Searching for Open Windows and Unlocked Doors: Port Scanning in Large-Scale Commodity Clusters

Adam J. Lee[†‡], Gregory A. Koenig[†‡], Xin Meng[†], and William Yurcik[†]
[†]National Center for Supercomputing Applications
[‡]Department of Computer Science
University of Illinois, Urbana-Champaign
Champaign, IL 61820
{adamlee, koenig, xinmeng, byurcik}@ncsa.uiuc.edu

Abstract

Current methods for monitoring the security of large-scale commodity clusters tend to treat these clusters as nothing more than collections of independent nodes. As such, the techniques used to secure these clusters have, for the most part, been adaptations of techniques developed for securing and monitoring enterprise computing environments. We have previously proposed the idea of monitoring the security-state of large-scale commodity clusters by examining their emergent properties, that is, properties that are only visible when one ceases to look at a cluster as a collection of disparate nodes and begins to look at the properties of the cluster as a whole. We show that by correlating the open network ports observed on cluster nodes with other emergent properties—such as active processes and the contents of important system files—security analysts can make insightful observations that can greatly restrict the actions that an attacker can carry out undetected.

1 Introduction

Over the course of the last decade, the average performance of large-scale commodity clusters deployed in academic, commercial, and research environments has increased steadily [18]. This increase in performance, coupled with the deployment of technologies that enable Grid computing such as the Globus toolkit [7], has also lead to an increase in the number of clusters deployed in these environments [17]. While the scientific community has embraced this explosion in computing power, the groups tasked with monitoring the security-state of these clusters have struggled to manage their systems.

In the past, clusters were treated in much the same manner as enterprise computing environments, which is to say, simply as large collections of independent machines. While large-scale commodity clusters are in some sense simply collections of nodes, the consequences of compromising a single cluster node are much more severe than those of compromising a single enterprise node. In fact, the compromise of a single cluster node or account can lead to a compromise of the entire cluster. Grid computing has extended these dependencies to exist not only within a single cluster, but to span multiple clusters. This is one of the key factors behind the TeraGrid compromises that occurred in the Spring of 2004 [8]. This wave of attacks clearly highlights the differences that exist between cluster and enterprise computing environments. To mitigate damages caused by future cluster compromises, security administrators must update their current practices.

In a previous work, we advocate the concept of monitoring the security of large-scale commodity clusters through the examination of their emergent properties [20]. Unlike an enterprise computing environment, the nodes of a cluster can easily be partitioned into a small number of equivalence classes (for example, compute nodes, head nodes, storage nodes, etc.). Within each equivalence class, the member nodes are essentially clones of one another that vary only in predictable ways.

Armed with this knowledge, it becomes easier to glean valuable security information through the comparison of individual nodes to other nodes in their respective equivalence class (or classes). Our experiences with the clusters at NCSA have helped us to identify four sources of information that can be used as described above: running processes, critical file contents, open network ports, and network traffic patterns.

In another work, we show that the set of processes running on a cluster can be monitored at a frequency of up to once per second with negligible impact on the performance of the cluster [10]. While an attacker cannot do anything

useful without running some process, it is possible that they could modify the daemon process that reports back the process list to mask their processes. For this reason, process monitoring cannot be considered an end-all solution in the quest to monitor the security of large-scale commodity clusters.

In this paper, we show that by augmenting our previous process monitoring solution with scans for unauthorized open ports we can effectively "tighten the noose" on would-be attackers by reducing the space of actions that can be taken on the nodes of the cluster that can go by undetected.

The remainder of this paper is organized as follows. In Section 2 we discuss related work. Section 3 presents our threat model. Section 4 discusses the architecture of our monitoring solution, NVisionCC, and the means though which data is collected. We discuss our port monitoring implementation in Section 5 and present some preliminary results in Section 6. We present our conclusions and a brief discussion of future work in Section 7.

2 Related Work

2.1 Cluster Monitoring

Monitoring properties of large-scale commodity clusters presents challenges not present in monitoring the same properties on individual hosts. Several cluster monitoring solutions have been presented in the literature, including RVision [2], Supermon [14], and Ganglia [9]. These tools support high sampling rates and have little performance impact on the clusters that they monitor. However, these tools mainly report performance statistics such as CPU utilization, page fault rates, and network I/O statistics.

Clumon [4] is an open-source cluster performance monitoring system developed at the National Center for Supercomputing Applications. Clumon provides a central repository for performance metrics collected from each node using PCP [13], information from the job scheduler, and any metrics collected by user-developed plug-ins. We have built a tool, NVisionCC, which leverages Clumon's flexible architecture and extends it to support the monitoring of security properties of large-scale commodity clusters [22].

2.2 Port Scanning

Computer security analysts have long used port scanning to aid in the task of monitoring large networks. Tools such as Foundstone [3] and Nessus [12] incorporate port scanning as a part of their functionality. One use of port scanning in traditional security monitoring is to check for the existence of hosts on the network being monitored. This typically takes place via a "ping sweep" of the subnets being monitored. Other uses for port scanning include looking for hosts running a particular service (e.g., an FTP server) and ensuring that hosts are not violating the security policies of their domain by running illegal services.

In this paper, we present an extension to our cluster monitoring tool, NVisionCC, that exploits the nature of large-scale commodity clusters to enhance the ease with which illegal services can be detected. Enterprise monitoring tools such as Foundstone and Nessus cannot make any assumptions about the environments that they will be deployed in, so detecting illegal services often means that a human must interpret scan results to find irregularities. We show that a large amount of symmetry exists within the large-scale commodity cluster environment and illustrate that our tool can leverage this regularity to quickly and effectively detect illegal services running on previously unoccupied ports or masquerading as legitimate services.

3 Threat Model

Throughout this paper, we focus on the use of port scanning as a single tool at the disposal of system administrators tasked with monitoring the security state of large-scale commodity clusters. As such, we do not advocate the use of port scanning to detect all forms of cluster intrusion, but rather to locate only a particular class of attacks. Any anomalies detected through the use of port scanning are meant to be cross-referenced and correlated with the findings of other security monitoring techniques—a task which our comprehensive cluster monitoring tool, NVisionCC, is designed to carry out.

Port scanning allows system administrator to probe a given network to determine information about the hosts available on a given network, the ports open on each hosts, and even information about the operating system and particular services running on each host. We assume that an attacker who compromises a given node in a cluster and opens a network port wishes to do so for the purpose of providing a service to himself or others. Examples of such services could include opening a back door into the system or providing a file storage repository. This is in contrast to situations where the mere existence of the open port is of primary interest, e.g., if the existence of a port is used as a covert channel. In addition, we assume that such services can be opened on any port, not simply the well-known ports of the services in question.

4 Architecture

In this section, we present an overview of the architecture of our cluster security monitoring tool, NVisionCC.

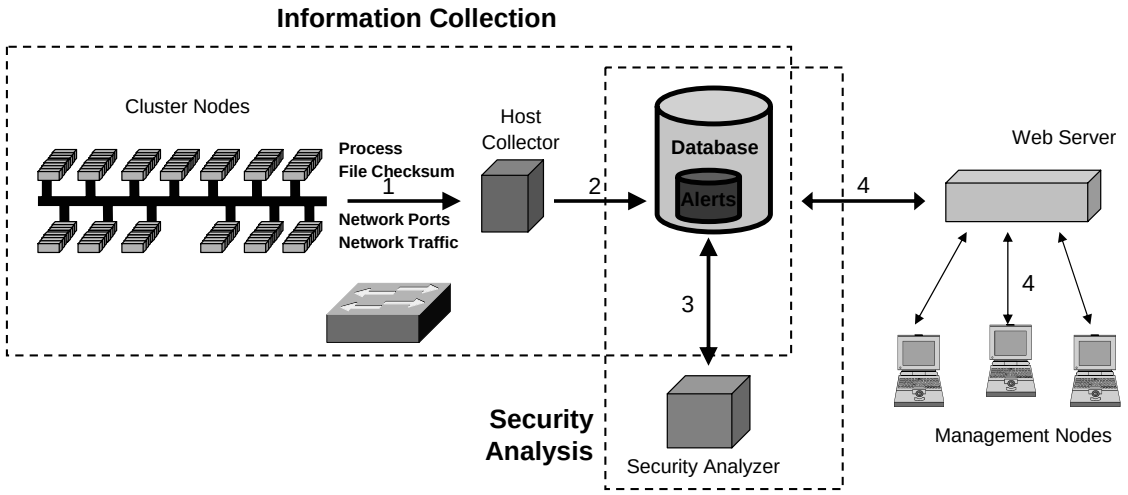

Figure 1. The operation of NVisionCC consists of three steps: information collection, security analysis, and visualization. Key to this process is the central database of profiles and alerts.

NVisionCC is designed to be an extension of Clumon, a widely used in cluster community as performance monitoring tool. Our tool is designed to monitor the security state of large numbers of cluster nodes in a manner that incurs a minimal performance overhead.

Figure 1 provides an overview of the NVisionCC architecture. Security monitoring using NVisionCC is an iterative three-step process: data of interest are collected from the nodes being monitored, analyzed, and finally visualized. Each of these steps is discussed in greater detail below.

4.1 Information Collection

During the information collection stage, various NVisionCC plug-ins are activated to gather data of interest from nodes on the cluster. While it would be impossible to know everything about every host, from a security perspective, this is largely unnecessary. In a previous work, we present several emergent properties of large-scale commodity clusters—including running processes, critical file contents, and open network ports—that may be useful for security analysis [20]. All configured monitoring plug-ins are activated at a regular intervals that can be configured within the NVisionCC interface.

4.2 Security Analysis

We have shown that the nodes of a large-scale commodity cluster can be partitioned into equivalence classes and that within these classes many security properties should be the same [20]. In NVisionCC, we utilize a centralized MySQL [11] database to store node class profiles for the various properties that are being monitored. Such profiles are usually easy for a cluster administrator to generate and need only be changed as the cluster is upgraded. In general, these profiles define the steady-state appearance of nodes on the cluster. For instance, lists of running processes, hashes of critical system files, or lists of open network ports are potential properties that could be present in a node class profile.

The node class profiles described above are vital to the security analysis performed by NVisionCC. As data are collected during the Information Collection stage, they are compared with the existing profiles stored in the database. Any values that differ from the stored node class profiles generate alerts indicating a potential security breach. For example, an alert would be generated if the process monitor of a given node reports a process that is not listed in the running processes profile for that node's equivalence class. These alerts are stored in the NVisionCC MySQL database.

4.3 Visualization and Management

A final objective of NVisionCC is to provide a user-friendly interface to visualize alert conditions in the cluster [21]. To accomplish this, NVisionCC presents a web interface that allows security administrators to monitor the state of an entire cluster in one screen. Administrators can drill down to view the details of a particular node by clicking on that node.

In the following section, we illustrate the ways in which

port scanning can be used to detect security incidents in a large-scale commodity cluster and describe our NVisionCC plug-in designed to facilitate this task.

5 Port Scanning in the Cluster Environment

In this section, we discuss the benefits of port scanning in a cluster environment and present an NVisionCC plug-in designed to automate this process.

5.1 The Benefits of Port Scanning

Just as a baseline NVisionCC installation profiles the running processes on cluster nodes based upon the observation that cluster nodes of the same type should be running the same processes, we can make a similar assumption about the open ports present on a particular cluster node. More specifically, cluster nodes of a given type should have the same TCP and UDP ports open. There are two main reasons that this assumption tends to hold in practice. First, unallocated nodes are typically built from the same system image and thus run the same network services. Second, allocated nodes tend to communicate over a high-bandwidth, low-latency fabric, such as Myrinet, rather than over TCP/IP links, so additional TCP and UDP ports are rarely opened while an allocated cluster node is in use. A notable exception to this is the case of computational-steering jobs, though this is easily accounted for in practice.

Given that uncompromised cluster nodes will tend to follow a known open-port signature, it becomes very easy to use port scanning as a tool to detect nodes that may have been compromised. Many times, when an attacker compromises a host, the attacker installs some sort of remotely accessible service on the compromised host. Examples of these services include IRC "bots" that provide file storage and access capability to users on the Internet [1] and DDoS "zombies"that sit idly until awakened by a master node which issues commands for the zombie to begin attacking a particular target [19, 16, 15]. Given the high profile of DDoS attacks and file sharing on the Internet in recent years, the ability to detect these types of compromises in an automated fashion is advantageous to security administrators.

5.2 Plug-in Architecture

After it was observed that the open ports of cluster nodes could be profiled easily, we began the development of an NVisionCC plug-in that provides the monitoring and error reporting functionality needed to detect deviations from these profiles. To facilitate such a plug-in, a repository was needed to store the permissible open port signatures of the various cluster node types. To this end, the NVisionCC database has been augmented to store open port profiles for the node types present in the cluster. Table 1 describes the `open_ports` table which contains these open port profiles. To support easy population of the `open_ports` table, a simple Perl script was written to convert the output from a Nmap [5] scan of a single node into an open port profile for a particular class of cluster node.

The port scanning plug-in itself was written in Perl and scheduled with NVisionCC to be run at each "collection interval." The collection interval controls how often data should be collected from the cluster and is tunable from within the NVisionCC/Clumon interface. At each invocation, our plug-in gathers the list of node-types present in the cluster being monitored. For each node type, our tool retrieves a list of the nodes of this type from the database along with the rows of the `open_ports` table pertaining to this node type. An Nmap scan of these nodes is then conducted, though the exact parameters of this scan can vary depending on the exact usage scenario; several such scenarios are discussed in Section 5.3. By default, only idle nodes and nodes that have not been scanned for some user-defined period of time are scanned, as to avoid causing a crash of poorly-written user code on the cluster. However, in a highly utilized cluster, administrators may wish to scan every node, regardless of its idle status. The plug-in can be easily tuned to function in this way.

The main contribution of our tool is its ability to distill the copious output generated by scans of thousands of ports on thousands of hosts into a meaningful picture of the security-state of the cluster. To accomplish this, the visualization capabilites of NVisionCC are leveraged. Upon the detection of any illegal open ports or miscreant services found to be masquerading on legitimate ports, an alert is written into the NVisionCC error log. All alerts in this log are visualized in the same fashion, making it trivial for security administrators familiar with Clumon to incorporate port scanning into their repertoire of security monitoring practices.

5.3 Usage Scenarios

As previously mentioned, Nmap is used to carry out the actual port scans conducted by our plug-in. Nmap is an extremely configurable tool and our plug-in is designed to support a wide array of scan parameters.

By default, Nmap performs a TCP `connect()` scan of all ports listed in the `nmap-services` file. This type of scan runs incredibly fast and is the default scan performed by our tool. However, this scan type not only misses many ports of interest, but completely ignores the UDP protocol. For sites wishing for to implement custom scans, we support all of Nmap's basic TCP and UDP scans. The "stealth" scanning options were not tested, however, as it seems un-

Field	Data Type	Description
`port`	int	The port number that this row corresponds to
`protocol`	char(5)	The protocol that the above port should be open on (eg., TCP)
`service`	char(50)	An optional service description
`host_type`	char(60)	The host type that this port should be open on

Table 1. The `open_ports` table

likely that a network security administrator would need to conceal the source of his scans from himself. When using these "standard" port scanning methods, errors are logged to the database when an illegal open port is detected.

A more interesting scan type is the "version scan" supported by Nmap [6]. This scan type probes open ports to detect the actual service running on that port and any version information that it can obtain, rather than reporting the service name listed in the `nmap-services` file. Administrators can utilize the `service` field in the `open_ports` table to store the correct version information and have our scanner compare the version information collected against this fingerprint. In this way, illegal services running on ports that are permitted to be open can be detected easily.

Version scanning takes a considerable amount of time to run, however. Incorporating version scanning is perhaps best done in addition to more rapid port scans that simply probe for open ports. For instance, an administrator could configure one instance of this plug-in to carry out rapid TCP and UDP port scans of all ports on each cluster node. Another instance of the plug-in could be configured to version scan only the permitted open ports. In this way, the administrator can be assured that no new ports are opened during very short intervals, while the services running are probed less often to look for malicious processes hiding on known "good" ports. Configuring two instances of our plug-in is an simple task, making this a viable scan technique.

6 Preliminary Performance Results

In this section we examine the overheads associated with using port scanning to examine the security state of large-scale commodity clusters. Specifically, we show that cluster nodes can be scanned at a high enough rate to provide useful insight into the security state of a cluster.

The timing measurements that we present were measured on a cluster with dual-processor nodes running Red Hat Linux with kernel 2.4.21-15.ELsmp. The node performing the scans was located on the same gigabit Ethernet as the cluster nodes as to minimize round-trip times. This configuration seems plausible in practice, as the entity performing the security analysis of the cluster would likely have the ability to attach monitor nodes to the Ethernet used by the cluster. Measurements reported are averages over 10 scan trials.

6.1 Open TCP Port Scanning

The first type of scan that we examined was open TCP port scanning. This type of scan will report whether or not a service is running on a particular network port and the service name for that port, as it is listed in `nmap-services` or `/etc/services`. To measure the speed of this scan type, we had Nmap perform a TCP `connect()` scan of some subset of the 65535 TCP ports.

Scanning the 1601 ports listed in the `nmap-services` file took on average 0.391 seconds. While this appears fast, it is important to note that this type of scan would not discover any services running on unusual ports, most notably those running on the ephemeral ports. A scan of the full range of possible TCP ports took 4.219 seconds on average.

The rates observed scale linearly with the number of hosts being scanned. Given this observation, all ports of a 512-node cluster could be scanned in roughly 36 minutes. If two monitoring nodes were to partition the IP space of the cluster, the scan time could be reduced to about 18 minutes. Given our assumption that services running on a compromised node will do so for some time, monitoring the cluster once an hour or even once a day would suffice, so a scan time on the order of 15-30 minutes is perfectly reasonable in practice.

6.2 TCP Version Scanning

As mentioned in Section 5.3, a nice complement to the standard TCP port scan is Nmap's TCP version scan. This scan type probes open ports in an effort to determine the service name and version running on that port. While the TCP `connect()` scan discussed above can be used to locate services running on previously unused ports, the version scan can locate rogue services masquerading on legitimate open ports.

Due to the probing nature of an Nmap version scan, these scans take considerably longer than a `connect()` scan to complete. This, does not hinder its usefulness, however, as we do not wish to version scan every possible port, simply the ports defined as open in our cluster node profile. On average, we found that it took 5.404 seconds to version scan 4 open ports. At this rate, it would take approximately 46 minutes to version scan the permissible open ports on a 512

node cluster, making this scan reasonable to carry out in practice.

7 Conclusions and Future Work

We have previously proposed the idea of partitioning a cluster into sets of equivalence classes based upon the functionality of each node. Such equivalence classes may include the sets of compute nodes, head nodes, storage nodes, and management nodes. Within each class, we have found striking similarities between the member nodes. These similarities are easily profiled and deviations from these profiles have been shown to occur during attack situations. In [10], we showed that monitoring the list of running processes could occur with minimal impact on the nodes comprising the cluster.

In this paper, we extended this profiling to include the set of open network ports found on cluster nodes. We proposed various scanning methods that are of use in the large-scale commodity cluster environment. The cluster monitoring tool NVisionCC was extended via a plug-in to carry out the various scans described in this paper. Preliminary performance results indicate that such scanning is possible to carry out at a high enough frequency to be useful in practice.

In the future, we plan to examine different scanning frequencies and determine the threshold at which port scanning noticeably effects the performance of the cluster. In addition, we hope to incorporate the examination of properties such as network traffic patterns and node log files into our security monitoring toolkit. For instance, monitoring the syslog files for the cluster as a whole could give insight into attempted brute-force password-guessing attacks that would otherwise go undetected on a single node, due to the time delay between login attempts on each individual node.

References

[1] Eggheads.org: Eggdrop Development. Web Page, Nov. 2004. ⟨http://www.eggheads.org/⟩.

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

[3] Foundstone. Web Page, Nov. 2004. ⟨http://www.foundstone.com/⟩.

[4] J. Fullop. Clumon. Web Page, Jul. 2004. ⟨http://clumon.ncsa.uiuc.edu/⟩.

[5] Fyodor. The art of port scanning. *Phrack Magazine*, 7(51), Sep. 1997. ⟨http://www.insecure.org/nmap/p51-11.txt⟩.

[6] Fyodor. Nmap version scanning. Web Page, Apr. 2004. ⟨http://www.insecure.org/nmap/versionscan.html⟩.

[7] The Globus Alliance. Web Page, Aug. 2004. ⟨http://www.globus.org⟩.

[8] Grid Attacks Raise Concerns Among Security Experts. *Grid Today*, 3(17), Apr. 2004. ⟨http://www.gridtoday.com/04/0426/103080.html⟩.

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

[10] G. A. Koenig, X. Meng, A. J. Lee, M. Treaster, N. Kiyanclar, and W. Yurcik. Cluster Security with NVisionCC: Process Monitoring By Leveraging Emergent Properties. In *IEEE Cluster Computing and the Grid (CCGrid*, May 2005.

[11] MySQL: The World's Most Popular Open Source Database. Web Page, Nov. 2004. ⟨http://www.mysql.com/⟩.

[12] Nessus. Web Page, Nov. 2004. ⟨http://www.nessus.org/⟩.

[13] Performance Co-pilot. Web Page, Jul. 2004. ⟨http://oss.sgi.com/projects/pcp⟩.

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

[15] CERT Advisory CA-2000-01 Denial-of-Service Developments (Stacheldracht). Web Page, Jan. 2000. ⟨http://www.cert.org/advisories/CA-2000-01.html/⟩.

[16] CERT Advisory CA-1999-17 Denial-of-Service Tools (TFN2K). Web Page, Mar. 2000. ⟨http://www.cert.org/advisories/CA-1999-17.html/⟩.

[17] Charts for June 2004: Application Area/Systems. Web Page, Jun. 2004. ⟨http://www.top500.org/lists/2004/06/overtime.php?c=0⟩.

[18] Top 500 Performance Development. Web Page, Jun. 2004. ⟨http://www.top500.org/lists/2004/06/PerformanceDevelopment.php⟩.

[19] CERT Incident Note IN-99-07: Distributed Denial of Service Tools (Trinoo). Web Page, Jan. 2001. ⟨http://www.cert.org/incident_notes/IN-99-07.html/⟩.

[20] W. Yurcik, G. A. Koenig, X. Meng, and J. 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.

[21] W. Yurcik, X. Meng, and N. Kiyanclar. NVisionCC: A Visualization Framework for High Performance Cluster Security. In *CCS Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)*, Oct. 2004.

[22] W. Yurcik, X. Meng, and G. A. Koenig. A Cluster Process Monitoring Tool for Intrusion Detection: Proof-of-Concept. In *29th IEEE Conference on Local Computer Networks (LCN)*, 2004.