METRICS FOR DETECTING COMPROMISED SYSTEMS IN DISTRIBUTED SYSTEMS

Shivaraj Tenginakai

ABSTRACT

Current electronic commerce systems are built using centralized client-server architecture; and their constituent processes are deployed to trusted computers. Therefore, attacks against these systems are characteristically data attacks; with the goal of accessing or compromising their confidential data. However with emergence of distributed applications, untrusted systems can now participate in electronic commerce. Attacks against these systems can compromise business processes that constitute electronic commerce. Advances in cryptography and network technology have made data security a tractable and well understood problem. However, process security continues to be a challenge. The goal of this paper is to show that some of these challenges may be due to misunderstanding of the problem; and a better understanding may lead to practical and effective solutions.

INTRODUCTION

Traditionally, electronic commerce systems have been built using computers that are housed in secure datacenters. For example, Amazon.com, eBay, or Google control the hardware, operating system, and applications on their systems. Therefore, for example, attacking eBay's auction algorithm or Google's search algorithm is not easy. However, with emergence of platforms such as Facebook and MySpace, it is now possible to build electronic commerce processes that may execute across untrusted systems. The behavior of such processes, therefore, may be more easily compromised.

Consider, for example, an auction process implemented using Facebook application platform. In various embodiments, this process may consist of set of participating auction agents on user computers. A malicious user may compromise the auction process by, for example, changing the clock on his or her computer.

Detecting compromised processes is currently an area of active research. Karnik and Tripathi describe Ajanta mobile agent that uses secure log files and Java security model to detect compromised behavior [1][4]. Further, Haeberlen et al. describe PeerReview system that also uses secure log files for detecting compromised systems [5]. These approaches, thus, reduce process security to characteristic data in secure log files; and use data security methods.

Using log files may not be practical in many situations. Some systems may not have sufficient resources; and further, their data may not be always available for analysis. For example, space may be at premium in some participating systems such as mobile devices; additionally, for systems with solid state storage frequent writes to log files may decrease the storage's lifetime. Additionally, suspected malicious users may be unwilling to provide access to their log files citing, for example, their rights against self incrimination.

Other approaches involve creating a sandbox for detecting malicious applications [1]. However, again these approaches are limiting as they may detect applications created with malicious intent; but they may be unable to detect a compromised instance of a well behaved application.

Using distributed system terminology compromised systems may be considered as *faulty* nodes; and theories developed for distributed systems may be applied for their detection. Current approaches to detecting faulty nodes fall into two categories: first, they rely on a set of correct nodes to detect faulty nodes; second, they use data generated by all nodes in the system. In Byzantine agreement problem studied by Lamport et al. they proved that for a system with k faulty nodes, at least $2k + 1$ correct nodes must be present to detect the k faulty nodes [2]. In second approach, developed by Vogels, nodes regularly provide and exchange information about their current state [3]; and based on this information nodes can infer their faulty counterparts. Both these approaches are again not practical for distributed electronic commerce applications; as the number of nodes may either be too large for them to participate in a Byzantine agreement protocol; or, fraudulent nodes may mask information about their state.

One reason for perceived difficulty in detecting compromised systems is that current approaches assume that compromised applications display Byzantine fault characteristics; that is deviations from their expected behavior is not deterministic. However, electronic commerce applications that have been compromised with intent of committing fraud need not display Byzantine fault; because, for a fraud attempt to be adequately profitable either it has to repeated many times, or it has to target high valued targets. Therefore, in various instances, this behavior may be sufficiently deterministic to be detected by macro-level metrics.

This paper describes methods to detect systems with fraudulent electronic commerce applications using certain macro-level metrics. Systems compute these metrics based on their message communications and report them to a trusted system. The trusted system analyzes the metrics' values and detects compromised systems.

In various instances, application nodes may be compromised in three ways: *structural*, *temporal*, and *data*. A structurally compromised node may, for example, withhold information from other systems, or fail to provide information about itself. A temporally compromised node may, for example, mislead about timelines; and a data compromised node may, for example, mislead about its data.

In various instances, data compromised nodes may be detected using existing data security methods. This paper introduces metrics for detecting structural and temporal compromises.

For example, encryption may be used to safeguard data from unauthorized modifications. Further, for non-Byzantine faults, tracer bullet approaches may be used to detect compromised nodes.

MESSAGE COUNTER

In various instances, an application node may be considered as a black box; that for a given input messages, M_i, message dispatches a set of output messages, M_{o1}, M_{o2}, etc.

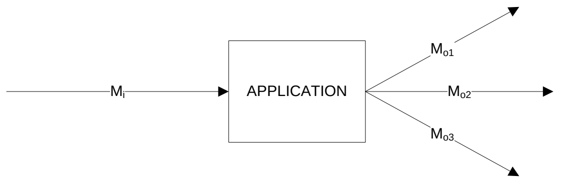

Fig. 1: Application Black Box

Message Counter *MC* for an application node is defined as:

$$MC = f(MC_c, M_i, M_{o1}, M_{o2}, M_{o3}, \ldots)$$

Where,

MC_c is the current value of *MC*.

Since electronic commerce processes are deterministic, thus for a given operation, a node's *MC* may be deterministic. Therefore, for a given operation, a trusted node may be able to predict the *MC* for untrusted systems. Untrusted systems compute their *MC* and report it to the trusted system; by analyzing the difference between predicted and received values for nodes' *MC* the trusted node may detect structurally compromised nodes.

In any distributed system, it is common for messages to be dropped, retried and delayed. Therefore under normal circumstances, there may be occasional differences between predicted and actual value of a node's *MC*. However, if the node is compromised and its behavior does not reflect Byzantine failure, then the difference between predicted and actual value of its *MC* may occur at a frequency higher than normal. Even, if a node has not been compromised consistent difference between its predicted and actual message counter value may be indicative of other issues such as network connectivity, data corruption, and/or inconsistencies in software versions and/or message formats. Therefore, in various instances, message counter may capture *MC* macro-level correctness for nodes in a distributed system.

LAMPORT LOGICAL CLOCKS

Lamport logical clocks are used in distributed systems to order messages. In various instances, each application node may maintain a Lamport clock, and it may use it to timestamp its outbound messages. These timestamps may be used to infer causality between messages. For example, using Lamport clock the outbound messages M_{o1}, M_{o2}, M_{o3}, etc. will have a timestamp greater than inbound message M_i. Moreover, a node may reject messages with timestamps less than its current Lamport clock value.

Again as electronic commerce are deterministic; for a given operation, a node's resultant Lamport clock value may be deterministic. Thus a trusted node may predict a node's Lamport clock value for a given operation; consistent difference between predicted and reported value may be an indicator of temporally compromised nodes.

COMPROMISED NODE DETECTION

The following figure shows a system based on these metrics.

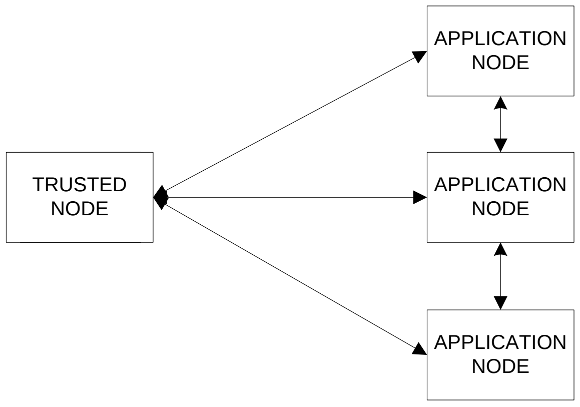

Fig. 2 Compromised Node Detection.

Application nodes may compute their message counter and update their Lamport clocks for each operation. Correspondingly, the trusted node may also predict these values based on operation's characteristics.

Further, application nodes may report their metrics values periodically, after each operation or before the operation is committed. If the predicted and reported values differ, trusted node may analyze this difference to detect compromised nodes. Such nodes may be dropped from future operations, or the process may be suspended.

As there may be communication between application nodes, a node that reports false metric values may still be detected unless all the nodes in the system cooperatively report false values.

Since, these metrics may be maintained in memory; there is minimal I/O overheard. Further, these metrics are simple enough to be used in devices with limited resources.

Additionally, this approach is more scalable than using log files; as the data generated is minimal. Assuming that both message counter and Lamport clocks are 8 byte doubles, for a million nodes this size of dataset is 16 MB. This is miniscule compared to data that would be generated using log files.

IMPLEMENTATION

An implementation of this system is currently underway for building an ecommerce system using limited resource systems.

Empirical data about the effectiveness of this metrics will be presented at the conference.

REFERENCES

1. Distributed Systems: Principals and Paradigms (2nd Edition), Andrew S. Tanenbaum and Maarten van Steen, Prentice Hall, October 2006.
2. The Byzantine Generals Problem, Leslie Lamport, Robert Shostak, and Marshall Pease, ACM Transactions on Programming Languages and Systems, 1982.
3. Tracking Service Availability in Long Running Business Activities, Werner Vogels, First International Conference on Service Oriented Computing, 2003.
4. Mobile Agent Programming in Ajanta, Anand R. Tripathi, Neeran M. Karnik, Manish K. Vora, Tanvir Ahmed, and Ram D. Singh, Proceedings of the 19th International Conference on Distributed Computing Systems, 1999.
5. PeerReview: Practical Accountability for Distributed Systems, Andreas Haeberlen, Petr Kuznetsov, and Peter Druschel, Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP '07), Stevenson, WA, October 2007.