METHODOLOGY OF RISK ASSESSMENT IN MOBILE AGENT SYSTEM DESIGN

By

Ingo McLean

A Thesis Submitted to the Graduate

Faculty of Rensselaer Polytechnic Institute

in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE

Approved:

Thesis Adviser

Rensselaer Polytechnic Institute
Troy, New York

April 2003
(For Graduation May 2003)

CONTENTS

LIST OF TABLES iv

LIST OF FIGURES v

ACKNOWLEDGMENT vi

ABSTRACT vii

1. Introduction 1
   - 1.1 Motivation 2
   - 1.2 Outline of this Thesis 2

2. Agent Makeup and Framework 4
   - 2.1 Introduction 4
   - 2.2 Agent Basics 4
   - 2.3 Agent Systems 5
   - 2.4 Conclusion 7

3. Security Scope 8
   - 3.1 Introduction 8
   - 3.2 Considerations for Security 8
   - 3.3 Levels of Security 9
   - 3.4 Costs of Security 12
     - 3.4.1 Cost of SSL Usage 13
     - 3.4.2 Cost of Agent Server/Host Authentication 13
     - 3.4.3 Cost of Authorization 14
     - 3.4.4 Cost of Message Digest 14
     - 3.4.5 Cost of Code Signing 15
     - 3.4.6 Cost of Auditing 15
   - 3.5 Conclusion 16

4. Abstract Mobile Agent Model 17
   - 4.1 Introduction 17
   - 4.2 Components 17
   - 4.3 Conclusion 20

5. Risk Management 21
5.1 Introduction 21
5.2 Basics 21
5.3 The Five Step Process 22
5.3.1 Step 1, "Identify Risks" 22
5.3.2 Step 2, "Assess Risks" 23
5.3.3 Step 3, "Develop Controls and Make Risk Decisions" 25
5.3.4 Step 4, "Implement Controls" 26
5.3.5 Step 5, "Supervise and Evaluate" 27
5.4 Application 27
5.4.1 Repository 27
5.4.2 Mobile Agents and the Polling Station 28
5.4.3 Other Components 29
5.4.4 Application of Risk Assessment to DOORS 29
5.5 Future Work 30
5.6 Conclusion 31
LITERATURE CITED 32
APPENDICES
A. Risk Assessment Worksheet 35
A.1 Instructions of Use 35

LIST OF TABLES

5.1 Step 2: Risk Probability 24

5.2 Step 2: Risk Severity 25

5.3 Step 2: Risk Matrix 26

LIST OF FIGURES

2.1 Basic Agent Architectures 4

2.2 Agent System in a Leader Environment 6

2.3 Agent System in a Secondary Leader Environment 6

3.1 Security Implementation Resource Matrix 12

4.1 Abstract Agent Model 18

5.1 Risk Management Process 23

5.2 DOORS Architecture 28

A.1 Risk Assessment Worksheet Instructions 35

A.2 Blank Risk Assessment Worksheet 36

A.3 Example Risk Assessment Worksheet 37

ACKNOWLEDGMENT

I want to thank my advisor Dr. Boleslaw K. Szymanski, for his guidance and mentorship. I would also like to thank my wife and children for putting up with the late nights of coding and continuous conversations that are frequently refferred to as "Speaking Geek".

Special thanks also go to David Kotfilla for the special assistance and use of the CISCO Academy Router Lab at RPI

Alan Bivens

Chris Rygaard

ABSTRACT

Current practices of software development use incomplete security models and result in implementations that leave security features either dangling in the midst of afterthoughts, or relegated to "future" upgrades or patches. Yet, security functions should be considered at the outset of any system design process. In Mobile Agent Systems, depending on their purpose, various approaches may be taken to provide security based on the vulnerability of the features. Starting early to plan security and its implementation in such systems is particularly important because by design they operate in open environments with little central control. With widespread information sharing between potential system breakers ranging from amateur hackers to professional crackers, system security features are tested every day. As evident from frequent break-ins, security is often proven either inefficient or non-existent. Clearly, a pragmatic methodology of determining the system security requirements is needed.

This Thesis provides new insights to agent security methodology, agent system abstraction and the evaluation to agent risk management. First, we catalog and classify numerous security techniques and practices for mobile agent systems. We also provide the definitions and describe implementations of security features relevant to mobile agent systems. Additionally, impact measurements relative to agent system design and performance vis-à-vis these security concepts are also presented. Secondly, a higher–level approach to viewing an agent system will be modeled. This Abstract Mobile Agent Model (AMAM) further defines the basic fundamental functions relevant to most agent systems. Thirdly, a detailed Risk Management Process defining and setting risk levels to the security concepts as applied to the AMAM will be demonstrated. In all we will discuss which security features are suitable for an Agent System and how they impact the performance of said systems.

CHAPTER 1
Introduction

Mobile Agents play a significant role in today's computing environment, from Brokering [4] and auctions [28] to network management [3] and military simulation [12, 5]. Demand for reliable, unaltered data and/or operations require that their implementations include assurance of some form of safety and integrity for the agent system. By agent system, we refer to all components of the system that are involved in or impacted by the agent activities. Typically, these components include the agent itself, agent host, agent home, communication links between either agents or hosts, and data transfers initiated by or directed to the agents. Each component of the agent system presents a security risk that requires a security solution. Addressing these risks requires proper planning throughout the systems life-cycle, yet most often, security holes, risk management or programming glitches become "top priority" once something goes wrong [13]. To start this process, we must first answer a few basic questions. At what point in time should security be involved? What options are available, and where can they be applied? How should the selected options be implemented? What impact will the security measures have on system performance and robustness? What forms of risk are we willing to accept on this type of application?

Security: This can be broadly defined as the protection of any or all components of an agent system. Security should be planned and integrated from the start of any programming project. This not only reduces the risk of a compromised system, but also lowers the complexity of rewriting code. Additionally, the system users have deeper trust in the system and use it more eagerly if they know their data remains intact. There are numerous mobile agent architectures [25] in existence that range from insecure to highly secure. However, a higher level of security usually means higher overhead and more cumbersome system access. Hence, the system designers need to take a close look at the goals and objectives of an agent system to determine which risks are acceptable and which need securing, since this

will determine the complexity of security code design. It follows, then, that the implementation of an unneeded security feature not only reduces program and programming efficiency, but also opens the door for more programming mistakes thus increasing the possibility of breaches in security. These decisions will determine how security will be implemented. We will discuss and categorize the main security features for agent systems in this paper.

1.1 Motivation

Each agent system sets unique goals for itself during design and development. The use of agents allows us to place more functionality into a modular, distributed framework where each "request" or "task" performed by an agent is perceived to be done autonomously. In order to evaluate what security measures need to be set, a methodological dissection of requirements, capabilities and functions must be conducted. Once done, a risk factor analysis based on those requirements will be developed which will allow us to focus on those security aspects that need to be handled. For instance, a routine agent used for fetching basic web-pages for a PDA will most likely sacrifice the overhead of an encrypted link for the sake of performance, while a bank transaction would pay that overhead to ensure security of data, possibly even adding encrypted transaction data and user authentication.

1.2 Outline of this Thesis

The paper will present information relative to agent systems in a sequential order. The next chapter covers agent basics. These basics present the mechanics and lay the groundwork for the remaining chapters. Once an understanding of agents, agent systems and the way they work is obtained, chapter 3 will stress the different reasons for the motivation and emplacement of security in an application along with the rationale for security considerations. Results from numerous minor tests evaluating each security implementation are also presented. Chapter 4, takes a look at the agent system and forms an Abstract Mobile Agent Model demonstrating the components of an agent system running from the most common to these that are somewhat esoteric.

Chapter 5 builds on the previous chapters by introducing the Risk Assessment process and exemplify the importance of security conscious decisions in mobile agent systems. The security methods discussed in the Risk Assessment will then be evaluated in relation to an agent system called DOORS (Distributed Online Object Repository System) [2]. A selection of features from DOORS will be applied during the Risk Management Process. Finally, a discussion of future work and possible additional considerations for this methodology will be given together with the conclusion.

CHAPTER 2
Agent Makeup and Framework

2.1 Introduction

This thesis concentrates its focus on a distributed agent system. For that reason focused discussion on mobile agents as opposed to stationary agents allow for a more diverse range in capabilities and configurations. For clarity, I will provide the basic framework of agent systems and their associated parts [17, 16, 14, 15]. This education provides a basis for claims laid out in further chapters.

2.2 Agent Basics

Let's begin with the agent. Agents are autonomous, mobile programs with variable levels of complexity [26, 27], depending on their role. Given these characteristics, agents perform various tasks on one to numerous systems. Most notabley, agents are programmed in universal, or machine-independent languages such as Tcl or Java [8], though forms of agent systems written in C and C++ exist in minor numbers yet provide less inter-platform robustness. For the remainder of this thesis, Java will be used as the primary means of programming. I selected Java as the programming language of choice because it provides a diverse and structured modus operandi with which I am familiar, and extended the most diverse capabilities and platforms to run on.

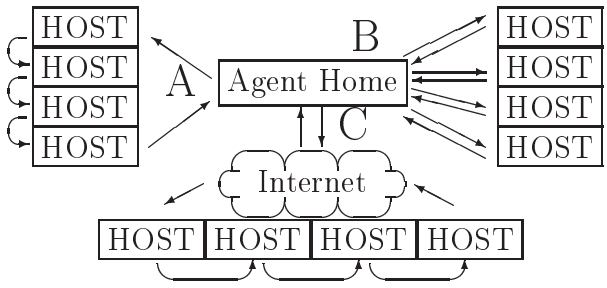

Figure 2.1: Basic Agent Architectures

Figure 2.1 encapsulates various forms of architectures of mobile agent systems.

Element A illustrates the classic example of an agent migrating from one host to another, following a list of host addresses or deterministically searching for systems, performing tasks such as gathering information to installing patches. This is typical for distributed and network management computing. Element B shows a typical client/server construction effectively used for agent monitoring or cluster type distributed computing, providing granular control of agent actions. This approach in the scope of agents, produces a less dynamic, more restrictive method of agent use, yet it is still viable and provides a high level of security [1]. Also, this approach typically models advanced distributed computing practices such as the SETI distributed computation model, RPI's twin enumeration project [6, 7], and Beauwolf Clusters allowing for compiling or calculating a portion of complex code. Element C depicts a distributed, less secure approach to Element A by allowing the Agent to roam the Internet and perform the specific tasks assigned to it. The Peer-to-Peer architecture [1] will not be discussed here because the framework does not allow for a higher level of authority, security or efficient scalability. Though security may be implemented in a Peer-to-Peer network application, the responsibility for ensuring the execution of security tasks falls out of the realm of an administrator's control.

2.3 Agent Systems

There are many different common forms of mobile agents in use today: applets downloaded via the web, applications in email attachments, aglets, proxies used in Remote Procedure Call(RPC) and Remote Method Invocation(RMI) stubs just to name a few [25]. The type of agents discussed herein deal with complete code mobility, meaning that at a minimum, code and data will be transmitted from host to host primarily through basic communication and synchronization tools such as RMI, RPC, CORBA or TCP/IP. More complex systems also provide the means to transfer the state of execution of the agent thus allowing dynamic processing. If an agent recognizes a host shutdown, it will have the capability to transfer itself and its execution state to another host. Though very powerful, the programming complexity of this feature is beyond the scope of this thesis.

[1] not pictured

The two most fundamental components of an agent system are the agent host and the agent. Agent host responsibilities include applying security measures, allocating resources, transporting and execution of the agent. Once an agent arrives, the host may enforce security policies, if implemented, or be programmed to allow the agent to perform only basic functions through a common interface the host grants to the agent. Agent transportation involves the originating station sending an agent to another host through any of the common means (i.e. TCP/IP, RMI or CORBA) through a request from the agent or a central control structure. The agent does not have the means to send itself; rather, it relies on the underlying structure and methods of communication for all transportation.

The agent, as described earlier, represents a programmed purpose with the capability to interact with hosts, other agents and possibly with the environment it transverses. These tidbits of code, instantiated with basic data, and a purpose, jump from host to host performing their functions or data collection. This poses numerous security threats for both agent and host, to be described in the next chapter. Typically, an agent will be homed, and initiated at a central server. To illustrate this concept, refer to Figure 2.3 which represents a Leader environment where an agent system maintains agents from a central server, where it starts, loggs and transports the agent to the first host. Another architecture that may be used, Secondary Leader, (Figure 2.3), allows for a client to create an agent and sends it to the central server where it will then continue operations as in the Leader Environment.

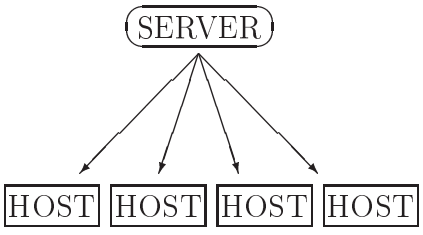

Figure 2.2: Leader Environment

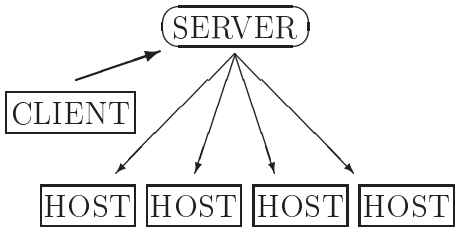

Figure 2.3: Secondary Leader Environment

In the Leader environment, an agent may travel the network in a ring topology,

leaving the Central Server and jumping from one host to another, until it returns to the Central server. In all respects, however, the overall control of the agent relies upon the Central Server, thus we inject the ring topology under the Leader environment.

2.4 Conclusion

I have covered the basic definitions and descriptions of agent and their associated agent systems. I now go in depth as to the purposes and multiple implementations of agent security.

CHAPTER 3
Security Scope

3.1 Introduction

As stated previously, agent systems provide many avenues where malicious hackers may threaten either a component or the communications of the agent system. Because of the threat of security breaches and possible exploitation of agents or data, programmers and agent system designers must be informed of security methods to implement or they risk the potential of violating their clients trust in the agent system they code or use.

3.2 Considerations for Security

The current security paradigm consists of three main goals, Authentication, Confidentiality and Integrity. The first pertains to the authorization of use, transmission or execution. This typically involves a login/password combination or lists of hosts that are allowed to run or receive agents. The second, relates to the secrecy of data, or anonymity of use. And the last goal involves the trueness of data. The following list enumerates various forms of *Security Threats* that pertain to these three categories of security goals.

Alteration: While executing, agents may be modified in their respective object space. When agents are transferred to the next host, opportunities to make modifications to the code present themselves once they enter the copper wired realm and effortlessly when using wireless technology. Communications between agents or between agents and hosts may have message transmissions altered. This poses a risk to the agent and data held within the agent because of the threat of corruption

Unauthorized Access: Pulling or reading data from agent or host without explicit consent by monitoring communications either within the communica-

tions stream or within the virtual machines object space. Unauthorized retrieval of data.

Masquerading: Pretending to be a part of the Agent System through a copy of an agent or duplicating a communication to glean information or confuse the system.

DoS: Abusing the resources of the host. Flooding a part of the Agent System with messages to either disrupt the current work in progress or make it halt.

Repudiation: Denying that an action or communication ever happened. This presents a significant threat, especially when dealing with financial transactions or contracts that effect human life.

This list may be used within the Risk Analysis to identify threats to the system. Once these threats are labeled and evaluated for the amount of risk that they introduce to the system, then solutions can be generated to determine how much security should be implemented.

3.3 Levels of Security

When determining the security risks of an Agent System, the interaction of the Agent Hosts, Agent issuing point (which could be a simple Agent host or a Central Server) and the Agent itself must be taken into consideration. Through these considerations multiple levels of security may be implemented. One of the drawbacks of any type of encryption, be it single key or a pair of Private/Public keys, relates to the fact that additional processing power will be used, thus slowing the agent execution process. This is highly dependent on the type of host and the desired level of encryption. A PDA will take quite a bit longer to decrypt a 128 bit code than a multiprocessor server will. But first let us look at the some ways of hardening an agent system:

Link Encryption: Encryption of the links between two hosts is typically done through Secure Sockets Layer (SSL). Developed by Netscape Communications Corp., the SSL protocol operates between the Network and Session

layer of the Open Source Infrastructure(OSI) Model. Other forms of link encryption include public-key encryption such as PKI(Public-Key Infrastructure), PGP(Pretty Good Privacy), SDSI(Simple Distributed Security Infrastructure), which avoid the transmission of secret keys, and secret-key encryption processed [21, 22, 23]. This process protects data transmitted "on the wire" from one host to another, leaving only a garbled transmission for any sniffer to discover. Before leaving the Host, data is encrypted before reaching the Transport layer. Upon arrival, the Transport layer will release the encrypted data to be unencrypted and processed.

Agent Authentication: Verification of an agent through a central source, ensuring the validity of the agent that is about to execute or be received. Examples of this use and implementation can be seen in these articles [10, 24]

Authorization: Security Policy, implemented on a host, set in place that grants or denies privileges to agents. Through the Security Policy, a programmer may severely restrict an agent or allow control of system functions.

Agent Server/Host Authentication: A security precaution on the agent system verifying the authenticity of the agent server on which it will run. This provides the assurance that a mobile agent needs to execute and certifies a host to which an agent will transfer.

Message Digest/Digital Signature: A digital fingerprint used to check for message tampering during transmission. A digest used with a key is called a Digital Signature and requires the use of a public key to decrypt the digest. The used of a digest/signature provides the security in the form of trust that the data was not tampered with either on a system or through transmission [18, 19]. A digest is made by computing the binary form of the data and calculating a 20 byte hash that is unique specifically to that data at that point in time. A single change to one bit will return a different hash. The effective use of a digital signature also allows for providing accountability of data based on the account information stored in the signature.

Code Signing: Digitally signing a part of the Agent system, from the Agent/Agent Server file(s) to transmitted Agent and/or Agent data files. This signature, allows for the tracking and providing the author of specific code, while also protecting the contents of agent objects. The process consists of creating a wrapper around the code and adding the signature of the signing host/user. Used for both agent and host affirmation, an object may also be "Sealed". This process further secures an agent by encrypting the data within the signed object [20]. To use the object, the data will be sent through a filter, to read the wrapper, verify the contents and the signature and unencrypt (if sealed) the data and pass the object to the calling function.

Auditing: Auditing logs in the agent server, agent and central server are used to protect the system and services. If unchecked, this could lead to a large data block, based on required element logging.

Code Filtering: Jumping Beans implements a powerful method of code filtering in which the agent returns to the central server for "cleansing." The agent code will be checked for alterations when compared to the Agent file stored on the Server. If there are modifications to the code from the last Agent Host, it will be discarded and a fresh copy of the locally stored agent will be used. The agent's audit logs will also be copied to the local Central Server making it impossible for the next Agent Host to glean the past history of the agent that it receives []. However, this feature requires a complex implementation strategy and therefore it was not included in the test reported later.

Agent Encryption: A theoretical concept, where the agent will remain encrypted, even during execution. The Agent Encryption is beyond the scope of this paper and, therefore, it will not be discussed herein.

Implementing each of these hardening methods needs to be weighed against the type of agent system being created. One must understand that the resources involved and possibly the amount of programming required to implement may be beyond the scope of the threat to secure.

Resource $\Longrightarrow$ / Security $\Downarrow$	**CPU**	**Memory**	**Time**	**Bandwidth**	**Number of Messages**
SSL	Low	Low	$\approx 2sec$	$> 4kB$	22
MD5/SHA1	Low	$\approx 700kB$	$\approx 1sec$	150	2
Authorization	Min	$> 1kB$	Min	$> 1kB$	0
Agent Authentication	Min	Min	Min	Var	0
Host/Server Authentication	Min	Min	Min	Var	≥ 2
Code Signing	Min	$\approx 1kB$	Min	$\approx 1kB$	1
Auditing	Min	Var	Min	Var	0
Rating Scheme					
Min- Minimal usage $< 5\%$					
Low- Low Usage $< 10\%$					
Var- Variable Values, based on programming					

Figure 3.1: Resource Matrix

3.4 Costs of Security

In order to determine the feasibility of a particular security implementation, a gauge of the resources consumed must be established. Total time of execution, memory used, CPU utilization, network bandwidth consumed and the number of transmissions are factors tested to determine resource consumption. Baseline measurements were taken from Unix Solaris and Linux systems through the use of *top*, *ps*, *time*, and *tcpdump* evaluating simple JAVA programs implementing each specific security feature. Other methods were also used with the testing programs themselves and will be noted accordingly.

To appraise the cost of security implementation, numerous tests were conducted on basic programs that did minor functions. Each program, written in java, would perform a specific feature, without the hardening implementation. After measuring the amount of resources used, each program then enabled the specific security function in order to determine the overhead incurred . The matrix in Figure 3.1 depicts the amount of overhead each element introduced.

3.4.1 Cost of SSL Usage

Link encryption, through SSL, has numerous resource factors to consider. Bandwidth, number of network transmissions, CPU overhead, memory usage and time are all affected through SSL. The connection will consist of 9 to 14 separate additional transmissions (14 if client authentication is required) on top of the normal *tcp* setup and teardown connection. Further communications between hosts will remain the same throughout the connection without regard to data being encrypted. We can see that through extended use, as the amount of data increases, the cost of the connection, with respect to bandwidth and number of messages, is negligible, while the overhead of memory and CPU utilization due to the encryption and decryption process remains a constant factor. When dealing with nonpersistent connections and small data transfers, bandwidth and the number of messages become a major factor of concern on slow or congested links. Due to different computer system architectures, various CPU and memory values were gathered, due to encryption/decryption. Yet none effected performance or reached levels high enough to warrant higher than a Low rating. Hardware solutions are available to perform these functions in real–time but none were used in any testing ssl and the cost of hardware supported encryption devices currently exceed $1000 and therefore must also be accounted for when deciding on the performance of SSL.

3.4.2 Cost of Agent Server/Host Authentication

Agent and Agent Server/Host Authentication provides the simplest application of security for an Agent System. Actual performance measurements of this implementation are subject to programming requirements in the system. A straightforward method would include setting a simple password at either point and a verification of passwords. A more complex method, by comparison consists of a central server managing all hosts and agents with a unique key so mileage may vary. But for the purpose of this paper we will entertain the concept of the central server holding a valid list of hosts that are allowed to accept and run agents. This server will also maintain a list of agents that are allowed to traverse the network to various hosts. In determining the basic measurements, the simple system test was used. I will not

differentiate between an Agent System that uses a push method and the one that employs a pull method for agent transportation. However, we assume that prior to any agent transmission, the central server and destination will authenticate, and an agent transmission will also be authenticated once with the central server. Through this process, CPU utilization and memory usage have minimal impact on the system, but the number of transmissions and bandwidth used are contributing factors to the overhead at the start of agent execution by adding to the total execution time. Yet, with network speeds growing faster, time would not be a factor on small networks, but will be effected greatly by the number of hosts that can accept agents and the frequency of agent transmissions. Test results on a basic authentication resulted in an additional 6 transmissions (3 for server/host authentication and 3 for agent authentication) with an 8 byte payload consuming an additional $(64 + 8) * 6 = 492$ bytes.

3.4.3 Cost of Authorization

The process of adding a security policy in JAVA requires programming but need few system resources to implement. Once programmed, only the additional steps of programming the agents with a security policy, or modifying the policy on each host are time consuming elements. The only penalty incurred for using a security policy is a little extra bandwidth and a minimal amount of memory to hold the policy, unless the default security policy is used. Yet, when creating an enterprise level Agent System, this will be a standard programming procedure to allow for the capability of file access, granular control and access to system properties.

3.4.4 Cost of Message Digest

Calculating the Message Digest or Digital Signature of an agent also provides a means of authentication, but does not put the process of authentication within control of the agent. This provides authenticity of an agent or its data's transmission. The only concern with regard to resource consumption is the time added to execute, but with systems getting faster, even this is negligible. This test was used with multiple file sizes ranging from 64kb to 1Mb and tested for execution time and memory usage. The resulting checksum is a fixed size of bytes thus resulting in

equivalent bandwidth on all test cases, this is also the case for the number of transmissions. Assuming a simple verification of transmitting the checksum separately from the data, and a reply back of checksum validity, two extra messages are sent with a total of $send(64 + 20) + receive(64 + 2) = 150$ bytes transmitted. Memory usage resulted in an average of approximately 700kB for the digest calculation.

3.4.5 Cost of Code Signing

The process of applying a signature to a file adds one message (confirmation from the receiving side of file authenticity) and adds to the bandwidth the size of the signature $\approx 1kB$. Added resource usage comes from generating the signature and attaching it to the file, but if this is done to agents during compilation then we can safely say the additional resources used in this process do not add to the total resources used during execution.

3.4.6 Cost of Auditing

Agent auditing will require more resources in terms of memory, numbers of messages and bandwidth than time and CPU. This is also contingent on the usage of auditing. If adding data to the agent audit log is frequently necessary, then more memory is needed for storage since holding the transactions in a log rather than writing them to a file is more memory intensive. This also adds to the bandwidth used upon each subsequent transmission to a new host, or return to the central server. And unless purged at periodic points in time, memory consumption will continuously grow in size. With the increased growth of this audit data, the number of messages may increase because of fragmentation, thus also increasing the total execution time. For example, consider Agent A that is required to visit 20 company servers to install a 100KB software patch. It will maintain an audit log that will record at a minimum, the host IP address, time of arrival, time of completion, status of completion, and time of departure. Assuming that each IP address is 4 bytes (unless IPv6 is used, in which case this number will jump to 32 bytes), each timestamp is the number of seconds that have elapsed from the time the agent left the Central Server and stored in 4 bytes, and a 2–byte status. Hence, we can calculate that each audit host will generate a minimum of $4+4+4+2+4 = 22$ bytes

of data at each server visited. A total of $20 * 22 = 440$ bytes will be collected in the audit log, estimating conservatively. For those agents that are traversing large number of hosts, or generating more detailed data in the audit logs, this number could be proportionally higher.

3.5 Conclusion

Understanding what objective an agent system has is crucial to determining its security requirements. This goes hand in hand with what type of systems they will run on and the level of trust between hosts, the network, and agents. This becomes especially critical since a break at any point could mean the potential loss of revenue for a corporation, loss of sensitive personnel data, or something as simple such as a chat or a basic program patch. This chapter presented the knowledge to better understand why we need security through threats, and has demonstrated the different ways to combat those threats. But countering each security threat demonstrably increases programming complexity, system resource usage and time delay. These costs must be considered when determining a plan of execution acceptable to the scope and goals of the agent system.

CHAPTER 4
Abstract Mobile Agent Model

4.1 Introduction

I have introduced the basic concepts of agent systems by providing an elemental framework, common topology layouts, and core purposes of agent usage. But many agent systems are comprised of numerous basic fundamental functions. Once identified, we can sort and arrange these functions into an abstract agent model. This model will be applied to the existing/planned agent system to indicate what is pertinent to the overall system. Such an approach will assist designers and programmers a better understanding of the feasible methods that need to be incorporated into the agent system.

4.2 Components

The model depicted in Figure 4.1, shows the four basic functions of an Agent System. These four functions, Originator, Executor, Communication Network and Agent Communication, depict the fundamental abstract components of most Agent Systems. These functions may be encompassed on one computer or numerous distributed systems; descriptions of each component of the Abstract Agent Model appear below. Within these basic functions, we next identify what we think needs to be addressed in terms of security. Here, programmers and designers contemplate the need for any security implementation based on the possible threat to each component and the risk compromise. A prime example consists of the difference between the type of connection a bank would use versus a general purpose chat program. A bank would not want to have clear text transactions broadcast for fear of having customers' account numbers sniffed. A simple chat program, on the other hand, will not require the security of an encrypted connection for simple text messages (though a plug-in to allow for it may be created). Utilizing an abstract model affords us the opportunity to visualize thh security needs and determine where liability may be accepted and what solutions are available for those places where security plays a

prominent role. This abstract agent model, demonstrates the common traits of most agent systems.

The format for this abstract model follows this **Template**:

a) Specific task/function: Definition of task/function(s)

- Security issues(s)
  - Security Issue Solution(s)

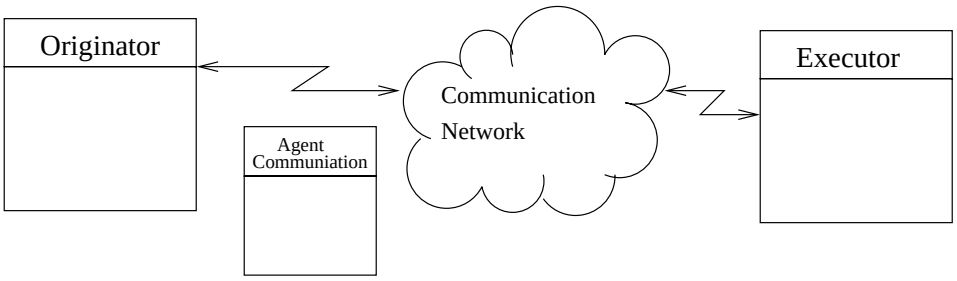

Figure 4.1: Abstract Agent Model

Though not inclusive to all agent systems, this model covers a substantial portion of any agent system.

a) Agent Launcher/Originator: a platform that creates an agent and sends it on a specific task.

- How to identify who created an agent
  - Agent Authentication through central approving site
  - Trusted digital signatures/code signing
- How to authenticate that the agent would not exceed its authority
  - Mobile Agent policy
  - Agent policy on Agent Host
  - Level of authorization of various agents (or deploying users)
- How to negotiate with the originator what the agent can do
  - Permission set on agent

  - Agent "Security Level" listed on Central Site for verification

b) Agent Executor: any computer that provides the means for the agent to run on. This may also be the Originator on single systems.

- How to verify that the agent has not changed during execution
  - Signed Code
  - "Cleansing"
  - Checksum Verification
  - Platform Authentication
- How to verify if the host belongs to the list platforms authorized to received the agent
  - Negotiation between agent and host
  - Central Site that holds a list of authorized hosts
- How to protect agent from revealing privileged information to the host
  - Code interface specifications
- How to protect platform from malicious agent
  - Agent verification through a central site
  - Agent Authentication through central approving site
  - Trusted digital signatures/code signing

c) Communication network for transporting agents: the method of and media of transporting agents from platform to platform. Possible mediums of communication consist of Ethernet, fiber, wireless, local system

- How to avoid tapping, clogging etc.
  - "Cleansing"
  - Central Server agent Verification
  - Trusted digital signatures/code signing
- How Hostile is the communication media

  - Encrypted links
  - Verified list of hosts through Central Server

d) Agent Communication: Inter agent or agent to host communication messages.

- All the problems of the regular communication.
  - Known list of trusted agents
  - Send messages to Central server for disbursement
  - Specific broadcast address and port number
  - Encrypted messages
  - Encrypted connection

4.3 Conclusion

This abstraction of viewing of interactions and the associated components of a mobile agent system give great understanding as to how to go about breaking down specific functions that indicate the need and point for implementing or increasing security. Now we can classify the threats according to vulnerabilities that they target and program methods according to vulnerabilities that they try to protect. Moreover, coupled with the last chapter on Security, we can assign costs, relative to attacks (what happens if the adversary is successful) and to defenses (what is the cost of applying methods such as encryption) to specific agent system components and identify the feasibility of such implementation.

CHAPTER 5
Risk Management

5.1 Introduction

In order to fully develop the security considerations intended for integration, let us evaluate which risks are considered acceptable and which are critical. Clearly, it would be best if a complete system, devoid of risks, could be implemented. Yet, due to time constraints, available hardware, and the priorities of customer needs, programming a perfectly secure system may not always be practical. To identify what is deemed important to the overall goal of the system, a developer must run through multiple levels of analysis to determine the security hazards present in the system. Of these hazards, the designer must assess what risks, if any, are acceptable. In, in order to do this, however, a formal method of identification and organization makes tracking, solving and implementing less chaotic. To do this, we will use a system that the military uses for every mission, a Risk Management Process [11]. This worksheet is constructed to identify risks to the mission. A project leader plans for anything that will deter or detract from the system's success, from the type of environment and systems in which the programs run, to the different levels of access afforded to ordinary and specific users.

5.2 Basics

Initial planning requires that the risk be identified. Once identified, a risk level is applied. This risk level will be plotted on a matrix on the basis of severity and probability that determines the comfortable level of risk that is acceptable. Next, measures of control are addressed to combat the risk previously stated. Once done, pseudocode is written to layout the implementation to provide a feasible plan. In this thesis, pseudocode will not be created, for there are various sources available that describe how to implement such codes in each specific programming language. A reassessment of the original risk level will be conducted based on the implementation of the code. Once complete, testing and evaluation is performed under supervision.

A more comprehensive explanation appears below.

5.3 The Five Step Process

The Risk Management Process is conducted in five simple steps. This processes is initiated once tasks and responsibilities of an agent system are established. Figure 5.1 demonstrates a simple flow of how the steps are worked through from genesis to evaluation. This process need not be run on a complete agent system. Rather, it may be conducted on a small portion of an agent system if needs dictate. To be sure, various security practices may effect and be related to different agent system components. The beauty of this process, however, is in how scaled the security posture can be for each component/project. Though this figure depicts steps of operation, this process is actually fluid, meaning that decisions in one step may affect previous steps. Thus, a constant change may better define how security is implemented. To put these steps into a more tangible form, consider using Appendix A. This appendix presents a set of instructions (Table A.1) and illustrates a blank Risk Management Worksheet (Figure A.2) that will be used in demonstrating the Risk Management Process. Before going through each of the steps, the tasks need to be written onto the form in section D of the Risk Assessment Worksheet. This provides the main focus of the Risk Assessment Process, tailoring the process to each specific task, thus eliminating possible irrelevant tangents.

5.3.1 Step 1, "Identify Risks"

The initial step of security risk assessment is identifying what elements will break the system or compromise the confidentiality of its data. In fact, many points within the Agent System may be broken within the Agent System, but a careful gauge of importance is also required. This is one of the most crucial steps since not only does "Identifying Risks" initiate the security posture of the Agent System but it also helps define the elements the programmer, the designer and the company deem important.

Some issues to consider would include, data protection, agent protection, link protection, authenticity of executor and all other risks listed in section 4.2. *Example:*

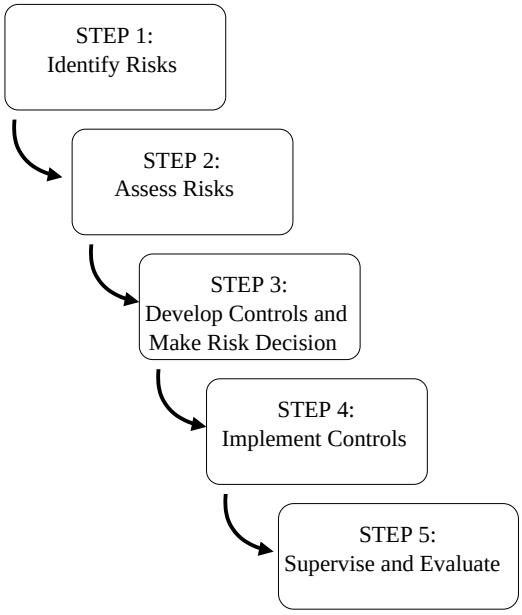

Figure 5.1: Risk Management Process

Sniffed packets on the data stream would provide user ID and password. These risks are listed in section E of the Risk Assessment Worksheet.

5.3.2 Step 2, "Assess Risks"

This step identifies the impact on the system or those concerned if a breach occurs. The designer or programmer needs to assess each risk through a three level sub-process. The first sub-process relates to the probability of occurrence of an incident as depicted in Table 5.1. These values will be subject to change for each type of risk. *Example:* A controlled, closed network would not generate the same probability of a malicious host sniffing the data stream as the threat of sniffing from the Internet.

The second sub-process in Step 2 refers to the severity of each hazard. Possible ways to evaluate the severity are past occurrences, financial obligations to the customer, or overall reliability of system performance. Table 5.2 provides differing levels of severity.

The final sub-process in Step 2 combines the data we have already gathered into a matrix like that in Table 5.3 to show the estimated level of risk for each

Frequent(A): Occurs often, continuously experienced	
Probability:	Occurs continuously during the execution of the program. It is expected to happen frequently. Expected to happen continuously.
Likely(B): Occurs several times	
Probability:	Occurs very often. Expected to happen several times.
Occasional(C): Occurs sporadically	
Probability:	Will occur at some point. Occurrence is not surprising.
Seldom(D): Remotely possible	
Probability:	May occur during execution. Occurrence is rare and isolated.
Unlikely(E): Will not occur, but not impossible	
Probability:	Safe to assume it will not occur. Occurs very rarely, but not impossible.

Table 5.1: Step 2: Risk Probability

identified hazard in Step 1. The matrix presented may change with each Agent System and one is depicted as an example of an average matrix. The levels of risk may be modified to encompass the specific goals of the project. Plotting the gathered severity and probability values to extract the level of risk is straightforward. Once determined, the result is placed in Section F of the Risk Assessment Worksheet.

Through Steps 1 and 2, we determine what liabilities the agent system presents and how detrimental they may be to the overall execution. The risks involved include malicious hackers gathering data from the network stream, power outages that would result in loss of an agent, host or data, and possible network congestion that would delay possibly time sensitive processes or cause a loss in data transmission. The amount of programming knowledge, time necessary to create the agent system and the availability of resources available also add to considerations for development. These first few processes build processes, builds the initial documentation of security practices viewed for this project.

Catastrophic(I)	
Severity:	Extreme loss of vital information that will compromise an individual's or corporation's privacy of financial information or equivalent privacy. Fatal compromise in system integrity allowing for the loss of vital information or execution of malicious code
Critical(II)	
Severity:	Greatly degraded capability in execution or loss in information.
Marginal(III)	
Severity:	Degraded functionality in system usage. Information lost does not hinder or effect system integrity or capability
Negligible(IV)	
Severity:	Loss does not effect or greatly impact system performance or capability. Information lost is trivial.

Table 5.2: Step 2: Risk Severity

5.3.3 Step 3, "Develop Controls and Make Risk Decisions"

Once security hazards have been identified and classified at various risk levels, controls must be established to eliminate or at least reduce the level of risk. By pairing the appropriate Security method that hardens the respective agent system component from Chapter 4, controls can then be identified and an evaluation of the relevance and capabilities accomplished. Are the resources available to implement the stated controls? Will the benefit of the control justify the resources and time needed to develop and execute it? Will implementing this control create more risks that may ultimately threaten the Agent System? These are some of the decisions that require careful attention. Through brainstorming and organizing of pseudocode, a security structure emerges steadily. Once complete, reevaluation of the identified security hazard is made against the Risk Matrix Table 5.3 to determine if the risk has been reduced or eliminated. *Example:* An agent system being

Risk Management Matrix						
		Probability				
Severity		**Frequent (A)**	**Likely (B)**	**Occasional (C)**	**Seldom (D)**	**Unlikely (E)**
Catastrophic	I	E	E	H	H	M
Critical	II	E	H	H	M	L
Marginal	III	H	M	M	L	L
Negligible	IV	M	L	L	L	L
E- Extremely High Risk **H-** High Risk **M-** Moderate Risk **L-** Low Risk						

Table 5.3: Step 2: Risk Matrix

developed to install security patches to time critical software on a corporation's server bank. The operators believe that the risk of a power outage is "Likely", and that the loss of an agent is "Critical". As a result, their overall assessment based on the Risk Matrix would be "H" for high. Thus auditing will provide an excellent tool with which to control and maintain accountability of servers in the event of a power loss. After war gaming the feasibility of pseudocoding the agent system to implement this feature, the operators then re-assess their initial finding of the loss of an agent from "Critical" to "Negligible" based on the fact the agent system maintained an audit log of its progress thus reducing the Risk Level from "H" to "L". The controls developed to mitigate the possible effects of the identified risk are placed in Section G, while the Residual Risk is placed in section H of the Risk Management Worksheet.

5.3.4 Step 4, "Implement Controls"

Programmers ensure compliance with the overall decisions that have been made. Here, a better feel for the complexity of control and design take over and, if need be, modifications to the existing plan may be in order. This "How To" is placed in section I of the Risk Assessment Worksheet.

5.3.5 Step 5, "Supervise and Evaluate"

During development and after the completion of the Agent System, trial runs are made to evaluate the success or failure of the decisions made to defeat the security hazard. A reassessment of the current plan must be undertaken to evaluate whether the security controls meet the intended expectations.

Once the form is completed, for each task, a detailed look at the listing of Section H of the Risk Assessment Worksheet (Determine Residual Risk) will need to be conducted to identify the highest risk within the list. This will determine the overall risk level of the system/project and Section J of the Risk Assessment Worksheet should reflect that value. A completed sample A.3 worksheet is provided to aid in the understanding of this process.

5.4 Application

The methods mentioned previously will now be applied to an agent system called DOORS which is a system for agent based network performance data collection [2]. Agents travel to the specified network device to measure, collect, process and then send performance data to the repositories distributed over the network. Building an outline of the detrimental risks to the system became the first step of our study. As mentioned before, Agents are constructed to perform tasks and return with or send back the data. In this case, agents are built dynamically on a repository server based on user needs. During this creation, an agent is assigned the router to poll and performance data to collect, including any processing that needs to be accomplished before the collected data is sent to the repository. DOORS operates on both a local network and over the Internet using several components to retrieve and process network data. A simple graphical representation of these components working together for a local data request (involving only one repository) appears in Fig. 5.2. In this section, we will briefly discuss the purpose of these components.

5.4.1 Repository

The repository controls agents and coordinates requests from different clients as well as other repositories. Depending on the appropriate action for a particular

client request, the repository will either consult the database, create a new agent, send a message to expand the role of an existing agent, or simply add another client to the list of already distributed data.

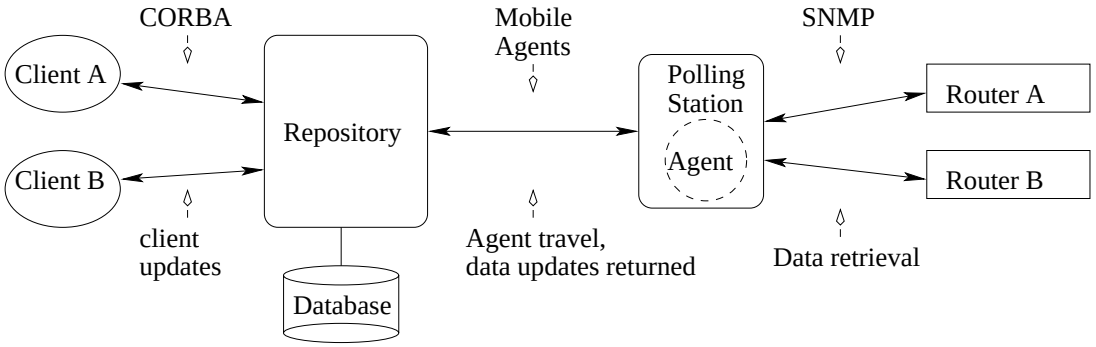

Figure 5.2: DOORS Architecture

5.4.2 Mobile Agents and the Polling Station

The mobile agents travel from the repository to a polling station to request the actual data from the destination object at a close range. Agents reduce the network load by passing back only data necessary for the client and the repository. This is especially useful when additional processing functionality is added to the agent. In such a case, the agent returns only the result of some computation or manipulation of data.

DOORS agents communicate with their sender using the TCP protocol that provides acknowledgment–based reliability. In contrast, traditional SNMP uses UDP that allows for undetected data losses during communication. The reliability benefits of TCP are desirable but come at the cost of increased bandwidth needed for acknowledgments.

The polling station is a critical component since it is difficult to send an agent to sit on the router itself. Many routers use proprietary operating systems. In addition, running the collection programs directly may introduce unacceptable loads on the routers. The polling station simply needs to run an agent server that can receive the agents and allow them to execute their tasks close to the managed node.

5.4.3 Other Components

We also use two minor components on an infrequent basis. CORBA name-server resolves object references for client-repository communication while a super-server correlates polling stations and managed nodes.

5.4.4 Application of Risk Assessment to DOORS

Protection of the agent data during transmission is of paramount importance since this data reveals a router and its community name. With this information, a malicious hacker could launch a wave of Denial of Service attacks during which an overwhelming number of SNMP requests could cause a router to fail.

Task Transmit of agent from repository to polling station and back.

Identify Risk: Router IP and Community name in Agent Data.

Assess Risk: High.

Develop Controls Implement link encryption for the transmission of initial agent transfer.

Determine Residual Risk Low

Implement Controls: Require link encryption, from the Repository to the Polling Station, through SSL. Modify code to allow for modular switching between normal and encrypted transmissions.

Impact on System: Additional processing needed for encrypting and decrypting of keys. Registration of key control is needed as well.

Viewing Figure 5.2 again, we see a reduction to the risk of line sniffing, between the repository and the polling station, though link encryption.

To prevent any user from requesting SNMP data, an action which would be a security risk since sending multiple agents may flood a polling station and the SNMP requests from those multiple agents would overwhelm the polled device, client authentication should be used.

Task Client request of SNMP data

Identify Risk: Unauthorized request of SNMP data

Assess Risk: Moderate

Develop Control Employ user client authentication

Determine Residual Risk Low

Measure of Control: Only registered users/client licenses will have authorized access to the repository, verified through encrypted user/password tables or registered licenses on the repository

Impact on System: Integration with existing user/password database through Kerberos or other OS specific means may simplify implementation of this feature. Cataloged used and issuance of licenses increase complexity of system management.

5.5 Future Work

This work demonstrates only a specialized scope of the effectiveness of Security through a Risk Management Process. While agent systems play a growing function in the present and near term security field concepts, they also pose a significant problem in the future. Currently, intrusion detection systems are in development to aid in identifying infringement through sniffing data [9]. With the threat of network data being picked out of the data stream through spoofed MAC addresses, false credentials, and weak encryption, a strong security posture, as opposed to ease of programming, will better suit a company's agenda while maintaining a better reputation in the eyes of the client. With this, the Risk Assessment Process can further promote the security conscious platform a company may portray. Another useful venture will be its use in targeting threats in a company's current network plan by focusing solely on the security aspect of network security. From firewalls and email servers to CAT 5 cable and wireless transmissions, these are all definitive components overflowing with ways for a malicious hacker to gain unwanted access.

5.6 Conclusion

In many projects, security methods are attached after the fact rather than being integrated from genesis. As a result, many security implementations are patched, added, or wrapped around the existing framework. Still other security components that were belatedly integrated into the system may be weak at best (not verifying user input, buffer overflows, etc.). I have reviewed in this paper, numerous security concepts and the effects that they presented to the system during execution. Armed with these separate security features, and how they fall in line with the Abstract Mobile Agent Model, a well planned security map, with the help of the Risk Assessment Process, will pave the way for a more concise program. I have shown that a simple process exists that can incorporate many different security implementations. This knowledge, coupled with appropriately illustrative object lessons and workplace experiences, should underscore the belief that there is no excuse for an incomplete security blueprint.

LITERATURE CITED

[1] Aramira. Jumping beans security. White paper, Aramira Corp, September 2002.

[2] A. Bivens, P. Fry, L. Gao, M. Hulber, B. Szymanski, and Q. Zhang. Distributed object-oriented repositories for network management. In *Proc. 13th Int. Conference on Software Engineering*, pages CS169–174, Las Vegas, NV, August 1999.

[3] A. Bivens, R. Gupta, I. McLean, B. Szymanski, and J. White. Scalability analysis for a network management middleware. submitted to IEEE International Conference on Communications, May 2003.

[4] D. Chieng, I. Ho, A. Marshall, and G. Parr. A mobile agent brokering environment for the future open network marketplace. In *Proc. of 7th International Conference on Intelligence in Services and Networks, (IS&N 2000)*, pages 3 – 15, Athens, Greece, Febuary 2000.

[5] J. A. Dickie. Modeling robot swarms using agent-based simulation. Master's thesis, Naval Postgraduate School, June 2002.

[6] P. Fry, J. Nesheiwat, and B. K. Szymanski. Computing twin primes and brun's constant: A distributed approach. In *Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing*, pages 42–49, 1998.

[7] P. Fry, J. Nesheiwat, and B. K. Szymanski. Experiences with distributed computation of twin primes distributions. *Parallel and Distributed Computing Practices*, 2:293–313, 1999.

[8] M. S. Greenberg, J. C. Byington, T. Holding, and D. G. Harper. Mobile agents and security. *IEEE Communications Magazine*, pages 76–85, July 1998.

[9] S. McClure and J. Scambray. Once-promising intrusion detection systems stumble over switched networks. *InfoWorld*, 22:58, 2000.

[10] J. Musser and P. Feuer. All that jaas scalable java security with jaas. *Java World*, September 2002. http://www.javaworld.com/javaworld/jw-09-2002/jw-0913-jaas.html.

[11] D. of the Army. *FM 100-14, Risk Management*, April 1998.

[12] B. A. Osborn. *An Agent-Based Architecture for Generating Interactive Stories*. PhD thesis, Naval Postgraduate School, September 2002. Computer Science.

[13] R. S. Pressman. *Software Engineering : A Practitioner's Approach, 4th Edition*. McGraw-Hill, August 1996.

[14] T. Sundsted. Agents can think too! *Java World*, November 1998. http://www.javaworld.com/javaworld/jw-10-1998/jw-10-howto.html.

[15] T. Sundsted. Agents on the move. *Java World*, July 1998. http://www.javaworld.com/javaworld/jw-07-1998/jw-07-howto.html.

[16] T. Sundsted. Agents talking to agents. *Java World*, September 1998. http://www.javaworld.com/javaworld/jw-09-1998/jw-09-howto.html.

[17] T. Sundsted. An introduction to agents. *Java World*, June 1998. http://www.javaworld.com/javaworld/jw-06-1998/jw-06-howto.html.

[18] T. Sundsted. Signed and delivered: An introduction to security and authentication. *Java World*, December 1998. http://www.javaworld.com/javaworld/jw-12-1998/jw-12-howto.html.

[19] T. Sundsted. In java we trust. *Java World*, January 1999. http://www.javaworld.com/javaworld/jw-01-1999/jw-01-howto.html.

[20] T. Sundsted. Signed and sealed objects deliver secure serialized content. *Java World*, November 2000. http://www.javaworld.com/javaworld/jw-11-2000/jw-1117-howto.html.

[21] T. Sundsted. Construct secure networked applications with certificates, part 1. *Java World*, January 2001. http://www.javaworld.com/javaworld/jw-01-2001/jw-0112-howto.html.

[22] T. Sundsted. Construct secure networked applications with certificates, part 2. *Java World*, February 2001. http://www.javaworld.com/javaworld/jw-02-2001/jw-0216-howto.html.

[23] T. Sundsted. Construct secure networked applications with certificates, part 3. *Java World*, March 2001. http://www.javaworld.com/javaworld/jw-03-2001/jw-0316-howto.html.

[24] T. Sundsted. Construct secure networked applications with certificates, part 4. *Java World*, April 2001. http://www.javaworld.com/javaworld/jw-04-2001/jw-0413-howto.html.

[25] B. Tarr, D. Nebesh, and S. Foster. Introduction to mobile agent systems and applications. In *Tools USA 2000*, Santa Barbara, CA, July 2000.

[26] M. Wooldridge. The computational complexity of agent design problems. In *Proceedings of the Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*. IEEE Press, July 2000.

[27] M. Wooldridge and P. E. Dunne. The computational complexity of agent verification. In *Intelligent Agents VIII Springer-Verlag Lecture Notes in AI Volume*, March 2002.

[28] M. Yokoo and K. Suzuki. Secure multi-agent dynamic programming based on homomorphic encryption and its application to combinatorial auctions. In *Proceedings of the First International Conference on Autonomous Agents and Multiagent Systems (AAMAS-2002)*, 2002.

APPENDIX A
Risk Assessment Worksheet

Provided herein explains the process of completing a Risk Assessment Worksheet.

A.1 Instructions of Use

Block	Work Sheet Instructions
A-C	Self Explanititory
D	**Task:** Identify a task relating to the System or Project in Block A.
E	**Identify Risks:** Identify risks detracting from the security conscious goals of the system/project. Additional risks may be from past experience, judgment, or network characteristics.
F	**Assess Risks:** Assess the severity of a risk failure and the probability of occurrence. Determine the initial risk level for each identified risk through the application of the Risk Assessment Matrix (Figure 5.3).
G	**Develop Controls:** Develop one or more controls for each identified risk to reduce, deter or eliminate the risk.
H	**Determine Residual Risk:** Determine the residual risk level for each risk by applying the Risk Assessment Matrix (Figure 5.3).
I	**Implement Control:** Decide how each control will be implemented. This may be done with pseudocode, specified class structures or referenced material.
J	**Determine Overall System/Project Risk:** Select the highest residual risk and circle it. This becomes the overall project/system risk level. Here controls are reviewed to determine if they meet the security goals of thd system/project.

Figure A.1: Risk Assessment Worksheet Instructions

A. System or Project:				B. Date Prepared:	
C. Prepared By:					
D. Task	E. Identify Risks	F. Assess Risks	G. Develop Controls	H. Determine Residual Risk	I. Implement Controls ("How To")
J. Determine the overall System/Project risk level after controls are implemented: LOW (L) MODERATE (M) HIGH (H) EXTREMELY HIGH (E)					

Figure A.2: Risk Assessment Worksheet

A. System or Project: Sample Agent System				**B. Date Prepared:** 02APR03	
C. Prepared By: Joe Snuffy					
D. Task	**E. Identify Risks**	**F. Assess Risks**	**G. Develop Controls**	**H. Determine Residual Risk**	**I. Implement Controls** ("How To")
Agent communications with sensitive user and account number	Loss of account number in communication's stream	High(H)	1. Encrypt the data stream 2. Encrypt the data	Low (L)	Encrypt link with SSL Encrypt data with "Sealed" wrapper
	Maintain accountability of agent to ensure processing of user account	High(H)	1. Maintain audit logs on all hosts 2. Employ Agent authentication 3. Employ Server/Host authentication	Low (L)	Host audit logs that track every all agent movement (with timestamp) in or out of host. Maintain a Central server for Host authentication thus ensuring the transmission to a valid host Maintain agent authentication through unique, one time use, agent IDs, with verification of agent at the receiving host
J. Determine the overall System/Project risk level after controls are implemented: LOW (L) MODERATE (M) HIGH (H) EXTREMELY HIGH (E)					

Figure A.3: Example Risk Assessment Worksheet