10.3745/JIPS.2008.3.2.043

Addressing Mobile Agent Security through Agent Collaboration

Evens Jean*, Yu Jiao, and Ali R. Hurson***

Abstract: The use of agent paradigm in today's applications is hampered by the security concerns of agents and hosts alike. The agents require the presence of a secure and trusted execution environment; while hosts aim at preventing the execution of potentially malicious code. In general, hosts support the migration of agents through the provision of an agent server and managing the activities of arriving agents on the host. Numerous studies have been conducted to address the security concerns present in the mobile agent paradigm with a strong focus on the theoretical aspect of the problem. Various proposals in Intrusion Detection Systems aim at securing hosts in traditional client-server execution environments. The use of such proposals to address the security of agent hosts is not desirable since migrating agents typically execute on hosts as a separate thread of the agent server process. Agent servers are open to the execution of virtually any migrating agent; thus the intent or tasks of such agents cannot be known a priori. It is also conceivable that migrating agents may wish to hide their intentions from agent servers.
In light of these observations, this work attempts to bridge the gap from theory to practice by analyzing the security mechanisms available in Aglet. We lay the foundation for implementation of application specific protocols dotted with access control, secured communication and ability to detect tampering of agent data. As agents exists in a distributed environment, our proposal also introduces a novel security framework to address the security concerns of hosts through collaboration and pattern matching even in the presence of differing views of the system. The introduced framework has been implemented on the Aglet platform and evaluated in terms of accuracy, false positive, and false negative rates along with its performance strain on the system.

Keywords: *Disk Striping, Multimedia System, Bandwidth*

1. Introduction

Mobile Agent refers to the ability for a program to halt its execution, move to a new environment where execution can then be resumed. Even with the development of numerous mobile agent platforms such as Aglet [18], an open source system originally released by IBM, the use of mobile agents have not transcended from theoretical to practical applications due to the numerous security threats plaguing the paradigm. The security threats facing mobile agents, including Aglets, have been studied in depth and categorized into host-to-agent and agent-to-host [9]. Solutions to such threats have to this point only been introduced from a theoretical perspective. In order to foster the emergence of mobile agents to address practical issues, we conducted an analysis of the security options available in the Aglet platform. The study resulted in the introduction of a Secured Aglet Server (SAS) providing:

- Secured communication,
- Controlled resource consumption of agents, and
- Integrity and reliability of agent's data.

The introduction of SAS attacks the issue of securing the agent paradigm from a centralized standpoint. It is fair to note that any mobile agent system is inherently suitable to support distributed applications; hence, securing such systems need to take into account the distributed nature of the environment. Malicious agents are not a threat solely to the current execution environment but to any host to which they may migrate. As such, there is an inherent need for hosts to collaborate and learn from each other's experience in executing an agent's code. The Computer Security Division of the National Institute of Standards and Technology has also suggested that one of the main hindrances to the adoption of mobile agent technology stems from the security concerns of hosts [12]. Thus, the protection of hosts needs to occur from both a centralized and distributed standpoints. In light of this observation, we herein introduce a boosting-based monitoring system that allows hosts to learn and classify agents, not only based on their own experiences, but also based on collaboration with other hosts in the network, some of which may have been victims to agent attacks.

The issue of protecting a resource in a distributed environment is not novel; in fact, it has been thoroughly explored in the literature under the banner of Intrusion Detection Systems (IDS). In general, IDS attempt to detect

Manuscript received December 6, 2007; accepted December 26, 2007.
The National Science Foundation under the contract IIS-0324835 in part has supported this work.

Corresponding Author: Ali R. Hurson(hurson@mst.edu)
* Computer Science & Engineering, The Pennsylvania State University, University Park, PA 16802
** Computational Sciences and Engineering Division, Oak Ridge National Laboratory Oak Ridge, TN 37831

malicious activities of users in a system in order to maintain the system's integrity. The use of IDS should help in protecting hosts against misbehaving agents. However, due to the fact that agent servers, or hosts, in agent platforms are unlike traditional systems, the application of IDS cannot adequately address the security threats of such servers. Migrating agents execute as independent threads of the agent server process, thus limiting the ability of present day IDS systems to determine the identity of misbehaving agents. As an open system, the range of execution patterns of migrating agents in the paradigm is essentially unlimited. Through our proposal of having hosts collaborate based on their experience with specific agent applications, we contend that protection of hosts in the distributed environment can be rendered more efficient. Within the proposed framework, security agents collaborate to learn from each other's experience in dealing with any migrating agent. Our contribution can thus be summarized as follows:

- Collaboration between hosts to identify malicious agents, and
- Ability for the security agents of hosts to learn from experience and thus prevent attacks.

In further detailing our work, we will start by introducing the necessary background in section 2. Section 3 will then discuss our work in securing Tahiti, the Aglet server. The adaptive security-monitoring framework is introduced in section 4. We will then proceed to evaluating the proposed framework and draw conclusions highlighting our contribution and future work in sections 5 and 6, respectively.

2. Background

To ensure that the reader gains an in depth appreciation of our proposal, we find it imperative to provide a thorough coverage of agent security, intrusion detection, and supervised learning along with other pertinent works related to our proposal.

2.1 Agent Security

Mobile agents lend themselves nicely to searches and computation that requires parallel and distributed processing, as well as network roaming. The mobility of mobile agents may depend on predetermined itinerary or intermediate computation results. Along with flexibility in system design, agent mobility also introduces security concerns. The categorization of the threats plaguing mobile agents is done based on the origination of the attack; as such we have agent-to-host, as well as host-to-agent attacks. The security issues in mobile agents have been studied and some of the proposed solutions include but are not limited to the following:

- Code signing, access control, proof carrying code, and path histories to protect the hosts [7, 9, 21],
- Tracing, obfuscation, trusted hardware as well as encrypted functions and data to protect the mobile agents [2, 7, 9, 21].

Research in mobile agent security is still an open field, and many of these approaches remain theoretical at best. Moreover, most of these proposals further suffer from reliance on an isolated view of agent systems.

2.2 Intrusion Detection Systems

Intrusion Detection Systems (IDS) refer to the detection and reporting of suspicious behaviors in a network with the purpose of identifying intruders and thus securing the system. Such techniques can be categorized based on the approaches taken to accomplish their goals, thus, we have anomaly detection and misuse detection approaches [5, 19, 23]. Misuse detection leaves the specification of abnormalities to the administrators of the system. An exhaustive definition of such abnormalities is very difficult to achieve and even more so in agent systems where the intent or tasks of agents cannot be known a priori. On the other hand, anomaly detection relies on the system's training to learn the expected behaviors of users and is thus able to detect novel forms of attacks and represent the approach adopted in this article to address agent security. Dotted with the ability to learn from novel attacks, anomaly detection is an interesting approach to protecting hosts, as precise prior knowledge of agent's intents and tasks are not necessary. In general, current proposals in IDS suffer from the inability to detect precisely which agent is responsible for an attack as the agents are typically independent threads of execution of the hosting agent server process. Applying anomaly detection to securing agent hosts would thus require an IDS system that is intimate with the agent platform in use and able to detect anomalies at fine granularities.

The agent paradigm has inevitably found its use in IDS [5, 19]. Deeter et al. aim at achieving bandwidth performance, scalability, minimizing analysis delay as well as allowing integration with new IDS systems without the need for global change [5]. As such, the mobile agent paradigm was adopted to establish a middle-ware layer allowing for different IDS to collaborate and secure a system. The mobile agents migrate to sources of potential attacks and determine whether to raise an alarm or not through analysis of data collected by various IDS. The mobility of agents adds virtualization and serialization overhead to the performance of the IDS architecture [5]. The system's agents could also potentially be used to carry out a denial of service attack on hosts, as the introduced IDS architecture was not designed to address agent security [5].

2.3 Supervised Learning

Supervised learning focuses on the ability to extract patterns from a set of raw data whose categories are known. Various algorithms have been introduced to allow extraction of existing patterns in a data set. Such algorithms include Support Vector Machines (SVM), neural networks, decision trees, as well as boosting [8, 10, 20]. Boosting has an interesting property, in the fact that training occurs in stages. In each stage of boosting, a weak classifier is trained using a subset of the raw data. The set of trained classifiers yield the learning function used to determine how to categorize future data samples. As the boosting learning function emanates from several weak classifiers, it is easily adaptable to a distributed environment where each weak classifier may operate from different sources. It is this inherent ability of boosting that we attempt to harness in addressing the issue of identifying malicious agents operating across several hosts.

2.4 Related Works

Agent collaboration has been the focus of various research efforts in recent years. Becker et al. [1] studied the issue of confidence determination to ascertain its effect in collaborative agent systems. The study showed that, in a multi-agent system, incorrect confidence-integration might propagate and thus change the collaborative answers of the agents. The problem was simplified by assuming that trust is not an issue between the collaborating agents. Within our approach, each of the collaborating agents is extremely flexible in integrating confidence factors to yield a collaborative result. The collaborating agents do take trust into account along with confidence in determining their results to provide distributed security.

Chen et al. [3] presented a boosting-based hierarchical learning algorithm for experience classification. The work was motivated by the need for agents within a team to collaborate and learn from their past experiences, which may differ from one agent to another, as individual agents may only have a partial view of the team's environment. The learning algorithm [3] attempts to take advantage of boosting by building a hierarchical framework where agents at the lowest level may only have a partial view of the system. Agents at the lowest level are trained using decision stumps based only on the feature set available to them. On the other hand, agents higher up in the hierarchy are trained, not based on their observations, but using the classification results of the corresponding agents at the lower level. Training is hierarchical and tightly coupled amongst agents as the classifiers are inter-dependent. The hierarchical learning system is not suitable to address security concerns that we have discussed as the system is built upon the assumptions that the agents are members of the same team, thus ignoring any trust issues. The fact that the system is built in a hierarchical fashion means that the final decision must originate from the root of the structure if it is to take into account the experience of every possible agent involved. The latter also means that such a system would be highly unsuitable for distributed security-monitoring since it would introduce a single point of failure, namely the root, in properly classifying agents.

3. Securing the Aglet Platform

The security threats facing the agent paradigm are extensive and hinder the development of agent-based applications. As a result, we will study agent security by focusing on one of the available agent platforms to help secure the paradigm's execution environment. Aglet is one of the numerous platforms introduced to support agent development; a pictorial representation of the platform's execution environment is depicted in figure 1.

Aglet is a library written in Java, released by IBM to support the development of mobile code; the platform is fairly documented, easy to install, has received great press coverage, and is currently maintained by the open-source community [14]. Numerous application prototypes have already been introduced on the platform [14, 15, 16], thereby making it a suitable choice in analyzing the security infrastructure in place to support secure agent applications. We have discussed the general classification of security threats to agent systems as being agent-to-host or host-to-agent. As thus, our intent is to analyze the security framework in place in Tahiti, the Aglet server, to ascertain that both entities are properly secured. Our study has revealed the following vulnerabilities in the Aglet platform [14]:

- Communication vulnerability established through the use of the Dsniff package to intercept agents as they are being transmitted from one host to another. This has led us to conclude that the Aglet framework cannot satisfy the requirement of agents to migrate exclusively to intended hosts.

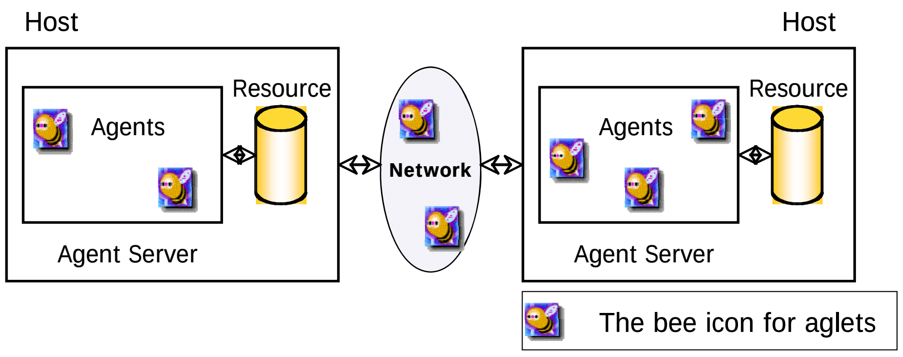

Fig. 1. Aglet Execution Environment

- Data vulnerability has not been addressed in the Aglet framework. While there is no acclaimed solution to the issue, it is imperative that users be able to determine if their data has been tampered with, and determine the malicious host.
- Resource vulnerability determined through analysis of the lifecycle of agents in the environment. Through seemingly normal transition of lifecycle states (see Figure 2), agents can wreak havoc in hosts through repeated state transitions such as cloning.

Details on the aforementioned studies can be found in [14]. Based on our experimental results, the task of addressing agent security resulted in the introduction of a new server namely Secure Aglet Server (SAS). The following subsections detail our contribution in addressing agent security in the Aglet platform that has led to SAS.

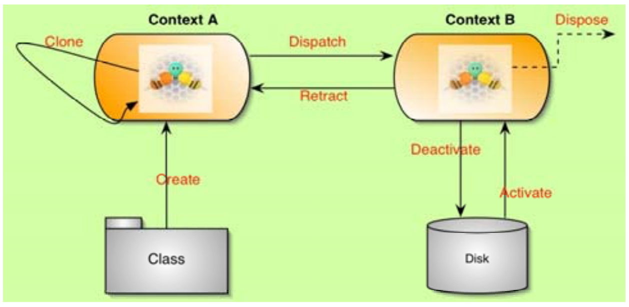

Fig. 2. Lifecycle states of Aglets

3.1 Securing the Communication Channels

The Communication Layer of Aglets makes use of an unsecured protocol, thus leaving the framework open to a range of attacks, which we intended to address in our research. SSL is the current industry standard in addressing communication security. Keeping in line with our intent of fostering the adoption of agents in commercial applications, we opted to implement SSL in Tahiti, using the Java Secure Socket Extension (JSSE). Through the use of SSL sockets, we have endowed the Aglet framework with the ability to communicate over secure channels capable of authenticating the parties involved, refusing unsecured connections and adjusting the security level on the channels. Through the availability of SSL in SAS, administrators will gain control of the level of security enforced on network links; most importantly, it provides a standard solution trusted in the industry to handle secure communication.

3.2 Securing Agent's Data

Granting Aglets the ability to detect tampering with their data required that we extend the functionalities of the server. Consequently, the Runtime Layer of SAS was extended to support the creation of Message Digests using the Java Cryptography Extension (JCE) as well as the ability to digitally sign objects. We provided Aglets with a java class library that implements the concept of Read-Only data. With the new functionalities of SAS in place, the library obtains a signed copy of the message digest computed by the host along with the host's certificate. An Aglet can retrieve the message digests stored by the library and use the corresponding certificate to ensure that its data has not been tampered with. The introduction of computed message digests and digital signatures in the Runtime Layer of SAS provides Aglets with the capability of detecting active malicious hosts in the agent's itinerary.

3.3 Securing Hosts

Securing the hosts from a centralized aspect meant dealing with the possibility of an Aglet overusing the resources of a host through seemingly normal transition between its lifecycle states (Figure 2). As a result, we needed a scheme to not only specify and track the resources in use by an Aglet but also to take proper actions once an Aglet attempts to overuse the host's resources. The design of such a scheme led us to the introduction of a MonitorAglet in SAS. The MonitorAglet tracks the number of instances of an Aglet, based on the Aglet's properties such as the corresponding ID, to ensure that the specified limit is never exceeded. Within the scope of SAS, we defined instances as the instantiation of an Aglet or Message object.

Definition 1: An Aglet *B* is an instance of an Aglet *A* if and only if one of the following is true:

- *B* belongs to the same resource object as *A*,
- *A* has created, retracted, or activated *B*, and
- *B* is a clone of *A*.

Once an Aglet has reached its instance limit, the MonitorAglet prevents the creation, cloning, activation, or retraction of any other instances of the Aglets in the system until one of the instances has been deactivated, dispatched, or disposed of. As we attempt to protect hosts against malicious agents, we have introduced powerful capabilities to the Aglet framework. It is now possible to limit the number of instances of an Aglet executing on a host, thereby preventing the aforementioned attack.

In addressing centralized agent security, we have introduced a new agent server, namely SAS, endowed with secured communication, ability to detect tampering of agent's data along with prevention of over-usage of host's resources. The interested reader is referred to [14] for further details on the aforementioned schemes. Note that the introduced security mechanism to protect host is only an extra layer of protection. While effectively addressing the security issues of hosts, SAS merely reacts to malicious agents attempting denial of service (DoS) attacks. The malicious agent itself is never destroyed and the occurring attack is thwarted by controlling the resources in use by instances of the attacking agent. The system does not take into account the fact that a misbehaving agent may travel from one host to another and repeat its actions. A malicious agent that attacks one host is very likely to migrate and attack another host in the near future. As SAS only controls the number of instances of an agent, a malicious entity could abuse its privileges and migrate to another host once it has reached its instance limit. Such a malicious entity could indeed migrate over numerous hosts in a domain and effectively wreak havoc. Improving the security of SAS further requires:

- Collaboration between hosts to identify malicious agents.
- Ability for the MonitorAglet of hosts to learn from experience and thus prevent attacks instead of merely reacting to such occurrences.

4. Distributed and Adaptive Security-Monitoring Through Agent Collaboration

Agents interact in a distributed environment; hence, similarly, agent security needs to be validated in a distributed manner. As hosts monitor agents, data regarding the actions of the agent can be recorded. Our work is based on the assumption that there is a relationship, though not clearly defined, between the actions of an agent and the intent of such agent; whether the intent is malicious or not. The definition of the set of actions that can help determine whether an agent is malicious will vary from one host to another and such actions are herein referred to as threatening actions. The consistent fact will remain however, that a malicious agent on one host is highly likely to represent a threat to the security of future hosts. Due to the variation in what constitutes a malicious agent, any proposed learning scheme must allow for such flexibility in identifying potential threats.

Our approach in tackling the problem is through the introduction of a variation of the Boosting-learning algorithm, namely a Distributed and Adaptive Security-Monitoring through Agent Collaboration (DASAC). To determine whether an agent is malicious, DASAC relies on collaboration between the current host and past hosts visited by the agent. The current host acts as a decision maker; every hosts including the current one act as base learners. We attain the required flexibility by allowing each host in the system, as base learners, to be trained independently and based on different feature sets. A discussion of what feature sets could possibly be used is deferred at this point and discussed later (section 5). The base learners in DASAC are trained as follows:

- Implement a binary classifier, which can be a decision tree or any other classifier, where 1 is the class of malicious agent and -1 otherwise.
- Train the classifier using a sample data set with the threatening actions against the host as the various features of each training instance.

Note that each host in the system may serve as a base learner and as a decision maker depending upon its contribution to the current decision-making process. The base learners, being trained independently, may implement various classifiers depending on the host's administrator.

Upon arrival of an agent to a host, one of two cases may be true. The host may be seeing the agent for the first time or the host may have had a personal experience with the agent. In either of these two cases, the host needs to determine whether to allow the agent to execute or not. If the host had no prior experience with the agent in question, it does not have any pertinent information about the agent to classify it as malicious or not using its base learner. It must thus rely on the hosts that the agent has visited in the past. If the agent had in the past executed on the host, the host's base learner can classify the agent.

Within DASAC, classification of an agent by the decision maker is based on the following steps:

- If the host has had prior experience with the agent, the base learner of the host is used to classify the agent; else, the agent is assigned the default value of 0.
- The classification of the agent from every host in the agent's history as determined by their respective base learners are collected by the decision maker.
- Using the possibly diverse experiences of other hosts, the decision maker determines whether to allow an agent to execute or not.

In essence, a Decision Maker (DM) interacts with the various base learners of the hosts in the distributed environment to thwart attacks. In the final steps, a DM could use techniques such as majority-vote to reach a consensus. We however recommend a version of weighted sum tailored to the problem at hand as specified in Equation 1 where Ψ_i represents the class to which an agent has been assigned by the base learner of a host. We allow Ψ_i to possibly have a value of 0 to ignore a base learner that does not have any information on the agent as such may be the case for the learner on the current host. Furthermore, τ_i and λ_i, represent the trust, and confidence levels, respectively, associated with each host being contacted.

$$x = \begin{cases} 1 & \forall \quad \sum_{i=0}^{n} \tau_i * \lambda_i * \Psi_i \quad > 0 \\ -1 & \forall \quad \sum_{i=0}^{n} \tau_i * \lambda_i * \Psi_i \quad < 0 \\ 0 & \mathit{Otherwise} \end{cases}$$

Equation 1. Trust and Confidence based weighted sum

The recommended version of majority vote derives from our observations of the underlying mechanisms in inter-human collaboration. Consider the case where a person, *A*, asks a friend, *B*, for his/her opinion on a puzzling question; *A* does not blindly believe *B*'s assertion. Instead, *A* weighs his/her opinion and confidence on the topic with *B*'s recommendation based on two factors; namely, how much does *A* trust *B* and how confident is *B* in his/her assertion. The confidence level, in the proposed majority vote scheme, is determined by the accuracy of the classifier used in a host and varies between 0 and 100. The confidence of a host is communicated to the DM along with the classification of an agent.

The trust level, on the other hand, can be defined statically by the system administrator of a host based on the reputation of a particular host. We propose trust levels to be defined as a value between 0 and 10. A default value can be specified for use whenever a remote host's trust information is not available. Notice that setting the default value of trust to 0 would effectively allow the monitoring system to not take into account the experience/ classification of unknown hosts. As the definition of trust levels does not carry over from one host to another, administrators are free in setting the limits of trust values in their systems.

If an agent is allowed to execute in the system, the decision maker keeps track of the actions of the agent. It can then periodically attempt to re-classify the agent and thus adapt to agents that may execute malicious code only on specific hosts. The frequency upon which to re-classify an agent is left as an implementation detail as it will vary upon the requirements of a host.

$$\Delta = n * \left[\max(\tau_i) * \max(\lambda_i) * \max(\Psi_i)\right]$$

$$SL = \left\lceil \beta * \frac{\sum_{i=0}^{n} \tau_i * \lambda_i * \Psi_i}{\Delta} \right\rceil$$

Equation 2. Security Level of an agent

Although DASAC, as described, can be made to be completely autonomous, except during training, we understand that administrators may need to have hands-on control on whether or not an agent should be allowed to continue or start execution. To cope with such a need, we introduce the notion of Security Levels (SL) of agents on a host. The SL of an agent is defined (Equation 2) as the ceiling of the product of its weighted-sum, as computed in Equation 1, and the number of security levels in the system (β). The result is divided by Δ, representing the maximum sum of products multiplied by the number of cooperating hosts. Note that Δ is always greater than 0 as n takes into account the current host as well. While the SL could be calculated for all possible value of the weighted-sum, one should note that it is not of importance when the weighted-sum is 0 or less as such agents have not been classified as malicious.

Using the SL, the system can be made to be semi-autonomous, requiring human assistance once a threshold has been reached. Agent-human interaction can further increase the efficiency of the system as the agent can be made to adjust its classifier based on such interactions. Thus, DASAC may decide to use the collected data about an agent, classify it based on its interaction with an administrator, and inserts the information in the pool of training data. The classifier can be periodically retrained thereby leading to an adaptive security system.

Evaluation of our proposal requires its implementation in one of the numerous agent platforms that have been thus far advanced in the literature. Addressing the security of the hosts further requires that an adequate security mechanism be in place to protect the agents as well. As we have introduced a secured server for the Aglet platform, the implementation of DASAC on SAS was the logical choice. The following section details our work in implementing DASAC on the chosen server.

5. DASAC Implementation on SAS

We have previously addressed the motivation behind the presence of the MonitorAglet within SAS (i.e., controlling the amount of resources being used by any particular agent). In implementing the DASAC scheme into SAS, the use of the MonitorAglet as the DM of a host was an obvious option, which we adopted. The base learner on the other hand is implemented as an independent agent for flexibility and performance. The implementation of the base learner as an agent allows for separate threads of execution to handle classification and training. Moreover, as agent entities, a host may have multiple base learners trained using independent feature sets; however within our implementation of DASAC we assumed that each host has one base learner. All base learners in our implementation are built using the classifiers from the weka data-mining library [22]. Figure 3 shows a pictorial representation of the interaction between DMs and base learners within SAS. The dotted lines in the figure denotes the trajectory of the agent migrating from one host to another; the solid lines depict the communication between the DM and the base learners in order to decide whether to allow the arriving agent to execute. In a nutshell, using the MonitorAglet as the DM of DASAC with other agents acting as base learners extended SAS. The implemented version of DASAC makes use of the suggested version of weighted sum (equation 2) to classify agents. Moreover, we employed the notion of security levels which are used to determine whether interaction with a system administrator is needed to dispose of an agent classified as malicious. Five distinct security levels were defined:

- The MonitorAglet is augmented with the capability of requesting interaction from a system administrator to determine the appropriate set of actions to undertake once an agent has been identified as a malicious entity below or at level 3.
- Malicious agent of levels 4 and 5 are automatically denied execution.

The choice of level 3 is an arbitrary cutoff point that can be tuned by the system administrator of a host.

The set of features selected to train the base learners need to correlate in a manner that differentiates malicious entities from non-malicious ones in the system. Within SAS, agents are monitored and based upon the number of instances running, the MonitorAglet can allow or reject a requested action such as cloning, dispatching etc. Such actions of agents have been shown [14] to be potentially detrimental to the security of a system and thus constitute

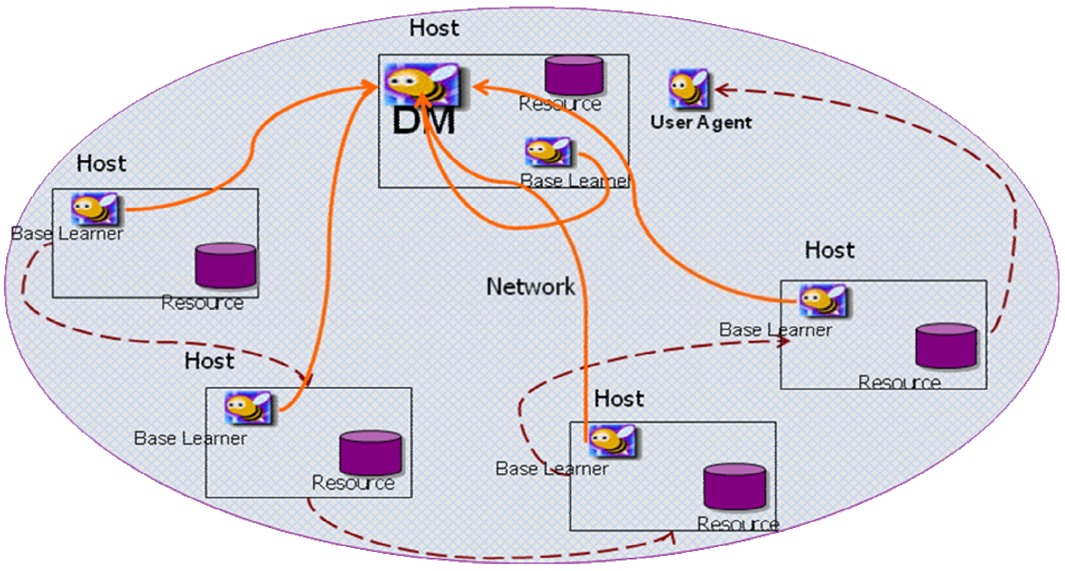

Fig. 3. Overview of DASAC implementation in SAS

good candidates for inclusion in the feature set. It is our belief that the frequency at which an agent requests the right to transition from one state to another, along with the total amount of time spent on the host, can help in identifying malicious actions. Due to Java's sandboxing techniques, a security manager prevents access to entities lacking the proper access rights to local resources. The malicious agents may attempt to access various resources in the hope that the security policy in effect is not well defined. As a result, we take into account the frequency of security exceptions generated by an agent. The security manager resides within the Runtime layer of SAS and in order to track the security exceptions generated by an aglet, we introduced a listener class through which interested parties can be notified whenever such events occur.

To sum up, we selected the features in table 1 to train the classifiers based on CPU usage times, lifecycle state transitions, and access to low-level system resources. Once the feature set has been selected, we trained the classifiers to extract data patterns that may help classify an agent based on its behavior. We used a system consisting of 5 hosts. Hosts 1 thru 5 have different classifiers, namely, an alternating decision tree with 0 boosting iterations, a fast decision tree learner, a decision stump, a naïve Bayes classifier and an alternating decision tree with 3 boosting iterations respectively. The implementations of the classifiers used are from the weka library [22]. Moreover, the hosts in the experiment have different trust levels.

Due to the fact that training data for classifiers are not readily available, we trained our classifiers using a data set generated by tracking the features of interest during actual runs of the following agent-based applications:

- MAMDAS [16]
- Private Information Retrieval prototype [14]
- Instance and Message DoS attack generator agents [14].

The choice of applications used to collect the data was based on their availability, along with the fact that their classifications were known a priori, as discussed in SAS [14]. The choice of applications encompasses both classes of agents, malicious and benign, that are of interest to our work. The MonitorAglet tracks every event generated by agents in the system and construct a data sample for the corresponding agent. The constructed sample consists of the features that we identified as having the potential to help identify malicious entities (see Table 1).

Table 1. Classification Features for Each Agent

Feature	Feature Description
Biased Running Time	Biased Length of Execution Time on System
Cloning Frequency	Ratio of number of cloning to average time between cloning
Activation Frequency	Ratio of number of activation to average time between activations
Dispatch Frequency	Ratio of number of dispatching to average time between dispatching
Retract Frequency	Ratio of number of retraction to average time between retractions
Arrival Frequency	Ratio of number of arrivals to average time between arrivals
Security Access Request Frequency	Ratio of number of security access requests to average time between such requests
Security Access Grant Frequency	Ratio of number of security access grants to average time between such grants
Security Access Denial Frequency	Ratio of number of security access denials to average time between such denials

The generated data set consists of over 3000 samples, manually classified, of which, roughly 15% is used to train each classifier. The classifiers used the remainder of the data set for accuracy assessment. The accuracy of each host classifier is used as the confidence level of the host in question, as suggested by DASAC, in all our experiments.

Figure 4 presents the average accuracy rate of the hosts in the system over multiple runs. The accuracy of hosts shown was determined based on the ability of the host's classifier to properly classify the remainder of the 3000 samples in the data set not used for training. As we noted earlier, the hosts use different classifiers resulting in different accuracy rates based on the strength of the classifier being used. As thus, Host1 exhibits the worst accuracy as it uses an alternating decision tree with 0 boosting iterations. Through the use of decision stumps, naïve Bayes classifier and alternating decision trees with numerous boosting iterations, the other hosts are able to achieve a higher accuracy rate.

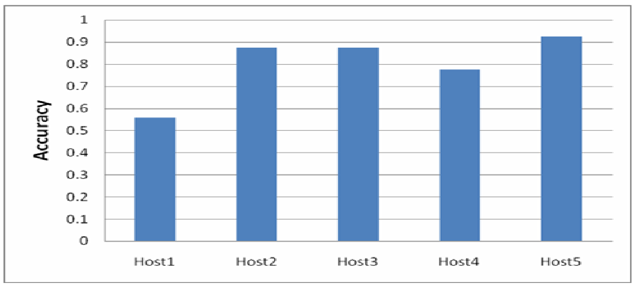

Fig. 4. Training Accuracy of hosts

6. System Evaluation

DASAC is evaluated based on several performance metrics such as accuracy (i.e., the ability of the framework to properly classify agents), the false and missed alarm rates and the time it takes to evaluate whether an agent is malicious. The following subsections details the results of our evaluation of the framework.

6.1 System Accuracy and The Impact of trust

To evaluate DASAC, the first set of experiment focused on measuring the accuracy of the system in identifying malicious agents as well as studying the impact of trust level on the accuracy. We chose the first host in the system (Host1 from Figure 4) as the local classifier against which DASAC will be compared. This choice was based on the fact that host1 showed a slightly better performance than random guessing. Our goal is to analyze how DASAC can help improve the performance of a host, through collaboration. We thus proceeded to launching the agent applications used in training the system classifiers. We also introduced a PortScannerAglet (malicious) which repeatedly attempts to connect to numerous ports in the system whether it has access to conduct such actions or not. Furthermore, we manually created, dispatched, and retracted each of the following benign agents: CirculateAglet, WebServerAglet and HelloAglet. These agents are available as part of the Aglet framework. Lastly, we created and used a MigratingWebServer agent that migrates to hosts and attempts to set up a server on random ports, restricted or not, repeatedly. The MigratingWebServer, a malicious agent, migrates to a new host, once it has been denied access to ports over 10 times. Our reasoning behind the introduction of new Aglets that were not used during the training phase is to gain insights into the ability of our classifiers to perform well even in the presence of previously unseen behaviors.

The DM of each host dynamically classifies agents to determine whether or not they are malicious. We evaluated the system based on the accuracy at which the local classifier and DASAC recognize malicious agents. We also tracked the best accuracy recorded in the system and computed the average accuracy of the hosts including Host1. While the confidence levels used in the experiments were as described in section 3, the trust levels, on the other hand, were assigned randomly. The local classifier is assigned a trust level of 9, close to the maximum of 10, to reflect the trust that we expect administrators to have in their own systems.

Figure 5 depicts the measured accuracy of DASAC compared to that of Host1's classifier, the best accuracy rate measured in the system, and the average accuracy rate of hosts in the experiment. On average, one can conclude that DASAC's performance lies between the average accuracy of the involved hosts and that of the most accurate classifier. The fluctuations in the accuracy rate measured are due to the fact that the hosts are trained for every experimental run on a random sample. Thus, their performance is slightly dependent on the sample used during training.

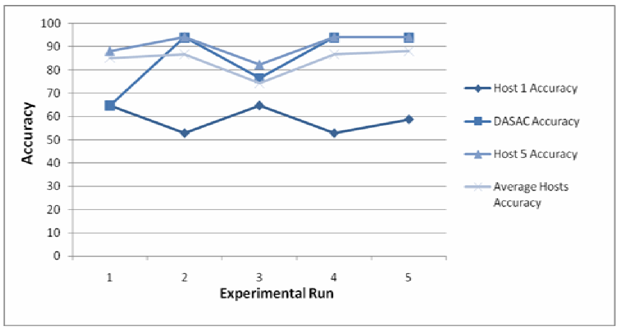

Fig. 5. Local vs. DASAC accuracy

While DASAC generally outperforms the worst classifier in the system, it matched the first host's performance during the first run of the experiment. The only reasonable explanation for such a poor performance by DASAC is the trust levels used during the first experiment. We noted that the total trust levels of hosts 2 thru 5 varied from one experiment to the next as follows: 5, 16, 23, 27, and 36. When the trust levels of the other hosts are low compared to that of the local host, DASAC's performance seems to be more dependent on that of the local classifier. This brings us to the second set of experiments that were carried out to further investigate the effect of trust levels on DASAC. During the second experiment, we kept the trust levels of all hosts, including host1, identical. The trust levels were however varied from one experimental run to the next starting at 0 up to 10.

The results of the second experiment are depicted in Figure 6. The figure shows that DASAC still outperforms the average accuracy rate computed. The experiment revealed two crucial points; Firstly, all the classifiers in the system can indeed outperform DASAC, as is the case when the trust levels are 0. The explanation behind such an occurrence is due to the fact that DASAC will classify all samples as 0, which is in effect non-malicious. Thus, DASAC will fail to classify any malicious agents possibly degrading to an accuracy rate of 0. The second interesting point is the fact that DASAC's accuracy seems to quickly become dependent upon the accuracy rates of the best classifiers in the system. This is justified by the fact that DASAC is in effect designed to take advantage of the strength and experience of other hosts in the system. Once the trust levels are identical for all hosts, the only determining factor in classifying an agent becomes the confidence of hosts. The more confident hosts have a more significant weight on the system's classification of an entity. The experiment allowed us to assert the accuracy of the herein introduced framework and analyze the impact of the use of trust levels on the observed accuracy. The next step that we undertook was to analyze the response time of DASAC in determining whether to allow an agent to execute along with the false and missed alarm rates.

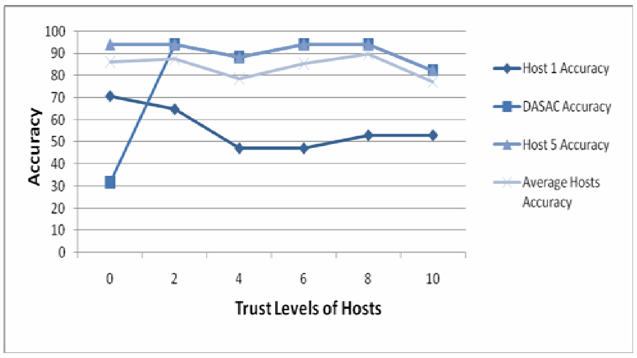

Fig. 6. Effect of trust levels on DASAC accuracy

6.2 Response Time, False and Missed Alarms Rate of DASAC

As noted earlier, determining the false and missed alarm rate of an IDS system is crucial in evaluating the system. Moreover, the time that it takes to determine the classification of an entity needs to be as low as possible, since a slow response time would allow attackers to quickly execute their malicious code and migrate to the next target. Keeping in line with such facts, the focus of this set of experiment was geared towards evaluating the accuracy, missed and false alarm rate of DASAC along with the response time of the framework. In setting up the experiment, we decided to use two hosts and collect the information of interest to our study. The two hosts used in the experiment are Host1 and Host2 from Figure 5 with training accuracy of 55.6% and 87.4%, respectively. We have also designed a new Aglet whose sole task is to execute on Host2 and repeatedly create agents and dispatch them to Host1. The intent behind such a setup is to study the response time of the framework when the agents are arriving and having been to one and only one host prior to the current one. Moreover, the experiment is intended to allow us to study the efficiency of the system when it is collaborating under the aforementioned conditions.

As we have mentioned, agents are created on Host2 and dispatched to Host1 (one-hop away). The agents that are being dispatched are the DoS attack generators noted earlier (section 5), the PortScannerAglet, and the MigratingWebServer along with the followings from the Aglet platform:

- examples.simple.DisplayAglet
- examples.hello.HelloAglet
- examples.itinerary.CirculateAglet
- examples.mdispatcher.HelloAglet
- examples.http.WebServerAglet
- examples.talk.TalkMaster

The choice of such agents was motivated by the fact that they constitute a fair representation of the types of agents that can exist in the system in terms of their intentions being malicious or not. The agent executing on Host2 continuously generates and dispatch the agents to Host1; from that location, we measure the accuracy, false and missed alarm rate, and the time it takes for Host1 to contact Host2 and determine whether to allow an arriving agent to execute. The key point here is that as DASAC is currently implemented, Host1 needs to formally deploy an Aglet to Host2 and collect the information necessary to classify any arriving agent. The result of the experiment is presented in Figure 7 showcasing the accuracy of DASAC on Host1, along with the false (FAR) and missed (MAR) alarm rates of the host. As expected, DASAC outperforms Host1 and draws closer to the accuracy of Host2 while the false and missed alarm rates decrease before settling.

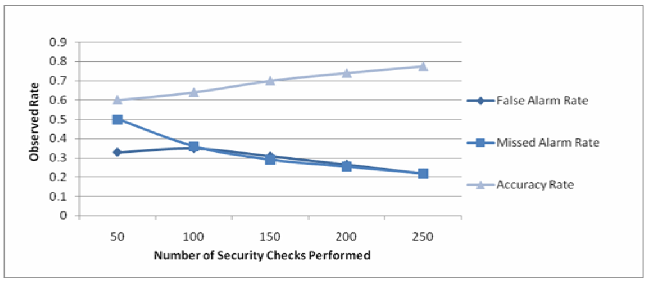

Fig. 7. False alarm rate, Miss alarm rate and Accuracy Rate

The average amount of time spent solely on contacting Host2 to classify an arriving agent was also measured. As expected, the ratio of time spent on communicating with Host2 compared to the total time required to classify arriving agents was drastic (see Figure 8).

The communication time alone represents over 90% of the time it takes to determine whether to allow an agent to execute. Such an observation is a direct result of the fact that the current implementation of DASAC has to dispatch an Aglet to past hosts and collect the required information to classify an agent. In order to avoid such a significant cost in classifying an agent, we slightly modified our implementation of the framework in DASAC. The modification occurred by not only attaching the address of the current host to agents being dispatched as is currently done, but also pertinent information that the framework requires such as the class to which the agent has been assigned prior to migrating from the current system, along with the confidence factor of originating host. Having altered our implementation of DASAC on Aglet we noted close to a 90% reduction in the response time of the Host1, as one would expect. Such a drastic reduction is due to the fact that Host1 no longer has to use network bandwidth to collaborate with Host2, nor does it have to wait for such information to be available before making a decision. With the changes in the framework's implementation, the average response time in classifying agents is effectively reduced.

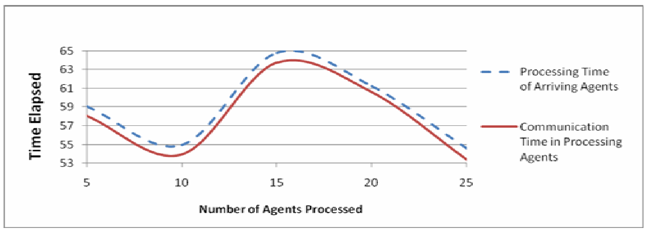

Fig. 8. Processing vs. Communication Time of Arriving Agents

7. Conclusion

This article has introduced a novel distributed and adaptive security-monitoring framework achieved through agent collaboration across multiple hosts. To the best of our knowledge, this work represents the first in its kind to attack agent security through collaboration between the hosts in the system. While we have only implemented DASAC within SAS at this point, it can be easily applied to any agent platform. The framework, as we have shown, builds on the idea of boosting to allow host protection by classifying agents based on their reputation. The system is flexible enough to support the incorporation of various classifiers that may be trained using independent variables, as the hosts do not communicate their feature sets to each other. Moreover, DASAC introduces the notion of security levels to support human-agent interaction in order to render the system even more flexible and robust. We have also secured the Aglet framework from a centralized aspect, providing secured communication, ability for agents to detect tampering of their data, and allowing hosts to restricts the actions of malicious agents that may lead to denial of service attacks.

While DASAC is very powerful and flexible, its weakness lies in its dependence upon the choice of classifiers and feature sets used to train the classifiers. Future work on the subject should analyze agent patterns in more detail especially as more agent applications become available. One possible approach to addressing the issue may be to reduce the scope of the problem and study agent patterns based on specific class of applications. As such, administrators will be better equipped in choosing a feature set along with classifiers that may be used on a host based on the services such host may provide.

References

[1] R. Becker, D. D. Corkill, "Determining Confidence When Integrating Contributions from Multiple Agents" In *Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems* (AAMAS 2007), Honolulu, Hawaii, May 2007.

[2] E. Bierman, E. Cloete, "Classification of Malicious Host Threats in Mobile Agent Computing" In *Proceedings of SAICSIT*, 2002, pp. 141-148.

[3] P.-C. Chen, X. Fan, S. Zhu, J. Yen. "Boosting-based learning agents for experience classification" In Proceedings of the 2006 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, pp. 385-388, 2006.

[4] J. Claessens, B. Preneel, J. Vandewalle, "(How) Can Mobile Agents Do Secure Electronic Transactions on Untrusted Hosts? A Survey of the Security Issues and the Current Solutions" In *ACM Transactions on Internet Technology*, 3(1): 28-48, 2003.

[5] K. Deeter, K. Singh, S. Wilson, L. Filipozzi, S Vuong. "APHIDS: A Mobile Agent-Based Programmable Hybrid Intrusion Detection System" MATA 2004, LNCS 3284, pp. 244-253, 2004. © Springer-Verlag Berlin Heidelberg 2004.

[6] W. Diffie, M. E. Hellman. "New Directions in Cryptography" In *IEEE Transactions on Information Theory,* vol. IT-22: 644-654, 1976.

[7] O. Esparz, M. Fernandez, M. Soriano, "Protecting mobile agents by using traceability techniques" In *IEEE* © 2003.

[8] Y. Freund. Boosting a weak learning algorithm by majority. In *Information and Computation*, 121: 256-285, 1995.

[9] M. S. Greenberg, J. C. Byington, T. Holding, D. G. Harper "Mobile Agents and Security" In *IEEE Communications Magazine*, 1998.

[10] T. Hastie, R. Tibshirani, J. H. Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[11] K. E. B. Hickman "Secure Socket Library" Netscape Communications Corp., Internet Draft RFC (1995).

[12] W. Jansen, T. Karygiannis, "NIST Special Publication 800-19 – Mobile Agent Security" National Institute of Standards and Technology, 2000.

[13] JCE Internet Reference Guide. (n.d). Retrieved December 5th 2006, from http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html

[14] E. Jean, Y. Jiao, A. R. Hurson, T. E. Potok "SAS: A secure aglet server" In *Proceedings of Computer Security Conference 2007*.

[15] E. Jean, Y. Jiao, A. R. Hurson, T. E. Potok "Boosting-based Distributed Adaptive Security-Monitoring through Agent Collaboration" In *Second International Workshop on Agent and Data Mining Interaction* ADMI 2007.

[16] Y. Jiao, A. R. Hurson, "Application of mobile agents in mobile data access systems: A prototype" In *Journal of Database Management*, 15(4): 1-24, 2004.

[17] JSSE Internet Reference Guide. (n.d). Retrieved December 5th 2006, from http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html

[18] D. B. Lange, M. Oshima. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, 1998.

[19] A. Patcha, J.-M. Park. "An overview of anomaly detection techniques: Existing solutions and latest technological trends" In *Computer Networks: The International Journal of Computer and Telecommunications Networking*. Vol. 51, Issue 12 (August 2007) pp. 3448-3470

[20] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197-227, 1990.

[21] C. F. Tschudin. "Mobile Agent Security" In *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, M. Klusch, Ed., Springer-Verlagu, New York, 1999, Chapter 18 pp. 431–446.

[22] I. H. Witten, E. Frank. *Data Mining: Practical machine learning tools and techniques 2nd Edition*, Morgan Kaufmann, San Francisco, 2005.

[23] Y.-S. Wu, B. Foo, Y. Mei, S. Bagchi. "Collaborative Intrusion Detection System (CIDS): A Framework for Accurate and Efficient IDS" In *Proceedings of the 19th Annual Computer Security Applications Conference* (ACSAC 2003) © 2003 IEEE

Evens Jean is a Ph.D. student at the Computer Science and Engineering department of the Pennsylvania State University. He is a member of the Global Information Systems Research Group directed by Dr. Ali R. Hurson. He completed his undergraduate studies in Computer Science with honors at the City College of the City University of New York in 2003. Mobile agents constitute the underlying theme of his research as a Ph.D. student, as thus, he has investigated the security issues faced by the paradigm. His research interests also encompass sensor network, ubiquitous and reconfigurable computing.

Dr. Yu (Cathy) Jiao is a research scientist at the Oak Ridge National Laboratory. She received her B.S. degree from the Civil Aviation Institute of China, Tianjin, China in 1997, and her M.S. and Ph.D. degrees from The Pennsylvania State University, in 2002 and 2005, respectively, all in computer science. Her main research interests include mobile agent technology, pervasive computing, dynamic data stream mining, and natural language processing.

A. R. Hurson is the chair and professor of Computer Science department at the Missouri University of Science and Technology (Formerly known as the University of Missouri-Rolla). Prior to his appoinemt at MST, he was a professor of Computer Science and Engineering department at The Pennsylvania State University. His research for the past 25 years has been directed toward the design and analysis of general as well as special purpose computer architectures. His research has been supported by NSF, NCR Corp., DARPA, IBM, Lockheed Martin, ONR, Pennsylvania State University, and Missouri University of Science and Technology. He has published over 250 technical papers in areas including database systems, multidatabases, application of mobile agent technology, Mobile computing environment, computer architecture and cache memory, parallel and distributed processing, dataflow architectures, and VLSI algorithms.

Professor Hurson has been active in various IEEE/ACM Conferences and has given tutorials for various conferences on global information sharing, database management systems, supercomputer technology, data/knowledge-based systems, dataflow processing, scheduling and load balancing, and parallel computing.