2-1-2003

Applying Byzantine Agreement Protocols to the Intrusion Detection Problem in Distributed Systems

Fernando C. Colon Osorio
Worcester Polytechnic Institute

Xiaoning Wang
Worcester Polytechnic Institute

Applying Byzantine Agreement Protocols to the Intrusion Detection Problem in distributed systems

By

Fernando C. Colon Osorio
and Xiaoning Wang

Worcester Polytechnic Institute
Computer Science Department
100 Institute Road, Worcester, MA 01609

Abstract

"Fixed fortifications are monuments to the stupidity of man"

General George Patton, Jr.

Solutions to the Byzantine General Problem are applied to the design of an Intrusion Detection & Countermeasure Systems, called SAFE, being developed at the Worcester Polytechnic Institute System Security Research Laboratory (WSSRL). As described in this paper, Byzantine Agreement Protocols (BAP) arrived at a consensus on (identify) which nodes have been compromised, through the use of a series of synchronized, secure rounds of message exchanges. Having arrived at such consensus, offending or compromised nodes are isolated and countermeasure actions initiated by the system. Specifically, we consider in this manuscript the necessary and sufficient conditions for the application of Byzantine Agreement Protocols to the intrusion detection problem. In addition, the set of necessary assumptions needed for the BAP protocol to operate correctly, such as the need for a secure communication channel, are presented

1. Introduction

In recent years, there has been an exponential increase, as reported by CERT, see Figure 1, in security threats or "attacks" by both individuals attempting to use a system without authorization (i.e., crackers) and those who have legitimate access to the system but are abusing their privileges (i.e., the insider threat"). The nature and cost of these threats has contributed to the increased focus, research, and development in both academia and industry in the design and implementation of systems that are resistant to "hackers /crackers" attack. More specifically, dating back to the early nineties the study, research and development, and the implementation of Intrusion Detection & Countermeasure Systems (***IDCS)***, has become an essential element of the design of modern computer systems and applications, see [1], [2], [3], [4].

Unfortunately, current Research in the area of "Systems Security" in general, and in the design and implementation of Intrusion Detection & Countermeasure Systems in particular, has shown to have some significant limitations. For a complete assessment of the current state of the art of such systems, see Allen [2], Jansen [4]. Modern approaches to deal with such limitations have resulted in the design of distributed systems (a.k.a, network based systems), Spafford [1], Balasubramaniyan [5], where autonomous local agents collect and analyze information but cooperate with other agents in the system to arrive at decisions on the potential set of intrusions. All such systems suffer from several limitations as describe in [4]. Amongst the most important limitation of current approaches is the inherent vulnerability of the agents to attack, rendering the system designed to protect **"the System"** inoperable.

In recent months, at Worcester Polytechnic System Security Research Laboratory ***(WSSRL),*** research into the design of *an* Intrusion Detection and Countermeasure System, name ***SAFE,*** that can effectively addressed this limitation, has taken shape. The key to such a system is a distributed module called the distributed Trust Manager or ***TTM.*** The ***TTM*** is an independent entity serving three major system functions. These are:

- It knows which logical nodes in the system can be trusted – this is accomplished through a trust relationship matrix;
- It uses a Byzantine Agreement Protocol to identify and isolate nodes that have been compromised;
- It relays a "Last Gasp Message" to other nodes in the system; and
- Prevents the trust system from being partition (Quorum formation).

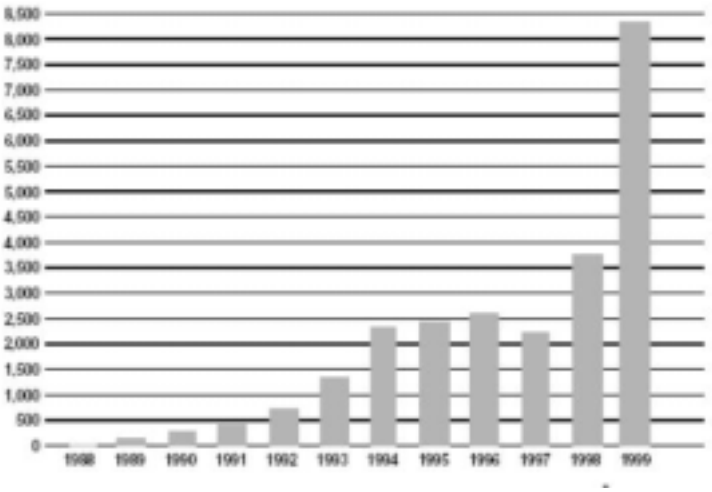

Figure 1 – Growth in Number of Incidents Handle by CERT/CC®

In ***SAFE***, the primary security mechanism used (at a system level[1]) is the creation, updating, and maintenance of a trust relationship matrix. This matrix, which is manage by the Trust Manager, contains up to date information on the trust relationship betweens all the nodes in the system. In ***SAFE*** both "weak" and "strong" trusted relationships exist among the nodes. The concept of a "weak/ strong" relationship, as used here, refers to the level of access that one node, say node n_a has established with another node, let's say node n_b.

A ***weak relationship*** will be, for example, the situation where users in one system can ftp to users in another system only. At the same time, a ***strong relationship*** will refer to a relationship where a node has complete access to all the privileged functions of the other node. Trust relationships are not symmetrical. In Figure 2, shown below, nodes a and node e have the same level of trust with each other, while node a is trusted by node b (node a can access privileged functions in node b) and not vice-versa. The arrow in the diagram shows the direction of trust.

A trust function $T_{ij}(t)$ for $i \neq j$, exist between two nodes, and as mentioned earlier, it is not necessarily symmetrical. T_{ij} is a function of time. In addition, the lack of trust between two nodes will be denoted as having a trust relationship of zero value, Tij $(t) = 0$. In this example, **Node a** is the source of the intruder attack, while **Node h** is the target of the attack. However, because there is not a direct trust relationship between Node a and Node h, the intruder is force to a set of attempted intrusions into nodes e and f, before attempting to compromised h. Due to the topology of the trust relationships (in Figure 2, the topological description is a logical one and not physical) between nodes, compromising any node other than nodes b or f will not allow the intruder to compromise the target node h. This topological constraint amongst nodes in a network has a significant advantage over other approaches. That is, it allows the designer of the *IDCS System* to create multiple logical layers of defense against intruders, in effect, creating time to detect potential intrusions and dwarfed them. Further, the concept of a trust relationship can be extended to include trust relationships with nodes that do not belong to the immediate system under protection. For example, a system like SAFE could pro-actively engage other nodes outside the local network to determine how trustworthy they are, and then, create a periphery trust matrix to manage the interactions with such nodes.

An example, of using a topological constraint to manage and impede intrusions will be as follows. Let's say that nodes b and e suspect an intrusion by using traditional audit methods. Then, nodes b and e can invoke a state change on their trust relationships with other nodes in such a way that,

Equation 1:

$T_{aj}(t) = 0$ for all $j \neq a$ and $t > t$ of intrusion; and

$T_{ej}(t) = 0$ for all $j \neq e$ and $t > t$ of intrusion.

[1] SAFE implements different layers of security protection and containment at different levels in the system. For example, at the local layer an autonomous agent, called SHM, is responsible for detecting Intrusions locally. However, information on the state of the system is share with other nodes through the use of TTM protocols.

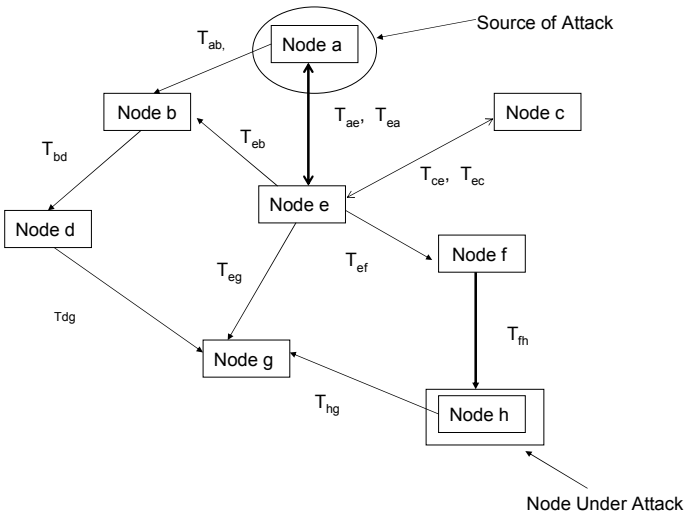

Figure 2: A Simplified Intrusion Detection Model

Once, the problem of intrusion detection is formulated in this context, then well know solutions to the Byzantine Generals Problem are readily available. In the remaining of this paper, we will present how one such solution, a Byzantine Agreement Protocol (BAP), can be used in the design of the Trust Manager ***(TTM).***

1.1 The Byzantine Generals Problem

The Byzantine Agreement Protocol ***(BAP)***, which aims at establishing a fault-tolerant agreement when one or more of the nodes in a system have been compromised or failed, received considerably attention in the literature during the late 80' and early 90's. The primary application then was the design Fault Tolerant Systems. Recently, and mainly because of the increased importance of the "security problem", interest on its usage has increased. Lamport, et al. described the Byzantine General Problem in [7, 8]. Specifically, the Byzantine General problem formulation is as follows.

Imagine that several divisions of a Byzantine army are camped outside an enemy city, each division commanded by its own general. The generals can communicate with one another only by messenger. After observing the enemy, they must decide upon a common plan of action. However, some of the generals may be traitors, trying to prevent the loyal generals from reaching agreement. The generals must have an algorithm to guarantee that:

A. All loyal generals decide upon the same plan of action. The loyal generals will all do what the algorithm says they should, but the traitors may do anything they wish. The algorithm must guarantee condition A regardless of what the traitors do. The loyal generals should not only reach agreement, but should agree upon a reasonable plan.

We therefore also want to insure that

B. A small number of traitors cannot cause the loyal generals to adopt a bad plan.

Further, in [7], *the* ***Byzantine Generals Problem*** was described as the design of an algorithm such that when the commanding general sends an order to his (n – 1) lieutenant generals, then the algorithm guarantees that

IC1. ALL LOYAL LIEUTENANTS OBEY THE SAME ORDER.

IC2. IF THE COMMANDING GENERAL IS LOYAL, THEN EVERY LOYAL LIEUTENANT OBEYS THE ORDER HE SENDS.

The *BAP* algorithm is essentially a distributed algorithm designed to achieve consensus. Borrowing from Lampson [9], "several processes achieve consensus if they all agree on some allowed value called the 'outcome' (if they could agree on any value the solution would be trivial: always agree on 0). Thus the interface to consensus has two actions: allow a value, and read the outcome. A consensus algorithm terminates when all non-faulty (un-compromised) processes know the outcome".

BAP as a consensus algorithm can be applied to two critical problems, which are of great importance in the design of secure systems. These are:

Membership, where a group of processes cooperating to provide a highly available service need to agree on which processes are currently functioning as members of the group.

Every time a process fails (is compromised) or starts working[2] again a new consensus must be generated.

Electing a leader amongst a group of processes prior to an exact determination on the number of processes that belong to the group.

Within this context, if we substitute ***Generals*** for nodes in distributed systems, and **consensus** for the need to agree on which processes (agents, modules, etc.) are ***safe/sane*** (i.e., have not been compromised). Then, the problem of identifying an isolating compromised nodes can be easily described as follows:

Imagine in **SAFE** several nodes, which cooperate with each other to detect intrusions. Each node runs an autonomous agent, the TTM, which continuously sends messages to other nodes. The message that is sends has two possible values. These are:

Message A1: *"keep sane" or "0";* and

Message A2: *"I am potentially compromised" or "1"*. Enclosed is the "*signature that implies a potential intrusion".*

In this context, the nodes that can be trusted should be able to determine which nodes are compromised, if there's any, and arrive at a consensus. A small number of nodes that have been compromised should not be able cause the other nodes to adopt a wrong or even malevolent message. The Byzantine Agreement Protocol (*BAP*), by which we can detect intrusion and minimize the success of attacks, fits this context perfectly.

Note that *BAP* doesn't always hold. There is a threshold *t*. That is, if more than *t* nodes are compromised, the *BAP* will also fail. The resiliency of the algorithms to the number of nodes compromised depends heavily on the communication mechanism used to exchange messages, and its characterize by the value of *t.* Lamport et al. [7], suggested two distinct algorithms depending on the mechanism used for communication. The first one, which he called an *"Oral Message Algorithm"*, makes use of unauthenticated sequence of oral message exchanges. The second algorithm, known as *"Signed Message Algorithm*", depends on the assumption that messages are signed, and that the identity of the sender can be guaranteed. In the case of the Oral Message Algorithm it can be shown that consensus can be achieved when there are at most $\lfloor (m-1)/3 \rfloor$ nodes that have been compromised among a total number of *m* nodes. Similarly, he demonstrates that the Signed Message Algorithm can effectively achieve consensus if at most *m-2* nodes have been compromised amongst a total of *m* nodes.

In order to improve the intrusion-resiliency of ***SAFE***, as well as to make the implementation of the Byzantine Agreement protocol simpler in our system, we will assume the presence of a communication channel, which is both reliable and secure. Hence, the algorithm implemented in our system, to achieve consensus, will be the so-called SM (*m-2*) algorithm. Here, each node will implement the message exchange mechanism described by Hoffstein [10]. That is, in order to exchange messages:

(1) The commander signs and sends its message to every node N_i it can reach directly.

(2) For each node N_i:

(a) Node, N_i, maintains a running list of all the messages it has received;
(b) Node, N_i then, signs (authenticates) all the messages that node it has received, and then sends a copy with its signature to all other nodes that:

i. are directly connected (one hop away) to node N_i., and

ii. Whose messages he has yet to received.

[2] In SAFE, a process (node) starts working again after the local control module SHM has: (1) isolated the process from the network; (2) launch the prescribed set of countermeasures; and (3) certified that the node is "sane" again. When such actions are completed, SHM makes a request to the local Trust Manager module to re-integrate.

(c) The node will repeat step (b) until node N_i does not received any additional signed messages.

(3) N_i then will arrive at a decision on the course of action based on all the signed messages at hand.

It is also well noting that a compromised node will not necessarily follow the above ***Signed Message Algorithm.*** For that matter, a set of compromised nodes may collude with each other in an attempt to mislead or compromise all other sane nodes. Collusion may take any of the following forms:

(1) Falsify signatures, i.e., use each others signatures as part of the authentication process;
(2) Forge messages; and
(3) Send incorrect values or not send any values at all.

However, as long as at least (*m-2*) trusted nodes could exchange messages to arrive at consensus, the Byzantine Agreement Protocol (*BAP*) holds. This is true if and only if we can guarantee the following:

A1. Every message that is sent is delivered correctly.
A2. The receiver of a message knows who sent it.
A3. (a) A sane node's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.
(b) Anyone can verify the authenticity of a general's signature.
A4. The absence of a message can be detected.

2.0 Building a Trusted System - Overview

By "Trusted System", we mean a system composed of a set of autonomous agents that include at least a TTM and a SHM (Secure Host Manager) per node. A Secure Host Manager (SHM) is an autonomous agent residing in each node in the system that performs the following set of functions:

- Low level (or host based) intrusion detection using a set of traditional misuse or anomalous detection models;
- Provides intrusion packets to the Trust Manager for consensus forming;
- Incorporates learning algorithms for the low level intrusion detection functions; and
- It is responsible for node isolation, countermeasure issuance, and recovery.

The relationship between the TTM and SHM is shown here in Figure 3. In this system EGO's are low-level Intrusion detectors (also known as probes) which can be turned on/off to collect information and minimize low level processing A complete discussion of the EGO's roles and functions is not needed at this time.
.

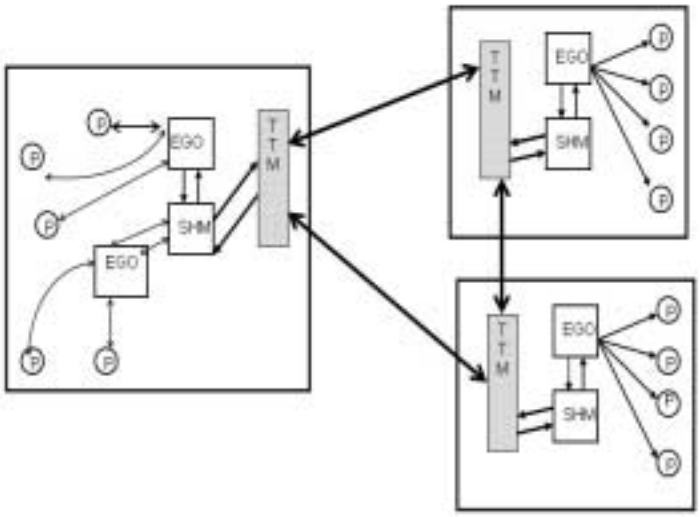

Figure 3 – SAFE Architecture

For the purpose of this discussion, consider a system ***S***, in which all nodes are sane and trusted a time t_0. At some pre-specified later time, to + Δ, every Trust Manager in the system exchanges a set of signed messages, following the above protocol to determine if the nodes in the system are still to be trusted. After receiving all the messages broadcasted in the system, a consensus must be reached to determine if the system and its nodes are still trustworthy. Here, we assume that every node is either sane, which means the node is believable, or insane, that is a node has been compromised. As pointed out earlier, SAFE uses a Signed Message algorithm, which can

detect at most t insane nodes among m ones in the network. Once, a node or set of nodes are deemed to be compromised, then, the Trust Matrix of every sane node is updated, setting Tij = 0, for all compromised nodes, and effectively isolates them. Finally, using a one-way channel, the System Trust Manager will force the Trust Manager in each compromised nodes to a root state. While in its root state, the TTM will no accept any further messages, and will issue commands to SHM in order to initiate countermeasures. A compromised node will be allowed to join the trusted set of nodes again, if and only if, a known secure state is reached by the local TTM and SHM.

2.0 Building Trusted System – Detailed Discussion

In order to simplify our discussion, we will assume that: (1) during the period of time while the system is attempting to arrive at a consensus, a sane node cannot be compromised[3], and (2) the number of nodes in the system does not change. In addition, any non-trusted nodes that want to join a network community (or trusted system), will use a Quorum Algorithm that we define as follows.:

Consider the following case. There is at least one node that does not belong to a trusted system. Such node upon boot,

- Will verify the absence of a trusted system by "broadcasting" on all links a "*join trusted system message"*. If a trusted system already exists, then a consensus process is initiated amongst all nodes in the trusted system.

- Otherwise, the node assumes that a "trusted system" is not present; it will declare itself the commanding general, it will build a community of one, and then will wait for other nodes to join in.

As described earlier, each node N_i maintains a one-dimensional Trust Matrix locally in the form of $[t_{i1}, t_{i2}……t_{in}]$, where t_{ij} is a measurement of the degree to which the node N_i can believe that another node N_j is sane. A higher value for t_{ij} indicates N_i has greater confidence that N_j is sane. A value of 0 means N_i believes firmly that N_j is insane. For simplicity, we set t_{ij} to be either 0 or1, that's, t_{ij} is Boolean. The system can, then, be represented by an n X n Trust Matrix that represents the entire system trust relationship.

We also set a *Tolerant Parameter t*, which is the maximum number of compromised nodes a secure system can tolerate. If more than *t* nodes in the system are compromised, the system cannot be trusted any longer because its nodes have lost their capability to detect intrusion. Our system uses Signed Message Algorithm and hence $t = n-2$. If the number of compromised nodes exceeds *t*, then the whole network will be compromised. In this case,

(1) TTM at each node will issue a command to SHM in such a way that the node is taken offline, and it disconnect itself from the whole network.
(2) TTM will set all the elements in the local Trust Matrix as zero, namely, $t_{ij} = 0$ for each node j, which means any connection from this node to others cannot be trusted.
(3) SHM will then repair the compromised node locally.
(4) After step (3) has been accomplished successfully, SHM will communicate with TTM and request that the Quorum Algorithm is run to either join the existing community or create a new one.
(5) Based on the trust relationship, TTM will assign new values to t_{ij}.

After a node broadcasts a certain message to other nodes, in order for consensus to yield a trusted system, the following two conditions must be satisfied:

1. All sane nodes must use the same message as input, so they can get the same output or we say they can reach consensus.
2. If the node that sends the message is sane, then all sane nodes use the message it provides as input, so they produce the correct output.

These are just our interactive consistency conditions IC1 and IC2, where the

[3] This constraint can be relaxed, and SAFE is still capable of achieving consensus. We will explore such algorithms as part of our ongoing research.

"commander" is the node sending the message, the "lieutenants" are the nodes receiving the message.

In order for our system to work properly, a Signed Message SM (m) algorithm must be implemented. Implementing such a system requires adherence to the following set of assumptions, A1 through A4.

A1. Every message that is sent is delivered correctly.

A2. The receiver of a message knows who sent it.

A3. (a) A sane node's signature cannot be forged, and any alteration of the contents of his signed messages can be detected.

(b) Anyone can verify the authenticity of a general's signature.

A4. The absence of a message can be detected.

Assumption A2 states that a node can determine the identity of the owner for any message that it receives. What is actually required is that an insane node should not be able to impersonate a sane one. Note that, under our set of conditions, A2 is encompassed by A3, and therefore not needed.

We now consider the assumption A1, A3 and A4 in order, and show they are guaranteed in SAFE.

A1. Here we assume the existence of a secure, fast physical communication channel. This is a well know research problem, see [10], [11], [12]. However, in a real system, the channel may fail. But in this case, a failed communication channel, from our perspective, just has the same effect as simply isolating it from the network and the countermeasure is building a correct, trusted network connection again. A3 and A4 still hold.

A3. Property A3 (a) can not be guaranteed with 100% certainty since any message is just a binary data item, and theoretically an insane node could generate any data item, as it likes. However, we can reduce the probability of this occurrence to any comfort level that we so desire. Using classical public key cryptography solutions can insure assumption A3. That is,

(1) The sending node N_i will

(a) Bind the message, starting time and its name together and encrypt this message block B_i by its private key.

(b) Bind B_i with N_i's name, a random data chunk D_i, and a Time-stamp T_i into a new block by N_j's public key. (T_i will be used for synchronization in the last assumption.)

(c) Send the final message, which is actually the tuple $\{B_i:N_i:D_i:T_i,\}$ to the target node N_j.

(2) Only N_j is able to decrypt the message block with its own private key. Then, after knowing whom the originator of the message is, N_j will decrypt the message B_i by N_i's public key, read the content and verify the authenticity of N_i.

(3) If N_j is the destination, this branch of sending message will stop; Otherwise, it will:

(a) Bind B_i and its name together and encrypt this message block B_j by its private key.

(b) Bind B_j with N_i's name, N_j's name, and a random data chunk D_j, and a Time-stamp T_j, into a new block by N_k's public key.

(c) Send the final message block, which is actually $\{B_j:N_i:N_j:D_j:T_j,\}$ to the target node N_k.

(d) N_k will repeat (2) and (3) until stop. The only difference is that a multi-layer encryptions/decryptions will be needed, which are very expensive.

Any decryption failure in step (2) and (3) will force the receiving node to assume that the sender of the message is insane. On the other hand, if an insane node fails to sign and encrypt its identity, then, the lack of a signed message will be used to declare such node insane at final decision consensus-making time. At that time, each sane node will look thoroughly at all the available signatures. We will discuss this latter case, and provide a solution by the creation of a time bound in Assumption 4.

We also iterate here that there are two copies of the nodes' names in the message block. The copy placed at outer layer is used to tell the receiving node which public key it should

choose for decryption and the inner copy, together with private keys, virtually forms a digital signature.
The random data chunk D used in (1) and (3) is also critical, since the message B is predictable to some extent. For example in step (2), suppose that an insane node N_t, after having received a message B_i from node N_i, attempts to forge it, and sends it out as $\{B_j:N_i:N_j:D_j:T_t\}$, in effect a forged copy. The receiving node cannot decide that the time-stamp T_t is forged and will use it to synchronize the whole distributed system as explained in Assumption 4, which will be disastrous. But with this random data item, it will be extremely difficult for the insane node to do so undetectably. We can even rearrange the order of $\{B_j:N_i:N_j:D_j:T_j\}$ to further reduce the probability of such an event, as long as the receiving node can understand it.

Note, that despite the fact that current cryptosystem cannot insure 100% secrecy, we can reduce the probability for an insane node to succeed in impersonating sane nodes to such a small probability that we can in effect ignore this problem. In the paragraphs that follow, we present one such solution, and describe its error characteristics.

Selecting a cryptosystem should be based on at least the following two factors:

1) High speed for both encryption and decryption. This is of great importance especially when considering the fact that:

   a) The execution of Byzantine Agreement Protocol can introduce a substantial overhead due to the number of messages required to arrive at consensus;

   b) The system requires multi-layer cryptography.

2) Security. This includes both a correct decryption process to insure an extremely low probability of successful malicious decryption.

Our approach is to adopt NTRU, a ring-based public key cryptosystem that features reasonably short, easily created keys, high speed, and low memory requirements. [10, 12] The novelty of NTRU lies in the use of the mixture of polynomial algebra and reduction modulo two numbers *p* and *q* where *gcd(p, q) = 1* and *q* is significantly larger than *p*. So, both the public key and the private key are polynomials or vectors.

The biggest advantage that NTRU over other PKCS is its considerably fast speed in both encryption and decryption with a much shorter key length. In [10, 12], the most time-consuming elements of an NTRU implementation are the computations of polynomials in the vector ring modulo *p* and *q*, which can be reduced to *O(qNlog(N))*, by using Fast Fourier Transforms.

Based on analysis in [4, 6], NTRU signature scheme has about the same security level as RSA and other signature schemes. Moreover, extensive experiments show the decryption failure of NTRU can be controlled to a low probability less than $5*10^{-5}$. So, the receiving nodes can decrypt the message correctly and authenticate the sender.

A4. This assumption requires that the absence of a message can be detected and the receiving nodes won't be left waiting infinitely. The absence of a message can only be detected by its failure to arrive at the destination within some fixed length of time. Before we start our system, we can manually specify this maximum time to be ΔT_{max}, according to the network speed for sending message, the diameter [7] of the system and the time used for generating a message. Since for distributed systems, no two nodes can run at exactly the same rate [7, 13] and the nodes lack a common notion of global time, we must use an algorithm to synchronize the clocks in such a distributed system.

Leslie Lamport implemented "logical clocks" based on a mathematically rigorous "happen before" relation in [7]. The "Physical Clocks" model he proposed in [13] can solve the synchronization problem for our system. This model has three assumptions:

(1) Every clock C_i in the distributed system must run continuously, rather than in discrete ticks, at approximately the speed of 1. That's,

$$\forall i, \exists k \text{ , so that } \left| \frac{dCi(t)}{dt} - 1 \right| < k$$

where *k<<1*.

(2) $\exists\varepsilon$, a sufficiently small constant, so that

$$|Ci(t) - Cj(t)| < \varepsilon \text{ for all } i,\ j.$$

(3) The clocks satisfy the ordinary Clock Condition in [7].

We also need to set a parameter ΔT_{min}, which denote the minimum delay for a message to go from one node to the next node. Theoretically,

$$\Delta T_{min} = \frac{\text{the shortest physical distance between two nodes}}{\text{the speed of light}}$$

However, the actual value of this parameter should be much larger and we can assign a proper value to it. Then, when a node N_i receives a message with time-stamp T_m, it updates its clock by the formula:

$$C_i(t) = \max(C_i(t), T_m + \Delta T_{min}) \text{ where } C_i(t)$$

denotes the reading of the clock C_i at physical time *t* at N_i.

A theorem in [13] can help us calculate how much time the clocks will take to be approximately synchronized after the system is first started. After that, we can run our Byzantine Agreement Protocol (*BAP*). If the receiver has not received the message by the time $t_i + \Delta T_{max}$ where t_i is the starting time for i^{th} execution of BGA, it concludes that the message isn't sent at all. Any message that arrives at a time later than $t_i + \Delta T_{max}$ will be viewed as "unsent" and thus ignored by the BAP protocol. Hence, a node that does not send a message within the time constraint is considered insane. Messages received before this maximum time will be used to reach consensus. After consensus is reached, then, all nodes whose messages have not been received will be declared insane. The countermeasure and repair process will be initiated as described in Section 2.0.

3.0 Conclusion and Future Work

In this manuscript, solutions to the Byzantine Generals Problem, and associated Agreement Protocols to achieve consensus, have been applied to the problem of designing a trusted system. The Agreement protocols described are encompassed as part of the functionality of an autonomous and distributed agent called ***The Trust Manager*** or TTM. The TTM is responsible, among other things, for identifying and isolating compromised nodes in the system.

The work, although theoretical in nature, is being use today in the design of an Intrusion Detection and Countermeasure System called ***SAFE*** currently under development at the WPI System Security Research Laboratory, ***WSSRL***. The current work needs to be extended in several areas. These are:

1. Performance evaluation of the algorithms and the TTM under real-time intrusion workloads;
2. Extension of the TTM concepts to other areas in the system, such as lower level intrusion detectors; and
3. Comparison of the current approach to other similar systems that depend on autonomous or mobile agents for their operation.

Bibliography:

[1] Spafford, Eugene H., Diego Zamboni, "Intrusion Detection Using Autonomous agents", Computer Networks 34 (2000), pp. 547-570.

[2] Allen, Julia et.al., *"State of the Practice of Intrusion Detection Technologies,* Technical Report CMU/SEI-99-TR-028 ESC-99-102.

[3] Vorwoerd, Theuns & Ray Hunt, "Intrusion Detection Technicques and Approaches", Computer Communications, 25 (2002) pp. 1356-1365.

[4] Jansen, Wayne A., "Intrusion detection with mobile agents", Computer

Communications, 25 (2002) pp. 1392-1401.

[5] Balasubramaniyan, J.S., Jose Omar Garcia-Fernandez, David Isacoff, Eugene Spafford, and Diego Zamboni, COAST Laboratory Technical Report 98/05.

[6] H.Cohen, *A course in Computational algebraic Number Theory*, Springer-Verlag, Berlin, 1993.

[7] L. Lamport, R. Shostak, and M. Pease, *The Byzantine Generals Problem* ACM Transactions on Programming Languages and Systems, July 1982, pages 382-401

[8] L. Lamport, R. Shostak, and M. Pease , *The Byzantine Generals Problem* ***http://www.cs.wisc.edu/~sschang/OS-Qual/reliability/byzantine.htm***

[9] Lampson, Butler, "How to Build a Highly Available System Using Consensus"
10th International Workshop on Distributed Algorithms (WDAG 96)

[10] J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A ring-based public key cryptosystem.*

[11] J. Hoffstein, J. Pipher, J.H. Silverman, *NTRU: A new high speed public key cryptosystem*, in Algorithmic Number Theory (ANTS Ш), Portland, OR, June 1998, Lecture Notes in Computer Science 1423 (J.P. Buhler, ed.), Springer-Verlag, Berlin, 1998, 267-288.

[12] Adam Avilez, Adam Wolfe, Lauren Kennell, Eric Purdy, *NTRU Public-Key Cryptosystem*.

[13] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558-565, July 1978.