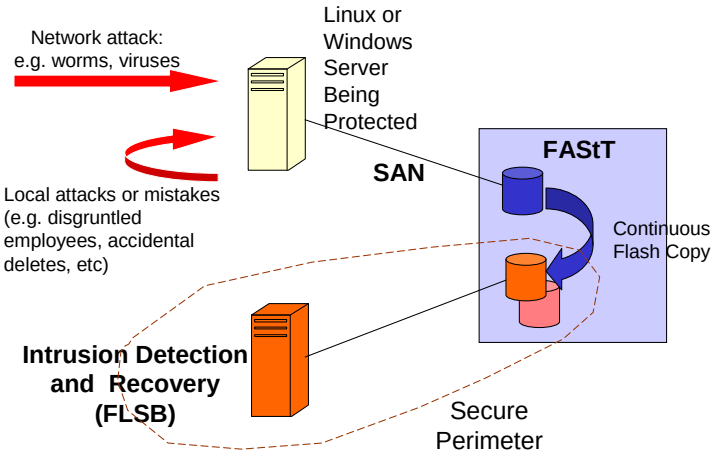

Figure 4. The FLSB IDS.

virtual machines even though we have not used FLSB in such environments.

The FLSB system uses the time and space efficient point-in-time copy operation provided by modern block storage systems. In our implementation, FLSB periodically creates a space efficient point-in-time copy of the LUN(s) of interest by using the DS4000 FlashCopy operation and performs necessary operations on them to determine if any storage volumes has been compromised. The newly created copy LUN is mounted and examined at file system level. Therefore, file-based ID techniques can be easily applied. In our implementation, the Tripwire software is used for this purpose. During the initialization of the system or when the IDS configuration is modified, files of interest are scanned and various signatures are created. These signatures are stored in a different LUN where they cannot be mapped to (or zoned with) any host machine and therefore cannot be compromised by them. Then, periodically point-in-time copies of LUNs of interest are created and the ID software is run against them to create new signatures. The original signatures are used to determine if any file has been compromised. Since the point-in-time copy operation is space efficient, the copy LUN is created almost instantaneously and with a small overhead and minimal storage IO traffic. Frequency of copy operations depends on the number of LUNs (and the number of files in each of them) which are being monitored. Multiple machines can be used in a FLSB system as the number of LUNs being monitor increases. Therefore, this scheme scales easily. In the current implementation by default the copy operation is executed every five minutes. We will discuss a method for improving the scalability of the system in Section 8.

Considering that in this system, intrusions are detected with a delay, necessary measures are considered for providing the capability of recovering compromised data. In the prototyped system, at least one *good* copy of the LUNs of interest are saved such that if and when an intrusion is detected, compromised data can be recovered. A copy is called *good* when it passes the ID examination and no violation is detected. Keeping such a copy requires that the FLSB system can create at least two copies of a LUN. When a copy is recognized as *good*, it is not deleted until the next periodic copy is created, examined and recognized as good. Once a violation is detected, the good copy is protected for future reference and recovery of compromised data.

Similar to original signatures, the periodic copies are created such that they are not accessible by the host machines or any entity other than the FLSB system. Fibre Channel software zoning can be directly used for this purpose. In our implementation, we use the mapping feature of the DS4000 storage system through which the hosts can access a given LUN.

Modern SAN storage systems which support point-in-time copy operations usually support the notion of consistency groups. A consistency group is made of two or more LUNs and operations such as point-in-time copy on any of the LUNs operate on all members of the group. This provides the added benefit of support for file systems which consist of more than one LUN. It should be noted that journal file systems are particularly suitable for use with FLSB systems as they do not require coordination between host systems and the storage system for performing the FlashCopy operation such that the copy is in a consistent state.

In the implemented prototype, once a violation is detected, the good copy is saved permanently until the administrator chooses to remove it. This copy is then used to recover a copy of the file or files which have been compromised. These files are then stored in a separate folder for inspection by the administrator. Alternatively, these copies could be transferred to the LUN of interest replacing the compromised version of files.

6. Performance Evaluation

We performed several experiments in order to determine the impact of adding ID to the storage system. We ran synthetic benchmarks and performed worst case analysis. In order to determine the impact of the incorporation of the ID software into the storage system, we measured the performance of our block storage system with and without our additions. As discussed in Section 5, the RTSB IDS inspects all incoming requests to determine if they may be violating one or more access rules. The software overhead comes from two components: the inspection of block numbers in incoming requests, and then examining the content of write operations when need be.

We used Iometer[12] to access multiple LUNs (up to 12 16-Gigabyte LUNs) through our storage system. We used a variety of request sizes with varying percentage of reads and writes. We also changed the distribution of requests from 100% sequential to 100% random. For all measurements we waited for the reading to stabilize and reported the average of three readings. In order to determine if an access to a data block violates any access rules, we used a bitmap lookup scheme. In order to reduce the memory requirement for storing the bitmap, we maintained the bitmap only for LUNs which required monitoring. We also divided LUNs into regions and avoided keeping the bitmap for regions which did not contain any blocks to be monitored. We only maintained a list of LUNS and their regions of interest which require a much smaller data structure. For the first set of experiments none of the accesses required the inspection and comparison of old and new content. We measured the peak bandwidth under various conditions as described earlier and noticed no measurable change in our readings when this component of our software was deployed. In other words, the change in sustained storage bandwidth was negligible.

In order to measure the impact of comparing the current content of data blocks with that of incoming write requests, we marked various percentage of blocks as blocks to watch for any modification of their content. We also performed the comparison with the current content for the whole request even when requests spanned across multiple file system blocks. In order to quantify the results easily, we first made all the requests write requests. Figures 5 and 6 show the results for 16-KB and 128-KB message sizes. In these figures, the peak bandwidth for Sequential and Random access patterns can be seen. We report the results for cases where the content of 1, 10, and 100 percent of blocks where being inspected.

The results show that when the content of all accessed blocks is monitored, the performance is reduced by a factor of 2. For example it can be seen that with 12 disks the peek observed bandwidth reduces from 90 to 44 for the 128KB writes with random access pattern. This is expected as now each write requests results in two accesses to disk: one to read the current content and one to write back the data. It can be observed that, when the percentage of inspections reduces, the performance gets closer to the performance of the original system. A previous study has shown that the percentage of accesses which require inspection of the content of data blocks is well below 1% [7].

Figures 7 and 8 show the results for 30-70 write-read access type distribution with 1%requests being inspected. The results show the peek bandwidth for two cases: 1) where all disk blocks are accessed and 2) where only a fraction of disk blocks are accessed and the whole working set fits in the storage system cache. It can be seen that in all cases the impact of our ID software is negligible.

7. Related Work

An extensive survey of ID systems was performed by Stefan Axelsson [2]. In this survey, ID systems were classified into three groups according to the detection methods they used: anomaly detection-based, policy-based, and hybrid IDSs. Our prototype IDSs fall into the policy-based category. Hybrid systems combine both above approaches. A large number of ID systems from each of these categories are discussed in this survey.

Another approach for classifying IDSs is based on the part of the system in which the IDS operates. In such a classification most IDSs are classified into one of two groups: network-based IDSs and host-based IDSs. Host-based IDSs such as those presented in [2, 13, 16] operate as a part of the host operating environment and monitor local activities for signs of intrusion. Network-

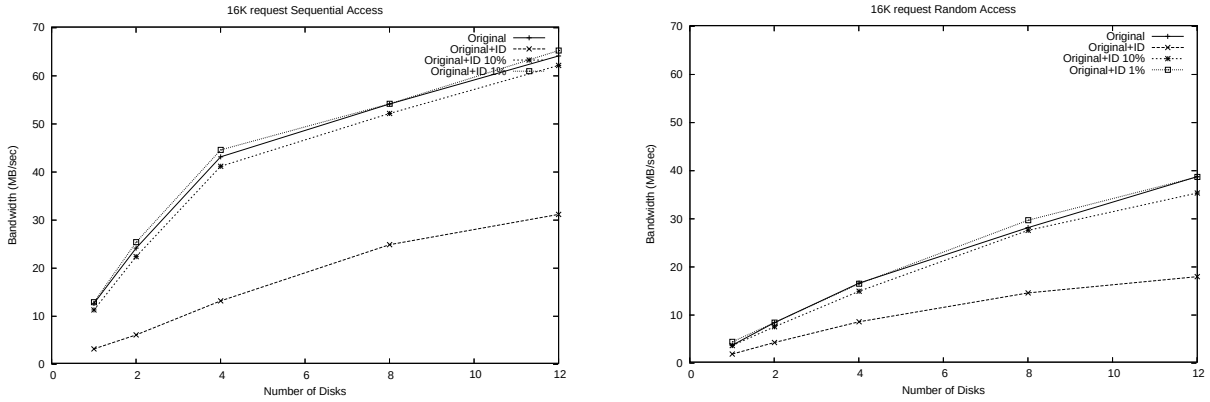

Figure 5. Peak bandwidth for 16-KB data transfers.

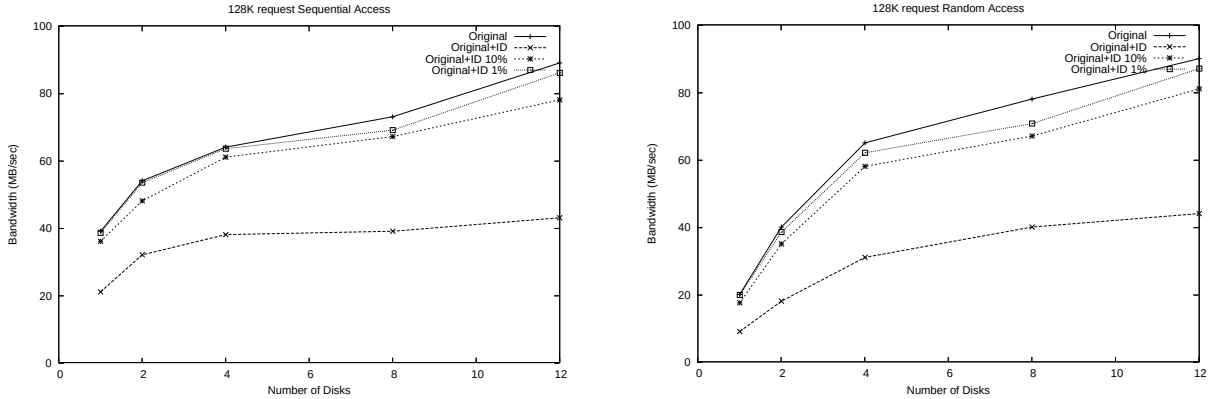

Figure 6. Peak bandwidth for 128-KB data transfers.

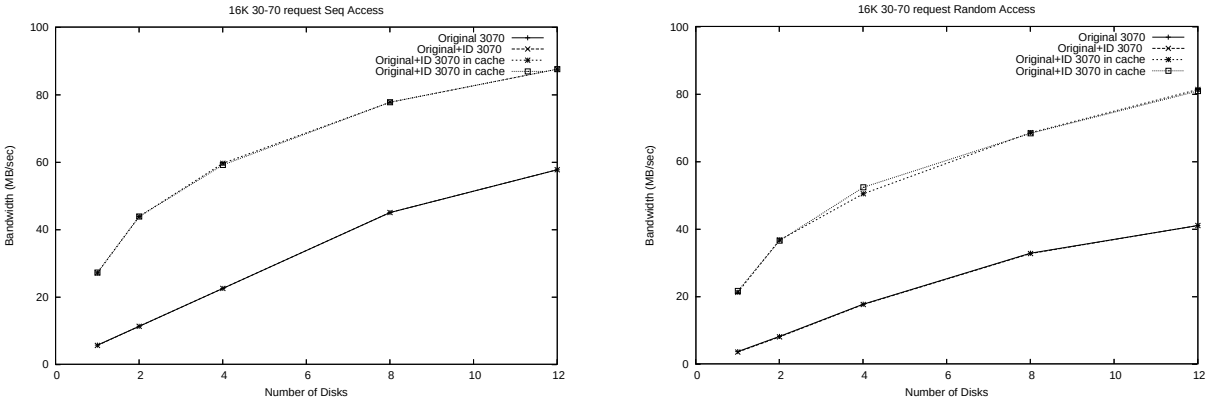

Figure 7. Peak bandwidth for 16-KB data transfers.

based IDSs such as [3, 14] monitor network traffic for signs of suspicious activities.

Storage-based IDSs constitute a new class of IDSs in this classification. These systems and their benefits are discussed extensively in [15]. Furthermore, the implementation and evaluation of a prototype storage-based IDS, embedded in an NFS server, are discussed in this paper. Feasibility of performing ID inside disks is discussed in [7]. Implementation of a disk-based IDS prototype on top of a storage device emulator is presented in this paper. This prototype is similar to the RTSB system with the major difference being that RTSB is

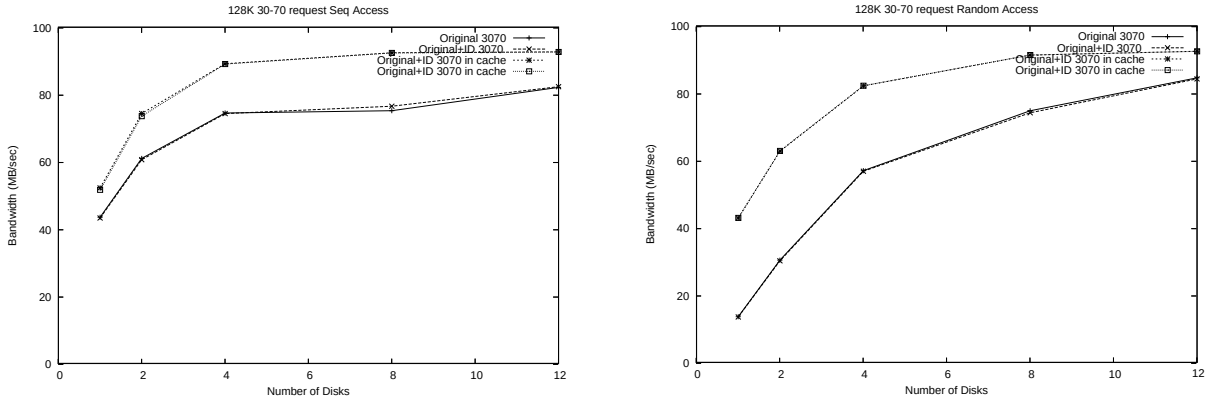

Figure 8. Peak bandwidth for 128-KB data transfers.

implemented as part of a real storage system while the prototype in [7] is implemented in an emulator.

Tripwire [11] is a file system integrity checker that we have used in one of our prototypes. The FLSB system uses Tripwire for detecting signs of any intrusion. FLSB uses features of SAN storage systems in order to perform intrusion detection independent of host systems. It should be also noted that the RTSB system uses rules similar to those used by Tripwire.

Embedding ID techniques into storage systems is in a way similar to other efforts for moving more processing power into storage systems and devices. Active Disks [1, 17, 18], Intelligent Disks [10], and Wise Drives [8] are among efforts advocating storage devices and systems with a higher degree of processing capability in comparison with that of current commercially available disks and storage systems.

8. Conclusions and Future Work

In this paper we presented two storage-based IDSs for block storage environments. We showed that the impact on storage system performance is negligible. We used two different approaches for performing ID. One which is real time and acts at block storage level and another system which does not operate in a real time manner and performs at file system level.

We are working on several approaches to improve on presented systems. We are working on providing support for more file systems by RTSB. We plan to perform more performance analysis for both IDSs. We also plan to extend the work presented in this paper to other types of storage systems such as iSCSI storage systems.

We are also working on improving the scalability of the FLSB by enhancing the storage system to provide an interface through which the IDSs can obtain the list of modified blocks from a particular point in time such as the last time a copy for ID purposes was created. Modern storage systems have certain data structures for their point-in-time copies which can be enhanced for this purpose.

This feature can be implemented by providing the means for creating and initializing a bitmap corresponding to all or certain blocks of LUNs. In the proposed system, before a new flash copy is created, a bitmap data structure is created and initialized. Whenever a block is modified the corresponding bit is set. After each copy, the list of modified blocks (since the previous copy) is obtained and sent to the IDS. Such bitmaps can be created and kept with minimal impact on the performance of the storage system.

When a LUN is being examined by the IDS, the corresponding bitmap is examined to see what files require examination. This of course requires that IDS can perform the file to block translation. The IDS unit can be running on one or more hosts with support for the file systems of interest such that the file to block translation can be performed more easily. In this scheme, files whose corresponding data and metadata blocks are not modified are not checked at all. Those with modified blocks are checked for signs of intrusion. Minimizing the amount of data read for ID purposes also reduces the storage system cache pollution. Furthermore, many storage systems provide the capability of marking LUNs as uncachable. By using such a feature, the impact of accessing copy LUNs for ID on the storage system cache can be kept at a minimum.

Acknowledgments

We would like to thank our shepherd Ahmed Amer as well as the anonymous reviewers for their valuable feedback and insights.

References

[1] A. Acharya, M. Uysal, and J. Saltz. Active Disks: Programming model, algorithms and evaluation. In *Proceedings of Architectural Support for Programming Languages and Operating Systems*, pages 81–91, 1998.

[2] S. Axelsson. Research in intrusion-detection systems: a survey. Technical Report 98-17, Chalmers University of Technology, 1998.

[3] B. Cheswick and S. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley, 1994.

[4] M. D. Flouris and A. Bilas. Clotho: Transparent data versioning at the block I/O level. In *Proceedings of NASA/IEEE Conference on Mass Storage Systems and Technologies*, 2004.

[5] G. A. Gibson and R. V. Meter. Network attached storage architecture. *Communications of the ACM*, 43(11):37–45, 2000.

[6] J. S. Glider, C. F. Fuente, and W. J. Scales. The software architecture of a SAN storage control system. *IBM Systems Journal*, 42(2):232–249, 2003.

[7] J. L. Griffin, A. G. Pennington, J. S. Bucy, D. Choundappan, N. Muralidharan, and G. R. Ganger. On the feasibility of intrusion detection inside workstation disks. Technical Report CMU-PDL-03-106, Carnegie Mellon University, 2003.

[8] G. F. Hughes. Wise drives. *IEEE Spectrum*, pages 37–41, August 2002.

[9] IBM. IBM Total Storage DS4000 Series. http://www.storage.ibm.com/disk/fastt.

[10] K. Keeton, D. A. Patterson, and J. M. Hallerstein. A case for intelligent disks (IDISKs). *SIGMOD Record*, 27(3):42–52, 1998.

[11] G. H. Kim and E. H. Spafford. The design and implementation of Tripwire: A file system integrity checker. In *Proceedings of the 2nd ACM Conference on Security*, 1994.

[12] O. S. D. Lab. Iometer. http://www.iometer.org.

[13] T. F. Lunt and R. Jagannathan. A prototype real-time intrusion-detection expert system. In *Proceedings of IEEE Symposium on Security and Privacy*, 1988.

[14] NFR. NFR Security. http://www.nfr.net/.

[15] A. G. Pennington, J. D. Stunk, J. L. Griffin, C. A. Soules, G. R. Goodson, and G. R. Ganger. Storage-based intrusion detection: Watching storage activity for suspicious behavior. In *Proceedings of the 12th USENIX Security Symposium*, 2003.

[16] P. A. Porras and P. G. Neumann. Emerald: Event monitoring enabling responses to anomalous live disturbances. In *Proceedings of National Information Systems Security Conference*, 1997.

[17] E. Riedel, C. Faloutsos, G. A. Gibson, and D. Nagel. Active disks for large-scale data processing. *IEEE Computer*, pages 368–74, June 2001.

[18] E. Riedel, G. A. Gibson, and C. Faloutsos. Active storage for large-scale data mining and multimedia applications. In *Proceedings of International Conference on Very Large Databases*, pages 62–73, 1998.