Cluster Security with NVisionCC: Process Monitoring by Leveraging Emergent Properties

Gregory A. Koenig, Xin Meng, Adam J. Lee, Michael Treaster, Nadir Kiyanclar, and William Yurcik
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana-Champaign
{*koenig, xinmeng, adamlee, treaster, nadir, byurcik*}*@ncsa.uiuc.edu*

Abstract

We have observed that supercomputing clusters made up of commodity off-the-shelf computers possess emergent properties that are apparent when these systems are considered as an indivisible entity rather than as a collection of independent nodes. By exploiting predicatable characteristics inherent to supercomputing clusters coupled with these emergent properties, we propose several mechanisms by which cluster security may be enhanced. In this paper, we describe NVisionCC, a cluster security tool that monitors processes across cluster nodes and raises alerts when deviations from a predefined profile of expected processes are noted. Additionally, we demonstrate that the monitoring infrastructure used by NVisionCC incurs a negligible performance penalty on the computational and network resources of the cluster.

Keywords: Cluster Security, Process Monitoring, Intrusion Detection

1. Introduction

In recent years, the number of large-scale commodity clusters operating in academic, research, and commercial environments has increased at a rapid pace. In addition, the number of machines contained in each cluster is growing. Now that large-scale commodity clusters are becoming more widely accepted and deployed, the security issues that were ignored during their nascence must now be fully addressed.

In a previous work [12], we describe the security of large-scale clusters as being an *emergent property* that arises from the independent security aspects of the individual cluster nodes and is at the same time irreducible with regard to the overall cluster system. That is, by treating a cluster as a single indivisible entity, we observe properties related to security that are not visible when the cluster is viewed as a collection of hundreds or thousands of individual nodes. For example, consider the case of host-based security mechanisms meant to defeat brute-force attempts at guessing passwords. When an attacker attempts to log in to a machine, he guesses passwords either from a set of often-used passwords or sequentially (e.g., "aaaaa," "aaaab," "aaaac," etc.). After a few unsuccessful attempts, however, the machine typically begins to take evasive action by slowing down the rate at which passwords are checked, by disconnecting the attacker after a few unsuccessful attempts, and by writing messages into the system log. In the case of a cluster of machines that share a common set of users, however, the attacker can employ a different strategy. Instead of focusing his efforts on a single machine, the attacker can instead step through the nodes in the cluster, attempting a brute-force attack against the same username on each node but specifying different ranges of attempted passwords per node. By keeping the number of login attempts below the threshold that an individual node uses to activate its host-based security mechanisms, the attacker may be able to successfully carry out a brute-force password attack against the cluster while simultaneously evading all security mechanisms to prevent such an attack.

Fortunately, there are a number of properties that can be leveraged to monitor security within the context of a cluster that are not possible with traditional enterprise hosts. That is, while clusters are vulnerable to entirely new types of attacks that are not possible with individual hosts in an enterprise, the circumscribed environment of the cluster contains many properties that can be leveraged to monitor the security of the cluster. Perhaps the most important property of clusters is that they are

comprised of a number of homogeneous components. For example, the nodes that make up a cluster can be grouped into categories such as head nodes (used by end-users to access the cluster, compile software, and submit and monitor jobs), compute nodes (allocated to users to run serial or parallel jobs), and storage nodes (containing the disk space used to hold datasets for cluster jobs). Within each category, the steady-state characteristics of every node is nearly identical. For example, because compute nodes in a cluster are typically cloned from the same installed operating system image, each node tends to look the same. That is, the set of processes on a node when no job is running on the machine, the set of open network ports on a machine, and the set of files for system-level binary programs on a machine are identical across all compute nodes. Further, deviations from these expected characteristics tend to happen only in well-defined ways, such as when a batch job that is launched on a compute node causes the set of processes on the node to deviate from the expected set. These deviations, however, can be correlated with other sources of information about the running state of the cluster, for example by querying the cluster batch job scheduler. We believe that these unique characteristics of the cluster can be leveraged to monitor security.

By examining various sources of information about the running state of a cluster and searching for deviations from an expected steady-state signature, potential security penetrations can be identified. We have identified four such sources of information that are of interest.

1. *Running processes* – A change in the process list of an idle node from the expected steady-state set of processes can indicate that an attacker has gained access to the node and is attempting some kind of malicious activity.

2. *Open network ports* – Finding unexpected open ports on a node suggests that an attacker may be running network software which opens ports to facilitate malicious activity.

3. *Network traffic patterns* – The appearance of network traffic from a supposedly idle node could be warning signs that a compromise has taken place.

4. *Critical file contents* – Any modification to system-level program binaries suggests that an intruder may have altered system software to serve some agenda such as capturing passwords.

In this paper, we describe our work in developing NVisionCC, a tool specifically focused on monitoring security within the context of a cluster [14]. This paper focuses on our work surrounding process monitoring. We believe that process monitoring is perhaps the most far-reaching source of information about cluster security because an attacker cannot gain access to cluster nodes without running at least some processes. The appearance of unexpected processes on a cluster node is a strong indicator that the security of the node has been compromised.

The remainder of this paper is organized as follows: We begin by describing related work in process monitoring in Section 2. Section 3 describes the state-of-the-art in cluster security technology. Section 4 discusses the design and implementation of NVisionCC, and Section 5 presents a performance analysis of the tool and experiences using process monitoring to aid in identifying security incidents. Finally, we summarize our work and propose future directions for additional research in Section 6.

2. Related Work

Several research projects described in the literature are related to the work that we present in this paper. In this section, we reference various efforts and show differences between these tools and our prototype process monitoring system.

The Distributed Security Infrastructure (DSI) [10, 9] is a security framework designed to operate on carrier-class clusters used in the telecommunications industry. Requirements for the operation of such a cluster include high availability and reliability, and scalable performance. DSI takes advantage of clustering to provide the failover capabilities needed to maintain high availability, but also provides for security via encrypted communication between Security Manager processes running on all cluster nodes, as well as on a central monitoring console.

In contrast to tools such as the DSI for a well-defined environment, we next consider tools that are designed to monitor general-purpose computational clusters. Four popular cluster monitoring tools are Ganglia, Supermon, Rvision, and Clumon.

Ganglia [5] is a scalable, distributed monitoring package for high performance cluster systems. It uses a real-time monitoring framework with a hierarchical design to track utilization data for a single cluster or for a grid of clusters dispersed over a wide area network. Data are represented via XML descriptions and exchanged via XDR binary transfers. Under Ganglia, each cluster node runs the Ganglia Monitoring Daemon (gmond)

which multicasts changes in node state and responds to queries requesting an XML description of cluster status. The Ganglia Meta Daemon (gmetad) pulls together XML reports from one or more clusters, stores the information in Round Robin databases, and exports summary XML information to a web front-end used to monitor cluster status. Ganglia offers a scalable tool for cluster monitoring but does not seem entirely suited to our purposes due to the fact that it collects data concerning CPU utilization of cluster nodes in order to point out to a cluster administrator (or to job placement software) which nodes in a cluster are utilized. In contrast, our requirements are focused on a tool for monitoring the large set of processes running across a cluster.

Supermon [11] is a flexible set of tools for high speed, scalable cluster monitoring. The architecture of Supermon is similar to that of Ganglia. Each node has a loadable kernel module that obtains kernel-level performance data about the machine and reports the data to a per-node data server (mon). A data concentrator (Supermon) composes samples from many nodes into a single data sample. A novel concept in Supermon is that data is represented internally as S-expressions (similar to those used in the LISP programming language) and because of this, virtually any amount of data nesting can be represented. Supermon is designed to collect performance metrics at an extremely high sampling rate, with several thousand samples per second supported. Unfortunately, like Ganglia, Supermon is focused on collecting kernel-oriented performance data such as CPU utilization, page fault rates, and network I/O statistics.

RVision [3] (Remote Vision) is an open architecture, highly configurable tool for cluster monitoring. It provides flexibility in selecting events to monitor and allows users to expand the monitoring capabilities with self-defined procedures. Such procedures can be used to monitor virtually any system or hardware event. Another important feature of RVision is its flexibility to facilitate the monitoring of distinct clusters, clusters of clusters, and heterogeneous clusters. RVision distinguishes itself from other available tools for cluster monitoring because of its open architecture, high configurability, and low performance intrusion. While RVision is highly configurable and may be the closest match to our requirements, it appears that most of the current work with this tool centers on collecting kernel-oriented performance data such as CPU utilization and memory and swap usage statistics.

Clumon [2] is an open source cluster performance monitoring system developed for monitoring Linux-based clusters at the National Center for Supercomputing Applications. Not only does it provide system and performance information for each individual node but it also collects job information from the job scheduler. By integrating those two different kinds of data and storing them into a central database, Clumon provides a unique visual overview of the current state of a cluster [13]. We leverage the existing technical features and installed user base of the Clumon software by developing NVisionCC as a plugin to this system.

3. State-of-the-Art Practices in Cluster Security

Clusters are typically the highest-value asset within an organization's environment and as such should receive priority security protection. While existing security tools for enterprise computing have been applied to the cluster environment, they do not address emergent security properties such that clusters are being protected as multiple separate machines as opposed to a single cohesive unit. However, the dominant factor is the trade-off between security and high-performance. When faced with the task of balancing the security of a high-value computing asset against the needs of users, cluster administrators generally favor minimal security and often disable security functions in favor of performance and convenience. What follows is a list of presently used techniques used to guard high-performance clusters.

3.1. Network Considerations

To reduce the risk of unauthorized access, a site can adopt an enclosed cluster design. In extreme cases, this can be achieved by keeping a cluster on a physically isolated network. A more common and convenient approach is to limit direct user access to dedicated login or head nodes. The compute nodes can then be placed in a private non-routable address space, or alternately kept behind a firewall. In situations where it is feasible, this approach limits the scope of outside threats, and correspondingly lessens the work of administrators. A Grid computing environment can present problems with this type of enclosed cluster, however, as Grid jobs can be allocated nodes on multiple clusters, all of which may have to intercommunicate. A more open design, with all nodes Internet accessible, is necessary to support this functionality.

To prevent a potentially compromised machine from sniffing cluster communications, no unmanaged machines or machines with different security models

should be allowed on the same network segments as any cluster computers.

3.2. Centralized Software Configuration

A tightly-constrained software environment on clusters is important for both performance and security. Only specific software should be installed on clusters and permitted programs must be current and patched. Recognizing the distinct types of cluster nodes as equivalence classes with regards to their configuration can ease administration and bolster cluster security. By restricting the available software on a given node type, fewer computers may be affected by the update of a given package. Moreover, certain classes of cluster nodes may receive higher priority based on the security impact of a compromise on them. Central configuration management can be implemented by either network mounting common files or by utilizing a mechanism for automatic distribution of such files to various subsets of the cluster as needed. Tools such as Cfengine[1], which exist for the purpose of centralized configuration management and repair in a general network setting, have been adapted for use in a cluster environment.

3.3. Authentication

Authentication on cluster machines is another area of security concern. Traditional means of authentication, like /etc/passwd and shadow files, present some configuration issues in any distributed system. The number of machines in a large cluster can create a problem with synchronizing these files in a timely manner. Therefore, when new users are added, or someone has a password change, all machines need to be updated.

Public key mechanisms such as RSA authentication using SSH provide another means of security. Here the user manages their own keys which are kept in their home directories. Public key systems such as these, while cryptographically secure, rely on the users protecting their private key, and adding additional protection with the use of a passphrase on their key. Many users will forego this last step, instead preferring the ease of a passwordless login.

Centralized authentication methods like Kerberos are typically used in cluster environments. Using a service like Kerberos users can authenticate once and then have access to any cluster resource they are authorized to use. Kerberos and related systems also provide better protection of user authenticators and can enforce varying policies on passwords for users (length, character classes, expiration, etc.).

PKI systems provide another means for maintaining cluster security. It is becoming more common in cluster environments for users to authenticate with something like X.509 certificates to authenticate to services. One of the current drawbacks of PKI is that you are placing the responsibility on the users to protect their keys, and users may not be very security conscious.

It is always possible for an intruder to masquerade as an authorized user. This can be achieved by exploiting protocol flaws, or by local keyboard sniffing for passwords. Thus root access to a cluster should demand a higher standard of security. Under no circumstances should remote root logins be permitted, only direct console access.

3.4. Intrusion Detection Systems

Host-Based Intrusion Detection Systems (HIDS) such as Tripwire [6] are commonly used to monitor high-value assets such as clusters. Tripwire is typically configured to report file and operating systems changes once every 24 hours. While Tripwire is reliable, it has usability problems due to the cryptic nature of its reports as well as false positives. In the context of clusters, Tripwire makes no priority distinctions between different nodes, so that security staff have a difficult time obtaining situational awareness of file/operating system changes when considering a cluster as one system. Since Tripwire reports all file/operating system changes, many of the alerts it generates are actually legitimate user or system administration activity. Faced with a large volume of false positives, a cluster administrator making changes across a large number of nodes will often disable an HIDS for significant periods of time.

While an HIDS is capable of detect signs of intrusion, ultimately their reports must be validated since it is possible (and likely) that upon a successful root-level compromise, an intruder will replace the binaries used by that system. Network-based Intrusion Detection Systems (NIDS) can be used to verify the output from individual hosts, in addition to scanning for generally suspicious traffic. NIDS passively monitor network flows, and can be configured to send alerts if traffic matching attack signatures are detected. Neither HIDS or NIDS have been adapted for the unique cluster environment.

3.5. Packet Filtering

It is possible to individually firewall each host to specifically tailor cluster node access. Pfilter [4] compiles security policies into either iptables or ipchains rule

sets for Linux. While the advantages in using an automated tool like Pfilter in larger cluster environments may be clear, cluster administrators are often reluctant to firewall cluster nodes aggressively due to concerns of either performance or user inconvenience. In a Grid-enabled cluster, or a cluster where the policy is to allow users relatively free reign in the used of allocated nodes, firewalling individual nodes may be unacceptable without some provision for dynamically adjusting firewall rules on a per-host basis.

3.6. Summary of Best Practices

In this section we have highlighted the primary techniques currently used to protect high-performance clusters. Other security processes, such as centralized logging and proactive vulnerability scanning, are also used in HPC environments, as are other secondary processes we may not have mentioned (our list is not intended to be an exhaustive one). We emphasize, however, that the techniques we have mentioned do not take advantage of the unique emergent properties which arise when a cluster is viewed as a cohesive whole.

4. Design

The development of our tool, NVisionCC, has been our attempt at improving upon the current state-of-the-art in cluster monitoring by focusing on the emergent properties of large-scale commodity clusters. By leveraging the emergent properties of the cluster, we can glean valuable information from the context that is ignored by current solutions, as these tend to be extensions of enterprise monitoring tools and practices.

In order for NVisionCC to be successful, it must first be effective and easy to use. In addition, the cluster monitoring community needs to accept the tool and incorporate it into their security monitoring practices. To aid in these goals, NVisionCC was designed around five key requirements:

1. *Performance Impact* – Our monitoring framework had to have a minimal impact on performance of the cluster compute nodes. Providing compute cycles is the primary goal of a cluster system, and any performance penalty incurred by running the monitoring software would detract from the value of the tool. Additionally, cluster systems are increasing in size, with the largest systems comprising thousands of nodes. NVisionCC had to perform equally well on small clusters, the massive systems of today, and the even larger systems of the future while creating a negligible load on the cluster infrastructure.

2. *Central Control* – The tool had to provide a means of central control to monitor all nodes through a single machine. Although a distributed control mechanism would be efficient from the system perspective, it would not be efficient from the perspective of an administrator. He would have to manually access each individual machine in a cluster to view monitoring data or to reconfigure the process profile. Additionally, gathering the data in a centralized location allows a user to visualize data across the entire cluster and therefore observe the emergent properties of the system.

3. *Leverage Existing Software* – Although no security tools exist specifically designed for cluster environments, open-source tools do exist which, when combined, provide much of the functionality required by our design. We wanted to maximize the reuse of these tools and minimize the need to write the code that ties these tools together into a single, coherent package.

4. *Configurability* – In order to make the software as widely applicable as possible, the software had to be easily customizable to run on different types of nodes on different types of clusters. This is a step beyond cross-platform portability. Clusters based on the same architecture are likely to have different usage characteristics or administrative practices that would need to be accounted for in any security solution.

5. *Effectiveness* – As a security tool, it is crucial that NVisionCC be effective at detecting intrusions while minimizing the rate of false positives. To facilitate the tracking of an attack to its source, the software has to log historical data such that a cluster administrator may investigate previous events in order to study the various stages of an intrusion.

By designing to meet these five criteria, we believe that our tool will have low overhead, high effectiveness, and low cost. As all of the building blocks used to compose our system are open source, NVisionCC is available at no cost to any entities wishing to monitor the security-state of their clusters. Since these groups can evaluate and use NVisionCC for free, the adoption of NVisionCC can occur at a rapid pace, thereby improving the overall security of today's large-scale commodity clusters.

4.1. Performance Co-Pilot (PCP)

Performance Co-Pilot (PCP) [7] is an Open Source framework and set of services for performance monitoring and performance management developed by SGI. It uses a distributed architecture where a daemon on each node monitors a variety of performance-related system-level metrics about the node and transfers this data to an application-specific central data-collection server. This provides high scalability and makes the system ideal for monitoring large numbers of hosts. The server can theoretically be a bottleneck on the monitoring system, but since this machine is typically located outside the cluster it does not interfere with cluster performance.

The default framework supports the gathering of hundreds of different metrics about a machine, ranging from data about user programs to statistics on low-level performance. The framework also provides an API for developing customized monitoring agents to gather data about user-defined metrics for when the default metrics are inadequate.

Each node runs a single-threaded PCP service, `pmcd`, which listens for requests from a TCP/IP connection. When queried, the service then gathers the requested data from the node and returns it over the same connection. Little state is maintained between requests. A concise encoding of the data keeps overhead of processing and network traffic to a minimum. Since all data is sent over a TCP/IP network, the traffic does not interfere with a cluster system employing some other network for interprocess communication, such as Myrinet.

The user can specify the polling frequency of the nodes to adjust the precision of the data as well as the performance impact on the system. In Section 5 we show that this impact is minimal even when polling as often as once per second.

4.2. Alert Criteria

NVisionCC relies on profiles of normal system states to detect anomalous behavior. For process monitoring, a cluster administrator must create a list of processes for each classification of node (head node, storage node, compute node, management node) that are expected to run on that type of node when it is idle. This defines the normal, expected behavior of a node. When deviations from this list are noted, an alert is raised.

There are four possible types of deviations. First, an unexpected process could be running. Second, extra instances of an expected process could be found. Third, an expected process could be missing. Fourth, an expected process could be replaced by a suspicious doppelganger, for example `/sbin/init` might be replaced with `sbin/init`. Although these profile anomalies can occur through benign means such as accidents or improper termination of a process, even in these cases an administrator would probably prefer to be notified of the problem to ensure consistent system performance over time.

These types of deviations are easy to detect on an idle node where the profile of legitimate processes is known. However, on an active node this is more difficult. A cluster administrator does not want to be alerted every time a user runs a job on the system. To detect anomalies concerning extra running processes in this situation, NVisionCC correlates the process list with the job scheduling system that manages the cluster. If an extra process is noticed on an active node, the owner of the process is noted. The system then queries the job scheduler for the user to whom the node is assigned. Since no one else should have access to the node, if the user of the anomalous process does not match the assigned user of the node, an alert is raised. If the user of the anomalous process matches the user of the node, the process is assumed to be the legitimate work of the user and is ignored.

4.3. Process Monitoring Architecture

The process monitoring architecture consists of three separate components. The information collector gathers process data from cluster nodes and stores it in a central database. The data analyzer retrieves the data from this database and searches for deviations from the user-defined profiles for each type of node. The visualization client combines the data from the database with the results from the data analyzer to provide a user with a visualization of the cluster state.

4.3.1 Information Collector

The information collector component is responsible for gathering the process data from each node of the cluster and storing it in a centralized repository for analysis. It uses the `pmcd` daemon provided by PCP to gather information about each node in the cluster. NVisionCC has a server process running on a monitoring machine separate from the cluster which polls each `pmcd` instance at regular intervals to retrieve the data for each node. Collected data consists of the process name, process owner, process ID (PID), and parent process ID (PPID), as shown in Table 1. The server process stores the new records in a database for later analysis.

Host	PID	Process Name	PPID
compute001	1324	001324 /sbin/agetty	1
compute001	7106	007016 /usr/local/pbs/ia64/sbin/pbs_mom	1
compute001	11931	011931 /bin/sh	7106
compute001	11937	011937 /bin/sh	7106

Table 1. Collected Process Information

4.3.2 Data Analyzer

The process analyzer is built around the idea that "normal" behavior for a cluster node is easy to define, and that deviations from this behavior are easy to detect.

The user must provide a profile which lists the expected processes for each classification of node. Separating types of nodes is important because nodes will have different usage characteristics depending on their purpose, and this will be reflected in the processes that run on that type of node. Common types of nodes are listed in Table 2.

A profile is a table of information describing the process table signature of a particular classification of node when it is in a normal state. This information consists of a list of processes and an expected number of instances of each process on the machine. NVisionCC allows all profiles to be combined into a single master table, as shown in Table 3.

Some processes, such as `mysqld` in Table 3, should appear with the number of instances falling within a certain range. In this case, the minimum and maximum number of instances can be specified in the profile. If a process should always appear with exactly the same number of instances, the minimum and the maximum should both be set to the same value. This is illustrated by the `pbs_mom` process in the example. If a process should never appear on a certain type of node, the maximum instance count would be set to 0, as represented by `lpd` in the table.

With these data, NVisionCC is able to detect four types of security problems.

1. The monitoring system can detect if an explicitly forbidden process is running. To extend the previous example, if an instance of sendmail is found running on a compute node, this will immediately be noticed when the sendmail process is seen to have a max_instances of 0 in the profile. Any running process with the same name as a process denied in this manner in the profile will be flagged as a bad process.

2. The tool can detect suspicious processes that are not explicitly forbidden. These are processes that do not appear at all in the profile, and further diagnosis is required to see if it is malicious or not. One technique of automating this diagnosis is to collaborate with the job scheduler for the cluster. If a node is supposed to be idle, then any suspicious process is likely cause for alarm. If a node is allocated to a particular user, any process owned by that user is likely benign, while processes owned by any other user are likely malicious.

   This analysis also detects doppelganger processes, where a legitimate process is replaced by a malignant process with a similar name. For example, `sbin/init` might appear in place of `/sbin/init`.

3. It can detect if a required process is missing. If a process is listed in the profile with a minimum instance count greater than zero, then that process is expected to appear in the list of running processes at all times. A process is missing if no running process matches the name of a required process. This would occur if an attacker disabled certain processes to compromise the system. It might also occur as a result of an unexpected system error such as a process crash.

4. If the number of instances of a certain process is outside the allowable range but greater than zero, a process of unknown origin will be detected and an alert will be raised. For example, a hacker could replace the executable for an expected, critical, system process with a malicious version while leaving the existing process running. He could then run his new version, adding an extra entry for the process to the process list.

4.3.3 Visualization Client

Our design suggests using a Web-based visualization client to enable a user to quickly view the state of a cluster from any other machine without requiring the installation of special software. Figure 1 depicts the interface used by NVisionCC. It shows a graphical summary of

Type	Description
Monitor	Cluster Monitor Nodes
Host	Compute Nodes
User	User Nodes
Storage	Storage Nodes
Management	Cluster Management Nodes

Table 2. Types of Cluster Nodes

Rule ID	Type	Process Name	Min	Max
1	Monitor	mysqld	3	8
2	Monitor	lpd	0	0
...				
8	Host	lpd	0	0
9	Host	pbs_mom	1	1
...				

Table 3. Example Profile

each node on the cluster with a visual indication of the state of the node. Different icons are used to represent normal, suspicious, bad, and critical states on a node.

The collected data can also be presented in a variety of other ways to assist the administrator in investigating alerts efficiently. The visualization client can create a list of all anomalies presently detected on the system, organized by severity. This is demonstrated in Figure 2. This allows the administrator to quickly obtain a more detailed overview of the system with the alerts prioritized by severity.

Additionally, the system can provide the user with access to the raw process data collected for a particular node, presented as the current process list for the node. This allows the administrator to look for patterns in the process list that might not be apparent when viewing only the anomalies detected by the system. An example of this node-level view is shown in Figure 3.

4.4. Clumon

Clumon is a CLUster MONitoring application deployed on many large cluster systems at a number of sites. It provides much of the software foundation for NVisionCC. Clumon uses PCP to gather performance statistics such as CPU load and memory usage from cluster nodes. It stores this information in a central data repository, and a web-based visualization tool retrieves the data and presents it to a user.

NVisionCC is implemented as a plugin to the Clumon cluster monitoring tool. The information collector component uses Clumon's process monitoring infrastructure and central server, but requests extra PCP metrics beyond that Clumon usually gathers. The data analyzer component retrieves the data from Clumon's database, generates any necessary alerts, then displays those to a web view based on that used by Clumon.

5. Performance Impact

The value of a cluster is directly proportional to the computational power it is able to provide to its users. Any security solution that noticeably impacts either the computational speed or the network bandwidth of the cluster will not be deployed in production environments. The `pmcd` daemon process used by NVisionCC has a relatively small footprint on the machines in question: less than 1% of the CPU and under one megabyte of core memory are utilized by `pmcd` while in its idle state. However, its effect on the performance of jobs submitted to the cluster is unclear. In this section, we examine the impact of the process monitoring used by NVisionCC on both CPU and bandwidth performance.

5.1. Experimental Environment

We collected performance analysis data using a 13-node Linux cluster intended primarily for small-scale production use. Each node contains two 2.0 GHz Intel Xeon processors with Hyperthreading (a total of four logical processors) in symmetric multiprocessing (SMP) configuration. The nodes each have one gigabyte of memory shared between both processors, and each runs Red Hat Linux with kernel 2.4.25smp. The nodes are interconnected with both Gigabit Ethernet and Myrinet, with Myrinet as the primary means of MPI communication. On average, each node has between 40 and 50 processes running at any given time.

To evaluate the performance of the cluster under a scientific workload, we used the High Performance Linpack (HPL) software package [8]. This benchmark runs on distributed-memory clusters and solves random dense linear systems in 64-bit precision. HPL tests the speed of the computation and the accuracy of the results generated, making it an effective tool for judging the overall performance of a cluster under a scientific workload.

Our performance tests ran HPL on 16 logical processors spread across four nodes. Two test cases were run. The first tested the impact on processor cycles delivered to scientific jobs. The second tested the impact on the throughput provided by the cluster network. NVisionCC was running on the same nodes monitoring 16 PCP met-

Figure 1. Visualization Client – Main View

Cluster Hosts Running suspicious process

Time	Host Name	Process Name
2003-11-12 22:59:04	titan068	root
2003-11-12 22:59:04	titan068	-bash
2003-11-12 22:59:04	titan071	/u/ncsa/nrudins/titan.stats/pokeutmp
2003-11-12 22:59:06	titan095	/u/ncsa/nrudins/titan.stats/pokeutmp
2003-11-12 22:59:09	titan123	/u/ncsa/nrudins/titan.stats/pokeutmp

Cluster Hosts Running process out of range - Too many

Time	Host Name	Process Name	Current No	Normal Range
2003-11-12 22:59:04	titan079	init	2	1 - 1

Cluster Hosts Process Should be running but NOT

Time	Host Name	Process Name
2003-11-12 22:59:04	titan068	/sbin/agetty
2003-11-12 22:59:04	titan079	/sbin/agetty
2003-11-12 22:59:10	titan135	/usr/local/pbs/ia64/sbin/pbs_mom

Figure 2. Visualization Client – Process Alert View

Time	Process Name	Status	Current No.	Normal Range	Suggested Action
2003-11-12 23:14:04	init	> Max	2	1 - 1	Kill some of the processes
2003-11-12 23:14:04	(keventd)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(ksoftirqd_CPU0)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(ksoftirqd_CPU1)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(kswapd)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(bdflush)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(kupdated)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(scsi_eh_0)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(scsi_eh_1)	Normal	1	1 - 1	N/A
2003-11-12 23:14:04	(mdrecoveryd)	Normal	1	0 - 1	N/A
2003-11-12 23:14:04	(kjournald)	Normal	4	1 - 4	N/A

Figure 3. Visualization Client - Node View

	Metric	PCP Data
1	OS name	kernel.uname.sysname
2	OS version	kernel.uname.version
3	OS release	kernel.uname.release
4	System Load	kernel.all.load
5	Free Memory	mem.freemem
6	Total Syscalls	kernel.all.syscall
7	% Filesys Used	filesys.full
8	Filesys Free	filesys.free
9	Filesys Used	filesys.used
10	Filesys Capacity	filesys.capacity
11	Mount Points	filesys.mountdir
12	Number of CPUs	hinv.ncpu
13	Processor Speed	hinv.cpu.clock
14	Number of Disks	hinv.ndisk
15	PPID[1]	proc.psinfo.ppid
16	Defunct Processes	proc.runq.defunct

Table 4. PCP Metrics Monitored by NVisionCC

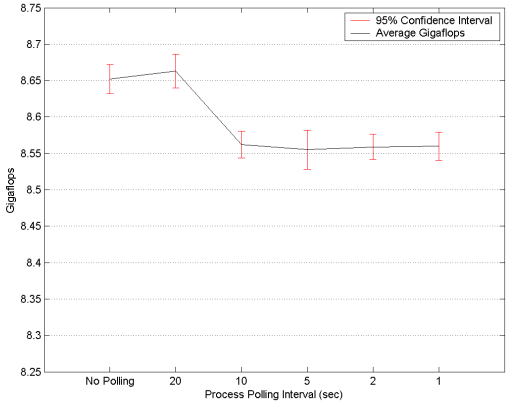

Figure 4. Cluster performance with process monitoring (IPC over Myrinet, 16 PCP metrics)

rics, shown in Table 4. The polling frequency used by NVisionCC ranged from one second to one minute, and the benchmark was also executed with NVisionCC disabled. We did not test frequencies of less than one request per second because it is unlikely that the analysis machine could process the incoming data at that speed, nor is it necessary for NVisionCC to work. Each polling frequency was tested twenty times for the CPU test and twenty times for the network test to achieve a 95% confidence level.

5.2. Processor Impact

First, we tested the impact of the process monitoring used by NVisionCC on the CPU cycles delivered of HPL. If the process monitoring consumed a significant amount of processor cycles, the HPL results should degrade as the frequency of polling is increased and more CPU is allocated to the PCP monitor. During these tests, processor performance was isolated from network performance by routing the interprocess communication for HPL the cluster's Myrinet network while the data collected by NVisionCC was transmitted over the Gigabit Ethernet network. This prevented the monitoring traffic from congesting the links used by HPL so its communications could proceed unhindered.

Figure 4 summarizes the results of these tests. It indicates that collecting the set of running processes from hosts on the cluster at intervals as short as one second does not significantly impact the performance of the cluster on this benchmark. There is a performance difference of less than 0.1 gigaflops between the case in which no process monitoring is occurring and the case in which polling occurs once per second. On our test cluster, this accounts for a degradation of less than 1.16%.

5.3. Network Impact

Next, we tested the impact of the process monitoring on the network. To measure this interference, the interprocess communication for HPL and the monitoring data gathered by NVisionCC were both routed over the Gigabit Ethernet network. If PCP generated a large amount of network congestion, the HPL results would suffer due to the increased latency in its communication. Not all jobs submitted to a cluster will run over the Myrinet, and some clusters do not have Myrinet networks at all, so it was important to test whether or not the process monitoring would interfere with other traffic on the network links.

Figure 5 shows the results from this experiment. As in the CPU tests, the graph shows the results of running HPL while the process lists of the hosts in the cluster are polled at varying intervals. The average performance difference between the test with no polling and the test with the most frequent polling was an increase of .13 gigaflops, or +0.2% on our test cluster. This perfor-

[1]This metric provides the process name, process ID, and process parent ID.

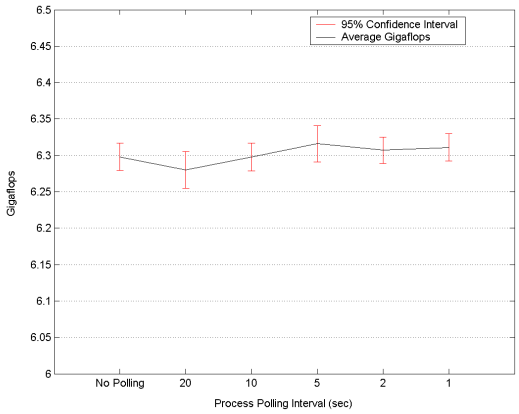

Figure 5. Cluster performance with process monitoring (IPC over Gigabit Ethernet, 16 PCP metrics)

mance improvement is smaller than the confidence interval measured, so it cannot be considered conclusive. However, these results do indicate that polling the process lists of cluster nodes at intervals as small as one second had no significant performance impact on the network infrastructure of the cluster.

It is possible that as the number of machines on the network grows, the increase in process data being transmitted could cause contention on the Ethernet bus, though this has not been tested. Should this problem arise, however, the collection of process lists could be organized in a more hierarchical fashion to reduce the number of messages transmitted by a logarithmic factor. We are encouraged by the fact that the Clumon framework has been used to monitor (without the NVisionCC plugin) aspects of a 1450+ node cluster using PCP without causing bandwidth problems.

5.4. Preliminary Results

In addition to the 13-node Linux cluster, we have also deployed NVisionCC on a 512-node IA-32 Linux cluster. Although we have not formally collected performance data on this platform, we have made some preliminary observations.

We expected to find approximately 40-50 processes running on each node. We were surprised to find that there were close to 100 processes on each node. The distributed nature of the monitoring resulted in negligible load on the cluster nodes. However, the large number of processes per node combined with the large total number of nodes resulted in a substantial load on the central data-collecting machine, and the analysis computation was very slow. We are currently considering ways of improving the efficiency of the data analyzer component.

This cluster violated our assumption that nodes of the same classification would have homogeneous hardware and software configurations. There are a small number of storage nodes that double as compute nodes, and these have additional processes running to fulfill the storage functions. Additionally, for an unknown reason some processes were running under names that differed from those specified in the profile. These two anomalies resulted in NVisionCC generating many false positives. Future work on this point will focus on determining the cause of these diversions from the expected homogeneous view and reconciling the situation within the monitoring framework.

6. Conclusions and Future Work

In this paper, we have presented a mechanism for detecting compromised nodes in large-scale commodity clusters via process monitoring. As opposed to the current state-of-the-art in enterprise security monitoring, our system looks at the cluster as a whole rather than as a collection of independent nodes and makes use of the emergent properties of a cluster environment.

In Section 4, we presented our prototype process monitoring tool, NVisionCC. NVisionCC is plug-in for the Clumon cluster monitoring package, which uses Performance Co-Pilot (PCP) to collect statistics about the nodes in the cluster. We conducted tests to measure the impact of process monitoring on the performance of the cluster under a scientific load and found that the list of running processes on each host could be polled as frequently as once per second without significantly affecting the performance of the cluster.

In the future, we plan to augment NVisionCC to monitor the other emergent properties mentioned in Section 1. Development is underway to visually flag nodes that have abnormal ports open while in their idle state. This type of check will allow us to detect nodes that are listening for abnormal network connections while in an unallocated state. Additionally, we are implementing a tool that allows us to check for altered system files on each cluster node.

By combining examining multiple emergent properties of cluster nodes, we increase the probability of successfully detecting an intrusion by decreasing the

amount of freedom the attacker has once a host is compromised. For instance, if an attacker compromises a host and is able to alter the `pmcd` daemon, illegal processes could be masked from our process monitor. However, if the attacker attempted to contact other nodes in the cluster or alter any system files, the intrusion could be detected. The interplay of the emergent properties of the cluster represents a great advantage to security administrators attempting to monitor modern clusters.

7. Acknowledgments

A special thanks for Joseph P. (Joshi) Fullop, IV of NCSA for partnering with us on this project. Joshi is focused on monitoring cluster performance and we leverage off of the software he developed, CLUMON, as a plug-in for monitoring cluster security. Other members of our NCSA security research group have also made significant, although indirect, contributions to this paper: (in alphabetical order) Cristina Abad, Jim Barlow, Jim Basney, Rafael Bonilla, Tim Brooks, DeVaris Brown, Nada Cagle, Peter Enstrom, Neil Gorsuch, Kiran Lakkaraju, Yifan Li, Chao Liu, Jeff Rosendale, Ken Sartain, Aashish Sharma, Adam Slagell, Shyama Sridharan, Jun Wang, and Xiaoxin Yin. We would also like to acknowledge guidance and instrumental details from the following NCSA high performance cluster system administrators: (in alphabetical order) Tim Bouvet (Copper), Karen Fernsler (Tungsten), Eric Kretzer (Titan & Platinum), Dan Lapine (Mercury), and Thomas E. Roney (TRECC cluster) under the leadership of their Technical Program Manager Wayne Louis Hoyenga. Comments from the TeraGrid Security Working Group have also proved useful.

References

[1] M. Burgess. Cluster Management with GNU cfengine. In *Newsletter of the IEEE Computer Society's Task Force on Cluster Computing*, 2002.

[2] Clumon project website. http://clumon.ncsa.uiuc.edu/main.html.

[3] T. C. Ferreto, C. A. F. De Rose, and L. De Rose. RVision: An Open and High Configurable Tool for Cluster Monitoring. In *IEEE International Workshop on Cluster Computing*, 2002.

[4] N. Gorsuch. Linux Cluster Security. In *Linux Revolution Conference*, 2001.

[5] F. Hoffman. Cluster Monitoring with Ganglia. *Linux Magazine*, 2003.

[6] Gene H. Kim and Eugene H. Spafford. The Design and Implementation of Tripwire: A File System Integrity Checker. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 18–29. ACM Press, 1994.

[7] Performance co-pilot website. http://oss.sgi.com/projects/pcp.

[8] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary. HPL – A Portable Implementation of the High–Performance Linpack Benchmark for Distributed–Memory Computers. Web Page, January 2004. http://www.netlib.org/benchmark/hpl/.

[9] M. Pourzandi, I. Haddad, C. Levert, M. Zakrewski, and M. Dagenais. A Distributed Security Infrastructure for Carrier Class Linux Clusters. In *Ottawa Linux Symposium*, 2002.

[10] M. Pourzandi, I. Haddad, C. Levert, M. Zakrewski, and M. Dagenais. A New Architecture for Secure Carrier-Class Clusters. In *IEEE International Workshop on Cluster Computing*, 2002.

[11] M. Sottile and R. Minnich. Supermon: A High-Speed Cluster Monitoring System. In *IEEE International Workshop on Cluster Computing*, 2002.

[12] William Yurcik, Gregory A. Koenig, Xin Meng, and Joseph Greenseid. Cluster Security as a Unique Problem with Emergent Properties: Issues and Techniques. In *The 5th LCI International Conference on Linux Clusters: The HPC Revolution 2004*, May 2004.

[13] William Yurcik, Xin Meng, and Nadir Kiyanclar. NVisionCC: A Visualization Framework for High Performance Cluster Security. In *ACM CCS Workship on Visualization and Data Mining for Computer Security (VIZSEC/DMSEC)*, 2004.

[14] William Yurcik, Xin Meng, and Gregory A. Koenig. A Cluster Process Monitoring Tool for Intrusion Detection: Proof-of-Concept. In *29th IEEE Local Computer Networks (LCN)*, 2004.