“Bodh”, BPIT’s International Journal of Technology & Management
ISSN: 2454-8421, Volume 2, 2016, Page 20-26

A Way to Trust Deportment for Security in Distributed Systems

Preeti Arora[1], Shipra Varshney[2]

[1]*Assistant Professor, CSE Dept, BPIT, GGSIPU, India*
[2]*Assistant Professor, MCA Dept, NIEC, GGSIPU, India*

***Abstract* – A distributed system is a decentralized network consisting of a collection of autonomous computers that communicate with each other by exchanging messages. These systems are scalable and fault tolerant, and they allow easy resource sharing, concurrent processing, and transparent operation. With the rapid growth of the information age, open distributed systems have become increasingly popular. The need for protection and security in a distributed environment has never been greater. The conventional approach to security has been to enforce a system-wide policy, but this approach will not work for large distributed systems where entirely new security issues and concerns are emerging. Existing authorization mechanisms fail to provide powerful and robust tools for handling security at the scale necessary for today's Internet. We argue that a new model is needed that shifts the emphasis from “system as enforcer” to user-definable policies. Users ought to be able to select the level of security they need and pay only the necessary overhead. Moreover, they must be responsible for their own security. This research is being carried out in the context of the trust-management approach to distributed-system security developed as an answer to the inadequacy of traditional authorization mechanisms with a very popular architecture analyzed of Java by SUN. In this paper, we introduce the concept of trust management, explain its basic principles. We also survey the current research on trust management in distributed systems and explore some open research areas and examine existing authorization mechanisms and their inadequacies.**

***Keywords*-- Java, SUN, Trust Management**

I. Introduction

T**rust** is an important issue in distributed systems. Transactions in distributed systems can cross domains and organizations and not all domains can be trusted to the same level. Even within the same domain, user’s trustworthiness can differ. A flexible and general-purpose trust management system can maintain current and consistent trustworthiness information for the different entities in a distributed system. In e-commerce, for example, a trust-management system lets a buyer and seller become acquainted with each other and estimate the risk of participating in a transaction, thus minimizing the loss. In ***P2P*** systems, where each entity acts as both client and server and is expected to contribute to the system, trust management can help reduce free riding, which can seriously degrade P2P system performance. Finally, in mobile ad hoc networks—a type of distributed system that has no infrastructure and lets nodes move freely, trust management can mitigate node’s selfish misbehavior such as dropping or refusing to forward packets for other nodes to save its battery power while still requiring other node’s services.

Much research exists on trust management and reputation management. We don’t distinguish trust management from reputation management because both can be generalized as dynamic rating systems. Here, we survey the current research on trust management in distributed systems and explore some open research areas.

Trust-management engines avoid the need to resolve “iden-tities” in an authorization decision. Instead, they express privileges and restrictions in a programming language. This allows for increased flexibility and expressibility, as well as standardization of modern, scalable security mechanisms. Further advantages of the trust-management approach include proofs that requested transactions comply with local policies and system architectures that encourage developers and administrators to consider an application's security policy carefully and specify it explicitly.

II. RELATED WORK

A. *Trust Models*

Trust is a complex subject, and no unanimous definition of trust exists. The Merriam-Webster's Dictionary defines trust as "assured reliance on the character, ability, strength, or truth of someone or something." Dictionary.com describes trust as the "firm reliance on the integrity, ability, or character of a person or thing." We define trust as the belief that an entity is capable of acting reliably, dependably, and securely in a particular case. Trust management entails collecting the information necessary to establish a trust relationship and dynamically monitoring and adjusting the existing trust relationship. The various models for describing trust and trust establishment in distributed systems include public-key cryptography, the resurrecting duckling model, and the distributed trust model.

1) Public-key cryptography

Many networked services have security mechanisms based on cryptographic techniques such as the Pretty Good Privacy (PGP) 6 or X.5097 certificate systems, which implicitly use the trust-management concept. A public-key certificate is a digital certificate issued by a trusted third party to certify a public key's ownership. A certificate contains an entity's identity, public key, and other information, such as the trusted third party's digital signature. Service users are assumed to know the trusted third party's public key so that they can verify the certificate. The trusted third party only vouches for the association between an identity and a public key. It doesn't guarantee the entity's trustworthiness. In X.509, the trusted third party is a certificate authority (CA), which is usually a trustworthy entity for issuing certificates (VeriSign, for example). Another CA might also certify a particular CA.
When a user generates a public/ private key pair, it registers its public key with a CA and has the CA certify it. If the same CA certifies two users and they want to communicate securely, they need only exchange their certificates. If different CAs certifies two users, they must resort to higher-level CAs, which certify their CAs until they reach a common CA. So, X.509 uses a hierarchical structure, which constructs a tree of trust.
PGP doesn't use a CA. Instead, every entity certifies the binding of IDs and public keys for other entities. For example, an entity A might think it has good knowledge of an entity B and is willing to sign B's certificate. An entity might assign a degree of trust—unknown, untrusted, marginally trusted, or fully trusted—to its certifiers. The user chooses how to use the certificate. User C might be confident about A's trustworthiness and accept B's certificate, which A has signed. A pessimistic user might only accept certificates certified by fully trusted entities, whereas an optimistic user might trust marginally trusted signers. Traditional certificate schemes like X.509 and PGP only bind public keys to identities. Because binding an identity to access rights or authorized actions is outside the certificate framework, a certificate framework only provides partial trust management.

2) Resurrecting duckling model

Ross Anderson and Frank Stajano's resurrecting duckling model also has a hierarchical structure. The entities in a network have a master-slave relationship. The master entity is the mother duck and the slave entity is the duckling. A slave entity recognizes the first entity that sends it a secret key through an out-of-band secret channel (through physical contact, for example) as its master in a process called imprinting. The master passes instructions and access control lists to its slaves, and the slaves always abide by their master. The master, a time-out, or a specific event can break the relationship between a master and a slave. After that, other entities can imprint, or resurrect, the slave. A slave entity can also become a master to other entities through the imprinting process. Thus, the relationship among nodes is a tree-like trust relationship. An entity controls all the entities in its subtree. Breaking the relation between two entities causes the relationships in the entire subtree to break. This model is appropriate for devices that can't perform public-key cryptography. However, the model requires an out-of-band secret channel to deliver the secret key, which might not be feasible in some networks, such as ad hoc networks.

3) Distributed trust model

The distributed trust model assumes asymmetrical trust.

Stephen Hailes and Alfarez Abdul-Rahman developed a distributed recommendation-based trust model. They propose conditional transitivity of trust, which hypothesizes that trust is transitive under some conditions.
For example, if A trusts B, and B trusts C, we can't simply conclude that A trusts C, because trust generally isn't transitive. Abdul-Rahman and Hailes claim that we can conclude that A trusts C if the following conditions are true:

• B recommends its trust in C to A explicitly;
• A trusts B as a recommender; and

• A can judge B’s recommendation and decide how much it will trust C, irrespective of B’s trust in C.

Although, using trust management is a recent research field, there are many works in this area. We also survey the current research on trust management in distributed systems and explore some open research areas and examine existing authorization mechanisms and their inadequacies. There are three popular architectures for distributed systems applications and their security implications. The architectures analyzed are Java by Sun, CORBA by the OMG, and COM+ from Microsoft. It is extremely important for developers to consider the security implications when designing distributed applications, as many of these applications offer access to crucial resources: financial, medical, and military information, just to name a few [2].
The model’s motivation comes from human society, where human beings get to know each other via direct interaction and through a grapevine of relationships. The same is true in distributed systems. In a large distributed system, every entity can’t obtain first-hand information about all other entities. As an option, entities can rely on second-hand information or recommendations. However, because recommendations have uncertainty or risk, entities need to know how to cope with second-hand information. The distributed trust model assumes asymmetrical trust. It defines two types of trust relationships: direct trust and recommender trust. It categorizes a trust relationship between two entities in terms of different interactions. Trust in one category is independent of trust in other categories. This model uses continuous trust values for direct trust and recommender trust and can define values. Other researchers fix trust value within the range (0, 1). The recommendation protocol is straightforward. For example, entity A needs a service from entity D (say car service). A knows nothing about the quality of D’s service so A asks B for a recommendation with respect to the car service category, assuming that A trusts B’s recommendation within this category. When B receives this request and finds that it doesn’t know D either, B forwards A’s request to C, which has D’s trustworthiness information within the car service category. C sends a reply to A with D’s trust value. The path A _ B _ C _ D is the recommendation path.
We use the following formula to calculate the trust value from the returned value : tv_T = [rtv(1)/4] _ [rtv(2)/4] _ ... _ [rtv(i)/4] _ ... _ [rtv(n)/4] _ tv(T), where rtv(i) is the trust value of the ith recommender in the recommendation path, tv(T) is the trust value of target T returned by the last recommender, and tv_T is the calculated trust value of target T.
When multiple recommendation paths exist between the requester and the target, the target’s eventual trust value is the average of the values calculated from different paths.
This model has some weaknesses:

- It doesn’t consider false recommendations and assumes that a recommender with a good recommender trust value always makes reliable recommendations, which might not be true.
- It doesn’t provide a mechanism for monitoring and reevaluating trust, which is dynamic.

Trust shouldn’t be considered a binary concept (that is, either to trust or not to trust). Hailes and Abdul-Rahman quantified trust as a multiple value concept. Many trust-management systems use the same approach. The key challenge then is how to process the trust values to minimize the influence of false recommendations.

B. Trust Management

Trust Management, introduced by Blaze et al. [BFL96], is a unified approach to specifying and interpreting security policies, credentials, and relationships that allows direct authorization of security-critical actions. In particular, a trust- management system combines the notion of specifying security policy with the mechanism for specifying security credentials. Credentials describe specific delegations of trust among public keys; unlike traditional certificates, which bind keys to names, trust-management credentials bind keys directly to authorizations to perform specific tasks. Trust-management systems support delegation, and policy specification and refinement at the different layers of a policy hierarchy, thus solving to a large degree the consistency and scalability problems inherent in traditional Access Control Lists (ACL). Furthermore, trust-management systems are by design extensible and can express policies for different types of applications.

1) Trust Management in Mobile AD HOC Networks

P2P systems assume that the network layer is reliable and that data delivery, such as request and response, can be guaranteed. This isn’t true for ad hoc networks. Therefore, it isn’t directly possible to apply the previous approaches to trust management in ad hoc networks. An ad hoc network relies on all participants actively contributing to network activities such as routing and packet forwarding. An ad hoc network’s special characteristics— such as limited memory, battery power, and bandwidth—can cause nodes to act selfishly (refuse to participate in routing and provide services to other nodes, for example). Trust management can help mitigate this selfishness and ensure the efficient utilization of network resources.

2) Monitoring-based trust-management systems

In ad hoc networks, a node can only sense the packets transmitted within its transmission range. Sonya Buchegger and Jean-Yves Le Boudec's Confidant (Cooperation of Nodes, Fairness in Dynamic Ad Hoc Networks) protocol promotes cooperation in ad hoc networks by detecting and isolating malicious nodes.

Each node in the network runs the Confidant protocol. Confidant's monitor component observes the behavior of neighbor nodes to detect misbehavior, such as packet dropping. This requires nodes to run in promiscuous mode. When the monitor finds misbehavior, it notifies the reputation system, which manages a table containing nodes and their ratings. The rating is a number within a certain range depending on the implementation. If the number of times a node misbehaves exceeds a threshold, the reputation system updates the node's rating. If a node's rating falls below a threshold, the system considers it a malicious node. The reputation system maintains a blacklist containing the malicious nodes. When forwarding packets, nodes avoid next-hop nodes on the blacklist.

When the reputation system detects a malicious node, it notifies the trust manager to broadcast an alarm message in the network. Trust managers also receive alarms from other trust managers. A trust manager only distributes and accepts alarms from senders on its friends list. (Establishing friendship is a research topic. One possible method is the resurrecting duckling model.) Each trust manager maintains a table with the trust levels of received alarms.

The path manager ranks the path according to the ratings of the nodes on the path. It deletes all paths containing malicious nodes and drops route requests received from malicious nodes.Buchegger and Boudec didn't discuss how to compute reputation values. In addition, Confidant can't prevent malicious nodes from disseminating false information about other nodes, and trustworthy nodes can lie.Sergio Marti and his colleagues proposed two methods to improve an ad hoc network's throughput in the presence of misbehaving nodes: a watchdog method and a path rater method. They assumed that a wireless interface supports the promiscuous mode.

The watchdog is a misbehaving node locator running on every node that maintains a buffer of recently sent packets. After overhearing a packet, the watchdog com pares it with the packets in the buffer to see if there's a match. If there is, the packet has been forwarded and the watchdog removes the packet from the buffer. If a packet stays in the buffer for longer than a certain period, the watchdog increases a failure count for the node responsible for forwarding the packet. If the count exceeds a threshold value, the watchdog considers that node as misbehaving.

A path rater at a node maintains a rating for every other node that it knows in the network. To pick a route that is most likely to be reliable, it computes a path metric by averaging the rating of the nodes on the paths and chooses the path with the highest metric. It assigns misbehaving nodes a very low rating, and thus excludes them from routing.

Because of ad hoc networks' characteristics, the proposed approaches can't accurately detect misbehaving nodes in situations such as packet collisions and collusion of malicious nodes.

3) Evidence-based trust management

Eschenauer and his colleagues present a framework for trust management in ad hoc networks based on evidence distribution. They consider trust as a set of relationships established with the support of evidence. In their framework, evidence can be anything a policy requires to establish a trust relationship, such as public key, address, and identity. Any entity can generate evidence for itself and for other entities. Evidence can be obtained either online or offline, such as through physical contact.

One way to generate evidence is through public-key cryptography. An entity can create a piece of evidence, define its valid time, sign it with the entity's private key, and disseminate it to others. To verify this piece of evidence, other entities will need the originator's public key and certificate. In the Internet, entities can use X.509. However, in an ad hoc network, where there is no CA, PGP might be an option. An entity can invalidate its evidence by generating a revocation certificate at any time.

Eschenauer and colleague's approach also lets an entity revoke other entities evidence by generating and disseminating contradictory evidence. However, allowing such actions is open to attack. A malicious entity can distribute bogus evidence to invalidate other node's legitimate evidence, which can cause chaos in the network. A malicious entity might generate fake evidence for its own purposes—for example, to impersonate other nodes.

To prevent these attacks, Eschenauer and his colleagues proposed using redundant and independent evidence from various sources. However, they didn't discuss how to evaluate evidence, which is important for trust management. Also, because each node's trustworthiness is not dynamically adjusted, the framework is mainly useful for authentication.

4) Trust Management in E-Commerce

Trust or reputation management is an important issue in e-commerce, where traders might have never met and know nothing about each other's trustworthiness. This lack of information about traders' reputations causes uncertainty and mistrust, which influences the e-market's economic efficiency.

Considerable research has explored trust and reputation management in e-commerce. One possibility is to build a centralized system, like a credit history agency, to manage users' reputations. However, this approach neglects personal preferences and standards. Online auction and shopping sites, such as eBay and Amazon.com, use reputation management. eBay assigns sellers a rating of 1, 0, or –1 for trustworthiness after one interaction, and computes a seller's reputation as the accumulation of all the ratings received within the past 180 days. New eBay users receive a reputation of 0. Amazon.com rates both sellers and buyers after each interaction. It calculates reputation as the average of all the feedback ratings received during the system's use. A new Amazon.com user has no reputation value. Users can easily misbehave in e-marketing. After cheating and obtaining a bad reputation, a user can simply discard a current identity, obtain a new one, and reenter the market. This kind of misbehavior causes low economic and system utilization efficiency. To solve this problem, Amazon.com and eBay apply pseudonyms. New users must register with some personal information so the system can trace their real identity. At the same time, pseudonyms provide anonymity.

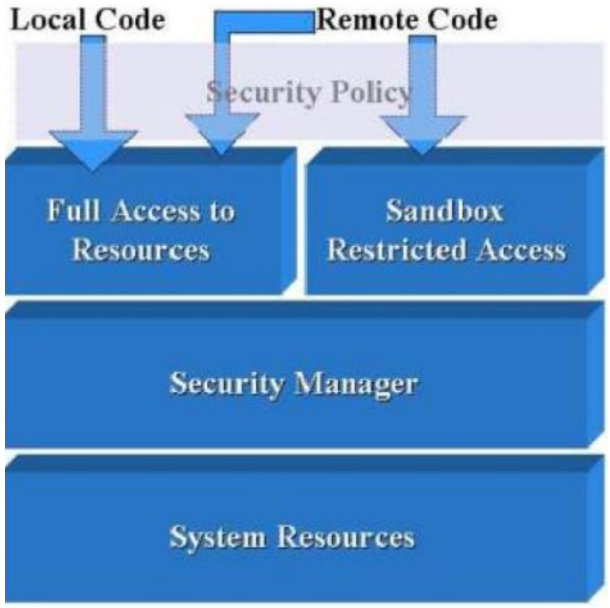
III. Methodology of Distributed Systems – Java

The Java architecture for distributed systems computing was designed taking security requirements into consideration. Developers need to create programs that are executed on remote distributed systems. An architecture needed to be put in place, however, that would not leave these systems vulnerable to malicious code. This was accomplished through the Java architecture. The source code is written and then converted to byte code and is stored as a class file, which is interpreted by the Java Virtual Machine (JVM) on the client. Class loaders then load any additional classes that are needed by the applications. Several security checks are put between the remote server distributing the program, and the client executing it, such as the "sandbox" security model, them byte code verifier, the applet class loader, the security manager, and through other security measures that can be implemented through Java's security APIs.

A PROPOSED MODEL

1) Sandbox Security Model

In a distributed architecture, the end users would ultimately be responsible for determining which applets to run on their systems. Most of these users would not be able to determine whether a particular applet should be trusted or not. In order to have all applets run in a protected environment, the sandbox security model was developed. Applets that run from a remote site would be permitted only limited access to the system, while code run locally would have full access. If the applet is signed and trusted, then it can run with full local system access. Permissions can be set by a security policy that allows the administrator to define how the applets should be run.



II.

Figure 1: Java Model.

A. Byte Code Verifier

The byte code verifier looks at the class files that are to be executed and analyzes them based on specific checks. The code will be verified by three or four passes (MageLang Institute, 1998) depending on whether or not any methods are invoked. Gollmann (2001) states that some of the checks performed are to ensure that the proper format is used for the class, to prevent stack overflow, to maintain type integrity, to verify that the data does not change between types, and that no illegal references to other classes are made. Hartel and Moreau (2001) further state that the byte code verifier ensures that jumps do not lead to illegal instructions, that method signatures are valid, access control, initialization of objects, and that "subroutines used to implement exceptions and synchronized statements are used in "FIFO order".

B. Applet Class Loader

As a Java application is executed, additional classes may be called. These classes are not loaded until they are needed. When they are called the applet class loader is responsible for loading the specified applets. Classes in Java are organized by name spaces, and each class loader is responsible for one name space. The class loaders are therefore responsible to protect the integrity of the classes in its name space (Gollmann, 2001). Java has built-in classes that reside locally, however, that are loaded automatically without any security checks. The path to these classes is indicated by the CLASSPATH environment variable.

C. Security Manager

When writing applications, developers often wish to protect variables and methods from being modified by classes that do not belong to the group of classes they have written. In order to create this division, classes are grouped into packages. When a variable or method is declared in a class, it can be private (access only through same class), protected (access through class or subclass), public (any class can access), or they may chose not to use any of the former, in which case only classes within the same package will have access. Depending on the package that a class belongs to, the class will have different access to the other classes in the package, so security could be compromised if an unauthorized class attaches itself to the package. The security manager makes sure that only classes that actually belong to the package in question are able to declare themselves in this package. The security settings are configured through a security policy. Browsers and applet viewers have a security manager, but by default Java applications do not (Sun Microsystems, n/d). Java has provided developers the means to create their own security manager. To create it, the developer must create a subclass of the SecurityManager class, and override whichever methods are necessary to implement the required security. For example, the developer may decide to impose a stricter policy for reading and writing files. This could be attained through overriding the read and write methods already defined in the superclass.

D. API Security

Java offers further security through several security APIs. Among the different APIs provided, the developer can make use of signed applets, digital signatures, message digests, and key management. When an applet is signed it is given full access to the system as if it were run locally. As mentioned in the section on the security manager, the security policy defines what permissions are given to an application or applet when executed. The default Java Runtime Environment provides digital signatures, message digests, and key management, and encryption can be implemented through the Java Cryptography Extension (JCE).

E. Outstanding Issues

As with any system, whether it has been designed around security or not, the Java distributed architecture contains several outstanding security problems. One problem is with the CLASSPATH system environment variable. As mentioned previously, the CLASSPATH variable is used to determine the location of the built-in Java system classes. If the CLASSPATH variable is altered, it could point to a set of altered classes that may execute what the original classes intended, but also insert malicious code. The code would be executed, and the user may not notice any difference in the behavior of the application.

Wheeler, Conyers, Luo, and Xiong (2001) found that there are several Java vulnerabilities if a computer serving Java applications is either compromised from the inside, or if an attacker is able to compromise an account on the server. They note that many of these vulnerabilities exist either because of code that provides backwards compatibility, or because of decisions made to increase the ease of implementation. In other words, the vulnerabilities are due
to design choices rather than software defects. First they found that "many critical components of the Java environment are only protected by the underlying operating system's access control mechanisms". System administrators may not be aware of the loose access controls, and critical components could be compromised, such as the key store and system classes. If the key store is compromised then signed files could be spoofed, and if the classes are modified, malicious code could be inserted. Wheeler et al. further note the ease of reverse-engineering of class files, which would allow an attacker to obtain the original source code. They

note that there are tools for obfuscation, but suggest in their work that further obfuscation would be necessary for a higher level of security.

As discussed earlier, a security policy can be set to limit the access of applications or applets to the local system. Wheeler et al. discuss that the permissions, although fine grained, can only be applied to a directory or JAR file. They state, “this is insufficient, except for the most rudimentary system”. Permissions applied to the entire directory or JAR file, which violates the principle of least-privilege. They suggest finer permissions that could extend down to the class level. The security policy can also be either modified or overwritten completely through the use of the “java.security.policy” option from the command line, negating any work put into the creation of the security policy. This behavior can be turned off, but is not by default – an example of vulnerabilities being introduced for the sake of ease of implementation. They suggest that the class loader should verify that an extended security manager is loaded prior to loading any classes.

Hassler and Then (1998) discuss the possibility of using applets to perform “a degradation of service attack”. Security policies can be created, and are usually part of the browser, to limit the access given to Java applets. They show in their research, however, that this does not prevent the applet from consuming sensitive resources such as CPU and memory. They suggest the implementation of a special applet that would allow other applets to be controlled, and note at the end of their work that the HotJava browser included this, but was found to be insufficient. One must wonder, however, if an average user would have the knowledge necessary to identify that a Java applet is creating the degradation of service, and how to stop it.

Finally, an outstanding issue is that of auditing. A major component of security systems is the ability to audit. Hartel and Moreau (2001) state that there is no known work presently being done to implement auditing capabilities in Java.

IV. Conclusion And Future Work

In the time since trust management first appeared in the literature in [BFL96], the concept has gained broad acceptance in the security research community. Trust management has a number of important advantages over traditional approaches such as distributed ACLs, hardcoded security policies, and global identity certificates. A trust-management system provides direct authorization of the security critical actions and decouples the problems of specifying policy.

V. RESULT

This paper on trust management has focused on one of the most common distributed systems application architectures as Java. Java has several published security vulnerabilities, but knowing what they are is half the battle towards finding a remedy. The difficulty of implementation must also be considered. If the system is overly complex, security problems may exist due to implementation problems. If the architecture is too simple however, there may not be enough flexibility to create the necessary security configurations.

As an area of future work as we are examining higher-level policy languages that are even more human understandable and capable of higher levels of abstraction. Such high- level policy would be combined with network and application specific information and compiled into a set to trust-management credentials. Similarly, a tool for translating trust-management credentials into application-native forms would give us all the advantages of trust management (delegation, formal proof of compliance, etc.) while requiring minimal changes to applications.

References

[1] D. S. Alexander, W. A. Arbaugh, M. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare Active Network Architecture. *IEEE Net work Magazine, special issue on Active and Programmable Networks*, 12(3):29-36, 1998.

[2] Chizmadia, D. (1998). A quick tour of the CORBA security service. Retrieved August 27, 2002, from http://www.itsecurity.com/papers/corbasec.htm

[3] A. Abdul-Rahman and S. Hailes, “A Distributed Trust Model,” Proc. New Security Paradigms Workshop, ACM Press, 1997, pp. 48-60.

[4] L. Xiong and L. Liu, “Building Trust in Decentralized Peer to Peer Electronic Communities,” Proc. 5th Int’l Conf. Electronic Commerce Research (ICECR-5), 2002; www.mathcs.emory.edu

[AAKS98] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. A Secure Active Network Environment Architecture: Realization in SwitchWare. *IEEE Network Magazine, special issue on Active and Programmable Networks*, 12(3):37-45, 1998

[5] [AAKS99] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. Security in active networks. In *Secure Internet Programming,* Lecture Notes in Computer Science. Springer-Verlag Inc., New York, NY, USA, 1999.

[6] R. Housley et al., Internet X.509 Public Key Infrastructure, Certificate and CRL Profile, IETF RFC 2459, Jan. 1999; www.ietf.org/rfc/rfc2459.txt.

[7] G. Zacharia, “*Trust Management through Reputation Mechanisms*,” Proc. Workshop in Deception, Fraud, and Trust in Agent Societies,

3rd Int’l Conf. Autonomous Agents (Agents99), ACM Press, 1999; www.istc.cnr.it/T3/download/aamas1999/zacharia.pdf.

[8] MageLang Institute. (1998). *Fundamentals of Java security*. Retrieved September 4, 2002, from http://developer.java.sun.com/developer/onlineTraining/Security