

A service-centric approach to access control and monitoring based on distributed trust

Jimmy McGibney and Dmitri Botvich

Telecommunications Software & Systems Group, Waterford Institute of Technology

Abstract

A service-oriented approach to dynamic refinement of security enforcement is described in this paper. This is based on a closed loop feedback system where live distributed trust measures are used to adapt access control settings in a changing threat environment. A general trust overlay architecture and model are presented. Some specific application scenarios are discussed, in particular spam filtering and distributed intrusion detection in mobile ad hoc networks. It is shown using simulations of some specific scenarios that this dynamic system is robust and has the potential to enhance security and access control efficiency.

1 Introduction

Trust is at the heart of the security of electronic communications and transactions. A successful online transaction requires that the parties involved sufficiently trust each other and the infrastructure that connects them.

Weaknesses in the currently approach to trust establishment and management include dependence by organisations on perimeter security that encourages a binary on/off approach (inside is trusted; outside is not) and a reliance on centralised trust references like certificate authorities that do not sit well with rich peer to peer interaction. In a truly peer to peer network, there is no way to store trust information centrally. In any case, trust by its nature is subjective and thus best managed by the entity doing the trusting.

In this paper, we present an architecture based on a distributed trust management overlay as well a simple model for recording and updating trust based on experience and recommendations. We also outline a collaboration protocol, discuss candidate algorithms, and report on the application of our approach to particular non-trivial scenarios. Algorithms need to take into account the risk of some “bad” nodes deliberately attempting to corrupt the system, possibly in collusion with each other.

Note that *services* are at the heart of the model that we present. In fact, all relevant activity is modelled as service usage. Nodes offer services and peer nodes use them. Access to some such services requires more trust than access to others. This service-centric approach is described in more detail in [1]. In the case of a mobile ad hoc network, peer-to-peer services could be available such as voice, video, instant messaging and file sharing. Having such services enabled on a node could expose that node to various types of attack, even though it might have good protection mechanisms in place and be kept up to date with patches, etc.

Distributed trust is of course not a new area and there is substantial body of published work. Section 6, on related work, discusses this. The novelty of our approach is in attempting to extract the essentials of distributed trust that are application-independent and to use these in a lightweight way to provide feedback from service usage monitoring to access control.

The remainder of the paper is organised as follows. An overlay architecture and protocol are proposed in section 2. Section 3 discusses some potential applications. Section 4 presents a general model. Section 5 describes experimental simulations. Related work is discussed in section 6, and section 7 concludes the paper.

Copyright © 2007 Jimmy McGibney and Dmitri Botvich. Permission to copy is hereby granted provided the original copyright notice is reproduced in copies made.

2 Distributed trust management

For the purposes of this work, we adopt a two layer model for communications between nodes. Nodes can either interact for service usage or to exchange trust information. For modelling purposes, each service usage interaction is a discrete event. A logically separate trust management layer handles threat notifications and other pertinent information.

We mimic social trust by setting a fuzzy trust level. Each different service can then make an appropriate decision based on this trust level – e.g. certain actions may be allowed and others not. In our system, we model trust as a vector. In the simplest case, at least if there is just one service, this can be viewed as a simple number in the range (0,1). Each node may then maintain a local trust score relating to each other node of which it is aware. If node A’s trust in node B is 1, then node A completely trusts node B. A score of 0 indicates no trust. Note that this is consistent with Gambetta’s widely accepted definition of trust as “*a particular level of the subjective probability with which an agent assesses that another agent or group of agents will perform a particular action, both before he can monitor such action and in a context in which it affects his own action*” [2]. If trust is to be a probability measure, then the (0,1) range is natural.

2.1 Architectural overview

We propose the overlay of a distributed trust management infrastructure on top of the service delivery infrastructure (which may itself contain multiple layers).

Figure 1 illustrates the relationship between underlying services and this new infrastructure. With our proposed trust overlay architecture, a trust management layer operates separately from all aspects of service delivery and usage. Two message passing interfaces are defined between the service usage layer and the trust management layer and another between the trust managers of individual nodes. The interfaces are as follows (Figure 1):

1. Experience reports: Service engine \rightarrow Trust manager
2. Trust recommendations: Trust manager \leftrightarrow Trust manager

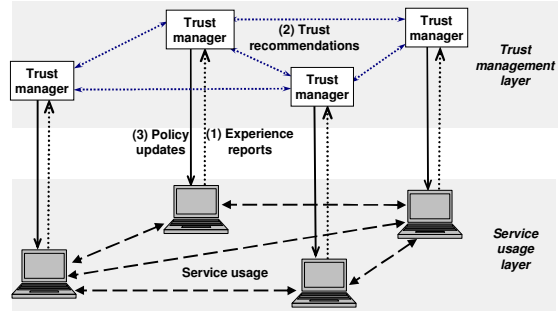


Figure 1: Trust overlay helps to secure usage of services in ad hoc network.

3. Policy updates: Trust manager \rightarrow Service engine

Service usage is as normal, with the trust information just influencing protection mechanisms. The trust manager gathers usage experience and uses this together with the experience of collaborators to inform its trust in other nodes.

In the initial case, all trust values are set to a low value. Having initial values set to zero would prevent the so called *Sybil Attack* [3], whereby attackers can take advantage of a default trust level by maintaining multiple identities. This is impractical though as we need to have some low-risk services enabled in order to allow nodes to get experience of other nodes.

We then need to have a system to build up trust as nodes gain experience of each other or learn of each other’s reputation. In our system, trust can be updated in two ways:

1) *Direct experience*: On completion of a service usage event between two nodes, each node updates its trust in the other based on a measure of satisfaction with the service usage event.

2) *Reputation*: Node A notifies other nodes in its *neighbourhood* of the trust score that it has for node B. This will change significantly following a security-related event.

How this neighbourhood is defined is significant. The neighbourhood of a node is the set of nodes with which it can communicate or with which it is willing to interact. The choice of neighbourhood nodes is up to each individual node to decide, and may depend on physical distance, topology, frequency of contact, or even trust level.

The main benefit of this system is in using these trust scores to tune security settings. Trusted nodes can be dynamically provided with more

privileges than untrusted nodes. In our system, as mentioned, we model all interaction between nodes in terms of services. Each node then sets a threshold trust level for access to each service in which it participates. If the trust score of a node decreases, for example due to detected suspicious activity by that node, services available to that node are reduced.

2.2 Trust overlay protocol

This section describes a generalisation of a protocol already proposed by the authors in [4] to support trust-based email filtering, named TOPAS (Trust Overlay Protocol for Anti Spam). The TOPAS protocol allows for the collection of spam statistics to calculate trust, sharing this trust information between nodes and feeding it back to mail hosts to allow them to more effectively filter spam.

The protocol executes using asynchronous message passing in the case of interfaces (1) and (2) of Figure 1. Interface (3) uses a simple synchronous request-reply technique. An underlying set of services is assumed, including message delivery, failure detection and timeout. It is assumed that each process receives queued messages of the form $(tag, Arg_1, \dots, Arg_n)$. This outline style of protocol specification is influenced by the Generic Aggregation Protocol (GAP, [5]).

(1) *Experience report: Service host* \rightarrow *Trust manager*

The service host has an associated access controller. Each service usage attempt is processed by the access controller, with the result that it is either accepted or blocked. Accepted service usages are then monitored (e.g. by an intrusion detection system (IDS), or by a spam filter if the service is email) and determined whether the usage was benign or malicious. The result of this assessment of the service usage needs to be made available to the trust manager. The following two messages, from the service host to the trust manager, provide this:

- $(singleUsage, i, a)$ may be sent from the local service host to the trust manager to report on a single service usage from node i . The value of a is 1 if the service has triggered an alarm and 0 otherwise. i identifies the node using the service.
- $(bulkUsage, i, n, a)$ may be sent from the local service host to the trust manager to report on a

sequence of service usage events involving node i . n is the number of discrete service usages by node i (since the last report) and a the number of these determined by the service host to be malicious.

Note that in both messages above, the node identifier i could be an IP address, a hostname, a domain name or any other identifier supported by the service host. It is up to the trust manager to process this.

Sending a *bulkUsage* message conveys the same information as would several *singleUsage* messages. It is included in the protocol for performance reasons. Where usage volumes are heavy, it is more efficient to send periodic *bulkUsage* updates rather than burden the server with generating a *singleUsage* message per usage event.

(2) *Trust recommendation: Trust manager* \leftrightarrow *Trust manager*

Nodes collaborate to share trust information with one another. This is done by the trust managers, using the primitives outlined here. Trust managers may issue recommendations spontaneously (for example on a usage alarm) or in response to a request from another node. Note that a request may be issued by one trust manager to another, but a reply is not guaranteed. As mentioned earlier, message passing is asynchronous and the protocol requires no state information to be maintained by the corresponding entities. The following five messages may be sent from one trust manager to another:

- $(getTrust, k)$ allows node i to ask another node j to report its trust in node k . This message should be interpreted as an indication from node i of a desire to receive a recommendation regarding node k . Node j may respond with a trust report. This might be expected to be used as follows. On receipt of a service usage request from node k , node i could issue a series of *getTrust* messages to neighbouring nodes requesting recommendations regarding node k . Only those neighbours with some experience or knowledge of k might reply, with the remainder staying silent. Note though that nodes are not obliged to respond, even if they do have knowledge or experience of k . Nodes could have other reasons to not reply. (e.g. lack of trust in the requester, i) Nodes that have no information are expected to stay silent.

- (*getTrustResponseRequested*, *k*) is a variation on *getTrust* that expects the recipient to respond (with a null value) even if it has no trust information on node *k*. Again, there is no strict obligation to respond.
- (*getTrustAll*) allows node *i* to ask another node *j* to report its trust in all known nodes. This message should be interpreted as an indication from node *i* of a desire to receive a recommendation regarding all nodes of which the recipient is aware. There is no obligation on the recipient to respond. Nodes with no information on any nodes should issue a null reply.
- (*setTrustReportingPreferences*, *t*, *c*, *r*, *f*) allows node *j* to specify to node *i* how spontaneous trust reports are sent to it (see discussion on “push” model later). This message also indicates a desire by node *j* to receive trust advertisements from node *i* (i.e. to be included in its neighbourhood).
t is a list of zero or more trust level thresholds. Trust updates are requested whenever trust exceeds or falls below any of these threshold values.
c is a confidence level threshold. Trust updates are only desired if the confidence level of the sender in this trust assessment is at least *c*.
r is a recency threshold. Trust updates are only desired if the trust information has been updated by the sender within the previous *r* time units.
f indicates the maximum frequency of update.
- (*trustReport*, *k*, *T*) allows node *j* to send a recommendation to another node *i*, in relation to node *k*. *T* is an object that encapsulates sender *j*'s trust in node *k*. *T* is set to null in the case where the sender *j* has no trust information regarding node *k*. Note that a *trustReport* may be issued either spontaneously or in response to a *getTrust* or *getTrustResponseRequested* message. Issuing of spontaneous *trustReport* messages to a node should be in accordance with that node's preferences (expressed using *setTrustReportingPreferences*).
- (*bulkTrustReport*, *l*) allows node *j* to send a recommendation to another node *i*, in relation to a set of nodes. Parameter *l* is a set of pairs (*k*, *T*) where *k* is the node identifier and *T* is an object that encapsulates sender *j*'s trust in node *k*. *l* is the empty set in cases where the local node has no trust scores to share. Note that a *bulkTrustReport* may be issued either

spontaneously or in response to a *getTrustAll* request.

(3) Policy update: Trust manager → Service host

The third part of this collaboration architecture is responsible for closing the loop. Direct experience is recorded by nodes and shared among them. The result of this experience and collaboration is then used to inform the service host to allow it to operate more effectively. Specifically, the service host, on receiving a usage request from node *i*, needs to be able to access the current trust information that its trust manager has on node *i*. Although the service host may be able to use local storage to, for example, cache trust values, we do not place any such requirements on it. Thus we need the ability for the service host to request this trust information from the trust manager and receive a timely reply.

- (*getTrustLocal*, *i*) allows the service host to request the trust score recorded for a particular node, *i*.
- (*trustReportLocal*, *i*, *T*) allows the trust manager to respond to a *getTrustLocal* request. *T* is an object that encapsulates the trust manager's trust in node *i*. *T* is set to null in the case where the trust manager has no information regarding node *i*.

Note on using this protocol

The functions specified at interface (2) above allow for either a “pull” or a “push” model for sharing trust information. Messaging is asynchronous for experience reports and recommendations.

“pull” model

The trust manager receives a *getTrust*, *getTrustResponseRequested*, or *getTrustAll* request for trust information about another node, or all nodes, and subsequently replies with a *trustReport* or *bulkTrustReport* message. The sender of the request will need to use a timeout mechanism. In the case of a *getTrust* message, there is no onus on the recipient to reply. With *getTrustResponseRequested* and *getTrustAll*, the recipient should issue a reply even if this contains no trust information.

“push” model

The trust manager may be configured to spontaneously issue trust updates, either to other nodes or to its local service host. This is typically for reasons of performance and efficiency. It is wasteful for nodes to repeatedly poll each other unless

there is useful new information. In the “push” case, each node decides when and to whom to issue trust advertisements. The *setTrustReportingPreferences* request, described above, allows each node to specify how spontaneous trust reports are sent to it

3 Some example applications

In this section, we identify some services for where this trust overlay has the potential to provide enhanced protection. In particular, we have already studied and carried out simulations on two very different types of service, namely email (filtering for spam) and ad hoc networks (intrusion detection). We also briefly mention other applications that we feel could benefit from this approach.

3.1 Email: spam filtering

Spam is probably the greatest single nuisance for users of the Internet, and is also a significant security threat. Despite significant efforts, the incidence of spam remains stubbornly high and there is a feeling that spammers are always one step ahead. When a Mail Transfer Agent (MTA) receives an email for relay or delivery to a local mailbox, it would like to be confident that this email is not spam. Mail servers may block mail based on source MTA or the contents of the mail itself. With our approach, each mail server establishes a live profile of the trustworthiness of other mail servers, with respect to tendency to send spam, either deliberately or because of slack anti-spam enforcement. This trust score in the sending (or relaying) mail server is then used to improve spam filtering by causing filter thresholds to be varied.

3.2 Mobile ad hoc networks: intrusion detection

New wireless and ubiquitous computing technologies encourage the set-up of ad hoc networks that may not have access to centralised resources such as a public key infrastructure to validate credentials like digital certificates. Despite this, the need to establish trust and security is at least as great as with traditional networks. Actually, the risks to security are inherently greater with the use of shared radio spectrum. Moreover, there has

been a history of weak wireless security protocols, and traditional perimeter security is of little benefit.

With our approach, such loosely connected nodes collaborate to isolate sources of attack. In Figure 2, Node A detects an attempted attack or service misuse by node X and notifies peers of this using a simple trust management overlay.

3.3 Other applications

We mention here some other applications and services that may be suitable for applying this kind of distributed trust overlay. Further work is required to evaluate these more closely.

General Internet communications would be much more secure if both the domain name system (DNS) and routing could be trusted. Given a destination domain name, a message sender would like to be confident that packets sent are securely delivered to the desired destination. Normal DNS and routing provide little assurance of this. Relatively new, but not widely implemented, protocol enhancements aim to address this. DNSsec proposes a DNS infrastructure that provides for hierarchical validation of signed DNS bindings. More secure versions of routing protocols, such as Secure BGP, also exist. Some work has already been done on applying distributed trust to ad hoc routing (see section 6 on related work).

So called digital ecosystems are also of interest. This envisages a world of small specialist service providers, from which more complex user-oriented services are composed by other service providers. Trust requirements here are a little complex. Users can only be expected to pay for the composed service provided, but what happens if some service providers deliver a high quality service and other do not (e.g. video without audio)?

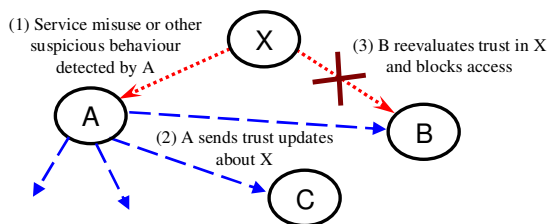


Figure 2: Node A detects service misuse or other suspicious behaviour by X, and notifies neighbour nodes. One of these, node B, chooses to block interaction with X as a result.

The composing service provider needs to trust the specialist provider, and the specialist providers may also need to trust one another if there is a risk of non-payment due to a peer not delivering as required.

4 Model description

4.1 Assumptions

In this section, we present a model for using trust information to enhance security through collaboration in a rich multi-service decentralised environment. One of the problems of modelling decentralised systems like ad hoc networks is that it is unrealistic to take a top-down “bird’s eye” view. Thus we model the set of nodes as those nodes of which a specific node is *aware*.

We also make several assumptions. We assume that neighbour discovery and routing services are in place. We also assume that some kind of service registration and discovery is available to allow nodes to reach an understanding of the set of services available and their associated trust thresholds. It should be noted that the assumption of a common threshold across all nodes that provide the same service is something of a simplification. Even having all nodes share an understanding of the relative meaning of trust threshold values is non-trivial.

We also assume authentication of identity. This could be done, for example, by having nodes exchange public keys on their first interaction (in fact the public keys could be used as unique node identifiers). Further messages between those nodes could then be signed by the sending node’s corresponding private key so that the recipient could at least be confident that the sender is the same entity as that for which it has been building up a trust profile.

4.2 General model

Topology: Let $V_i = \{1, \dots, N_i\}$ be the set of nodes of which node i is aware. There is no single view of all nodes in the system, as management is decentralised. Some of these nodes will be *adjacent* to node i , normally by reason of network topology. This can be modelled by an adjacency vector, $A = (a_{i,j})_{j=1, \dots, N_i}$, where $a_{i,j} = 1$ if pair (i, j) are neighbours and $a_{i,j} = 0$ otherwise.

Services: Let $S = \{S_1, \dots, S_M\}$ be the set of services that may be provided. Each individual node j provides a subset of these services $S^j \subset S$. Some nodes will just be service consumers, so S^j will in those cases be the empty set.

Trust thresholds: Each service S_x has an associated trust threshold t_x .

Representing Trust: We denote the local trust that node i has in node j as $T_{i,j}$. Each other node $k \in V_i$ will maintain its own local view of trust in node j , which may, as we shall see, influences the local trust of node i in node j . We model $T_{i,j}$ as a trust vector in general. In the simplest case, this can just a number, but such a number may be associated with other attributes that relate to it, such as confidence in the trust score or its recency, or the service(s) to which it relates.

Trust initialisation: In the case where node i has no prior knowledge of node j , we will have $T_{i,j} = x$, where x is the default trust.

Trust decision: (using trust in service protection) When node j attempts to use service S_x provided by node i :

- Service use is permitted if $f_x(T_{i,j}) > t_x$, where the function f_x maps trust vector $T_{i,j}$ onto a scalar number in the range (0,1) for service S_x
- Otherwise node j is blocked from using service S_x

Trust update following service usage: After a service usage event, by node j on node i :

- If the outcome is positive, then $T_{i,j}$ is increased according to some algorithm
- If the outcome is negative, then $T_{i,j}$ is reduced according to some algorithm

In general, following service usage, we update $T_{i,j}$ according to:

$$T_{i,j} := f_e(T_{i,j}, E)$$

where E is a vector of attributes related to the service usage event and f_e is a function defining how trust is updated based on service usage experience.

Trust update following referral by a third party: Node i may receive a message from a third party node, k , indicating a level of trust in node j . This can be modelled as node i adopting some of node k 's trust level in node j , $T_{k,j}$. In general, following such a third party referral, we update $T_{i,j}$ according to:

$$T_{i,k} := f_r(T_{i,k}, T_{i,j}, T_{j,k})$$

where f_r is a function defining how trust is updated. This trust transitivity depends on $T_{i,k}$ as node i can be expected attach more weight to a referral from a highly trusted node than one from a less trusted node.

4.3 Some algorithms for trust update

The way trust is updated based on both experience and recommendations has a profound impact on the usefulness of this kind of overlay system. Note that nodes are autonomous in our model and each might implement a different algorithm for updating and using trust. It can also be expected that a node's behaviour in terms of handling trust may change if it is hijacked. Potential trust update algorithms include:

Moving average: Advanced moving averages are possible, where data is "remembered" using data reduction and layered windowing techniques.

Exponential average: Exponential averaging is a natural way to update trust as recent experience is given greater weight than old values. Also very little memory is required in the system, making it more attractive for implementation than a moving average.

No forgiveness: This in a draconian policy where a node behaving badly has its trust set to zero forever. Even more extreme is where a node that is reported by a third party as behaving badly has its trust set to zero forever. This might be applied if a particularly sensitive service is misused.

Second chance (generally, n^{th} chance): Draconian policies are generally not a good idea. IDS, anti-spam, and other security systems are prone to false alarms. A variation on the "no forgiveness" approach is to allow some bounded number of misdemeanours.

Hard to gain trust; easy to lose it: To discourage collusion, there is a case for making trust hard to gain and easy to lose.

Use of corroboration: To prevent an attack by up to k colluding bad nodes, we could require positive recommendations from at least $k+1$ different nodes.

Use of trust threshold for accepting recommendations: It is possible to model the ability to issue a recommendation as a kind of service usage on that receiving node. Thus the receiving node can apply a trust threshold to decide whether to accept that recommendation in the same way as any service usage attempt is adjudicated.

5 Experiments

In this section, we present and discuss the results of several simulations to evaluate the stability of this system and its effectiveness in detecting attacks, including the incidence of false alarms. Two example applications of our approach are considered, namely spam filtering and intrusion detection in ad hoc networks.

Experiment 1: Convergence and stability of trust scores

In our first simulation, we have a small fully connected network of fifty nodes and a single service (e.g. email). A default initial trust score of 0.2 is used for illustration purposes (to distinguish the bad from the simply unknown) and the trust updating algorithm is a simple exponential average for both direct experience and recommendations. Of the fifty nodes, a small number produce a varying level of spam. Figure 3 shows that the trust recorded by a good node ("node 1") for each of these less reliable nodes converges to a different value, depending on the level of spam encountered. This is encouraging as we can envisage

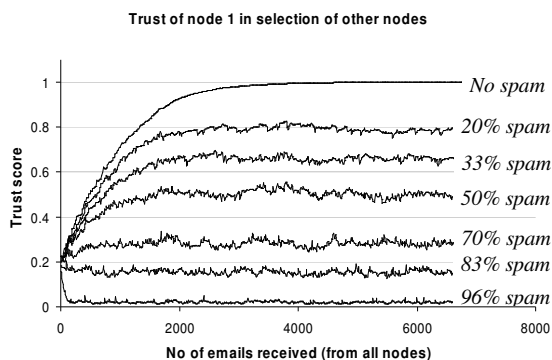


Figure 3. Trust level in node converges to different value depending on tendency of node to generate spam.

trust filters that work by subjecting mail to a degree of filtering scrutiny that is appropriate to the trust in the source.

Experiment 2: Single-service scenario (spam filtering)

The objective of this experiment is to see if we can get an improvement in the effectiveness of spam filtering by applying trust scores. The figures below show how this has been achieved in an illustrative case. In this experiment, a network of 50 nodes is again chosen; also there is a single spammer who is responsible for 50% of all email generated in the system. Trust convergence for normal nodes is a moderately fast exponential average. The neighbourhood consists on average of one-seventh of all nodes, so we have a sparse, but still connected, topology.

For this experiment, we choose relatively flat (but distinct) probability density functions for spam indicators for both spam and non-spam email. Both have the Gaussian (normal) distributions shown in Table 1. Note the overlap implied by the relatively large standard deviation value for spam. This attempts to model the fact that in practice parameters of non-spam mail tend to deviate less than those of spam (and hence there are fewer false positives than false negatives).

	Mean	Std Deviation
Spam	8.0	4.0
Non-spam	1.0	2.0

Table 1. Parameters for Gaussian (normal) distributions used in experiments.

As already mentioned, most spam filters combine a variety of measures into a suspicion score and compare this score with a pre-defined threshold. For our experiments, a fixed threshold of 5.0 is chosen (*SpamAssassin* [8] default) and used as a benchmark. As can be seen in Figures 4 and 5 below, a significant reduction in both false positives and false negatives can be achieved with auto-tuning of the threshold (based on trust values). Auto-tuning is of course most effective in a steady-state situation when trust values are quite stable. A range of other predefined threshold values were also tried, but with no better results than the value of 5.0 shown. Choosing a higher predefined threshold causes an increase in false negatives and choosing a lower predefined threshold causes an increase in false positives.

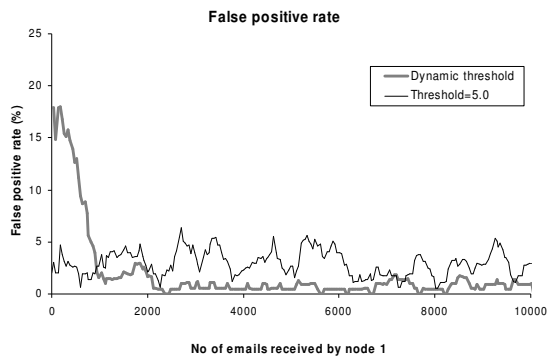


Figure 4. Comparison of dynamic vs. fixed threshold: impact on rate of false positives.

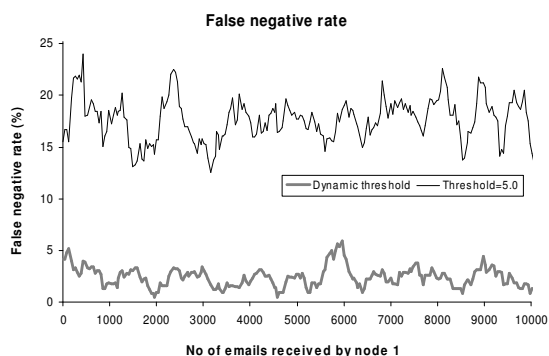


Figure 5. Comparison of dynamic vs. fixed threshold: impact on rate of false negatives.

Experiment 3: Multi-service scenario (IDS in mobile ad hoc network)

We now consider the situation where there are multiple services with different trust thresholds. For these simulations we have three services, A, B, and C with quite distinct trust thresholds, 0.2, 0.5, and 0.9 respectively

For this experiment, we have a simple fully connected network of twenty nodes. With a default initial trust score of 0.25, all nodes will have initial access to service A only and need to earn trust before they are allowed access to services B and C. Of the twenty nodes, five are bad and the remainder are good.

We initially consider a fully connected network – i.e. every node has every other node as a neighbour. In Figure 6, as well as in Figure 7, we plot the percentage of “good usage allowed” that is, the percentage of service usage attempts by good nodes that succeed. We also plot the percentage of “bad usage allowed” that is, the per-

centage of service usage attempts by bad nodes that succeed. Ideally, 100% of good usage attempts and 0% of bad ones should succeed.

We also model the fact that IDSs do not always get it right. Occasionally an attack goes unnoticed (false negative) or benign activity raises an alarm (false positive). We use a simple Gaussian (normal) distribution to model the characteristics of service usage that are being monitored by the IDS. In the case of good nodes, a number is tagged onto each service usage using a normal distribution with mean 2.0 and standard deviation 2.0. In the case of bad nodes, a number is attached to the service usage using a normal distribution with mean -2.0 and standard deviation 2.0. The sign of this number is then used to “detect” whether there has been an attack attempt.

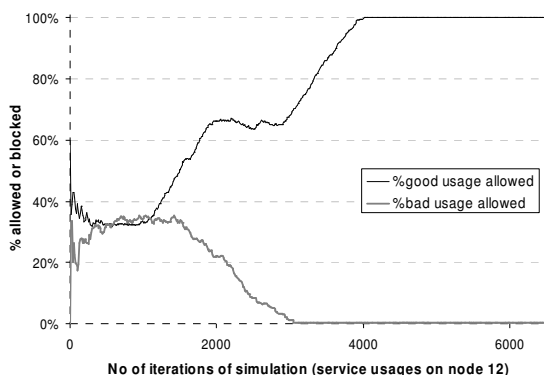


Figure 6. % good usage allowed and % bad usage allowed, aggregated over all three services: 15 good nodes, 5 bad nodes, fully connected topology.

The system works very well after trust has converged. When trust has been properly established, even with 25% of nodes behaving maliciously, all attempts by good nodes to gain access are allowed and the bad nodes are blocked. It is evident from the plateaus in Figure 6 how the good nodes initially can mostly access just service A, then both A and B, and finally all three services.

Influence of network topology & neighbourhood size

In the case described above, the neighbourhood of a node is defined as containing *every other* node (i.e. a fully connected network). Ad hoc networks are of course often less well connected and thus it is useful to study topological effects.

Figure 7 shows the effects of topology on service access adjudication. In the topology used for

this simulation, nodes are randomly distributed on a plane and connectedness depends on distance. Again, five of the nodes are “bad” and the remaining nodes are “good”. The main impact of this relatively sparse topology is that some nodes are more significantly impacted by the behaviour of bad nodes. The success rate for access attempts by good nodes does not quite reach 100%, due mainly to one good node being completely “hidden” behind a bad node.

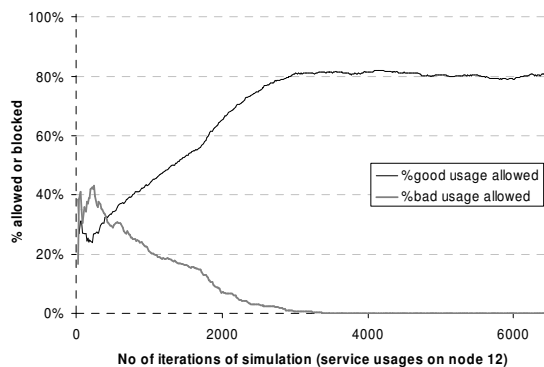


Figure 7. % good usage allowed and % bad usage allowed, aggregated over all three services: 15 good nodes, 5 bad nodes, sparser network topology.

Influence of mobility

For our initial simulations, we move just a single node and observe the influence on convergence and service access. The difference is most marked if moving the node brings it into or out of contact with the bad nodes. In the example shown in Figure 8, the node that was initially “hidden” (node 5) behind a bad node is slowly moved. Before this node moves, nothing can be done about its low

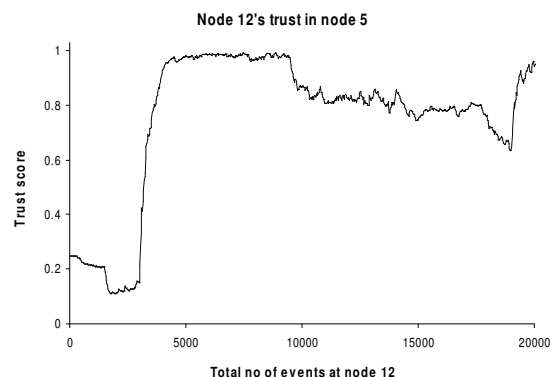


Figure 8. Node 5 is initially viewed by node 12 as untrustworthy as it has only bad recommendations (from node 19) to rely on. The situation improves as node 5 moves.

trust score as it is only accessible via a bad node. On moving though, it quickly gains trust by good behaviour and recommendations, though this falls off when it again comes in close contact with malicious nodes.

Experiment 4. Effects of collusion between bad nodes

Attacker interaction with the recommendations system is of particular interest, especially collusion between bad nodes to attempt to artificially raise their trust scores. Figure 9 compares the success rate of bad nodes in attempting to gain access to services when they act independently with when they act in collusion. Without collusion, each bad node, as well as behaving badly when granted service access, sends out low trust recommendations about all nodes. We model collusion as where the bad nodes are aware of each other and send out high trust recommendations about each other and low ones about the other (good) nodes. Not surprisingly they are more effective at gaining access when actively colluding.

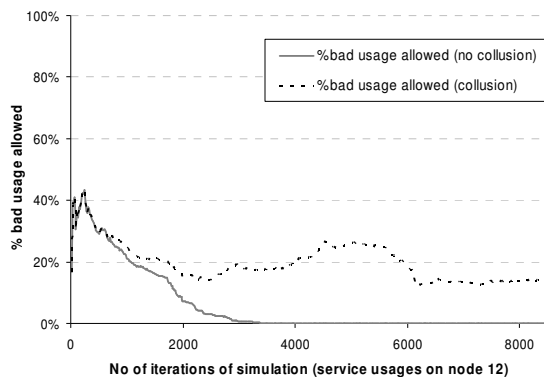


Figure 9. % bad usage allowed where nodes actively collude compared with the same measure where nodes behave badly but do not work together.

Other strategies are possible to attempt to break the system. Further work could examine the effect of a bad node acting benignly for a time in order to build up its reputation and gain access to more sensitive services than would otherwise have been possible.

6 Related work

Distributed trust and its application to security is a well-studied area, though many open issues exist and challenges remain. There are as yet few

widely used truly distributed trust systems. Several online marketplaces, social networks and review websites use reputation and ratings systems to give a measure of trust, but these mostly depend on some centralised storage and management. One example of a well-established and working distributed trust system is Pretty Good Privacy (PGP). The reader is referred to [9] for a comprehensive survey of trust and reputation systems.

Issues of trust and security in ad hoc and peer to peer networks have been under consideration for some time (e.g. [10]) and a fairly substantial body of published work exists.

Much existing work on securing ad hoc networks is focused on the protection of routing. Some of this work is focused on specific protocols. For example, the secure ad hoc on-demand distance vector (SAODV) routing protocol, initially proposed in [11], extends the AODV routing protocol to provide integrity and authentication. This and several other protocols (e.g. [12], [13]) rely on cryptographic methods that depend on centralised trust or some kind of key exchange system.

More recent work has taken a collaborative trust-based approach. Yang et al, for example, in [14] propose a collaborative solution that is applicable to a variety of routing and packet forwarding protocols. This system is reactive, dealing dynamically with malicious nodes.

Jensen and O’Connell [15] have also proposed trust-based route selection in dynamic source routing (DSR). Each router is assigned a trust score based on past experience, and the trustworthiness of a candidate path is a function of that of the routers that make up that path.

Several people have also examined aspects of trust and security in access to more general services in ad hoc and peer to peer networks (e.g. [16])

There has also been significant work in employing trust for mail filtering. Golbeck and Hendler [17] present a technique based on social networks for sharing reputation information among email users. This allows email users to sort received messages based on a rating value associated with the sender. This rating value is set by the user or is inferred from neighbours’ ratings using a weighted average technique. Kong et al [18] also focus on end user email addresses, but provide for the anonymous sharing of information with a wider community. When a user flags a mail as spam, this is made available to other users’

spam filters, which is useful as the same spam messages are usually sent to a large number of users. Han et al [19] also focus on users that generate spam, but more specifically blog spam, where undesirable comments are added.

Seigneur et al [20] use "hard" cryptographic authentication to support whitelists of known good guys in conjunction with "soft" trust management for new contacts. Trust derives from stored evidence, which includes recommendations, observations, certificates and reputations. Though our trust model is similar to this ("observations" are called "experience" in our model, and we consider certificates and reputations to be types of recommendations), our work uses this information for filter tuning and investigating the dynamics of the resulting system. Also, we focus on mail domains rather than individual email users.

7 Conclusions and further work

We have described a service-oriented approach to dynamic refinement of security enforcement in this paper. This is based on a closed loop feedback system where live distributed trust measures are used to adapt access control settings in a changing threat environment. A general trust overlay architecture and model are presented. Some specific application scenarios are discussed, in particular spam filtering and distributed intrusion detection in mobile ad hoc networks. It is shown in simulations of specific scenarios that this dynamic system is robust in these cases and has the potential to enhance security and access control efficiency.

Though initial results are interesting, significant further work is required. Our simulations to date have just used exponential averaging to update trust scores and a simple random technique for sharing trust information. There is significant scope to refine the dynamics of the system. New algorithms need to be developed and evaluated for trust updates. Further experiments are needed to explore the effects of how node neighbourhoods are defined. Other application scenarios need to be considered.

A game theoretical approach may be useful to examine possible strategies of both good and bad nodes. It needs to be assessed how nodes can be given an incentive to use this kind of system. As it stands, nodes that are otherwise good can be self-

ish, just receiving and never sharing trust information. Further possible collusion strategies of bad nodes need to be examined.

Acknowledgements

The authors' work is supported by the European Commission FP6 project *OPAALS* and by Science Foundation Ireland (*Foundations of Autonomics* project).

About the Authors

Jimmy McGibney is a lecturer in computer security at Waterford Institute of Technology. His main research interests are in distributed intrusion detection and trust models. He has also worked as a software developer in the telecoms industry. His email address is jmcgibney@tssg.org.

Dr. Dmitri Botvich is a principal investigator with the TSSG at Waterford Institute of Technology. His main current research interests are in the principles, architecture and methods to support autonomous management in future networks. His email address is dbotvich@tssg.org.

References

- [1] J. McGibney, N. Schmidt, and A. Patel, "A service-centric model for intrusion detection in next-generation networks," *Computer Standards & Interfaces*, pp 513-520, June 2005.
- [2] D. Gambetta, "Can we trust trust?" D. Gambetta (Ed.), *Trust: making and breaking cooperative relations*, pp 213-237, Blackwell, 1988.
- [3] J. Douceur, "The Sybil attack," *Proc. International Workshop on Peer-to-Peer Systems*, Cambridge, MA, March 2002.
- [4] J. McGibney and D. Botvich, "A trust overlay architecture and protocol for enhanced protection against spam", *Proc. 2nd International Conference on Availability, Reliability and Security*, Vienna, April 2007.
- [5] M. Dam and R. Stadler, "A generic protocol for network state aggregation", *Proc. Radio Science and Communication Conference (RVK)*, Linköping, Sweden, June 2005.

- [6] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *DNS Security Introduction and Requirements*, Internet Engineering Task Force, RFC 4033, March 2005.
- [7] S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol", *IEEE Journal on Selected Areas in Communications*, pp 582-592, April 2000.
- [8] A. Schwartz, *SpamAssassin*, O'Reilly, 2004
- [9] A. Jøsang, R. Ismail, and C. Boyd, "A survey of trust and reputation systems for online service provision," *Decision Support Systems*, pp 618-644, March 2007.
- [10] L. Zhou and Z. Haas, "Securing ad hoc networks," *IEEE Network*, Nov./Dec. 1999.
- [11] M. Guerrero Zapata and N. Asokan, "Securing ad hoc routing protocols," *Proc. ACM Workshop on Wireless Security (WiSe)*, Atlanta, September 2002.
- [12] Y. Hu, D. Johnson, and A. Perrig, "SEAD: Secure efficient distance vector routing for mobile wireless ad hoc networks," *Proc. IEEE Workshop on Mobile Computing Systems and Applications*, Callicoon, NY, June 2002.
- [13] P. Papadimitratos and Z. Haas, "Secure routing for mobile ad hoc networks," *Proc. Communication Networks and Distributed Systems Modeling and Simulation Conference*, San Antonio, January 2002.
- [14] H. Yang, J. Shu, X. Meng, and S. Lu, "SCAN: self-organized network-layer security in mobile ad hoc networks," *IEEE Journal of Selected Areas in Communications*, February 2006.
- [15] C. Jensen and P. O'Connell, "Trust-based route selection in dynamic source routing," *Proc. 4th International Conference on Trust Management*, LNCS 3986, Pisa, May 2006.
- [16] V. Cahill, et al., "Using trust for secure collaboration in uncertain environments," *IEEE Pervasive Computing*, July-September 2003.
- [17] J. Golbeck and J. Hendler, "Reputation Network Analysis for Email Filtering," *Proc. 1st Conference on Email and Anti-Spam (CEAS)*, Mountain View, CA, July 2004.
- [18] J. Kong, B. Rezaei, N. Sarshar, V. Roychowdhury, and P. Oscar Boykin, "Collaborative Spam Filtering Using E-Mail Networks," *IEEE Computer*, August 2006.
- [19] S. Han, Y. Ahn, S. Moon, and H. Jeong, "Collaborative Blog Spam Filtering Using Adaptive Percolation Search", *Proc. 15th International World Wide Web Conference*, Edinburgh, May 2006.
- [20] J.-M. Seigneur, N. Dimmock, C. Bryce and C. D. Jensen, "Combating Spam with TEA (Trustworthy Email Addresses)", *Proc. 2nd Annual Conf. on Privacy, Security and Trust*, Fredericton, Canada, October 2004.