T1B2 1555 Proceedings of the 2002 IEEE
Workshop on Information Assurance and Security
United States Military Academy, West Point, NY, 17–19 June 2002

Methodology of Risk Assessment in Mobile Agent System Design

Ingo McLean, Boleslaw Szymanski, Alan Bivens

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 paths 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.**

The purpose of this paper is two-folded. 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, we measure their impact on agent system design and performance. The second purpose is to illustrate the discussed methodology in the context of DOORS, a mobile agent system developed by us for network performance data collection. We will discuss which security features are suitable for DOORS and how they impact the performance of the DOORS system.

I. Introduction

Mobile Agents play a significant role in today's computing environment, from brokering [1] and auction [2] to network management [3] and military simulation [4], [5]. Demand for reliable, unaltered data and/or operations require that their implementations includes assurance of some form of safety and integrity of 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, communications 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 [6]. To start this process, we must first answer a few basic questions. At what point in time should security be involved? What approaches are there to be applied? How should the selected approaches be implemented? What impact will the security measures have on the system performance and robustness?

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 that their data remain intact.

There are numerous mobile agent architectures [7] 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 to determine which risks are acceptable for them and which risks need securing. These decisions will determine how security will be implemented. We will discuss and categorize the main security features for agent systems in this paper.

Each agent system sets unique goals for itself during design and development. In this paper, we focus on an agent system called Distributed Online Object Repositories (DOORS). The goal of this system is to provide a flexible, robust and scalable infrastructure for collecting network performance data. The DOORS system manages and schedules client data requests at its repositories. The repositories then configure mobile agents to travel to a node very close to the managed device. Once the agent arrives at its destination, it polls the managed device, performs client requested procedures, and sends the result back to the repository to be forwarded to respective clients. The use of agents allows us to place more functionality into what the client perceives as the "request." In order to evaluate what security measures need to be set, a methodological dissection of requirements, capabilities and functions needs to be evaluated. Once done, a risk factor analysis based on those requirements will be developed which will allow us

Ingo McLean: Rensselaer Polytechnic Institute, Troy, NY and United States Military Academy, West Point, NY.

Boleslaw K. Szymanski: Rensselaer Polytechnic Institute, Troy, NY. The author acknowledge a partial support from the URP Grant from CISCO Systems.

Alan Bivens: Rensselaer Polytechnic Institute, Troy, NY.

to focus on those security aspects that need to be handled. For instance, a routine agent used for fetching basic webpages for a PDA will most likely sacrifice the overhead of an encrypted link for the sake of performance, while a bank transaction would pay the overhead to ensure security of data, possibly adding encrypted transaction data and user authentication.

This paper first covers agent basics which will lay the groundwork for the upcoming detailed sections. The following sections will cover both the motivation and implementations of security in the application implementation. An Abstract Mobile Agent Model demonstrating the most common to somewhat esoteric components of an agent system will also be presented followed by the Risk Assessment process.

The security methods discussed in the Risk Assessment will then be evaluated in relation to an agent system DOORS for which a selection of features will be made. Finally, a discussion of future work and possible additional considerations for this methodology will be given together with the conclusion.

II. Agent Basics

In this paper, we will focus our discussion on mobile agents as opposed to stationary agents. For clarity, we provide here a basic framework of agents systems and their associated parts [8], [9], [10]. Let us first start with the agent. Agents are autonomous, mobile programs with variable levels of complexity [11], [12], depending on their role which changes from system to system.

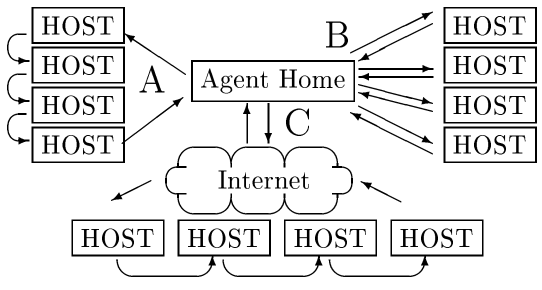

Fig. 1. Basic Agent Architectures

Figure 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, performing tasks such as gathering information or installing patches. This is typical for distributed and network management computing. Element B shows a typical client/server construction effectively used for agent monitoring and granular control of agent actions. This approach in the scope of agents, produces a less dynamic, more restrictive method of agent use, yet is still viable and provides a high level of security [13]. The Peer-to-Peer architecture will not be discussed here because this framework does not allow for a higher level of authority or security.

There are many different common forms of mobile agents in use today: applets downloaded via the web, applications in email attachments, proxies used in Remote Procedure Call(RPC), Remote Method Invocation(RMI) or Jini technologies [7]. 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 will also provide the means to transfer the state of execution of the agent, though this feature is beyond the scope of this paper.

III. Considerations for Security

As stated previously, agent systems provide many avenues where malicious hackers may threaten a component or communication of the agent system. As Fisch and White state in [14] "The goals of a risk assessment are to identify the areas of a computer or network that are susceptible to compromise and to determine the most appropriate protection for those areas. This is accomplished by analyzing three risk attributes: Asset Value, Threat and Vulnerability." The following list enumerates various forms of *Security Threats:*

Alteration: While executing, agents may be modified in their respective object space. When agents are transferred to the next host, modifications to the code and/or data are probable. 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.

Unauthorized Access: Pulling or reading data from agent or host without explicit consent by monitoring communications either within the communications 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 list may be used within the Risk Analysis Process 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 they can be used in an educated risk analysis to determine how much security should be implemented. To further round out the foundations of security, Asset Values should be analyzed, including Availability, Confidentiality and Integrity along with the severity of Vulnerabilities.

A. 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 can be implemented. One of the drawbacks of any type of encryption, be it single key or a pair Private/Public keys, relates to the fact that additional processing power will be used, thus slowing agent execution. 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. This is typically done through SSL.

Agent Authentication: Verification of agent through a central source, ensuring the validity of the agent that is about to be run.

Authorization: Security Policy 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 side verifying the authenticity of the agent server it will be run on.

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.

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.

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 what elements are being logged.

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 agents 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 complex implementation and therefore it was not included in the test reported later.

Agent Encryption: A theoretical concept, where the agent remain encrypted, even during execution. Due to its complexity, it will not be discussed here.

Resource $\Longrightarrow$ / Security $\Downarrow$	**CPU**	**Memory**	**Time**	**Bandwidth**	**# of Messages**
SSL	Low	Low	$\approx 2sec$	$> 4kB$	22
MD5/SHA1	Low	$\approx 700kB$	$\approx 1sec$	$150B$	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					

Fig. 2. Resource Matrix

B. Costs of Security

In order to determine the feasibility of a particular security implementation, a gauge of the resources consumed must be established. Total execution time, memory used, cpu utilization, network bandwidth consumed and the number of transmissions are factors tested to determine resource consumption. Baseline measurements were taken on 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 gauge the cost of security implementation, numerous tests were conducted on basic programs that did minor functions. After measuring the amount of resources used, each program implemented a specific security function in order to determine the overhead incurred. The matrix in Figure 2 depicts the amount of overhead each element introduced.

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 consisted 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 or not. We can see that through extended use, as the amount of data gets higher, 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 encryption and decryption 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. Hardware solutions are available to perform these functions in realtime but none were used in any testing. Yet none effected performance or reached levels high enough to warrant higher than a Low rating.

Agent and Agent Server/Host Authentication provides the simplest application of security for an Agent System. Actual measurements of this implementation depending on how it is programmed into the system. A simple method would include setting a simple password at either point and a verification of passwords, while a complex method consists of a central server managing all hosts and agents with a unique key so mileage may vary. But for the sake of the 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. We 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 an agents 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 of an 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. A basic authentication resulted in additional 6 transmissions (3 for server/host authentication and 3 for agent authentication) with an 8 byte payload consuming additional $(64 + 8) * 6 = 492$ bytes.

The process of adding a security policy in JAVA requires programming and takes little 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 parts. The only incurred penalty 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.

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. Hence, it can prove 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. We measured execution time and memory overhead for multiple file sizes ranging from 64kb to 1Mb. The checksum is of fixed size, thus resulting in the same bandwidth and number of transmissions for all test cases. A simple verification requires transmitting the checksum separately form the data, and sending a reply back of checksum validity. Thus, two extra messages are sent with a total of $send(64 + 20) + receive(64 + 2) = 150$ bytes transmitted. Memory used averaged about 700kB for the corresponding computations.

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 that is about 1kB. Added resource usage comes from generating the signature and attaching it to the file. This can be done during agent's compilation, eliminating this execution overhead.

Agent auditing will require more resources in terms of memory, number of messages and bandwidth than time and cpu. This is also dependent on the usage of auditing. If adding data to the agent audit log is frequent, then more memory is needed for storage. This also adds to the bandwidth used upon each subsequent transmission to a new host, or return to the central server. Unless periodically purged, this additional memory will continue to grow. With the growth of the audit data, the number of messages may increase because of fragmentation, thus also increasing the total execution time. For example, consider Agent A that visits 20 company servers to install a 100KB software patch. It will maintain an audit log that will record 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 a number of seconds expired from the time the agent left the Central Server and stored in 4 bytes, plus 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 detailed data in the audit logs, this number could be proportionally higher.

IV. Abstract Mobile Agent Model

Many agent systems are comprised of numerous basic fundamental functions. Once identified, we can sort and arrange the functions into an abstract agent model. This model will be applied to the existing/planned agent system to point out what is pertinent to the overall system.

Such an approach will allow the designers and programmers a better understanding of the feasible methods that need to be incorporated into the agent system. The model depicted in Figure 3, shows the basic four functions of an Agent System, namely *Originator, Executor, Communication Network* and *Agent Communication.* They depict the fundamental abstract components of most Agent Systems. Each function may be encompassed in one computer or numerous distributed systems. Descriptions of each component of the Abstract Agent Model are given below. Within these basic functions, we identify the needs for any security implementation. 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 transaction going over the wire for fear of having customers account numbers sniffed, while a simple chat program will not require the security of an encrypted connection for simple text messages. Utilizing this abstract model coupled with the Risk Management process described later, afford us the opportunity to visualize and make a decision of 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 uses the following **Template**:

a) Specific task/function: Definition of task/function(s)
• Security issues(s)
– Security Issue Solution(s)

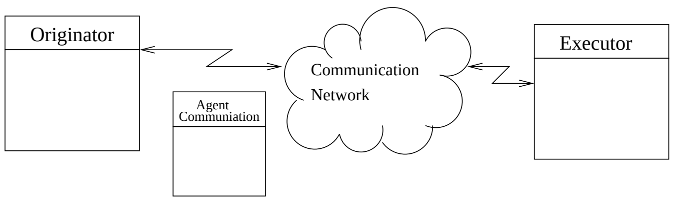

Fig. 3. Abstract Agent Model

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/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/authentication through a Central Site
– Trusted digital signatures/code signing

c) Communication network for transporting agents: the method 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

Now we can classify the threats according to vulnerabilities that they target and method according to vulnerabilities that they try to protect. Moreover, we can also assign costs, negative to attacks (what happens if the adversary is successful) and to defenses (what is the cost of applying a method such as encryption).

V. Risk Management

To decide what security aspects to integrate, let us evaluate which risks are considered acceptable and which are critical. Clearly, it would be best if a system completely devoid of risks could be implemented. Yet, because of time constraints, limits of available hardware, and priorities of the customer needs, programming a perfect 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, if any, what risks are willing to be taken. 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 Worksheet [15]. This worksheet is constructed to identify risks to the mission. A project leader plans for anything that will deter or detract from the systems success, from the type of environment and systems in which the programs run to the different levels of access that the ordinary and specific users have.

Initial planning requires that the risk be identified using the criteria described before. Once identified, then 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. For the sake of the paper's brevity, pseudocode will not be created, for there are various sources out there that describe how to implement such codes in each specific programming language.

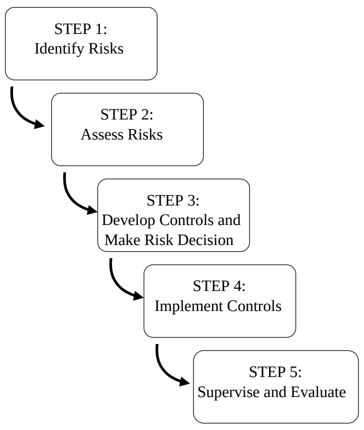

Fig. 4. Risk Management Process

A. The Five Step Process

The Risk Management Process is conducted in five simple steps. Figure 4 demonstrates a simple flow of how the steps are worked through from genesis to evaluation.

A.1 Step 1, "Identify Risks"

The initial step of security risk assessment is identifying what will break the system or compromise data of its confidentiality. Numerous points may be broken within the Agent System, but a careful gauge of importance is also required. This is one of the most crucial steps, not only does "Identifying Risks" initiate the security posture of the Agent System but also sets the tone for what the programmer/designer/company deems as important.

Some issues to consider would include, data protection, agent protection, link protection, authenticity of executor and all other risks listed in section IV. *Example:* Sniffed packets on the data stream would provide user ID and password.

A.2 Step 2, "Assess Risks"

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

The second 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 II provides differing levels of severity. The final process in Step 2, combines what we have gathered into a Matrix such as Table III, to show the estimated level of risk for each identified hazard in Step 1. The matrix presented may change with each Agent

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 I
Step 2: Risk Probability

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 II
Step 2: Risk Severity

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 pretty straight forward. Through both steps 1 and 2, we determine what liabilities the agent system presents and how detrimental these liabilities are 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, or network congestion that would delay possibly time sensitive processes, or loss in data transmission. Amount of programming knowledge, time to create the agent system and resources available also pertain to considerations for development.

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 III
Step 2: Risk Matrix

A.3 In 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 identified risks. Then, an evaluation of the relevance and capabilities must be made. Are the resources available to implement the stated controls? Will the benefit of the control justify the resources/time to develop and execute? Will implementing this control, create more risks that will threaten the Agent System? These are some of the decisions that require careful attention. Once complete, reevaluation of the identified security hazard is made against the Risk Matrix (Table III) to determine if risk had been reduced or eliminated.

A.4 In Step 4, "Implement Controls"

Programmers ensure compliance with the overall decisions made.

A.5 In Step 5, "Supervise and Evaluate"

After the completion of the Agent System, trial runs are made to evaluate the success/failure of decisions made to thwart the security hazard.

VI. CASE STUDY: Application to DOORS

The methods mentioned above was applied to an agent system called DOORS, that, as we stated earlier, is a system for agent based network performance data collection. 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 the creation, an agent is assigned the router to poll and performance data to collect, including any processing that needs to be done before the collected data is sent to the repository. DOORS operates on both a local network and over the Internet. Protection of the specific agent data is paramount because this data shows a router, and its community name. With this information, a malicious hacker could launch a wave of Denial of Service attacks during which too many SNMP request could overwhelm a router causing it to be incapable of functioning properly.
Task Transmit 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 again, we see a reduction to the risk of line sniffing, between the repository and the polling station, though link encryption.

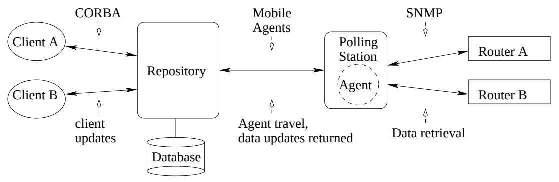

Fig. 5. DOORS Architecture

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 have authorized access to the repository, verify it 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.

VII. Conclusion

Combined testing of various security solutions provides a detailed picture of resource usage and allows a granular approach to allocating resources to security, when the same platforms are used for a variety of systems.

We described a formal method of identifying the risks involved with an agent system through a "Risk Management" approach. This approach allows a system designer to clearly identify the level of detail needed for security implementation. Also presented were the methods of security that a designer could use. Choice of particular methods for a system can then be based on an educated analysis of resource usage for each type of solution implemented.

VIII. Acknowledgments

The authors express their gratitude to the following people and organizations for the help and support. David Kotfila of Cisco Academy made the Cisco Lab at RPI available to us for benchmarking. The Aramira Corporation and its CTO Chris Rygaard donated their software, JumpingBeans®and their expertise to this project. Our colleagues, Rashim Gupta, Jerome White and Kenneth Fritzsche assisted us with testing and editing.

References

[1] 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)*, (Athens, Greece), pp. 3 – 15, Febuary 2000.
[2] 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.
[3] J. A. Bivens, L. Gao, M. Hulber, and B. Szymanski, "Agent-based network monitoring," in *Agent based High Performance Computing*, (Seattle, Washington), Autonomous Agents: Sponsered by ACM Sigart, May 1999.
[4] B. A. Osborn, *An Agent-Based Architecture for Generating Interactive Stories*. PhD thesis, Naval Postgraduate School, September 2002. Computer Science.
[5] J. A. Dickie, "Modeling robot swarms using agent-based simulation," Master's thesis, Naval Postgraduate School, June 2002.
[6] R. S. Pressman, *Software Engineering : A Practitioner's Approach, 4th Edition*. McGraw-Hill, August 1996.
[7] B. Tarr, D. Nebesh, and S. Foster, "Introduction to mobile agent systems and applications," in *Tools USA 2000*, (Santa Barbara, CA), July 2000.
[8] T. Sundsted, "Agents talking to agents," *Java World*, September 1998. http://www.javaworld.com/javaworld/jw-09-1998/jw-09-howto.html.
[9] T. Sundsted, "Agents can think too!," *Java World*, November 1998. http://www.javaworld.com/javaworld/jw-10-1998/jw-10-howto.html.
[10] T. Sundsted, "Agents on the move," *Java World*, July 1998. http://www.javaworld.com/javaworld/jw-07-1998/jw-07-howto.html.
[11] 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.
[12] 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.
[13] Aramira, "Jumping beans security," white paper, Aramira Corp, September 2002.
[14] E. A. Fisch and G. B. White, *Secure Computers and Networks: Analysis, Design, adn Implementation*. Boca Raton, FL: CRC Press, 2000.
[15] D. of the Army, *FM 100-14, Risk Management*, April 1998.