

Boosting-based Distributed and Adaptive Security-Monitoring through Agent Collaboration

Evens Jean[†], Yu Jiao[§], Ali R. Hurson[†], Thomas E. Potok[§]

[†] *Computer Science & Engineering
The Pennsylvania State University
University Park, PA 16801, USA
{Jean, Hurson} @cse.psu.edu*

[§] *Comp. Sciences & Engineering Division
Oak Ridge National Laboratory
Oak Ridge, TN 37831, USA
{Jiaoy, Potokte} @ornl.gov*

Abstract

Within agent systems, two entities, namely hosts and agents, subsist and require protection against potential malicious acts. The use of such agent systems to support the development of practical applications is limited primarily by the risks to which hosts in the system are subject to. This article introduces a distributed and adaptive security-monitoring framework to decrease such potential threats. The proposed framework is based on a modified version of the popular Boosting algorithm to classify malicious agents based on their execution patterns on current and prior hosts. Having implemented the framework for the Aglet platform, we herein present the results of our experiments showcasing the detection of agent entities in the system with intention deviating from that of their well-behaved counterparts.

1. Introduction

Mobile Agents refers to a programming paradigm focused around 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 [16], 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 [8]. A thorough study of the Aglet platform has been conducted in order to assess the security level provided by the Aglet server. The study resulted in the introduction of a new server, aptly named Secure Aglet Server (SAS) [13]. SAS provides secured communication through SSL, makes use of the Java Cryptographic Extension (JCE) [12] to support the notion of Read-Only Data thereby providing agents in

the system with the ability to verify the integrity of collected data. Furthermore, SAS introduces the notion of a MonitorAglet capable of preventing Aglets from initiating a Denial of Service (DoS) attack on host through seemingly normal transition from one lifecycle state to another.

It has been suggested by the Computer Security Division of the National Institute of Standards and Technology that one of the main hindrance to the adoption of mobile agent technology stems from the security concerns of hosts [11]. While SAS has rendered the Aglet platform more secure, it has approached the security problem from an isolated standpoint in regards to the host. 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 to. We herein introduce a boosting-based monitoring system that allows hosts to learn and classify agents through collaboration.

While effectively addressing the security issues within the confines of the specified goal, SAS merely reacts to malicious agents attempting 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. 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.

Moreover, as it now stands, SAS reacts to attacks but does not prevent any such occurrences. Hence, improving the security of the system requires:

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

This article introduces a novel distributed and adaptive security-monitoring framework that strengthens the security of SAS.

The remainder of this article starts out by introducing the necessary background relevant to our work in section 2. Section 3 discusses the proposed scheme, along with the design decisions with which we were faced in implementing the algorithm within SAS. Section 4 describes the experiments conducted along with the observed results. We conclude the paper in section 5 highlighting the benefits of our approach.

2. Background & Related Work

2.1. Mobile agent security

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. Such 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 [6, 8, 18].
- Tracing, obfuscation, trusted hardware as well as encrypted functions and data to protect the mobile agents [2, 6, 8, 18].

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

2.2. 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) [7, 9], which attempt to construct and maximize a separating hyper-plane upon mapping the data onto a higher dimensional space. Other approaches include neural networks, decision trees, as well as boosting [9]. 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. Furthermore, due to the fact that 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 this article to address the issue of identifying malicious agents operating across several hosts.

2.3. Related Work

Agent collaboration has been the focus of various research efforts in recent years. Becker et al. studied the issue of confidence determination to ascertain its effect in collaborative agent systems [1]. The study showed that incorrect confidence-integration may propagate in a multi-agent system 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. Moreover, the agents do take trust into account in determining the dependence of their results upon other agents in the system as they collaborate 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. This learning algorithm 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. Agents higher up in the hierarchy are trained, not based on their observations, but using the classification results of the corresponding agents one level down the hierarchy. Training in the proposed system is hierarchical and tightly coupled amongst agents as the classifiers are inter-dependent. The hierarchical learning system is not suitable to address security concerns as the system is built upon the assumptions that the agents are members of the same team, thus ignoring any trust issues. Furthermore, 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.

3. Distributed and Adaptive Security-Monitoring through Agent Collaboration (DASAC)

We introduce a distributed and adaptive way for hosts within a domain to collaborate and learn to identify malicious entities based on various parameters. By malicious, we mean any entity that deviates from the expected behavior of typical agents that visit a particular host.

3.1. The DASAC Framework

The introduction of the following security scheme stems from the realization that 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, here forth referred to as DASAC. In order 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; all 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 follows in the next section. The base learners 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 of 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 priori 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 to the default class of 0.
- Every host in the agent's history are contacted and asked to communicate to the decision maker their classification of the agent as determined by their respective base learners
- 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) forms a hierarchical structure with the various base learners of the hosts in the distributed environment to thwart attacks. In the final steps, a DM could use various techniques in order to reach a consensus such as majority-vote. We however recommend a version of weighted sum tailored to the problem at hand as specified in Equation 1 where ψ_i represent 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 in order 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 respectively the trust, and confidence levels associated with each host being contacted.

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

Equation 1: Trust and confidence based weighted sum

The recommended version of majority vote stems 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 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 assertion. Thus, 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 heuristically 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 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.

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.

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

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

Equation 2: Security level of an agent

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.

3.2. DASAC Implementation in SAS

We have previously discussed the presence within SAS of an agent entity, the MonitorAglet, 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. Having the base learner implemented as an agent allows for separate thread of execution to handle classification and training. Moreover, as agent entities, a host may have multiple base learners trained on independent feature sets; however within SAS we consider only the case where each host has one base learner. Furthermore, in our implementation, all base learners are built using the classifiers from the weka data-mining library [19]. Figure 1 presents a pictorial representation of the interaction between DMs and base learners within SAS. In a nutshell, we extended SAS by using the MonitorAglet as the DM of DASAC, with other agents

acting as base learners. The implemented version of DASAC makes use of the suggested version of weighted sum to classify agents. Moreover we implemented 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. The details of our implementation are provided in the remainder of this section.

Our implementation of DASAC defines 5 distinct security levels. The MonitorAglet is augmented with the capability of requesting interaction from a system administrator in order to determine the appropriate set of actions to undertake once an agent has been identified as a malicious entity of level 3 or lower. Malicious agent of levels 4 and 5 are automatically denied execution. As the choice of level 3 is an arbitrary cutoff point, we allow room for tuning by an administrator when the DM is first loaded onto the host.

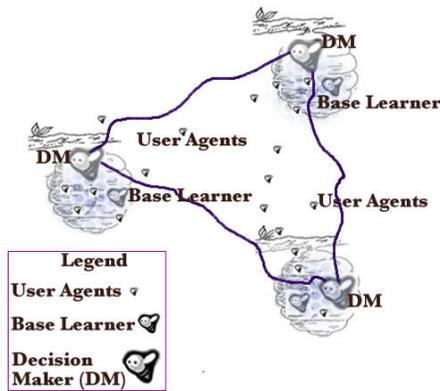


Figure 1: Distributed Interaction between DMs and base learners

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 proven in SAS to be potentially detrimental to the security of a system and thus constitute 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 also 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. As we anticipate that malicious agents may attempt to access various resources in the hope

that the security policy in effect in the system are not well defined, we also take into account the number of security exceptions generated by an agent as well as the frequency at which these exceptions occur. 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 following 9 features to train the classifiers:

- Biased Running Time of an agent
- The frequency of cloning events
- The frequency of activations of an Aglet
- The frequency of dispatch events
- The frequency of retract events
- The frequency of arrival events
- The frequency of security access requests
- The frequency of security access grants
- The frequency of security access denials

Once the feature set has been selected, we trained the classifiers in order to extract data patterns that may help classify an agent based on its behavior. We used a system consisting of 8 hosts. For simplicity, hosts 1 thru 4 have different classifiers, namely, an alternating decision tree with 0 boosting iterations, a fast decision tree learner, a decision stump and a naïve Bayes classifier. The remaining hosts implement the alternating decision tree as well but with 3 boosting iterations. Moreover, the first four hosts have different trust levels, with all others having the same 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 several agent-based applications such as MAMDAS [14] and the Private Information Retrieval prototype [13], along with the agents that were used in assessing the security of SAS [13]. The choice of applications used to collect the data was based not only on their availability, but also due to the fact that their classification is known a priori, as discussed in SAS, and encompasses both classes of agents 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.

The generated data set consists of over 3000 samples, manually classified, of which, roughly 15% is used to train each classifier. Each sample represents the set of features tracked by the MonitorAglet during a run of the agent on the host. 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 2 presents a pictorial view of the accuracy rate of the hosts in the system over multiple runs; note that host 5 really depicts the accuracy rate of hosts 5 thru 8. The accuracy of hosts shown in Figure 2 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. During each run of the experiment, a random set is selected to train the classifiers, hence the slight variation in hosts' accuracy rate over multiple runs.

4. Experimental Analysis

In evaluating the performance of DASAC, we were mainly concerned with two issues. The first one is the overall classification accuracy of the system compared to a host's local classifier, and the second is the impact of trust level on the accuracy.

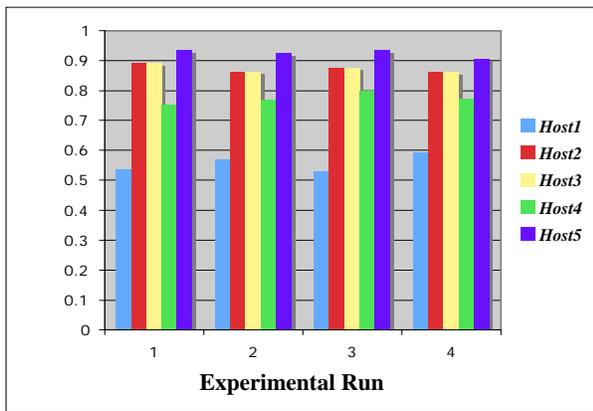


Figure 2: Accuracy of multiple hosts

We chose the first host in the system (Host1 from Figure 2) as the local classifier against which DASAC will be compared. Our choice is based on the fact that host 1 has a performance slightly better than random guessing. As such, 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 that 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 the CirculateAglet, WebServerAglet, and HelloAglet that comes with the Aglet framework. Lastly, we created and used a new version of the WebServerAglet that migrates to hosts and attempts to set up a server on random ports, restricted or not,

repeatedly. Our version of the WebServerAglet 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 evaluate 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 the first one. While the confidence levels used in the experiments were described in section 3.2, 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 3 depicts the measured accuracy of DASAC compared to that of Host1's classifier, the best accuracy rate measured in the system along with the average of host's accuracy rate. On average, DASAC's performance seems to lie 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.

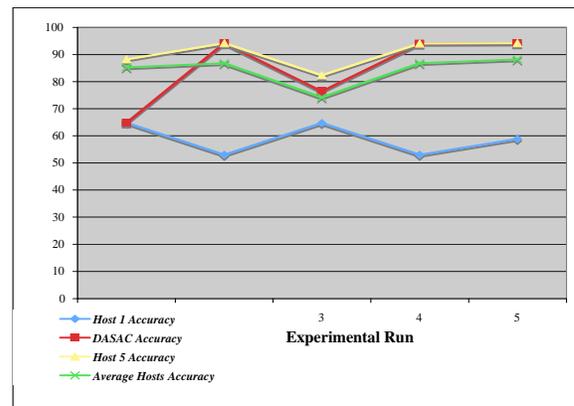


Figure 3: Local vs. DASAC accuracy

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 stems from the trust levels used during the first experiment. We noted that the total trust levels of hosts 2 thru 8 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 host 1, 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 presented in Figure 4 and shows that DASAC still outperforms the average accuracy rate computed. The experiment revealed two crucial points, the first being that all the classifiers in the system can indeed outperform DASAC, as is the case when the trust levels are all 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.



Figure 4: Effect of trust levels on DASAC accuracy

The second interesting factor revealed by the experiment is the fact that DASAC's performance seems to quickly become dependent upon the accuracy rates of the best classifiers in the system. This is explained by the fact that DASAC is in effect designed to that end in order 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 bigger weight on the system's classification of an entity.

5. 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.

6. 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*, p. 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*, Vol. 3 No. 1, 2003, pp. 28-48
- [5] W. Diffie, M. E. Hellman, "New Directions in Cryptography" In *IEEE Transactions on Information Theory*, vol. IT-22, 1976, pp. 644-654
- [6] O. Esparz, M. Fernandez, M. Soriano, "Protecting mobile agents by using traceability techniques". In *IEEE* © 2003.
- [7] Y. Freund. Boosting a weak learning algorithm by majority. In *Information and Computation*, volume 121, pages 256-285, 1995
- [8] M. S. Greenberg, J. C. Byington, T. Holding, D. G. Harper "Mobile Agents and Security" In *IEEE Communications Magazine*, 1998
- [9] T. Hastie, R. Tibshirani, J. H. Friedman. *The Elements of Statistical Learning*. Springer, 2001
- [10] K. E. B. Hickman "Secure Socket Library" Netscape Communications Corp., Internet Draft RFC (1995)
- [11] W. Jansen, T. Karygiannis, "NIST Special Publication 800-19 – Mobile Agent Security" National Institute of Standards and Technology, 2000.
- [12] JCE Internet Reference Guide. (n.d). Retrieved December 5th 2006, from <http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html>

- [13] E. Jean, Y. Jiao, A.R. Hurson, and T.E. Potok, "SAS: A secure aglet server," In *Proc. of Computer Security Conference 2007*,
- [14] Y. Jiao, A. R. Hurson, "Application of mobile agents in mobile data access systems: A prototype" In *Journal of Database Management*, 2004, pp. 1-24
- [15] JSSE Internet Reference Guide. (n.d). Retrieved December 5th 2006, from <http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html>
- [16] D. B. Lange, M. Oshima. Programming and deploying Java mobile agents with Aglets. Addison-Wesley, 1998.
- [17] R. E. Schapire. The strength of weak learnability. *Machine Learning*, 5(2):197-227, 1990
- [18] C. F. Tschudin. "Mobile Agent Security" In *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, M. Klusch, Ed., Springer-Verlag, New York, 1999, Chapter 18 pp. 431-446.
- [19] I. H. Witten, E. Frank. *Data Mining: Practical machine learning tools and techniques 2nd Edition*, Morgan Kaufmann, San Francisco, 2005

ACKNOWLEDGMENTS

The National Science Foundation under the contract IIS-0324835 in part has supported this work.

Notice: This manuscript has been authored by UT-Battelle, LLC, under contract DE-AC05-00OR22725 with the US Department of Energy. The United States Government retains and the publisher, by accepting the article for publication, acknowledges that the United States Government retains a non-exclusive, paid-up, irrevocable, world-wide license to publish or reproduce the published form of this manuscript, or allow others to do so, for United States Government purposes.