

# Detection of Privilege Escalation for Linux Cluster Security

Michael Treaster, Gregory A. Koenig, Xin Meng, William Yurcik  
{*treaster, koenig, xinnmeng, byurcik*}@ncsa.uiuc.edu

National Center for Supercomputing Applications (NCSA)  
University of Illinois at Urbana-Champaign (UIUC)

**Abstract.** Cluster computing systems can be among the most valuable resources owned by an organization. As a result, they are high profile targets for attackers, and it is essential that they be well-protected. Although there are a variety of security solutions for enterprise networks and individual machines, there has been little research focused specifically on securing cluster systems despite their great importance.

NVisionCC is a multifaceted security solution for cluster systems built on the Clumon cluster monitoring infrastructure developed at the National Center for Supercomputing Applications. This paper describes an NVisionCC component designed to detect unauthorized privilege escalation. This component enables security monitoring software to detect an entire class of attacks in which an authorized local user of a cluster is able to improperly elevate process privileges by exploiting some software vulnerability. Detecting this type of attack is one of many facets in an all-encompassing cluster security solution.

## 1 Introduction

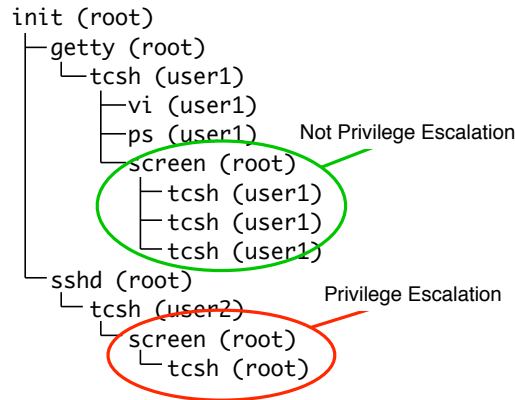
Historically, cluster computing systems have been seen as large collections of independent, heterogeneous machines for purposes of security. This model closely resembles that of an enterprise network, and administrators have attempted to apply enterprise network security practices to their cluster resources in order to protect them. This approach was recently demonstrated to be ineffective, however, when attackers successfully compromised a number of production cluster systems [2][13][21]. Although these intrusions were eventually detected, the vulnerabilities patched, and the damage repaired, security practices for cluster systems remain largely unchanged.

Because cluster computing systems are usually among the most valuable resources owned by an organization, securing these high-value assets is critical. Security penetrations into such systems can be extremely expensive for the organizations that own them. Users within the organization are often deprived of access to computing resources for extended periods following a break-in while the damage is repaired and the exploit that allowed the attack to succeed is patched. Further, because user accounts are usually shared across all nodes in a cluster, a security breach on a single node can easily spread to all other nodes. This puts the processing power of the entire cluster into the hands of the attacker, who can then utilize the system for malicious purposes. Finally, cluster systems, particularly those used within Grid computing environments, contrast traditional enterprise systems in that they tend to be centers of user accounts and activity. A security breach on a single system can be leveraged to quickly compromise a large number of other systems.

Fortunately, cluster systems are fundamentally different from traditional enterprise resources. In particular, the aggregation and organization of individual machines into a single cluster system results in emergent properties that can be exploited to make the system more secure [24]. For example, although clusters are composed of large numbers of machines, those machines are not heterogeneous and can be categorized into a small number of classes, each with expected characteristics. Clusters also typically run software, such as a job scheduler, which can be coordinated with intrusion detection software in order to improve the detection of attacks while reducing the occurrence of false positives. These types of properties are discussed in more detail in Section 4.

NVisionCC [25] is a multifaceted cluster security tool built on the Clumon [3][20] cluster monitoring infrastructure developed at the National Center for Supercomputing Applications. It provides security by exploiting the emergent aspects of cluster systems in combination with many types of monitoring. The results of this monitoring are summarized in a Web-based management component that provides a clear visualization of the state of all nodes in the system. This paper focuses on a component of NVisionCC designed to detect unauthorized privilege escalation. Specifically, it describes how this component works, why it is essential to NVisionCC, and how it depends on the other NVisionCC components.

New, serious vulnerabilities appear continuously in `setuid` applications. One recent example from January 2005 [8] involves flaws in the Linux binary format loader `use1ib()` functions that allow a local user to escalate



**Fig. 1.** A Sample Process Tree Showing Malicious and Innocuous Activity

their privileges to root access. Although keeping systems up-to-date with the latest patches is a crucial aspect of maintaining a secure system, in the case of large-scale production clusters, such updates often cannot be performed immediately. These systems are in use almost continuously in order to maximize their value, and installing updates on hundreds or thousands of nodes can be extremely time consuming. Interrupting production jobs to apply security updates simply is not feasible in many environments. When a new vulnerability is announced, the system is in extreme danger until the vulnerability is patched. Therefore, it is crucial that a general-purpose intrusion detection solution such as NVisionCC be deployed in order to detect attacks exploiting an unknown or unpatched vulnerability. Privilege escalation detection is one aspect of this broader solution.

## 2 Unauthorized Privilege Escalation Techniques

It is common practice for a computer intruder to attempt to gain root access to a machine after first attaining access to a less privileged account. There are several means through which this is typically accomplished. One possibility is that an attacker can attempt to acquire the root password, either through social engineering or by brute force. The former is easily avoided by a skilled administrator, and the latter can be prevented by monitoring login attempts and protecting the password file. Another possibility is to make use of an exploit in a system daemon that runs as root, and use that route to execute arbitrary code as a root user.

The privilege escalation detection described in this paper focuses on recognizing malicious privilege escalations that result from the exploitation of vulnerabilities in `setuid`-enabled programs. The `setuid` bit is a permission bit that can be enabled on a Unix executable file. When an application's `setuid` bit is set, the application runs with the privileges of the owner of the executable file rather than the privileges of the user running the application. Typically, the executable file is owned by some privileged system user, such as root, causing the executing user's privileges to be temporarily elevated while the application runs. This functionality is important within the Unix security model because it allows users to obtain controlled access to critical system resources which must be carefully protected. In this way, the `setuid` bit provides the user with elevated access privileges, but the use of these elevated privileges is regulated by the application.

An example of a `setuid` application is the `passwd` program. Administrative-level privileges are usually required to access the password file in order to prevent a number of possible attacks against a system. For example, with read access to the password file, an attacker could obtain the encrypted passwords for all users of a system and stage a brute-force attempt to determine their corresponding cleartext passwords. Alternatively, with write access to the password file, the attacker could alter the file to set the password of any account to an arbitrary value. However, there are cases where a legitimate, unprivileged user requires access to the file. In particular, users must be able to write to the password file in order to change their passwords. In this case, the `passwd` program provides access to the password file through the `setuid` mechanism, but this access is strictly controlled by the `passwd` application.

This approach to privilege escalation can be exploited when a `setuid` program is functioning improperly. Typically, an attacker looks for a buffer overflow vulnerability in a `setuid` program owned by the root user.

The attacker then exploits this vulnerability to cause the program to execute malicious code not intended by the program's author. The program, and therefore the malicious code, executes with root permissions, and the attacker can then proceed to subvert the system unhindered [14].

### 3 Related Work

Since malicious privilege escalation is such a serious threat to security, there has been a wide range of research dedicated to addressing the problem. Techniques can generally be categorized into one of three types of approaches: prevention, containment, and detection.

Prevention techniques attempt to address the security flaws that allow malicious privilege escalation to occur, rather than dealing with an intruder after a security weakness has been exploited.

Privilege separation is an approach in which a privileged program is divided into a secure component and an insecure component. The secure component is responsible for performing the privileged operations, while the insecure component handles the remainder of the program logic. The insecure component communicates with the secure component to indicate what privileged operations it needs the secure component to perform on its behalf. By separating the secure functionality into an independent software module, the size and complexity of critical code that must be defect-free to provide a secure system are reduced [9][17].

Many malicious privilege escalation techniques exploit buffer overflows in programs to cause a `setuid`-enabled program to execute arbitrary code by first overwriting a memory pointer with a carefully chosen value. Several prevention techniques that specifically address this type of attack exist. One technique is to randomize the memory address space of an application dynamically at runtime, such that an attacker cannot know what value to use when overwriting a memory pointer [23]. Although buffer overflows can still be attempted on software protected by this mechanism, they will usually result in a program crash instead of executing the code desired by the attacker.

A different software-based approach is to monitor the return addresses of functions in the program stack and stop program execution if an active function return address is modified. This can be accomplished by inserting a random, known value into the program stack adjacent to each return address. The location of this address must be written to in order to overwrite the return address, but the value at this address is monitored in order to detect any tampering. If the value changes, tampering is suspected and the program can take defensive action [4].

Several contemporary microprocessors include hardware-level protection against improper code execution. This technology allows memory regions to be flagged as non-executable by a special bit at the hardware level. The processor then checks every incoming instruction and refuses to execute the code if the tag appears. The stack and heap of a program can be marked by this flag such that if an attacker exploits a buffer overflow to execute instructions in these memory regions, the attempt will fail. This technology is known as "NX" (no execute) by AMD, "XD" (execute disable) by Intel, and "Data Execution Protection" by Microsoft [6].

Another approach to intrusion prevention is to associate root access to a system to some kind of physical device. For example, logging in as root could require the presence of a physical token or key that is plugged into the machine or communicates with it wirelessly to prove its presence. Without this token, root access is denied, regardless of the validity of the password.

Most Unix systems have the capability to block remote root logins to a machine, which ties root access to the actual machine the user seeks to log into instead of to a portable token. However, this is impractical in a cluster system, where most or all of the nodes are not equipped for a local login, and they can only be accessed remotely. A more usable approach to machine locality is to allow root access if the user has access to any of a set of physical machines. When a user attempts to escalate privileges to administrative levels, the system traces back through the sequence of remote logins to find the actual location of the user. If this location corresponds to an authorized machine, the privilege escalation is permitted. If not, the attempt is blocked. This is especially useful for blocking login attempts by users who have used a brute force attack to acquire the root password [18].

Containment techniques try to prevent an intruder from leveraging unauthorized escalated privileges. The typical approach to containing an attacker is to isolate untrusted programs [1][22] or users [7] from system resources that must be protected, such as network interfaces, critical system software, and areas of the file system. This containment persists even if a user is able to escalate privileges to attain root access to the system, preventing this access from being exercised in ways that could seriously affect the system or other users.

Detection techniques do not seek to interfere with an intruder's activities. Instead, they attempt to detect the presence of an intruder, then alert an administrator so that human intervention can be made in order to verify

Class	Description
Compute node	Performs computation for production processing jobs
Storage node	Provides an interface to large-scale storage subsystems
Management node	Hosts management software
Head node	Provides an interface for end-users to interact with the cluster
Monitor node	Hosts cluster monitoring software

**Table 1.** Classes of nodes in cluster systems

Property	Description
Process List	Ensure that only appropriate or correct process names are running
Process Privileges	Ensure that unauthorized privilege escalation does not occur
Open Network Ports	Ensure that no unusual network ports are open on a node
File Signatures	Ensure the critical system files maintain the MD5 hash or other signature

**Table 2.** Common properties of cluster nodes

the intrusion and determine the appropriate response. By including human decisions in the process, detection techniques to be more aggressive in their approaches since false positives are not as catastrophic when they are filtered by an administrator before action is taken.

There exists a large amount of work that seeks to specify behavior policies for programs that might have security vulnerabilities. Such systems monitor the behavior of `setuid` programs and ensure that they conform to the specified policy. If a violation of the policy occurs, malicious activity is assumed to be in progress, and appropriate security measures can be effected. This is useful for detecting and preventing attacks involving buffer overflows and other security defects in privileged programs [11][10][16].

Another detection technique is to examine files to look for code patterns that exploit a privilege escalation vulnerability. A feed forward neural network can classify ELF [19], C source code, and shell code [5] files as malicious or benign by looking for instruction patterns that might be malicious. Such networks rely on tell-tale features in the code, such as the existence of binary data in a C source code file that may represent the instruction data used to trigger the buffer overflow. This type of tool does not detect vulnerabilities in software, but it does detect code that an attacker might use to exploit such vulnerabilities.

All of these approaches focus on the security of single machines. The contribution of the work presented in this paper is twofold: (1) We focus on securing cluster systems by exploiting the emergent properties of such systems, and (2) We provide a real, usable tool that provides a visualization of system activity to provide security administrators with situational awareness of the events on their cluster. Additionally, our tool builds on an existing software infrastructure (Clumon) that is common on many cluster systems.

## 4 Emergent Properties of Clusters

Emergent properties are unplanned characteristics or behaviors that arise when simple components are assembled into a more complex system. Clusters possess characteristics that result from the aggregation of many individual computers into a single larger, more ordered machine. As mentioned earlier, these properties can all be exploited to facilitate greater security [24]. These properties make a cluster environment fundamentally different from an enterprise network of comparable size for purposes of security.

Foremost among the emergent properties of clusters is the ability to categorize each of the many nodes in a cluster into a small number of homogeneous classes. The most common classes are shown in Table 1. All nodes in a given class share a variety of expected behaviors. For example, management nodes are expected to have the same network ports open at any given time, and all storage nodes are expected to have an identical process list at any given time. A complete list of the common properties we have identified is shown in Table 2. These commonalities allow the expected behavior of all nodes in the cluster to be specified concisely from a central location, reducing the cost of managing the system and the complexity of performing the monitoring.

Another important emergent property is the ability to classify users more easily than in an enterprise network. Although cluster systems usually have a large number of users, the vast majority of them do not possess administrative privileges. Additionally, administrators of large-scale clusters are typically not users of the cluster as well. They do not run large processing jobs, except perhaps to test the system. This allows the behaviors of each class of users to be defined with little overlap.

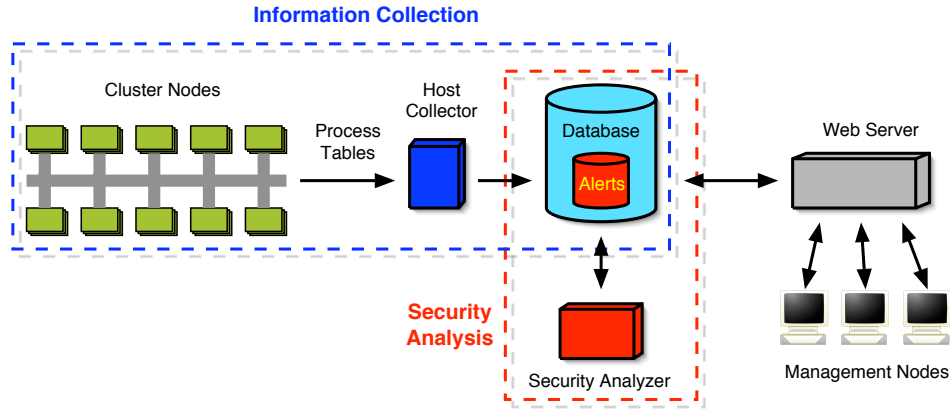


Fig. 2. Architecture of NVisionCC

A third emergent property is that cluster nodes, and especially compute nodes, are usually not general-purpose machines. They are likely to run only a limited class of applications and commands, and programs outside of this set can be flagged as anomalous. Similarly, user behavior on these machines is significantly different from, and more restricted than, user behavior on personal machines. This simplifies the construction of a profile of legitimate activity, allowing legitimate activity and malicious activity to be more easily differentiated.

The last emergent property that we have identified is related to the presence of the cluster batch job scheduler. The job scheduler is responsible for allocating compute nodes to user batch jobs and then launching these jobs onto the allocated nodes. Security monitoring software can leverage information from the job scheduler to more accurately detect suspicious behavior. For example, security software might detect the presence of an unknown process running on a compute node. The owner of this process can be correlated with information from the job scheduler. If the job scheduler has allocated the compute node to the owner of the questionable process, then it is likely that the process is legitimate. On the other hand, if the job scheduler indicates that the compute node is supposed to be idle, or assigned to a different user, then it is likely that the process is unauthorized.

## 5 PCP, Clumon, and NVisionCC

Performance Co-Pilot (PCP) [15] is an Open Source framework and set of services for performance monitoring and management developed by SGI. It uses a distributed architecture in which 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 has a negligible impact on cluster performance.

Clumon [3][20] is a CLUster MONitoring infrastructure that provides a framework for gathering data about various aspects of a cluster system. The Clumon framework includes components that carry out three primary security-related tasks:

- *Information Collection* – The Host Collector component collects data from the cluster nodes and delivers it to a database on a dedicated server. By default, the Host Collector gathers primarily performance data, allowing an administrator to identify nodes with unusual performance or usage characteristics.
- *Security Analysis* – The Security Analyzer component examines the contents of the database to search for unusual activity patterns. Any results generated by the data analyzer are stored back in the database.
- *Visualization* – This visualization component checks the database for the status of all cluster nodes and assembles a Web-based visualization that clearly depicts the state of the system. This interface also provides the ability to get more detailed information about many aspects of the system, including highlighted security alerts.

The default Clumon package does not provide any facilities for security. However, it provides a plugin mechanism to allow for additional features or performance attributes to be monitored. Information about these characteristics can be gathered using PCP or by using any other data collection tool. The plugin mechanism

Host Name	tgc12.trecc.org		
Time	2005-01-03	8:43:21	
<b>Process Tree</b>			
init (root)			
	getty (root)		
		tssh (user1)	
			vi (user1)
			ps (user1)
			screen (root)
			tssh (user1)
			tssh (user1)
			tssh (user1)
	sshd (root)		
		tssh (user2)	
			screen (root)
			tssh (root)

Fig. 3. Hypothetical process tree detail view for a single node

also allows for new data analysis techniques to be implemented and new visualizations to be shown, in order to make use of the new information that is gathered.

NVisionCC [26] is a security monitoring system consisting of a number of Clumon plugins. It focuses on exploiting the emergent properties of cluster systems to provide better security while reducing administrative load. Each plugin provides a different aspect of a complete security solution. Currently, the NVisionCC system includes modules that check for (1) the integrity of critical files, (2) the existence of unauthorized open ports, (3) the existence of unauthorized processes, and (4) the occurrence of inappropriate privilege escalation. This paper describes this last component and how it complements and is complemented by the other components of NVisionCC.

## 6 Privilege Escalation Detection

Our technique for detecting unauthorized privilege escalation involves analyzing the structure of the process tree on a cluster node. In the type of attack we are trying to detect, if a process  $p$  has obtained unauthorized root privileges it must meet the following three requirements:

1. The process  $p$  must have a `setuid` parent process. The program executing as this process is the program that was compromised by the attacker in order to acquire the unauthorized escalated privileges. Although it is possible that a process with unauthorized escalated privileges can have a parent that is not `setuid`, this can only occur if the parent itself has unauthorized escalated privileges. In this case, this parent or one of its ancestors in the process tree must have a `setuid` parent, and the child of this `setuid` process will trigger the privilege escalation detection alarm.
2. The process  $p$  must have a grandparent  $g$  owned by a user that does not own  $p$ . This is the user that originally ran the `setuid` program. If the owner of both  $p$  and  $g$  is the same, then no privilege escalation has occurred and there is nothing to detect.
3. The owner of  $p$  must not be on the list of users authorized to escalate their privileges. This list, which includes root, enumerates administrators who are authorized to have complete access to all resources on the system. If a process owned by a user from this list is found to have escalates privileges, it is assumed that this privilege escalation is legitimate. Such cases would be when an administrator uses the `su` or `sudo` commands to attain administrative-level access to a node.

For each process  $p$  in the process tree, the detection algorithm checks the characteristics of the parent process and the grandparent process. If the parent process is a `setuid` program and if the owner of the grandparent is different than the owner of  $p$ , and if the owner of the grandparent is not an authorized administrator, then some kind of unauthorized privilege escalation is evident. This algorithm is presented more formally in Figure 4.

Our technique takes advantage of several emergent properties of cluster systems. First, a cluster system is likely to have a large number of users, but few users that are authorized to escalate to root access. As a result, if an intruder compromises a random account, it is likely that this account will not be on the authorization list. Furthermore, the authorized accounts are likely to be trained security administrators who are more skilled at protecting the integrity of their own accounts, reducing the chance that an administrator account is compromised. Second, the set of user accounts and the list of authorized users is the same on all nodes in the

```

for each process p in n
  if p->parent->owner is setuid AND
    p->parent->parent->owner neq p->owner AND
    p->parent->parent->owner not in root_auth
    raise alarm

```

Fig. 4. Pseudocode for the privilege escalation detection technique

cluster. This eases the management overhead of securing the system, since the authorization list can be created and updated in a single, central location. Finally, the detection of any malicious activity results in an alert to NVisionCC, which acts as a centralized, existing visualization interface to the entire cluster.

The unique cluster environment that we leverage to apply our technique contrasts a traditional enterprise network environment. In an enterprise network, a few users are likely to have root privileges on all systems, however some users might have root privileges on their personal workstation as well as on a small number of additional machines. An authorization list in this environment would require a mapping between users and machines, instead of a simple list of users that is adequate in a cluster environment. Additionally, since there are more users on the authorization list, changes to the list would be required more often. The larger number of more complex changes would hinder the management of this security approach in an enterprise environment.

## 6.1 Weaknesses

Our approach has a number of weaknesses that would allow a skilled and knowledgeable intruder to circumvent the technique. First, due to the periodic nature of PCP monitoring on which the sensor depends, it is conceivable that an intruder could obtain escalated privileges, install a more legitimate backdoor program, and exit the system between monitoring rounds. Secondly, it is easily possible to disassociate a process from its parent process, thus disrupting the process tree structure that our approach attempts to detect. Finally, due to the `root_auth` white list that is used to reduce false positives, our approach cannot detect the use of a hacked administrator account used to maliciously escalate privileges.

The first two concerns can be addressed by replacing the PCP monitoring with an event-driven monitor that is triggered by system calls. Whenever a system call that changes access privileges occurs, or when a process is disassociated from its parent, the sensor could immediately check the state of the process tree before these system calls are resolved and raise any necessary alerts. This reactive approach to monitoring would address the timing issues present in our current implementation. However, this system would not exploit the existing Clumon infrastructure present on many cluster systems, and it has not been implemented at this time due to time and complexity concerns.

## 6.2 Integration With NVisionCC

Our privilege escalation detection component fills a void in the security coverage provided by NVisionCC. Process monitoring is a central component of NVisionCC which relies on the homogeneity within node classes (e.g., head nodes, storage nodes, compute nodes). The process tree for each machine in each class of machines is identical in the steady state, and deviations from this state occur only under proscribed circumstances [12]. Although this approach is ideal on compute nodes and storage nodes where the system behavior is highly predictable, it is inadequate on cluster head nodes, which bear a much stronger resemblance to general-purpose machines. This shortcoming of process monitoring is especially significant because the head nodes are where intrusions are most likely to occur.

Privilege escalation detection augments NVisionCC by addressing this failing of the process monitoring sensor. Privilege escalation detection is well-suited to monitoring the head nodes of cluster systems, where our process monitoring technique is ineffective. In fact, the technique described in this paper is not specific to cluster systems and could be applied to general-purpose computing resources in an enterprise environment. However, it is especially important in a cluster environment protected by NVisionCC, due to its otherwise weak protection of cluster head nodes.

The privilege escalation detection component is itself reliant on NVisionCC as well. In particular, it can be evaded if an attacker is able to modify the `root_auth` file. The NVisionCC file integrity monitor checks the integrity of this file to detect any malicious tampering. Additionally, file integrity monitoring can help counter the backdoor weakness described in Section 6.1 by detecting the backdoor installation.

## 7 Implementation in NVisionCC

We have implemented the privilege escalation detection component described in this paper as a plugin to NVisionCC. Our implementation allows alerts generated by this sensor to be correlated with other sensors, improving the accuracy of intrusion detection as a whole. Additionally, NVisionCC provides a centralized location for the administration of security parameters in the system.

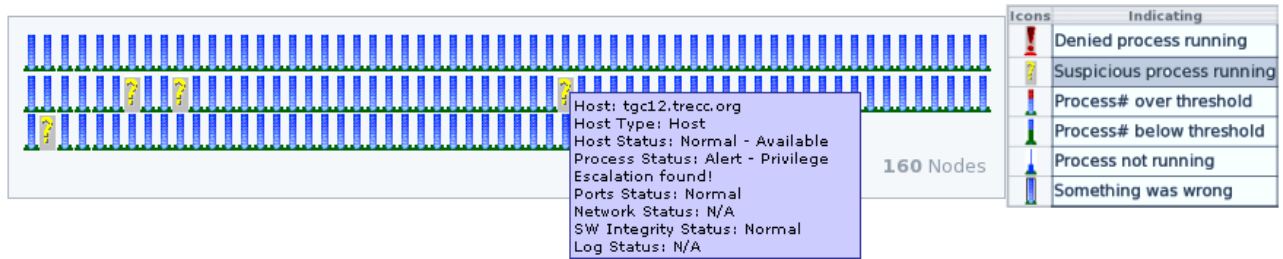


Fig. 5. Visualization of privilege escalation events on a cluster

In order to integrate the privilege escalation detection component with Clumon and the NVisionCC systems, it must provide each of the three components of Clumon with the functionality required for this new sensor.

- *Information Collection* – Like the basic version of Clumon, the information collection aspect of the plugin relies on PCP to gather data. It queries each node for each process running on that node, and also retrieves the owner and parent of the process. This allows the monitoring system to construct a process tree for each node, with any changes in process ownership visible in the tree. All of this information is inserted into the Clumon database.
- *Security Analysis* – The data analyzer component of the plugin implements the detection algorithm described in Section 6 and shown in Figure 4. It retrieves the necessary data from the Clumon database and executes the algorithm on each process on all nodes of the cluster. If any suspicious activity is detected, an alert is inserted into the database for display by the visualization component.
- *Visualization* – The privilege escalation detection plugin uses the visualization component common to Clumon and all NVisionCC components. The tool provides a graphical display representing all nodes in the cluster, as shown in Figure 5. Each node is coded with an icon to represent its current status. A blue bar indicates that the node is operating normally. The question mark icon indicates that a possible security event has occurred.

The visualization interface allows an administrator to obtain more information about the status of the node by hovering the mouse cursor over the node in question. The resulting pop-up window is also shown in Figure 5. Clicking on the node shows a detail view displaying the process tree with the suspicious processes highlighted. This view might look something like Figure 3. When combined with the other NVisionCC modules, this allows an administrator to quickly gain a comprehensive awareness of the situation.

## 8 Conclusion

Unauthorized privilege escalation allows an attacker to circumvent security mechanisms and gain unrestricted ability to damage or misuse cluster resources, and therefore is a grave threat to all computer systems. We present a technique for detecting this type of attack in Unix-based computing systems. Our technique involves analyzing the structure of the process tree looking for processes with `setuid` parents, and with grandparents owned by a different, non-administrative user. When this pattern is detected, an alarm is raised.

The technique presented in this paper is intended to be one of many indicators of malicious activity in a cluster computing system that can be correlated to generated warnings of higher accuracy. If just one of the sensors in the security monitoring framework is triggered, a low priority alert is generated. However, if multiple sensors all suggest that an intrusion is in progress, the probability of a false positive is dramatically reduced. These components are combined in the NVisionCC cluster security monitoring system to provide a centralized interface for managing the sensor parameters and viewing any alerts that are generated.

Although the technique presented in this paper does not itself exploit the emergent properties of clusters, it is an essential complement to the NVisionCC sensors that follow from emergent properties. Privilege escalation detection is applicable to cluster resources where the emergent properties are difficult or impossible to exploit. It is therefore an requisite component of the NVisionCC suite of tools.

We have performed a preliminary study to assess the performance impact of using PCP to monitor the state of cluster nodes [12]. These results suggest that the monitoring framework has minimal impact on the performance of common high-performance computing benchmarks, even when monitoring data is gathered as

often as once per second. Although we have not yet performed a full, large-scale performance analysis, such a study is underway and will be published in a future paper.

## Acknowledgments

We would like to thank Peter Enstrom for his initial work on this approach to detection of unauthorized privilege escalation.

## References

1. A. Acharya and M. Raje. MAPBox: Using parameterized behavior classes to confine applications. In *9th USENIX Security Symposium*, 2000.
2. Hackers hit supercomputing giants. Associated Press, April 2004.
3. CLUMON project website. <http://clumon.ncsa.uiuc.edu/main.html>.
4. C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton. StackGuard: Automatic adaptive detection and prevention of buffer-overflow attacks. In *Proceedings of the 7th USENIX Security Conference*, 1998.
5. R. K. Cunningham and C. S. Stevenson. Accurately detecting source code of attacks that increase privilege. In *Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection (RAID)*, 2001.
6. E. Grevstad. CPU-Based security: The NX bit. *www.hardwarecentral.com*, May 2004.
7. P. Hope. Using jails in FreeBSD for fun and profit. 27(3):48–55, June 2002.
8. <http://isec.pl/vulnerabilities/isec-0021-uselib.txt>, January 2005.
9. D. Kilpatrick. Privman: A library for partitioning applications. In *Proceedings of Freenix 2003*, 2003.
10. C. Ko, G. Fink, and K. Levitt. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Proceedings of the 10th Annual Computer Security Applications Conference (ACSAC)*, 1994.
11. C. Ko, M. Ruschitzka, and K. Levitt. Execution monitoring of security-critical programs in distributed systems: a specification-based approach. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.
12. G. A. Koenig, X. Meng, A. J. Lee, M. Treaster, N. Kiyancilar, and W. Yurcik. Cluster security with nvisioncc: Process monitoring by leveraging emergent properties. In *IEEE Cluster Computing and Grid (CCGrid)*, 2005.
13. B. Krebs. Hackers strike advanced computing networks. *Washington Post*, April 2004.
14. C. Ozancin. Securing the linux environment: Programs that need root access. pages 42–45, March/April 2001.
15. Performance co-pilot website. <http://oss.sgi.com/projects/pcp>.
16. N. Provos. Improving host security with system call policies. In *Proceedings of the 12th USENIX Security Symposium*, 2003.
17. N. Provos, M. Friedl, and P. Honeyman. Preventing privilege escalation. In *Proceedings of the 12th USENIX Security Symposium*, 2003.
18. A. Purohit, V. Navda, and T. Chiueh. Tracing the root of “rootable” processes. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC)*, 2004.
19. J. C. Rabek, R. K. Cunningham, and R. I. Khazan. Detecting privilege-escalating executable exploits. In *ICDM Workshop on Data Mining for Computer Security (DMSEC)*, 2003.
20. T. Roney. Cluster monitoring at NCSA. In *Second LCI International Conference on Linux Clusters*, 2001.
21. Multiple unix compromises on campus. <http://securecomputing.stanford.edu/alerts/multiple-unix-6apr2004.html>, April 2004. Stanford ITSS Security Alert.
22. A. Tucker and D. Comay. Solaris zones: Operating system support for server consolidation. In *3rd Virtual Machine Research and Technology Symposium*, May 2004.
23. J. Xu, Z. Kalbarczyk, and R. Iyer. Transparent runtime randomization for security. In *Proceedings of the 22nd Symposium on Reliable Distributed Systems (SRDS)*, 2003.
24. W. Yurcik, G. A. Koenig, X. Meng, and J. Greenesid. Cluster security as a unique problem with emergent properties: Issues and techniques. In *Fifth LCI International Conference on Linux Clusters*, May 2004.
25. W. Yurcik, X. Meng, and N. Kiyancilar. NVisionCC: A visualization framework for high performance cluster security. In *CCS Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)*, 2004.
26. W. Yurcik, X. Meng, and G. A. Koenig. A cluster process monitoring tool for intrusion detection: Proof-of-concept. In *IEEE Local Computer Networks (LCN)*, 2004.