A SECURITY MODEL FOR MOBILE AGENT ENVIRONMENTS USING X.509 PROXY CERTIFICATES

Subhashini Raghunathan, B.E.

Thesis Prepared for the Degree of

MASTER OF SCIENCE

UNIVERSITY OF NORTH TEXAS

December 2002

APPROVED:

Armin R. Mikler, Major Professor
Steve Tate, Committee Member
Azzedine Boukerche, Committee Member
Tom Jacob, Committee Member
Krishna Kavi, Chair of the Department of Computer Science
Robert Brazile, Graduate Advisor
C. Neal Tate, Dean of the Robert B. Toulouse School of Graduate Studies

Raghunathan, Subhashini, A Security Model for Mobile Agents using X.509 Proxy Certificates. Master of Science (Computer Science), December 2002, 76 pp., 1 table, 9 figures, 37 titles.

Mobile agent technology presents an attractive alternative to the client-server paradigm for several network and real-time applications. However, for most applications, the lack of a viable agent security model has limited the adoption of the agent paradigm. This thesis presents a security model for mobile agents based on a security infrastructure for Computational Grids, and specifically, on X.509 Proxy Certificates. Proxy Certificates serve as credentials for Grid applications, and their primary purpose is temporary delegation of authority. Exploiting the similarity between Grid applications and mobile agent applications, this thesis motivates the use of Proxy Certificates as credentials for mobile agents. A new extension for Proxy Certificates is proposed in order to make them suited to mobile agent applications, and mechanisms are presented for agent-to-host authentication, restriction of agent privileges, and secure delegation of authority during spawning of new agents. Finally, the implementation of the proposed security mechanisms as modules within a multi-lingual and modular agent infrastructure, the Distributed Agent Delivery System, is discussed.

ACKNOWLEDGEMENTS

Several people have been instrumental in the completion of this work. First of all, I want to thank my advisor Dr. Mikler for his guidance, insight, patience, and support throughout my thesis work. Thank you also for helping me find such an interesting and challenging topic as this, and helping me see what research is all about. The idea for my thesis sprung from work I had done at Argonne National Laboratories for the Globus Project™, and I would like to thank Steve Tuecke and Von Welch for guiding me through my work at Argonne and patiently listening and advising me on my ideas for a thesis.

Sincere thanks to Dr. Tate, Dr. Boukerche, and Dr. Jacob for taking time off their busy schedules to be a part of my thesis committee. A special thanks to Dr. Tate for putting up patiently with my incessant questions and doubts. Thanks also to the Computer Science department and faculty for providing me an opportunity to pursue my master's degree at UNT.

My heartfelt thanks go out to the folks at NRL - Prasanna, Vivek, Cliff, John, Kaizar, Sandeep, for always being around, always ready to help, and always being such a cool bunch. And my roommates - Anu and Sandhya - thanks for your moral support and food, and for being such fun people to live with.

Thanks to all my other friends - both here in the USA and in India - who go unmentioned, but definitely not unforgotten. And finally, a warm and loving thanks to my dad, mom, Badri, and Padmini for their continued support throughout my master's degree.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Most network applications currently employ the client-server model, where both the client and server are static processes that communicate data over a network using message passing or Remote Procedure Calls (RPC). Mobile agents present a paradigm shift in the way we traditionally view network or distributed applications, in that code travels over the network to where the data is, instead of vice versa as in the client-server paradigm. The benefits of this approach include reduced bandwidth requirements in data filtering applications, reduced network latency in highly interactive applications, server-side customization of software to add client-specific functionality, and disconnected operation in mobile client computing [18]. Other application areas for mobile agents include e-commerce, active networks [34], and deploying new software autonomously. Mobile agents are also well suited to distributed applications by virtue of their ability to clone themselves and designate a clone for each sub-task. In addition, agents are being proposed as an increasingly attractive solution for real-time applications such as routing and resource management in networks due to their reduced network overhead and fault-tolerant properties [23].

A mobile agent, in general, is a software entity (program) that acts on behalf of another entity (individual, organization, etc.). It is autonomous (capable of decision making), goal-directed, and is capable of suspending execution on one platform, moving to another platform, and resuming execution on the second platform. This implies that an agent encapsulates both program code and state variables such as the program counter, registers, stack, etc. Since an agent is nothing but a program, it requires a platform that will execute its instructions. Depending upon the language used to program the agent, it may be executed either as an independent process or in the context of a language interpretor. In addition, during its lifetime, an agent may perform various activities such as acquiring resources on a host, migrating from host to host, communicating with other agents on local or remote hosts, creating clones, or merging with other agents [23]. All these activities require the co-operation of the host that the agent executes on. The underlying system on the host that makes agent applications possible is known as the agent's infrastructure. Thus, in addition to providing an execution environment, the infrastructure must support all the functionality required by an agent application. It is also desirable that an infrastructure be able to support agents varying in their requirements for language, authentication, fault-tolerance, etc. Many existing agent infrastructures restrict the agents that they support to a single language and authentication mechanism, and by virtue of their

design, to one class of applications. Further, they do not provide interoperability between other agent infrastructures, which is one of the reasons the agent paradigm is not prevalent in today's network applications.

Another major concern with mobile agents is security. The security concerns for mobile agent applications include and surpass those for traditional distributed applications. This is due to the fact that mobile agents travel from host to host, and in doing so may cross over boundaries that define trust domains. The tightest boundaries are often drawn across the local host itself: processes started on the host by one of its valid users are assumed to be "safe", and allowed the same access rights as the user. This assumption is no longer valid for mobile agent applications, since an agent executing on a host may not have originated on that host. Trust domains could be extended to cover an entire network such as a private Local Area Network (LAN), but in order to support truly distributed applications, agents must be allowed to cross these boundaries as well. Thus, there is a need to protect hosts from potentially malicious agents. Similarly, an agent that originates on a host could assume that its execution environment is safe, but before it begins execution on a remote host, it must ensure that the latter is not malicious. Thus, security for agent-based computation is an important issue, and in fact, the lack of a practical security model for agents has further delayed the adoption of the agent paradigm in distributed applications. The

following section examines the security issues related to agents and agent platforms in a typical agent application.

Security Issues in Mobile Agent Applications

The introduction of code mobility in the mobile agent paradigm gives rise to a number of issues related to host and agent security. Jansen et al. [21] classify security threats into three main categories: disclosure of information, denial of service, and corruption of information, and examine how they apply to an agent system. Farmer et al. [6], in their paper, classify security goals for agent applications into what is impossible, what is easy, and what is possible but not easy. Although a number of different models for agent systems have been proposed [13], for the purpose of this discussion it is sufficient to consider a simple model consisting of the agent and the agent platform/host. An agent consists of a persistent code section that stores instructions, and a data section that stores the agent's state. The latter could further be divided into static data that does not change over the lifetime of the agent, and dynamic data that changes as a result of agent computation as it migrates from host to host. Here, the agent code and static data together will be referred to as the agent's *footprint*. An agent originates at a host, referred to as its *home platform*, migrates to remote hosts to perform its computation, and finally returns to the home platform

with its results. As long as the agent remains on its home platform, it is assumed to be safe from tampering or eavesdropping. This assumption no longer holds true when the agent begins migrating from host to host. The itinerary of the agent may cause it to cross the boundaries of the trust domain that it originated in. For instance, if the agent originated on a host that was part of an internal network behind a firewall, we could reasonably assume that it would be safe as long as it restricted its migration to hosts that were protected by the firewall. This assumption, however, restricts the flexibility and scalability of agent applications. Therefore, when developing a security model for agents, the assumptions about trust must be kept to a minimum. The security requirements for a mobile agent system could be examined under two categories: security for the host and security for the agent. While host security aims at prevention of tampering, agent security aims at detection of tampering. This is because a host is a static entity whereas an agent is mobile. An agent executing on a remote host is vulnerable to eavesdropping and corruption, since the host exercises complete control over the agent and has unrestricted access to its code and data. Consequently, it is easier for an agent to detect tampering, e.g., through the use of cryptographic checksums, than to prevent it.

Host Security

A host provides resources and an execution environment for agents that request its services. Since agents that do no originate on the host could be malicious, a host must protect its environment and resources from tampering by such agents. Some of the issues that must be considered here are agent authentication, agent authorization, code integrity and correctness, and data integrity.

The agent authentication mechanism verifies the identity of the incoming agent, which is merely the identity of the user on whose behalf the agent acts. If authentication was successful, the authorization process ensures that any resources requested by the agent are permitted it as specified by the host's security policy. Further, a host must be capable of recognizing and enforcing additional restrictions placed on the agent by its originator. Code integrity checking ensures that the agent code has not been tampered with by malicious entities, while code correctness proofs establish that the agent code executes under the constraints of the host's security policies [27]. A mobile agent may become malicious if its state is corrupted, even if its code has not been tampered with. Hence, a host may verify the integrity of an incoming agent's state using techniques such as state appraisal functions [7].

All of the above mentioned security measures could be applied by the host before accepting an agent for execution. If the agent fails any of these checks, it may be

denied access to the host's environment. Thus, host security measures are generally pro-active.

Agent Security

Depending upon its application and the sensitivity of data generated, an agent may need to protect its code and data from eavesdropping not only while in transit but also from the hosts it executes on. Some issues related to agent security are host authentication, confidentiality and integrity of agent code and data, integrity of dynamically generated agent data, and secure delegation.

The purpose of host authentication is to ensure that the remote host chosen for migration by an agent is sufficiently trusted to provide it a secure execution environment. A simple way to establish the set of trusted hosts is to store it as static data inside the agent, and authenticate a remote host using certificate-based authentication mechanisms such as Secure Sockets Layer (SSL) [12]. Upon successful authentication, the agent may need to migrate to the remote host over open and insecure network connections. Hence, it is important to keep the contents of the agent confidential to prevent eavesdropping over the network. Further, some measures must be employed to ensure integrity of the agent (detect tampering of its code and data) as it traverses the network. Confidentiality could be achieved by encrypting the agent before

transmitting it over the network, while integrity could be achieved through the use of cryptographic hash functions.

Besides hiding its contents from network eavesdroppers, an agent may sometimes wish to hide the functionality of its code from the host it executes on, for instance, when the agent uses a proprietary algorithm to predict stock rates. In order to ensure code privacy, Sander et al. [30] have proposed computing with encrypted functions and generating encrypted results that are later decrypted by the agent's originator. A weaker form of code privacy is provided by obfuscating agent code [19]. During the course of its execution, the agent may generate data that must be protected from corruption by potentially malicious hosts that it might visit in future. Partial Result Authentication Codes (PRACs) allow an agent to encapsulate intermediate information such that it preserves a property known as *forward integrity* [36].

One of the most important properties of an agent is its ability to clone itself when required. This implies that it must *delegate* or transfer its authority to its clone. The clone in turn must have the capacity to delegate authority to any agents it creates. The transfer of authority must be done securely such that it is difficult for a malicious entity to steal credentials from and impersonate either of the two entities involved in a delegation.

Motivating Example

The following discussion presents an example derived from a paper by Dunne [5] to illustrate a typical application area of mobile agents, namely, resource discovery in a heterogeneous network such as the Internet.

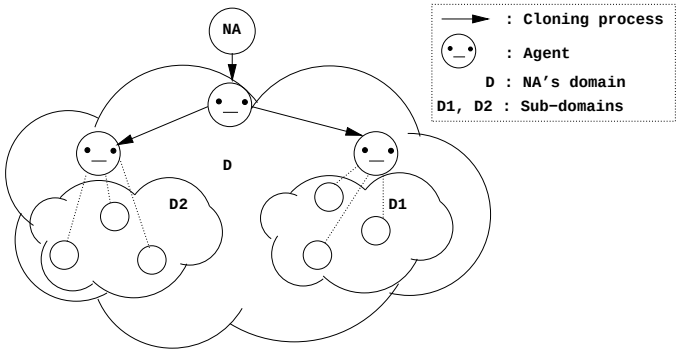

Figure 1.1: Mobile Agents for Resource Discovery

Consider the example network divided into several administrative domains, each with its own usage and security policies, and managed independently of others. As illustrated in figure 1.1, the network administrator (NA) of each domain injects a discovery agent (DA) into the network. The DA's function is to determine resource availability within the NA's domain, hence its itinerary is limited to those nodes that belong to the NA's administrative domain. The DA divides its domain into several sub-domains based on the network topology, clones itself, and designates a clone for each of these sub-domains. The clone's itinerary is limited to the nodes

that belong to its designated sub-domain. The clones subdivide their domains and this process continues till the domains are small enough that no further cloning is required. Whenever a DA visits a node, it updates itself with resource information on that node, and updates the node with resource information about other nodes it has previously visited. If two sibling agents (agents having the same parent) happen to arrive at the same node, the agent that arrived later destroys itself. The clones propagate resource information to their respective parents until complete resource information about the entire domain ultimately reaches the original DA.

The above example presents several scenarios where authentication, authorization, and delegation would be required. In general, an agent created by an entity acts on behalf of that entity. Thus, the DA acts on behalf of the NA. From a security standpoint, the following issues are identified:

- ▷ The discovery agents must be able to prove to a verifying host that they act on behalf of the NA. In other words, a DA must authenticate itself to each node it visits.
- ▷ The DA must be able to securely delegate its authority to any clones it creates.
- ▷ The NA must have some means of restricting the DA's authority to accessing specific files on specific hosts.

In later chapters, as mobile agent security issues are addressed, this example will be revisited to illustrate how security objectives for mobile agent applications could be achieved.

Scope of the Thesis

This thesis presents a security model for mobile agents that primarily focuses on the following issues:

- **Authentication:** A mechanism to authenticate an agent to remote hosts is presented.
- **Authorization:** Means to limit the rights of an agent to exactly those that are required by it to perform its task are discussed. The discussion also considers how an agent could further limit the rights of agents that it spawns or clones.
- **Delegation:** Methods are proposed to securely delegate the originator's authority from the originator to an agent, and from one agent to another. The delegation mechanism proposed could be used to not only propagate existing restrictions to the agent receiving the delegation, but also introduce additional restrictions on it.

Mobile agent applications are generally distributed in nature, since agents roam around a network and access resources on hosts distributed across administrative and trust domains. Therefore, a security infrastructure for the mobile agent environment must take into account both the distributed nature and mobility of the entities involved. The security model proposed in this thesis borrows from the Grid Security Infrastructure (GSI), a security infrastructure for large-scale distributed computing environments, or *Grids*, as they are commonly known. In addition, this thesis addresses the implementation aspects of the model in the context of a specific agent infrastructure, the Distributed Agent Delivery System (DADS). The DADS was chosen because it provides a flexible and modular interface that can support a heterogeneous blend of agents and agent applications. Therefore, the proposed mechanisms for authentication, authorization, and delegation could easily be incorporated as modules within the DADS.

The material is organized as follows. Chapter 2 introduces Computational Grids along with the security requirements of Grid applications. In particular, it focuses on X.509 Proxy Certificates and their use in authentication and restricted delegation for Grid applications. Finally, it presents the motivation for applying Grid security solutions to mobile agents. Following that, Chapter 3 presents a security model for mobile agents based on GSI, that focuses on authentication of agents, restriction

of agent privileges, and secure delegation during spawning of new agents. Next, Chapter 4 discusses the design of the DADS, its agents, and its component modules, and explains how the proposed agent security mechanisms could be incorporated into the DADS as modules. Chapter 5 presents a summary of the thesis along with a discussion of future work.

CHAPTER 2

SECURITY FOR GRID APPLICATIONS

Computational Grids [10] are high-performance distributed computing environments that provide pervasive access to computational resources through large-scale resource sharing among multiple heterogeneous networks. Thus, Grids attempt to provide computing power "on demand" to applications, much like the power grids of today which supply electricity on demand to consumers. The user in effect pays for using computational power but not for the cost of the computing equipment itself. The result is enormous processing power that could be used for complex scientific computation, modeling and visualization experiments, distributed data mining, and other super-computing applications [9].

A Grid application differs from traditional distributed applications in several ways. It is characterized by a large and dynamic user population that shares resources across large heterogeneous networks. The resource pool itself is large and dynamic and may include other hosts in the network, file and print servers, cluster computers, etc. During the course of its lifetime, a Grid application started by a user may acquire and release resources dynamically on several machines. These resources would typically

be situated in different geographic locations of the Grid and belong to different administrative and trust domains. Hence, it is necessary for the user to be authenticated by each of the required resources before accessing them. Due to the dynamic nature of the resource pool and the need for applications to acquire resources continually, it is not possible to establish trust relationships between the user and resources at the start of the computation itself. Further, Grid applications are usually long-lived, requiring several hours or even days to complete. Hence, it is desirable that the user authenticate herself just once at the start of the computation, and have a user process authenticate to resources on her behalf as and when required, without further intervention from her. This property is known as *single sign-on*, and it implies the need for a mechanism to delegate the user's authority to processes acting on her behalf. In addition to single sign-on, delegation is required in situations such as third-party data transfers, where a user may wish to transfer data between two remote hosts, say B and C. Instead of authenticating herself to B and reading data from B, then authenticating herself to C and writing data to C, the user could delegate her authority to both B and C, which could then mutually authenticate each other using the delegated credential and directly transfer data.

In a Grid, users and resources, by virtue of being located in different administrative domains, may employ different mechanisms for authentication and authorization

depending upon the local domain's security policies. These mechanisms could include Kerberos[1], Unix security, Secure Sockets Layer (SSL), Secure Shell [2], etc. Moreover, security policies on local sites are generally inflexible and vary widely in terms of credential requirements from one site to another. It is therefore impractical for a user to employ different mechanisms to authenticate to different resources. What is required is a global mechanism that provides inter-domain access to heterogeneous resources while preserving site-specific security policies. Hence, a security infrastructure for Grid applications must be able to inter-operate with local security mechanisms while providing uniform authentication and authorization mechanisms to Grid users and applications. The Globus Toolkit[3] [8], developed under the Globus Project[4] [15], provides core middleware services such as communication, information infrastructure, data management, fault detection, and security, which are required to support Grid applications. The Grid Security Infrastructure (GSI) [11, 17], which is the security component of the Globus Toolkit, was developed to address the security issues typical of Grid applications. GSI uses Public Key Infrastructure (PKI) and X.509 certificates to provide SSL-based authentication, confidentiality, and message integrity for Grid applications. It also provides multi-site authentication while allowing each site to

[1]™Massachusetts Institute of Technology (MIT), Cambridge, Massachusetts.
[2]®SSH Communications Security, Corporation Finland.
[3]™University of Chicago, Chicago, Illinois.
[4]™University of Chicago, Chicago, Illinois.

define its own security policy and authentication mechanisms. Thus, we could have different sites participating in a Grid application using GSI for inter-site authentication, while continuing to use site-specific mechanisms such as Kerberos, Unix security, etc. for intra-site authentication. Finally, GSI uses *Proxy Certificates* to achieve the essential requirements of single sign-on and delegation for Grid applications. Over the years, GSI has proved to be one of the most successful approaches to providing Grid security. It has been deployed not only in Grid environments, but also in dozens of supercomputers and storage systems, thus achieving a level of acceptance reached by few other security infrastructures [3].

Proxy Certificates

GSI addresses the issues of single sign-on and delegation through the use of X.509 Proxy Certificates, the specification for which is outlined in [32]. A proxy, as the term implies, stands temporarily in place of the original. An X.509 Proxy Certificate (PC) is a temporary credential that allows its owner to represent another entity for a limited time period. The entity that the PC owner stands in for could be either (1) an end-entity such as a user or service that possesses a valid X.509 certificate issued by a Certificate Authority (CA), or (2) an entity such as a process that possesses a valid Proxy Certificate. The user or process that signs a PC is referred to as the

Proxy Issuer (PI). Like all X.509 certificates, a PC has its own distinct public and private key pair. It can sign other PCs, but not any other end-entity certificates.

A PC is derived from its PI's X.509 certificate, and its format is illustrated in figure 2.1. Some of the fields of a PC are indicated here:

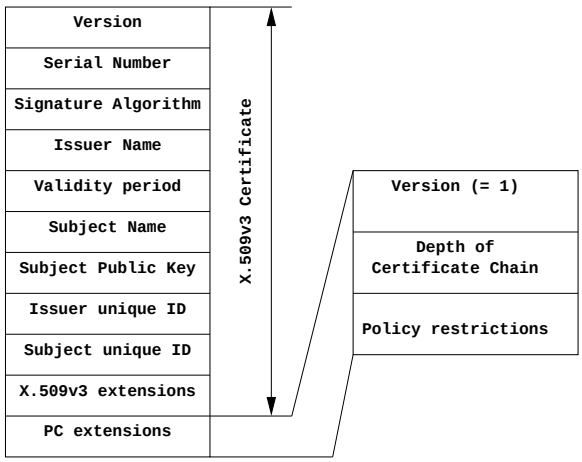

Figure 2.1: Proxy Certificate Format

- ▷ The `Serial Number` field is unique amongst all PCs issued by its PI.
- ▷ The `Issuer` field is the same as the `Subject` field of its PI.
- ▷ The `Validity` period is typically shorter than that of its PI, and is specified at the discretion of the PI.
- ▷ The `Subject` field is its issuer name (or the subject name of its PI) concatenated

with a component that is unique amongst all PCs issued by its PI. This ensures that each PC has a unique identity.

- ▷ The public key in `SubjectPublicKeyInfo` is a part of a newly generated key pair, the private key of which belongs to the PC owner.

- ▷ If the extensions `keyUsage` and `extKeyUsage` are present in the PI's certificate, they are present in the PC as well, and may restrict its key usage further.

- ▷ The `Basic Constraints` extension is present and is set to false to indicate that the PC cannot act as a CA.

In addition, every PC contains a new extension called `ProxyCertInfo` that identifies it as a PC and places additional restrictions on its use.

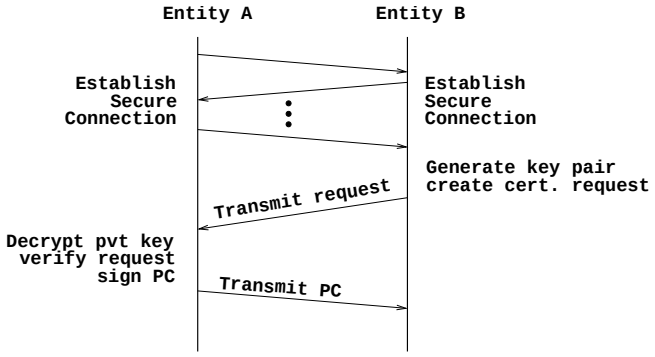

Figure 2.2: Delegation of authority from entity A to entity B

The purpose of creating a PC is to delegate authority from one entity to another. Figure 2.2 illustrates the delegation of authority from entity A to entity B located on different machines in a network. First, A and B mutually authenticate each other with their respective X.509 credentials using SSL and establish an authenticated, integrity-checked channel for communication. Next, B generates a key pair, uses it to create a Proxy Certificate request, and transmits it to A over the channel. Finally, A verifies that the certificate request conforms to the PC profile, optionally places restrictions inside the PC, signs it with its private key, and returns it to B. As a result, B becomes the owner of a proxy credential that represents A.

Authentication

The owner of a proxy credential could use GSI authentication to establish its identity to a verifier. As illustrated in figure 2.3, a PC owner authenticating itself to a verifier using GSI must present to the verifier its entire certificate chain starting from the CA certificate (or the end-entity certificate) to the last PC. The latter could then apply a path validation algorithm similar to the one employed for X.509 certificates [20], the differences to which are outlined in the Proxy Certificate draft [32]. The purpose of path validation is to verify the binding between the PC subject name and public key. Among others, the verifier must ensure that the certificate chain has

a valid depth, and that each certificate in the chain is valid at the current time. Additionally, for each certificate i in the chain, the issuer name of i must equal the subject name of $i-1$, and further, certificate i must be signed by certificate $i-1$. Once the chain is verified, the verifier could issue a challenge using procedures outlined in the SSL protocol [12] to establish that the authenticating entity was indeed the owner of the PC presented to the verifier.

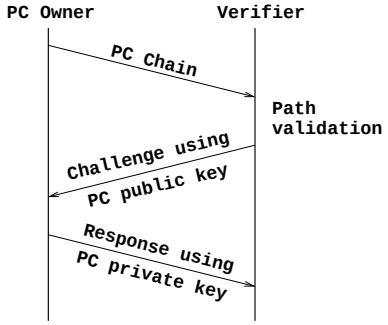

Figure 2.3: GSI authentication using X.509 Proxy Certificates

GSI provides cross-domain authentication through the use of global and local credentials, and a mapping between the two. Specifically, each user requires a Globus credential, which is an X.509 credential issued by a Globus CA, and accounts on each site that the user wishes to access resources on. Each site has a *Globus gatekeeper*, a trusted daemon that authenticates users using GSI. In order to access resources on a local site, the user first authenticates herself to the gatekeeper at that site using

her Globus credential. Upon successful authentication, the user's Globus identity is mapped to a corresponding local identity, such as a Unix user id, that is recognized by the site's security mechanisms, and used for authorization decisions when accessing the site's resources. In addition, the delegation of authority from a user/process A to another process B is mediated through the gatekeeper at process B's site. Thus, the Globus credential is used for single sign-on and delegation, while the local credential is used to enforce site-specific security policies.

Restricted Delegation

An entity that owns a Proxy Certificate could sign additional PCs and delegate its authority as many times as desired: thus user U could delegate her authority to process P, which in turn could delegate its authority to process Q, and so on. In this manner, a chain of authority could be established starting at the end-entity and ending at the entity that was delegated a proxy credential. At each level of delegation, the credential consists of the end-entity certificate and a set of Proxy Certificates. The entity at each level could associate restrictions with the Proxy Certificate that it signs, to indicate the set of operations that the credential owner at the next level is allowed to perform. Thus, PCs could be used for *restricted delegation* of rights.

Every Proxy Certificate contains an extension called `ProxyCertInfo` that identifies the certificate as a PC, and defines any restrictions that the PI may have placed on its use. This extension is also required to ensure proper path validation of a Proxy Certificate chain. Through the `pCPathLenConstraint` field inside this extension, the PI could specify the depth of the certificate chain that the PC owner may sign, and thus limit subsequent delegation of the PC. A depth of 0 indicates that the PC cannot sign any other PCs. The PI could also specify restrictions on the PC's use through the `proxyPolicy` field inside the extension. The purpose of placing policy restrictions inside a PC is to limit the amount of authority delegated to it. The set of rights granted a PC owner is then the intersection of the set of rights in each certificate along the certificate chain leading to the PC. Therefore, when making authorization decisions for the PC owner, a verifier must take into account not only the local host's security policies, but also the restrictions placed inside the PC. The presence of policies inside PCs places the burden of authorization decisions on the verifier, since it is expected to understand the policy language of the restrictions, and is the only entity that can enforce the restrictions specified in the PC.

The end-entity's private key is usually stored in encrypted form on her local machine. In the absence of PCs, single sign-on could be achieved by having a local process authenticate to resources on the user's behalf, but this would require the

user's private key to be stored in decrypted form somewhere in memory so that the process could access it, and thus increase the risk of it being compromised. The use of Proxy Certificates for single sign-on minimizes exposure of the user's private key, since it needs to be decrypted just once, at the time the PC is being created. Beyond this, single sign-on and further delegation do not require access to the user's private key. The PC's private key is now stored unencrypted on the machine, but its compromise does not have as harmful consequences as compromise of an end-entity's key for the following reasons. First, the user certificate is usually long-term, and if compromised, hard to replace. In contrast, a PC has a much shorter validity period and could be easily created by the user without requiring interaction with any third party such as a CA. Second, the authority of a PC could be limited by specifying restrictions on its use through the `ProxyCertInfo` extension. The same extension could also be used to limit the depth of the certificate chain created by a PC. Finally, the `keyUsage` extension could be placed inside a PC to restrict the usage of its private key.

The concept of a proxy credential is not unique to GSI. In a paper by Neuman [28], proxy credentials have been proposed for authorization and accounting in distributed systems. Kerberos [24] also employs the concept of a Ticket Granting Ticket (TGT), a temporary user credential that could be used for single sign-on, and forwardable

tickets that could be used for delegation. However, the creation of a Kerberos TGT involves a trusted third party, the Domain Controller. In contrast, a PC can be created by the end-entity without requiring interaction with any trusted third party. Further, in order to be useful for inter-domain authentication, Kerberos must also be used as the intra-domain authentication mechanism, a condition that is not feasible for Grid applications [3].

The GSI is implemented on top of the Generic Security Services Application Programming Interface (GSS-API) [35] to provide a generic interface for authentication and authorization mechanisms. Currently, GSI uses SSL libraries to provide X.509 certificate-based security services for Grid applications. Implementations of the GSS-API using Kerberos have also been provided.

Grid Computing and Mobile Agents

The distributed nature of Grid applications very closely resembles that of mobile agent applications. Thus, both are similar in many respects as outlined below:

- Both applications are characterized by heterogeneity of participating sites with varying trust levels between constituent sites. Hence, establishment of trust between two entities may need to cross administrative boundaries.

- A mobile agent could be equated to a long-lived process on a Grid, and similar to a Grid process, is characterized by reduced interaction with the user that created it.

- Both agents and processes on a Grid must have the power to act on behalf of the user that created them. Further, they must have the ability to delegate their authority to other processes or agents.

- In a Grid, the user and resource population is dynamic. Similarly, mobile agents (which can treated as both user and resource) have a dynamic population.

- Similar to applications on a Grid, mobile agents may acquire resources, create other agents, and release resources dynamically.

- Both Grid environments and mobile agent environments are designed to be scalable and capable of large-scale deployment of resources.

Due to the similarity in Grid environments and mobile agent environments, it is reasonable to conclude that the security infrastructure for Grids could be translated to the mobile agent environment. Special attention, however, must be paid to the factors that make the latter different, namely, mobility of the entities involved. The following chapter outlines mechanisms using Proxy Certificates that address several important security issues in the mobile agent environment.

CHAPTER 3

MOBILE AGENT SECURITY USING PROXY CERTIFICATES

A typical mobile agent application – such as the one illustrated in Chapter 1 – requires several agent-host and agent-agent interactions that may take place over a hostile communication channel between entities from different trust domains. Hence, it is necessary to secure both agents and hosts by providing basic authentication and authorization mechanisms that could be used for these interactions.

For most systems, decisions about resource access - namely, who might access a resource and to what extent - are based on the identity of the principal making the request. This requires that the principal possess a credential and some secret known only to the principal that could be used to identify it to the resource. Examples of such credentials are a login/password pair for password-based authentication, a Ticket Granting Ticket (TGT) for Kerberos, an X.509 certificate and a private key for Secure Sockets Layer (SSL) authentication, etc. Similarly, a mobile agent must possess a credential that allows it access to resources on remote hosts. Further, the agent must delegate this credential to any new agents that it creates. A mobile agent does not have an identity of its own, but acts on behalf of another entity. Therefore, at

all times, the credential must identify the principal that the mobile agent represents.

In a typical agent application, the entity that an agent acts on behalf of could be any one of the agent's creator, originator, or home platform. The creator is the entity that "manufactures" the agent, including its code, static data if required, and proof of correctness functions. For instance, the creator could be a software firm specializing in producing agent applications. The originator is the entity that injects an agent into the network. Typically, the originator (or the organization she belongs to) would purchase a mobile agent application from a creator and deploy it in their network. The agent's home platform is the host it originated on and is considered the most secure and trusted environment for agent execution. Each of these entities must have means of uniquely identifying themselves, such as an X.509 certificate issued by a trusted Certificate Authority (CA). Of these three entities, the host, however, is inanimate and hence cannot be held responsible for the actions of the agent. In many cases, the creator too cannot be held responsible for the actions of the agent, but can at best vouch for the correctness of the agent. Any harm caused by the agent (due to it becoming malicious, for example) is ultimately the responsibility of the entity that chooses to deploy it. Therefore, it is the originator that is responsible for the agent and its actions, and hence for the purpose of this discussion it is assumed that the agent acts on behalf of its originator. Viewing the agent in this manner has

some other advantages: the originator could customize the agent – e.g., change code parameters – before deploying it in an agent system. Further, in routing and resource management applications, the originator would typically be the network administrator of a local network where the agent is deployed, and is likely to have higher privileges than the creator. Hence, agents carrying their originator's credentials would have higher privileges than those carrying their creator's credentials, and could be used to perform privileged operations on the hosts they visit, such as updating the host's routing table.

For the following discussion, it is assumed that the agent's originator, as well as every host in the network that the agent is likely to visit, possesses a valid X.509 certificate issued by a trusted CA. This thesis proposes the use of X.509 Proxy Certificates (PCs) as credentials for mobile agents. A PC would allow its owner, the agent, to identity itself as representing its originator for authentication purposes, and at the same time facilitate the creation of new PCs for any clones that it might create. Additionally, through the values stored in its fields, a PC could be used to realize various other desirable security features in the mobile agent environment:

1. A mobile agent is created and deployed in a network for a specific purpose that lasts a finite amount of time, typically several hours. Hence, it is desirable to limit the lifetime of the agent to exactly the period of time required for its task.

This limits the damage caused by a stolen credential to the time for which it is valid. The lifetime of a PC could be specified through its `Validity` field that lists the start and end time during which it is valid. The validity period of a certificate chain is the intersection of the validity period of individual certificate inside the chain.

2. The `subject` field contains the identity of the agent, which is a concatenation of its issuer name and a unique field, thus making the agent uniquely identifiable. This is useful in situations such as logging an agent's activities on a host. Also, the agent's identity along with associated timestamp information could be used to leave trace information on a host, so that agents visiting the same host in future could be made aware of the last visitation time of an agent on the host, and clone or merge as required. The agent's identity could also be used to enforce policy restrictions on it. For instance, the originator may create an agent and propagate its identity, along with restrictions on resource access that apply to the identity, to other hosts in the network. In addition, a unique identity makes it possible to specify the agent's PC in a Certificate Revocation List (CRL). An agent itself could use another agent's identity as the basis for establishing agent-agent communication.

3. The `keyUsage` extension could be used to limit the usage of the PC's key. For instance, if the Proxy Issuer (PI) wished to restrict usage of the PC to authentication alone, a corresponding value for this field may be assigned.

4. The `pCPathLenConstraint` field inside the `ProxyCertInfo` extension allows an agent's originator to restrict the depth of the descendent tree rooted at the agent. The agent itself may use this field to restrict the level of delegation that its child agent may perform. Thus, if the originator creates an agent with a `pCPathLenConstraint` of 1, that agent could create clones, but the clones themselves would not be able to create other agents. The path validation algorithm for PCs must ensure that the depth of the certificate chain is consistent with the value of the `pCPathlenConstraint` field for each certificate in the chain.

5. The `ProxyPolicy` field inside the `ProxyCertInfo` extension contains restrictions on the PC owner's authority, and could be used by a host in conjunction with its own security policy to determine the privileges that the PC owner (the agent) is allowed. The absence of this field indicates an unrestricted proxy, implying that the proxy owner has the same rights as the Proxy Issuer. This field is discussed in later sections.

In addition to the above-mentioned fields, a new private extension for Proxy Certificates called `AgentInfo` is proposed having the following format:

```
AgentInfo ::= SEQUENCE{
    agentHash           AgentHash,
    delegateInfo        DelegateInfo }

AgentHash ::= SEQUENCE{
    algorithm           AlgorithmIdentifier,
    value               BIT STRING }

DelegateInfo ::= SEQUENCE{
    hostCertificate     Certificate,
    hostSignature       Signature }
```

The `AgentInfo` field is created by the PI with the co-operation of the host on which delegation takes place. The following fields are defined:

`agentHash` contains a cryptographic hash of the agent's footprint(i.e. code and static

data). It serves as a binding between the PC owner (the agent) and the PC itself. This field is verified by a host during the authentication process.

`delegateInfo` contains the `hostSignature` field, which is a signature using the private key of the host on which delegation took place. The signature is computed over the host id, the time at which delegation took place, and a cryptographic hash of the agent's footprint, and may be verified using the host's public key stored in `hostCertificate`. The `delegateInfo` field could be used by other hosts to verify the identities of the hosts through which the agent was delegated, and to ensure that delegation took place on trusted hosts only. This field is similar to the `delegationTrace` extension proposed in an earlier version of the X.509 Proxy Certificate draft [33].

The `AgentInfo` extension is not critical, but applications that rely on this extension to make authentication decisions for agents may reject an agent whose PC does not contain this extension. The use of this extension for authentication and delegation is detailed in further sections.

The security issues in a mobile agent application arise as a result of various interactions between agents and hosts, starting from the time an agent is created to the time it returns to its home platform. At the time of agent creation, the originator

must temporarily delegate her credential to the mobile agent and specify restrictions on it. During migration to a remote host, the agent must authenticate itself to the remote host. When executing on the remote host, the latter must verify that each resource requested by the agent is allowed it as per the host's policies and the agent's restrictions. During the course of its execution, the agent may choose to clone itself, which requires that the clone be delegated the agent's credential and optionally acquire further restrictions on its authority. In the following sections, the resource discovery example from Chapter 1 is revisited in order to explain how Proxy Certificates could be used to secure these interactions.

Agent Creation

As illustrated in the example from Chapter 1, the network administrator (NA) creates a discovery agent (DA) for the purpose of resource discovery. It is assumed that the private key corresponding to the NA's X.509 certificate is encrypted with a pass-phrase and stored on her workstation H1 in a local directory accessible only to the NA and processes started by her. In order to create the DA and grant it a temporary credential to represent the NA, the NA generates a Proxy Certificate from her X.509 certificate containing the following fields:

- ▷ The `subject` is the NA's Distinguished Name (DN) from her end-entity certificate, concatenated with a unique number.
- ▷ The `validity` period is set to the time for which the DA is expected to be active, say 12 hours. At the end of this period, a new Proxy Certificate must be generated if required.
- ▷ The delegation depth (`pCPathLenConstraint`) is left unspecified, allowing unlimited delegation.
- ▷ The restrictions stored in `proxyPolicy` limit the DA's authority to accessing network and user-specific system files on the hosts in its domain.
- ▷ A hash of the DA is stored in `agentHash`, thus binding the DA to its PC.
- ▷ The NA engages in an exchange with the local host H1, that results in the creation of the `delegateInfo` field containing H1's signature on its (H1's) identity, the current time, and a hash of the DA. This exchange is detailed in a later section.

The NA supplies a password to decrypt her private key and signs the PC generated above. The PC along with its unencrypted private key is stored on H1 in the NA's local directory. The NA then injects the DA on the host H1.

Agent Migration

The function of the DA is to gather resource information about its domain. Thus, the DA is required to migrate from host to host to perform its function. Before migration to a remote host can occur, the DA must authenticate itself to the remote host.

Agent-to-host Authentication

An unprotected mobile agent's code and data are open to corruption by malicious hosts or network eavesdroppers, hence a host that allows a remote agent access to its environment must be presented proof of the agent's identity and integrity. In the mobile agents example, the DA represents the NA in all its actions, hence, the DA must authenticate itself to a host *as the NA*. Furthermore, the DA must present proof to the host that it has not been tampered with. The DA could authenticate itself to a remote host using SSL. However, SSL cannot be used with standard X.509 certificates in the agent-host authentication scenario, since this would require the agent to possess the private key of its originator. Equipping the agent with this key is dangerous, since it allows any malicious platform or network eavesdropper to extract the key from the agent, thus invalidating the security of the originator's private key. In fact, it is almost impossible to store a key securely on an agent [6]. Certain methods such as

encrypted functions [1, 4] could be used to protect the privacy of the originator's key, but the solutions proposed are only of theoretical interest because of their lack of efficiency.

An agent cannot carry keys with it: however, it is still possible to prove that it acts on its originator's behalf. One way of doing this is to have the agent's originator apply a hash function on the agent, sign the hash with her private key, and include the signed hash along with her certificate in the agent's data area. A verifying host would decrypt the signed hash using the public key of the originator, apply the same hash function on the agent that the originator used, and compare the two hashes. A match would establish that the originator did indeed sign the agent, hence it is reasonable to conclude that the agent represents its originator. Further, it proves that the agent has not been modified in any way inconsistent with its functionality and thus establishes its integrity. An agent comprises code and data sections, of which the footprint(code and static data) does not change over its lifetime, hence proving its integrity is straightforward and follows from the discussion above. Proving integrity of non-static data, however, is much harder, and is by itself an active research area [22]. This document does not concern itself with non-static data integrity issues.

The method described above establishes the agent's authenticity as a consequence of verifying its integrity. While this approach to authentication does not require the

agent to carry the private key of its originator, it has some disadvantages:

1. The signed hash of the agent serves as its credential. Since the agent does not possess keys, it does not have the power to generate new credentials on its originator's behalf. Hence, it cannot create new agents on-the-fly that differ from the original either in code (different functionality) or data (restricted functionality). This greatly reduces the power and autonomy of the agent.

2. When an agent clones itself, the clones inherit the signed hash from their parent. Since the agent and all its clones possess the same digest, it is impossible to securely distinguish an agent from its clones. A unique agent identifier is necessary for activities such as communicating with other agents, merging of agents, etc.

Agent authentication using Proxy Certificates can overcome the shortcomings of the authentication mechanism outlined above. Authentication using PCs is a two-step procedure: first, the agent proves knowledge of its PC's private key to a remote host. Second, it transmits itself to the remote host so that the latter may verify the binding between the agent and its PC. As an example, consider the interaction between the DA on host H1 authenticating itself to a remote host H2, illustrated in figure 3.1:

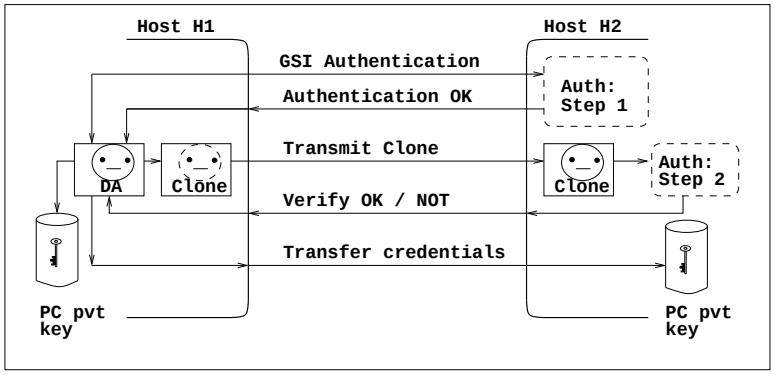

Figure 3.1: Agent-to-host Authentication

1. The DA with its proxy credential and H2 with its X.509 credential use the Grid Security Infrastructure's certificate exchange and challenge-response phase to authenticate each other (Chapter 2). During this step, H2 performs path validation on the DA's certificate chain consisting of the NA's certificate and the DA's PC. Further, the DA and H2 establish a session key to create a secure communication channel between them.

2. The DA clones itself and transmits the clone over the channel to H2. H2 computes a hash over the clone and verifies that it matches the `agentHash` field inside the DA's PC. If it does, the DA stands authenticated and H2 sends a positive response to the DA, otherwise it sends a negative response and terminates the connection.

3. Upon receipt of a positive response, the DA uses the session key established during step 1 to encrypt its credential, including its certificate chain and private key. It then transfers the same to H2, where it is stored in a directory accessible to the DA's identity alone.

4. The DA has now essentially migrated to H2. It therefore destroys its proxy credential on H1, and then terminates itself. The clone on H2, being an exact copy of the agent on H1 and possessing the DA's credential, could function as the original DA and resume execution on H2.

The above procedure (steps 1–4) is repeated each time the DA migrates to a new host. Additionally, during step 1, the host could verify the `delegateInfo` field for each PC in the agent's certificate chain, as explained in the next section. During agent migration, the agent's credentials are transferred from one host to another over an encrypted channel, and hence are safe from network eavesdroppers. However, the remote host that the agent migrates to has complete knowledge of the agent's credential. Thus, there is always the possibility that it may misuse the credential and create malicious agents impersonating the original. In the example, assume that host H2 is malicious. Since H2 has access to the DA's proxy credential, it could create a malicious discovery agent DA2 possessing the same credentials and hence the same

set of rights as the original DA, and use it to propagate false resource information in the network. The `agentHash` field, by dint of containing a cryptographic hash of the legitimate DA, guards against this possibility. Therefore, a malicious agent such as DA2 is prevented from impersonating the DA since the former would fail step 2 of the authentication process outlined above. Thus, the presence of the `agentHash` field ensures that no entity apart from the agent that owns the PC could be authenticated using the PC. In addition to authentication, the DA's integrity is established during step 2.

Agent execution on Remote hosts

Once an agent such as the DA in the agents example migrates to a remote host such as H2, it begins execution on the host. During the course of its execution, the agent may require access to the host's resources such as its file system, operating system services, etc., which requires that the host authorize these resource requests. Further, the agent may create new agents or clone itself, which requires a delegation of authority from the agent to its child agent.

Agent Authorization

Authorization is the process of ensuring that the agent performs only those operations that it is permitted. This requires that the host have knowledge of the actual set of operations the agent is allowed, and verify each resource requested by the agent against this set. The simplest method of performing these checks is to grant the agent the same set of rights as its originator, and leave the task of authorization to the operating system. For example, assume that the NA has a local account on each of the machines in the domain of the DA, and that when the DA travels to H2, it queries the routing table of H2. The DA would be granted access to H2's routing table only if the NA herself had access to it. This check could be performed by the operating system on H2 using the security policies pertaining to local user accounts.

The disadvantage with this approach is that an agent created by an originator, by dint of acting on its originator's behalf, may have more rights than the latter intended it to have. Granting an agent more power than is strictly required to complete its task is dangerous, especially since agents and their credentials are prone to compromise by malicious entities. For instance, assume that the NA had root privileges on all hosts in the DA's domain. In the absence of restrictions, the DA would automatically have root privileges on each host it visited, even though all it may require is access to certain system files. If the DA became malicious, it could use its root privileges to

compromise every host in its domain. The amount of damage could be contained by limiting the validity period of the agent's proxy credential, but it would be desirable to have a means of limiting the rights of the agent itself.

One way to achieve this is for the originator to associate a set of rights with an agent at the time of its creation, and store it as static data inside the agent. When an agent visits a host and requests resources, the host must consider the restrictions stored within the agent, in addition to its own local policy, in order to make an authorization decision for the agent. This approach permits the originator to create agents with varying rights to perform varying tasks, and serves to further limit the damage caused by malicious agents. However, once an agent has been created, all its descendents would have the same rights as the original agent. Since the original rights are included in the agent's static data, they cannot be altered without destroying the agent's integrity. It would be desirable, however, for an agent to have some means of restricting the rights of its clones and of other agents it might create dynamically. Proxy Certificates can be used for this purpose.

A PC generally inherits the rights of its PI. However, the PI could specify restrictions on the PC through its `proxyPolicy` field. The restrictions must be encoded in a policy language that is understood by the both the entity that specifies (i.e., the NA) and verifies (i.e., host H2) the restrictions. For example, assume that the NA

has read, write, and execute permissions on a file F1 on host H2, and wishes to grant the DA read and execute permissions for the same. The restriction for the required right could be of the form `DENY F1@H2 WRITE`, meaning that write permission on F1 is denied the agent. Similarly, the DA could create another agent DA1 and grant it read-only access to F1 by specifying a restriction of the form `DENY F1@H2 EXECUTE` inside DA1's PC.

The policies stored in a PC are extracted by the remote host to make authorization decisions. For each resource requested by an agent, the host must verify that the request satisfies the following two conditions:

–c1– Is the agent's identity (i.e. its originator) allowed the operation as specified by the operating system's security policies?

–c2– Has the agent's PI allowed the agent the operation, based on the restrictions inside the agent's PC?

In addition, if the agent's PI is itself a PC, it too must satisfy the conditions listed above. In other words, the rights associated with a PC are the intersection of the set of rights specified in every PC along the PC owner's certificate chain. These rights, in conjunction with the host's local security policies, determine the final set of rights that the agent possesses. When the agent clones itself or creates a new agent, it could

further limit the authority of the newly created agent by including policies inside the latter's proxy credential as well. Thus, with each level of delegation, the authority of the agent could be further restricted if required. An agent cannot delegate more authority than it possesses, but even if it does specify additional rights that it does not possess, the verifying host ensures through the process described above that these rights are disregarded while making authorization decisions.

Delegation

Credential delegation is the process by which an entity transfers its identity to another. The DA in the agents example creates several child discovery agents, which in turn create new agents as required. Each new agent created must have the capacity to authenticate itself as the NA. However, it is desirable that each agent be uniquely identifiable, and further, that the parent agent be able to associate a subset of its rights with the child's identity. An entity such as the DA possessing a PC could delegate its authority to another entity such as its clone DA1 through the credential delegation process outlined in Chapter 2. This provides the clone with its own PC that uniquely identifies it, and that contains restrictions placed on its authority by its parent.

Although credential delegation is a powerful tool, it could be misused by malicious

entities. Suppose that the DA, upon migrating to host H2, creates a clone DA1 and delegates its authority to DA1. This scenario could not be distinguished from one where the host H2 was malicious, created a malicious discovery agent DA2, and used the DA's PC along with its private key to create a new credential for DA2. This is because the process of delegation involves a signature by the entity, i.e. the DA, that owns the proxy credential. When a mobile agent performs delegation on a remote host, the private key of its credential is known to that host as well. Thus, the host could easily impersonate the DA and create malicious agents without the DA's knowledge.

In order to make delegation more secure, the concept of trusted hosts for delegation is introduced. A trusted host is one that is known to be "safe" for agent execution and is known to not be compromised. In a network of several hundred hosts, a few tens of hosts such as firewalls, routers, etc., could be deemed trusted. The set of trusted hosts could be stored as system-wide data accessible to every entity in the network, or could simply be established at the discretion of a verifying host. In order to ensure secure delegation, the following condition is imposed upon the mobile agent environment:

A mobile agent may delegate its authority to other entities on trusted hosts only.

The `delegateInfo` field inside a PC supports enforcement of the above rule. At the time of delegation, the following interaction takes place between the mobile agent and its host:

1. The agent conveys to its host a hash of the agent that it wishes to delegate its authority to. This step does not require any authentication checks since these are done prior to agent execution. Once the agent begins execution on the host, it is deemed trusted.

2. The host verifies that the agent's PC (obtained during the authentication phase)

   ▷ contains a valid (> 0) value for `pCPathLenConstraint`,

   ▷ has the ability to sign other PCs as specified by the `keyUsage` extension if present, and

   ▷ is valid at the current time.

3. The host then uses its private key to sign the host id, the current time, and the agent hash from step 1. The signature, along with the host's certificate, is returned to the agent.

4. The agent generates a PC for the agent it wishes to create, includes the host signature and certificate in the `delegateInfo` field, and signs the PC.

Thus, in the example, if the DA creates a clone DA1 on host H2, the `delegateInfo` field for DA1's PC would contain H2's certificate, along with H2's signature on its identity, time of creation of DA1, and DA1's hash. When DA1 authenticates itself to a remote host, the latter would verify the `delegateInfo` field inside each PC belonging to DA1's certificate chain as follows. First, the remote host decrypts the `hostSignature` field using the public key stored inside `hostCertificate`. Next, it verifies that:

- ▷ the host id in the decrypted signature matches that inside `hostCertificate`, and further, that it is the identity of a trusted host.
- ▷ the hash inside the decrypted signature matches the `agentHash` field inside the PC.
- ▷ the timestamp inside the decrypted signature is within the validity period of the PC. This prevents any replay attacks by a malicious host.

The verification of the `delegateInfo` field is not necessary for the first-level PC, since the first level of credential delegation from the originator to the mobile agent created by her is always assumed to have taken place securely. This is because the mobile agent's PC is signed using the originator's private key, which is stored encrypted on the originator's local host with a pass-phrase known only to her. Therefore, it

is difficult for a malicious entity to steal the originator's private key and create mobile agents at the first level of credential delegation. Thus, the first-level PC can be considered as secure as the originator's X.509 credential.

The discussion assumes that the agent itself is not malicious, but this is a valid assumption since malicious agents are likely to fail authentication and other integrity checks before they could even begin execution. The proposed delegation mechanism makes it much harder for malicious entities to impersonate a trusted host as long as the trusted host itself is not compromised.

The security of the delegation mechanism is dependent on the security of the trusted hosts themselves. If a trusted host is compromised, all the agents created on that host are suspect, since they may have been created by a malicious entity on the compromised host. The risk of compromise could be reduced by applying safeguards such as firewalls, intrusion detection systems, etc., to protect the trusted hosts, but it is impossible to prevent all attacks. In such cases, additional safeguards such as the validity period, restrictions, key usage, delegation depth, etc., that are built into Proxy Certificates could further reduce the damage caused by a compromised credential on a host. Further, if a host A is known to be compromised, all other hosts in the network could refuse to accept agents that have been delegated through the host A.

In this chapter, various mechanisms that are essential to the secure operation of mobile agent applications have been presented. Each of these mechanisms, when implemented in the context of an agent infrastructure such as the Distributed Agent Delivery System (DADS), could be used to provide basic security features for mobile agent applications. The following chapter describes the operation of these security mechanisms, and their interface with the DADS, when implemented as modules within the DADS framework.

CHAPTER 4

DESIGN AND IMPLEMENTATION

The Distributed Agent Delivery System (DADS) is being developed as part of an ongoing research project on mobile agents at the University of North Texas [1] (UNT). The motivation for the DADS arises from the fact that mobile agents may differ in their requirements of their corresponding agent platform, based on the application that they are designed for. For instance, an e-commerce application may emphasize on security, hence a mobile agent designed for e-commerce must be able to perform authentication, encryption, etc. Similarly, data transfer applications may require that agents compress data before transmitting it in order to make transfers more efficient. The functionality required by an agent must be supported by its underlying infrastructure, and in addition, it is desirable that the infrastructure adapt to the varying needs of agents and agent applications. The DADS is one such agent infrastructure that provides an extensible and customizable platform for agent applications. It is influenced in its design by other agent infrastructures, namely AgentTcl [16], TACOMA [25], and Mole [2].

[1] The description of the DADS that follows is based on Cliff Cozzolino's master's thesis at the University of North Texas.

The DADS is organized into a daemon process and a set of loadable modules, which may be used to provide a variety of services such as language availability, security, and fault tolerance. The DADS daemon performs the functions of listening on a standard TCP port for incoming agents, and managing the set of loaded DADS modules. A module is a child process started by the main DADS daemon, that communicates with the daemon using interprocess communication mechanisms. It receives input from the DADS, processes it, and writes the results back to the DADS. Further, modules may be *chained*, such that a module's output that is sent back to the DADS serves as input for another module. Modules may be loaded through a plug-in style interface, making them available to agents as and when required. The DADS is capable of supporting multiple modules for a single service, such as multiple language interpreters, multiple security mechanisms, etc., at the same time. Hence, the DADS can support multi-lingual agents that are not restricted to a particular design such as a specific security mechanism. In general, modules allow the DADS to support heterogeneity in agents and agent applications without having to change the underlying DADS core.

DADS agents, borrowing from TACOMA agents, act as containers for three distinct segments: a code segment that stores instructions, a data segment that stores the agent's state, and a properties segment that describes the agent's code and data

segments. While the code and data segments may contain arbitrary sequences of bits, the properties segment has a restrictive format consisting of three distinct elements: Property ID, Property Name, and Sub-properties, together referred to as a *property structure*. The Property Id is a standardized number that refers to a particular service such as language or security. The Property Name is a string that describes the Property Id, and the Sub-property field is a list of property structures that further describe the particular property. Every agent contains a root *Agent* property, which in turn may contain sub-properties for each of *Language*, *Security*, etc. The *Security* sub-property may contain a sub-property for *Authentication*, which in turn may contain sub-properties for *PKI*, *Kerberos*, etc. Thus, the properties segment stores the description of an agent in the form of a hierarchical tree called a *property hierarchy*. Similarly, the DADS on a host contains a property hierarchy rooted at the *Host* property, that describes its modules and the services it offers. Loaded modules are registered as subtrees in the host's property hierarchy. For instance, an authentication module for Proxy Certificates may be registered in the subtree
Host → Authentication → PKI.

The presence of the property tree on agents and hosts is essential to the agent migration process. Since the DADS is capable of supporting a broad range of functionality for agents, it is likely that the DADS on different hosts will offer different

services to agents. Hence, a migrating agent must ensure that the remote host it migrates to can support its requirements. Similarly, the DADS on each host must verify that an agent satisfies certain requirements such as security, before accepting it for execution. The DADS facilitates this decision making process via the Transfer Protocol (TP), which defines the exchange of property trees between a migrating agent and the host it migrates to. The TP consists of three phases. Phase (1) occurs when the agent connects to a remote DADS daemon and transmits to it a subset of its property tree describing its requirements. If the DADS can support the agent's requirements, it enters phase (2) of the protocol, where the DADS transmits its requirements to the agent. Finally, if the agent supports the DADS's requirements, the TP moves into phase (3), where the entire agent is transmitted to the remote host. At this point, the DADS may invoke an authentication module if specified by the agent or the DADS, to facilitate authentication between the two entities.

Thus, the DADS allows for the development of a customizable and modular agent infrastructure that supports a variety of agent languages and security services. The following section describes the implementation aspects of the proposed mechanisms for agent authentication, authorization, and delegation, in the context of their interface with the DADS. Further, Application Programming Interfaces (APIs) that

facilitate the development of the proposed mechanisms are examined, and some pertinent issues regarding implementation of modules for each of the above mechanisms are discussed.

Implementation Details

In order to realize the proposed agent security model, mechanisms are required to create proxy credentials with restricted rights, authenticate agents using their proxy credential, and perform authorization checks for agents based on their proxy credential. The Grid Security Infrastructure (GSI) implementation, currently at version 2.2, provides extended functionality to support all these requirements. It is available as open source software, and hence can readily be adapted to the implementation of a security model for mobile agents.

The process of agent creation involves a credential delegation step, during which the originator delegates her X.509 credential to the mobile agent. The agent's Proxy Certificate (PC) along with its private key is then stored as a file on the local host. The GSI implementation provides functionality to generate a proxy credential from a user's X.509 credential, and in addition, associate restrictions and other certificate extensions with the delegated credential. This functionality could be used by an agent's originator to generate a PC containing an `AgentInfo` extension specific to

each agent, and optional restrictions on the agent's authority. The agent itself must be able to access its credential on the local host for authenticating to remote hosts, and for destroying the credential when migrating to a different host. This requires that the originator store some information in the agent's data section, so that the agent may reference itself to determine the path to its credential. One possibility, as illustrated in figure 4.1, is to store a hash of the agent's identity (i.e. the subject Distinguished Name in its PC) in its static data, and construct a path to the credential using this hash. This ensures unique filenames for credentials across all agents.

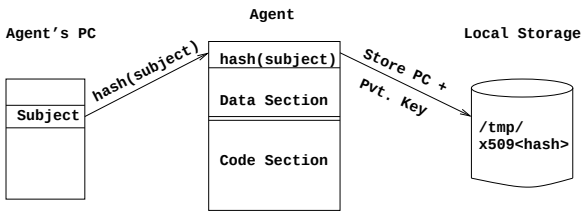

Figure 4.1: Generating a filename for a Proxy Credential

Agent-to-DADS Authentication

Once the agent is injected on the local host, it may be required to authenticate to a remote host in order to migrate to it. As outlined in Chapter 3, agent authentication is a two-step procedure: first, the agent proves knowledge of its private key, and then the host verifies the binding between the agent and its Proxy Certificate. Therefore,

the agent authentication mechanism is implemented as two modules within the DADS. As illustrated in Figure 4.2, agent authentication consists of the following steps:

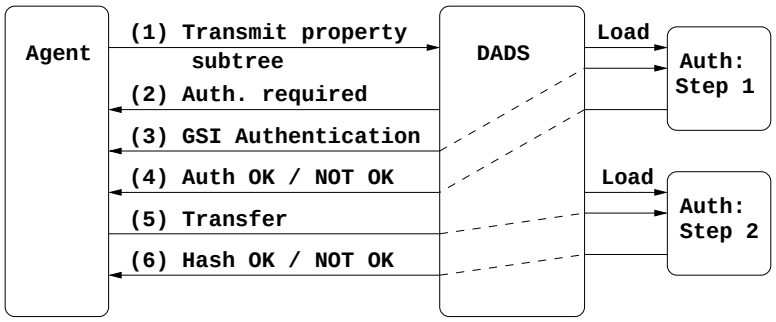

Figure 4.2: Agent-DADS Authentication

1. The agent opens a connection to the remote DADS daemon and transmits to it a part of its property tree containing its requirements for language, security, etc.

2. The DADS replies with a message that contains its requirements, in this case authentication, and subsequently invokes the two-step authentication module.

3. The authentication module uses GSI to establish the identity of the remote agent (step 1). All communication with the remote agent is routed through the DADS daemon.

4. The authentication module returns to the DADS an OK / NOT OK response

to step 1 of authentication, which in turn transmits the response to the remote agent.

5. If the agent receives a positive reply, it clones itself and transfers the clone to the remote DADS.

6. The transferred clone is sent to the integrity checking module, which implements step 2 of authentication. This module uses the agent's credential obtained in step 1 of the authentication phase to verify the agent's integrity against its certificate.

7. The module send an OK / NOT OK response to the DADS, which transmits this to the remote agent waiting on the migration step.

8. If the remote agent receives an OK, it may terminate itself or migrate to another host as desired. If it receives a NOT OK reply, it may select another host in its itinerary to migrate to, or report the problem to its home platform.

In the above procedure, the exchange of property trees between the agent and the remote DADS is defined by the Transfer Protocol (TP). However, the TP does not concern itself with the specifics of the authentication mechanism, which is instead defined by the authentication protocol. The two-step agent authentication procedure

is merely an extension of GSI authentication, and as such, may be implemented using GSI libraries.

Implementation using the GSS-API

The GSI provides an implementation of the Generic Security Services API (GSS-API) [26, 35] using *OpenSSL* libraries. The GSS-API provides a generic interface for security services to its callers. An application may use the GSS-API to avail of security services such as authentication, message integrity, etc., without concerning itself with details of the underlying security mechanism. This facilitates portability, since applications may be written without a specific security mechanism such as Kerberos, PKI, etc, in mind. In order to use the GSS-API, an application follows the procedure outlined below:

1. The application may first acquire credentials that enables it to establish its identity to authenticating peers.

2. It then initiates the establishment of a security context with its peer. This is a multi-step exchange, during which the context initiator is authenticated to the context acceptor. The initiator may optionally require the acceptor to authenticate itself, and may allow delegation of its authority to the acceptor.

3. Once the security context is established, per-message services may be invoked by either party to encrypt and apply integrity checking on data transferred between the two parties.

4. Upon completion of the session, the security context is deleted by the respective applications.

Credential management routines	
gss_acquire_cred	Acquire a handle to pre-existing credentials
gss_release_cred	Destroy an existing GSS-API credential
Security context routines	
gss_init_sec_context	Initiate a security context with a peer
gss_accept_sec_context	Accept a security context initiated by a peer
gss_delete_sec_context	Destroy an existing security context
Per-message routines	
gss_get_mic	Generate a cryptographic MIC for a message
gss_verify_mic	Verify a MIC against a message
gss_wrap	Attach a MIC to a message and optionally encrypt it
gss_unwrap	Optionally decrypt a message and verify its MIC

Table 4.1: Core GSS-API routines

Table 4.1 summarizes routines in the GSS-API that facilitate credential acquisition, establishment of security contexts, and per-message security services. The `gss_acquire_cred` routine provides a calling application a handle to a pre-existing credential, based on the name of the principal whose credential is required. An existing credential may be freed using the `gss_release_cred` routine. A context

initiator (i.e. client) may invoke the `gss_init_sec_context` routine to establish a security context with a context acceptor (i.e. server). The `gss_init_sec_context` routine returns a token that is transmitted to the acceptor, and presented to the `gss_accept_sec_context` routine invoked by the acceptor. The resulting token is in turn transmitted to the initiator. The context establishment routines are invoked in a loop and the resulting tokens transmitted to the peer, until a security context is established between the two. The context initiator may request peer authentication and may delegate its credential to the acceptor by setting appropriate flags in the parameter list of `gss_init_sec_context`. Once a security context is established between the peers, the `gss_get_mic` routine may be invoked by either peer to generate a cryptographic Message Integrity code (MIC) for a message. This routine returns a token that may be transmitted to the peer along with the message, and may be verified by the peer using the `gss_verify_mic` routine. Similarly, the `gss_wrap` routine may be used to attach a MIC to a message and optionally encrypt the message. The corresponding function, `gss_unwrap` is used by a peer to convert a message processed by `gss_wrap` back to its original form. This involves optionally decrypting the message, and verifying its MIC. Finally, the `gss_delete_sec_context` routine may be used to discard an existing GSS-API security context.

In addition, extensions to the GSS-API have been proposed [31] in order to facilitate credential delegation at any time and not just at the time of context establishment. The proposed extensions also allow the initiator of the delegation to associate restrictions with the delegated credential. Support for these extensions has been provided in the current release of the GSI (release 2.2).

A module designed for authenticating mobile agents may make use of the GSS-API implementation provided by the GSI libraries. In particular, the module may invoke routines to establish a security context with the remote agent. The agent in turn may initiate a security context with the remote DADS and authenticate itself. Further, the agent may encrypt its credential before transferring it to the DADS. The transferred credential, including the agent's certificate chain and private key, is stored by the remote DADS on its local storage. The location at which the credential is to be stored is inferred through a hash of the agent's identity stored in its data section, as explained earlier. The agent's Proxy Certificate chain obtained during authentication must be retained by the DADS for use in subsequent authorization and delegation decisions.

Agent Authorization

An authenticated agent is executed in its language environment through a language module such as a Perl interpreter. Whenever the agent requests system resources, the language module must hand-off the authorization decision to a module designed for that purpose. Figure 4.3 illustrates the process of authorizing an agent's request to access system resources. The procedure consists of the following steps:

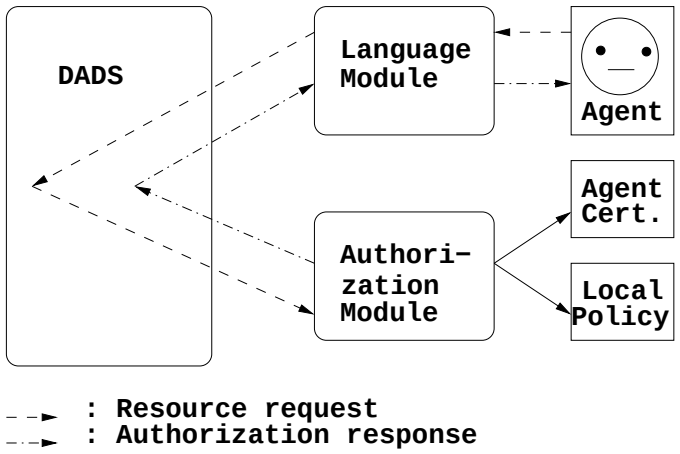

Figure 4.3: Agent Authorization

1. The language module transmits the agent's request to the DADS. Based on the agent's credentials obtained in the authentication phase, the DADS invokes the appropriate authorization module, and forwards the agent's resource request to it.

2. Based on the agent's identity and the restrictions inside its PC, the authorization module returns an OK / NOT OK response to the DADS, which is then transmitted to the language module.

3. Finally, the language module allows the agent's request (if the response was OK) or blocks it (if the response was NOT OK).

The agent authorization module may employ the Generic Authorization and Access control API (GAA-API) [29] to facilitate authorization decisions on resource access for agents. The GAA-API supports a variety of security mechanisms and authorization models. A typical GAA application creates a control structure that stores callback routines for policy evaluation, and a security context that contains the current user's credentials. Policy evaluator callbacks may be installed for any type of resource and resource access, and are invoked by the GAA-API when the user attempts to access the specified resource. Thus, the GAA-API provides a framework for mechanism-independent authorization decisions on resource access. A referential implementation of the GAA is available from [14].

An authorization module for agents may install a callback to evaluate a resource requested by an agent against the set of restrictions specified in its PC. The restrictions in the agent's PC may be extracted by the authorization module using functions

defined in the GSS-API extension. These restrictions would then be passed to the policy evaluator callback, along with the agent's request, to determine if the agent may be allowed access to the requested resource.

The authorization procedure requires modification of the agent's language environment to make it "agent-enabled", but at the same time simplifies matters by placing the burden of authorization decisions on the authorization module. Thus, the language environment may provide wrappers for system calls, which would simply invoke the authorization module, and based on its response, either allow or deny the agent access to the requested resource.

Delegation

Before an agent clones itself, it must create a new credential for the clone. The PC creation process for agents involves a signature on delegation information by the host that the agent executes on. Hence, before actually creating a clone, an agent must interact with a delegation module designed for the purpose of signing the agent's delegation information using the host's credentials. This implies that the delegation module must have the authority to sign information on behalf of the host. Figure 4.4 illustrates the process of delegation of authority from an agent to its clone:

1. The agent computes a hash over the new agent that it wishes to create, and

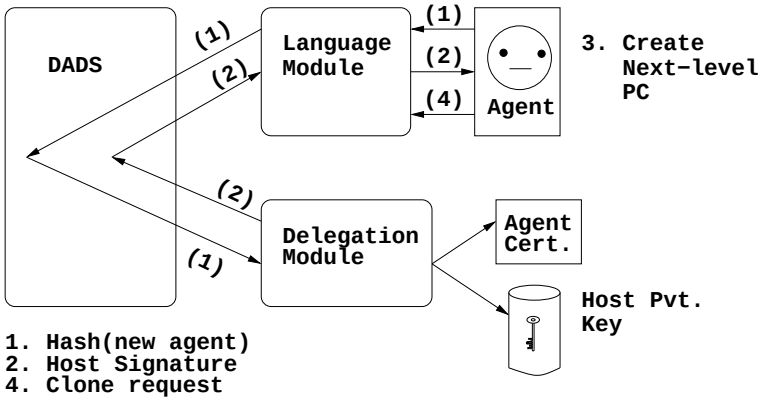

Figure 4.4: Delegation

requests the host's signature on the same. The request is sent to the language module and routed to the delegation module through the DADS.

2. The delegation module verifies the agent's request based on its credentials obtained during the authentication phase. If the request is valid, the module returns a signature, using the host's private key, on the agent hash, host identity, and the current time. Otherwise, it returns a NOT OK message.

3. The delegation module's response is transmitted to the executing agent. If the delegation module returns the host signature, the agent may optionally verify the signature and create a new PC for the clone. Finally, it creates a new agent by sending a "clone" request to its language module.

The delegation module may employ functions defined in the extensions to the GSS-API to generate a delegated credential for the mobile agent.

The modular design of the DADS makes it easy to add functionality to it such as PC-based agent security. Since modules do not form a part of the DADS core, they may be developed independently of the DADS, as long as their interface with the DADS is well defined. Thus, the mechanisms for agent authentication, authorization, and delegation, when implemented as DADS modules, customize it to support PC-based security for agent applications.

CHAPTER 5

CONCLUSION

This thesis has proposed a security model for mobile agent applications that is based on the Grid Security Infrastructure and X.509 Proxy Certificates . A Proxy Certificate issued to an agent by its originator serves as the agent's credential for authentication to remote hosts. The binding between an agent and its credential is maintained through a cryptographic hash of the agent code and static data stored inside its certificate. Further, restrictions encoded inside the certificate serve as a basis for making authorization decisions on resource access for agents. This allows the agent's originator a flexible selection of privileges for an agent, suited to the nature of the agent's task. The language used for encoding restrictions must be uniformly interpreted by both the issuer of the agent's certificate and the host that enforces authorization decisions based on certificate restrictions. Finally, credential delegation, when performed on trusted hosts, allows an agent to clone itself by creating a new Proxy Certificate for the clone and associating with it a subset of the parent agent's rights. Since a Proxy Certificate is nothing but a Public Key Infrastructure (PKI) credential, the proposed

mechanisms offer a scalable solution to achieving security in the mobile agent environment. A proof-of-concept implementation of these mechanisms is proposed in the context of a flexible and scalable agent infrastructure, the Distributed Agent Delivery System (DADS), which is capable of supporting multiple agent languages and security mechanisms though its modular interface. The security mechanisms are implemented as modules within DADS, thus making them flexible and extensible. Thus, future extensions to the agent security model could be easily incorporated within the DADS framework.

The proposed authorization mechanism relies upon policy restrictions encoded inside a Proxy Certificate to make authorization decisions. Although the grammar and semantics for the policy language have not been discussed, it is obvious that any such language must be rich enough to describe all resource types that might be required for a computation and the corresponding operations on the resources. Another desirable feature in the mobile agent security model is motivated by the heterogeneous nature of resources accessed by agents and their differing security requirements. Similar to the Grid, different resources may require different kinds of authentication. The security model presented here could be made interoperable with participating sites' local security policies and mechanisms through the co-operation of the DADS. In order to ensure uniform authentication across resources, the DADS

at each participating site could provide functionality for mapping an agent's proxy credential to a local credential, similar to the gatekeeper in Globus. This places the burden of managing multiple agent credentials on the DADS on each participating host, thus making the agent more light-weight than if it had to maintain multiple credentials for authenticating to multiple sites.

BIBLIOGRAPHY

[1] J. Algesheimer, C. Cachin, J. Camenisch, and G. Karjoth. Cryptographic security for mobile code. In *Proc. IEEE Symposium on Security and Privacy*, pages 2–11, May 2001.

[2] Joachim Baumann, Fritz Hohl, and Kurt Rothermel. Mole - Concepts of a Mobile Agent System. Technical Report TR-1997-15, 1997.

[3] R. Butler, D. Engert, I. Foster, C. Kesselman, S. Tuecke, J. Volmer, and V. Welch. A National-Scale Authentication Infrastructure. *IEEE Computer*, 33(12):60–66, 2000.

[4] C. Cachin, J. Camenisch, J. Kilian, and J. Müller. One-round secure computation and secure autonomous mobile agents. In *Proc. 27th International Colloquium on Automata, Languages and Programming (ICALP)*, volume 1853 of *Lecture Notes in Computer Science*, pages 512–523, 2000.

[5] Cameron Ross Dunne. Using Mobile Agents for Network Resource Discovery in Peer-to-Peer Networks. *Newsletter of the ACM Special Interest Group on E-commerce*, 2(3):1–9, 2001.

[6] W. M. Farmer, J. D. Guttman, and V. Swarup. Security for mobile agents: Issues and requirements. In *Proceedings of the 19th National Information Systems Security Conference*, pages 591–597, Baltimore, MD, 1996.

[7] William M. Farmer, Joshua D. Guttman, and Vipin Swarup. Security for Mobile Agents: Authentication and State Appraisal. In *Proceedings of the Fourth European Symposium on Research in Computer Security*, pages 118–130, Rome, Italy, 1996.

[8] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[9] Ian Foster. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *Lecture Notes in Computer Science*, 2150, 2001.

[10] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a new Computing Infrastructure*. Morgan Kaufman, San Francisco, California, 1999.

[11] Ian T. Foster, Carl Kesselman, Gene Tsudik, and Steven Tuecke. A Security Architecture for Computational Grids. In *Fifth ACM Conference on Computer and Communications Security*, pages 83–92, 1998.

[12] Alan O. Freier, Philip Karlton, and Philip Karlton. The SSL Protocol Version 3.0, March 1996. Internet draft.

[13] A. Fugetta, G. P. Picco, and G. Vigna. Understanding Code Mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361, May 1998.

[14] The GAA-API Homepage: http://www.isi.edu/gost/info/gaaapi.

[15] The Globus Project Website: http://www.globus.org.

[16] Robert S. Gray. Agent Tcl: A Flexible and Secure Mobile-Agent System. In *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL 96)*, Monterey, California, 1996.

[17] The GSI Website: http://www.globus.org/security.

[18] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile Agents: Are they a good idea? Technical report, T. J. Watson Research Center, Yorktown Heights, New York, 1995.

[19] Fritz Hohl. Time Limited Blackbox Security: Protecting Mobile Agents From Malicious Hosts. *Lecture Notes in Computer Science*, 1419:92–113, 1998.

[20] R. Housley, W. Ford, W. Polk, and D. Solo. RFC 2459: Internet X.509 Public Key Infrastructure Certificate and CRL Profile, January 1999. Status: PROPOSED STANDARD.

[21] W. Jansen and T. Karygiannis. NIST Special Publication 800-19 - Mobile Agent Security, 2000.

[22] G. Karjoth, N. Asokan, and C. Gulcu. Protecting the computation results of free-roaming agents. In *Proceedings of the Second International Workshop, MA'98*, pages 195–207, Stuttgart, Germany, 1998.

[23] Neeran M. Karnik and Anand R. Tripathi. Design Issues in Mobile-Agent Programming Systems. *IEEE Concurrency*, 6(3):52–61, July–September 1998.

[24] J. Kohl and C. Neuman. RFC 1510: The Kerberos Network Authentication Service (V5), September 1993. Status: PROPOSED STANDARD.

[25] Kre J. Lauvset, Dag Johansen, and Keith Marzullo. TOS: A Kernel of a Distributed Systems Management System, 2000.

[26] J. Linn. RFC 2743: Generic Security Service API Version 2, Update 1, January 2000.

[27] George C. Necula. Proof-Carrying Code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97)*, pages 106–119, Paris, January 1997.

[28] B. Clifford Neuman. Proxy-Based Authorization and Accounting for Distributed Systems. In *International Conference on Distributed Computing Systems*, pages 283–291, 1993.

[29] T. Ryutov, C.Neuman, and L.Pearlman. Generic Authorization and Access control Application Program Interface C-bindings, 2000. Internet Draft, CAT Working group draft-ietf-cat-gaa-cbind-05.txt: http://www.isi.edu/gost/info/gaaapi/doc/drafts/cbind_draft5.txt.

[30] T. Sander and C. Tschudin. Towards Mobile Cryptography. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 215–224, Oakland, CA, USA, 1998. IEEE Computer Society Press.

[31] S.Meder, V.Welch, S.Tuecke, and D.Engert. GSS-API Extensions, 2002. Global Grid Forum (GGF) proposed recommendation: http://www.gridforum.org/security/ggf5_2002-07/draft-ggf-gss-extensions-06.PDF.

[32] S.Tuecke, D. Engert, Ian Foster, M. Thompson, L.Pearlman, and C.Kesselman. Internet X.509 Public Key Infrastructure Proxy Certificate Profile, October 2002. Internet Draft: draft-ietf-pkix-proxy-03.

[33] S.Tuecke, D. Engert, I.Foster, M. Thompson, L.Pearlman, and C.Kesselman. Internet X.509 Public Key Infrastructure Proxy Certificate Profile, 2001. Internet Draft: draft-ietf-pkix-proxy-02.

[34] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A Survey of Active Network Research. *IEEE Communications Magazine*, 35(1):80–86, 1997.

[35] J. Wray. RFC 2744: Generic Security Service API Version 2: C-bindings, January 2000.

[36] Bennet S. Yee. A Sanctuary for Mobile Agents. In *Secure Internet Programming*, pages 261–273, 1999.