

# SAS : A Secure Aglet Server

Evans Jean

Department of Computer  
Science and Engineering  
The Pennsylvania State  
University  
University Park, PA 16802  
jean@cse.psu.edu

Yu Jiao

Computational Sciences  
and Engineering Division  
Oak Ridge National  
Laboratory  
Oak Ridge, TN 37831  
jiaoy@ornl.gov

Ali R. Hurson

Department of Computer  
Science and Engineering  
The Pennsylvania State  
University  
University Park, PA 16802  
hurson@cse.psu.edu

Thomas E. Potok

Computational Sciences  
and Engineering Division  
Oak Ridge National  
Laboratory  
Oak Ridge, TN 37831  
potokte@ornl.gov

## ABSTRACT

Despite the fact that mobile agents have received increasing attention in various research efforts, the use of the paradigm in practical applications has yet to fully emerge. With the presence of infrastructure to support the development of mobile agent applications, security concerns act as the primary deterrent against such trends. Numerous studies have been conducted to address the security issues of mobile agents with a strong focus on the theoretical aspect of the problem. This work attempts to bridge the gap from theory to practice by analyzing the security mechanisms available in Aglet. We herein propose several mechanisms, stemming from theoretical advancements, intended to protect both agents and hosts in order to foster the development of business applications that fully exploit the benefits of agent technology. The proposed mechanisms lay the foundation for implementation of application specific protocols dotted with access control, secured communication and ability to detect tampering of agent data. We demonstrate our contribution through application scenarios of a prototyped Information Retrieval system.

## Categories and Subject Descriptors

C.2.3 [Communication Network]: Network Operations, Network Monitoring, Network Management

D.4.6 [Operating Systems]: Security and Protection, Access Controls, Authentication

H.2.0 [Database Management]: Security, Integrity and Protection

## General Terms

Security, Economics, Reliability, Experimentation, Management

## Keywords

SAS, Aglet, MAMDAS

## 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, the use of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

*Computer Security Conference 2007*, April 11-13, 2007, Myrtle Beach, SC, USA. Copyright 2007 CSC 2007

mobile agents have not transcended from theoretical to practical applications. During recent years, research advances have led to the introduction of a Mobile Agent-based Mobile Data Access Systems (MAMDAS) [10, 11, 12], to facilitate the use of mobile agents in applications requiring database access as agents travel between hosts. The system has been prototyped using the Aglet mobile agent platform. The Aglet platform is fairly documented, easy to install, has received great press coverage and is currently maintained by the open-source community [14]. With the introduction of an infrastructure such as MAMDAS supporting the platform, aggressive development of mobile agent applications on the Aglet platform should be imminent. Unfortunately, such is not the case as the paradigm is plagued with numerous security issues. The security threats facing mobile agents, including Aglets, have been studied in depth and categorized into host-to-agent and agent-to-host [6]. Solutions to such threats have to this point only been introduced from a theoretical perspective.

In our effort to help foster the emergence of mobile agents to address practical issues in the business world, we conducted an analysis of the security options available in the Aglet platform. Our work resulted in the development of a Secured Aglet Server (SAS) providing:

1. Secured communication
2. Controlled resource consumption of agents
3. Integrity and reliability of agent's data

In discussing our findings we start out by introducing the necessary background in section 2. Section 3 analyzes the vulnerabilities of the current Aglet server. We introduce SAS in section 4. Section 5 presents an information retrieval prototype designed on top of MAMDAS to demonstrate the contribution of our work. We conclude our discussion in section 6 highlighting our future work.

## 2. BACKGROUND

Mobile agents lend themselves nicely to searches and computation that requires parallel processing and 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. Such security issues in mobile agents have been studied and some of the proposed solutions include but are not limited to the following:

1. Code signing, access control, proof carrying code and path histories to protect the hosts [6, 5, 18, 20].

2. Tracing, obfuscation, trusted hardware as well as encrypted functions and data to protect the mobile agents [6, 5, 1, 20]

Research in mobile agent security is still an open field, and many of these approaches remain theoretical. Our goal is to design and implement a secure agent platform based on the Aglet workbench, which incorporates existing theoretical solutions and provides a foundation for secure application development.

### 3. VULNERABILITY ANALYSIS OF IBM AGLET

In presenting our finding as we conducted a vulnerability analysis of the Aglet platform, we will start with a brief overview of the environment in section 3.1. The remaining subsections will showcase the vulnerabilities that we have identified in the platform.

#### 3.1 IBM Aglet Architecture

Aglet is a library written in Java, released by IBM to support the development of mobile code. The execution environment within which Aglets are executed is referred to as the Aglet's Context and is responsible for enforcing the security restrictions of the

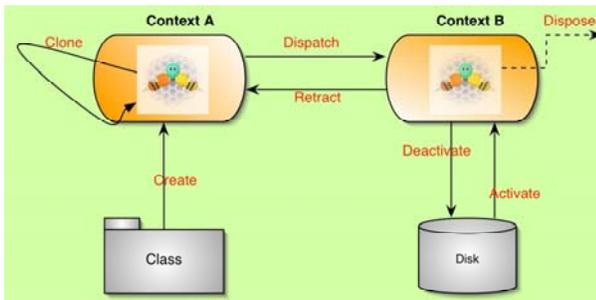


Figure 1: Aglet Lifecycle

mobile code. Aglets run on the Tahiti Server and may adopt seven different states (as illustrated in Figure1) during their life cycle [16, 17]:

- Activated: Aglets are loaded from storage and allowed to resume execution.
- Deactivated: Aglet's execution is halted and its state is saved.
- Cloned: Aglet is copied for concurrent execution.
- Disposed: Execution of the aglet ceases permanently
- Created: Aglet is initialized for execution
- Dispatched: Aglet is sent to another execution context.
- Retracted: Aglet is obtained from another execution context.

The Aglet architecture consists of two main layers which are the Runtime Layer running on top of the Communication Layer. The tight coordination between these two layers provides Aglets with their execution environment. The Runtime Layer manages Aglets' bytecode, and enforces the security restrictions in effect in the environment. The Communication Layer, on the other hand, provides the basic mechanisms to allow Aglet's mobility and message passing through support of Agent Transfer Protocol (ATP) [15] and RMI [19].

#### 3.2 Communication Vulnerability

Of primordial importance to any agent systems is the ability to allow agents to communicate and roam the network. It is therefore

imperative that such communication and mobility be performed in a secure manner. As mobile agents, Aglets suffer from the same security vulnerabilities that plague the programming paradigm. The interception of Aglets as they move from one host to another is a serious threat encompassed by the need to protect agents from malicious hosts. Protecting Aglets against malicious hosts entails ensuring that the agents and their messages are sent exclusively to the intended entities (hosts or agents). In fact, we believe that no mobile agent system can allege to protect agents from malicious hosts if it cannot verify the identities of the hosts in the system.

Presently, the Tahiti server supports authentication of the entities that wish to establish communication, typically the authentication of servers. The communicating parties in Tahiti are authenticated using a Challenge-Response scheme based on the Diffie-Hellman algorithm [3], a cryptographic protocol, which allows two parties to exchange a secret key over an insecure medium without any prior secrets. The authentication of the entities is done in phases once one party has initiated the cycle, and is based on security domains defined as a region with homogeneous level of security [18].

In our attempt to determine the efficiency of Tahiti in protecting the communication channels used by Aglets, we established a link between two instances of the server running on different machines. Our goal is to try and intercept the packets being exchanged between the two servers, as the interception of such packets can easily lead to the reconstruction of the Aglet or message being transmitted. We used a third machine and the readily available Dsniff software [4] to try and intercept the packets between the two servers and display the information being captured. Figure 2 displays some of the packets captured by the attacker, containing the Aglet in its serialized state as it is being transmitted over the unsecured communication channel. Pertinent information in Figure 2 is highlighted in red, such as the AgletID of the agent, and part of the Aglet's data. It is not surprising to us that we were able to easily carry such an attack being that Tahiti does not use any encryption during communication. The authentication scheme described earlier is used only to verify the

```

0x0030: 0000 0000 4449 5350 4154 4348 2061 7470 ....DISPATCH.atp
0x0040: 3a2f 2f31 3932 2e31 3638 2e30 2e34 2041 ../192.168.0.4.A
0x0050: 5450 2f30 2e31 0d0a 6461 7465 3a54 6875 TP/0.1..date:Thu
0x0060: 2044 6563 2031 3520 3233 3a31 363a 3339 .Dec.15.23:16:39
0x0070: 2045 5354 2032 3030 350d 0a75 7365 722d .EST.2005..user-
0x0080: 6167 656e 743a 4167 6c65 7473 0d0a 6672 agent:Aglets..fr
0x0090: 6f6d 3a61 7470 3a2f 2f6d 6f62 696c 6530 om:atp://mobile0
0x00a0: 322e 6c6f 6361 6c3a 3434 3334 0d0a 686f 2.local:4434..ho
0x00b0: 7374 3a31 3932 2e31 3638 2e30 2e34 0d0a st:192.168.0.4..
0x00c0: 6167 656e 742d 7379 7374 656d 3a41 676c agent-system:Aglet
0x00d0: 6574 730d 0a61 6765 6e74 2d6c 616e 6775 ets..agent-langu
0x00e0: 6167 653a 4a61 7661 0d0a 6167 656e 742d age:Java..agent-
0x00f0: 6964 3a30 3030 3130 3030 3832 3636 3066 id:000100082660f
0x0100: 436f 6e74 656e 742d 456e 636f 6469 6e67 Contel encoding
0x0110: 3a0d 0a43 6f6e 7465 6e74 2d4c 616e 6775 ..Co nt-Langu
0x0120: 6167 653a 0d0a 436f 6e74 656e 742d 4c65 age:..ent-Le
0x0130: 6e67 7468 3a37 ..320d 0a0d 0a ngth:75
0x0140: 2e6d 6973 632e 4172 6368 ..6513 707d ..misc.Archive.p)
0x0150: dbef 6cb8 3c02 0001 4c00 0000 4163 6865 ..l<...L..cache
0x0160: 7400 154c 6a61 7661 2175 7469 ..4861 t..Ljava/util/Ha
0x0170: 7368 7461 626c 653b 7870 7372 0010 ..shtable;xpsr..ja
0x0180: 7661 2e75 7469 6c2e 4861 7368 7461 6200 ..a.util.Hashtabl
0x0190: 6513 bb0f 2521 4ae4 b803 0002 4600 0a6c e...%J....F..I
0x01a0: 6f61 6446 6163 746f 7249 0009 7468 7265 oadFactor!..thre
0x01b0: 7368 6f6c 6478 703f 4000 0000 0000 0877 ..sholdxp?@.....w
0x01c0: 0800 0000 0b00 0000 0174 0022 6578 616d .....t."exam
0x01d0: 706c 6573 2f68 7474 702f 5765 6253 6572 ples/http/WebSer
0x01e0: 7665 7241 676c 6574 2e63 6c61 7373 7372 verAglet.classsr
0x01f0: 001e 636f 6d2e 6962 6d2e 6177 622e 6d69 ..com.ibm.awb.mi
0x0200: 7363 2e41 7263 6869 7665 2445 6e74 7279 sc.Archive$Entry
  
```

Figure 2: Captured Packets

identity of the servers but not as a prelude to a key exchange that would be used to encrypt future communication. The authentication scheme was designed to prevent security attacks, such as the reflection attack, against the communication layer of Tahiti.

Our experiment shows that the authentication scheme is still susceptible to more complex attacks such as man-in-the-middle (MITM) attacks carried out through eavesdropping on a communication channel with the ability to modify or insert data into the channel. It also exposes a great weakness in Tahiti in that Aglets can be easily intercepted while traveling on the insecure communication channels and be made to execute on a malicious host. Our first experiment has thus led us to conclude that the Aglet framework cannot currently satisfy the requirement of agents to migrate exclusively to intended hosts. Hence there is an obvious need to address such a shortcoming of the framework in order to increase its use in commercial applications.

### 3.3 Data Vulnerability

Protecting mobile agents against malicious hosts must occur on two different levels. Agents must first be able to freely roam the network and travel only to intended hosts (see section 3.2); moreover, agents must be able to freely execute on the hosts onto which they have migrated. To this day, there exists no highly acclaimed solution to the problem of protecting an agent from a malicious host, although numerous proposals have been submitted to address the issue of identifying malicious hosts [5, 1]. When an Aglet arrives at a host to perform a computation, it is at the complete mercy of the host in question. Hosts are capable of manipulating the Aglet's data collected from previous hosts [7]. The danger here is that if we consider the case where an Aglet is to collect ticket prices from different Airlines, one of the Airlines might raise the price of all other Airlines and thus force the user to purchase the ticket from its company even though it may not have the best price available. Tahiti presently has no mechanisms in place to help detect tampering of an Aglet's data from a malicious host nor does it provide any mechanism to help identify the existence of such hosts. It is thus imperative that such issue be addressed to foster the use of agents in privacy-aware applications.

### 3.4 Resource Vulnerability

The ability for hosts to support the execution of potentially malicious agents has been well discussed in the literature within the scope of mobile agents in general [6, 20, 1, 2]. Aglets suffer from the same limitations in that malicious Aglets could be detrimental to the proper functioning of a host. Being that Aglets are built on top of Java technology, the Tahiti server makes adequate use of the Java sandboxing techniques to protect hosts. Aglets execute within a runtime environment, which controls their access to system resources such as files and devices. Aglets are allowed to execute a set of actions on themselves that allows them to move from one execution state to another during the Aglets lifecycle.

During the course of our investigation, we have identified an inherent vulnerability of host resources stemming from the normal lifecycle of Aglets. To illustrate our novel observation, we implemented an Aglet application whose purpose is to attempt to consume as much resources as possible on a host through traversal of the Aglet's normal lifecycle stages. To ascertain the effect of such a behavior from an Aglet, we considered the best-case scenario in which the clones of the Aglet do not carry out any

computation, nor does the Aglet itself carry any computationally expensive operations. The developed Aglet application was instantiated from a Tahiti server running on a 1.0Ghz G4 processor with 1.0GB of memory. The application that we developed carried out the attacks on the host through repeated cloning, as well as by creating and dispatching new Aglets to a target host. The attack was successfully carried as well through the activation or retraction of Aglets. Figure 3 depicts the amount of memory used by the server based on the number of active Aglets. The running instances of the Aglets have a dramatic effect on the amount of memory being used by the Tahiti server. On average, with approximately 7000 instances of the Aglet application, the server reports a *java.lang.OutOfMemoryError* and is then unable to process any more requests including the creation of new Aglets.

Our research has successfully identified and exposed the resource vulnerability of Tahiti that can be easily exploited by a malicious

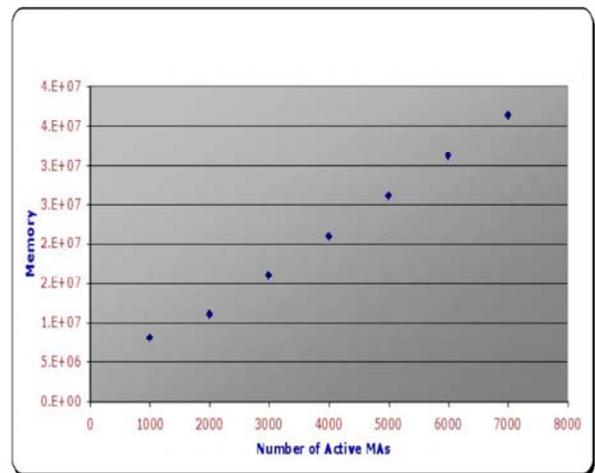


Figure 3: Effect of DoS Attack on Host's Memory

Aglet or even by an Aglet that has been improperly designed. The erroneous introduction of an infinite loop (or even a loop with a large number of repetitions) around a statement that would cause the Aglet to clone itself, or create an instance of another aglet could lead to disastrous effects on the resource utilization of a host. The lack of infrastructure to protect hosts is as detrimental to the adoption of the paradigm as is the threats faced by the agents and ought to be tackled to increase adoption of the paradigm.

## 4. SECURE AGLET SERVER (SAS)

In order to address the security vulnerabilities discussed in section 3, we designed and implemented a Secured Aglet Server (SAS). SAS' look and feel closely resembles that of Tahiti, its architecture is a replica of Tahiti's architecture with some added functionalities to support the new security mechanisms.

### 4.1 Secure Communication

As was shown earlier, the Communication Layer of Aglets makes use of unsecured protocol, thus leaving the framework open to a range of attacks, which we intend to address in our research. The current industry standard in addressing communication security being SSL [8], keeping in line with our intent of fostering the

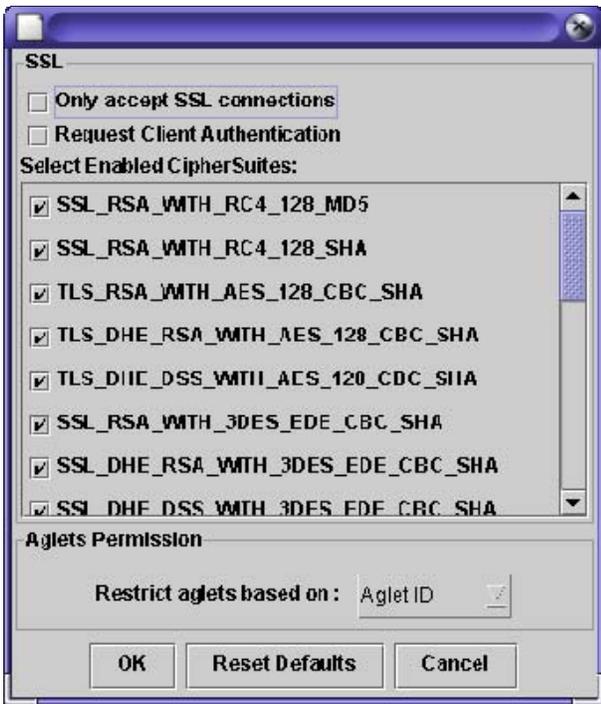


Figure 4: SAS Security Preference Window

adoption of agents in commercial applications, we opted to implement SSL in the Tahiti server, using the Java Secure Socket

Extension (JSSE) [13]. JSSE provides a standard Application Programming Interface (API) allowing for the creation of secure connection between hosts with the option of having both parties authenticated (see Figure 4). The sockets created by the JSSE API are protocol-independent and thus needs to be configured to work with the protocols used by the Aglets framework.

The implementation of the SSL sockets created by JSSE into SAS occurred at the Communication Layer with the introduction of a sub-layer (SSL Layer) below the protocols. The sub-layer handles the creation and management of the secure sockets for use by the different protocols. Our implementation of the sub-layer provides administrators with the mechanisms necessary to implement application-specific protocols. As such, administrators can customize the server to accept connections from clients, which do not support SSL or to reject such connections. Allowing for an administrator to request that unauthenticated clients be denied connectivity to the server is justified by the observation that communication in the Aglet framework occurs only between servers; hence it is reasonable to expect each server to possess a signed certificate. The Cipher suites to be used on the server's connections can also be specified; hence servers can enforce different encryption strength on their communication channels all within the boundaries of SSL. Through the use of SSL sockets in our implementation, 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.

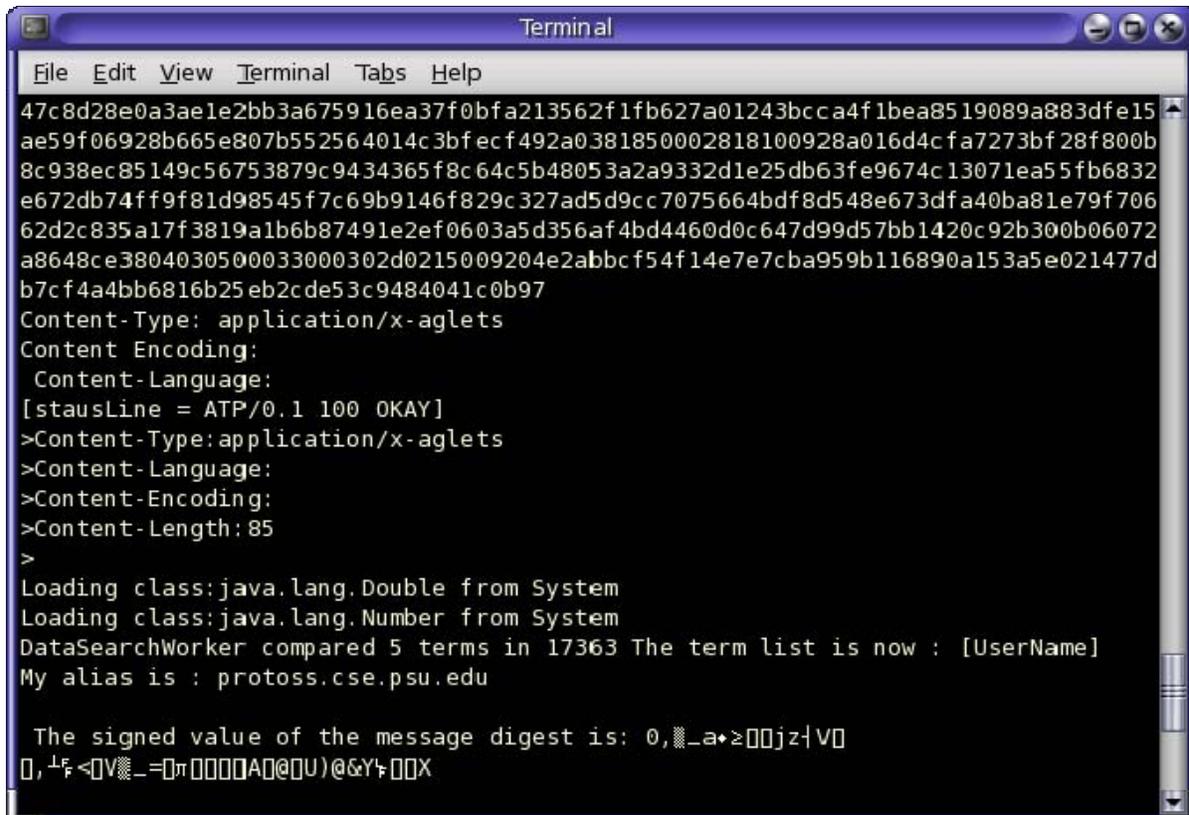


Figure 5: Signed Message Digest printout

## 4.2 Secure Data

While the scenario discussed in section 3.3 cannot be currently prevented, it is necessary for an Aglet to have a way of determining whether the information that it has collected over numerous hosts has been tampered with. Granting Aglets the ability to detect tampering with their data required that we extended the functionalities of the server. The Runtime Layer of SAS is extended to support the creation of Message Digests using the Java Cryptography Extension (JCE) [9] as well as the ability to digitally sign objects. We provide Aglets with a java class library that implements the concept of Read-Only data. The class library allows agents to save data but can only read the data set in question once it has been submitted to the class library. The library makes use of the extension of SAS to compute and store message digests of each datum as it is submitted to the library (see Figure 5). 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.

## 4.3 Secure Resources

Dealing with the possibility of an Aglet overusing the resources of a host as it traverses its normal lifecycle, requires the design of 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 is responsible for tracking the resources, in terms of instances, in use by an Aglet to ensure that the specified limit is never exceeded. The MonitorAglet makes use of a Resource data structure to track the number of instances of a particular Aglet that are present in the system. The data structure is built based on the properties of Aglets such as their IDs. As such, if the Resource data structure manages class names of the form "Attacks.\*"; any Aglet whose name starts with "Attacks" would belong to the structure.

Within the scope of SAS, we define instances as the instantiation of an Aglet or Message object. Furthermore, an Aglet *B* is an instance of an Aglet *A* if and only if one of the following is true

1. *B* belongs to the same Resource object as *A*
2. *A* created, retracted or activated *B*.
3. *B* is a clone of *A*.

Managing the number of instances of an Aglet that are present in the system is then reduced to keeping track of the number of instances of an Aglet a particular Resource has. Each Resource object contains a maximum number of instances to allow in the system; once that limit has been reached, the MonitorAglet prevents the creation, activation or retraction of any other instances of the Aglets in the Resource until one of the instances of the Resource has been deactivated, dispatched or disposed of (see Figure 6). An identical scheme is used to control the number of messages an Aglet can delegate in the system.

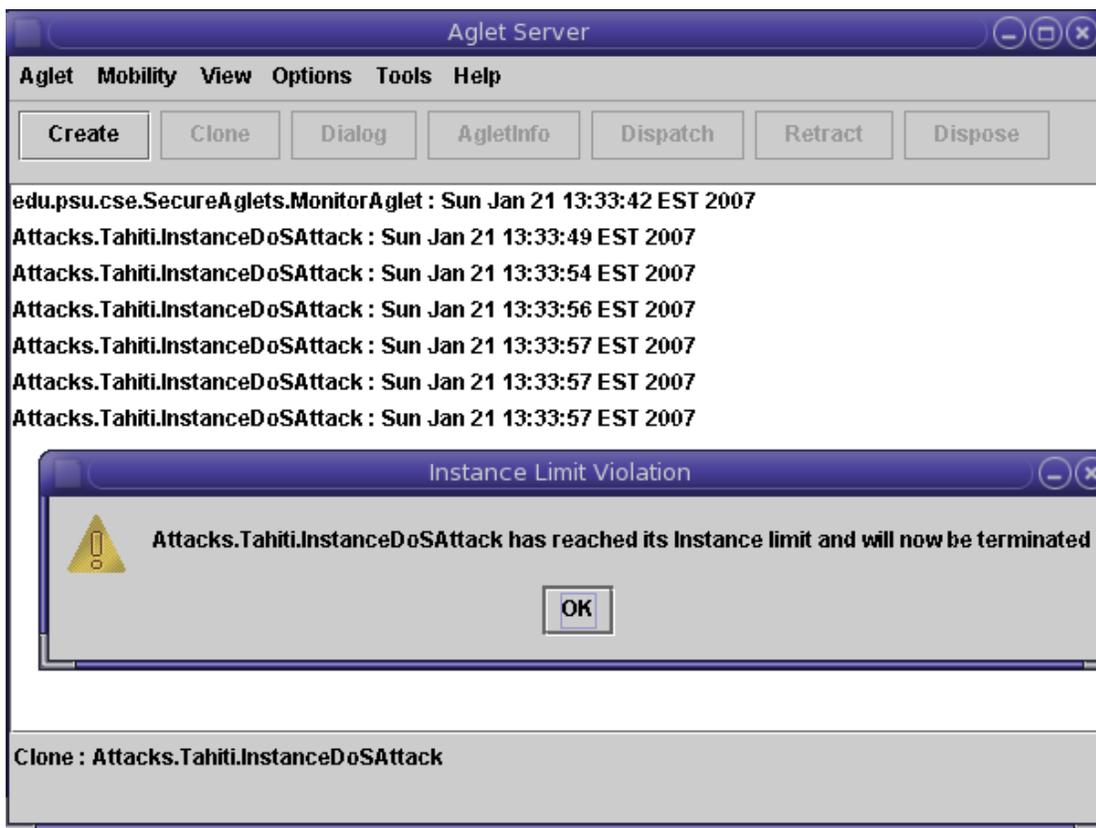


Figure 6: Detection of Instance Violation by MonitorAglet

A notable feature provided by the MonitorAglet is the ability for a remote Aglet to contact the Monitor and determine the Resources and Permissions that it should expect to have available if it were to migrate to the host where the Monitor Aglet is executing. This allows Aglets to decide whether they should move to a host depending upon their requirements.

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 attack described earlier. Moreover, Aglets are now able to determine in advance whether they should migrate to a host based on the resources and access restrictions in place in the host of interest

## 5. PRIVACY-PRESERVING INFORMATION RETRIEVER (PIR) – A CASE STUDY

As part of our research, we have prototyped a Privacy-preserving Information Retriever (PIR) to highlight the contributions of the proposed work. The PIR prototype (see Figure 7) is based on MAMDAS, a multidatabase access system tailored to support user mobility. We envision the use of the PIR to help companies make

a hiring decision about an individual. Government agencies along with private corporations maintain pertinent data that are, for the most part, readily available to the public. Arrest records, credit report as well as past salaries represent some of the information that may come into play in hiring a potential employee. The PIR has its own set of security requirements. As we explore such requirements, we will demonstrate the mechanisms available in SAS to provide the required level of security.

Being built on top of MAMDAS, PIR makes heavy use of agents to collect the information of interest. Notably, PIR uses an Aglet, DataSearchWorker (DSW), to roam a hierarchical network structure that reflects the semantic relationship of the datasets. Due to the potential sensitivity of the data collected by the DSW, it is imperative that the Aglet only travels to hosts of interest to the task at hand to ensure reliability of the data. As we have shown earlier, third parties can intercept agents in Tahiti, thereby compromising the security of PIR. On the other hand, through the use of SSL, SAS provides applications with the necessary mechanisms to ensure that delegated agents only travel to intended hosts. The prototyped PIR makes use of the available mechanism to attain the communication security needed to prevent access to pertinent data through authentication of the entities in the network.

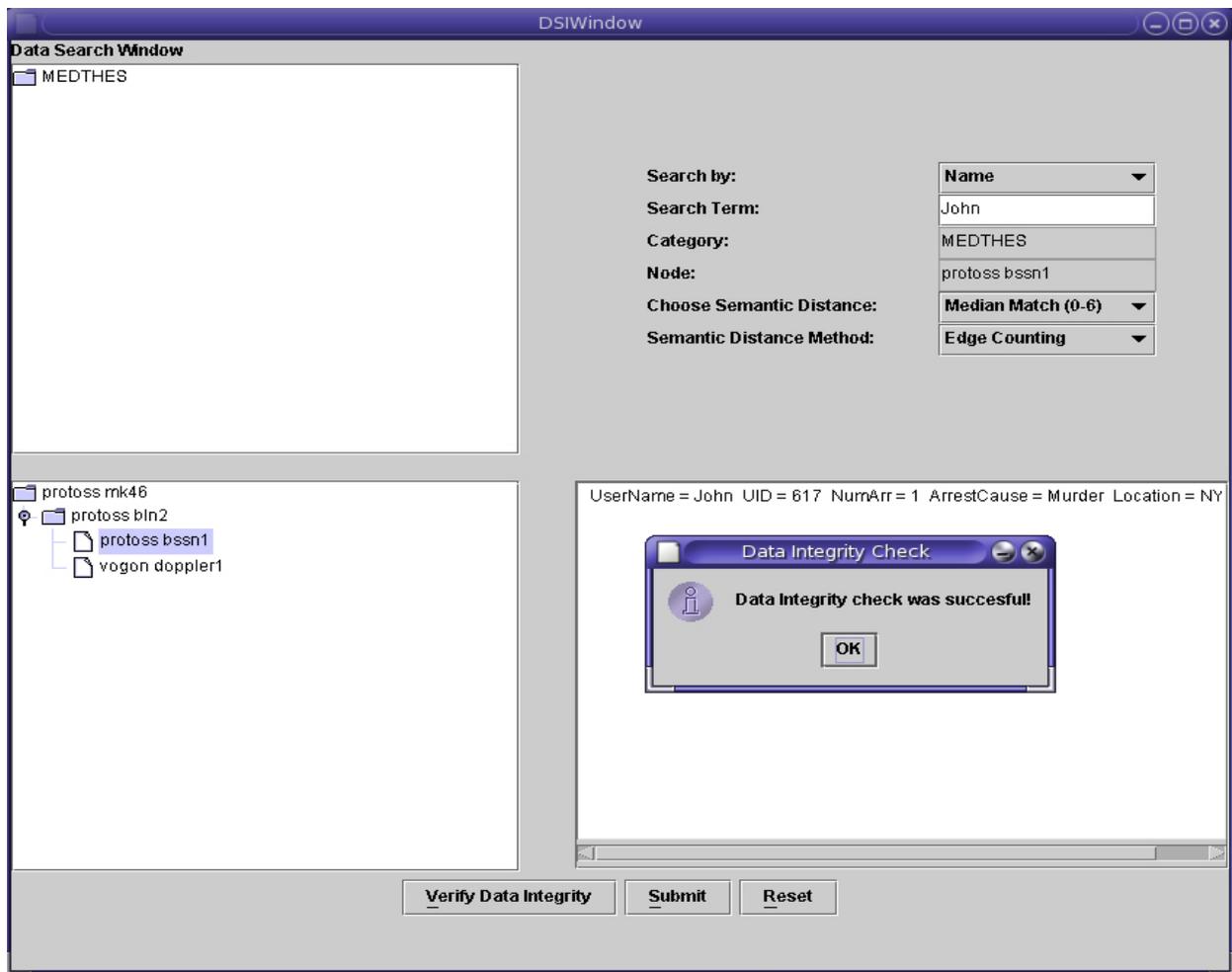
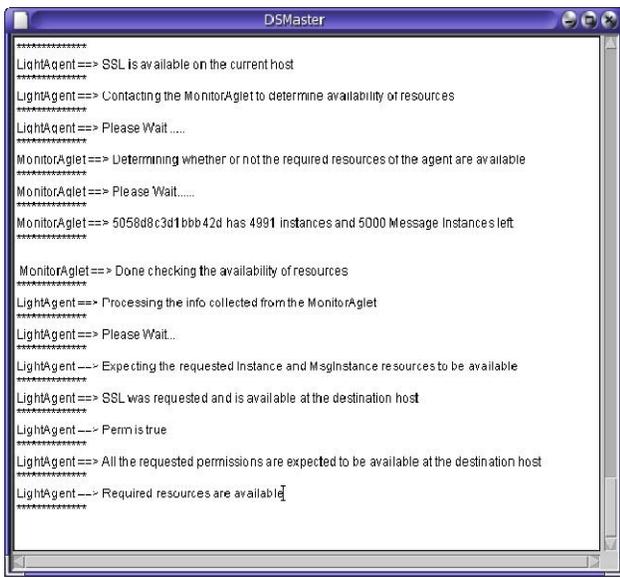


Figure 7: PIR Screenshot with Data Integrity Check

To collect the required data, the DSW will visit various hosts. The hosts visited may be past employers of the hiring candidate, and thus may benefit from sabotaging the candidate. If the candidate's records are altered, such as through insertion of arrests records or increase of past salaries, the collected data cannot be used to make a reliable decision. PIR requires that the integrity of the collected data be maintained as it will be used in a manner that will affect the future of the company. While Tahiti does not provide any support to applications with similar requirements, SAS provides the basic functionalities to allow detection of data tampering. The PrivateData library, discussed earlier, allows applications to profit from such functionalities. Through the use of the library, PIR allows users to verify the integrity of the result, as the DSW requires hosts to sign any collected data. If any of the collected data has been corrupted, PIR can notify the user, should the latter decide to go through with the verification step (see Figure 7).



**Figure 8: Negotiation between LightAgent and MonitorAglet**

Aside from communication and data security, equally important to PIR is the issue of ensuring that the resources of hosts are not consumed in vain. Malicious agents, through lifecycle operations, may subject hosts to DoS attacks; such occurrence would prevent PIR agents from accomplishing their goals and collect the information of interest. Within PIR, multiple agents may need access to the same host, as some personal information, such as credit reports or criminal records are on a limited set of hosts. Similar to any application, data availability is of primordial importance in our prototyped PIR. The existence of the system would be futile if it could not collect the criminal records of a potential employee. Tahiti, as we have discussed, can be subjected to such DoS attacks as it strays away from the micro-management of Aglets. SAS addresses the issue through the introduction of a MonitorAglet to track the actions of agents in the system. The MonitorAglet ensures the availability of DSW's hosts of interests by limiting the number of instances of agents.

The heterogeneity with which the PIR prototype operates yields an underlying issue. The set of operations that the DSW can perform will vary from one host to another. This results in the

possibility for the DSW to be dispatched to a host with relevant information only to be denied the right to execute on such hosts. The occurrence of the depicted scenario leads to inefficient use of network resources. While such a case is not in and of itself a security threat, MonitorAglet provides PIR with support to reduce such occurrences. The DSW agent in PIR ensures that it will be allowed to execute its code to completion before migrating to a host. This is accomplished by dispatching a surrogate agent (LightAgent) to the destination host. LightAgent contacts the MonitorAglet of the destination host and determines whether or not the host will support execution of the DSW (see Figure 8). As such DSWs can decide to migrate to a host based on whether or not they will have access to the required resources. The issue is highly important since agents may need to execute different portions of their code to cope with heterogeneity of the sources, such as the manner in which to access a database. This is evidenced by the possible need to use a package such as sun.jdbc.odbc to acquire the necessary data. Within our prototype, such scenario is modeled by the need for DSWs to access flat file databases. Under such a condition, PIR can dispatch the surrogate agent to determine whether the DSW will have the necessary permission to access the database of interest.

The PIR prototype has been achieved in SAS due to the fact that SAS provides the basic functionalities required to sustain a secure system. Figure 4 shows the prototype running on the SAS server. PIR allows users to specify the global term to use in the search along with the semantic specifications. Once the results are collected, the user can choose to verify their integrity. SAS allowed the agents in the prototyped PIR to travel only to intended hosts through authentication. Moreover, the integrity of the collected data can be verified through the use of the PrivateData library. Lastly, through the MonitorAglet, SAS provides data availability to the prototype along with the ability for PIR agents to not migrate to a host that will not support their full execution.

## 6. CONCLUSION

We have herein addressed the security issues present in the Aglet framework through the introduction of SAS. The implemented mechanisms in SAS are intended to help foster the emergence of mobile agents in commercial applications. We have addressed both aspects of Aglets security from a practical standpoint. SAS provides secured communication channel, detection of data tampering and controlled number of instances of agents on the server. Moreover, agents in SAS may determine expected resources from future hosts. Through the PIR prototype, we have demonstrated how the proposed mechanisms can be used to foster secure development of agent applications.

Presently, Aglets can detect the tampering of their data; however, protecting the privacy of the agent's data remains an issue. SAS can allow hosts to starve non-malicious agents, and such a case should be addressed in future work. Moreover, hosts with limited power are still at risk against gratuitous exploitation of CPU cycles. Further efforts to secure the framework should focus on the aforementioned issues.

## 7. 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.

## 8. REFERENCES

- [1] Bierman E., Cloete E., 2002. Classification of Malicious Host Threats in Mobile Agent Computing. In *Proceedings of SAICSIT*, 141-148
- [2] Claessens J., Preneel B., Vandewalle J. (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), 28-48
- [3] Diffie W., Hellman M. E. New Directions in Cryptography In *IEEE Transactions on Information Theory*, (vol. IT-22, 1976), 644-654
- [4] DSniff Networking Tools. (n.d). Retrieved December 28<sup>th</sup> 2006, from <http://monkey.org/~dugsong/dsniff/>
- [5] Esparza O., Fernandez M., Soriano M. Protecting mobile agents by using traceability techniques. In *IEEE* (© 2003)
- [6] Greenberg M. S., Byington J. C., Holding, T., Harper D. G. Mobile Agents and Security. In *IEEE Communications Magazine* (1998)
- [7] Gunter K., Lange D. B., Oshima M. A Security Model for Aglets. In *IEEE Internet Computing* (Vol.1, No. 4, 1997), 68-77
- [8] Hickman K. E. B. Secure Socket Library. *Netscape Communications Corp., Internet Draft RFC* (1995)
- [9] JCE Internet Reference Guide. (n.d). Retrieved December 5<sup>th</sup> 2006, from <http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html>
- [10] Jiao Y., Hurson A. R. Performance Analysis of Mobile Agents in Mobile Distributed Information Retrieval Systems – A Quantitative Case Study. In *Journal of Interconnection Networks*, (2004) 351-372
- [11] Jiao Y., Hurson A. R. Modeling and Performance Evaluation of Agent and Client/Server-Based Information Retrieval Systems - A Case Study. In *Proceedings of Communication Networks and Distributed Systems Modeling and Simulation Conference*, (2004), 1-6
- [12] Jiao, Y., Hurson, A.R. Application of mobile agents in mobile data access systems: A prototype. In *Journal of Database Management*, (2004) 1-24
- [13] JSSE Internet Reference Guide. (n.d). Retrieved December 5<sup>th</sup> 2006, from <http://java.sun.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html>
- [14] Kiniry J., Zimmerman D., 1997. A Hands-On Look At Java Mobile Agents. In *IEEE Internet Computing*, (1997) 21-30
- [15] Lange D., Aridor Y., Agent Transfer Protocol, IBM Tokyo Research Laboratory <http://www.trl.ibm.com/aglets/atp/atp.htm>
- [16] Lange D. B., Oshima M. Mobile Agents with JAVA: The Aglet API. In *World Wide Web 1*, (1998) 111-121
- [17] Lange D. B., Oshima M. *Programming and deploying Java mobile agents with Aglets*. Addison-Wesley, 1998.
- [18] Ono K., Tai H. A security scheme for Aglets. In *Software – Practice and Experience* (2002)
- [19] RMI White Paper. (n.d) Retrieved November 29<sup>th</sup> 2006, from <http://java.sun.com/javase/technologies/core/basic/rmi/whitepaper/index.jsp>
- [20] Tschudin C. F., 1999. Mobile Agent Security. In *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, M. Klusch, Ed., Springer-Verlag, New York, Chapter 18, 431–446.