Fine-Grained Tracking of Grid Infections

Ashish Gehani
Computer Science Laboratory
SRI International
Menlo Park, CA 94025, USA
Email: ashish.gehani@sri.com

Basim Baig, Salman Mahmood, Dawood Tariq, Fareed Zaffar
Department of Computer Science
Lahore University of Management Sciences
Lahore 54792, Punjab, Pakistan
Email: {basim.baig,salman.mahmood,dawood.tariq,fareed.zaffar}@lums.edu.pk

***Abstract*—Previous distributed anomaly detection efforts have operated on summary statistics gathered from each node. This has the advantage that the audit trail is limited in size since event sets can be succinctly represented. While this minimizes the bandwidth consumed and helps scale the detection to a large number of nodes, it limits the infrastructure's ability to identify the source of anomalies. We describe three optimizations that together allow fine-grained tracking of the sources of anomalous activity in a Grid, thereby facilitating precise responses. We demonstrate the scheme's scalability in terms of storage and network bandwidth overhead with an implementation on nodes running BOINC. The results generalize to other types of Grids as well.**

***Keywords*-anomalies, correlation, filtration, lineage, monitoring, provenance, temporal, vaccination**

I. Introduction

The typical Grid's infrastructure is exposed to the Internet at large. For example, the widely deployed Globus [15] uses a service-oriented architecture with daemons on each node to receive callbacks. Instances of BOINC [2] (which is used by the popular application SETI@home [49]) share a host runtime environment that may be infected with malware [41] and programs controlled by pseudonymous users. While a substantial amount of research has been undertaken to verify the correctness of the results computed on Grid nodes [39], considerably less effort has been invested in checking that the computing processes are performing as expected.

Grid applications are written to solve particular scientific or engineering problems. Consequently, the customized portions of the software may not undergo the quality assurance process of more mature, longer-lived software products and may contain vulnerabilities at the time of deployment. Combined with the large attack surface, this results in substantial exposure for the Grid.

As the computational and networking resources devoted to Grids grow, applications running on them become increasingly attractive to attackers. An attacker who gains control of a program running on a single node can leverage the pre-established trust relationships (present in Globus, for example) between Grid resources to attain control over a large collection of them. This threatens the data and programs of all the other users of the Grid. Even when clients do not allow inbound connections or single sign-on authentication, vulnerabilities in Grid applications manifest on all the nodes where it is executing, enabling an adversary to gain control over a large set of resources using infection vectors such as worms or viruses.

If a Grid application is successfully exploited, it can be used to launch a large-scale attack against Internet-connected hosts that are outside the Grid. However, such an attack would divulge the malware's presence and most likely lead to its removal from the Grid nodes. Of greater concern is the utilization of the Grid's resources for unauthorized activity. In recent years, collections of compromised hosts have been abused by *grayware* for activities such as serving advertisements, relaying spam, and illicitly storing copyrighted music, video, or programs. Since detection would lead to its elimination, the grayware typically operates with stealth, thereby maximizing its life and the resulting commercial gain. Even when the Grid application is cryptographically signed, as is the case with code that uses BOINC, verification attests the integrity only at the time the code is being installed and loaded from disk. If the code is compromised during execution, its integrity in future runs can no longer be relied upon.

Monitoring a Grid to watch for intrusions requires that the activity on each node must be audited and compared to a description of acceptable activity [11]. Since the event stream must be trusted, it must originate in the operating system. The most common source utilized is the record of system calls invoked, as this is the interface through which applications interact with the system [21]. Detecting Grid-wide attacks requires that the audit trails from all the nodes be collected and analyzed. However, recording the complete invocation signature of all system calls along with the arguments passed to them would result in a deluge of data. Consequently, current systems correlate activity from a distributed set of nodes by tracking the type of events but not their arguments [32].

We focus on mechanisms to facilitate fine-grained tracking of the source of Grid infections. Our approach (i) compresses the event stream at the point of origin, reducing the computational and storage overhead on each node, (ii) allows a tradeoff in the frequency and bandwidth of network communication, enabling faster responses at the cost of increased bandwidth or decreased accuracy, and (iii) enables detailed provenance analysis of most anomalous activity.

II. Motivation

We first expand on the reasons why we believe Grid infrastructures are at substantial risk.

A. Vulnerable code

Building software that is secure requires a significant investment of resources. It starts when use cases are being developed, adding the need to create abuse cases as well [22]. When the code's architecture is being designed, a risk analysis should be undertaken to determine if the expected losses from abuse can be tolerated [46]. When code is being developed, static analysis tools should be used to flag security vulnerabilities [47]. Software quality assurance tests should be augmented with security tests that confirm resistance to abuse [8]. Configuration testing must be extended with penetration tests to verify that weaknesses are not introduced in some states [48].

Grid infrastructure is designed to be used for an extended period of time. It is therefore worth making the investment to ensure that the software is developed to be secure. Grids are typically funded and managed by consortiums of universities, companies, and government bodies that can afford to make the necessary commitment of resources to achieve strong security. In contrast, Grid applications are produced by small groups of researchers, scientists, and engineers. These users may not be trained in secure software engineering, nor are they typically provided a budget that affords it. Frequently, they develop a piece of code to answer a focused question and may not reuse the code in the future. As a result, they are motivated to develop and deploy their code as rapidly as possible. They may view the overhead of secure software development practices as unwarranted if their programs are expected to have a short life cycle. Software engineering research [6] predicts that Grid applications are therefore likely to contain bugs at the time of deployment. An adversary will be able to exploit some fraction of these bugs to breach security.

B. Exposed services

Grid computing projects are often structured as collections of Web services [18], each of which operates independently from the others and provides distinct functionality. To facilitate interaction between components, the projects use standardized data representations, typically in XML [51], with available resources described in a language such as WSDL [50], and use communication protocols like SOAP [38] for remote invocations. Such a service-oriented architecture lets legacy systems be seamlessly integrated while simultaneously allowing them to be dynamically replaced at a later point [17]. It provides several other benefits as well, such as utilizing explicit service contracts, abstracting utilities, using loose coupling to minimize dependencies, promoting code reuse, facilitating composition of applications, and allowing the discovery of new functionality.

Applications running on such Grids can be dispatched to arbitrary nodes in the system from where they may initiate connections to (and receive callbacks from) external nodes on the Internet, such as application-specific database servers. Erecting a firewall between the Grid resources and the rest of the Internet will impede the normal operation of such applications. The ramification of using such an open architecture is that it creates a large attack surface for external adversaries to target. The probability of an attacker being able to detect a vulnerability in one of the services running on a Grid node is substantially higher than if that attacker had to find a weakness in the authentication service that is the sole externally accessible part of a firewalled distributed system.

C. Automatic privilege escalation

Early distributed computing authentication systems such as Kerberos [4] allowed a user to use the same credentials to access the resources of each node. However, the user was expected to furnish proof of identity (such as a password) to each node independently. As batch computing infrastructures like Condor [26] were built they incorporated functionality to transparently delegate rights from one node to another. Since Grid infrastructures farm subtasks out to a large number of nodes, it is necessary to configure the underlying infrastructure to utilize the delegation mechanisms to facilitate the migration of code along with its permissions to arbitrary nodes in the system without manual intervention. This convenience introduces system-wide risk in the event that a malicious user gains access to a single node in the system. That user can leverage the predefined trust relationships to execute code at many other nodes in the Grid infrastructure. Thus, an attacker who is able to compromise a part of an application running on a single node will be able to rapidly escalate access to a large subset of the Grid.

D. Attractive attack platform

The motivation of cyber attackers has changed over time. Their goals are now often economically driven rather than being limited to vandalism for community recognition [3]. In addition to directly targeting financially lucrative targets that are connected online, attackers attempt to garner revenue using commandeered resources to launch email mass marketing campaigns [20], host mimicked Web sites to collect identity information that can be sold on the black market [9], and boost advertising revenues of clients by generating fraudulent clicks on their Web pages [24].

Grid nodes form an attractive target to attackers for a variety of reasons. Each node is usually well provisioned with fast processors, large amounts of memory and disk storage, and high network bandwidth. Dedicated nodes (such as those used in TeraGrid [42]) are most likely poorly provisioned with hardware needed for interactive computing, such as graphics accelerators and sound cards, and input/output peripherals like displays and speakers. However, the absence of these devices is of no consequence for the type of workload that economically incentivized attackers run. Further, such Grid nodes are deployed in configurations least likely to detect and interfere with an attacker's activity. The absence of interactive users who may note anomalous application behavior, the lack of

egress filters deployed by Internet service providers to curtail spam originating from their customers' infected computers, and the dearth of intervening institutional firewalls make such Grid nodes particularly suitable for use by attackers.

E. Significant consequences

Data emitted from a Grid is usually the product of a significant amount of time and computational resources. For example, each piece of information that physicists use from Fermilab's Collider Detector is the output of a month of processing dozens of terabytes of raw data [40]. Similarly, the cost to analyze a single protein stored in the Protein Data Bank is $200,000 [35]. The cost of producing such data precludes its availability from an alternate source. Consequently, there is substantial adverse economic impact if large amounts of data in a Grid must be discarded because of anomalous activity. Minimizing this requires the ability to hone in on which information has been tainted.

F. Unreported incidents

Grid middleware, such as the Globus toolkit [15], is composed of code that is meant to be reused for many years. Further, it is distributed to the public and utilized by many users. Consequently, when a vulnerability in the code is discovered, in addition to addressing it, knowledge about the weakness is shared. Hence, external agencies that track security vulnerabilities and exploits are made aware of it and sources such as NIST's ICAT database [23] have a record of the issue. In contrast, application code (that executes on a Grid) is viewed as internal to the group developing it. When vulnerabilities are found, they may be addressed but not reported to external agencies. Thus, tallying known exploits for Grid application code using the ICAT or equivalent database results in a significantly more sanguine view of the issue than is warranted.

III. Architecture

Our focus is the development of a monitoring architecture that can scale to hundreds or thousands of nodes while tracking infection sources to individual files and processes. We do not create any new intrusion detection algorithms. Instead, we provide a framework for implementing previous schemes, such as those of Malan and Smith [29], and Oliner et al. [32].

A. Community monitoring with sets

The application community [27] paradigm postulates that if many instances of a piece of software collaborate and share information about anomalous activity as it is occurring, then despite the fact that the first few nodes may be damaged, the remaining can be protected. Grid applications are particularly well suited to utilize this model because they perform a large computation by parallelizing it and executing the same operations with different inputs on multiple nodes. It is precisely the portion that is distributed that is likely to contain vulnerabilities (as reasoned in Section II-A) since it typically has a shorter life cycle with limited effort expended

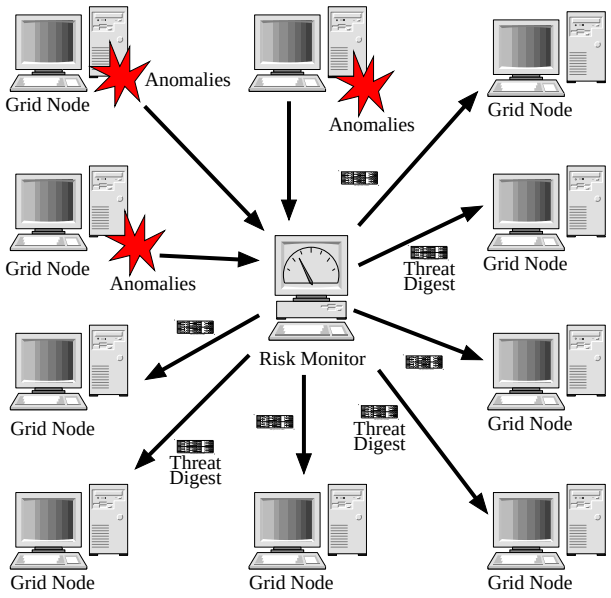

Fig. 1. Anomalies on Grid nodes are reported to a central monitor that correlates them and broadcasts a *threat digest* to all nodes in the community.

on hardening the code. Instead of requiring changes in the user's code, we audit and correlate the system call invocations of the target application.

System call-based intrusion detection [21] builds a profile of normal application behavior by recording all the observed sequences of a predefined length during a training period and then subsequently flagging as anomalous any sequence of system calls that is not in the profile. Distributed versions [29], [32] require the stream of all anomalous sequences to be sent to a central monitor that correlates the activity.

We leverage two properties of these anomaly detection algorithms. First, they operate on tuples of system calls, treating them as integral elements. Second, the three primary steps can be framed as set operations:

- The definition of normal application behavior consists of the union of all tuples observed during training. Analogously, anomalous activity is considered to occur when tuples seen in the operational phase do not satisfy membership queries for the normal behavior set.
- Logging anomalous activity consists of a Grid node constructing the union of anomalous tuples that it has observed and sending this set to the central monitor.
- Determining when the same activity is occurring on two Grid nodes can be effected by taking the intersection of the sets of anomalies originating at the nodes. If the set is empty, there is no correlation. The larger the cardinality of the set intersection, the greater the correlation between the anomalous activity occurring at the two nodes.

Finally, the same properties also allow us to define a fourth operation that acts on sets of tuples of system calls:

- We can create a Grid *vaccination* by assembling a set of dangerous tuples, which if observed on a Grid node indicate the presence of an attack. This can be effected

at the central monitor by constructing the union of all tuples that occur in the anomaly sets originating from at least some predefined threshold τ number of Grid nodes.

Anomaly detection systems can trade the accuracy with which they can identify previously known versus currently unknown attacks. Heavier training with known threats reduces the probability of a known threat generating a false positive alert. Simultaneously, this increases the likelihood that a new attack will go unnoticed (since the system's notion of an attack is more closely associated with the known attacks). Similarly, training with a more diverse set of attacks results in a system that has lower false negatives (since it is able to discern a wider variety of deviant phenomena) but also has a higher rate of false positive alerts (since normal behavior is likely to exceed one of the (lower) internal thresholds for flagging activity as anomalous). Consequently, maintaining a sufficiently low false negative rate is likely to result in substantial anomaly streams being sent from Grid nodes to the central monitor where correlation is performed and vaccinations are defined.

Oliner et al. [32] found that their central server received an average of 10 Mb/s and up to double that with 35 clients being monitored. This motivated us to frame our monitoring architecture in terms of set operations. The advantage this provides is that each set can be represented with a *Bloom filter* [5], which is a space-efficient data structure that can answer set membership queries in constant time without any false negatives and a false positive probability that can be made arbitrarily small by selecting a large enough size for the filter [7]. In Section IV, we describe the resulting decrease in storage requirements.

B. Near real-time correlation

The security of a Grid application is a function of global state that is distributed across many nodes with significant amounts of activity cumulatively occurring at each moment. A typical Grid computing node runs at 2 GHz and accesses a 64-bit memory at 667 MHz, receiving as many as 20 bits per cycle on average. Commodity network interfaces operate at 10 Gb/s, allowing intra-cluster communication at this speed. Even end-to-end links between Grid clusters that are separated by large distances now approach the 10 Gb/s throughput level [14], [30]. A processor can access as many as 5 bits per cycle on average from a remote node, the same speed that local access to memory was a decade ago. While this bandwidth is enough for most monitoring applications, including many security-related applications, it is still insufficient for nodes to share their system call-level audit trails in real time since thousands of events occur each second and logging a single call takes hundreds of bytes on average if its parameters are included, yielding hundreds of kilobytes of audit data from each node. On a Grid with a hundred nodes, each would have to receive tens of megabytes each second. Attempting to share the details of all system activity in a distributed system would saturate the communication infrastructure.

Despite the volume of data resulting from monitoring hundreds of nodes, real-time correlation remains feasible because

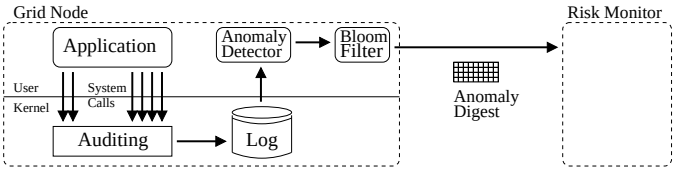

Fig. 2. To monitor hundreds of Grid nodes without saturating the network with audit data, the anomaly stream is transformed into Bloom filter digests.

the audit trails have low entropy for three reasons. First, the space of possible events is substantially larger than the set of values present in an audit trail. For example, an event logged with a hundred ASCII characters could have $256^{100} = 2^{800}$ possible values but in practice there are significantly fewer possibilities (even accounting for variation in parameter values). Second, much activity in a computational workload is repetitive with numerous tuples of event sequences occurring frequently, reducing the number of bits needed to represent them. Third, the training profile for what constitutes normal application behavior serves as a compression index that eliminates a large subset of symbols that would otherwise need to be sent in the data being transmitted from a Grid node to the central monitor.

Each Grid node maintains an *anomaly digest* as a Bloom filter, which tracks the set of anomalous sequences of system events that have been observed in the current interval of time, termed an *epoch*. Tuples of anomalous events are idempotently inserted into the Bloom filter as they are observed. At the end of an epoch, the Grid node transmits the anomaly digest to the central monitor, as illustrated in Figure 2. The correlation process maintains a *counting filter* [13] for each epoch, with as many buckets as there are bits in the Bloom filter used for anomaly digests. Each time the correlation process receives an anomaly digest, it increments each bucket in the counting filter that corresponds to a bit that is set in the anomaly digest. At the end of an epoch, it creates a *threat digest*, which is a Bloom filter of the same width as the anomaly digest. For each bucket in the counting filter with a value that exceeds a predefined threshold τ, the corresponding bit in the threat digest is set to one. The resulting threat digest is broadcast as a vaccination to all Grid nodes, as illustrated in Figure 1. When the threat digest is interrogated about any anomaly that was observed on more than τ nodes, it will yield a positive response. This allows a Grid node to use a vaccination to flag dangerous tuple sequences in real time as they are occurring based on the knowledge that the anomalous tuple had already occurred on τ other nodes.

During an epoch, a Grid node's notion of anomalous activity changes from that of the server. The server's view becomes consistent with that of the clients at the end of an epoch when it receives digests from all of them. To bound the inconsistency, a Grid node can repeatedly halve the length of an epoch until the level of anomalous activity captured in a single digest drops below a predefined threshold. However, each time the epoch is halved, the cumulative bandwidth used

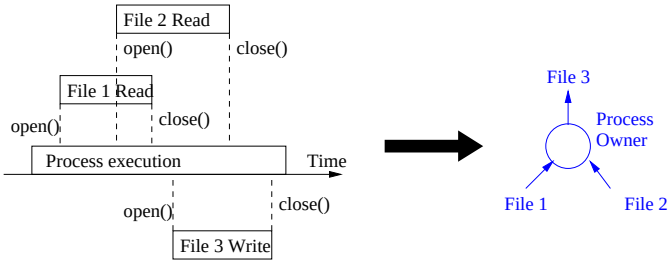

Fig. 3. By tracking the data flow between processes and the filesystem, a provenance graph representation can automatically be created.

to transmit digests to the server would double. To avoid this, the Bloom filter can be halved in size by replacing it with the bitwise OR of its first and second halves. This results in an increase in the false positive rate [7] when checking whether a potentially anomalous sequence is present in the threat digest. However, this automatically tunes the monitoring architecture to be more responsive when the level of anomalous activity increases while simultaneously maintaining a fixed bound on the network bandwidth consumed.

C. Anomaly provenance

When a Grid node records the occurrence of a sequence of system events that are present in one of the threat digests it has received from the central monitor, the node is under attack. If there was a detailed log of all activity on the node, including the arguments passed to system calls, the source of the attack could be inferred by tracing backward from the anomalous system calls. However, recording all the arguments of every system call invoked generates a large volume of data. We address the apparent tension with the insight that only a small subset of system call arguments needs to be audited to determine the provenance of the anomalous activity. Specifically, recording just the file and process identifiers provides a rich set of information with which to reason. This has the advantage that it results in a very small increase in the storage needed compared to the case where only the system calls are being logged without arguments.

On each Grid node, we maintain a provenance database populated with two types of records. The first is for the identifier and attributes of every file that has been read or written. We adopt the convention of identifying a file using both its logical location and its last time of modification to disambiguate different versions of the same file, which avoids cycles in the provenance graph. The second type of record is for process identifiers and contains associated attributes. When a file is read or written by a process, new records are created or existent ones updated (if the file is being read, for example), as appropriate. This suffices for constructing a data flow graph, as depicted in Figure 3, that is consistent with the model used by Grid projects [43].

When a Grid node flags an event sequence as anomalous, the arguments can be extracted. Any arguments that are file or process identifiers are looked up in the node's provenance database. The entire provenance graph of the process or file

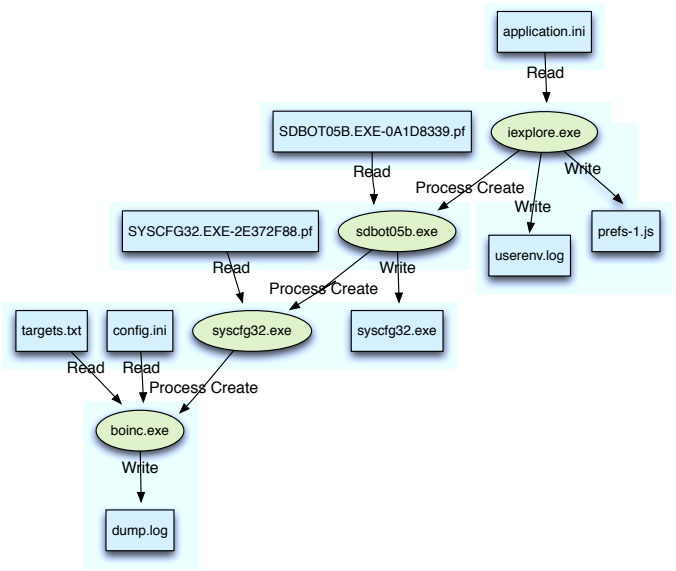

Fig. 4. An anomalous tuple includes a file write event with an argument *dump.log*. Inspecting the provenance graph (where ovals are used for processes and rectangles for files), the source can be tracked. In this synthetic example, Internet Explorer (*iexplore.exe*) had a buffer overflow through which a bot (*sdbot05b.exe*) was installed. The bot masks itself at initialization using Windows *CreateRemoteThread()* to inject activity into the address space of BOINC (*boinc.exe*), writing out a file (dump.log) that BOINC otherwise would not write.

can then be extracted, allowing a precise characterization of the source of the anomaly, as illustrated in Figure 4.

IV. Evaluation

To evaluate our monitoring architecture, we developed components that allowed us to gather data from real workloads. Since Windows-based hosts are the most likely platform to be infected by malware, we conducted our experimental analysis in the context of the volunteer computing Grid software BOINC [2]. The implementation was done on a testbed where the prototypical Grid node ran Microsoft Windows XP (with Service Pack 3 installed) on a 2.8 GHz Intel Core2Duo processor with 3.5 GB of memory.

The normal workload consisted of version 6.10.43 of BOINC executing while being audited with version 2.7 of the Process Monitor tool [34], which logs system activity. Events related to the file system, Windows registry, process life cycle, network communication, and thread management were recorded. Our monitoring component used the Open Bloom Filter library [33] to construct anomaly digests.

We synthetically constructed a scenario where BOINC had been attacked and was being used as a relay for spam. Such a situation could occur in a variety of ways. For example, malware may use the BOINC process to mask its own activity by adjusting the access control configuration and using the Windows *CreateRemoteThread()* interface to inject itself into the address space of the BOINC process at start-up. Verifying the integrity of the BOINC application binary will not detect this. To mimic the application behavior induced by an infection with a spam relay, we used MailBoy 2004 [28], which includes

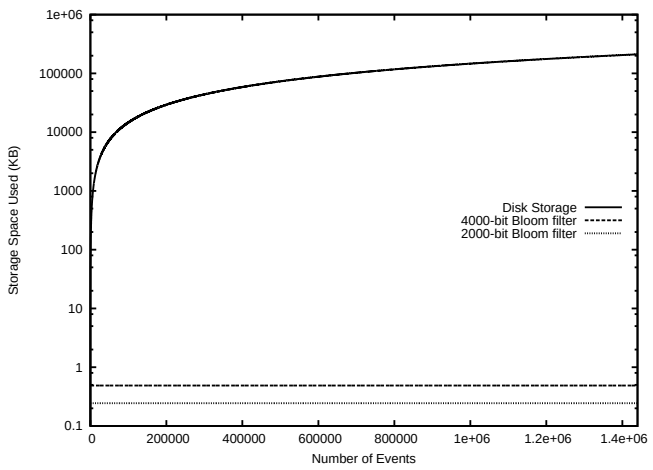

Fig. 5. The cumulative amount of storage used for the raw log versus a Bloom filter containing 11-tuples of all observed events, as a function of the number of events that have occurred in the system. The y-axis is log scale since the log size exceeds 200 MB after running for a day while the Bloom filter size remains 0.25 or 0.5 KB.

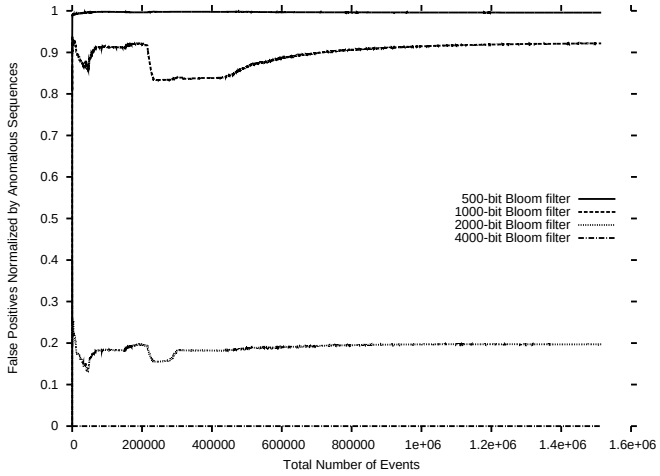

Fig. 6. Plots for a normal workload run over the course of 24 hours and 20 minutes.

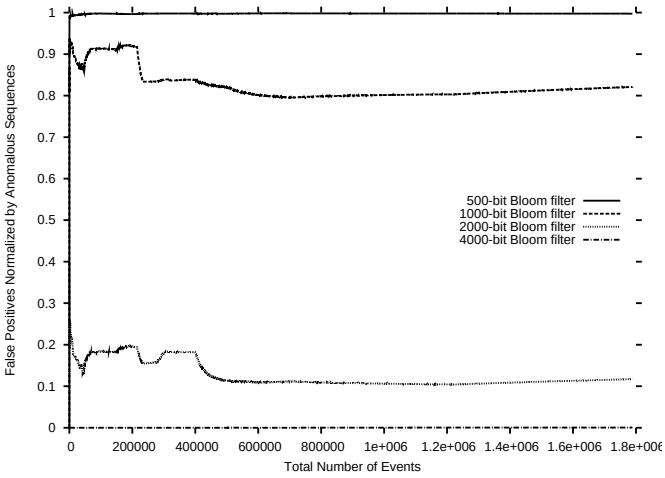

Fig. 7. Plots for a normal workload with malware mixed in and run over the course of 24 hours and 20 minutes.

internal SMTP and DNS servers. In the evaluation, MailBoy created 20 threads with timeout periods of 30 seconds to send messages to a mailing list with 1,700 email addresses.

During preliminary analysis, we found that using 6-tuples of Windows event sequences did not provide sufficient discrimination between anomalous and normal activity. We increased the tuple size until it was 11, at which point we achieved good detection performance. Figure 5 shows the amount of storage required to record all the 11-tuples of system event sequences that occur when running BOINC for 24 hours and 20 minutes. The graph plots the log file size as a function of the number of events that have been observed. Note that this is for a single application running on just one Grid node. At the end of the evaluation period, 1.5 million events had occurred that took 216 MB to store. The raw log for even five Grid applications running on 100 nodes would require a server to receive over 100 GB of log data in a day. In contrast, sending a 0.5 KB (4,000 bit) Bloom filter every minute would require that the server receive a total of 360 MB of data (from 5 applications on 100 nodes).

A primary concern when sending anomaly digests instead of the actual event tuples is the misidentification of normal activity as anomalous by the correlation process. This could occur if the central monitor checks whether a tuple is present in an anomaly digest from a Grid node and receives a false positive confirmation. By making the Bloom filter large enough, its false positive rate can be reduced below any predefined threshold. However, implementing our real-time correlation process potentially requires folding Bloom filters by factors of 2^n for small n. To characterize the effect of these actions, we performed the following analysis.

During normal system operation on a Grid node we collected all the event tuples that were reported as anomalous. Simultaneously, we constructed Bloom filters of size 500, 1000, 2000, and 4000-bits, corresponding to the scenario where the monitoring architecture starts with 4,000 bit filters and then folds them by factors of 2, 4, and 8. Next, as each new event occurred in the system, we checked to see if the resulting tuple was anomalous and if it was, whether it had been seen earlier. If it had not been observed, we queried the Bloom filters to check that they reported false. If they reported true, we knew that this was a false positive. As the system ran, we counted the frequency with which such false positives were being observed for all the filters (of different sizes). Since this count is expected to increase as time progresses and more activity (including that which is anomalous) occurs, we normalized the false positive rate by the number of anomalous tuples that had been observed, as plotted in Figure 6. Figure 7 shows the same analysis for a workload where malware was injected.

In both the normal and infected workloads we found that if we operated with anomaly digests using 4,000-bit Bloom filters, the false positive rate was negligible. However, if we "folded" the digest even once, the false positive rate increased

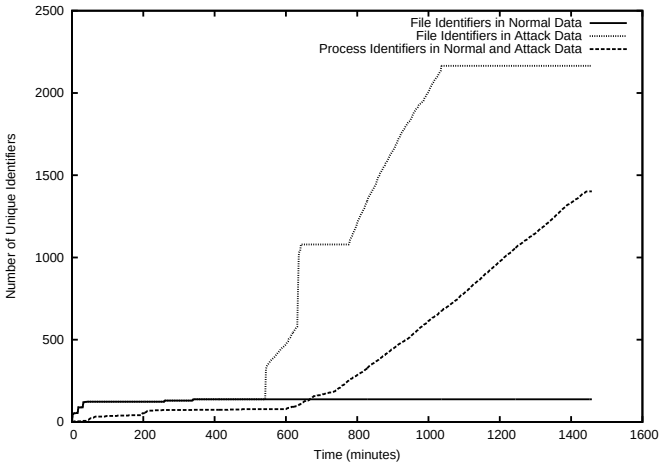

Fig. 8. The number of unique file and process identifiers recorded in the normal and infected workloads, as a function of time. The normal workload had over 1.6 million events and the infected workload had close to 2 million events. Both ran over the course of 24 hours.

to over 10%, which would be unacceptable for any anomaly detection system that used our monitoring architecture. If the digests were folded twice or thrice into 1,000 or 500 bits, the false positive rate increased dramatically. (The increase and decrease in false positive rates visible in the early part of the 1,000- and 2,000-bit plots is assumed to relate to the activity level of the normal workload.) We concluded that the smallest digest size to use was 4,000 bits. During normal operation, larger digests can be used and sent less frequently to remain below any predefined bandwidth threshold. When anomalous activity increases, the digests can then still be folded (from 32 KB to 16 KB, 8 KB, or 4 KB, for example).

Finally, we wanted to estimate the amount of storage that would be required on each Grid node to maintain the provenance database. To do this, we counted the number of unique identifiers for files and processes that were encountered during the 24-hour period that the normal and infected workloads were executed. In particular, a file at the same location but that had been modified will have a different identifier, allowing causal inferences to be made accurately. Figure 8 shows that over the course of a day, the number of file identifiers remains under 100 in the normal workload and under 2,000 in the infected workload. The number of process identifiers grows over the course of the day but to a smaller extent. This shows that only a few megabytes of storage will be needed to maintain the provenance database.

V. Related Work

Globus [15], one of the most widely used Grid infrastructure projects, developed the Grid Security Infrastructure (GSI) [16] to protect users from external abuse. It provides a *single sign-on* authentication service that allows Grid software to distinguish legitimate remote requests from unsanctioned Internet traffic. Since Grid resources are utilized by numerous principals from multiple organizations, the GSI also provides an authorization service to prevent internal abuse. The GSI leverages local site access control mechanisms to enforce security policy that prevents Grid insiders from breaching the confidentiality and integrity of other users' data. Similarly, numerous other Grid projects [1], [10], [12], [19], [31], [36], [44], [45] incorporate analogous security services. While these systems focus on proactive prevention of policy violations, others effect *post fact* detection of security breaches by correlating intrusion alarms from Grid nodes [25], [37].

Forrest et al. demonstrated that anomalous process behavior on a host could be detected by monitoring sequences of system calls [21]. During training, a set of "normal" sequences is recorded. Subsequently, sequences that fall outside the set serve as a predictor of anomalous activity on the host. However, the approach has an inherent tradeoff. A larger training window reduces false positives — that is, the flagging of innocuous sequences — but also increases false negatives — that is, the failure to flag malicious activity because it mimics normal sequences in the training set. Keromytis proposed that anomaly detection could be improved — that is, false positives could be reduced without increasing false negatives — by leveraging the homogeneity in an "application community" [27]. Oliner et al. showed this with temporal correlation of anomalous system call sequences from concurrently executing instances of the same program [32]. Malan and Smith [29] demonstrated that worms and bots could be detected using a complementary hypothesis — that strong temporal correlation in emitted system call sequences indicated the presence of the malware. None of these efforts investigated how to manage the large volume of data that results from auditing system call arguments, a step that is necessary for fine-grained response.

VI. Conclusion

We described a scalable monitoring architecture for use with Grid applications. It uses Bloom filters at each Grid node to create succinct digests of anomalies seen in an epoch. These digests are sent to a central correlation process that uses a counting filter to combine the anomaly digests. A threat digest is constructed representing all anomalies that have occurred on at least a predefined threshold number of nodes. The threat digest is broadcast to all Grid nodes that use it to flag dangerous anomalies as they occur. The provenance of the file and process arguments of such anomalous events can be retrieved using a database maintained locally on the Grid node. We reported the results of an empirical analysis that we used to validate our approach.

Acknowledgments

We thank Steven Dawson, Drew Dean, and Patrick Lincoln for helpful discussions, and the anonymous reviewers for their useful comments.

This material is based upon work supported by the National Science Foundation under Grant OCI-0722068. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] http://www.mobilegrids.org/

[2] David P. Anderson, BOINC: A System for Public-resource Computing and Storage, 5th IEEE/ACM International Workshop on Grid Computing, 2004.

[3] Paul Barford and Vinod Yegneswaran, An Inside Look at Botnets, Special Workshop on Malware Detection, Advances in Information Security, Springer-Verlag, 2006.

[4] Steven M. Bellovin and Michael Merritt, Limitations of the Kerberos Authentication System, Proceedings of the USENIX Conference, 1991.

[5] Burton H. Bloom, Space/time Trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, Vol. 13(7), 1970.

[6] B. W. Boehm, Software Engineering Economics, IEEE Transactions on Software Engineering, Vol. 10(1), 1984.

[7] A. Broder and M. Mitzenmacher, Network Applications of Bloom Filters: A Survey, Internet Mathematics, Vol. 1(4), 2005.

[8] Hao Chen and David Wagner, MOPS: An Infrastructure for Examining Security Properties of Software, 9th ACM Conference on Computer and Communications Security, 2002.

[9] Neil Chou, Robert Ledesma, Yuka Teraguchi, Dan Boneh, and John C. Mitchell, Client-Side Defense Against Web-Based Identity Theft, 11th Annual Network and Distributed System Security Symposium, 2004.

[10] Daidalos, http://www.ist-daidalos.org/

[11] D. Denning, An Intrusion-Detection Model, IEEE Transactions on Software Engineering, February 1987.

[12] EGEE, http://www.eu-egee.org/

[13] Li Fan, Pei Cao, and Jussara Almeida, Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol, IEEE/ACM Transactions on Networking, Vol. 8(3), 2000.

[14] W. Feng, J. Hurwitz, H. Newman, S. Ravot, R. L. Cottrell, O. Martin, F. Coccetti, C. Jin, X. Wei, and S. Low, Optimizing 10-Gigabit Ethernet for Networks of Workstations, Clusters, and Grids: A Case Study, Proceedings of the ACM/IEEE Conference on Supercomputing, IEEE Computer Society, 2003.

[15] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, Vol. 11(2), 1997.

[16] I. Foster, C. Kesselman, G. Tsudik, and S. Tuecke, A Security Architecture for Computational Grids, Proceedings of the 5th ACM Conference on Computer and Communications Security, 1998.

[17] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke, Grid Services for Distributed Systems Integration, IEEE Computer, Vol. 35(6), 2002.

[18] Steve Graham, Doug Davis, Simeon Simeonov, Toufic Boubez, Ryo Neyama, and Yuichi Nakamura, Building Web Services with Java: Making Sense of XML, SOAP, WSDL, and UDDI, Pearson Education, 2001.

[19] Grid Provenance, http://www.gridprovenance.org/

[20] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee, BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation, 16th USENIX Security Symposium, 2007.

[21] Steven Hofmeyr, Stephanie Forrest, and Anil Somayaji, Intrusion Detection Using Sequences of System Calls, Journal of Computer Security, Vol. 6, 1998.

[22] Paco Hope, Gary McGraw, and Annie I. Anton, Misuse and Abuse Cases: Getting Past the Positive, IEEE Security and Privacy, May/June 2004.

[23] ICAT, http://icat.nist.gov

[24] B. J. Jansen, Click Fraud, IEEE Computer, Vol. 40(7), 2007.

[25] C. Leordeanu, L. Arif, and V. Cristea, Correlation of Intrusion Detection Information in Grid Environments, Complex, International Conference on Intelligent and Software Intensive Systems, 2010.

[26] M. J. Litzkow, M. Livny, and M. W. Mutka, Condor - A Hunter of Idle Workstations, 8th IEEE International Conference on Distributed Computing Systems, 1988.

[27] Michael Locasto, Stelios Sidiroglou, and Angelos Keromytis, Application Communities: Using Monoculture for Dependability, 1st Workshop on Hot Topics in System Dependability, 2005.

[28] MailBoy 2004, http://massmailersoft.com/index.asp

[29] David Malan and Michael Smith, Exploiting Temporal Consistency to Reduce False Positives in Host-based, Collaborative Detection of Worms, 4th ACM Workshop on Recurring Malcode, 2006.

[30] C. Meirosu, P. Golonka, A. Hirstius, S. Stancu, B. Dobinson, E. Radius, A. Antony, F. Dijkstra, J. Blom, and C. de Laat, Native 10 Gigabit Ethernet Experiments Over Long Distances, Future Generation Computer Systems, Vol. 21(4), 2005.

[31] Next Grid, http://www.nextgrid.org/

[32] Adam J. Oliner, Ashutosh Kulkarni, and Alexander Aiken, Community Epidemic Detection with Syzygy, 13th International Symposium on Recent Advances in Intrusion Detection, 2010.

[33] Open Bloom Filter, http://www.partow.net/downloads/OpenBloomFilter.zip

[34] Process Monitor 2.7, http://technet.microsoft.com/en-us/sysinternals/bb896645.aspx

[35] Andrej Sali, 100,000 Protein Structures for the Biologist, Nature Structural Biology, Vol. 5, 1998.

[36] SIMDAT, http://www.scai.fraunhofer.de/simdat.html

[37] M. Smith, F. Schwarzer, M. Harbach, T. Noll, and B. Freisleben, A Streaming Intrusion Detection System for Grid Computing Environments, 11th IEEE International Conference on High Performance Computing and Communications, 2009.

[38] SOAP, http://www.w3.org/TR/soap/

[39] Eugen Staab and Thomas Engel, Collusion Detection for Grid Computing, 9th IEEE/ACM International Symposium on Cluster Computing and the Grid, 2009.

[40] Stefan Stonjek, Morag Burgon-Lyon, Richard St. Denis, Valeria Bartsch, Todd Huffman, Elliot Lipeles, and Frank Wurthwein, Using SAM Data Handling in Processing Large Data Volumes, UK e-Science All Hands Meeting, 2004.

[41] Internet Security Threat Report: Vol. XV, Symantec Corporation, April 2010.

[42] TeraGrid, http://teragrid.org/

[43] Douglas Thain, Todd Tannenbaum, and Miron Livny, Condor and the Grid, Grid Computing: Making the Global Infrastructure a Reality, John Wiley, 2003.

[44] TrustCom, http://www.eu-trustcom.com/

[45] UniGrids, http://www.unigrids.org/

[46] Denis Verdon and Gary McGraw, Risk Analysis in Software Design, IEEE Security and Privacy, July/August 2004.

[47] David Wagner, Static Analysis and Software Assurance, 8th International Static Analysis Symposium, 2001.

[48] C. Weissman, Penetration Testing, Information Security: An Integrated Collection of Essays, IEEE Computer Society Press, 1995.

[49] D. Werthimer, J. Cobb, M. Lebofsky, D. Anderson, and E. Korpela, SETI@home — Massively Distributed Computing for SETI, IEEE Computing in Science and Engineering, Vol. 3(1), 2001.

[50] WSDL, http://www.w3.org/TR/wsdl

[51] XML, http://www.w3.org/XML/