Engineering Multi-Agent Systems

Donovan Artz, Vincent A. Cicirello, William C. Regli, Moshe Kam
College of Engineering
Drexel University
3141 Chestnut Street
Philadelphia, PA 19104

Abstract

Security of an agent system is often limited, relying on basic cryptographic techniques without consideration of issues such as key maintenance, forming and communicating in secure groups, or interlayer security. From a security engineering perspective, multi-agent systems introduce new channels and possibly layers, resulting in additional security concerns. A comprehensive security engineering perspective – studying the information flow of the multi-layered system, identifying, analyzing and addressing multi-level security threats – is rarely taken.

This paper presents a security engineering process for multi-agent systems – motivating the need for comprehensive security engineering and showing how to proceed with the process within an agent system. One of the largest obstacles in security engineering is understanding how to decompose a system into the parts that require security. This paper provides a decomposition for agent systems that can be directly applied to the security engineering process. Examples are given that detail the application of the presented security engineering process to: 1) a FIPA-compliant agent system; and 2) peer-to-peer content lookup.

The most important contribution of this paper, is proposing a formal approach to addressing security within *an agent system, where there exist unique and application-specific threats that must be addressed.*

1. Introduction

Security of an agent system is often limited, relying on basic cryptographic techniques without consideration of issues such as key maintenance, forming and communicating in secure groups, or interlayer security (e.g., the network is assumed secure by means independent of the agent system). From a security engineering perspective, multi-agent systems introduce new channels and possibly layers, resulting in additional security concerns. A comprehensive security engineering perspective is rarely taken, studying the information flow of the multi-layered system, identifying, analyzing and addressing multi-level security threats.

The majority of work in multi-agent system security focuses on security mechanisms [17, 34, 27, 10, 31, 40, 39, 37]. The CoABS Grid, for example, provides authentication and encryption services [20]. RETSINA's security infrastructure [39] uses public key cryptography and a certificate authority for agent authentication and SSL for securing communications channels. Other work in multi-agent security begins to identify actors, assets and threats to agent systems [6, 18, 21]. For example, the developers of Aglets consider security issues through more of a security engineering approach, identifying some security threats for multi-agent systems and initiating work on a security architecture for Aglets [21]. Currently, however, the Aglets security architecture has only implemented authorization and resource access privileges [21].

This paper presents a security engineering process for multi-agent systems. It discusses how a multi-agent system should be analyzed and designed in the context of security engineering, motivating the need for comprehensive security engineering and showing how to proceed with the developed process within an agent system. One of the largest obstacles in security engineering is understanding how to decompose a system into the parts that require security. This paper provides a decomposition for agent systems that can be directly applied to the security engineering process.

Efforts in security engineering can be flawed for some or all of the following reasons:

1. The security engineer does not sufficiently understand the system or its parts. For example, an agent system might consider attacks against the system initiated by a "host". But, a host is a higher level concept that is comprised of multiple principals and layers from which threats can come.

2. The security engineer prematurely focuses on securing one part of the system. For example, a high level of en-

0-7803-8799-6/04/$20.00 ©2004 IEEE.

cryption is not necessary for an agent information service that provides freely available information.

3. The security engineer employs a mechanism with no or minimal consideration of the system. For example, one should consider the effects on system performance when employing a mechanism. If a computationally expensive mechanism is employed to counter a low-risk threat to a reasonably non-valuable asset, then the system can spend too much time securing itself and not enough time fulfilling its purpose.

If a security engineer does not completely understand a system, he or she cannot secure it. One goal of this paper is to enable agent system researchers to understand how to effectively secure an agent system. In particular, this paper prepares the developers of an agent system to correctly address the fundamental question in security:

How can the flow of information be controlled at or between any two principals?

A general understanding of a part and its purpose in a system may reveal some threats, but significant threats also exist in the implementation of that part. It is easy to inadvertently take much of the functionality in a computing system for granted, focusing only on the parts of a system that are understood. By analogy, increasing the strength of your front door may not help if you leave a ground level window open. One must not neglect any single part of a system when addressing security. This necessitates a deep understanding of each part, and its implementation in a system.

The most important contribution of this paper, is proposing a formal approach to addressing security *within* an agent system. The security of the "host" is a separate problem, including aspects of network and operating security. Within an agent system, there are unique and application-specific threats that must be addressed.

Section 2 of this paper gives definitions and provides background in two parts: multi-agent systems (Section 2.1) and security engineering (Section 2.2). The technical approach follows in Section 3 with an in depth description of our security engineering process for multi-agent systems. Section 4 provides examples, demonstrating the technical approach and evaluation, including an in depth explanation of the application of security engineering to a FIPA-compliant agent system (Section 4.1) and the application of security engineering to peer-to-peer content lookup (Section 4.2). Section 5 offers points of discussion. Finally, the paper concludes in Section 6.

2. Background

2.1. Multi-Agent Systems

There are currently many different approaches to and implementations of multi-agent systems (e.g., [7, 14, 16, 19, 24, 30, 33, 35, 36]). A common platform for an agent system is a group of networked computers. An agent system that allows agents to move between hosts is a mobile agent system, thus freeing units of software from limiting their runtime existence to only one host or set of resources (e.g., [5, 22, 29, 38]).

In this paper, the FIPA abstract architecture [11] is used to serve as a common language for agent terminology. A FIPA-compliant system has several mandatory elements; this papers addresses the following:

- An **Agent**, which may or may not be mobile, is an autonomous unit of software within an agent system.
- A **Service** is a service provided for agents.
- An **Agent Communication Language** that is used by agents to communicate.
- An **Agent Directory Service** that tracks and responds to queries regarding the location of all agents within an agent system.
- A **Message Transport Service** that facilitates communication between agents.
- A **Service Directory Service** which tracks and responds to queries regarding the location of all services within an agent system.

The FIPA abstract architecture is sufficiently general as to be applicable to any agent system, including mobile agent systems. However, due to many possible implementation-specific approaches, agent mobility is not specified in the FIPA abstract architecture. In this paper, we assume that the Message Transport Service is able to transport an agent and its state. In general, agent mobility may be represented as an additional service in the agent system.

2.2. Security Engineering Review

The security engineering process used in this paper is informally described in [1]. More information on threat determination can be found in [2, 9, 23]. Classic security policies are described in [8, 3, 13, 25, 4].

2.2.1. General Principles. Comprehensive security engineering requires that all principals and channels are considered at all layers of a given system. If any principal or channel is ignored, then all of the threats to information requiring protection have not been considered. It is essential that security engineers understand their system in depth so that every principal, channel, and layer can be evaluated, allowing the security engineers to focus on the parts of their system at most risk.

In any system, information can be thought of as being either at rest on in motion. Data at rest is simply data in storage. The registers of a CPU, the cache on a hard drive, and the human brain are all examples of places where data can be stored. Data in motion is data moving between two units of storage. Wireless network traffic, a bus connecting a CPU and memory, and a human typing on a keyboard are all examples of data in motion. We use the term *principal* to mean a place where data can be at rest and the term *channel* to refer to a place where data can be in motion. A *path* is the sequence of principals and channels through which data flows.

Principals and channels are abstract units that describe a system independent of implementation details. However, in security engineering, the implementation details are critical in determining how principals and channels may be attacked. A channel may exist across many layers of implementation, thus one must consider all layers of hardware and software implementation in which these principals and channels exist. A *layer* is a unit of hardware or abstraction of hardware or software where one can identify principals and channels with the implementation of a system. The OSI Reference Model (OSI-RM) [41] for computer networks is an example of a set of layers. Each layer in the OSI-RM has a unique set of principals and channels. Even though each layer may be referring to the same perceived principals and channels, they are working with some unique data. All places where data may exist can be described in terms of principals, channels, and layers.

The ordering of data sent between two principals is determined by a *protocol*, consisting of rules that determine how, when, and what data is sent over a channel. It is not uncommon for principals to have a restricted type of information that may be transfered between them (e.g., a bank only provides a customer with information about that customer's accounts). A protocol facilitates restricted flow of information, and thus an attack on the protocol may be a threat to the information at the principals or in the channel between them.

2.2.2. Definitions. In any given computing system, there are *assets*, *actors*, *threats*, and *outcomes*. The security needs of a system are identified in terms of these features.

An asset is an entity that is or will be worth protecting. A security engineer considers not just the location of an asset (a principal), but also the means and medium by which an asset is moved (a channel). A security mechanism may protect an asset, but it can fail to protect information about that asset. This oversight can be caused by not having a full understanding of the asset in question. It is critical to the security engineering process to harness detailed knowledge and experience about the assets of the system in question.

An actor is an entity that can act within one or more layers of a system. More specifically, an actor is an entity with the potential to enact a threat against an asset. While an actor may be traced back to a human, the actors within a given layer may not be human. Through the determination of the actors, it becomes known who or what must be restricted in order to minimize the risk of a related threat.

A threat represents how an undesirable event could occur. Once the assets and actors in a given layer are understood, it is possible to determine what threats the actors pose to the assets. In most cases, it is impossible and impractical to eliminate all possible threats. Instead, it is important to carefully consider the set of all possible threats, and to address only the most significant ones.

An outcome is an undesirable result caused by an actor in a system. There are many ideas on categories of outcomes in the security literature. One general set of outcomes is described in the OCTAVE threat profiling system [26]. These categories generally describe the types of outcomes that apply to all assets in a system:

1. Disclosure: information in the asset is released to an unintended recipient.
2. Loss: information in the asset is irretrievably lost.
3. Modification: information within the asset is modified.
4. Interruption: channels to and from the asset are made unusable.

Security is the state of a system where the risk to all assets has been minimized to an acceptable level. Security engineering is the practice of addressing the protection needs of all assets in a system. In security engineering, one can only minimize risk, not eliminate it. Furthermore, one cannot reasonably provide a countermeasure to every threat in a system, but rather only counter the threats that are determined to be most serious. One must recognize that security is overhead, and a balance must be struck between security and the functional requirements of a system. The security engineering process presented in the next section depends on these guidelines to be effective in application.

3. Security Engineering of Multi-Agent Systems

There are several existing processes for security engineering described in the literature. This paper adapts a general process as informally described by Anderson [1]: 1) Determine Assets; 2) Profile Threats; 3) Create Policy; 4) Develop Mechanisms; and 5) Evaluate Security.

3.1. Determine Assets

Enumerating assets determines what needs to be protected. One of the most common mistakes in security engineering is to ignore an asset or threat. This happens when an engineer is not aware that the asset or threat exists, or immediately dismisses an asset or threat as being irrelevant to security.

In agent systems, the majority of work focuses on threats against agents realized by a "host." The difficulty of the *hostile host* problem is evidence that some do not always go into sufficient depth to address agent security. What is a "host" in an agent system? There are many implementation dependent answers, but in all cases, a host is composed of more than one principal and more than one layer.

Formulation. An agent principal is denoted by A, and a service principal is denoted by S. The set of all agents is $\mathcal{A}$, and the set of all services is $\mathcal{S}$. The uni-directional channel from one principal, A, to another principal, S, is denoted by the ordered principals: $\overline{AS}$. A bi-directional channel between principals A and S is denoted by the unordered principals: $\overline{\overline{AS}}$. If the bidirectional channel $\overline{\overline{AS}}$ exists, so do the unidirectional channels $\overline{AS}$ and $\overline{SA}$. The set $\mathcal{P} = A \cup S$ denotes all principals. in a system, and the set $\mathcal{C} = \mathcal{P} \times \mathcal{P}$ denotes all potential channels in a system. The set $\mathit{Assets} = \mathcal{P} \cup \mathcal{C}$ of all principals and channels form the assets of the system.

3.2. Profile Threats

Threats are realized by actors in a system. Each principal $P \in \mathcal{P}$ is an actor. An actor realizes a threat against another principal, channel, or protocol through an instance of a channel to another principal. Each realized threat results in an outcome involving the assets of the targeted principal, channel, or protocol. Through the enumeration of all uni-directional channels within a layer, it is possible to create a complete characterization of the possible threats against assets within that layer.

For each uni-directional channel, an actor can realize a threat against: the channel, the targeted principal, or the protocol corresponding to that channel.

Given a complete picture of the assets and threats, it is possible to determine which threats pose the most risk to the assets. It is seldom the case that all threats in a system can be addressed, and it is never the case that all threats can be eliminated. Given a set of significant threats to address, a policy is needed to determine how to counter them.

Formulation. An outcome is denoted by O, and in general, there are four possible outcomes: O_D (disclosure), O_I (interruption), O_M (modification), and O_L (loss). The tuple $\langle \alpha, \beta, \omega \rangle$ defines the threat against asset β by actor α resulting in outcome ω, such that $\alpha \in \mathcal{P}$, $\beta \in \mathcal{P} \cup \mathcal{C}$, and $\omega \in \{O_D, O_I, O_M, O_L\}$.

Example. The tuple $\langle A_i, S_j, O_I \rangle$, refers to threat of agent A_i causing interruption of service S_j.

3.3. Create Policy

A policy describes how to address the selected threats in a system. This step is frequently avoided, or construed as unnecessary. The purpose of a policy is to enable one to choose mechanisms that directly address the threats. Without a policy, one cannot evaluate whether the mechanisms have been successful at minimizing the risk to the assets as expected.

Example policies. A good policy is highly dependent on the implementation of an agent system and its application. However, there are several informally described policies that re-occur in existing agent and security research:

- *all principals trusted*: all hosts, agents, and services are trusted.
- *trusted hosts, untrusted agents*: all hosts are trusted by all agents, but neither hosts nor agents trust other agents.
- *trusted agents, untrusted hosts*: all agents are trusted by all agents, but some or all hosts cannot be trusted, nor do they trust the agents in return.
- *Bell-Lapadula*: an agent or host can only access other hosts or agents at or below its own level of access.
- *Lattice*: an agent or host can only access other hosts or agents at or below its own level of access in more than one access category (e.g. level of access can be "top-secret" for the Message Transport Service, but only "restricted" for the Service Directory Service.)
- *RBAC*: an agent or host can access other agents or hosts based on its current role within the agent system.

3.4. Develop Mechanisms

Given a security policy, mechanisms must be selected or developed to enforce it. The mechanisms are selected only after it is known exactly which assets, which threats, and what policy are being employed within a layer. There is a plethora of mechanisms available, especially for using cryptography and key management in a specific layer of a system.

The inclusion of a mechanism may introduce a new principal, and subsequently new channels and protocols, into the system. In such cases, it is necessary to repeat the process, beginning with the determination of new assets.

Example types of mechanisms. Authentication verifies a principal's identity (e.g., PKI). Integrity ensures that data content remains unmodified and in its original form (e.g., checksums). Confidentiality precludes disclosure of information to unintended recipients (e.g., cryptography). Non-repudiation precludes principals from rescinding previous data. Revocation enables a principal to rescind privileges given to other principals (e.g., using a Security Mediator).

3.5. Evaluate Security

The final step is to evaluate how well the selected mechanisms enforce the security policy. The ultimate consideration is whether or not the risk to the system's assets has been minimized as expected. There are several formal methods for evaluating the security of any general computing system. The Common Criteria [1] is the current U.S. standard for the evaluation of security in information technology. Formal methods of evaluation are useful for obtaining certifications and for comparing security between systems.

Active probing and "controlled attacks" against a system are a good practice in addition to formal analysis, as the implementation of mechanisms, or other implementation details (e.g., bugs) may yield security holes that are not addressed as intended. For example, Fischmeister et al benchmark the security of a set of agent frameworks by performing active probing [12]. Additionally, existing security tools (e.g., SATAN, COPS, Internet Scanner) can be used to actively evaluate security in commonly studied layers, such as the networking and applications in UNIX-based systems. Due in part to the implementation-dependent details of agent systems, there is currently no known tool for probing agent system security.

4. Applying the Method

4.1. FIPA-Compliant Agent System

The security of a multi-agent system, while dependent on the services of operating systems and networking, is separate from the security of other layers. Focusing only on the principals and channels in an agent system, there are some parts of the security engineering process which all agent systems have in common.

In a FIPA-compliant agent system, S_M represents an instance of the Message Transport Service, S_{SD} represents an instance of the Service Directory Service, and S_{AD} represents an instance of the Agent Directory Service. Also note that $S_M, S_{SD}, S_{AD} \in \mathcal{S}$.

A host comprises several mandatory services: S_M, S_{SD}, and S_{AD}. It is also mandatory that at least one instance of an agent, A, exists. These entities are the principals, and the cross product of these represents the possible bi-directional channels. Together, the principals and channels form the minimum set of assets for a FIPA-compliant agent system. This set grows as an agent (a principal) and its means of communications (channels) are added to the system.

The first step of the security engineering process is to determine the assets. The number of principals in a FIPA-compliant system depends directly on the number of agents and services. In this example, a system has n agents, $A_1, ..., A_n$, and m services, $S_1, ..., S_m$. The set of principals include:

$$\mathcal{P} = \{A_1, ..., A_n, S_1, ..., S_m, S_M, S_{SD}, S_{AD}\}$$

1 See http://csrc.nist.gov/cc/

The principals and their cross-product (the channels) form the assets.

A core security concern in FIPA-compliant agent systems is the availability of S_{SD}. Each agent may need to complete a set of tasks, where each task requires one of the m services. In order to locate a service, an agent must use the Service Directory Service S_{SD}. The agents are competitive and not trusted. All agents will fail to complete their tasks if S_{SD} becomes unavailable. The agent systems described in [15] serve to illustrate that agents are dependent on the ability to obtain needed services in order to complete their tasks. The principal S_{SD} provides this fundamental capability to agents, and is thus the most valuable asset of a FIPA-compliant system. A formal security engineering analysis of a FIPA-compliant system reveals the threats against this service and how to address them.

The next step is to profile threats. In profiling possible threats, those with the most valuable assets and most untrusted actors should be addressed. In this agent system, we know that S_{SD} is the most valuable asset, and that the agents are the most "untrusted actors". Thus, the most significant threats in this system include:

$$\langle A_i, S_{SD}, \omega \rangle, 1 \leq i \leq n, \omega \in \{O_I, O_D, O_L, O_M\}$$
$$\langle A_i, \overline{\overline{A_i S_{SD}}}, \omega \rangle, 1 \leq i \leq n, \omega \in \{O_I, O_D, O_L, O_M\}$$

The next step is to form a policy that addresses these threats. The first part of the policy should state that all threats beyond those listed above are insignificant or irrelevant. The listed threats resulting in disclosure are irrelevant, as all information (including services requested by other agents) in S_{SD} should be freely available to any agent. Modification and loss of information in S_{SD} can be prevented by using a protocol over $\overline{\overline{A_i S_{SD}}}$ that limits an agent's ability to only request and receive information from S_{SD}. This limitation on the channel protects both the channel and the asset S_{SD} from Modification and Loss.

The outcome of interruption may be addressed in two possible ways: 1) the S_{SD} should be highly available, and able to respond to any demand placed on it by the agent system; or 2) the S_{SD} should limit the amount of service it provides to any single entity for some period of time. One or both of these policies may prevent the outcome of interruption of S_{SD}. In order to prevent interruption on the channel $\overline{\overline{A_i S_{SD}}}$, the channel must be implemented such that it is able to accommodate any possible demand that may be placed upon it by the agents and S_{SD}.

Given a policy, mechanisms must be selected or developed to enforce it. One possibility for ensuring that S_{SD} is highly available is a decentralized S_{SD} distributed over multiple hosts. Such a mechanism would require synchronization of data between hosts as needed, occurring between multiple instances of S_{SD}, thus introducing new principals and channels into the system. A possible mechanism for limiting access to S_{SD} by actor would be to verify the identity of each agent as it requests information from S_{SD}. If verification is successful, and the requesting agent has not exceeded its limit, the request for information is forwarded to S_{SD}. This authentication mechanism would introduce an entirely new principal into the agent system and create additional assets. Other mechanisms may also enforce the policy; the cost of the mechanisms should be considered in addition to their ability to address threats.

The final step is to evaluate the effectiveness of the selected mechanism(s) in enforcing the policy. The best evaluation may be through empirical methods using experiments to realize threats against the system before it goes into production. A possible empirical experiment is to create the maximum possible number of agents, having each agent query S_{SD} as quickly as possible. If S_{SD} and $\overline{\overline{A_i S_{SD}}}$ are not interrupted, the mechanisms are successful.

4.2. Peer-to-peer Content Lookup

A system may employ peer-to-peer content lookup in order to distribute data and make it available to hosts in a decentralized manner. Examples of this peer-to-peer technology are described in [28, 32]. An agent system is an ideal framework for implementing peer-to-peer content lookup, as it is designed to accommodate decentralized algorithms. Agents can both distribute data across hosts, and respond to queries for data in the system.

In this example, there exists a peer-to-peer data lookup service, S_P, which is used by agents to determine on which host a particular unit of data is located and to store data distributed across the system's hosts.

In real-world peer-to-peer systems, the network topology may not be constant. This is especially true if the agent system is implemented on mobile computation devices connected via a mobile ad hoc network. A mobile ad hoc network eliminates the need for all hosts to have a direct connection to each other. However, in order for information to travel between two hosts without a direct connection, that information must travel through at least one intermediate host. If an intermediate host is not an intended recipient of the information it passes along, then disclosure may occur. Figure 1 illustrates an ad hoc network where not all hosts can communicate directly with each other..

Another feature found in real-word systems is the capability of hosts to form arbitrary groups in which all group members should be able to communicate without disclosing information to non-members. When two hosts in the same group must use a non-member intermediate host to communicate, the threat of disclosure exists. This example demonstrates how the security engineering process formally addresses security for groups using an ad hoc wireless network to facilitate communication.

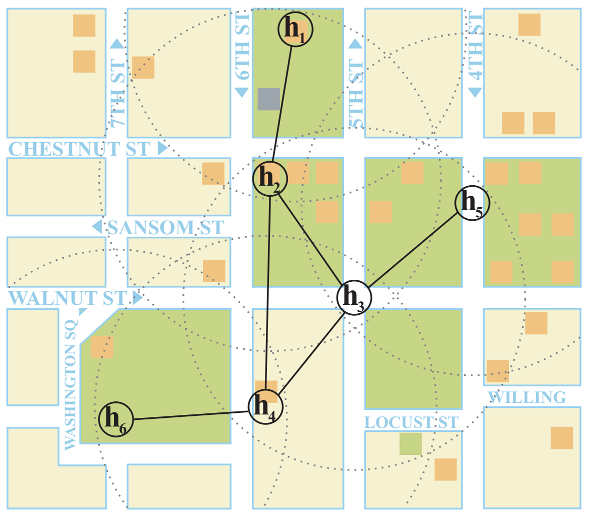

Figure 1. An ad hoc network, using intermediate hosts to enable communication between hosts without a direct connection, may employ a peer-to-peer lookup service to keep data available without a single point of failure.

A typical agent based peer-to-peer system has many significant assets, but for simplicity, this example focuses on S_P. Agents, $A_1, ..., A_n$ are used to distribute information between a group of hosts, and to answer queries for data in that group. In this example, the principals (which determine the assets) are:

$$\mathcal{P} = \{A_1, ..., A_n, S_P, S_M, S_{SD}, S_{AD}\}$$

The next step is to profile threats. Informally, it has been stated that information may be disclosed to non-members of a group under certain conditions. Specifically, the possibility exists that an agent may unintentionally disclose information to S_P on a non-member host as the information is in transit to its destination. It is assumed (for simplicity) that active attacks are not possible, therefore the possibility of loss, manipulation, or interruption are insignificant. The threats to be addressed are:

$$\langle A_i, S_P, O_D \rangle, 1 \leq i \leq n$$

Given these threats, a policy is needed to address them. One possible policy is to not communicate with a host unless all intermediate hosts are also members of the same group. This, of course, limits communications under certain conditions. Another possible policy is to render information readable only to group members. This allows information to be passed through non-member hosts while preventing disclosure of the content of the information[2]

Using this policy, mechanisms must be selected to enforce it. Cryptography is the most obvious choice for rendering information unreadable to unintended parties. However, successful cryptography requires several additional mechanisms in order to be effective. These additional mechanisms (for authentication and distribution of cryptographic keys) introduce several new principals into the system. Another possible mechanism is to split the information across two possible paths in the ad hoc network. This does not introduce new principals, but also is not a solution under all conditions.

The final step is evaluation. Given an implementation of this system, regular empirical and formal proofs can be made of the effectiveness of cryptography in enforcing the security policy. The simplest method is to observe the data passing through non-member intermediate hosts. If the applications on a host are consistently unable to read received information from another group, then the selected mechanism is proven to be effective at enforcing the policy.

5. Discussion

The security engineering of multi-agent systems offers several research challenges:

Tradeoff of Security and Cost of Security: While the security of every part of an agent system must be addressed, it is impractical to provide a countermeasure to every possible threat to every part of that system. Implementing, enforcing, and maintaining security in any domain incurs a cost. Likewise, not all threats to all parts of a system pose a significant risk. A general rule to apply is that the cost of security should not be greater than the cost of having the system compromised. However, the cost of security does not justify a security plan that has not at least evaluated all parts of a system. Further, how does one compare the cost of security and the cost of having the system compromised? Are the metrics of each the same? Can they be equated?

Determining an Asset's True Value: What is an asset to one person may be inconsequential to another. It is very important to the security engineering process that those who use an asset are given the opportunity to determine the true value (as much as this is possible) of an asset before spending time and money to secure it. Without accurate relative values of assets, it is possible to spend too much time securing one asset, and not enough time securing others.

Security of a Multi-Agent System's External Dependencies: Consideration must also be given to the principals and channels of a given layer that exist in other layers outside of the agent system. Security of a system requires that external dependencies of that system are also secure. For example, if we have an agent system that exists on several networked hosts, then the security of the agent system can be dependent on the security of the network. Given the high level nature of agent systems, there are several external dependencies that must be considered. Furthermore, these external dependencies are highly dependent on how the agent system has been implemented. An agent system implemented in native code will have different issues from one that is implemented for a virtual machine.

Evaluating Security: Due in part to the implementation-dependent details of agent systems, there is currently no known tool for probing agent system security. Can implementation-independent tools be developed to assist in the evaluation of an agent system's security? What general types of tools would be most useful to probing the agent system layer?

6. Conclusions

In general, agent systems often suffer from the lack of a comprehensive analysis of their security needs. The approach presented in this paper demonstrates a methodology towards engineering more secure multi-agent systems. As standards like FIPA continue to foster a common language for agent systems, it will be possible to refine the security engineering process and to better understand how agents and security interoperate.

References

[1] R. Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. John Wiley and Sons, Inc., 2001.

[2] T. Aslam, I. Krsul, and E. H. Spafford. Use of a taxonomy of security faults. In *Proc. 19th NIST-NCSC National Information Systems Security Conference*, pages 551–560, 1996.

[3] D. E. Bell and L. J. LaPadula. Secure computer systems: Mathematical foundations. Technical Report 2547, Mitre Corporation, March 1973.

[4] K. Biba. Integrity considerations for secure computer systems. Technical Report 76-372, Mitre Corporation, 1977.

[5] D. Chess, B. Grosof, C. Harrison, D. Levine, C. Parris, and G. Tsudik. Itinerant Agents for Mobile Computing. *IEEE Personal Communications*, 2(5):34–49, 1995.

[6] D. Chess, C. Harrison, and A. Kershenbaum. Mobile Agents: Are They a Good Idea? Technical Report RC 19887 (December 21, 1994 - Declassified March 16, 1995), IBM T.J. Watson Research Center, Yorktown Heights, New York, 1994.

[7] P. R. Cohen, A. Cheyer, M. Wang, and S. C. Baeg. An open agent architecture. In M. N. Huhns and M. P. Singh, editors, *Readings in Agents*, pages 197–204. Morgan Kaufmann, San Francisco, CA, USA, 1997.

2 However, it still may be disclosed to any intermediate host that some information is being transfered.

[8] D. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[9] D. L. Drake and K. L. Morse. The security-specific eight stage risk assessment methodology. In *17th NIST-NCSC National Computer Security Conference*, pages 441–450, 1994.

[10] W. M. Farmer, J. D. Guttman, and V. 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.

[11] Fipa abstract architecture specification, 2002.

[12] S. Fischmeister, G. Vigna, and R. A. Kemmerer. Evaluating the security of three java-based mobile agent systems. In *Proceedings of the International Conference on Mobile Agents*, December 2001.

[13] J. Goguen and J. Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, April 1982.

[14] J. R. Graham, K. S. Decker, and M. Mersic. DECAF – a flexible multi agent system architecture. *Autonomous Agents and Multi-Agent Systems*, 7:7–27, 2003.

[15] J. Hendler. Agents and the semantic web. *IEEE Intelligent Systems*, 16(2):30–37, 2001.

[16] C. Hoile, F. Wang, E. Bonsma, and P. Marrow. Core specifications and experiments in DIET: A decentralised ecosystem-inspired mobile agent system. In *Proceedings of the 1st International Conference on Autonomous Agents and Multiagent Systems (AAMAS2002)*, pages 623–630, 2002.

[17] W. Jansen. Countermeasures for mobile agent security. In *Computer Communications, Special Issue on Advances in Research and Application of Network Security*, November 2000.

[18] W. Jansen and T. Karygiannis. Mobile agent security. Technical report, NIST, September 1999. Publication 80019.

[19] N. R. Jennings, K. Sycara, and M. Wooldridge. A roadmap of agent research and development. *Journal of Autonomous Agents and Multi-Agent Systems*, 1(1):7–38, 1998.

[20] M. L. Kahn and C. D. T. Cicalese. The CoABS grid. In W. Truszkowski, C. Rouff, and M. Hinchey, editors, *Innovative Concepts for Agent-Based Systems: First International Workshop on Radical Agent Concepts, WRAC-2002*, volume LNCS 2564 of *Lecture Notes in Computer Science*. Springer-Verlag, 16-18 January 2002.

[21] G. Karjoth, D. B. Lange, and M. Oshima. A security model for aglets. *IEEE Internet Computing*, pages 68–77, July-August 1997.

[22] E. A. Kendall, P. V. M. Krishna, C. V. Pathak, and C. B. Suresh. Patterns of intelligent and mobile agents. In *International Conference on Autonomous Agents*, pages 92–99, May 1998.

[23] I. Krsul, E. Spafford, and M. Tripunitara. Computer vulnerability analysis. Technical report, COAST Laboratory, Purdue University, May 1998.

[24] R. Lentini, G. P. Rao, J. N. Thies, and J. Kay. Emaa: An extendable mobile agent architecture. In *AAAI Workshop on Software Tools for Developing Agents*, July 1998.

[25] J. McLean. Security models. In J. Marciniak, editor, *Encyclopedia of Software Engineering*. John Wiley & Sons, 1994.

[26] Octave: Information security risk evaluation, 2002.

[27] T. Qian and R. Campbell. Dynamic agent-based security architecture for mobile computers. In *Second International Conference on Parallel and Distributed Computing and Networks*, pages 291–299, December 1998.

[28] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proceedings of ACM SIGCOMM*, pages 161–172, August 2001.

[29] D. Rus, R. Gray, and D. D. Kotz. Transportable Information Agents. In W. L. Johnson and B. Hayes-Roth, editors, *Proceedings of the 1st International Conference on Autonomous Agents*, pages 228–236, New York, 1997. ACM Press.

[30] N. Sadeh, D. Hildum, D. Kjenstad, and A. Tseng. MASCOT: an agent-based architecture for coordinated mixed-initiative supply chain planning and scheduling. In *Agents 1999 Workshop on Agent-Based Decision Support in Managing the Internet-Enabled Supply-Chain*, May 1999.

[31] T. Sander and C. F. Tschudin. Protecting mobile agents against malicious hosts. *Lecture Notes in Computer Science*, 1419:44–61, 1997.

[32] I. Stoica, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, pages 149–160, August 2001.

[33] V. S. Subrahmanian, P. Bonatti, J. Dix, T. Eiter, S. Kraus, F. Ozcan, and R. Ross. *Heterogeneous Agent Systems*. MIT Press/AAAI Press, Cambridge, MA, USA, 2000.

[34] V. Swarup and J. Fabrega. Trust: Benets, models and mechanisms. In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*. Springer-Verlag, 1999. 1603.

[35] K. Sycara, K. Decker, A. Pannu, M. Williamson, and D. Zeng. Distributed intelligent agents. *IEEE Expert*, December 1996.

[36] K. Sycara, M. Paolucci, M. van Velsen, and J. Giampapa. The RETSINA MAS infrastructure. *Journal of Autonomous Agents and Multiagent Systems (JAAMAS)*, 7:29–48, 2003.

[37] G. Vigna, B. Cassell, and D. Fayram. An intrusion detection system for aglets. In *Proceedings of the International Conference on Mobile Agents*, October 2002.

[38] J. White. Mobile agents. In J. Bradshaw, editor, *In Software Agents*. MIT Press, 1997.

[39] H. C. Wong and K. Sycara. Adding security and trust to multi-agent systems. In *Proceedings of the Autonomous Agents '99 Workshop on Deception, Fraud, and Trust in Agent Societies*, pages 149–161, May 1999.

[40] B. S. Yee. A sanctuary for mobile agents. In *Foundations for Secure Mobile Code Workshop*, pages 21–27, 1997.

[41] H. Zimmerman. OSI reference model - the ISO model of architecture for open systems intercomm. *IEEE Transactions on Communications*, April 1980.