

Bad Member Ejection in Shamon

Laxman Vembar, Archana Viswanath and Nazneen Irani
Dept. of Computer Science & Engineering
Pennsylvania State University, State College, PA -16801
{vembar, aviswana, irani}@cse.psu.edu

Abstract

Shamon Architecture is a security policy enforcement mechanism for distributed computation. It establishes trust among members collaborating together ("coalition") through code attestation and the enforcement of a mutually agreed Mandatory Access Control (MAC) Policy. However, Shamon does not address the issue of maintaining this distributed trust when a member in the coalition no longer complies with the policy (i.e. when a member goes "bad"). Identification of this bad member by either a single or a subset of members of the same coalition, and reaching a common consensus on the state of that member is important for maintaining trust in the coalition. This paper discusses how to attain consistency in belief of all members in the face of the above scenario. We propose four models which can be used to achieve this consistent state and analyze the advantages/drawbacks of each of them.

1 Introduction

Shamon is a mechanism of providing security in a distributed system by using code attestation and enforcing a common Mandatory Access Control(MAC) policy. A MAC policy enforces mandatory rules for information flow unlike Discretionary Access Control(DAC) (eg Unix file system) where each user can set permissions for information he owns. We assume there are several client systems which are involved in performing a particular distributed task and term the collection of these systems as a "coalition". The idea here is that we do not need to have complete trust in the client systems in their entirety but only the part of the system which is involved in the distributed application.

Figure 1 shows three clients running several applications but only a subset of these application (those marked in red) are involved in our Coalition. Thus it is sufficient if we can verify the integrity of this subset of

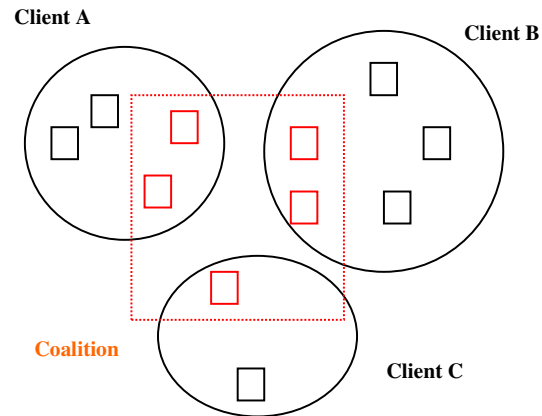


Figure 1: A coalition in Shamon

applications to establish trust in the coalition *IF* there is no way that the untrusted applications(the ones marked in black) can communicate or affect the working of our trusted applications. This isolation is achieved in Shamon through the use of MAC policies which are enforced using a reference monitor.

Whenever a new client wants to join an existing coalition, his MAC policies are verified against the coalition's and checked to see if the required isolation can be provided. If so the new member is accepted into the system. The integrity of the trusted applications themselves is established through remote attestation of code and static data(configuration files etc). In Remote Attestation, the values of the code/data are hashed from the bottom up. The Trust is established from the Trusted Computing Base(TCB) consisting of the Trusted Platform Module(TPM). This piece of hardware hashes the software stack including the bios, bootloader, operating system and trusted applications. The hash generated is signed using the private key stored in the TPM hardware to verify its authenticity.

Our focus here is on removing a member from a Shamon coalition once it has been established that the member no longer runs trusted code. The main issues here are in notifying all members of the coalition about

the state of a member who has gone bad and ensuring the cost effective verification of these claims so as to enable the establishment of a common belief among members of the coalition. We propose four models in this paper to satisfy these requirements. Section 2 of the paper looks at the related work in this area and why this particular issue has not been addressed before. In section 3 we introduce our models explaining the advantages and drawbacks of each. Section 4 details some analytical measurements on these models and section 5 includes our conclusion and future work in this area.

2 Related Work

Several mechanisms have been proposed to address the problem of trust in Distributed Systems. The BIND system[5] for example attempts to establish trust by binding the data generated with the code that was used to create the data. The approach here is to use this at a finer granularity level to allow the system to scale easily. BIND also uses the concept of transitive trust to establish the integrity of the input data to a piece of trusted code. BIND however does not talk about how primitive data entering the system can be verified to establish its integrity. This is a significant issue since systems which do not rely on low integrity inputs of any form are very rare and constraining. The concept of transitive trust here is important because it allows us to verify the integrity at any given stage in a single step rather than have to trace the entire path of execution in the distributed system. There are other systems such as Terra[3] which base their trust on remote attestation to verify the integrity of the code running on a system.

The main issue with any system of code attestation is that the only solution is to halt when the integrity check fails. They do not talk about any mechanism of recovering from the problem and continuing execution.

There are some systems which proactively recover from errors using replication[6]. Here we have a secure coprocessor which checks the integrity of the code running at regular intervals and if the code signature does not match(the code has been compromised), it resets the code. These systems are also however vulnerable to physical attacks and may be running even though they are compromised during the timeframe between checks.

A lot of work has also been done specifically addressing the field of trust management in distributed/P2P systems[7] which use the concept of varying degrees of trust to thwart the effect of malicious users. However

such a system increases complexity in achieving a consistent state among all members.

The main problem here is that none of these systems address the problem of ejecting a node from a distributed system when they are found to be compromised. In the next section, we identify the requirements to achieve this consistency and propose models to achieve the same.

3 Trust models for member ejection

Our goal here is to establish consistency among members of a coalition on the noncompliant state of a member with regard to the commonly accepted policy of the coalition. Such a member is identified as bad and is ejected from the coalition. This implies that no trusted member of the coalition proceeds to communicate with a bad member once he has been identified.

Maintaining consistency of beliefs of all members of a coalition is important in Shamon Architecture. Having a mistrusted member in the coalition can lead to the compromise of the entire network through several ways like information leakage, excessive use of network resources, distribution of malicious data, etc. Therefore communicating the identity of such a bad member is necessary as this prevents the trusted members from communicating with the bad member thereby protecting them. Moreover, all other members of the coalition should agree on the state of the bad member.

Ejecting a bad member involves three stages:

- **Report:** In this stage, when a member in a group identifies someone as being bad, he propagates that information to other members in the group
- **Confirm:** On receiving the report message in the previous stage, each member now needs to validate the integrity of the message and confirm its authenticity
- **Eject:** Finally, every member in the group must remove the identified bad member from its list of trusted members so they do not communicate with the bad node.

We propose several approaches to achieve this consistency but all models in general include the above three stages.

I. Trivial Model

We first propose the ‘Trivial Model’ to perform ejection of a bad member from the Shamon coalition. This model, as the name suggests, is the simplest approach to report a bad member and eject him after confirming his state. The model can be described completely in three stages as follows:

1] Report: The identification of a bad member is reported to all members of the coalition through a broadcast. Informing other members about the state of a bad member is essential to maintain trust in the coalition and avoid good members from communicating with the bad member. This information is required for good members to eject the bad member from their list of trusted members.

2] Confirm: There is a possibility that the member reporting a bad member is actually bad. Hence the message reporting a bad member cannot be implicitly believed. Each member verifies this notion by remote attesting the claimed bad member. This confirmation enables the maintenance of trust in the system.

3] Eject: This is the final stage where all members who have confirmed the state of a bad member, remove him from their individual list of trusted members. No further communication is carried out with that member. In this way, by each member removing the bad member from their individual lists, a state of consistency on the belief of a member is attained.

Consistent state of trust is arrived at per individual through remote attestation. The entire system does not collectively unanimously arrive at a consistent decision about the bad member and then collectively eject the bad member from each of their list of good members. Byzantine Fault model would not work effectively to make a decision for ejection as there always exists a possibility of there being bad members in the coalition who have not been discovered but are contributing to the decision. A unanimous consistent state of the coalition is indirectly achieved, when all good members recognize and remove the “bad” member from their list.

The member identifying the bad member may be malicious, trying to cause Denial of Service of the claimed bad member through the large number of remote attestations performed by all the group members.

II. Lazy consistency model with remote attestation and Group decision

The Trivial model described earlier involved the remote attestation of a bad member by all members of the group. Remote attestation would involve the sending of

all the code, data and measurement list running on the bad member to every group member along with considerable overhead on each member of the group to verify the same. This approach not only increases network overhead but also computational overhead for every member.

Group Consistency requires every member of the group to have the same notion about the members of the group[11]. With respect to trust, group consistency involves every member of the group to have the same notion of the good and bad members in the group.

Release Consistency is one of the consistency models used in the domain of concurrent programming. This model is used in the implementation of Distributed Shared Memory[9]. It has two synchronization operations – release and acquire. A system is said to provide release consistency if all write operations to an object by a node are visible to all other nodes on the former releasing it and before the later acquires it. Eager release consistency is a coherence protocol implementing release consistency, where coherence actions are performed by all nodes on release operation by the former. In Lazy release consistency model, all coherence actions are delayed until after each node performs an acquire on the object.

The similar notion of Release Consistency is applied to our models to achieve group consistency on the state of a bad member. The Trivial model described above implements Eager Release Consistency where every group member on being communicated about a bad member tries to ascertain immediately through remote attestation the bad member’s state. Although consistency will be achieved through this model, it is an infeasible option due to high network and computational overhead. A second model has been proposed here based on the Lazy Release Consistency model

In the Lazy Release Consistency model, the network and computational overhead caused due to the Trivial model can be alleviated by attesting the bad member only before communicating with him. As all group members may not want to communicate with the bad member, the network and computational overhead is proportional to the number of members wanting to communicate with the bad member.

Here we achieve a *safe state* but not necessarily a consistent state as only when all members want to communicate with the “bad” member will consistency as a whole be achieved through individual attestations. As we use lazy consistency in this model, nodes wanting to communicate with the “bad” member would

be in a safe state owing to their decision made through their remote attestation.

The possibility of Denial of Service attacks of the claimed bad member is minimized due to fewer attestations happening only during individual node communication

III. Gossip Model

One of the main issues with the models proposed so far is that there is a considerable load placed on a member who identifies that someone is bad. Using broadcast allows us to minimize the number of messages required to inform all clients (because every other good client receives only one message) but requires that the initiating member send all messages through unicast to each host. This is because we cannot assume that the underlying network supports a form of multicast/broadcast to allow a single message to be sent to all good nodes.

Several gossip models have been proposed which take into account the presence of malicious users in the network [13,14,15,16,17]. Those that do allow malicious failures [16] usually assume the availability of unforgeable signatures. In general our model has to address the following:

- What information is exchanged using gossip messages?
- When and to whom are gossip messages forwarded by members?
- How do we verify the integrity of the gossip messages?
- How can we establish an order for the gossip messages?
- How does gossiping stop or when does a message become old?
- How do we ensure that everyone hears the gossip message?

We propose to use gossiping only to spread bad information. Thus we use gossiping to spread information only when a member has identified to be bad. We propose this approach because of the inherent difficulty in ensuring everyone has heard each gossip message and maintaining strict ordering among the messages. Using gossiping minimally in this way (since we presume that members going bad is a relatively rare event) limits this problem. We also propose some measures which will help mitigate the problem further.

Another crucial issue that needs to be addressed here is how we verify the integrity of the messages. There have been several approaches proposed to achieve this but most of them boil down to accepting a message only after we receive the same message from a certain threshold number of members in the group. However since we would like our system to be able to function correctly even when we have a significantly large number of bad members, we propose a different approach. We propose the formation of trusted groups such that members within a trusted group may implicitly believe each other. Every gossip message includes the value of the extended hash of the software stack stored in the PCR (Platform Configuration Register) of the TPM and we form the trusted groups of each member using the value stored in this hash. In particular each member of the coalition forms two groups for himself namely the Incoming trusted group and the Outgoing trusted group.

The members in the coalition form their trusted groups based on the following criteria:

- Each member of the coalition forms his trusted group through mutual remote attestation.
- Every member maintains an incoming and outgoing hash list in his database.
- The incoming hash list is populated with the most recent hash values of the members that have been attested.
- The outgoing hash list is the list of hash values of this member (the value in the PCR) that was sent to the other members in the during the most recent mutual attestation.

The idea behind using the PCR hash value in the gossip message is that this can be used to verify the integrity of the sender and thus the integrity of the message itself. The reason we are maintaining two groups here is because this relationship need not necessarily be symmetric. In particular, our incoming trusted group will consist of the list of members whose current hash value is the same as that they had during the last mutual attestation. This is the group of members who we can implicitly trust because there has been no change in their state. Similarly our outgoing trusted group is the list of members for whom the current value in our PCR is the same as that the hash value we had during the most recent mutual attestation. This is the group of members who we know will implicitly trust our messages and thus we ensure that all members in our outgoing trusted group are informed as soon as we identify a change of state (i.e. we have heard about it and have verified its veracity) for any member in the coalition.

Member ID	Local Hash (Our PCR)	Remote Hash (Member PCR)
Member1	2222	3333
Member2	2222	4444
Member3	1111	5555

Current PCR: 2222

Figure 2: Illustration of hash values in database

It is important to note that the incoming and outgoing trusted groups are not necessarily different tables stored in each member's database. The database could contain a list of members who we have attested along with the tuple of our value of the PCR and the remote member's value of the PCR during the most recent mutual attestation. Our outgoing group would dynamically change depending on our current value of the PCR (This can change when we load new code into a trusted subject). Also at no point can we actually have a list of members in our incoming trusted group. When we hear a message from a member, if the hash contained in the message is the same as that we have in our table for that member, we identify him as being implicitly trusted.

Some of the other issues include whether we forward the gossip message only after we have verified its validity[] or whether we send it across even though we may not necessarily believe in it at this stage[]. Here again our concept of trusted groups plays a great part. Since all members in our outgoing trusted group will implicitly believe our messages, we do not want to forward this information to them. Below is our proposed method for message communication using the gossip style protocol:

- During an outgoing message from a member, a subset of the members in the database for whom the outgoing hash value is the same as the value we currently have are selected.
- If the members believes in the message, it is sent to all the members in its outgoing trusted group.
- In order to avoid localization of gossip at every stage, few nodes from the coalition are selected at random and are sent this message.
- The random nodes that receive the message believe in the message only after remote attestation of the "bad" member.
- The random nodes do not include any of the members stored in the database of the sender that do not have the current hash(PCR) value of the sender. This is to avoid the nodes from reattesting the sender on finding out that the

old hash value maintained by them for the sender and the current sender's hash value do not match.

- This point has been added so as to enable the possibility of taking advantage of the localization of gossip messages. Through the localization of messages we expect that there could be a high probability for the non-hash matching trusted group of the sender to hear from some other member of their trusted group instead of performing a remote attestation.
- If the member does not have any basis for believing the message, it forwards it to some random members but does not send it to any of the members stored in the database. This can potentially cause problems in certain edge conditions which we will elaborate on in the limitations of the model.

In the sample database shown in figure 2, our outgoing trusted group would consist of members (Member1, Member2) and we would implicitly trust incoming messages from members Member1, Member2 and Member3 if the hash value in the messages are 3333, 4444 and 5555 respectively.

Since gossip messages may take several alternative routes to reach a particular member based on the state of the system, it is important that we have a way of identifying if a particular member has seen a specific gossip message. We propose a simple solution here to include, for each gossip, a unique identifier generated by the initiator of the original gossip message. This identifier is retained when the message is forwarded by other members in the system. Thus as long as each member can include his unique id(ipaddress/hostname or some other unique identifier or a combination of identifiers) and some variable to distinguish between successive messages(counter/local timestamp) in the identifier for the gossip message, all the gossip messages in the system will be unique.

Deciding when a gossip message should stop propagating is also important because we do not want to have the same message being passed around indefinitely in the network. We can use the concept of a timestamp or a Time to Live field to achieve this. We propose using a Time to Live field because we believe maintaining the notion of some consistent time among all the members(even loosely synchronized) may not scale easily.

Finally we come to the issue of ensuring that everyone in the coalition hears about the gossip message. Because of the inherent randomness in any form of gossip

protocol, it is very hard to perform a mathematical analysis of the protocol to come up with an upper bound for the time by which everyone in the system receives the message or to come up with probabilities for everyone hearing the message after a certain amount of time. The difficulty in coming up with such models have been identified by others also[] and simulations are commonly used to judge the efficiency of the gossip protocols. However there are some heuristics we can use to try to ensure that the messages are propagated to all members expediently. To achieve this we propose the following approaches:

- Before forwarding a message the sender adds to the message the list of all the members he is forwarding the message to.
- On receiving the message, a member checks the list of members who have received the message and sends it to other members in the coalition who do not overlap with this list.
- On a member receiving the same message again later(through a different path), he does not resend the message to his trusted group. He instead sends the message to a random set of nodes in the coalition that have not received the message before.

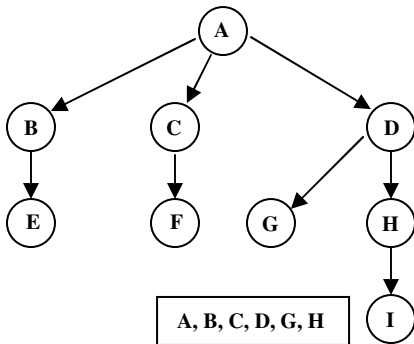


Figure 3: List of members in message sent by A as seen by I

The concept of including the path in the gossip message has been proposed before[]. In that case it was used to verify the integrity of the message by requiring that a message be received from more than a certain threshold of distinct paths before it is accepted. Our approach is a little different because we not only include the path the message has followed but also the list of members who received this message from the predecessors along the path. This adds extra overhead to the message as it propagates through the system but we feel this is

acceptable as the member identifier is usually very small.

The ejection of the bad member here again is on a per individual basis as in the previous models. The main drawback of this approach however is that our goal of achieving a consistent state is entirely dependent on whether the gossip message reaches all members in the coalition.

IV. Hierarchical Model

Some of the problems identified in the broadcast and gossiping models can be mitigated if we use a hierarchical approach for trust management. However it also introduces new issues not found in the previous models. Here we explore how a hierarchical mechanism may be used to achieve our goal of a consistent state among members.

Hierarchical Layering Setup: We incorporate the idea of layering in the Shamon Architecture. We assume a root node or authority whose primary responsibility is to maintain integrity and distributed trust in the coalition. It maintains information about all nodes in the coalition. There is no limitation on the number of layers that exist below the root node. Each node at a level or layer is responsible for a certain number of nodes below itself.

Joining the Coalition: A member who wants to join the coalition sends a request to a random node in the coalition. After considering certain factors like workload, that member may reply to the node wanting to join. Mutual attestation is carried out before formally accepting a node to the coalition. The node is required to comply with the commonly accepted policy of the coalition. The new member joins the coalition at a level that is one lower than the level of the member it attested to for joining and is added as a child to that node in the tree.

Once a node accepts a new member to the coalition, it is henceforth responsible for that node. It also is required to inform the root node about the new member. This information is passed using direct communication with the root node and is accepted by the root node after attestation of the member.

Member Ejection: Member ejection in a hierarchical model is more complicated as there are various issues related to the position of a member in the layering Setup of the model. We have analyzed these issues mainly from two different perspectives. We have considered different cases in terms of their method of propagating

the message of an identified bad member to other members in the coalition.

The stage where confirmation of the bad node is carried out and the stage when the node is ejected from the coalition on an individual basis, is essentially the same in both the cases. However the reporting stage differs drastically and has been discussed below. Each of these two approaches use the lazy consistency model and each method has its advantages and limitations discussed later.

Case 1: When a node identifies that any node apart from its predecessor in the tree is bad

A member that learns of a bad member passes that message to all other members who are one level below itself in the hierarchy as well as the member who is responsible for it in the layer above itself. Now there are two scenarios or cases that need to be considered here;

If a node receives this message from a member above itself in the hierarchy, he passes that message to all members who are one level lower to itself. This is the case when a node at an intermediate level finds out that a member is bad. He propagates the news to all members a level lower as well as to the node responsible for it. The nodes at the lower level have to continue passing on this message to make sure that even the leaf nodes get this message eventually.

If a node receives this message from a node below itself in the hierarchy, it passes this message to all nodes one level below (excluding the informer node), as well as to the node that is responsible for it in the higher level.

In this manner, it is assured that the message is propagated till it reaches all members till the leaf node in the hierarchy. All members receiving this message simply flag both, the informing node as well as the claimed bad node. In this model too, lazy consistency is implemented. No immediate attestations result from receiving a message about a bad node. Attestation is required only when need for communication arises.

Case 2: When a node identifies that its immediate predecessor in the tree is bad

When a member has identified that its immediate predecessor in the tree is bad, he directly informs the root node. The root node performs attestation of the claimed bad member before accepting the information to be true. In case the claimed bad member is found to be good, the informer node is assumed to have gone bad

and is ejected from the root node's list of trusted members. Once the root node has confirmed the identity of the bad member and has ejected that member, he informs nodes which are one layer below it in the hierarchy and all sub-nodes of the bad node. A lazy consistency model is assumed here. These nodes flag both, the claimed bad member and the member sending the message, as being in unknown state if they are not part of our incoming trusted group as described earlier. The nodes do not immediately attest the bad node or the node informing it of the bad node since they do not need to immediately communicate with either of the nodes. This continues till the message reaches all members at the leaf nodes.

While studying these arrangements we came across various limitations of this model and cases where the model will fail. Consider, that a bad node initiates a false message claiming a good node to be bad, and passes that message down the hierarchy. Now, if each node flags only the informing node which is one level higher and the claimed bad node, after the message has propagated down a few levels in the tree, the nodes will not know the identity of the actual bad node who initiated the message. The pair of untrusted nodes maintained by each node will not include the identity of the actual bad node. Therefore before communicating, a check of the integrity of the claimed bad node will be carried out through attestation. The integrity check will succeed as that node was falsely claimed to be bad. The informer node at the level higher will now also next clear the check if one is performed or worse still be falsely marked as bad if no check is deemed necessary. Hence, the actual bad node is never ejected from this particular nodes' list of trusted members.

This can be a serious problem. We suggest having the identity of the originator of a message (the node who initially claimed some member to be bad), in the message. Therefore all nodes receiving a message about the claimed bad node will flag the originator of the message and the claimed bad node.

This solution seems feasible initially, but it has very serious security flaws. A bad node may spoof the identity of the originator in the message it sends out. Therefore, all nodes will flag that spoofed node as being untrusted. They will, before communicating with him, attest him. Thus, if many such spoofed messages with a certain originator address are sent out, a denial of service attack can be launched as the supposed originator node will not be able to provide his normal services or communicate with any other member without prior attestation. We are currently exploring more options to mitigate this problem

Irrespective of the specific model followed for reporting, the confirmation stage remains the same. If a member needs to communicate with a member that it has flagged, he attests the node. If the node is actually good, then the informer node is attested.

4 Experiment

Experimental procedure: Theoretical models of consistency of belief regarding the state of a bad member is sketched in Shamon architecture. These theoretical models are analyzed for the following metrics.

The following parameters are varied in each mode :

- No. of members in the coalition (N)
- No. of members claiming a member has gone bad(X)
- Type of consistency in communicating removal of bad member.

The following parameters are measured:

- Validity of the state of the coalition
- No. of messages communicated
- No. of attestations required

The variables in our system are:

- N = Number of nodes in the network
- X = Number of nodes who have identified a person as bad
- W = weight of an attestation message with respect to other messages(inform client bad messages etc)
- B : The Client claimed to be bad by the group X
- S_B : Subset of nodes who want to communicate with the claimed bad node B
- S_X : Subset of nodes who want to communicate with one or more nodes in group X

We consider each run of the protocol for a single bad member because additional members going bad do not influence the working of the protocols. This is because in all our models, information is accepted only after remote attestation or verification of the current hash values of the other member. In the trivial and lazy model, we assume that the underlying network is reliable any message sent is delivered to the recipient. The gossip model is susceptible to bad nodes dropping gossip messages but the effect is mitigated by using

multiple paths for gossip to each node. In the hierarchical model this problem is directly addressed by having the root node manually skip the bad node is distributing the update messages.

Trivial Model

In order to compare the performance of this model, we calculate the number of messages that will be required to report a bad member, confirm his state (done by all nodes individually), and finally eject him from the coalition.

[1] Number of messages (M_1) to inform other members of the coalition that a client has gone bad;

$$M_1 = (N-2) X$$

Here, (N-2) implies that the node will send the message to all nodes excluding itself and the node it has identified as being bad. Broadcast is assumed as the mode for transmitting the message in this case.

[2] Number of messages (M_2) required to attest the node that is claimed to be bad;

$$M_2 = W (N-(X+1))$$

All nodes who have not identified the node to be bad are (N-X). All these nodes will try to attest the bad node. However (N-X) also includes the bad node. Therefore, excluding that node, the number of attestations = N-X-1. W as stated earlier is the weight of the attestation message.

[3] Number of messages (M_3) to inform the node that others were claiming it to be bad when it is actually identified as good after remote attestation;

$$M_3 = N-(X+1)$$

This is required when a node that was claimed to be bad by X, has been identified as good after remote attestation by the other nodes in the coalition. Therefore, the node claimed to have gone bad, should be informed of that fact that there are X members in the coalition who are passing bogus information about that node.

[4] Number of messages (M_4) to carry out attestation for checking the nodes claiming that a node is bad, i.e., for checking X;

$$M_4 = W ((N-X) X)$$

Since all the X nodes wrongly claimed someone was bad, each good node in the system will now try to attest

the X members to validate their state. This procedure requires X attestations (messages).

Scenarios [3] and [4] are realized only when a node claimed to be bad is recognized to be good by means of remote attestation by the members of the coalition. It is therefore necessary to consider the Truth value, i.e., the probability of the claim that a member is bad, being true (of value 1).

Using these cases, the Total number of messages (M) that will actually be transmitted to eject a bad member using this model can be calculated using the formula stated below;

$M = M1 + M2 + T(M3 + M4)$ where, T is the Truth value of the claim, as explained above Using [1], [2], [3], [4],

Total Number of messages (M) = $(N-2)X + W(N - (X+1)) + T((N-(X+1)) + W((N-X)X))$

Lazy Model

Clients we want to communicate with	Number of Attestations	Number of messages inform B	of
Only with bad Client B	$ S_B $		
With one or more in Group X and B who is bad	$ S_B $		
With one or more in Group X and B who is good	$ S_B $	$ S_B $	
Only with one or more in Group X and node/nodes in X found good	$ S_X $		
Only with one or more in Group X and node/nodes in X found bad	$ S_X $	$ S_X $	

There are five cases possible here:

1. Number of messages to inform others that a client has gone bad assuming broadcast: $(N-2)X$, a node will send the message to all nodes excluding itself and the node it has identified as being bad

2. Number of Attestations when a node is claimed to be bad: $|S_B| + |S_X|$, since all clients who either communicate with B or any client in group X will require attestation. Note that we use only one attestation of any client in group X to make a decision on all clients in group X.

3. Number of messages to inform the a node that others were claiming it to be bad if it is identified to be good: $|S_B| + |S_X|$, because both these subsets will want to inform B that the group X wrongly claimed that B was bad

4. Number of Attestation by B for checking the nodes in group X: 1 if B wants to communicate with one or more clients in group X

Gossip Model

No. of Gossip Messages(M):

For all the nodes in the network –

\sum If_Node_Visited_Truth_Value[[(No. of nodes in hash matching subset of trusted group)+ (No. of Random Nodes) + [No. of non-visited random nodes]]

No. of Attestations(A) = No. of Random Nodes Informed for eager consistency or $|S_B| + |S_X|$ for lazy consistency.

Total No. of Messages = $M+W(A)$, where W is the weight of the attestation

Hierarchical Model

Total number of messages:

$X + W + T(\text{Truth Value of claim}) (W) (X) + (N-4)$

Where:

X: Number of messages to inform root node about a bad member

W: Number of attestation messages when root node attests the bad members (W) (X): Number of messages to attest X nodes incase the bad member is recognized as actually being good

(N-X-2): Number of messages to inform all nodes eventually about the bad node (excluding initial informer members, root node, bad member).

5 Conclusion and Future work

Maintaining consistency of beliefs of all members of a coalition is important in Shamon Architecture. It is vital to implement a model which allows members of a coalition to identify a bad member and inform others about its identity so no good member in the coalition communicates with the bad member once he has been identified. On comparing the above stated theoretical models for establishment of consistency in the group, the gossip and hierarchical models pose a good trade off between performance and number of messages communicated in achieving consistency of the group on the state of a bad member. The Gossip model stated above has higher probability of resisting DOS attacks as the message that is being gossiped has the list of all the members it has passed to and hence a good member will not resend the same message to any member who has already received it. However, this does not prevent the malicious members from flooding nodes. This flooding can be prevented by band limiting the network channels. The above Gossip protocol gossips only through push messages i.e messages are only pushed from one member to the other and not pulled from others when required. The Gossip protocol can be modified to have separate band limited push and pull channels so that even if one of them is flooded communication among members is still possible through the other channel. We will explore these options in future. The hierarchical model also achieves good performance with the only drawback being that the compromise of the root node significantly degrades the performance of the system. To alleviate this problem we will explore options for electing new root nodes in a coalition when the current root node has been compromised.

References

1. Jonathan M. McCune, Trent Jaeger , Stefan Berger, Ramon Caceres, Rainer Sailer. Shamon: A System for Distributed Mandatory Access Control.
2. T. Jaeger, R. Sailer, and U. Shankar. Prima: Policy-reduced integrity measurement architecture. In Proceedings of the 11th ACM Symposium on Access Control Models and Technologies (SACMAT), February 2006.
3. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A virtual machine-based platform for trusted computing. In Proceedings of the 19th ACM Symposium on Operating System Principles(SOSP 2003), Bolton Landing, NY, USA, Oct. 2003.
4. T. Jaeger, S. Hallyn, and J. Latten. Leveraging IPsec for Mandatory Access Control of Linux Network Communications. In Proceedings of the 21st Annual Computer Security Applications Conference, Tuscon, AZ, USA, Dec. 2005.
5. E. Shi, A. Perrig, and L. van Doorn. BIND: A Fine-grained Attestation Service for Secure Distributed Systems. In Proceedings of the 2005 IEEE Symposium on Security and Privacy, Oakland, CA, USA, May 2005.
6. Miguel Castro and Barbara Liskov. Proactive Recovery in a Byzantine Fault Tolerance System. Laboratory of Computer Science, MIT.
7. Ali Aydin Selcuk, Ersin Uzun , Mark Resat Pariente. A Reputaion Based Trust Management System for P2P Networks. Department of Computer Engineering, Bilkent University, Ankara.
8. D. McDermott and J. Doyle. Nonmonotonic logic I. Artificial Intelligence, 13:41–72, 1980
9. Pete Keleher, Alan L. Cox, Willy Zwaenepoel. Lazy Release Consistency for Software Distributed Shared Memory. Proc. of the 19th Annual Int'l Symp. on Computer Architecture (ISCA'92), (1992).
10. Robbert van Renesse, Yaron Minsky, Mark Hayden. A Gossip-Style Failure Detection Service. Dept. of Computer Science, Cornell University. (1996).
11. Matt Blaze, Joan Feigenbaum, Angelos D. Keromytis. KeyNote: Trust Management for Public-Key Infrastructures. Lecture Notes in Computer Science (1998) .
12. Matt Blaze, Joan Feigenbaum, Jack Lacy. Decentralized Trust Management. Proceedings of the 1996 IEEE Symposium on Security and Privacy
13. Dahlia Malkhi, Elan Pavlov, Yaron Sella, Gossip with Malicious Parties (2003)
14. Dahlia Malkhi, Yishay Mansour, Michael Reiter, On Propagating Updates in a Byzantine Environment Aug 1999
15. Minsky Y., Schneider F., “Tolerating Malicious Gossip”, The Distributed Computing Journal., 16(1):49-68, February 2003
16. van Renesse R, Scalable and secure resource location, IEEE Proceedings of the 33rd Annual Hawaii International Conference on System Services, 2000.
17. R. Van Renesse, R. Minsky, and M. Hayden, “A Gossip-style Failure Detection Service,” Proc. of the IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing Middleware, England, September 15-18, 1998, pp. 55-70.